Fondamenti di Informatica II – Ingegneria Informatica

COMPITINO 1
(24/11/1999)

Durata: 1h50m

Esercizio 1 [10 punti] Si consideri la seguente equazione di ricorrenza, definita per valori positivi del parametro n:

$$T(n) = \begin{cases} 2, & n \leq 6, \\ T\left(\lfloor \frac{n}{3} + 1 \rfloor\right) + T\left(\lceil \frac{n}{6} \rceil\right) + 2n, & n > 6. \end{cases}$$

Dimostrare che $T(n) \in O(n)$.

Esercizio 2 [10 punti] Sia n una potenza di due. Dati due polinomi $A(x)$ e $B(x)$ di grado limitato da n, rappresentati dai due vettori degli n coefficienti $\boldsymbol{a}$ e $\boldsymbol{b}$, si vuole calcolare il vettore $\boldsymbol{c}$ dei coefficienti del polinomio prodotto $C(x) = A(x)B(x)$ **senza ricorrere alle trasformate**. Si sviluppi un algoritmo divide-and-conquer che, dati in input $\boldsymbol{a}$ e $\mathbf{b}$ restituisca $\boldsymbol{c}$ in tempo $O(n^{\log_2 3})$ (*Suggerimento:* si sviluppi un approccio analogo a quello di Karatsuba per la moltiplicazione di interi.)

Esercizio 3 [10 punti] Sia n un numero pari. Sia Π il seguente problema computazionale: dato un insieme A di n interi, si determini un sottoinsieme $A' \subset A$ di $n/2$ elementi che contenga il minimo di A.

(a) **[5 punti]** Si dimostri che un qualsiasi algoritmo deve effettuare almeno $n/2$ confronti per risolvere Π;

(b) **[5 punti]** Si sviluppi un algoritmo che risolve Π eseguendo **esattamente** $n/2$ confronti.

Attenzione Tutte le risposte vanno motivate con rigorose **analisi di correttezza**. Gli algoritmi devono essere descritti utilizzando lo **pseudocodice** visto in classe.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITINO 2
(14/1/2000)

Durata: 2h

Esercizio 1 [10 punti] Si ricorda che il numero $P(n)$ di parentesizzazioni bilanciate di una espressione con n operandi soddisfa la seguente ricorrenza:

$$P(n) = \begin{cases} 1, & n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k), & n > 1. \end{cases}$$

Progettare e analizzare un algoritmo ricorsivo **memoizzato** per il calcolo di $P(n)$.

Esercizio 2 [10 punti] Sia $X = \langle x_1, x_2, \ldots, x_n \rangle$ una stringa di numeri reali positivi arbitrari. Data una sottostringa $X_{i..j} = \langle x_i, x_{i+1}, \ldots x_j \rangle$, $1 \leq i \leq j \leq n$, si definisca

$$\text{costo}\,(X_{i..j}) = \prod_{k=i}^{j} x_k = x_i \cdot x_{i+1} \cdot \ldots \cdot x_j.$$

Si consideri il problema di determinare la quantità $\max\{\text{costo}\,(X_{i..j}) : 1 \leq i \leq j \leq n\}$.

(a) Si determini un appropriato spazio dei sottoproblemi e la ricorrenza associata, provandone la correttezza (*Suggerimenti:* si partizionino le sottostringhe di X rispetto al loro punto di inizio. Attenzione ai valori < 1.)

(b) Si sviluppi e si analizzi il codice di un algoritmo di programmazione dinamica che risolve il problema in tempo $O(n)$.

Esercizio 3 [10 punti] Si consideri il seguente problema decisionale:

TWO-OUTPUT XOR CIRCUIT SAT (TOXCS):
ISTANZA: $\langle C(x_1, x_2, \ldots x_n) \rangle$, C circuito booleano con n input e *due* output y_1 e y_2.
DOMANDA: Esiste un assegnamento di valori di verità agli ingressi per cui **esattamente** uno dei due output vale 1 ($y_1 = 1$ e $y_2 = 0$ o viceversa)?

Dimostrare che TOXCS e' NP-Hard.

Attenzione Tutte le risposte vanno motivate con rigorose **analisi di correttezza**. Gli algoritmi devono essere descritti utilizzando lo **pseudocodice** visto in classe.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(24/1/2000)

Durata: 2h 45m

Esercizio 1 [6 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro $n = 2^{2^i}$:

$$T(n) = \begin{cases} 2T(\sqrt{n}) + (\sqrt{2}-1)\sqrt{\log n}, & n > 2, \\ 0 & n = 2. \end{cases}$$

Esercizio 2 [6 punti] Detto $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ un generico vettore a componenti complesse, si definisca il vettore $\omega(\boldsymbol{x}) = (x_0 \cdot \omega_n^0, x_1 \cdot \omega_n^1, \ldots, x_{n-1} \cdot \omega_n^{n-1})$. Determinare la relazione esistente tra le componenti del vettore $\boldsymbol{X} = DFT_n(\boldsymbol{x})$ e le componenti del vettore $\boldsymbol{\Omega X} = DFT_n(\omega(\boldsymbol{x}))$.

Esercizio 3 [8 punti] Siano m e n due interi positivi. Data una stringa di interi $X = \langle x_1, x_2, \ldots, x_n \rangle$, una sottosequenza $Z = \langle x_{j_1}, x_{j_2}, \ldots, x_{j_{|Z|}} \rangle$ di X si dice *a somma 0 mod m* se $\sum_{k=1}^{|Z|} x_{j_k} \equiv 0 \bmod m$. Si progetti un algoritmo di programmazione dinamica che calcoli in tempo $O(nm)$ la lunghezza di una sottosequenza a somma 0 mod m di lunghezza massima.
(*Suggerimento:* Potrebbe essere necessario calcolare anche le lunghezze massime delle sottosequenze a somma diversa da zero mod m ...)

Attenzione: Si fornisca la ricorrenza, se ne provi la correttezza e infine si scriva lo pseudocodice dell'algoritmo. Risoluzioni immotivate saranno **fortemente** penalizzate.

Esercizio 4 [6 punti] Si consideri il seguente problema decisionale:

CIRCUIT-VALUE PROBLEM (CVP):

ISTANZA: $\langle C(x_1, x_2, \ldots x_n), K \rangle$, C circuito booleano con n input e $k = \lfloor \log K \rfloor + 1$ output $y_{k-1}, y_{k-2}, \ldots, y_0$.

DOMANDA: Esiste un assegnamento di valori di verità agli ingressi di C sotto cui la stringa dei valori assunti dagli output $\langle y_{k-1}, y_{k-2}, \ldots, y_0 \rangle$ corrisponde alla rappresentazione binaria di K?

Dimostrare che CVP e' NP-Hard.

Esercizio 5 [6 punti] Si consideri un criptosistema a chiave pubblica di tipo RSA in cui il dominio dei messaggi è $\boldsymbol{Z}_n$, con $n = 13 \times 11 = 143$. Data la chiave pubblica $P = (7, 143)$ determinare la corrispondente chiave segreta S e la relativa funzione di decodifica $S(M)$ ad essa associata.
(*Suggerimento:* Si ricorda che il codice dell'algoritmo EXTENDED_EUCLID(a, b) è il seguente:

```
EXTENDED_EUCLID(a, b)
if b = 0 then return {a, (1, 0)}
  else {d, (x', y')} ← EXTENDED_EUCLID(b, a mod b)
       return {d, (y', x' − ⌊a/b⌋ · y')}
```

)

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(7/2/2000)

Durata: 2h 45m

Esercizio 1 [6 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro n che siano potenze di due:

$$T(n) = \begin{cases} 2T(n/2) + n \log n, & n > 1, \\ 0 & n = 1. \end{cases}$$

Esercizio 2 [7 punti] Sia n una potenza di due. Dato in ingresso un vettore di n numeri complessi $\boldsymbol{a} = (a_0, a_1, \ldots a_{n-1})$ si vuole calcolare in uscita il vettore dei coefficienti del polinomio $P(x) = \prod_{i=0}^{n-1}(x - a_i)$. Si progetti e si analizzi un algoritmo divide-and-conquer che risolve il problema in tempo $O(n \log^2 n)$.

Esercizio 3 [6 punti] Sia n un arbitrario intero positivo. Sia Π il seguente problema computazionale: dato un insieme A di n interi e un valore m, $1 \leq m < n$, si determini un sottoinsieme $A' \subset A$ di m elementi che contenga il massimo di A.

(a) **[4 punti]** Si dimostri che un qualsiasi algoritmo deve effettuare almeno $n - m$ confronti per risolvere Π;

(b) **[3 punti]** Si sviluppi un algoritmo che risolve Π eseguendo **esattamente** $n - m$ confronti.

Esercizio 4 [7 punti] Siano $L_1, L_2 \subset \{0,1\}^*$ due linguaggi con $L_1 \in NP$-Hard, $L_2 \in P$ e $L_1 \cap L_2 = \emptyset$. Si dimostri rigorosamente la seguente proposizione:

$$L_1 \cup L_2 \neq \{0,1\}^* \Rightarrow L_1 \cup L_2 \in NP\text{-Hard}.$$

(*Suggerimento:* Dimostare che ogni linguaggio in NP si riduce a $L_1 \cup L_2 \neq \{0,1\}^*$ modificando la relativa riduzione a L_1.)

Esercizio 5 [6 punti] Si consideri un criptosistema a chiave pubblica di tipo RSA in cui il dominio dei messaggi è $\boldsymbol{Z}_n$, con $n = 17 \times 19 = 323$. Data la chiave pubblica $P = (7, 323)$ si determini la chiave segreta S e la relativa funzione di decodifica $S(M)$ ad essa associata.
(*Suggerimento:* Si ricorda che il codice dell'algoritmo EXTENDED_EUCLID(a, b) è il seguente:

```
EXTENDED_EUCLID(a, b)
if b = 0 then return {a, (1, 0)}
  else {d, (x', y')} ← EXTENDED_EUCLID(b, a mod b)
       return {d, (y', x' − ⌊a/b⌋ · y')}
)
```

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(23/6/2000)

Durata: 2h 45m

Esercizio 1 [7 punti] Si consideri la seguente equazione di ricorrenza per valori arbitrari del parametro n:

$$T(n) = \begin{cases} 1, & n < 24, \\ T\left(\lfloor \frac{n}{6} \rfloor\right) + T\left(\left\lceil \frac{3n}{4} \right\rceil\right) + 2n, & n \geq 24. \end{cases}$$

Si dimostri (usando l'induzione parametrica) che $T(n) \in O(n)$.

Esercizio 2 [7 punti] Detto $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ un generico vettore a componenti complesse, si definisca, per un dato k, $0 \leq k < n$, il vettore $\boldsymbol{\omega x}^k$ la cui i-sima componente, per $0 \leq i < n$, è $\boldsymbol{\omega x}_i^k = x_i \cdot \omega_n^{ik}$. Determinare la relazione esistente tra le componenti del vettore $\boldsymbol{X} = DFT_n(\boldsymbol{x})$ e le componenti del vettore $\boldsymbol{\Omega X}^k = DFT_n(\boldsymbol{\omega x}^k)$.

Esercizio 3 [7 punti] Data una stringa di interi positivi $\langle x_1, x_2, \ldots, x_n \rangle$, si definisca il *costo* di una sottosequenza $Z = \langle x_{j_1}, x_{j_2}, \ldots, x_{j_{|Z|}} \rangle$ di X come costo$(Z) = \sum_{k=1}^{|Z|} (-1)^{k-1} x_{j_k}$. (Si assuma che costo$(\epsilon) = 0$.) Si progetti un algoritmo di programmazione dinamica che calcoli in tempo $O(n)$ il costo massimo di una sottosequenza di X.
(*Suggerimenti:* Potrebbe essere necessario calcolare anche il costo minimo.)

Attenzione: Si fornisca la ricorrenza, se ne provi la correttezza e infine si scriva lo pseudocodice dell'algoritmo. Risoluzioni immotivate saranno **fortemente** penalizzate.

Esercizio 4 [6 punti] Si consideri il seguente problema decisionale:

REDUNDANT SAT (RSAT):

ISTANZA: $\langle \Phi(x_1, x_2, \ldots x_n) = C_1 \wedge C_2 \ldots \wedge C_m \rangle$, Φ formula booleana CNF con n variabili e m clausole.

DOMANDA: Esiste un assegnamento di valori di verità alle n variabili per cui ogni clausola contiene almeno due letterali veri?

Dimostrare che RSAT è NP-Hard.

Esercizio 5 [6 punti] Si consideri un criptosistema a chiave pubblica di tipo RSA in cui il dominio dei messaggi è $\boldsymbol{Z}_n$, con $n = 7 \times 19 = 133$. Data la chiave pubblica $P = (7, 133)$ determinare la corrispondente chiave segreta S e la relativa funzione di decodifica $S(M)$ ad essa associata.
(*Suggerimento:* Si ricorda che il codice dell'algoritmo EXTENDED_EUCLID(a, b) è il seguente:

```
EXTENDED_EUCLID(a, b)
if b = 0 then return {a, (1, 0)}
  else {d, (x', y')} ← EXTENDED_EUCLID(b, a mod b)
       return {d, (y', x' − ⌊a/b⌋ · y')}
```

)

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(5/7/2000)

Durata: 2h 45m

Esercizio 1 [7 punti] Si consideri la seguente equazione di ricorrenza per valori arbitrari del parametro n:

$$T(n) = \begin{cases} 0, & n \leq 4, \\ T\left(\lfloor \frac{n}{4} \rfloor\right) + T\left(\left\lfloor \frac{3n}{4} \right\rfloor\right) + n, & n > 4. \end{cases}$$

Si dimostri (usando l'induzione parametrica) che $T(n) \in O(n \log n)$.

Esercizio 2 [7 punti] Siano $\boldsymbol{a} = (a_0, \ldots, a_{n-1})$ e $\boldsymbol{b} = (b_0, \ldots, b_{n-1})$ due vettori complessi a n componenti (n potenza di due). Sia $\boldsymbol{b}^k$ il vettore ottenuto applicando a $\boldsymbol{b}$ uno shift ciclico destro di k posizioni, cioè $\boldsymbol{b}^0 = \boldsymbol{b}$, e $\boldsymbol{b}^k = (b_{n-k}, \ldots, b_{n-1}, b_0, b_1, \ldots, b_{n-k-1})$, per $1 \leq k \leq n-1$. Si fornisca un algoritmo efficiente che determini in tempo $O(n \log n)$ il valore k' che massimizza il prodotto scalare $\boldsymbol{a} \cdot \boldsymbol{b}^k = \sum_{j=0}^{n-1} \boldsymbol{a}_j \boldsymbol{b}^k{}_j$, per $0 \leq k \leq n-1$. (*Suggerimento:* Si riconduca il problema a un'opportuna convoluzione.)

Esercizio 3 [7 punti] Si consideri la seguente relazione di ricorrenza per il calcolo dei cosiddetti *numeri di Stirling del secondo ordine* $S(n,k)$, per $1 \leq k \leq n$:

$$S(n,k) = \begin{cases} 1, & k = 1 \text{ o } k = n, \\ k \cdot S(n-1,k) + S(n-1,k-1) & 1 < k < n. \end{cases}$$

Si fornisca il codice e l'analisi di un algoritimo ricorsivo **memoizzato** che calcoli il valore $S(n,k)$ in tempo e spazio $O(nk)$.

Esercizio 4 [6 punti] Si consideri il seguente problema decisionale:

ALTERNATE SAT (ASAT):

ISTANZA: $\langle \Phi(x_1, x_2, \ldots x_n), \Psi(x_1, x_2, \ldots x_n) \rangle$, Φ, Ψ formule booleane.

DOMANDA: Esiste un assegnamento di verità che soddisfa una sola tra Φ e Ψ?

Dimostrare che ASAT è NP-hard.

Esercizio 5 [6 punti] Sia $n = 5 \times 7 = 35$. Si determini il numero x in Z_n corrisipondente alla rappresentazione $(4,3) \in Z_5 \times Z_7$ implicata dal teorema cinese dei resti.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(6/9/2000)

Durata: 2h 45m

Esercizio 1 [6 punti] Si determini la soluzione della seguente relazione di ricorrenza, definita per valori del parametro $n = 2^{2^i}$:

$$T(n) = \begin{cases} 0 & n = 2, \\ nT(\sqrt{n}) + n^2 & n > 2. \end{cases}$$

Esercizio 2 [7 punti] Sia n una potenza di due. Dato un arbitrario vettore complesso di n componenti $\boldsymbol{x}$, si consideri la matrice circolante $C(\boldsymbol{x})$ che ha $\boldsymbol{x}$ come prima colonna. Si fornisca lo pseudocodice e si analizzi un algoritmo efficiente per determinare la prima colonna $\boldsymbol{y}$ della matrice

$$C(\boldsymbol{y}) = \sum_{j=0}^{\log n - 1} [C(\boldsymbol{x})]^{2^j}$$

ottenuta sommando le prime $\log n$ potenze di $C(\boldsymbol{x})$ con esponente uguale a una potenza di due.

Esercizio 3 [7 punti] Si consideri il seguente problema computazionale Π: dato un insieme A di n interi e un intero x, si determini il sottoinsieme $A' \subset A$ che contenga tutti e soli gli elementi di A minori di x.

Utilizzando la tecnica dell'albero dei confronti si dimostri che:

(a) Nel caso in cui l'insieme A non sia ordinato, un qualsiasi algoritmo deve effettuare almeno n confronti per risolvere Π;

(b) Nel caso in cui l'insieme A sia ordinato, un qualsiasi algoritmo deve effettuare almeno $\log n$ confronti per risolvere Π.

Esercizio 4 [7 punti] Si consideri il seguente problema decisionale:

FORMULA COINCIDENCE (FC):

ISTANZA: $\langle \Phi(x_1, x_2, \ldots x_n), \Psi(x_1, x_2, \ldots x_n) \rangle$, Φ, Ψ formule booleane.

DOMANDA: Esiste un assegnamento di valori di verità alle n variabili su cui Φ e Ψ coincidono?

Si dimostri che FC è NP-Hard.

Esercizio 5 [6 punti] Sia $n = \prod_{i=1}^{k} n_i$ con $n_i > 1$, $1 \leq i \leq k$ e $\gcd(n_i, n_j) = 1$, $1 \leq i < j \leq k$. Si descriva la corrispondenza diretta e quella inversa tra Z_n e $Z_{n_1} \times Z_{n_2} \times \cdots \times Z_{n_k}$ indotta dal teorema cinese dei resti.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(20/9/2000)

Durata: 2h 45m

Esercizio 1 [6 punti] Si determini la soluzione della seguente relazione di ricorrenza, definita per valori del parametro $n = 4^i$:

$$T(n) = \begin{cases} 6 & n = 1, \\ 16T\left(\frac{n}{4}\right) + 3n^2 & n > 1. \end{cases}$$

Esercizio 2 [7 punti] Sia n una potenza di due. Dato un arbitrario vettore complesso di n componenti $\boldsymbol{x}$, si consideri la matrice circolante $C(\boldsymbol{x})$ che ha $\boldsymbol{x}$ come prima colonna. Si fornisca lo pseudocodice e si analizzi un algoritmo efficiente per determinare la prima colonna $\boldsymbol{y}$ della matrice

$$C(\boldsymbol{y}) = \prod_{j=0}^{\log n - 1} [C(\boldsymbol{x})]^{2^j}$$

ottenuta moltiplicando le prime $\log n$ potenze di $C(\boldsymbol{x})$ con esponente uguale a una potenza di due.

Esercizio 3 [7 punti] Sia $X = \langle x_1, x_2, \ldots, x_n \rangle$ una stringa di interi **arbitrari** (negativi, positivi o nulli). Data una sottostringa $X_{i..j} = \langle x_i, x_{i+1}, \ldots, x_j \rangle$, $1 \leq i \leq j \leq n$, si definisca

$$\text{costo}\,(X_{i..j}) = \prod_{k=0}^{j-i} x_{i+k} \ .$$

Si sviluppi e si analizzi il codice di un algoritmo di programmazione dinamica che restituisca gli indici di inizio e di fine di una sottostringa di costo massimo, in tempo $O(n)$. (*Suggerimento*: si considerino come sottoproblemi i suffissi di X e si osservi che vanno calcolati anche i costi minimi...)

Esercizio 4 [7 punti]

Sia dato il seguente problema decisionale:

FORMULA COINCIDENCE (FC):

ISTANZA: $\langle \Phi(\boldsymbol{x}), \Psi(\boldsymbol{x}) \rangle$, Φ, Ψ formule booleane in CNF di variabili $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$.

DOMANDA: Esiste un assegnamento di valori di verità alle n variabili su cui Φ e Ψ coincidono?

Supponendo di avere a disposizione un algoritmo DECIDE_FC($\langle \Phi(\boldsymbol{x}), \Psi(\boldsymbol{x}) \rangle$) che decide FC in tempo **lineare** nella taglia dell'istanza, si progetti un nuovo algoritmo RETURN_ASSIGNMENT($\langle \Phi(\boldsymbol{x}), \Psi(\boldsymbol{x}) \rangle$) che restituisca in uscita un assegnamento (se esiste) su cui le formule coincidono, in tempo quadratico nella taglia dell'istanza.

Esercizio 5 [6 punti] Su quale assunzione computazionale si fonda la sicurezza del criptosistema RSA? (*Attenzione:* Fornire una risposta sintetica. Divagazioni irrilevanti ai fini della risposta saranno penalizzate.)

Fondamenti di Informatica II – Ingegneria Informatica

COMPITINO 1 + SOLUZIONI
(13/11/2000)

Durata: 1h45m

Attenzione Risposte immotivate non saranno considerate. Gli algoritmi da sviluppare vanno descritti utilizzando lo **pseudocodice** usato in classe.

Esercizio 1 [10 punti] Si supponga di portare il valore del parametro n_0 (base) nell' algoritmo SELECT(A, i, p, r) da 80 a 70, e di modificare la fase di *divide* considerando A come diviso in $\lceil n/7 \rceil$ gruppi (anzichè $\lceil n/5 \rceil$) e scegliendo l'elemento X per partizionare A in A_1 e A_2 come la mediana delle mediane di tali gruppi.

(a) [3 punti] Si dimostri che $\max\{|A_1|, |A_2|\} \leq \lfloor \frac{5n}{7} + 8 \rfloor$.

(b) [7 punti] Si scriva l'equazione di ricorrenza per la complessità in tempo $T(n)$ del nuovo algoritmo e si dimostri che $T(n) = O(n)$.

Answer:

(a) In the new algorithm there will be at least $\lceil \frac{1}{2} \lceil \frac{n}{7} \rceil - 2 \rceil$ groups of 7 elements which contain 4 elements smaller (resp., larger) than X, hence

$$|A_1|, |A_2| \geq 4 \left\lceil \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil - 2 \right\rceil \geq \frac{2n}{7} - 8.$$

Recall that $|A_1| = n - |A_2|$, hence

$$|A_1|, |A_2| \leq n - \left(\frac{2n}{7} - 8 \right) \leq \frac{5n}{7} + 8.$$

Finally, cardinalities are integer numbers, therefore $|A_1|, |A_2| \leq \lfloor \frac{5n}{7} + 8 \rfloor$.

(b) We have:

$$T(n) = \begin{cases} c_0 & n \leq 70, \\ T\left(\lceil \frac{n}{7} \rceil\right) + T\left(\left\lfloor \frac{5n}{7} + 8 \right\rfloor\right) + cn & n > 70. \end{cases}$$

Let us try parametric induction to determine a constant $\bar{c}$ such that $T(n) \leq \bar{c}n$ for all values of n. For the base values $n \leq 70$ we have:

$$T(n) \leq c_0 \leq c_0 n \stackrel{?}{\leq} \bar{c}n,$$

whence we obtain $\bar{c} \geq c_0$. Assuming that the thesis holds for all values $k < n$, $n > 70$, we obtain

$$T(n) \quad \leq \quad \bar{c} \left\lceil \frac{n}{7} \right\rceil + \bar{c} \left\lfloor \frac{5n}{7} + 8 \right\rfloor + cn$$

$$
\begin{aligned}
&\leq \bar{c}(\frac{6n}{7}+9)+cn \\
&\overset{?}{\leq} \bar{c}n
\end{aligned}
$$

The last inequality si satisfied whenever the graph of the straigth line $y_1 = \bar{c}(\frac{n}{7}-9)$ is above the graph of $y_2 = cn$ for $n > 70$. Therefore it is sufficient to set $\bar{c} \geq 70c$.

Now, we are ready to apply induction to verify that indeed $T(n) \geq \bar{c}n$, with $\bar{c} = \max\{c_0, 70c\}$. For the base case, we have $T(n) \leq c_0 \leq \bar{c} \leq \bar{c}n$. Assuming that the thesis holds for all values $k < n$, $n > 70$, we have

$$
\begin{aligned}
T(n) &\leq \bar{c}\left\lceil \frac{n}{7} \right\rceil + \bar{c}\left\lfloor \frac{5n}{7}+8 \right\rfloor + cn \\
&\leq \bar{c}(\frac{6n}{7}+9) + \frac{\bar{c}}{70}n \\
&\quad \text{(since } c \leq \tfrac{\bar{c}}{70}\text{)} \\
&\leq \bar{c}n(\frac{6}{7}+\frac{9}{70}+\frac{1}{70}) \\
&\quad \text{(since } \tfrac{n}{70} > 1\text{)} \\
&= \bar{c}n.
\end{aligned}
$$

□

Esercizio 2 [10 punti] Sia n una potenza di due. Un intero a n-bit a si dice *simmetrico* se $n = 1$ oppure $a = 2^{n/2}b + b$, con b intero simmetrico a $n/2$ bit. Si fornisca lo pseudocodice e l'analisi di un algoritmo divide-and-conquer efficiente per la moltiplicazione di due interi simmetrici in tempo $\Theta(n)$.

Answer: Let a and b be the two symmetric n-bit integers to be multiplied. If $n = 1$ we simply have to return (a **and** b). For $n > 1$, we have that $a = 2^{n/2}c + c$ and $b = 2^{n/2}d + d$, where c and d are two $n/2$-bit symmetric integers. In this case we have

$$
\begin{aligned}
a \cdot b &= (2^{n/2}c + c)(2^{n/2}d + d) \\
&= 2^n(c \cdot d) + 2^{n/2+1}(c \cdot d) + (c \cdot d). \qquad (1)
\end{aligned}
$$

(Note that $a \cdot b$ is not necessarily symmetric.) From the last expression it is clear that we need one single recursive call in the algorithm, to compute $c \cdot d$. From that product, $a \cdot b$ will then be obtained in linear time through sums and linear shifts.

The pseudocode for the algorithm is the following.

```
SYMMETRIC_MULT(a, b)
    n ← length(a)
    if n = 1 then return (a and b)
    c ← a_{n/2-1} a_{n/2-2} ... a_0
    d ← b_{n/2-1} b_{n/2-2} ... b_0
    z ← SYMMETRIC_MULT(c, d)
    return SUM(SUM(SHIFT(z, n), SHIFT(z, n/2 + 1)), z)
```

The algorithm is correct by virtue of Equation (1). Its running time obeys to the simple recurrence $T(1) = 1$, $T(n) = T(n/2) + \Theta(n)$ whose solution, by the Master Theorem, is $T(n) = \Theta(n)$.

An alternative, more elegant nonrecursive solution to the problem can be obtained by noting that the only two n-bit symmetric integers are the number 0 (represented on n bits) and the number $2^n - 1$ (whose representation is a string of n 1's). This can be easily proved by induction on i, where $n = 2^i$. The base $i = 0$ trivially holds, since by definition both binary digits are symmetric integers. If the property holds for $n = 2^{i-1}$, it also holds for $n = 2^i$ since by concatenating two identical $(n/2)$-bit representations of 0 (resp., $2^{n/2} - 1$) we obtain 0 (resp., $2^n - 1$). Based on such property, we can write an algorithm where we first check if either $\boldsymbol{a}$ or $\boldsymbol{b}$ are 0, and return 0 if this is the case. Otherwise we must return $(2^n - 1)^2$. Noting that $(2^n - 1)^2 = 2^{2n} - (2^n - 1) - 2^n$ we see that in this case we must return a $2n$-bit string whose most significant $n - 1$ bits are 1, the next n are 0 and the least significant bit is 1. Note that in this case we also need $\Theta(n)$ bit operations to check whether the operands are zero or not, hence the algorithm still requires $\Theta(n)$ time. The pseudocode for this alternative algorithm is the following.

```
SYMMETRIC_MULT2(a, b)
    n ← length(a)
    if (a = 0) or (b = 0)
        then return 0
        else return 11...1 00...0 1
                    n-1    n
```

□

Esercizio 3 [10 punti] Sia n una potenza di due e sia $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1}) \in \boldsymbol{C}^n$ un arbitrario vettore a n componenti complesse.

(a) **[7 punti]** Si dimostri che vale la seguente relazione:

$$\mathrm{DFT}_n(\boldsymbol{x}_{\mathrm{rev}}) = n \cdot \mathrm{DFT}_n^{-1}(\boldsymbol{x})$$

(si ricorda che $(\boldsymbol{x}_{\mathrm{rev}})_i = x_{(n-i) \bmod n}$).

(b) **[3 punti]** In base alle relazione provata al Punto **(a)**, si fornisca lo pseudocodice di un semplice algoritmo INV_FFT_2 che calcola DFT_n^{-1} utilizzando l'algoritmo FFT come subroutine.

Answer:

(a) By definition of DFT_n, for $0 \le i \le n - 1$ we have:

$$\begin{aligned}
\left(\mathrm{DFT}_n(\boldsymbol{x}_{\mathrm{rev}})\right)_i &= \sum_{j=0}^{n-1} (\boldsymbol{x}_{\mathrm{rev}})_j \, \omega_n^{ij} \\
&= \sum_{j=0}^{n-1} x_{(n-j) \bmod n} \omega_n^{ij}
\end{aligned}$$

Let us now change summation index by setting $k = (n - j) \bmod n$. Clearly, when j ranges from 0 to $n-1$, so does k. Moreover, we have that $j = (n-k) \bmod n$. Therefore

$$\begin{aligned}
\left(\mathrm{DFT}_n(\boldsymbol{x}_{\mathrm{rev}})\right)_i &= \textstyle\sum_{k=0}^{n-1} x_k \omega_n^{i((n-k) \bmod n)} = \sum_{k=0}^{n-1} x_k \omega_n^{i(n-k)} \\
&= \textstyle\sum_{k=0}^{n-1} x_k \omega_n^{in} \omega_n^{-ik} = \sum_{k=0}^{n-1} x_k \omega_n^{-ik} = \qquad n \left(\mathrm{DFT}_n^{-1}(\boldsymbol{x})\right)_i
\end{aligned}$$

(b) We compute $\mathrm{DFT}_n^{-1}(\boldsymbol{x})$ by first obtaining $\mathrm{DFT}_n(\boldsymbol{x}_{\mathrm{rev}})$ and then dividing each of its components by n. The algorithm follows.

```
INV_FFT_2(x)
    n ← length(x)
    for i = 0 to n − 1 do xr_i ← x_{(n−i) mod n}
    y ←  FFT(xr)
    for i = 0 to n − 1 do y_i ← y_i/n
    return y
```

□

Fondamenti di Informatica II – Ingegneria Informatica

COMPITINO II
(20/12/2000)

Durata: 2h

Attenzione Risposte immotivate non saranno considerate. Gli algoritmi da sviluppare vanno descritti utilizzando lo **pseudocodice** usato in classe.

Esercizio 1 [10 punti] Per $n \geq 1$ e $0 \leq k \leq n$, i coefficienti binomiali $C(n,k) = \binom{n}{k}$ soddisfano la seguente relazione di ricorrenza:

$$C(n,k) = \begin{cases} 1 & k = 0 \text{ o } k = n, \\ C(n-1,k) + C(n-1,k-1) & 0 < k < n. \end{cases}$$

Si fornisca lo pseudocodice di un algoritmo ricorsivo memoizzato che, dati in ingresso n e k, con $0 \leq k \leq n$ e $n \geq 1$ restituisca $C(n,k)$, e se ne valuti la complessità in funzione del numero di operazioni aritmetiche compiute dall'algoritmo.

Answer: As usual, we realize the memoized algorithm for computing $C(n,k)$ as a pair of subroutines, namely, INIT_C(n,k) and REC_C(n,k). The former is used to compute the base cases and initialize the look-up table. The latter is the recursive routine which computes the remaining (nonbase) table entries.

```
INIT_C(n, k)
    if [(k = 0) or (k = n)] then return 1
    for i ← 1 to n do
        C[i, 0] ← C[i, i] ← 1
    for i ← 2 to n do
        for j ← 1 to MIN(i − 1, k) do
            C[i, j] ← 0
    return REC_C(n, k)

REC_C(n, k)
    if C[n, k] = 0
        then C[n, k] ← REC_C(n − 1, k) + REC_C(n − 1, k − 1)
    return C[n, k]
```

Note that in INIT_C(n,k), particular care has been taken in order to initialize only $\Theta(nk)$ table entries. Indeed, no value $C(i,j)$, with $k < j \leq i$ will ever be needed to compute $C(n,k)$. As for the running time of the above algorithm, observe that each of the relevant $\Theta(nk)$ table entriies is computed just once, and that each such computation requires a single sum. Therefore the running time of the overall algorithm is $\Theta(nk)$. □

Esercizio 2 [10 punti] Per una stringa di reali positivi $Z = \langle z_1, z_2, \ldots, z_k \rangle$ sia peso$(Z) = \prod_{i=1}^{k} z_i$ (si noti che peso$(\epsilon) = 1$). Date due stringhe di reali positivi $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$, si consideri il problema di determinare una Common Subsequence (CS) Z di X e Y di peso massimo.

(a) **[5 punti]** Si provi una proprietà di sottostruttura ottima per Z e si derivi da essa una ricorrenza appropriata per il peso massimo di una CS.

(b) **[5 punti]** Si scriva lo pseudocodice iterativo per il calcolo bottom-up della ricorrenza sviluppata al Punto **(a)**. L'algoritmo deve avere complessità $\Theta(mn)$.

Suggerimento: Si faccia attenzione ai valori < 1.

Answer:

(a) Let $Z = \langle z_1, z_2, \ldots z_k \rangle$ be an MWCS of X and Y, with $Z = \langle x_{j_1}, x_{j_2}, \ldots x_{j_k} \rangle = \langle y_{h_1}, y_{h_2}, \ldots y_{h_k} \rangle$, with $1 \leq j_1 < j_2 < \ldots < j_k \leq m$ and $1 \leq h_1 < h_2 < \ldots < h_k \leq n$. Note that for $1 \leq i \leq k$, it must be $z_i \geq 1$, or otherwise $Z' = \langle z_1, \ldots z_{i-1}, z_{i+1}, \ldots z_k \rangle$ (which is a CS of X and Y) would have a higher weight than Z. Furthermore, we can assume without loss of generality that $z_i \neq 1$, since otherwise we can simply remove z_i, so that Z' remains an MWCS of X and Y. We have:

1. *If $x_m = y_n \leq 1$, then Z is an MWCS of X_{m-1} and Y_{n-1}.*
   Proof: Since Z contains only numbers greater than one, it must be $j_k < m$ and $h_k < n$. Hence Z is a CS of X_{m-1} and Y_{n-1}. Clearly, it must be an MWCS of X_{m-1} and Y_{n-1}, or otherwise an MWCS of X_{m-1} and Y_{n-1} would also be a CS of X and Y heavier than Z.

2. *If $x_m = y_n > 1$, then $z_k = x_m$ and Z_{k-1} is an MWCS of X_{m-1} and Y_{n-1}.*
   Proof: If $z_k \neq x_m = y_n$, it must be $j_k < m$ and $h_k < n$. But then $Z' = \langle Z, x_m \rangle$ is still a CS of X and Y with weight $x_m \cdot \text{weight}(Z) > \text{weight}(Z)$, a contradiction. Also, Z_{k-1} is a CS of X_{m-1} and Y_{n-1}. Clearly, it must be the one of maximum weight.

3. *If $x_m \neq y_n$ then Z is the sequence of maximum weight between an MWCS of X_m and Y_{n-1} and an MWCS of X_{m-1} and Y_n.*
   Proof: Since $x_m \neq y_n$, either $j_k < m$ or $h_k < n$. Hence Z must be either an MWCS of X_{m-1} and Y (first case) or an MWCS of X and Y_{n-1} (second case). Clearly, Z must be the sequence of largest weight among the two.

Let $W[i,j]$ be the weight of an MWCS of X_i and Y_j, with $0 \leq i \leq m$ and $0 \leq j \leq n$. The above property implies the following recurrence for $W[i,j]$.

$$W[i,j] = \begin{cases} 1 & i = 0 \text{ or } j = 0, \\ W[i-1, j-1] & \text{if } x_i = y_j \leq 1, \\ x_i \cdot W[i-1, j-1] & \text{if } x_i = y_j > 1, \\ \max\{W[i-1, j], W[i, j-1]\} & \text{otherwise.} \end{cases}$$

(b) The bottom-up computation of the recurrence of Point **(a)** can be performed as follows:

```
MWCS(X, Y)
    m ← length(X)
    n ← length(Y)
    for i ← 0 to m do W[i, 0] ← 1
    for j ← 0 to n do W[0, j] ← 1
    for i ← 1 to m do
        for j ← 1 to n do
            if (x_i = y_j)
                then if (x_i ≤ 1)
                        then W[i, j] ← W[i − 1, j − 1]
                        else W[i, j] ← x_i · W[i − 1, j − 1]
                else W[i, j] ← MAX(W[i, j − 1], W[i − 1, j])
    return W[m, n]
```

The correctness of the above algorithm follows from Point **(a)** and from the fact that the values $W[i-1,j-1], W[i,j-1]$ and $W[i-1,j]$ have already been computed when $W[i,j]$ is being computed. The algorithm's running time is clearly $\Theta(mn)$. □

Esercizio 3 [10 punti] Una funzione di riduzione f: $\{0,1\}^\star \to \{0,1\}^\star$ si dice *bivariata* se $|\{y \in \{0,1\}^\star : \exists x \in \{0,1\}^\star : y = f(x)\}| = 2$ (cioè, la funzione assume solo due valori distinti). Si dimostri che, dato un arbitrario linguaggio $L \in P$, $L \neq \{0,1\}^\star$, esiste una funzione di riduzione bivariata da L al linguaggio CLIQUE. Si ricorda che CLIQUE è il problema con istanze $\langle G = (V,E), k\rangle, 1 \leq k \leq |V|$ e domanda associata "Esiste in G un sottografo completo di k nodi?"

(*Suggerimento:* L'algoritmo per il calcolo di f usa al suo interno l'algoritmo di decisione A_L per L...)

Answer: Let $L \in P$, and let $A_L(x)$ be the polynomial-time decision algorithm for L. Consider two undirected graphs consisting of one and two isolated nodes, respectively, that is, $G_1 = (\{1\}, \emptyset)$ and $G_2 = (\{1,2\}, \emptyset)$. Finally, let $y_1 = \langle G_1, 1\rangle$ and $y_2 = \langle G_2, 2\rangle$. Note that $y_1 \in$ CLIQUE and that $y_2 \notin$ CLIQUE. Consider now the following bivariate reduction function:

$$f(x) = \begin{cases} y_1 & x \in L, \\ y_2 & x \notin L. \end{cases}$$

Function f is computed by the following algorithm:

```
A_f(x)
    y_1 ← ⟨({1}, ∅), 1⟩
    y_2 ← ⟨({1, 2}, ∅), 2⟩
    if A_L(x) = 1
        then return y_1
        else return y_2
```

Since $A_L(x)$ is a polynomial-time algorithm, so is A_f. Moreover:

$$\begin{aligned} x \in L &\Rightarrow f(x) = y_1 \in \text{CLIQUE} \\ x \notin L &\Rightarrow f(x) = y_2 \notin \text{CLIQUE} \end{aligned}$$

Hence, f is a bivariate reduction function from L to CLIQUE. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(15/1/2001)

Durata: 2h 45m

Esercizio 1 [6 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro $n = 2^{2^i}$:

$$T(n) = \begin{cases} \sqrt{2}T(\sqrt{n}) + \sqrt{\log n}, & n > 2, \\ 0 & n = 2. \end{cases}$$

Esercizio 2 [6 punti] Detto $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ un generico vettore a componenti complesse, si definisca il vettore $\overline{\omega}(\boldsymbol{x}) = (x_0 \cdot \omega_n^0, x_1 \cdot \omega_n^{-1}, \ldots, x_{n-1} \cdot \omega_n^{-(n-1)})$. Determinare la relazione esistente tra le componenti del vettore $\boldsymbol{X} = DFT_n(\boldsymbol{x})$ e le componenti del vettore $\overline{\boldsymbol{\Omega}}\boldsymbol{X} = DFT_n(\overline{\omega}(\boldsymbol{x}))$.

Esercizio 3 [7 punti] Sia $G = (V, E)$, con $|V| = n$, n pari, un grafo diretto pesato. Data W la matrice dei pesi per gli archi di G, con

$$1 \leq i, j \leq n: \quad W[i,j] = \begin{cases} w_{ij} \in \boldsymbol{R}^+ \cup \{0\} & (i,j) \in E, \\ 0 & i = j, \\ +\infty & \text{altrimenti}, \end{cases}$$

si vuole determinare, per ogni coppia (i, j), $1 \leq i, j \leq n$, la lunghezza di un cammino minimo tra i e j *che utilizza come nodi intermedi solo nodi di indice pari* (ad es., $\pi_{1,5} = \langle 1, 4, 8, 2, 5 \rangle$ e' un possibile cammino di questo tipo tra il nodo 1 e il nodo 5.). Si fornisca un algoritmo di programmazione dinamica che risolva il problema in tempo $\Theta(n^3)$ e spazio $\Theta(n^2)$. (*Suggerimento:* Si modifichi l'algoritmo di Floyd-Warshall)

Attenzione: Si fornisca la ricorrenza, se ne provi la correttezza e infine si scriva lo pseudocodice dell'algoritmo. Risoluzioni immotivate saranno **fortemente** penalizzate.

Esercizio 4 [7 punti] Si consideri il seguente problema decisionale:

REDUNDANT SAT (R-SAT):

ISTANZA: $\langle \Phi(x_1, ..., x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m \rangle$, Φ formula booleana CNF (ogni clausola C_i è una disgiunzione di un numero arbitrario di letterali).

DOMANDA: Esiste un assegnamento di valori di verità alle n variabili sotto cui *ogni* clausola C_i contiene almeno due letterali veri?

Dimostrare che R-SAT è NP-Hard.

Esercizio 5 [6 punti] Si consideri un criptosistema a chiave pubblica di tipo RSA in cui il dominio dei messaggi è $\boldsymbol{Z}_n$, con $n = 11 \times 23 = 253$. Data la chiave pubblica $P = (13, 253)$ determinare la corrispondente chiave segreta S e la relativa funzione di decodifica $S(M)$ ad essa associata.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(29/1/2001)

Durata: 2h 45m

Esercizio 1 [6 punti] Si consideri la seguente equazione di ricorrenza definita per ogni valore positivo del parametro n:

$$T(n) = \begin{cases} 7T\left(\lfloor \frac{n}{2} \rfloor\right) + n^3, & n > 1, \\ 8 & n = 1. \end{cases}$$

Utilizzando l'induzione parametrica, si dimostri che $T(n) = O(n^3)$.

Esercizio 2 [7 punti] Sia n una potenza di quattro. Si analizzi e si fornisca lo pseudocodice di un algoritmo ricorsivo FFT2 per il calcolo della DFT_n, il cui tempo di esecuzione obbedisca alla seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 4T\left(\lfloor \frac{n}{4} \rfloor\right) + O(n), & n > 1, \\ 0 & n = 1. \end{cases}$$

(*Suggerimento:* L'algoritmo deve chiamare ricorsivamente se stesso quattro volte su vettori di taglia $n/4$ e ricombinare i risultati delle chiamate in tempo $O(n)$.)

Esercizio 3 [7 punti] Sia n una potenza di due e sia $A[1 \ldots n]$ un array contenente n interi distinti. Per $0 \leq i \leq \log n - 1$, sia $y_i = \sum_{j=1}^{2^i} (j\text{-o.s.}(A))$, dove j-o.s.(A) denota la j-sima order statistic di A. Si analizzi e si fornisca lo pseudocodice di un algoritmo ricorsivo che, dato in ingresso A, restituisca in uscita un vettore $Y[0 \ldots \log n - 1]$ con $Y[i] = y_i$ per $0 \leq i \leq \log n - 1$. La complessità dell'algoritmo deve essere $O(n)$.

Esercizio 4 [6 punti] Si dimostri che INDEPENDENT_SET $<_P$ VERTEX_COVER.

Esercizio 5 [6 punti] Si consideri un criptosistema a chiave pubblica di tipo RSA in cui il dominio dei messaggi è $\boldsymbol{Z}_n$, con $n = 9 \times 29 = 261$. Data la chiave pubblica $P = (111, 261)$ determinare la corrispondente chiave segreta S e la relativa funzione di decodifica $S(M)$ ad essa associata. Perchè non è una buona idea utilizzare questa chiave pubblica?

Attenzione Risposte immotivate non saranno considerate. Gli algoritmi da sviluppare vanno descritti utilizzando lo **pseudocodice** usato in classe.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO + SOLUZIONI
(11/6/2001)

Durata: 2h 45m

Esercizio 1 [6 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro $n = 2^{2^i}$:

$$T(n) = \begin{cases} \sqrt{n}T(\sqrt{n}) + n\log n, & n > 2, \\ 0 & n = 2. \end{cases}$$

Answer: Use the general formula taking into account that $s(n) = \sqrt{n}$, $f^i(n) = n^{2^{-i}}$, $f^*(n,2) = \log\log n - 1$, and $w(n) = n\log n$. We have that:

$$\begin{aligned} \left[\prod_{j=0}^{\ell-1} s(f^j(n))\right] w(f^\ell(n)) &= \left[\prod_{j=0}^{\ell-1} n^{2^{-(j+1)}}\right] n^{2^{-\ell}} \log n^{2^{-\ell}} \\ &= n^{\sum_{j=1}^{\ell} 2^{-j}} n^{2^{-\ell}} 2^{-\ell} \log n \\ &= n^{1-2^{-\ell}} n^{2^{-\ell}} 2^{-\ell} \log n \\ &= 2^{-\ell} n \log n. \end{aligned}$$

Hence

$$T(n) = n\log n \sum_{\ell=0}^{\log\log n - 1} 2^{-\ell} = 2n(\log n - 1).$$

□

Esercizio 2 [7 punti] Sia n una potenza di due. Si sviluppi e si analizzi un algoritmo che, dati in ingresso due vettori binari $\boldsymbol{a}$ e $\boldsymbol{b}$ a n componenti, restituisca 1 se $\boldsymbol{b}$ è uno shift ciclico di $\boldsymbol{a}$, e 0 altrimenti. La complessità dell'algoritmo deve essere $O(n\log n)$. (*Suggerimento:* E' necessario usare un'opportuna convoluzione tra $\boldsymbol{a}$ e ...)

Answer: Observe that the i-th component of $\boldsymbol{a} \circledast \boldsymbol{b}$ is the sum of the n component-wise products of $\boldsymbol{a}$ and r_shift$(\boldsymbol{b}^R, i)$, where $\boldsymbol{b}^R$ denotes vector $(b_0, b_{n-1}, b_{n-2}, \ldots, b_1)$, and r_shift$(\boldsymbol{b}^R, i)$ denotes the cyclic right shift of $\boldsymbol{b}^R$ of i components. Since $(\boldsymbol{b}^R)^R = \boldsymbol{b}$, the i-th component of $\boldsymbol{a} \circledast \boldsymbol{b}^R$ is the sum of the n component-wise products of $\boldsymbol{a}$ and r_shift$(\boldsymbol{b}, i)$.

Let k_a (resp., k_b) be the number of 1-components of $\boldsymbol{a}$ (resp., $\boldsymbol{b}$). Clearly, if $k_a \neq k_b$ then $\boldsymbol{b}$ cannot be a cyclic shift of $\boldsymbol{a}$. Otherwise, $\boldsymbol{a} =$ r_shift$(\boldsymbol{b}, i)$ if and only if the i-th component of $\boldsymbol{a} \circledast \boldsymbol{b}^R$ is equal to k (all the ones of a "match" all the ones of r_shift$(\boldsymbol{b}, i)$). This immediately yields a simple $O(n\log n)$ algorithm for our problem. □

Esercizio 3 [6 punti] Sia $G = (V, E)$, con $|V| = n$, un grafo diretto non pesato. Data A, la matrice di incidenza per gli archi di G, con

$$1 \leq i, j \leq n: \quad A[i,j] = \begin{cases} 1 & (i,j) \in E, \\ 0 & \text{altrimenti}, \end{cases}$$

si vuole determinare, per ogni coppia di nodi (i, j), $1 \leq i, j \leq n$, se esiste in G un cammino tra i e j. Si fornisca un algoritmo di che risolva il problema in tempo $\Theta(n^3)$ e spazio $\Theta(n^2)$.

Attenzione: Si provi la correttezza dell'algoritmo e se ne scriva lo pseudocodice.

Answer: We can simply use the Floyd-Warshall algorithm with weight function $w(e) = 1$ if $e \in E$ and $w(e) = +\infty$ otherwise. Indeed, any noninfinite entry in the output matrix corresponds to a pair of nodes for which a directed path exists, and viceversa. The Floyd-Warshall algorithm requires $O(n^3)$ time and $O(n^2)$ space, as required by the exercise. □

Esercizio 4 [7 punti] Si consideri il seguente problema decisionale:

3-REDUNDANT SAT (3R-SAT):

ISTANZA: $\langle \Phi(x_1, ..., x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m \rangle$, Φ formula booleana CNF (ogni clausola C_i è una disgiunzione di un numero arbitrario di letterali).

DOMANDA: Esiste un assegnamento di valori di verità alle n variabili sotto cui *ogni* clausola C_i contiene almeno tre letterali veri?

Dimostrare che 3R-SAT è NP-Hard.

Answer: We prove that 3R-SAT is NP-Hard by reducing from 3-CNF SAT. The reduction function f is defined as follows. Given an instance of 3-CNF SAT $\langle \Phi(x_1, ..., x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m \rangle$, we consider two new variables, x_{n+1} and x_{n+2}, and set

$$f(\langle \Phi(x_1, \ldots, xn) \rangle) = \langle \Phi'(x_1, \ldots, x_n, x_{n+1}, x_{n+2}) = C'_1 \wedge C'_2 \wedge \ldots \wedge C'_m \rangle,$$

where $C'_i = C_i \cup \{x_{n+1}, x_{n+2}\}$, for $1 \leq i \leq m$. If Φ is in 3-CNF SAT, then there exists an assignment to $x_1, \ldots, x_n$ under which every clause C_i contains a true literal. Hence, by augmenting that assignment with $x_{n+1} = x_{n+2} = 1$, we obtain an assignment for Φ' under which every clause C'_i contains at least three true literals. Viceversa, if Φ' is in 3R-SAT, then there is an assignment under which each of its clauses contains at least a true literal relative to one of the first n variables. By restricting such an assignment to $x_1, \ldots, x_n$ we then obtain a satisfying assignment for Φ. Clearly, f is computable in polynomial (in fact, linear) time, hence 3-CNF SAT $<_P$ 3R-SAT. □

Esercizio 5 [6 punti] Si consideri il seguente sistema di equazioni modulari:

$$\begin{cases} x &= 2 \bmod 3 \\ x &= 4 \bmod 5 \end{cases}$$

Utilizzando il teorema cinese dei resti, determinare l'unica soluzione al sistema in $\boldsymbol{Z}_{15}$.

Answer: One could use the analytic definition of the bijection between $\boldsymbol{Z}_{15}$ and $\boldsymbol{Z}_3 \times \boldsymbol{Z}_5$ implied by the Chinese Reminder Theorem. We have $m_1 = 15/3 = 5$, $m_2 = 15/5 = 3$, ${m_1}^{-1} \bmod 3 = 2$, ${m_2}^{-1} \bmod 5 = 2$, $c_1 = 5 \cdot 2 = 10$, $c_2 = 3 \cdot 2 = 6$. Hence the (unique) solution to the above system of equations is $a = (2 \cdot 10 + 4 \cdot 6) \bmod 15 = 44 \bmod 15 = 14$, whose correctness can be easily checked directly. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO + SOLUZIONI
(25/6/2001)

Durata: 2h 45m

Esercizio 1 [6 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro $n = 3^{3^i}$:

$$T(n) = \begin{cases} \sqrt[3]{n}\,T(\sqrt[3]{n}) + \sqrt[2]{n}\log_3 n, & n > 3, \\ 0 & n = 3. \end{cases}$$

Answer: Use the general formula taking into account that $s(n) = \sqrt[3]{n}$, $f^i(n) = n^{3^{-i}}$, $f^*(n,3) = \log_3\log_3 n - 1$, and $w(n) = \sqrt{n}\log_3 n$. We have that:

$$\begin{aligned} \left[\prod_{j=0}^{\ell-1} s(f^j(n))\right] w(f^\ell(n)) &= \left[\prod_{j=0}^{\ell-1} n^{3^{-(j+1)}}\right] n^{(1/2)3^{-\ell}} \log_3 n^{3^{-\ell}} \\ &= n^{\sum_{j=1}^{\ell} 3^{-j}} n^{(1/2)3^{-\ell}} 3^{-\ell}\log_3 n \\ &= n^{(1/2)(1-3^{-\ell})} n^{(1/2)3^{-\ell}} 3^{-\ell}\log_3 n \\ &= 3^{-\ell}\sqrt{n}\log_3 n. \end{aligned}$$

Hence

$$T(n) = \sqrt{n}\log_3 n \sum_{\ell=0}^{\log_3\log_3 n - 1} 3^{-\ell} = (3/2)\sqrt{n}(\log_3 n - 1).$$

□

Esercizio 2 [7 punti] Sia n una potenza di due. Si sviluppi e si analizzi un algoritmo che, dati in ingresso due vettori binari $\boldsymbol{a}$ e $\boldsymbol{b}$ a n componenti, restituisca 1 se $\boldsymbol{b}$ è uno shift ciclico destro di indice pari (cioè, di un numero pari di posizioni) di $\boldsymbol{a}$, e 0 altrimenti. La complessità dell'algoritmo deve essere $O(n\log n)$.

(*Suggerimento:* E' necessario usare un'opportuna convoluzione tra $\boldsymbol{a}$ e ...)

Answer: Observe that the i-th component of $\boldsymbol{a} \circledast \boldsymbol{b}$ is the sum of the n component-wise products of $\boldsymbol{a}$ and r_shift$(\boldsymbol{b}^R, i)$, where $\boldsymbol{b}^R$ denotes vector $(b_0, b_{n-1}, b_{n-2}, \ldots, b_1)$, and r_shift$(\boldsymbol{b}^R, i)$ denotes the cyclic right shift of $\boldsymbol{b}^R$ of i components. Since $(\boldsymbol{b}^R)^R = \boldsymbol{b}$, the i-th component of $\boldsymbol{a} \circledast \boldsymbol{b}^R$ is the sum of the n component-wise products of $\boldsymbol{a}$ and r_shift$(\boldsymbol{b}, i)$.

Let k_a (resp., k_b) be the number of 1-components of $\boldsymbol{a}$ (resp., $\boldsymbol{b}$). Clearly, if $k_a \neq k_b$ then $\boldsymbol{b}$ cannot be a cyclic shift of $\boldsymbol{a}$. Otherwise, for $0 \leq i < n/2$, $\boldsymbol{a} =$ r_shift$(\boldsymbol{b}, 2i)$ if and only if the $2i$-th component of $\boldsymbol{a} \circledast \boldsymbol{b}^R$ is equal to k (all the ones of a "match" all the ones of r_shift$(\boldsymbol{b}, 2i)$). This immediately yields a simple $O(n\log n)$ algorithm for our problem. □

Esercizio 3 [6 punti] Sia $G = (V, E)$, con $|V| = n$, un grafo diretto non pesato. Data A, la matrice di incidenza per gli archi di G, con

$$1 \leq i, j \leq n: \quad A[i,j] = \begin{cases} 1 & (i,j) \in E, \\ 0 & \text{altrimenti}, \end{cases}$$

si vuole determinare, per ogni coppia di nodi (i, j), $1 \leq i, j \leq n$, se esiste in G un cammino tra i e j di lunghezza al più $n/2$. Si fornisca un algoritmo di che risolva il problema in tempo $\Theta(n^3)$ e spazio $\Theta(n^2)$.

Attenzione: Si provi la correttezza dell'algoritmo e se ne scriva lo pseudocodice.

Answer: We can simply use the Floyd-Warshall algorithm with weight function $w(e) = 1$ if $e \in E$ and $w(e) = +\infty$ otherwise. Indeed, any noninfinite entry in the output matrix corresponds to a pair of nodes for which a directed path exists, and viceversa. Hence, we can simply check each entry of the output matrix and reset it to one if its value does not exceed $n/2$. The Floyd-Warshall algorithm requires $O(n^3)$ time and $O(n^2)$ space, as required by the exercise. □

Esercizio 4 [7 punti] Si consideri il seguente problema decisionale:

4-REDUNDANT SAT (4R-SAT):

ISTANZA: $\langle \Phi(x_1, ..., x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m \rangle$, Φ formula booleana CNF (ogni clausola C_i è una disgiunzione di un numero arbitrario di letterali).

DOMANDA: Esiste un assegnamento di valori di verità alle n variabili sotto cui *ogni* clausola C_i contiene almeno quattro letterali veri?

Dimostrare che 4R-SAT è NP-Hard.

Answer: We prove that 4R-SAT is NP-Hard by reducing from 3-CNF SAT. The reduction function f is defined as follows. Given an instance of 3-CNF SAT $\langle \Phi(x_1, ..., x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m \rangle$, we consider three new variables, x_{n+1} and x_{n+2} and x_{n+3} and set

$$f(\langle \Phi(x_1, \ldots, xn) \rangle) = \langle \Phi'(x_1, \ldots, x_n, x_{n+1}, x_{n+2}) = C'_1 \wedge C'_2 \wedge \ldots \wedge C'_m \rangle,$$

where $C'_i = C_i \cup \{x_{n+1}, x_{n+2}, x_{n+3}\}$, for $1 \leq i \leq m$. If Φ is in 3-CNF SAT, then there exists an assignment to $x_1, \ldots, x_n$ under which every clause C_i contains a true literal. Hence, by augmenting that assignment with $x_{n+1} = x_{n+2} = x_{n+3} = 1$, we obtain an assignment for Φ' under which every clause C'_i contains at least four true literals. Viceversa, if Φ' is in 4R-SAT, then there is an assignment under which each of its clauses contains at least a true literal relative to one of the first n variables. By restricting such an assignment to $x_1, \ldots, x_n$ we then obtain a satisfying assignment for Φ. Clearly, f is computable in polynomial (in fact, linear) time, hence 3-CNF SAT $<_P$ 4R-SAT. □

Esercizio 5 [6 punti] Si consideri il seguente sistema di equazioni modulari:

$$\begin{cases} x &= 1 \bmod 3 \\ x &= 2 \bmod 5 \end{cases}$$

Utilizzando il teorema cinese dei resti, determinare l'unica soluzione al sistema in $\boldsymbol{Z}_{15}$.

Answer: One could use the analytic definition of the bijection between $\boldsymbol{Z}_{15}$ and $\boldsymbol{Z}_3 \times \boldsymbol{Z}_5$ implied by the Chinese Reminder Theorem. We have $m_1 = 15/3 = 5$, $m_2 = 15/5 = 3$, ${m_1}^{-1} \bmod 3 = 2$, ${m_2}^{-1} \bmod 5 = 2$, $c_1 = 5 \cdot 2 = 10$, $c_2 = 3 \cdot 2 = 6$. Hence the (unique) solution to the above system of equations is $a = (1 \cdot 10 + 2 \cdot 6) \bmod 15 = 22 \bmod 15 = 7$, whose correctness can be easily checked directly. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(7/9/2001)

Durata: 2h 45m

Esercizio 1 [6 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro $n = 2^{2^i}$:

$$T(n) = \begin{cases} \sqrt{n}T\left(\sqrt{n}\right) + n\log^2 n, & n > 2, \\ 0 & n = 2. \end{cases}$$

Esercizio 2 [7 punti] Sia n una potenza di due. Si fornisca lo pseudocodice e si analizzi un algoritmo che, dato in ingresso un vettore complesso $\boldsymbol{a} \in \mathbf{C}^n$, fornisca in uscita il numero di soluzioni distinte in $\mathbf{C}^n$ all'equazione vettoriale

$$(\boldsymbol{x} + \boldsymbol{a}) \circledast (\boldsymbol{x} - \boldsymbol{a}) = 0,$$

dove $\boldsymbol{x}$ è il vettore delle n incognite complesse e $\circledast$ denota l'operatore di convoluzione ciclica. L'algorimo deve avere complessità $O(n \log n)$.

Esercizio 3 [7 punti] La seguente ricorrenza definisce i *numeri di Stirling del secondo tipo*:

$$\begin{cases} S_2(0,0) = 1 & \\ S_2(n,0) = 0 & n > 0 \\ S_2(n,n) = 1 & n > 0 \\ S_2(n,k) = kS_2(n-1,k) + S_2(n-1,k-1) & 0 < k < n. \end{cases}$$

(a) Si determini il tempo necessario per calcolare $S_2(n,k)$ con l'algoritmo divide and conquer basato sulla definizione della ricorrenza.

(b) Si fornisca lo pseudocodice e si analizzi la complessità della versione memoizzata dello stesso algoritmo.

Esercizio 4 [7 punti] Si consideri il seguente problema decisionale:

FORMULA DOUBLE DISCREPANCY (F-DD):

ISTANZA: $\langle \Phi(x_1, \ldots, x_n), \Psi(x_1, \ldots, x_n) \rangle$, Φ, Ψ formule booleane.

DOMANDA: Esistono due assegnamenti **distinti** $(b_1, \ldots, b_n)$ e $(b'_1, \ldots, b'_n)$ per cui $\Phi(b_1, \ldots, b_n) \neq \Psi(b_1, \ldots, b_n)$ e $\Phi(b'_1, \ldots, b'_n) \neq \Psi(b'_1, \ldots, b'_n)$?

Dimostrare che F-DD è NP-Hard.

Esercizio 5 [6 punti] Si consideri il seguente sistema di equazioni modulari:

$$\begin{cases} x &= 1 \bmod 2 \\ x &= 6 \bmod 11 \end{cases}$$

Utilizzando il teorema cinese dei resti, determinare l'unica soluzione al sistema in $\boldsymbol{Z}_{22}$.

Answer: One could use the analytic definition of the bijection between $\boldsymbol{Z}_{22}$ and $\boldsymbol{Z}_2 \times \boldsymbol{Z}_{11}$ implied by the Chinese Reminder Theorem. We have $m_1 = 22/2 = 11$, $m_2 = 22/11 = 2$, ${m_1}^{-1} \bmod 2 = 1$, ${m_2}^{-1} \bmod 11 = 6$, $c_1 = 11 \cdot 1 = 11$, $c_2 = 2 \cdot 6 = 12$. Hence the (unique) solution to the above system of equations is $a = (1 \cdot 11 + 6 \cdot 12) \bmod 22 = 83 \bmod 22 = 17$, whose correctness can be easily checked directly. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO +SOLUZIONI
(21/9/2001)

Esercizio 1 [6 punti] Si determini l'ordine di grandezza della seguente ricorrenza, per valori arbitrari del parametro $n > 0$:

$$T(n) = \begin{cases} 1 & n = 1, \\ 2T\left(\lfloor \frac{n}{2} \rfloor\right) + n^2, & n \geq 2. \end{cases}$$

Answer: When the parameter n is a power of two, the above recurrence yields $T(n) = \Theta(n^2)$. Let us now extend this result to arbitrary values of n. Since $T(k) \geq 0$ for any natural k, we have that $T(n) = 2T(\lfloor n/2 \rfloor) + n^2 \geq n^2$, whence $T(n) = \Omega(n^2)$. Next, we "guess" that $T(n) \leq cn^2$, for some constant $c > 0$ and try to find suitable values for c by means of parametric induction. For the base case, we have

$$T(1) = 1 \leq c \cdot 1^2 \Leftrightarrow c \geq 1.$$

Assuming that $T(k) \leq ck^2$, for $1 \leq k < n$, we get:

$$\begin{aligned} T(n) &\leq 2c\left(\left\lfloor \frac{n}{2} \right\rfloor\right)^2 + n^2 \\ &\leq 2c\left(\frac{n}{2}\right)^2 + n^2 \\ &= \left(\frac{c}{2} + 1\right) n^2. \end{aligned}$$

It then suffices that c be such that

$$\left(\frac{c}{2} + 1\right) n^2 \leq cn^2,$$

which is true whenever $c \geq 2$. As a consequence, $T(n) \leq \max\{1, 2\}n^2 = 2n^2$, which can be easily verified by induction. □

Esercizio 2 [7 punti] Sia n una potenza di due. Si sviluppi e si analizzi un algoritmo che, dati in ingresso due vettori reali $\boldsymbol{a}$ e $\boldsymbol{b}$ a n componenti, restituisca il valore

$$M = \max_{0 \leq i < n} \{\boldsymbol{a} \cdot \text{r\_shift}(\boldsymbol{b}, i)\}$$

dove r_shift$(\boldsymbol{b}, i)$ denota lo shift ciclico destro di i posizioni del vettore $\boldsymbol{b}$, e $\cdot$ denota l'operazione di prodotto scalare tra due vettori. La complessità dell'algoritmo deve essere $O(n \log n)$.
(*Suggerimento:* E' necessario usare un'opportuna convoluzione tra $\boldsymbol{a}$ e ...)

Answer: Recall that for any two n-component vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, the scalar product $\boldsymbol{x} \cdot \boldsymbol{y}$ is defined as $\sum_{j=0}^{n-1} x_j y_j$. Let now $\circledast$ denote cyclic convolution. From the definition, for

$0 \leq i < n$, we have

$$\begin{aligned}(\boldsymbol{x} \circledast \boldsymbol{y})_i &= \sum_{j=0}^{n-1} x_j y_{(i-j) \bmod n} \\ &= x_0 y_i + x_1 y_{i-1} \cdots + x_i y_0 + x_{i+1} y_{n-1} + \cdots + x_{n-1} y_{i+1}\end{aligned}$$

and it can be immediately seen that the latter quantity above is $\boldsymbol{x} \cdot \text{r\_shift}(\boldsymbol{y}^R, i)$, where $\boldsymbol{y}^R$ denotes the vector $(y_0, y_{n-1}, y_{n-2}, \ldots, y_1)$.

Since $(\boldsymbol{b}^R)^R = \boldsymbol{b}$, it follows from above that the i-th component of $\boldsymbol{a} \circledast \boldsymbol{b}^R$ is $\boldsymbol{a} \cdot \text{r\_shift}(\boldsymbol{b}, i)$. Then, it suffices to compute such a convolution in $O(n \log n)$ using the cyclic convolution theorem and the FFT algorithm and then compute its maximum component in $O(n)$ time. This immediately yields a simple $O(n \log n)$ algorithm for the problem. □

Esercizio 3 [7 punti] Data una stringa di naturali $Z = \langle z_1, z_2, \ldots, z_k \rangle$, si definisca peso($Z$) la quantità $\sum_{i=1}^{k} z_i$. Si consideri ora il seguente problema: date due stringhe di naturali $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$, si voglia determinare il peso massimo di una Longest Common Subsequence (LCS) di X e Y (si ricordi che possono esserci molte LCS distinte di due stringhe). Si progetti un algoritmo di programmazione dinamica che risolve il problema in tempo $O(mn)$.

Attenzione: E' necessario provare la correttezza dell'algoritmo e fornirne lo pseudocodice.

Answer: Let $Z = \langle z_1, z_2, \ldots z_k \rangle$ be a Max-Weight LCS (MW-LCS, for short) of X and Y. The optimal substructure property upon which we will base our algorithm is the following:

(a) If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-1} is a MW-LCS of X_{m-1} and Y_{n-1};

(b) If $x_m \neq y_n$ then Z is a maximum length string between an MW-LCS of X_m and Y_{n-1} and an MW-LCS of X_{m-1} and Y_n. In case both such LCS's have the same length, Z is the one with maximum weight between the two.

Part (a) can be proved by first observing that in case $z_k \neq x_m = y_n$ then Zx_n would be a longer LCS for X and Y, which is absurd. Similarly, Z_{k-1} must be an LCS for X_{m-1} and Y_{n-1}, moreover, it must be a MW-LCS or otherwise we could produce a heavier LCS than Z for X and Y. Part (b) can be proved in a similar fashion: clearly, if $x_m \neq y_n$, then Z must either be an MW-LCS of X_m and Y_{n-1}, or an MW-LCS of X_{m-1} and Y_{n-1}, whichever is the longest. If both the above LCS's have the same length, then Z must be the one of largest weight between the two, or Z would not be a MW-LCS for X and Y.

Finally, observe that MW-LCS(X, ϵ) = MW-LCS$(\epsilon, X) = \epsilon$.

Let $L[i, j]$ represent the length of an LCS of X_i and Y_j (with $X_0 = Y_0 = \epsilon$). We have:

$$\begin{aligned}L[i,0] &= 0 && 0 \leq i \leq m \\ L[0,j] &= 0 && 0 \leq j \leq n \\ L[i,j] &= \begin{cases} 1 + L[i-1, j-1] & X_i = Y_j, \\ \max\{L[i, j-1], L[i-1, j] & X_i \neq Y_j \end{cases} && 1 \leq i \leq m, 1 \leq j \leq n\end{aligned}$$

Now, let $MW[i,j]$ represent the weight of a MW-LCS of X_i and Y_j. We can write the following recurrence:

$$MW[i,0] = 0 \qquad 0 \le i \le m$$

$$MW[0,j] = 0 \qquad 0 \le j \le n$$

$$MW[i,j] = \begin{cases} X_i + MW[i-1,j-1] & X_i = Y_j, \\ MW[i,j-1] & X_i \neq Y_j \\ & \text{and } L[i,j-1] > L[i-1,j], \\ MW[i-1,j] & X_i \neq Y_j \\ & \text{and } L[i-1,j] > L[i,j-1], \\ \max\{MW[i,j-1], MW[i-1,j]\} & \text{otherwise,} \end{cases} \quad \text{otherwise.}$$

The above two recurrences can be immediately transformed into an algorithm for computing $MW[m,n]$ in time $O(mn)$. □

Esercizio 4 [7 punti] Dati in input due naturali x e y si consideri il seguente algoritmo per il calcolo di $\lfloor x/y \rfloor$:

```
DIV(x, y)
    if y = 0 then return errore
    if x < y then return 0
    return 1+ DIV(x − y, y)
```

Si dimostri che l'algoritmo DIV ha complessità esponenziale, al caso pessimo, sotto un encoding binario degli input, e complessità lineare sotto un encoding unario degli stessi.

Answer: Let $k = \lfloor x/y \rfloor$. DIV(x,y) computes $\lfloor x/y \rfloor$ as

$$\underbrace{1 + \cdots + 1}_{k \text{ times}},$$

hence its running time, as a function of the *values* x and y, is $k = \lfloor x/y \rfloor$. Note that when $y = 1$, we have $k = x$. Since, under a binary encoding, we have $| < x, 1 > | = \Theta(\log x)$, the running time is then exponential in the size of the encoding. However, under the unary encoding, we have $| < x, y > | = \Theta(x + y) = \Omega(k)$, hence the running rime is at most linear in the size of the encoding. □

Esercizio 5 [6 punti] Si risolva il seguente sistema di equazioni modulari in $\boldsymbol{Z}_{909}$ utilizzando il Teorema Cinese dei Resti:

$$\begin{cases} x &= 2 \bmod 9 \\ x &= 83 \bmod 101 \end{cases}$$

Answer: We could use the analytic definition of the bijection between $\boldsymbol{Z}_{909}$ and $\boldsymbol{Z}_9 \times \boldsymbol{Z}_{101}$ implied by the Chinese Reminder Theorem, but we can get to the solution more directly by noting that 83 trivially satisfies both equations, since $83 = 2 \bmod 9$ and $83 = 83 \bmod 101$. This completes the exercise, since the Chinese Reminder Theorem implies that there is a single solution to the system in $\boldsymbol{Z}_{909}$. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITINO 1
(19/11/2001)

Durata: 1h30m

Esercizio 1 [10 punti] Sia n una potenza di due. Dato un problema Π, sia A^1_Π un algoritmo ricorsivo per Π la cui complessità in tempo $T_1(n)$ obbedisce alla seguente ricorrenza:

$$T_1(n) = \begin{cases} 1 & n = 1, \\ 2T_1\left(\frac{n}{2}\right) + 2n & n > 1. \end{cases}$$

Sia A^2_Π un ulteriore algoritmo per Π di complessità $T_2(n) = n^{3/2}$. Si determini la complessità in tempo del migliore algoritmo ibrido per Π ottenibile combinando A^1_Π e A^2_Π.

Answer: Let $i \in \mathcal{I}_\Pi$ be an instance of Π, with size$(i) = n$, and let n_0 be a constant parameter (power of two) to be determined. In order to obtain a hybrid algorithm by combining A^1_Π and A^2_Π, we change the base case in the code of A^1_Π, as follows.

```
HYBRID(i)
n ← size(i)
if n ≤ n0
    then return A2Π(i)
...
{ same code as in A1Π }
...
```

The recurrence associated with the new algorithm is parametric in both n and n_0:

$$T_{n_0}(n) = \begin{cases} n^{3/2} & n \leq n_0, \\ 2T_{n_0}\left(\frac{n}{2}\right) + 2n & n > n_0. \end{cases}$$

Let $n \geq n_0$. By using the iterated method, we obtain:

$$T_{n_0}(n) = 2^i T_{n_0}\left(\frac{n}{2^i}\right) + i \cdot 2n, \quad 1 \leq i \leq \log(n/n_0).$$

By substituting $i = \log(n/n_0)$ in the above formula, we obtain:

$$T_{n_0}(n) = \left(\frac{n}{n_o}\right) n_0^{3/2} + \log(n/n_0) \cdot 2n = 2n \log n + n(\sqrt{n_0} - 2 \log n_0)$$

(note that $T_{n_0}(n_0) = n_0^{3/2}$). In order to obtain the optimal choice for n_0, it suffices to compute the partial derivative of $T(n, n_0)$ with respect to n_0 and equate it to zero:

$$\frac{\partial T_{n_0}(n)}{\partial n_0} = n\left(\frac{1}{2\sqrt{n_0}} - \frac{2}{n_0 \ln 2}\right) = 0 \Leftrightarrow \sqrt{n_0} = \frac{4}{\ln 2} \Leftrightarrow n_0 = \frac{16}{\ln^2 2} \simeq 33.3.$$

Therefore, the best choice is either $\bar{n}_0 = 32$ or $\bar{n}_0 = 64$. Since $\sqrt{32} - 2\log 32 < \sqrt{64} - 2\log 64$, we choose $\bar{n}_0 = 32$, which yields the best hybrid algorithm. □

Esercizio 2 [10 punti] Sia n una potenza di due. Siano A e B due matrici complesse $n \times n$, con $A = C^c(\boldsymbol{a})$ matrice circolante con prima colonna $\boldsymbol{a}$ e B matrice arbitraria. Si progetti e si analizzi un algoritmo che, dati in ingresso il vettore $\boldsymbol{a}$ e la matrice B, restituisca il prodotto $A \times B$ in tempo $\Theta(n^2 \log n)$. Nell'analisi di complessità, si considerino di costo unitario e non nullo solo le operazioni aritmetche tra scalari.

Answer: Let $C = C^c(\boldsymbol{a}) \times B$, and let C^j denote the j-th column of C. Since

$$C_i^j = C_{i,j} = \sum_{k=0}^{n-1} [C^c(\boldsymbol{a})]_{i,k} [B]_{k,j} = \sum_{k=0}^{n-1} [C^c(\boldsymbol{a})]_{i,k} [B^j]_k \quad 0 \leq i, j < n,$$

it follows that C^j is obtained as the matrix-vector product of $C^c(\boldsymbol{a})$ with the j-th column of B, B^j. By the properties of circulant matrices we have:

$$C^j = C^c(\boldsymbol{a}) \times B^j = B^j \circledast \boldsymbol{a}.$$

This observation immediately yields the following algorithm for the problem:

```
CIRC_PROD(a, B)
n ← length(a)
A ← FFT(a)
for j ← 0 to n − 1 do
    B^j ← (B_{0,j}, B_{1,j}, ..., B_{n−1,j})
    BJ ← FFT(B^j)
    for i ← 0 to n − 1 do
        CJ_i ← A_i · BJ_i
    C^j ← INV_FFT(CJ)
    for i ← 0 to n − 1 do
        C_{i,j} ← C_i^j
return C
```

The time complexity of CIRC_PROD can be easily established as follows: there are n iterations, and in each iteration we perform one call to FFT, one to INV_FFT, and a cycle with n arithmetic operations, for a total of $\Theta(n \log n)$ time per cycle. Hence the overall time complexity is $\Theta(n^2 \log n)$, as required. □

Esercizio 3 [10 punti] Dati in ingresso un grafo connesso non orientato $G = (V, E)$ e due nodi $u, v \in V$, progettare e analizzare un algoritmo che ritorni il valore **true** se esistono in G due cammini da u a v, $\pi_{u,v} = \langle u = v_0, v_1, \ldots, v_{k-1}, v_k = v \rangle$ e $\pi'_{u,v} = \langle u = v'_0, v'_1, \ldots, v'_{h-1}, v'_h = v \rangle$ **disgiunti negli archi**, ovvero tali che

$$\{\{v_i, v_{i+1}\} : 0 \leq i < k\} \cap \{\{v'_i, v'_{i+1}\} : 0 \leq i < h\} = \emptyset,$$

e il valore **false** se tali due cammini non esistono. L'algoritmo deve avere complessità $\Theta(|V| + |E|)$.

Answer: We can solve the problem by applying the BFS algorithm twice. More specifically, we first call BSF(G, u) and obtain vectors $\boldsymbol{p}$, which represents a shortest-path tree $\mathcal{T}$ rooted at u for G, and vector $\boldsymbol{d}$, which reports, for each node v, the path length in $\mathcal{T}$ from u to v. Let $k = d[v]$. For the second run of BFS, we will consider

G as a directed graph and modify its edge set appropiately (recall that the adjacency list representation of an undirected graph is undistinguishable from that of a directed graph where for each edge $(u, v) \in E$, E also contains edge (v, u)). More specifically, we eliminate the following (directed) edges: consider the path $\pi_{u,v} = \langle u = v_0, v_1, \ldots, v_k = v \rangle$ and remove the **directed edges** $\{(v_i, v_{i+1}) : 0 \leq i < k\}$ from E. In other words, edges on $\pi_{u,v}$ can only be traversed backwards in the modified graph.

Finally, we run another BFS with source u on the modified graph. If there exists a second path from u to v in G, then v will still be reachable from u, hence $p[v] \neq \textbf{nil}$, in which case we return **true**. Otherwise, we return **false**. The algorithm follows immediately from the above discussion.

```
TWO-PATHS(G, u, v)
(p, d) ← BFS(G, u)
s ← p[v]; t ← v
for i ← 1 to d[k] do
    Adj[s] ← Adj[s] − {t}
    t ← s; s ← p[s]
    { In order to remove directed edge (s, t) from E
      we must simply remove t from Adj[s]}
(p, d) ← BFS(G, u)
if p[v] ≠ nil
    then return true
    else return false
```

The time complexity $T(|V|, |E|)$ of the above algorithm is determined by the two calls to BFS plus a double linear scan of the adjacency lists of $O(|V|)$ nodes (to remove the edges on the first path), which all take $\Theta(|V| + |E|)$ time . As a consequence, we obtain $T(|V|, |E|) = \Theta(|V| + |E|)$, as required.

The correctness of the algorithm requires a more complicate argument, which is based on the following property: there are no two disjoint paths in the network if and only if there is a a way to partition V in two sets, V_u and V_v, with $u \in V_u$ and $v \in V_v$, so that there is a single edge $\{s, t\}$ with $s \in V_u$ and $t \in V_v$. (this is a corollary, for instance, of the Min-Cut/Max-Flow theorem). Note that if this is the case, all the (nondisjoint) paths from u to v must use $\{s, t\}$ *in the same direction*, i.e., either (s, t) or (t, s). Then, by changing the orientation of the edges on the first path, we disconnect u from v. If, instead, u and v are still connected after removing the (directed) edges on the first path, we can state that two disjoint paths in G exist.

Indeed, the two disjoint paths can be obtained by "combining" the two paths found by the algorithm as follows. If the paths are edge-disjoint, then we are finished. If they are not, then each common (undirected) edge is used by the two paths in opposite directions. We can simply remove these edges and rearrange the remaining (disjoint) segments to obtain the two paths. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITINO 2
(21/1/2001)

Durata: 1h45m

Esercizio 4 [10 punti] Sia $n > 0$. Si supponga che una certa strategia di programmazione dinamica conduca alla seguente ricorrenza, definita per tutti i valori di i e j con $1 \leq i \leq j \leq n$:

$$C(i,j) = \begin{cases} 1 & (i = 1) \textbf{ and } (j = n), \\ \sum_{r=1}^{i-1} C(r,j) + \sum_{s=j+1}^{n} C(i,s) & \text{altrimenti.} \end{cases}$$

Si fornisca un algoritmo iterativo *bottom-up* che restituisca i valori $C(i,j)$, $1 \leq i \leq j \leq n$. Si analizzi la correttezza e la complessità dell'algoritmo.

Answer: Let us first begin with a simple $O(n^3)$ algorithm. Consider a table $C[i,j]$ which stores the value of $C(i,j)$, for $1 \leq i \leq j \leq n$. From the definition of T it is clear that we can compute $C[i,j]$ if the table entries $C[r,j]$, for $1 \leq r < i$ and $C[i,s]$, for $j < s \leq n$ have already been computed. Let $\ell = j - i + 1$ (ℓ can be seen as the "size" of "subproblem" (i,j)). We have that $j - r + 1 > j - i + 1 = \ell$ and $s - i + 1 > j - i + 1 = \ell$, hence we can organize the iterations by decreasing size. The code immediately follows.

```
COMPUTE_T(n)
C[1,n] ← 1
for ℓ ← n − 1 downto 1 do
    { process subproblems of decreasing length ...}
    for i ← 1 to n − ℓ + 1 do
        j ← i + ℓ − 1
        { ready to compute C[i,j] }
        C[i,j] ← 0
        for r ← 1 to i − 1 do
            C[i,j] ← C[i,j] + C[r,j]
        for s ← j + 1 to n do
            C[i,j] ← C[i,j] + C[i,s]
return C
```

The correctness of the code follows from the fact that the values $C[r,j]$ and $C[i,s]$ have already been (correctly) computed when used. The running time of the above algorithm is evaluated as follows:

$$\begin{aligned} T_{\text{COMP}}(n) &= \sum_{\ell=1}^{n-1} \sum_{i=1}^{n-\ell+1} [(i-1) + (n-i-\ell+1)] \\ &= \sum_{\ell=1}^{n-1} \sum_{i=1}^{n-\ell+1} (n-l) = \sum_{\ell=1}^{n-1} (n-\ell+1)(n-\ell) \\ &= \sum_{\ell=1}^{n-1} (\ell+1)\ell = \sum_{\ell=1}^{n-1} \ell^2 + \sum_{\ell=1}^{n-1} \ell = \Theta(n^3) \end{aligned}$$

A much more efficient algorithm can be obtained by observing that each $C(i,j)$ is the sum of two terms. By keeping the two terms separate, storing them into two vectors, we can reduce the time to compute a single entry of C to a constant. Specifically, define the following two vectors as a function of vector C:

$$Up[i,j] = \begin{cases} 1 & (i=1) \textbf{ and } (j=n), \\ \sum_{r=1}^{i-1} C[r,j] & \text{otherwise;} \end{cases}$$

and

$$Right[i,j] = \sum_{s=j+1}^{n} C[i,s].$$

Note that from the definition of $C(i,j)$, it immediately follows that $C[i,j] = Up[i,j] + Right[i,j]$. Moreover, the following recurrences hold:

$$Up[i,j] = \begin{cases} 1 & i=1, j=n, \\ 0 & i=1, j \neq n \\ Up[i-1,j] + C[i-1,j] & \text{otherwise;} \end{cases}$$

and

$$Right[i,j] = \begin{cases} 0 & j=n \\ Right[i,j+1] + C[i,j+1] & \text{otherwise;} \end{cases}$$

Note that in order to compute the values at coordinates (i,j) for vectors C, Up and $Right$, we only need values at coordinates $(i-1,j)$ and $(i,j+1)$. Hence we can keep the same type of scan used in the previous algorithm. We obtain the following code:

```
COMPUTE2_T(n)
Up[1,n] ← 1
Right[1,n] ← 0
C[1,n] ← 1
for ℓ ← n − 1 downto 1 do
    Up[1,ℓ] ← 0
    Right[1,ℓ] ← Right[1,ℓ+1] + C[1,ℓ+1]
    C[1,ℓ] ← Right[1,ℓ]
    {Element of length ℓ on the first row}
    for i ← 2 to n − ℓ do
        j ← i + ℓ − 1
        Up[i,j] ← Up[i−1,j] + C[i−1,j]
        Right[i,j] ← Right[i,j+1] + C[i,j+1]
        C[i,j] ← Up[i,j] + Right[i,j]
    Right[n−ℓ+1,n] ← 0
    Up[n−ℓ+1,n] ← Up[n−ℓ,n] + C[n−ℓ,n]
    C[n−ℓ+1,n] ← Up[n−ℓ+1,n]
    {Element of length ℓ on the last column}
return C
```

As before, all values needed in an iteration have been computed previously, hence the code is correct. The new algorithm performs a constant number of operations for each entry of C to be computed, hence its running time is $\Theta(n^2)$. □

Esercizio 5 [10 punti] Si consideri il seguente problema decisionale:

2-NON-TAUTOLOGY (2NT):
ISTANZA: $\langle\Phi(x_1, x_2, \ldots x_n)\rangle$, $\Phi(x_1, x_2, \ldots x_n)$ formula booleana
DOMANDA: Esistono almeno due assegnamenti distinti di valori di verità alle variabili di Φ che **falsificano** la formula?

Si dimostri che 2NT è NP-Hard.

Answer: We show that FNT is NP-Hard by reducing from SAT. Let $\Phi(x_1, x_2, \ldots x_n)$ be an instance of SAT, and consider the following reduction:

$$f(\Phi(x_1, x_2, \ldots x_n)) = \Psi(x_1, x_2, \ldots x_n, x_{n+1}) = \neg\Phi(x_1, x_2, \ldots x_n) \vee (x_{n+1} \wedge \neg x_{n+1})$$

Clearly, f can be computed in polynomial (in fact, linear) time. Let us now show that f reduces SAT to 2NT.

If $\Phi \in$ SAT, then there exists a truth assignment $(b_1, b_2, \ldots b_n)$ to the n variables such that $\Phi(b_1, b_2, \ldots b_n) =$ **true**. Then, the two assignments $(b_1, b_2, \ldots b_n, 0)$ and $(b_1, b_2, \ldots b_n, 1)$ both falsify $\Psi = f(\Phi)$, hence $f(\Phi) \in$ 2NT.

Conversely, if $\Phi \notin$ SAT, then $\neg\Phi$ is a tautology, which remains a tautology when disjuncted with $(x_{n+1} \wedge \neg x_{n+1})$. Since $\Psi = f(\Phi)$ is a tautology, we have that $f(\Phi) \notin$ 2NT. □

Esercizio 6 [10 punti] Si consideri un criptosistema a chiave pubblica di tipo RSA in cui il dominio dei messaggi è $\boldsymbol{Z}_n$, con $n = 11 \times 23 = 253$. Data la chiave pubblica $P = (7, 253)$ determinare la corrispondente chiave segreta S e la relativa funzione di decodifica $S(M)$ ad essa associata.

Answer: We have that $\phi(n) = (11 - 1) \cdot (23 - 1) = 220$. Given the public key $P = (7, 253)$, in order to obtain the corresponding secret key, we must determine the multiplicative inverse of 7 in $\boldsymbol{Z}^{\star}_{220}$. In turn, such an inverse can be obtained by applying Algorithm EXTENDED EUCLID (EE) to determine Bezout relation between 1=gcd(220, 7), 220 and 7. On (220, 7), EE performs calls on (7, 3), (3, 1), and finally (1, 0). From bottom up, the values returned by these calls are $(1, \{1, 0\})$, $(1, \{0, 1\})$, $(1, \{1, -2\})$ and finally $(1, \{-2, 63\})$. It follows that $S = (63, 253)$, hence, for any message $M \in \boldsymbol{Z}_{253}$, $S(M) = M^{63} \bmod 253$. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(30/1/2002)

Durata: 2h 45m

Esercizio 7 [6 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro $n = 2^{2^i}$:

$$T(n) = \begin{cases} 4T(\sqrt{n}) + \log^2 n, & n > 2, \\ 0 & n = 2. \end{cases}$$

Answer: Let us apply the general formula with $s(n) = 4$, $f(n) = \sqrt{n}$, $w(n) = \log^2 n$, $T_0 = 0$ and $n_0 = 2$. We have $f^{(i)}(n) = n^{1/2^i}$ and $f^\star(n, 2) = \log\log n - 1$. Therefore:

$$\begin{aligned} T(n) &= \sum_{\ell=0}^{\log\log n - 1} 4^\ell \log^2 n^{1/2^\ell} \\ &= \sum_{\ell=0}^{\log\log n - 1} \frac{4^\ell}{2^{2\ell}} \log^2 n \\ &= \sum_{\ell=0}^{\log\log n - 1} \log^2 n \\ &= \log^2 n \log\log n. \end{aligned}$$

The correctness of the above solution can be immediately verified by induction. □

Esercizio 8 [7 punti] Sia $n = 2^{2k}$ una potenza pari di due. Dato un vettore $\boldsymbol{a} \in \boldsymbol{C}^{\sqrt{n}}$, si consideri il vettore a n componenti $\boldsymbol{b} \in \boldsymbol{C}^n$ ottenuto concatenando $\boldsymbol{a}$ con se stesso per $\sqrt{n}$ volte:

$$\boldsymbol{b} = (\underbrace{\boldsymbol{a}|\boldsymbol{a}\ldots|\boldsymbol{a}}_{\sqrt{n}\text{ volte}}).$$

Si progetti e si analizzi un algoritmo che, dato in ingresso $\boldsymbol{a}$, restituisca $DFT_n(\boldsymbol{b})$ in tempo $\Theta(\sqrt{n}\log n)$. Nell'analisi di complessità, si considerino di costo unitario e non nullo solo le operazioni aritmetiche tra numeri complessi.

Answer:

In order to gain information on the structure of $DFT_n(\boldsymbol{b})$, let us apply the Cooley-Tukey algorithm by folding $\boldsymbol{b}$ into a $\sqrt{n} \times \sqrt{n}$ (square) matrix B. The first step of the algorithm transforms each column B^j of B independently, for $0 \le j \le \sqrt{n} - 1$. Note that $B^j = (\underbrace{a_j, a_j, \ldots a_j}_{\sqrt{n}\text{ times}})$, hence

$$F_{\sqrt{n}} B^j = a_j \cdot F_{\sqrt{n}}(\underbrace{1, 1, \ldots 1}_{\sqrt{n}\text{ times}}) = (\sqrt{n} \cdot a_j, \underbrace{0, 0, \ldots 0}_{\sqrt{n}-1\text{ times}}),$$

by the summation lemma. In other words, after this step, the first row of B becomes $\sqrt{n} \cdot \boldsymbol{a}$, while all the other rows are zeroed. Consequently, the following multiplication

of each matrix entry $B_{i,j}$ by the twiddle factor ω_n^{ij} leaves B unchanged. Also, we can now limit ourselves to transform the first row, which clearly becomes $\sqrt{n} \cdot \boldsymbol{A}$, where $\boldsymbol{A} = DFT_{\sqrt{n}}(\boldsymbol{a})$, while all the other rows stay null. Finally, by reading the entries of B in colum-major order, we obtain:

$$DFT_n(\boldsymbol{b}) = \sqrt{n} \cdot (A_0, \underbrace{0, 0, \ldots 0}_{\sqrt{n}-1 \text{ times}}, A_1, \underbrace{0, 0, \ldots 0}_{\sqrt{n}-1 \text{ times}}, \ldots, A_{\sqrt{n}-1}, \underbrace{0, 0, \ldots 0}_{\sqrt{n}-1 \text{ times}}).$$

Analytically,

$$[DFT_n(\boldsymbol{b})]_i = \begin{cases} \sqrt{n} \cdot [DFT_{\sqrt{n}}(\boldsymbol{a})]_{i/\sqrt{n}}, & \text{if } i \bmod \sqrt{n} = 0, \\ 0, & \text{otherwise.} \end{cases}$$

This immediately yields the following code

```
CONCAT_FFT(a)
s ← length(a)
n ← s · s
A ← FFT(a)
for i ← 0 to n − 1 do
    if (i mod s) = 0
        then  B_i ← s · A_{i/s}
        else  B_i ← 0
return B
```

The correctness of the above algorithm follows from our previous discussion, while its running time is dominated by the $\Theta(\sqrt{n} \log \sqrt{n}) = \Theta(\sqrt{n} \log n)$ scalar complex operations performed by the call FFT($\boldsymbol{a}$). □

Esercizio 9 [7 punti] Siano $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots y_n \rangle$ due stringhe di m e n interi. Si definisca il *coefficiente di antinomia* tra le due stringhe come segue:

$$A(X, Y) = \begin{cases} 0, & (m < n) \vee (n = 0), \\ \max\{\sum_{j=1}^{n} |z_j - y_j| : \text{Z sottoseq. di } X, |Z| = n\}, & m \geq n \geq 1. \end{cases}$$

Si progetti e si analizzi un algoritmo di programmazione dinamica che, date in ingresso le stringhe X e Y, restituisca il loro coefficiente di antinomia $A(X, Y)$ in tempo $\Theta(mn)$.

Answer: Let us choose as our subproblem space the cartesian product of all prefixes of strings X and Y. Our dynamic programming algorithm will be based on the following optimal substructure property. Consider X_i and Y_j. Clearly, if $i < j$ or $j = 0$, it follows from the definition that $A(X_i, Y_j) = 0$. Otherwise, consider the subsequence $Z = \langle z_1, z_2, \ldots, z_j \rangle$ of X_i which realizes the maximum antinomy with Y_j. We have two cases. If $z_j = x_i$, then Z_{j-1} must be a subsequence of X_{i-1} realizing the maximum antinomy with Y_{j-1}, or we could obtain a larger antinomy for X_i and Y_j than the one realized by Z. If $z_j \neq x_i$, then Z is a subsequence of X_{i-1} realizing the maximum antinomy with Y_j, or again the antinomy between X_i and Y_j would be larger than the one realized by Z.

Let $a(i,j) = A(X_i, Y_j)$. The above substructure property immediately yields the following recurrence for $a(i,j)$:

$$a(i,j) = \begin{cases} 0 & \text{if } i < j \text{ or } j = 0 \\ \max\{|x_i - y_j| + a(i-1, j-1), a(i-1, j)\} & \text{otherwise.} \end{cases}$$

From the recurrence, we immediately obtain the desired algorithm.

```
ANTINOMY(X, Y)
m ← length(X)
n ← length(Y)
if(m < n) ∨ (n = 0) then return 0
for i ← 0 to m do
    a[i, 0] ← 0
    for j ← 1 to MIN(i, n) do
        a[i, j] ← MAX(|x_i − y_j| + a[i − 1, j − 1], a[i − i, j])
    if (i < n) then a[i, i + 1] ← 0
    { these are the only entries with j > i which might ever be used }
return a[m, n]
```

The correctness of the above algorithm follows from the correctness of the recurrence and from the fact that the row-major scan correctly computes every table entry prior to any of its uses. The running time of the algorithm is clearly $\Theta(mn)$. □

Esercizio 10 [7 punti] Si consideri il seguente problema decisionale:

LAST-INPUTS-FIXED SAT (LIF-SAT):

ISTANZA: $\langle \Phi(x_1, x_2, \ldots x_n), k \rangle$, Φ formula booleana e $1 \leq k \leq n$.

DOMANDA: Esiste un assegnamento $(b_1, b_2, \ldots, b_n)$ alle n variabili che soddisfa Φ, con $b_i =$ **false** per $n - k + 1 \leq i \leq n$?

Si dimostri che LIF-SAT è NP-Hard.

Answer: Consider the following reduction from SAT:

$$f(\langle \Phi(x_1, x_2, \ldots x_n) \rangle) = \langle \Phi'(x_1, x_2, \ldots x_n, x_{n+1}) = \Phi(x_1, x_2, \ldots x_n) \wedge \neg x_{n+1}, 1 \rangle.$$

Clearly, f is computable in polynomial (in fact, linear) time. Let us now show that f is a valid reduction from SAT to LIF-SAT. If $\langle \Phi(x_1, x_2, \ldots x_n) \rangle \in$ SAT, there exists a truth assignment $(b_1, b_2, \ldots, b_n)$ such that $\Phi(b_1, b_2, \ldots, b_n) =$ **true**. Therefore, the truth assignment $(b_1, b_2, \ldots, b_n, b_{n+1})$ with $b_{n+1} =$ **false** satisfies $\Phi'(x_1, x_2, \ldots x_n, x_{n+1}) = \Phi(x_1, x_2, \ldots x_n) \wedge \neg x_{n+1}$, whence $f(\langle \Phi \rangle) \in$ LIF-SAT. Vice versa, if $\Phi(x_1, x_2, \ldots x_n)$ is not satisfiable, neither can be $\Phi'(x_1, x_2, \ldots x_n, x_{n+1})$, hence $f(\langle \Phi \rangle) \notin$ LIF-SAT, which completes the proof. □

Esercizio 11 [6 punti] Si risolva il seguente sistema di equazioni modulari in $\boldsymbol{Z}_{808}$ utilizzando il Teorema Cinese dei Resti:

$$\begin{cases} x &= 6 \bmod 8 \\ x &= 78 \bmod 101 \end{cases}$$

Answer: We could use the analytic definition of the bijection between $\boldsymbol{Z}_{808}$ and $\boldsymbol{Z}_8 \times \boldsymbol{Z}_{101}$ implied by the Chinese Reminder Theorem, but we can get to the solution more directly by noting that 78 trivially satisfies both equations, since $78 = 6 \bmod 8$ and $78 = 78 \bmod 101$. This completes the exercise, since the Chinese Reminder Theorem implies that there is a single solution to the system in $\boldsymbol{Z}_{808}$. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(13/2/2002)

Durata: 2h 45m

Esercizio 12 [7 punti] Al variare dei due parametri interi positivi a e b, si consideri la seguente famiglia di ricorrenze, definite per $n = 2^{2^i}$:

$$T_{a,b}(n) = \begin{cases} 2^a T_{a,b}(\sqrt{n}) + \log^b n, & n > 2, \\ 0 & n = 2. \end{cases}$$

Si determini l'ordine di grandezza di $T_{a,b}(n)$ in funzione di n e dei due parametri a e b. (*Suggerimento:* Si applichi la formula generale e si discutano vari casi a seconda dei valori di a e b ...)

Answer: Let us apply the general formula with $s(n) = 2^a$, $f(n) = \sqrt{n}$, $w(n) = \log^b n$, $T_0 = 0$ and $n_0 = 2$. We have $f^{(i)}(n) = n^{1/2^i}$ and $f^*(n, 2) = \log\log n - 1$. Moreover, $\prod_{j=0}^{\ell-1} 2^a = 2^{a\ell}$ and $w(f^{(\ell)}(n)) = \log^b n^{1/2^\ell} = (1/2^{b\ell}) \log^b n$. Therefore:

$$\begin{aligned} T(n) &= \sum_{\ell=0}^{\log\log n - 1} 2^{(a-b)\ell} \log^b n \\ &= \log^b n \sum_{\ell=0}^{\log\log n - 1} (2^{a-b})^\ell. \end{aligned}$$

Let $S(n) = \sum_{\ell=0}^{\log\log n-1} (2^{a-b})^\ell$. We have three cases, depending on the values of a and b. If $a < b$, then $S(n) = \Theta(1)$, hence $T_{a,b}(n) = \Theta(\log^b n)$. If $a = b$, then $S(n) = \log\log n$, hence $T_{a,b}(n) = \Theta(\log^b n \log\log n)$. Finally, if $a > b$, then $S(n) = \Theta((2^{a-b})^{\log\log n}) = \Theta(\log^{a-b} n)$, hence $T_{a,b}(n) = \Theta(\log^a n)$. □

Esercizio 13 [6 punti] Sia $n = 2^{2k}$ una potenza pari di due. Dato un vettore $\boldsymbol{a} \in \boldsymbol{C}^{\sqrt{n}}$, si consideri il vettore $\boldsymbol{b} \in \boldsymbol{C}^n$ ottenuto inserendo $\sqrt{n} - 1$ componenti nulle dopo ogni componente di $\boldsymbol{a}$,

$$\boldsymbol{b} = (a_0, \underbrace{0, 0, \ldots 0}_{\sqrt{n}-1}, a_1, \underbrace{0, 0, \ldots 0}_{\sqrt{n}-1}, \ldots, a_{\sqrt{n}-1}, \underbrace{0, 0, \ldots 0}_{\sqrt{n}-1}).$$

Si progetti e si analizzi un algoritmo che, dato in ingresso $\boldsymbol{a}$, restituisca $DFT_n(\boldsymbol{b})$ in tempo $\Theta(\sqrt{n} \log n)$. Nell'analisi di complessità, si considerino di costo unitario e non nullo solo le operazioni aritmetiche tra numeri complessi.

Answer: In order to gain information on the structure of $DFT_n(\boldsymbol{b})$, let us apply the Cooley-Tukey algorithm by folding $\boldsymbol{b}$ into a $\sqrt{n} \times \sqrt{n}$ (square) matrix B. The first step of the algorithm transforms each column B^j of B independently, for $0 \le j \le \sqrt{n} - 1$. Note that $B^0 = \boldsymbol{a}$, while $B^j = \boldsymbol{0}$, for $1 \le j \le \sqrt{n} - 1$. In other words, after this step, the first column B^0 of B becomes $\boldsymbol{A} = DFT_n(\boldsymbol{a})$, while all the other columns remain null. Consequently, the following multiplication of each matrix entry $B_{i,j}$ by the twiddle factor

ω_n^{ij} leaves B unchanged. We now have to transform the rows of the matrix. Since row B_i is the vector $(A_i, 0, \ldots, 0)$, we have $DFT_{\sqrt{n}}(B_i) = F_{\sqrt{n}} B_i = (A_i, \ldots, A_i)$. Finally, by reading the entries of B in colum-major order, we obtain:

$$DFT_n(\boldsymbol{b}) = (\underbrace{\boldsymbol{A}|\boldsymbol{A}|\ldots|\boldsymbol{A}}_{\sqrt{n} \text{ times}}).$$

Analytically,

$$B_i = [DFT_n(\boldsymbol{b})]_i = [DFT_{\sqrt{n}}(\boldsymbol{a})]_{i \bmod \sqrt{n}} = A_{i \bmod \sqrt{n}}, 0 \leq i \leq n-1.$$

This immediately yields the following code

SQUARE_SPACED_FFT($\boldsymbol{a}$)
$s \leftarrow$ length($\boldsymbol{a}$)
$n \leftarrow s \cdot s$
$\boldsymbol{A} \leftarrow$ FFT($\boldsymbol{a}$)
for $i \leftarrow 0$ **to** $n-1$ **do**
    $B_i \leftarrow A_{i \bmod s}$
return $\boldsymbol{B}$

The correctness of the above algorithm follows from our previous discussion, while its running time is dominated by the $\Theta(\sqrt{n} \log \sqrt{n}) = \Theta(\sqrt{n} \log n)$ scalar complex operations performed by the call FFT($\boldsymbol{a}$). □

Esercizio 14 [7 punti] Dato $G = (V, E)$ un grafo non orientato con $V = \{v_1, v_2, \ldots, v_n\}$ e $|E| = m$, si consideri il problema di determinare se G è un albero. Si progetti e si analizzi un algoritmo che, dato in ingresso G (rappresentato con liste di adiacenza), n e m, risolva il problema in tempo $O(n)$. (**Attenzione:** Si noti che la complessità è indipendente da $m = |E|$.)

Answer: We make use of the following necessary and sufficient condition: an undirected graph $G = (V, E)$ is a tree if and only if G is connected and $m = |E| = |V| - 1 = n - 1$. We organize our algorithm as follows. First, the algorithm checks whether $m = n - 1$. If the check fails, then the algorithm returns **no** and correctly terminates (in constant time). Otherwise, the algorithm performs a BFS rooted at an arbitrary node s to obtain the vector of the shortest path lengths $\boldsymbol{d}$ and the predecessor vector $\boldsymbol{\pi}$. In order to check whether the graph is connected, we check whether $d[u] \neq \infty$ for all $u \in V - \{s\}$. If this is the case, the algorithm returns **yes**. Otherwise, the algorithm returns **no**. Observe that BFS is invoked on a graph with $|E| < |V|$, hence its running time is $O(|V| + |E|) = O(|V|) = O(n)$, which is also the running time of the entire algorithm, in the worst case. The code of the algorithm is the following.

IS_TREE($G = (V, E), n, m$)
if $m \neq n - 1$ **then return no**
$s \leftarrow v_1$
$(\boldsymbol{d}, \boldsymbol{\pi}) \leftarrow$ BFS(G, s)
for $i \leftarrow 2$ **to** n **do**
    if $d[v_i] = \infty$ **then return no**
return yes

The correctness and the running time analysis of the above algorithm follow immediately from the above discussion. □

Esercizio 15 [7 punti] Si consideri il seguente problema decisionale:

CLIQUE OR VERTEX COVER (CoVC):

ISTANZA: $\langle G = (V, E), h, k \rangle$, G grafo non orientato, $1 \leq h, k \leq |V|$.

DOMANDA: G ha una clique di taglia h *oppure* un vertex-cover di taglia h?

Si dimostri che CoVC è NP-Hard.

Answer: Consider the following reduction from VERTEX_COVER (V_C):

$$f(\langle G = (V, E), k \rangle) = \langle G' = (V \cup \{u'\}, E), |V| + 1, k \rangle, \quad u' \notin V.$$

Clearly, f is computable in polynomial (in fact, linear) time. Let us now show that f is a valid reduction from V_C to CoVC. If $\langle G = (V, E), k \rangle \in$ V_C, then G has a vertex-cover $\hat{V}$ of size k. Since G' has the same edge set as G, $\hat{V}$ is also a vertex-cover for G', hence $f(\langle G, k \rangle) \in$ CoVC. Viceversa, if $\langle G, k \rangle \notin$ V_C, then G does not have a vertex-cover of size k or smaller, hence G' cannot have a vertex cover of size k. Since G' is surely not a clique of $|V| + 1$ nodes (as node u' is isolated), we have that $\langle G', |V| + 1, k \rangle = f(\langle G, k \rangle) \notin$ CoVC. We have proved that f is a valid polynomial-time reduction from V_C to CoVC, hence the latter problem is NP-Hard.

Reductions from CLIQUE rather than VERTEX_COVER are also possible but slightly more complex. Indeed, let

$$f'(\langle G = (V, E), h \rangle) = \langle G' = (V \cup \{u', u'', v', v''\}, E \cup \{ \{u', u''\}, \{v', v''\} \}), h, 1 \rangle,$$

with nodes $u', u'', v', v'' \notin V$. Again, f' is computable in polynomial (in fact, linear) time. Since we added two new edges between two pairs of new nodes, G' cannot have a vertex-cover of size 1, hence $\langle G, h \rangle \in$ CLIQUE $\Leftrightarrow \langle G', h, 1 \rangle \in$ CoVC. □

Esercizio 16 [6 punti] Si supponga che Alice voglia spedire un messaggio segreto M a Bob. Purtroppo, Alice non può comunicare direttamente con Bob, ma solo con un terzo agente Charlie, che invece può comunicare con Bob. Si supponga infine che Alice non si fidi di Charlie e che ogni agente conosca la chiave pubblica degli altri agenti. Si descriva la struttura dei messaggi cifrati passati da Alice a Charlie e da Charlie a Bob che garantiscano, alla fine degli scambi, che Charlie non conosca il messaggio M e che Bob sia in grado di stabilire se il messaggio originario di Alice sia stato manomesso.

Answer: Observe that Charlie must simply relay the message from Alice to Bob, therefore Charlie plays the same role as an insecure channel of communication. This means that there is no need to use Charlie's keys in the protocol: Alice simply authenticates the message by appending her digital signature to it and then encrypts the resulting message using Bob's public key. More specifically, Alice first produces the new message $M' = (M, S_A(M))$ and then sends $M'' = P_B(M')$ to Charlie. In case Charlie is honest, he simply forwards the message to Bob. When Bob receives a message $\hat{M}$ from Charlie (which may or may not be equal to M'', depending on Charlie's behaviour) he first applies his secret key obtaining $S_B(\hat{M})$, interprets the latter as a pair (M_1, M_2) and then checks whether $M_1 = P_A(M_2)$. In case the check fails, Bob knows that the

message has been forged, or corrupted. Otherwise, Bob knows that $M_1 = M$ has been surely produced by Alice. Note that there is no way for Charlie to fool Bob by sending him a message different from the one sent by Alice, since, in order to do so, he should know Alice's secret key. Also, Charlie cannot learn M, since that would require knowing Bob's secret key. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(24/6/2002)

Durata: 2h 45m

Esercizio 17 [6 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro $n = 2^{2^i}$:

$$T(n) = \begin{cases} \sqrt{n}T\left(\sqrt{n}\right) + 2n\log n, & n > 2, \\ 0 & n = 2. \end{cases}$$

Answer: Use the general formula taking into account that $s(n) = \sqrt{n}$, $f^i(n) = n^{2^{-i}}$, $f^*(n,2) = \log\log n - 1$, and $w(n) = 2n\log n$. We have that:

$$\begin{aligned} \left[\prod_{j=0}^{\ell-1} s(f^j(n))\right] w(f^\ell(n)) &= \left[\prod_{j=0}^{\ell-1} n^{2^{-(j+1)}}\right] 2n^{2^{-\ell}} \log n^{2^{-\ell}} \\ &= n^{\sum_{j=1}^{\ell} 2^{-j}} n^{2^{-\ell}} 2^{-\ell+1} \log n \\ &= n^{1-2^{-\ell}} n^{2^{-\ell}} 2^{-\ell+1} \log n \\ &= 2^{-\ell+1} n \log n. \end{aligned}$$

Hence

$$T(n) = n\log n \sum_{\ell=0}^{\log\log n - 1} 2^{-\ell+1} = 4n(\log n - 1).$$

□

Esercizio 18 [7 punti] Sia n una potenza di due. Si fornisca lo pseudocodice e si analizzi un algoritmo che, dati in ingresso i vettori complessi $\boldsymbol{b}, \boldsymbol{c} \in \mathbf{C}^n$, fornisca in uscita il numero di soluzioni distinte in $\mathbf{C}^n$ all'equazione vettoriale

$$\boldsymbol{x} \circledast \boldsymbol{x} + \boldsymbol{b} \circledast \boldsymbol{x} + \boldsymbol{c} = 0,$$

dove $\boldsymbol{x}$ è il vettore delle n incognite complesse e $\circledast$ denota l'operatore di convoluzione ciclica. L'algorimo deve avere complessità $O(n\log n)$.

Answer:

Let $\boldsymbol{X} = DFT_n(\boldsymbol{x})$, $\boldsymbol{B} = DFT_n(\boldsymbol{b})$ and $\boldsymbol{C} = DFT_n(\boldsymbol{c})$. By applying the cyclic convolution theorem we can easily argue that the number of distinct solutions of the given vectorial equation is the same as the number of solutions of the following system, made of disjoint, scalar degree-two equations:

$$\forall i, 0 \le i < n : X_i^2 + B_i X_i + C_i = 0.$$

There are two possibile complex values for X_i whenever $\Delta_i = B_i^2 - 4C_i \neq 0$, and only one otherwise. Therefore, letting $s = |\{i : \Delta_i \neq 0, 0 \le i < n\}|$, we have that the number of distinct solutions to the original vectorial equation is 2^s. The pseudocode is the following:

```
NUMSOL(b, c)
n ← length(b)
B ← FFT(b)
C ← FFT(c)
s ← 0
for i ← 0 to n − 1 do
    Δ ← B_i^2 − 4C_i
    if Δ ≠ 0 then s ← s + 1
return 2^s
```

The correctness of the above algorithm follows from our previous argument. Its running time $T(n)$ is clearly dominated by the two FFT calls, hence $T(n) = \Theta(n \log n)$ □

Esercizio 19 [6 punti] Date due stringhe di interi distinti Z^1 e Z^2 con $|Z^1| = |Z^2| = k$, di definisca la loro *discrepanza* $d(Z^1, Z^2) = \sum_{i=1}^{k} |Z_i^1 - Z_i^2|$. Si progetti e si analizzi un algoritmo di programmazione dinamica che, date in ingresso due stringhe X e Y, calcoli la massima discrepanza realizzata da una sottosequenza di X e una sottosequenza di Y di egual lunghezza. La complessità dell'algoritmo deve essere $\Theta(|X||Y|)$.

Answer: Let us choose as our subproblem space the cartesian product of all prefixes of strings X and Y, that is, the set $\{(X_i, Y_j) : 0 \le i \le |X|, 0 \le j \le |Y|\}$. Our dynamic programming algorithm will be based on the following optimal substructure property. Clearly, if either $i = 0$ or $j = 0$ then $d(X_i, Y_j) = 0$. Otherwise, let $Z^1 = \langle z_1^1, z_2^1, \ldots, z_k^1 \rangle$ and $Z^2 = \langle z_1^2, z_2^2, \ldots, z_k^2 \rangle$ be the two subsequences of X_i and Y_j, respectively, which achieve maximum discrepancy. We can have three cases. If $z_k^1 = x_i$ and $z_k^2 = y_j$, then Z_{k-1}^1 and Z_{k-1}^2 must be subsequences of X_{i-1} and Y_{j-1}, respectively, achieving maximum discrepancy or we could obtain a larger discrepancy for X_i and Y_j than the one realized by Z. Using a similar argument, it is easy to see that if $z_k^1 \neq x_i$, then Z^1 is a subsequence of X_{i-1} and it must achieve maximum discrepancy with Y_j. Analogously, if $z_k^2 \neq y_j$, then Z^2 is a subsequence of Y_{j-1} realizing maximum discrepancy with X_i.

Let $D(i, j) = d(X_i, Y_j)$. The above substructure property immediately yields the following recurrence for $D(i, j)$:

$$D(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{|x_i - y_j| + D(i-1, j-1), D(i-1, j), D(i, j-1)\} & \text{otherwise.} \end{cases}$$

From the recurrence, we immediately obtain the desired algorithm.

```
DISCREPANCY(X, Y)
m ← length(X)
n ← length(Y)
for i ← 0 to m do
    D[i, 0] ← 0
for j ← 0 to n do
    D[0, j] ← 0
for i ← 1 to m do
    for j ← 1 to n do
        D[i, j] ← MAX(|x_i − y_j| + D[i − 1, j − 1], D[i − i, j], D[i, j − 1])
return D[m, n]
```

The correctness of the above algorithm easily follows from the correctness of the recurrence and from the fact that the row-major scan computes every table entry prior to any of its uses. The running time of the algorithm is clearly $\Theta(mn) = \Theta(|X||Y|)$. □

Esercizio 20 [7 punti] Si consideri il seguente problema decisionale:

VERTEX COVER or INDEPENDENT SET (VCoIS):

ISTANZA: $\langle G = (V, E), h, k \rangle$, G grafo non orientato, $1 \leq h, k \leq |V|$.

DOMANDA: G ha un vertex cover di taglia h *oppure* un independent set di taglia k?

Si dimostri che VCoIS è NP-Hard.

Answer: Consider the following reduction from VERTEX_COVER (V_C):

$$f(\langle G = (V, E), h \rangle) = \langle G' = (V \cup \{u', u''\}, E \cup \{ \{u', u''\} \}), h + 1, |V| + 2 \rangle, \quad u', u'' \notin V.$$

Clearly, f is computable in polynomial (in fact, linear) time. Let us now show that f is a valid reduction from VC to CoVC. If $\langle G = (V, E), h \rangle \in$ VC, then G has a vertex-cover $\hat{V}$ of size h. Since G' has only an extra edge that can be covered with, say, u', then $\hat{V} \cup \{u'\}$ is a vertex-cover of size $h + 1$ for G', hence $f(\langle G, h \rangle) \in$ CoVC. Viceversa, if $\langle G, h \rangle \notin$ VC, then all vertex covers of G are of size $h + 1$ or larger, hence G' can only have vertex covers of size $h + 2$ or larger (since edge $\{u', u''\}$ cannot be covered by nodes in V). Moreover, since G' has at least one edge, it cannot have an IS of $|V| + 2$ nodes (only a graph with an empty edgeset has such a large IS). It follows that $\langle G', h + 1, |V| + 2 \rangle = f(\langle G, h \rangle) \notin$ VCoIS. We have proved that f is a valid polynomial-time reduction from VC to CoVC, hence the latter problem is NP-Hard. □

Esercizio 21 [6 punti] Si consideri un criptosistema a chiave pubblica di tipo RSA in cui il dominio dei messaggi è $\boldsymbol{Z}_n$, con $n = 7 \times 29 = 203$. Data la chiave pubblica $P = (101, 203)$ determinare la corrispondente chiave segreta S e la relativa funzione di decodifica $S(M)$ ad essa associata.

Answer: We have that $\phi(n) = (7 - 1) \times (29 - 1) = 168$. By running the Extended Euclid Algorithm with inputs (101,168), we find the following Bezout's identity:

$$1 = 168 \times (-3) + 101 \times 5$$

whence 5 is the multiplicative inverse of 101 in $\boldsymbol{Z}^{\star}_{\Phi(n)}$. It follows that the secret key associated to $P = (101, 203)$ is $S = (5, 203)$, and the decryption function is

$$S(M) = M^5 \bmod 203$$

□

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(8/7/2002)

Durata: 2h 45m

Esercizio 22 [7 punti] Si consideri la seguente equazione di ricorrenza, definita per valori arbitrari parametro $n > 0$:

$$T(n) = \begin{cases} \lfloor\sqrt{n}\rfloor T(\lfloor\sqrt{n}\rfloor) + n^2, & n > 2, \\ 1 & n = 1, 2. \end{cases}$$

Si dimostri, utilizzando l'induzione parametrica, che $T(n) \in O(n^2)$.

Answer:

We postulate that there exists a positive constant c for which $T(n) \leq cn^2$, for any value of $n > 0$, and then proceed to find an appropriate value for c using parametric induction. Consider the base cases first. It must be $T(1) = 1 \leq c \cdot 1^2$ and $T(2) = 1 \leq c \cdot 2^2$, whence $c \geq \max\{1, 1/4\} = 1$. For $n \geq 3$, assume that $T(k) \leq ck^2$ for $k < n$. When $k = n$ we obtain:

$$\begin{aligned} T(n) &\leq \lfloor\sqrt{n}\rfloor T(\lfloor\sqrt{n}\rfloor) + n^2 \\ &\leq cn\sqrt{n} + n^2 \end{aligned}$$

It then suffices to choose a constant c such that for all $n \geq 3$,

$$\begin{aligned} cn\sqrt{n} + n^2 &\leq cn^2 \Leftrightarrow \\ c + \sqrt{n} &\leq c\sqrt{n} \Leftrightarrow \\ c &\geq \sqrt{n}/(\sqrt{n} - 1) \end{aligned}$$

Note that $f(n) = \sqrt{n}/(\sqrt{n}-1)$ is a strictly decreasing function over its domain, hence it suffices to choose $c \geq f(3) = \sqrt{3}/(\sqrt{3} - 1) > 1$. Putting it all together, we have proved that

$$T(n) \leq \frac{\sqrt{3}}{\sqrt{3} - 1} n^2.$$

□

Esercizio 23 [7 punti] Sia n una potenza di quattro. Data una matrice complessa A di dimensioni $n \times n$, si consideri la sua decomposizione in n *blocchi* $A_{i,j}$, per $1 \leq i, j \leq \sqrt{n}$, ognuno di dimensioni $\sqrt{n} \times \sqrt{n}$. La matrice A si dice *circolante a $\sqrt{n}$-blocchi* se tutti i blocchi $A_{i,j}$ sono matrici circolanti.

Progettare e analizzare un algoritmo che, date in ingresso due matrici circolanti a $\sqrt{n}$-blocchi, restituisca il loro prodotto in tempo $O(n^{2.5})$. Nell'analisi di complessità, si considerino di costo unitario e non nullo solo le operazioni aritmetiche tra scalari complessi. (*Suggerimento:* Si consideri un algoritmo ibrido ...)

Answer: We want to exploit the fact that the product of two circulant matrices of size $k \times k$, where k is a power of two, can be obtained through the cyclic convolution of their first columns in $\Theta(k \log k)$ arithmetic operations between scalars as follows:

CIRCMULT(A, B)
$k \leftarrow \text{rows}(A)$
$\boldsymbol{a}^0 \leftarrow (A[0,0], \ldots, A[k-1,0])$
$\boldsymbol{b}^0 \leftarrow (B[0,0], \ldots, B[k-1,0])$
$\boldsymbol{c}^0 \leftarrow \boldsymbol{a}^0 \circledast \boldsymbol{b}^0$
for $i \leftarrow 0$ **to** $k-1$ **do**
  for $j \leftarrow 0$ **to** $k-1$ **do**
    $C[i,j] \leftarrow c^0_{(i-j) \bmod n}$
return C

(Note that the two nested **for** loops only contain assignments between complex scalars, hence their complexity is null in terms of arithmetic operations.)

In order to exploit CIRCMULT, we consider the basic divide-and-conquer algorithm for matrix multiplication, stopping the recursion at $n_0 = \sqrt{n}$, where CIRCMULT is invoked.

BLOCKMULT(A, B, n_0)
$k \leftarrow \text{rows}(A)$
if $k = n_0$ **then return** CIRCMULT(A, B)
$\star\star$ for $1 \le i, j \le 2$ consider the 4 $k/2 \times k/2$ blocks
$A_{i,j}$ of A, $B_{i,j}$ of B, and $C_{i,j}$ of $C = AB$ $\star\star$
for $i \leftarrow 1$ **to** 2 **do**
  for $j \leftarrow 1$ **to** 2 **do**
    $C_{i,j} \leftarrow$ MAT_SUM(BLOCKMULT$(A_{i,1}, B_{1,j}, n_0)$, BLOCKMULT$(A_{i,2}, B_{2,j}, n_0)$)
return C

Clearly, the initial call will be BLOCKMULT$(A, B, \sqrt{\text{rows}(A)})$. The running time of the above algorithm is upper bounded by the following recurrence

$$T(n) = \begin{cases} 8T(n/2) + n^2, & n > n_0, \\ cn_0 \log n_0 & n = n_0, \end{cases}$$

where c is a suitable constant. By applying the general formula we obtain

$$\begin{aligned} T(n) &= \sum_{j=0}^{\log(n/n_0)-1} 8^j (n/2^j)^2 + 8^{\log(n/n_0)} cn_0 \log n_0 \\ &= n^2 \sum_{j=0}^{\log(n/n_0)-1} 2^j + c(n/n_0)^3 n_0 \log n_0 \\ &= \Theta(n^3/n_0) + \Theta(n^3 \log(n_0)/n_0^2) \\ &= \Theta(n^3/n_0). \end{aligned}$$

By substituting $n_0 = \sqrt{n}$, we obtain $T(n) = \Theta(n^{2.5})$, as required. □

Esercizio 24 [6 punti] Sia $n > 0$. Si considerino le seguenti due *ricorrenze incrociate*, definite, cioè, l'una in funzione dell'altra per tutti i valori di i e j con $1 \le i, j \le n$:

$$C(i,j) = \begin{cases} 1 & (i = 1) \textbf{ or } (j = n), \\ C(i-1,j) + C(i,j+1) + D(i,j) & \text{altrimenti.} \end{cases}$$

e

$$D(i,j) = \begin{cases} 0 & (i = 1) \textbf{ or } (j = n), \\ D(i-1,j) + C(i-1,j) & \text{altrimenti.} \end{cases}$$

Si fornisca un algoritmo iterativo *bottom-up* che restituisca i valori $C(i,j)$ e $D(i,j)$, $1 \leq i,j \leq n$ in tempo $\Theta(n^2)$. Si analizzi la correttezza e la complessità dell'algoritmo.

Answer: We will store the values $C(i,j)$ and $D(i,j)$, for $1 \leq i,j \leq n$, in two bidimensional arrays C and D. We have to determine an order in computing the table entries so that all entries are computed prior to their subsequent use. By considering the two recurrences, it is clear that a column-major scan of the two tables, starting from column $n-1$ down to column 1 will correctly compute all the table entries, as long as we are careful to compute $D[i,j]$ prior to computing $C[i,j]$ (since $C[i,j]$ makes use of $D[i,j]$). The code immediately follows.

```
COMPUTE_CD(n)
for i ← 1 to n do
    C[i,n] ← C[1,i] ← 1
    D[i,n] ← D[1,i] ← 0
for j ← n − 1 downto 1 do
    for i ← 2 to n do
        D[i,j] ← D[i − 1,j] + C[i − 1,j]
        C[i,j] ← C[i − 1,j] + C[i,j + 1] + D[i,j]
return (C,D)
```

Each table entry is computed in constant time, hence the running time of the above algorithm is $\Theta(n^2)$. □

Esercizio 25 [6 punti] Si consideri il seguente problema decisionale:

DISJOINT INDEPENDENT SETS (D-IS):

ISTANZA: $\langle G = (V,E), h, k \rangle$, G grafo non orientato, $1 \leq h,k \leq |V|$.

DOMANDA: G ha due insiemi indipendenti $V', V'' \subseteq V$, $V' \cap V'' = \emptyset$, con $|V'| = h$ e $|V''| = k$?

Si dimostri che NA-IS è NP-Hard.

Answer: We can resort to the following simple reduction from IS

$$f(\langle G = (V,E), k \rangle) = \langle G' = (V \cup \{u\}, E \cup \{\, \{u,v\} : v \in V \,\}), 1, k \rangle, \text{ with } u \notin V$$

Clearly, f can be computed in linear time. If $\langle G = (V,E), k \rangle \in$ IS, there exists an independent set $V' \subseteq V$ of size k in G. Now, V' is also an independent set for G' (the only extra edges in G' connect nodes in V to $u \notin V$), and $\{u\}$ is an independent set of size 1 for G' disjoint from V', therefore $f(\langle G = (V,E), k \rangle) \in$ NA-IS. Conversely, assume that $k > 1$ (the case $k = 1$ is trivial). If $f(\langle G = (V,E), k \rangle) = \langle G', 1, k \rangle \in$ NA-IS then G' contains an independent set V' of size $k > 1$. Note that such set cannot contain u, since this latter node is adjacent to all the other nodes of the graph, hence $V' \subseteq V$. Since G is a subgraph of G', it follows that V' is an independent set for G, hence $\langle G = (V,E), k \rangle \in$ IS. □

Esercizio 26 [6 punti] Si consideri il seguente sistema di equazioni modulari:

$$\begin{cases} x &= 2 \bmod 3 \\ x &= 4 \bmod 5 \end{cases}$$

Utilizzando il teorema cinese dei resti, determinare l'unica soluzione al sistema in $\boldsymbol{Z}_{15}$.

Answer: One could use the analytic definition of the bijection between $\boldsymbol{Z}_{15}$ and $\boldsymbol{Z}_3 \times \boldsymbol{Z}_5$ implied by the Chinese Reminder Theorem. We have $m_1 = 15/3 = 5$, $m_2 = 15/5 = 3$, ${m_1}^{-1} \bmod 3 = 2$, ${m_2}^{-1} \bmod 5 = 2$, $c_1 = 5 \cdot 2 = 10$, $c_2 = 3 \cdot 2 = 6$. Hence the (unique) solution to the above system of equations is $a = (2 \cdot 10 + 4 \cdot 6) \bmod 15 = 44 \bmod 15 = 14$, whose correctness can be easily checked directly. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(9/9/2002)

Durata: 2h 45m

Esercizio 27 [6 punti] Si consideri la seguente equazione di ricorrenza, definita per valori arbitrari del parametro $n > 0$:

$$T(n) = \begin{cases} \lfloor \frac{n}{2} \rfloor T\left(\lfloor \sqrt{n} \rfloor\right) + n^2, & n > 2, \\ 2 & n = 1, 2. \end{cases}$$

Si dimostri, utilizzando l'induzione parametrica, che $T(n) \in O(n^2)$.

Answer: We postulate that there exists a positive constant c for which $T(n) \leq cn^2$, for any value of $n > 0$, and then proceed to find an appropriate value for c using parametric induction. Consider the base cases first. It must be $T(1) = 2 \leq c \cdot 1^2$ and $T(2) = 2 \leq c \cdot 2^2$, whence $c \geq \max\{2, 1/2\} = 2$. For $n \geq 3$, assume that $T(k) \leq ck^2$ for $k < n$. When $k = n$ we obtain:

$$\begin{aligned} T(n) &\leq \left\lfloor \frac{n}{2} \right\rfloor T(\lfloor \sqrt{n} \rfloor) + n^2 \\ &\leq \frac{n}{2} c \left(\lfloor \sqrt{n} \rfloor\right)^2 + n^2 \\ &\leq \frac{cn^2}{2} + n^2 \\ &\leq \left(\frac{c}{2} + 1\right) n^2 \end{aligned}$$

Therefore, it suffices to choose a constant c such that, for all $n \geq 3$,

$$\begin{aligned} \left(\frac{c}{2} + 1\right) n^2 &\leq cn^2 \Leftrightarrow \\ \left(\frac{c}{2} + 1\right) &\leq c \Leftrightarrow \\ c &\geq 2 \end{aligned}$$

Putting it all together, we have proved that for any $n > 0$,

$$T(n) \leq 2n^2.$$

□

Esercizio 28 [7 punti] Sia n una potenza di due. Si fornisca lo pseudocodice e si analizzi un algoritmo che, dati in ingresso i vettori complessi $\boldsymbol{a}$ e $\boldsymbol{b} \in \mathbf{C}^n$, fornisca in uscita una delle soluzioni al seguente sistema di equazioni con incognite $\boldsymbol{x}$ e $\boldsymbol{y} \in \mathbf{C}^n$:

$$\begin{cases} \boldsymbol{x} \circledast \boldsymbol{y} = \boldsymbol{a}, \\ \boldsymbol{x} + \boldsymbol{y} = \boldsymbol{b}, \end{cases}$$

dove $\circledast$ denota l'operatore di convoluzione ciclica. L'algorimo deve avere complessità $O(n \log n)$.

Answer: Let us denote by $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{A}, \boldsymbol{B}$ the Discrete Fourier Transforms (DFTs) of vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{a}, \boldsymbol{b}$, respectively. Then, by multiplying both equations in the system by F_n we obtain the following (equivalent) system:

$$\begin{cases} \boldsymbol{X} \odot \boldsymbol{Y} = \boldsymbol{A}, \\ \boldsymbol{X} + \boldsymbol{Y} = \boldsymbol{B}, \end{cases}$$

where $\odot$ denotes component-wise multiplication. A solution to the above system can be obtained component by component by solving, for $0 \leq i < n$, the following scalar system:

$$\begin{cases} X_i Y_i = A_i, \\ X_i + Y_i = B_i. \end{cases}$$

Let $\Delta_i = \sqrt{{B_i}^2 - 4A_i}$. Simple algebra suffices to show that a possible solution is $X_i = (B_i + \Delta_i)/2$ and $Y_i = (B_i - \Delta_i)/2$. A solution to the original system can then be obtained by taking the inverse transform of $\boldsymbol{X}$ and $\boldsymbol{Y}$. The algorithm immediately follows.

SOLVE_SYSTEM($\boldsymbol{a}, \boldsymbol{b}$)
$n \leftarrow$ length($\boldsymbol{a}$)
$\boldsymbol{A} \leftarrow$ FFT($\boldsymbol{a}$)
$\boldsymbol{B} \leftarrow$ FFT($\boldsymbol{b}$)
for $i \leftarrow 0$ **to** $n - 1$ **do**
    DELTA $\leftarrow \sqrt{{B_i}^2 - 4A_i}$
    $X_i \leftarrow (B_i +$ DELTA$)/2$
    $Y_i \leftarrow (B_i -$ DELTA$)/2$
$\boldsymbol{x} \leftarrow$ INV_FFT($\boldsymbol{X}$)
$\boldsymbol{y} \leftarrow$ INV_FFT($\boldsymbol{Y}$)
return $(\boldsymbol{x}, \boldsymbol{y})$

The complexity of the above algorithm is dominated by the four calls to the FFT routine, hence its running time is $\Theta(n \log n)$. □

Esercizio 29 [7 punti] Date due stringhe di interi (positivi, negativi o nulli) Z^1 e Z^2, con $|Z^1| = |Z^2| = \ell$, si definisca il loro *prodotto scalare* $\langle Z^1, Z_2 \rangle$ come $\sum_{i=1}^{\ell} Z_i^1 \cdot Z_i^2$, dove $\cdot$ denota il prodotto tra interi. Si progetti e si analizzi un algoritmo di programmazione dinamica che, date in ingresso due stringhe arbitrarie X e Y, calcoli il massimo prodotto scalare realizzabile da una sottosequenza di X e una sottosequenza di Y di egual lunghezza. La complessità dell'algoritmo deve essere $O(|X||Y|)$.

Answer: Let us choose as our subproblem space the cartesian product of all prefixes of strings X and Y, that is, the set $\{(X_i, Y_j) : 0 \leq i \leq |X|, 0 \leq j \leq |Y|\}$, and let us denote by $M(i, j)$ the maximum scalar product obtanaible by a pair of subsequences of X_i and Y_j of equal length. Our dynamic programming algorithm will be based on the following optimal substructure property. Clearly, if either $i = 0$ or $j = 0$ then $M(i, j) = 0$. Otherwise, let $Z^1 = \langle z_1^1, z_2^1, \ldots, z_k^1 \rangle$ and $Z^2 = \langle z_1^2, z_2^2, \ldots, z_k^2 \rangle$ be the two

subsequences of X_i and Y_j, respectively, which achieve maximum scalar product. We can have three cases. If $z_k^1 = x_i$ and $z_k^2 = y_j$, then Z_{k-1}^1 and Z_{k-1}^2 must be subsequences of X_{i-1} and Y_{j-1}, respectively, achieving maximum scalar product or we could obtain a larger scalar product for X_i and Y_j than the one realized by Z^1 and Z^2. Using a similar argument, it is easy to see that if $z_k^1 \neq x_i$, then Z^1 is a subsequence of X_{i-1} achieving maximum scalar product with a subsequence of Y_j. Analogously, if $z_k^2 \neq y_j$, then Z^2 is a subsequence of Y_{j-1} realizing maximum scalar product with a subsequence of X_i.

The above substructure property immediately yields the following recurrence for $M(i,j)$:

$$M(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{x_i \cdot y_j + M(i-1,j-1), M(i-1,j), M(i,j-1)\} & \text{otherwise.} \end{cases}$$

From the recurrence, we immediately obtain the desired algorithm.

```
MAX_SCALAR_PRODUCT(X, Y)
m ← length(X)
n ← length(Y)
for i ← 0 to m do
    M[i, 0] ← 0
for j ← 0 to n do
    M[0, j] ← 0
for i ← 1 to m do
    for j ← 1 to n do
        M[i, j] ← MAX(x_i · y_j + M[i − 1, j − 1], M[i − i, j], M[i, j − 1])
return M[m, n]
```

The correctness of the above algorithm easily follows from the correctness of the recurrence and from the fact that the row-major scan computes every table entry prior to any of its uses. The running time of the algorithm is clearly $\Theta(mn) = \Theta(|X||Y|)$. □

Esercizio 30 [7 punti] Siano $L, L' \subseteq \{0,1\}^\star$. Si dimostri **rigorosamente** la seguente proposizione:

$$(L \in \text{NP}) \textbf{ and } (L' <_\text{P} L) \Rightarrow L' \in \text{NP}.$$

Answer: We will prove that there exists a polynomial-time verification algorithm for L', hence $L' \in$ NP. From the hypothesis, we know the following two facts:

1. ($L \in$ NP) There exists a polynomial-time verification algorithm $V_L(x,y)$ for L, that is, an algorithm running in time $O\left((|x| + |y|)^{c_1}\right)$ such that L is the language verified by V_L and if $x \in L$ then there exists a string $y \in \{0,1\}^\star$, with $|y| = O(|x|^{c_2})$, such that $V_L(x,y) = 1$.

2. ($L' <_\text{P} L$) There exists a function $f(x)$ computable in time $O(|x|^{c_3})$ such that $(x \in L') \Leftrightarrow (f(x) \in L)$.

We construct a verification algorithm for L' as follows

```
V_L'(x, y)
    return V_L(f(x), y)
```

Let us prove that $V_{L'}$ is indeed a poly-time verification algorithm for L'. First, observe that $|f(x)| \in O(|x|^{c_3})$, hence the running time of $V_{L'}(x,y)$ is $O\left((|x|^{c_3}+|y|)^{c_1}\right)$, which is still polynomial in $(|x|+|y|)$. Moreover, $x \in L'$ if and only if $f(x) \in L$, hence $V_{L'}(x,y) = 1$ if and only if $x \in L'$. Finally, if $f(x) \in L$, then there exists a certificate y for $f(x)$ of size $O(|f(x)|^{c_2}) = O(|x|^{c_2 c_3})$ (which is polynomial in $|x|$) for which $V_L(f(x),y) = 1$. This completes the proof that $V_{L'}$ is indeed a verification algorithm for L'. □

Esercizio 31 [6 punti] Si consideri il seguente sistema di equazioni modulari:

$$\begin{cases} x &= 2 \bmod 3 \\ x &= 4 \bmod 11 \end{cases}$$

Utilizzando il teorema cinese dei resti, determinare l'unica soluzione al sistema in $\boldsymbol{Z}_{33}$.

Answer: One can use the analytic definition of the bijection between $\boldsymbol{Z}_{33}$ and $\boldsymbol{Z}_3 \times \boldsymbol{Z}_{11}$ implied by the Chinese Reminder Theorem. We have $m_1 = 33/3 = 11$, $m_2 = 33/11 = 3$, ${m_1}^{-1} \bmod 3 = 2$, ${m_2}^{-1} \bmod 11 = 4$, $c_1 = 11 \cdot 2 = 22$, $c_2 = 3 \cdot 4 = 12$. Hence the (unique) solution to the above system of equations is $a = (2 \cdot 22 + 4 \cdot 12) \bmod 33 = 92 \bmod 33 = 26$, whose correctness can be easily checked directly. □

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(23/9/2002)

Durata: 2h 45m

Esercizio 32 [6 punti] Si determini la soluzione della seguente relazione di ricorrenza, definita per valori del parametro $n = 4^i$:

$$T(n) = \begin{cases} 0 & n = 1, \\ 16T\left(\frac{n}{4}\right) + 2n^2 \log_4 n & n > 1. \end{cases}$$

Answer: We use the general formula taking into account that $n_0 = 1$, $T_0 = 0$, $s(n) = 16$, $f^i(n) = n/4^i$, $f^*(n, n_0) = \log_4 n - 1$, and $w(n) = 2n^2 \log_4 n$. Since $T_0 = 0$, the second term of the general form evaluates to 0. Therefore we have:

$$\begin{aligned} T(n) &= \sum_{\ell=0}^{\log_4 n - 1} \left[\prod_{j=0}^{\ell-1} s(f^j(n))\right] w(f^\ell(n)) \\ &= \sum_{\ell=0}^{\log_4 n - 1} 16^\ell 2 \left(n/4^\ell\right)^2 \log_4(n/4^\ell) \\ &= 2 \sum_{\ell=0}^{\log_4 n - 1} n^2(\log_4 n - \ell) \\ &= 2n^2 \log_4^2 n - 2n \sum_{\ell=0}^{\log_4 n - 1} \ell \\ &= n^2 \left(2 \log_4^2 n - \log_4 n(\log_4 n - 1)\right) \\ &= n^2 \log_4 n(\log_4 n + 1) \end{aligned}$$

□

Esercizio 33 [7 punti] Dato in ingresso un grafo non orientato e connesso $G = (V, E)$ rappresentato con liste di adiacenza, si dia lo pseudocodice e si analizzi un algoritmo che restituisca in uscita un sottoinsieme V' di V di cardinalità $\lceil |V|/2 \rceil$ tale che il sottografo G' di G indotto da V', ($G' = (V', E')$, con $E' = \{\{u, v\} \in E : u, v \in V'\}$) sia ancora connesso. La complessità in tempo dell'algoritmo deve essere lineare nella taglia del grafo di ingresso. (*Suggerimento:* Si modifichi opportunamente il codice della BFS).

Answer: We can simply run a simplified version of BFS (using only two colors and without computing distances and predecessor vector) starting from an arbitrary node (v_1, for simplicity). The algorithm keeps count of the number of nodes enqueued so far and exits after it has enqueued exactly $\lceil |V|/2 \rceil$ nodes, returning these nodes. The correctness of the above approach follows from the fact that during a BFS, when a node is enqueued there is a path from that node to the source made of all nodes that have been previously enqueued. As a consequence, any two such nodes are reachable from each other by composing these two paths, which implies that the subgraph induced by the nodes that have been enqueued is connected. In the code below, variable NUM_EN counts the number of nodes enqueued so far, while set V' stores the nodes to be returned.

```
SUBCOMPONENT(G)
Q ← ∅
for each v ∈ V − {v_1} do
    color[v] ← white
color[v_1] ← black
ENQUEUE(Q, v_1)
NUM_EN ← 1
V' ← {v_1}
while Q ≠ ∅ do
    u ← DEQUEUE(Q)
    for each v ∈ Adj[u] do
        if color[v] = white then
            color[v] ← black
            if NUM_EN < ⌈|V|/2⌉
                then
                        ENQUEUE(Q, v)
                        V' ← V' ∪ {v}
                        NUM_EN ← NUM_EN + 1
                else return V'
return V' {needed for the case |V| = 1 }
```

The running time of SUBCOMPONENT(G) is dominated by the running time of a full BFS on G, hence it is linear in the size of G, as required. □

Esercizio 34 [7 punti] Dato il seguente codice di programmazione dinamica

```
DP_SUM(n)
for i ← 1 to n do A[i, i] ← i
for ℓ ← 2 to n do
    for i ← 1 to n − ℓ + 1 do
        j ← i + ℓ − 1
        A[i, j] ← A[i, j − 1] + A[i + 1, j]
return A[1, n]
```

si fornisca un codice memoizzato equivalente e se ne analizzi la complessità.

Answer: The given code computes the following recurrence:

$$A(i,j) = \begin{cases} i & 1 \le i = j \le n \\ A(i,j-1) + A(i+1,j) & 1 \le i < j \le n \end{cases}$$

The memoized code to compute this recurrence is given by the following pair of routines:

```
INIT_A(n)
for i ← 1 to n do
    A[i, i] ← i
    for j ← i + 1 to n do
        A[i, j] ← 0 { note that A(i, j) is always positive}
return REC_A(1, n)

REC_A(i, j)
if A[i, j] = 0
    then A[i, j] ← REC_A(i, j − 1) + REC_A(i + 1, j)
return A[i, j]
```

To obtain $A(1, n)$, we call INIT_A(n). Note that each entry is computed exactly once for every pair (i, j), with $1 \leq i \leq j \leq n$, hence INIT_A(n) triggers $\Theta(n^2)$ recursive calls to REC_A. Since each call performs constant work, the running time of the memoized code is $\Theta(n^2)$, which is also the running time of the corresponding dynamic programming code. □

Esercizio 35 [7 punti] Sia dato il seguente problema decisionale:

CNF FORMULA DISJOINTNESS (CFD):

ISTANZA: $\langle \Phi(\boldsymbol{x}), \Psi(\boldsymbol{x}) \rangle$, Φ, Ψ formule booleane in CNF di variabili $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$.

DOMANDA: Esiste un assegnamento di valori di verità alle n variabili su cui Φ e Ψ differiscono?

Si dimostri che CFD è NP-hard.

Answer: We show that 3-CNF-SAT $<_P$ CFD, hence proving that the latter problem is NP-hard. Let $\Phi(\boldsymbol{x})$ be a 3-CNF formula. We define the following reduction:

$$f(\langle \Phi(\boldsymbol{x}) \rangle) = \langle \Phi(\boldsymbol{x}), \Xi(\boldsymbol{x}) = x_1 \wedge \neg x_1 \rangle.$$

Observe that $\Xi(\boldsymbol{x}) = x_1 \wedge \neg x_1$ can be seen as a CNF formula made of two clauses: $\{x_1\}$ and $\{\neg x_1\}$. Moreover, $\Xi(\boldsymbol{x})$ is a contradiction, yielding **false** for all possible truth assignments.

Clearly, f can be computed in polynomial (in fact, linear) time. Let us now prove that f is a valid reduction. If $\langle \Phi(\boldsymbol{x}) \rangle \in$ 3-CNF-SAT then there is a truth assignment $\boldsymbol{b}$ such that $\Phi(\boldsymbol{b}) =$ **true**. Under such an assignment Φ and Ξ differ, hence $f(\langle \Phi(\boldsymbol{x}) \rangle) \in$ CFD. Conversely, if $\langle \Phi(\boldsymbol{x}) \rangle \notin$ 3-CNF-SAT, then $\Phi(\boldsymbol{x}) = \Xi(\boldsymbol{x}) =$ **false** for all truth assignments, hence $f(\langle \Phi(\boldsymbol{x}) \rangle) \notin$ CFD. □

Esercizio 36 [6 punti] Sia $n = \prod_{i=1}^{k} n_i$ con $n_i > 1$, $1 \leq i \leq k$ e $\gcd(n_i, n_j) = 1$, $1 \leq i < j \leq k$. Si fornisca la descrizione analitica della corrispondenza diretta e di quella inversa tra Z_n e $Z_{n_1} \times Z_{n_2} \times \cdots \times Z_{n_k}$ indotta dal teorema cinese dei resti.

Answer: The direct correspondence is the following:

$$\forall x \in Z_n : \; f(x) = (x \bmod n_1, x \bmod n_2, \ldots, x \bmod n_k).$$

The inverse correspondence can be obtained as follows. Let $m_i = n/n_i$, and $c_i = m_i({m_i}^{-1} \bmod n_i)$. Then

$$\forall \boldsymbol{x} = (x_1, x_2, \ldots x_k) \in Z_{n_1} \times Z_{n_2} \times \cdots \times Z_{n_k} : \; f^{-1}(\boldsymbol{x}) = \left(\sum_{i=1}^{k} x_i c_i \right) \bmod n.$$

□

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(20/3/2003)

Durata: 2h 30m

Esercizio 1 [8 punti] Si risolva e si verifichi la seguente equazione di ricorrenza per valori del parametro $n = 2^{2^i}$:

$$T(n) = \begin{cases} 16T\left(\sqrt{n}\right) + \log n, & n > 2, \\ 1 & n = 2. \end{cases}$$

Answer: Use the general formula taking into account that $s(n) = 16 = 2^4$, $f^i(n) = n^{1/2^i}$, $f^*(n, 2) = \log\log n - 1$, and $w(n) = \log n$. We have that:

$$\prod_{j=0}^{\ell-1} s(f^j(n)) = \left[\prod_{j=0}^{\ell-1} 16\right] = 16^\ell = 2^{4\ell},$$

while

$$\begin{aligned} \left[\prod_{j=0}^{\ell-1} s(f^j(n))\right] w(f^\ell(n)) &= 2^{4\ell} \log n^{1/2^\ell} \\ &= \frac{2^{4\ell}}{2^\ell} \log n \\ &= 2^{3\ell} \log n. \end{aligned}$$

Hence

$$\begin{aligned} T(n) &= \log n \sum_{\ell=0}^{\log\log n - 1} 2^{3\ell} + 2^{4\log\log n} \\ &= \log n \left(\frac{2^{3\log\log n} - 1}{7}\right) + \log^4 n \\ &= \frac{8\log^4 n - \log n}{7}, \end{aligned}$$

which can be easily verified using induction (verification omitted for brevity). □

Esercizio 2 [8 punti] Per $n > 0$, si consideri la seguente equazione vettoriale:

$$(\boldsymbol{x} \circledast \boldsymbol{x}) + \boldsymbol{u_n} = \mathbf{0},$$

dove $\boldsymbol{x}$ è il vettore delle n incognite complesse, $\circledast$ denota l'operatore di convoluzione ciclica, $\mathbf{0}$ il vettore nullo e $\boldsymbol{u_n} = (1/n, 1/n, \ldots, 1/n)$ il vettore le cui n componenti sono tutte uguali a $1/n$. Si dimostri che l'equazione ammette solo due soluzioni distinte e si determinino tali soluzioni.

Answer: Let $\boldsymbol{X} = DFT_n(\boldsymbol{x})$ and $\boldsymbol{U_n} = DFT_n(\boldsymbol{u_n})$. Note first that by the summation lemma it follows that

$$[\boldsymbol{U_n}]_i = (1/n) \sum_{j=0}^{n-1} \omega_n^{ij} = \begin{cases} 1/n \cdot n = 1, & i = 0 \\ 0, & 1 \le i \le n-1. \end{cases}$$

By applying the cyclic convolution theorem, we can solve the vector equation by first solving the equation

$$\boldsymbol{X} \odot \boldsymbol{X} = (-1, 0, \ldots, 0)$$

(where $\odot$ denotes component-wise product) and then computing the inverse Fourier transform of each distinct vector solution. The above vector equation is equivalent to the following system, made of decoupled, scalar, degree-two equations:

$$\begin{cases} X_i^2 = -1, & i = 0 \\ X_i^2 = 0, & 1 \leq i \leq n-1. \end{cases}$$

The only two solutions are then $\boldsymbol{X_0} = (i, 0 \ldots, 0)$ and $\boldsymbol{X_1} = (-i, 0 \ldots, 0)$. Finally, we obtain the desired solutions $\boldsymbol{x_0}$ and $\boldsymbol{x_1}$ by anti-transforming $\boldsymbol{X_0}$ and $\boldsymbol{X_1}$. Recalling that $DFT_n^{-1}(\boldsymbol{y}) = (1/n)DFT_n(\boldsymbol{y^{rev}})$, we obtain $\boldsymbol{x_0} = (i/n, i/n, \ldots, i/n)$ and $\boldsymbol{x_1} = (-i/n, -i/n, \ldots, -i/n)$ □

Esercizio 3 [8 punti] Date due stringhe di interi positivi Z^1 e Z^2 con $|Z^1| = |Z^2| = k$, di definisca il loro *pseudorapporto*, $\mathrm{psr}(Z^1, Z^2) = \sum_{i=1}^{k} Z_i^1/Z_i^2$ (si ponga, per convenzione, $\mathrm{psr}(\epsilon, \epsilon) = 0$). Si progetti e si analizzi un algoritmo di programmazione dinamica che, date in ingresso due stringhe X e Y, calcoli il massimo pseudorapporto realizzato da una sottosequenza di X e una sottosequenza di Y di egual lunghezza. La complessità dell'algoritmo deve essere $\Theta(|X||Y|)$.

Answer: Let us choose as our subproblem space the cartesian product of all prefixes of strings X and Y, that is, the set $\{(X_i, Y_j) : 0 \leq i \leq |X|, 0 \leq j \leq |Y|\}$. Our dynamic programming algorithm will be based on the following optimal substructure property. Clearly, if either $i = 0$ or $j = 0$ (i.e., one of the two prefixes is the empty prefix), then the maximum psuedoratio achievable by two equal-length subsequences of X_i and Y_j is $\mathrm{psr}(\epsilon, \epsilon) = 0$. Otherwise, let $Z^1 = \langle z_1^1, z_2^1, \ldots, z_k^1 \rangle$ and $Z^2 = \langle z_1^2, z_2^2, \ldots, z_k^2 \rangle$ be the two subsequences of X_i and Y_j, respectively, which achieve maximum pseudoratio. We have three cases: (1) if $z_k^1 = x_i$ and $z_k^2 = y_j$, then Z_{k-1}^1 and Z_{k-1}^2 must be subsequences of X_{i-1} and Y_{j-1}, respectively, achieving maximum pseudoratio among all equal-length subsequences of X_{i-1} and Y_{j-1}, or we could obtain a larger pseudoratio in X_i and Y_j than the one realized by Z^1 and Z^2. Using a similar argument, it is easy to see that (2) if $z_k^1 \neq x_i$, then Z^1 is a subsequence of X_{i-1} and it must achieve maximum pseudoratio with some subsequence of Y_j. Analogously, (3) if $z_k^2 \neq y_j$, then Z^2 is a subsequence of Y_{j-1} realizing maximum pseudoratio with some subsequence of X_i.

Let $P(i, j)$ be the maximum pseudoratio achievable by equal-length subsequences of X_i and Y_j. The above substructure property immediately yields the following recurrence for $P(i, j)$:

$$P(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{\frac{x_i}{y_j} + P(i-1, j-1), P(i-1, j), P(i, j-1)\} & \text{otherwise.} \end{cases}$$

From the recurrence, we immediately obtain the desired algorithm.

```
PSEUDORATIO(X, Y)
m ← length(X)
n ← length(Y)
for i ← 0 to m do
    D[i, 0] ← 0
for j ← 1 to n do
    D[0, j] ← 0
for i ← 1 to m do
    for j ← 1 to n do
        D[i, j] ← MAX(x_i/y_j + D[i − 1, j − 1], D[i − i, j], D[i, j − 1])
return D[m, n]
```

The correctness of the above algorithm easily follows from the correctness of the recurrence and from the fact that the row-major scan computes every table entry prior to any of its uses. The running time of the algorithm is clearly $\Theta(mn) = \Theta(|X||Y|)$. □

Esercizio 4 [8 punti] Si consideri il seguente problema decisionale:

3-CNF DISTINCT FORMULAE (3-CNF-DF):

ISTANZA: $\langle \Phi(x_1, \ldots, x_n), \Psi(x_1, \ldots, x_n) \rangle$, Φ, Ψ, formule booleane in forma 3-CNF.

DOMANDA: Esiste un assegnamento di valori di verità $\boldsymbol{b}$ alle n variabili tale che $\Phi(\boldsymbol{b}) \neq \Psi(\boldsymbol{b})$?

Si dimostri che 3-CNF-DF è NP-Hard (*Suggerimento:* si pensi a come esprimere la formula sempre falsa in forma 3-CNF...)

Answer: We reduce fron 3-CNF SAT as follows. Consider the following 3-CNF formula:

$$\Psi(z_1, z_2, z_3) = \bigwedge_{y_1 \in \{z_1, \neg z_1\}, y_2 \in \{z_2, \neg z_2\}, y_3 \in \{z_3, \neg z_3\}} (y_1 \vee y_2 \vee y_3).$$

In other words, Ψ is made of eight disjunctive clauses, which are all the possible clauses containing one literal each for z_1, z_2 and z_3. Note that Ψ is always false, since, for each truth assignment (b_1, b_2, b_3) to the three variables (z_1, z_2, z_3), there is exactly one false disjunction (e.g., $(b_1 = 1, b_2 = 0, b_3 = 0)$ falsifies $(\neg z_1 \vee z_2 \vee z_3)$) which in turn falsifies the entire formula[1]. Our reduction from 3-CNF SAT is then

$$f(\langle \Phi(x_1, \ldots, x_n) \rangle) = \langle \Phi'(x_1, \ldots, x_n, z_1, z_2, z_3), \Psi'(x_1, \ldots, x_n, z_1, z_2, z_3) \rangle.$$

with $\Phi'(x_1, \ldots, x_n, z_1, z_2, z_3) = \Phi(x_1, \ldots, x_n)$ and $\Psi'(x_1, \ldots, x_n, z_1, z_2, z_3) = \Psi(z_1, z_2, z_3)$. Clearly, f is computable in polynomial (in fact, linear) time.

We now show that f is a valid reduction.

($\Rightarrow$) If Φ is satisfiable, then there exists a truth assignment $\boldsymbol{b}$ to its n variables such that $\Phi(\boldsymbol{b}) = 1$. Then, under any truth assignment $\boldsymbol{b'}$ obtained by extending $\boldsymbol{b}$ with three arbitrary values, we have $\Phi'(\boldsymbol{b'}) = \Phi(\boldsymbol{b}) = 1$ and $\Psi'(\boldsymbol{b'}) = \Psi'(b'_{n+1}, b'_{n+2}, b'_{n+3}) = 0$, hence the two formulae differ under $\boldsymbol{b'}$.

($\Leftarrow$) If Φ is not satisfiable, it follows that both Φ' and Ψ' are false under all truth assignments to their $n + 3$ variables. Hence they never differ. □

[1] Here, we are complying with the restrictive definition of 3-CNF form, where a clause cannot contain two or more literals of the same variable. If we relax such a restriction, then the simpler formula $\Xi(z_1) = (z_1 \vee z_1 \vee z_1) \wedge (\neg z_1 \vee \neg z_1 \vee \neg z_1)$ would be sufficient to express a contradiction in 3-CNF form.

Attenzione: Risposte immotivate e prive di prova di correttezza non saranno considerate. Inoltre, l'algoritmo da sviluppare nell'Esercizio 3 va descritto utilizzando lo **pseudocodice** usato in classe.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(4/4/2003)

Durata: 2h 30m

Esercizio 1 [8 punti] Si consideri la seguente equazione di ricorrenza definita per valori arbitrari del parametro $n > 0$:

$$T(n) = \begin{cases} T\left(\lfloor \frac{n}{5} + 1 \rfloor\right) + T\left(\lceil \frac{n}{7} \rceil\right) + 3n & n \geq 5, \\ 0 & n < 5. \end{cases}$$

Dimostrare che $T(n) = O(n)$.

Answer: Rather than using parametric induction, let us bound $T(n)$ from above with a function $T'(n)$ obeying a recurrence solvable with the Master Theorem. For $n \geq 5$, we have:

$$\left\lfloor \frac{n}{5} + 1 \right\rfloor \leq \left\lfloor \frac{n}{5} + \frac{n}{5} \right\rfloor = \left\lfloor \frac{2n}{5} \right\rfloor,$$

while

$$\left\lceil \frac{n}{7} \right\rceil \leq \frac{n}{7} + 1 \leq \frac{n}{7} + \frac{n}{5} = \frac{12n}{35}.$$

Since $\lceil \frac{n}{7} \rceil$ is an integer, it follows that

$$\left\lceil \frac{n}{7} \right\rceil \leq \left\lfloor \frac{12n}{35} \right\rfloor.$$

Since $2/5 > 12/35$, we have that $T(n) \leq T'(n)$, where $T'(n)$ obeys the following recurrence:

$$T'(n) = \begin{cases} 2T'\left(\left\lfloor \frac{2n}{5} \right\rfloor\right) + 3n & n \geq 5, \\ 0 & n < 5. \end{cases}$$

The above recurrence is solvable through the Master Theorem. Specifically, since $\log_b a = \log_{5/2} 2 < 1$, we have that for $\epsilon = 1 - \log_{5/2} 2 > 0$: $w(n) = 3n = \Omega(n^{\log_b a + \epsilon})$, hence $T'(n) = O(w(n)) = O(n)$. It follows that $T(n) = O(T'(n)) = O(n)$. □

Esercizio 2 [8 punti] Siano n e k potenze di due, con $1 \leq k \leq n$ e sia z un arbitrario numero complesso. Un vettore complesso $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ a n componenti si dice (k, n, z)*−padded* se $x_i = z$ per $k \leq i \leq n-1$. In altre parole, $\boldsymbol{x}$ ha le prime k componenti arbitrarie e le ultime $n - k$ fissate a z. Si sviluppi e si analizzi un algoritmo che, dato in ingresso un vettore $(k, n, z)-$padded $\boldsymbol{x}$, ritorni $DFT_n(\boldsymbol{x})$ eseguendo $O(n + n \log k)$ operazioni aritmetiche.

Suggerimento: Si studi il passo di *divide* dell'algoritmo ricorsivo FFT e se ne deduca una proprietà ricorsiva dei vettori $(k, n, z)-$padded con caso di base $k = 1$.
Attenzione: una risoluzione basata sull'algoritmo di Cooley-Tukey richiede un'analisi molto più impegnativa.

Answer: The key observations upon which we base our recursive algorithm are the following:

1. When $k = 1$, a $(k, n, z)-$padded vector $\boldsymbol{x}$ has a single arbitrary component x_0, while the remaining $n-1$ components are all z. In this case, letting $\boldsymbol{y} = DFT_n(\boldsymbol{x})$

we have:

$$\begin{aligned} y_0 &= \sum_{j=0}^{n-1} x_j \omega_n^{0\cdot j} \\ &= x_0 + \sum_{j=1}^{n-1} z \\ &= x_0 + (n-1)z, \end{aligned}$$

while, for $1 \leq i \leq n-1$,

$$\begin{aligned} y_i &= \sum_{j=0}^{n-1} x_j \omega_n^{ij} \\ &= x_0 + \sum_{j=1}^{n-1} z\omega_n^{ij} \\ &= x_0 - z + \sum_{j=0}^{n-1} z\omega_n^{ij} \\ &= x_0 - z + 0 = x_0 - z, \end{aligned}$$

by the summation lemma. This will be our base case in the recursive algorithm.

2. When $k > 1$, let $\boldsymbol{x}^{[0]}$ and $\boldsymbol{x}^{[1]}$ be the vectors containing the even and odd components of $\boldsymbol{x}$, respectively. Then, both $\boldsymbol{x}^{[0]}$ and $\boldsymbol{x}^{[1]}$ are clearly $(k/2, n/2, z)$–padded, since their last $(n-k)/2$ components are equal to z. Then, we can apply our algorithm recursively to obtain $\boldsymbol{y}^{[0]} = DFT_{n/2}(\boldsymbol{x}^{[0]})$ and $\boldsymbol{y}^{[1]} = DFT_{n/2}(\boldsymbol{x}^{[1]})$ and then combine the two vectors in the standard way to obtain $\boldsymbol{y} = DFT_n(\boldsymbol{x})$.

The algorithm follows immediately from the above considerations.

PADDED_FFT($\boldsymbol{x}, k$)
$n \leftarrow \text{length}(\boldsymbol{x})$
$z \leftarrow x_{n-1}$
if $k = 1$ **then**
    $y_0 \leftarrow x_0 + (n-1)z$
    for $i \leftarrow 1$ **to** $n-1$ **do** $y_i \leftarrow x_0 - z$
    return $\boldsymbol{y}$
$\boldsymbol{x}^{[0]} \leftarrow (x_0, x_2, \ldots, x_{n-2})$
$\boldsymbol{x}^{[1]} \leftarrow (x_1, x_3, \ldots, x_{n-1})$
$\boldsymbol{y}^{[0]} \leftarrow$ PADDED_FFT($\boldsymbol{x}^{[0]}, k/2$)
$\boldsymbol{y}^{[1]} \leftarrow$ PADDED_FFT($\boldsymbol{x}^{[1]}, k/2$)
$\omega n \leftarrow e^{2\pi i/n}$; $\omega \leftarrow 1$
for $i \leftarrow 0$ **to** $n/2 - 1$ **do**
    $y_i \leftarrow y_i^{[0]} + \omega y_i^{[1]}$
    $y_{i+n/2} \leftarrow y_i^{[0]} - \omega y_i^{[1]}$
    $\omega \leftarrow \omega \cdot \omega n$
return $\boldsymbol{y}$

The correctness of the above algorithm follows from the above considerations and from the correctness of the conquer phase of the FFT algorithm. Its complexity is

described by the following recurrence, parametric in both k and n:

$$T(k,n) = \begin{cases} c_1 n & k = 1, \\ 2T\left(\frac{k}{2}, \frac{n}{2}\right) + c_2 n & 1 < k \leq n, \end{cases}$$

where c_1, c_2 are two given positive constants. By iterating the above recurrence, we obtain, for all values i, $1 \leq i \leq \log k$,

$$T(k,n) = 2^i T\left(\frac{k}{2^i}, \frac{n}{2^i}\right) + c_2 i n,$$

whence $T(k,n) = c_1 k \frac{n}{k} + c_2 n \log k = O(n + n \log k)$. □

Esercizio 3 [8 punti] Sia $n > 0$. Dato un vettore di interi $\boldsymbol{a}$ a n componenti, si consideri la seguente ricorrenza, definita per tutti i valori (i,j), con $1 \leq i \leq j \leq n$:

$$B(i,j) = \begin{cases} a_i & i = j,\ 1 \leq i \leq n, \\ \sum_{k=i}^{j-1} (B(i,k) - B(k+1,j)) & 1 \leq i < j \leq n. \end{cases}$$

Si sviluppi e si analizzi un algoritmo iterativo che, dato in ingresso il vettore $\boldsymbol{a}$, restituisca il valore $B(1,n)$ in tempo $O(n^3)$.

Answer:

In order to organize a bottom-up computation, let $\ell_{i,j} = j - i + 1$ and note that the computation of $B(i,j)$ depends on elements $B(i,k)$ and $B(k+1,j)$, with $i \leq k < j$. For such range of values of k, we have $\ell_{i,k} = k - i + 1 < j - i + 1 = \ell_{i,j}$ and $\ell_{k+1,j} = j - k \leq j - i < j - i + 1 = \ell_{i,j}$. Hence, we can organize the computation of $B(1,n)$ in a bottom-up fashion by computing the table entries by increasing values of ℓ. This corresponds to scanning the upper triangle of the table storing the values $B(i,j)$, for $1 \leq i \leq j \leq n$, by diagonals parallel to the principal one. The algorithm follows.

```
COMPUTE_B(a)
n ← length(a)
for i ← 1 to n do
    B[i,i] ← a_i
for ℓ ← 2 to n do
    for i ← 1 to n − ℓ + 1 do
        j ← i + ℓ − 1
        B[i,j] ← 0
        for k ← i to j − 1 do
            B[i,j] ← B[i,j] + B[i,k] − B[k+1,j]
return B[1,n]
```

The correctness of the above algorithm follows from the fact that it correctly implements a bottom-up computation of $B[1,n]$, as argued above. Its running time is dominated by the overall number of iterations of the three nested loops, which yield a time complexity of

$$\Theta\left(\sum_{\ell=2}^{n} \sum_{i=1}^{n-\ell+1} \sum_{k=i}^{i+\ell-2} 1\right) = \Theta\left(n^3\right).$$

(This is the same analysis of the matrix-chain multiplication algorithm seen in class.) □

Esercizio 4 [8 punti] Si dimostri che 3-CNF SAT $<_P$ INDEPENDENT SET fornendo una esplicita riduzione polinomiale dal primo al secondo problema.

Suggerimento: Si ricordino le due riduzioni viste in classe da 3-CNF-SAT a CLIQUE e da CLIQUE a INDEPENDENT SET.

Answer: Recall that in order to reduce a 3-CNF-SAT instance

$$\langle \Phi(x_1, \ldots, x_n) = \bigwedge_{i=1}^{m} (y_i^1 \vee y_i^2 \vee y_i^3) \rangle \text{ with } y_i^j \in \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$$

to an instance $\langle G^\Phi = (V^\Phi, E^\Phi), k \rangle$ of CLIQUE, we set $k = m$, $V^\Phi = \{y_i^1, y_i^2, y_i^3, 1 \leq i \leq m\}$, (each node corresponds to a literal, for a total of $3m$ nodes), and, finally, $E^\Phi = \{\{y_r^{j_1}, y_s^{j_2}\} : 1 \leq r \neq s \leq m \text{ and } y_r^{j_1} \neq \neg y_s^{j_2}\}$. In turn, when reducing and instance $\langle G = (V, E), k \rangle$ of CLIQUE to an instance $\langle G' = (V', E'), h \rangle$ of INDEPENDENT SET, we set $G' = G^c = (V, E^c)$ and $h = k$. In order to obtain an explicit reduction from 3-CNF-SAT to INDEPENDENT SET, it is then sufficient to compose the two different reductions. In other words, the sought reduction, on input $\langle \Phi \rangle$ returns the instance $\langle G^{\Phi^c}, m \rangle$ of INDEPENDENT SET. The correctness of the above reduction follows from the transitivity property of polynomial reducibility. □

Attenzione: Risposte immotivate e prive di prova di correttezza non saranno considerate. Inoltre, gli algoritmi da sviluppare negli Esercizi 2 e 3 vanno descritti utilizzando lo **pseudocodice** usato in classe.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(26/6/2003)

Durata: 2h 30m

Esercizio 1 [8 punti] Si consideri la seguente equazione di ricorrenza definita per valori arbitrari del parametro $n > 0$:

$$T(n) = \begin{cases} T\left(\left\lfloor \frac{2n}{7} + 1 \right\rfloor\right) + T\left(\lceil \frac{n}{7} \rceil\right) + 2n & n \geq 7, \\ 0 & n < 7. \end{cases}$$

Dimostrare che $T(n) = O(n)$.

Answer: Rather than using parametric induction, let us bound $T(n)$ from above with a function $T'(n)$ obeying a recurrence solvable with the Master Theorem. For $n \geq 7$, we have:

$$\left\lfloor \frac{2n}{7} + 1 \right\rfloor \leq \left\lfloor \frac{2n}{7} + \frac{n}{7} \right\rfloor = \left\lfloor \frac{3n}{7} \right\rfloor,$$

while

$$\left\lceil \frac{n}{7} \right\rceil \leq \frac{n}{7} + 1 \leq \frac{n}{7} + \frac{n}{7} = \frac{2n}{7}.$$

Since $\lceil \frac{n}{7} \rceil$ is an integer, it follows that

$$\left\lceil \frac{n}{7} \right\rceil \leq \left\lfloor \frac{2n}{7} \right\rfloor.$$

Since $3/7 > 2/7$, we have that $T(n) \leq T'(n)$, where $T'(n)$ obeys the following recurrence:

$$T'(n) = \begin{cases} 2T'\left(\left\lfloor \frac{3n}{7} \right\rfloor\right) + 2n & n \geq 7, \\ 0 & n < 7. \end{cases}$$

The above recurrence is solvable through the Master Theorem. Specifically, since $\log_b a = \log_{7/3} 2 < 1$, we have that for $\epsilon = 1 - \log_{7/3} 2 > 0$: $w(n) = 2n = \Omega(n^{\log_b a + \epsilon})$, hence $T'(n) = O(w(n)) = O(n)$. It follows that $T(n) = O(T'(n)) = O(n)$. □

Esercizio 2 [8 punti] Siano n e k potenze di due, con $1 \leq k < n$ e sia z un arbitrario numero complesso. Un vettore complesso $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ a n componenti si dice $(k, n, z) - \textit{interleaved}$ se $x_i = z$ per $i \bmod k \neq 0$, $0 \leq i \leq n-1$. In altre parole, $\boldsymbol{x}$ può avere componenti diverse da z solo in corrispondenza di indici che sono multipli di k. Si fornisca lo pseudocodice e si analizzi un algoritmo che, dato in ingresso un vettore (k, n, z)–interleaved $\boldsymbol{x}$, ritorni $DFT_n(\boldsymbol{x})$ eseguendo $O((n/k)\log(n/k))$ operazioni aritmetiche tra scalari complessi.

Suggerimento: Prima di scrivere lo pseudocodice, si consiglia di studiare la struttura della trasformata servendosi dell'algoritmo di Cooley-Tukey.

Answer: Let us "simulate" the Cooley-Tukey algoritm on a (k, n, z)–interleaved vector for $n = n/k \times k$. When we first fold the vector (in row-major) into a matrix X, we have that the first column contains elements whose indices are multiples of k, while all the other elements correspond to indices i such that $i \bmod k \neq 0$, hence they are all equal to z. This means that we only have to compute $\boldsymbol{w} = DFT_{n/k}(X^0)$, since all the other

column transforms can be obtained analytically as

$$DFT_{n/k}(X^j) = ((n/k)z, 0, \ldots, 0), 1 \leq j \leq k-1.$$

The subsequent multiplications by twiddle factors leaves the matrix unchanged, since, after the colum transforms, $X_{i,j} = 0$ for $i \cdot j \neq 0$. Finally, the row transforms can be computed analytically again. We have:

$$DFT_k(X_0) = DFT_k(w_0, (n/k)z, \ldots, (n/k)z)) = (w_0+(n(k-1)/k)z, w_0-(n/k)z, \ldots, w_0-(n/k)z)$$

and, for $1 \leq i \leq n/k-1$,

$$DFT_k(X_i) = DFT_k(w_i, 0, \ldots, 0) = (w_i, w_i, \ldots, w_i).$$

The final transform can be obtained by reading the matrix entries in colum-major order.

The above passages are implemented in the algorithm below.

SPARSE_FFT($\boldsymbol{x}, k$)
$n \leftarrow$ length($\boldsymbol{x}$)
$z \leftarrow x_{n-1}$
$\boldsymbol{X^0} \leftarrow (x_0, x_k, x_{2k}, \ldots, x_{(n/k-1)k})$
$\boldsymbol{w} \leftarrow$ FFT($\boldsymbol{X^0}$)
$y_0 \leftarrow w_0 + (n(k-1)/k)z$
for $i \leftarrow$ 1 **to** $n/k-1$ **do**
    $y_i \leftarrow w_i$
{first n/k elements of the transform}
$t \leftarrow w_0 - (n/k)z$
for $j \leftarrow 1$ **to** $k-1$ **do**
    $y_{(n/k)j} \leftarrow t$
    for $i \leftarrow$ 1 **to** $n/k-1$ **do**
    $y_{(n/k)j+i} \leftarrow w_i$
return $\boldsymbol{y}$

The correctness of the above algorithm follows from the correctness of the Cooley-Tukey algorithm. As for its complexity, note that we only compute an FFT on an n/k-vector and make a constant number of additional operations between complex scalars in the subsequent code, for an overall running time of $\Theta(n/k \log(n/k))$, as required. □

Esercizio 3 [8 punti] Sia $n > 0$. Dato un vettore di interi $\boldsymbol{a}$ a n componenti, si consideri la seguente ricorrenza, definita per tutti i valori (i, j), con $1 \leq i \leq j \leq n$:

$$B(i,j) = \begin{cases} a_i & i = j,\ 1 \leq i \leq n, \\ \max\{B(i,k) \cdot B(k+1,j) : i \leq k \leq j-1\} & 1 \leq i < j \leq n. \end{cases}$$

Si sviluppi e si analizzi un algoritmo iterativo che, dato in ingresso il vettore $\boldsymbol{a}$, restituisca il valore $B(1,n)$ in tempo $O(n^3)$.

Answer:

In order to organize a bottom-up computation, let $\ell_{i,j} = j - i + 1$ and note that the computation of $B(i,j)$ depends on elements $B(i,k)$ and $B(k+1,j)$, with $i \leq k < j$. For such range of values of k, we have $\ell_{i,k} = k - i + 1 < j - i + 1 = \ell_{i,j}$ and $\ell_{k+1,j} = j - k \leq j - i < j - i + 1 = \ell_{i,j}$. Hence, we can organize the computation of $B(1,n)$ in a bottom-up fashion by computing the table entries by increasing values of ℓ. This corresponds to scanning the upper triangle of the table storing the values $B(i,j)$, for $1 \leq i \leq j \leq n$, by diagonals parallel to the principal one. The algorithm follows.

```
COMPUTE_B(a)
n ← length(a)
for i ← 1 to n do
    B[i, i] ← a_i
for ℓ ← 2 to n do
    for i ← 1 to n − ℓ + 1 do
        j ← i + ℓ − 1
        B[i, j] ← −∞
        for k ← i to j − 1 do
            B[i, j] ← MAX(B[i, j], B[i, k] ∗ B[k + 1, j])
return B[1, n]
```

The correctness of the above algorithm follows from the fact that it correctly implements a bottom-up computation of $B[1, n]$, as argued above. Its running time is dominated by the overall number of iterations of the three nested loops, which yield a time complexity of

$$\Theta\left(\sum_{\ell=2}^{n}\sum_{i=1}^{n-\ell+1}\sum_{k=i}^{i+\ell-2}1\right) = \Theta\left(n^3\right).$$

(This is the same analysis of the matrix-chain multiplication algorithm seen in class.) □

Esercizio 4 [8 punti] Si consideri il seguente problema decisionale:

0-1 KNAPSACK (01K):

ISTANZA: $\langle \boldsymbol{a}, b\rangle$, $\boldsymbol{a} = (a_1, a_2, \ldots, a_k) \in \mathbf{N}^k, a_i \neq a_j$ per $i \neq j$ e $b \in \mathbf{N}$.
DOMANDA: Esiste un vettore $x \in \{0,1\}^k$ tale che $\boldsymbol{a}^T\boldsymbol{x} = \sum_{i=1}^{k} a_i x_i = b$?

Si dimostri che 01K è NP-Hard. (*Suggerimento:* Si riduca da SUBSET-SUM (SS).)

Answer: The reduction from SS is immediate. Our reduction function is defined as follows:

$$f(\langle S, t\rangle) = \langle \boldsymbol{a'}, b'\rangle$$

with $\boldsymbol{a'} = (s_1, s_2, \ldots, s_{|S|})$ being a vector containing all the elements of S, and $b' = t$.

Clearly,f can be computed in polynomial (in fact, linear) time by simply scanning S and copying its elements into $\boldsymbol{a'}$. To see that f correctly reduces SS to 01K, consider a positive instance $\langle S, t\rangle$ of SS. Then, there is a subset $S' \subseteq S$ such that $\sum_{s' \in S'} s' = t$. Let $S' = \{s_{j_1}, s_{j_2}, \ldots, s_{j_{|S'|}}\}$. Then, by considering a 0-1 vector $\boldsymbol{x'}$ such that $x_i = 1$ if and only if $\exists h : i = j_h$, we have that $\boldsymbol{a'}^T\boldsymbol{x'} = b'$.

Viceversa, if $f(\langle S, t\rangle) = \langle \boldsymbol{a'}, b'\rangle \in$ 01K, then there is a 0-1 vector x' such that $\boldsymbol{a'}^T\boldsymbol{x'} = b'$. The nonzero entries of x' identify a subset $S' \subseteq S$ such that $\sum_{s' \in S'} s' = b' = t$. Hence $\langle S, t\rangle \in SS$. □

Attenzione: Risposte immotivate e prive di prova di correttezza non saranno considerate. Inoltre, gli algoritmi da sviluppare negli Esercizi 2 e 3 vanno descritti utilizzando lo **pseudocodice** usato in classe.

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(4/9/2003)

Durata: 2h 30m

Esercizio 1 [8 punti] Si consideri la seguente equazione di ricorrenza, definita per valori arbitrari del parametro $n > 0$:

$$T(n) = \begin{cases} 2T(\lfloor n/3 \rfloor) + n & n > 3, \\ 1 & n \leq 3. \end{cases}$$

Si dimostri utilizzando l'induzione parametrica che $T(n) \in O(n)$

Answer: In order to apply parametric induction, let us assume that $T(n) \leq cn$ for some constant c (to be determined by the analysis). For the base case $n \leq 3$, we have $T(n) = 1 \leq 1 \cdot n$, whence $c \geq 1$ (obtained for $n = 1$). For $n > 3$ we have:

$$T(n) = 2T(\lfloor n/3 \rfloor) + n \leq 2c\lfloor n/3 \rfloor + n \leq 2cn/3 + n.$$

Therefore, it suffices that

$$2cn/3 + n \leq cn \Leftrightarrow c \geq 3.$$

Therefore, we have that $T(n) \leq 3n$ for $n > 0$, which can be easily verified using standard induction (verification omitted for brevity). □

Esercizio 2 [8 punti] Per $n > 0$, si consideri la seguente equazione vettoriale:

$$(\boldsymbol{x} \circledast \boldsymbol{x}) + \boldsymbol{b_n} = \boldsymbol{0},$$

dove $\boldsymbol{x}$ è il vettore delle n incognite complesse, $\circledast$ denota l'operatore di convoluzione ciclica, $\boldsymbol{0}$ il vettore nullo e $\boldsymbol{b_n} = (1, 1, \ldots, 1)$ il vettore le cui n componenti sono tutte uguali a 1. Si dimostri che l'equazione ammette solo due soluzioni distinte in $\boldsymbol{C}^n$ e si determinino tali soluzioni.

Answer: Let $\boldsymbol{X} = DFT_n(\boldsymbol{x})$ and $\boldsymbol{B_n} = DFT_n(\boldsymbol{b_n})$. Note first that by the summation lemma it follows that

$$[\boldsymbol{B_n}]_i = \sum_{j=0}^{n-1} b_j \omega_n^{ij} = \sum_{j=0}^{n-1} \omega_n^{ij} = \begin{cases} n, & i = 0 \\ 0, & 1 \leq i \leq n-1. \end{cases}$$

By applying the cyclic convolution theorem, we can solve the vector equation by first solving the equation

$$\boldsymbol{X} \odot \boldsymbol{X} = (-n, 0, \ldots, 0)$$

(where $\odot$ denotes component-wise product) and then computing the inverse Fourier transform of each distinct vector solution. The above vector equation is equivalent to the following system, made of decoupled, scalar, degree-two equations:

$$\begin{cases} X_i^2 = -n, & i = 0 \\ X_i^2 = 0, & 1 \leq i \leq n-1. \end{cases}$$

The only two solutions are then $\boldsymbol{X_0} = (i\sqrt{n}, 0 \ldots, 0)$ and $\boldsymbol{X_1} = (-i\sqrt{n}, 0 \ldots, 0)$. Finally, we obtain the desired solutions $\boldsymbol{x_0}$ and $\boldsymbol{x_1}$ by anti-transforming $\boldsymbol{X_0}$ and $\boldsymbol{X_1}$. Recalling that $DFT_n^{-1}(\boldsymbol{y}) = (1/n)DFT_n(\boldsymbol{y^{rev}})$, we obtain $\boldsymbol{x_0} = (i/\sqrt{n}, i/\sqrt{n}, \ldots, i/\sqrt{n})$ and $\boldsymbol{x_1} = (-i/\sqrt{n}, -i/\sqrt{n}, \ldots, -i/\sqrt{n})$ □

Esercizio 3 [8 punti] Date due stringhe di interi Z^1 e Z^2 con $|Z^1| = |Z^2| = k$, di definisca la loro *pseudodifferenza*, $\text{psd}(Z^1, Z^2) = \sum_{i=1}^{k} (Z_i^1 - Z_i^2)$ (si ponga, per convenzione, $\text{psd}(\epsilon, \epsilon) = 0$). Si progetti e si analizzi un algoritmo di programmazione dinamica che, date in ingresso due stringhe X e Y, calcoli la massima pseudodifferenza realizzata da una sottosequenza di X e una sottosequenza di Y di egual lunghezza. La complessità dell'algoritmo deve essere $\Theta(|X||Y|)$.

Answer: Let us choose as our subproblem space the cartesian product of all prefixes of strings X and Y, that is, the set $\{(X_i, Y_j) : 0 \leq i \leq |X|, 0 \leq j \leq |Y|\}$. Our dynamic programming algorithm will be based on the following optimal substructure property. Clearly, if either $i = 0$ or $j = 0$ (i.e., one of the two prefixes is the empty prefix), then the maximum pseudodifference achievable by two equal-length subsequences of X_i and Y_j is $\text{psd}(\epsilon, \epsilon) = 0$. Otherwise, let $Z^1 = \langle z_1^1, z_2^1, \ldots, z_k^1 \rangle$ and $Z^2 = \langle z_1^2, z_2^2, \ldots, z_k^2 \rangle$ be the two subsequences of X_i and Y_j, respectively, which achieve maximum pseudodifference. We have three cases: (1) if $z_k^1 = x_i$ and $z_k^2 = y_j$, then Z_{k-1}^1 and Z_{k-1}^2 must be subsequences of X_{i-1} and Y_{j-1}, respectively, achieving maximum pseudodifference among all equal-length subsequences of X_{i-1} and Y_{j-1}, or we could obtain a larger pseudodifference in X_i and Y_j than the one realized by Z^1 and Z^2. Using a similar argument, it is easy to see that (2) if $z_k^1 \neq x_i$, then Z^1 is a subsequence of X_{i-1} and it must achieve maximum pseudodifference with some subsequence of Y_j. Analogously, (3) if $z_k^2 \neq y_j$, then Z^2 is a subsequence of Y_{j-1} realizing maximum pseudodifference with some subsequence of X_i.

Let $P(i, j)$ be the maximum pseudodifference achievable by equal-length subsequences of X_i and Y_j. The above substructure property immediately yields the following recurrence for $P(i, j)$:

$$P(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{x_i - y_j + P(i-1, j-1), P(i-1, j), P(i, j-1)\} & \text{otherwise.} \end{cases}$$

From the recurrence, we immediately obtain the desired algorithm.

```
PSEUDODIFFERENCE(X, Y)
m ← length(X)
n ← length(Y)
for i ← 0 to m do
    D[i, 0] ← 0
for j ← 1 to n do
    D[0, j] ← 0
for i ← 1 to m do
    for j ← 1 to n do
        D[i, j] ← MAX(x_i − y_j + D[i − 1, j − 1], D[i − 1, j], D[i, j − 1])
return D[m, n]
```

The correctness of the above algorithm easily follows from the correctness of the recurrence and from the fact that the row-major scan computes every table entry prior to any of its uses. The running time of the algorithm is clearly $\Theta(mn) = \Theta(|X||Y|)$. □

Esercizio 4 [8 punti] Si consideri il seguente problema decisionale:

CONSTRAINED SUBSET INTERSECTIONS (CSI) :

ISTANZA: $\langle S, C_1, \ldots, C_m, V_1, \ldots, V_n \rangle$, $S \subseteq \mathbf{N}$, finito, $C_i, V_j \subseteq S$, $1 \leq i \leq m$, $1 \leq j \leq n$.

DOMANDA: Esiste $T \subseteq S$ tale che $|T \cap C_i| \geq 1$ e $|T \cap V_j| = 1$ per $1 \leq i \leq m$ e $1 \leq j \leq n$?

Si dimostri che CSI è *NP*-Hard. (*Suggerimento:* Si riduca da 3-CNF SAT.)

Answer: We reduce as follows. Consider a generic 3-CNF SAT instance

$$\Phi(x_1, x_2, \ldots, x_n) = \bigwedge_{i=1}^{m} K_i,$$

where each K_i, $1 \leq i \leq m$, is a disjunction of exactly three literals over the set of n variables of Φ. In the reduction, we let $S = \{1, 2, \ldots, n, n+1, \ldots, 2n\}$, where, for $1 \leq j \leq n$, value j corresponds to the literal x_j, and value $j+n$ to the literal $\neg x_j$. Note that for $1 \leq i \leq m$, each clause K_i can now be represented as a subset C_i of three elements over S. Finally, for $1 \leq j \leq n$ we define subset V_j as $\{j, j+n\}$. The above reduction is clearly computable in polynomial (in fact, linear) time.

We now show that f is a valid reduction.

($\Rightarrow$) If $\Phi \in$ 3-CNF SAT, then there exists a truth assignment $\boldsymbol{b}$ to its n variables such that $\Phi(\boldsymbol{b}) = 1$. Let T be the subset of S containing, for each j, $1 \leq j \leq n$, j itself if $b_j =$ **true** and $j+n$ otherwise. By construction, we then have that $|T \cap V_j| = 1$ for $1 \leq j \leq n$. Moreover, since $\boldsymbol{b}$ is a satisfying assignment, there must be at least one true literal in each clause. Since T contains all possible true literals, we have that $|T \cap C_i| \geq 1$ for $1 \leq i \leq m$.

($\Leftarrow$) If $f(\Phi) \in$ CSI, consider the n intersections between T and the V_j's and define a truth assignment $\boldsymbol{b}$ such that $b_j =$ **true** if $T \cap V_j = \{j\}$, and **false** if $T \cap V_j = \{n+j\}$ (note that it must be $|T \cap V_j| = 1$.) It is then clear that Φ is satisfied by $\boldsymbol{b}$, since for $1 \leq i \leq m$, $|T \cup C_i| \geq 1$ implies that clause K_i contains at least one literal which is true under $\boldsymbol{b}$. $\square$

Fondamenti di Informatica II – Ingegneria Informatica

COMPITO
(18/9/2003)

Durata: 2h 30m

Esercizio 1 [8 punti] Si risolva e si verifichi la seguente equazione di ricorrenza definita per valori del parametro $n = 3^{3^i}$:

$$T(n) = \begin{cases} n^{4/3} T\left(n^{1/3}\right) + n^2 & n > 3, \\ 0 & n = 3. \end{cases}$$

Answer: Use the general formula taking into account that $s(n) = n^{4/3}$, $f^i(n) = n^{1/3^i}$, $f^*(n, 3) = \log_3 \log_3 n - 1$, and $w(n) = n^2$. We have that:

$$\prod_{j=0}^{\ell-1} s(f^j(n)) = \prod_{j=0}^{\ell-1} \left(n^{\frac{1}{3^j}}\right)^{\frac{4}{3}} = n^{\frac{4}{3}\sum_{j=0}^{\ell-1}\frac{1}{3^j}} = n^{2\left(1-\frac{1}{3^\ell}\right)},$$

whence

$$\left[\prod_{j=0}^{\ell-1} s(f^j(n))\right] w(f^\ell(n)) = n^{2\left(1-\frac{1}{3^\ell}\right)} \cdot n^{\frac{2}{3^\ell}} = n^2.$$

Therefore

$$\begin{aligned} T(n) &= \sum_{\ell=0}^{\log_3 \log_3 n - 1} n^2 \\ &= n^2 \log_3 \log_3 n, \end{aligned}$$

which can be easily verified using induction (verification omitted for brevity). □

Esercizio 2 [8 punti] Per n potenza di due, sia $\boldsymbol{x}$ un vettore di n componenti complesse e sia $C(\boldsymbol{x})$ la matrice circolante la cui prima colonna è $\boldsymbol{x}$. Descrivere ed analizzare un algoritmo **efficiente** ($O(n \log n)$ per avere punteggio pieno) per il calcolo del vettore $\boldsymbol{z}$ tale che $C(\boldsymbol{z}) = [C(\boldsymbol{x})]^n$.

Answer: Since $C(\boldsymbol{x}) \times C(\boldsymbol{y}) = C(\boldsymbol{x} \circledast \boldsymbol{y})$, it immediately follows that $\boldsymbol{z}$ is the convolution of n vectors all equal to $\boldsymbol{x}$:

$$\boldsymbol{z} = \underbrace{\boldsymbol{x} \circledast \boldsymbol{x} \circledast \cdots \circledast \boldsymbol{x}}_{n \text{ times}}.$$

Let $\boldsymbol{X} = F_n(\boldsymbol{x})$ and $\boldsymbol{Z} = F_n(\boldsymbol{z})$. From the cyclic convolution theorem, we have:

$$Z_k = (X_k)^n, \quad \text{for } k = 0, 1, \ldots, n-1.$$

The algorithm immediately follows:

```
CIRCULANT_EXPONENTIATE(x)
    n ← length(x)
    X ← FFT(x)
    for i ← 0 to n − 1
        do Z_i ← EXPONENTIATE(X_i, n)
    return INV_FFT(Z)
```

In the above algorithm, we made use of the subroutine EXPONENTIATE (as seen in class), which computes the n-th power of a complex number through repeated squaring in time $O(\log n)$.

The correctness of the algorithm follows from the previous discussion. Its running time is dominated by the time for two FFT computations ($O(n \log n)$ time) and n calls to EXPONENTIATE ($n \times O(\log n)$ time), for an overall running time of $O(n \log n)$. □

Esercizio 3 [8 punti] Per una stringa di interi $Z = \langle z_1, z_2, \ldots, z_k \rangle$ sia peso$(Z) = \sum_{i=1}^{k} z_i$ (si noti che peso$(\epsilon) = 0$). Date due stringhe di interi $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$, si consideri il problema di determinare una Common Subsequence (CS) Z di X e Y di peso massimo.

(a) **[4 punti]** Si provi una proprietà di sottostruttura ottima per Z e si derivi da essa una ricorrenza appropriata per il peso massimo di una CS.

(b) **[4 punti]** Si scriva lo pseudocodice iterativo per il calcolo bottom-up della ricorrenza sviluppata al Punto **(a)**. L'algoritmo deve avere complessità $\Theta(mn)$.

Suggerimento: Si faccia attenzione ai valori < 0.

Answer:

(a) Let $Z = \langle z_1, z_2, \ldots z_k \rangle$ be a Maximum-Weight Common Subsequence (MWCS, for short) of X and Y, with $Z = \langle x_{j_1}, x_{j_2}, \ldots x_{j_k} \rangle = \langle y_{h_1}, y_{h_2}, \ldots y_{h_k} \rangle$, with $1 \leq j_1 < j_2 < \ldots < j_k \leq m$ and $1 \leq h_1 < h_2 < \ldots < h_k \leq n$. Note that for $1 \leq i \leq k$, it must be $z_i \geq 0$, or otherwise $Z' = \langle z_1, \ldots z_{i-1}, z_{i+1}, \ldots z_k \rangle$ (which is a CS of X and Y) would have a higher weight than Z. Furthermore, we can assume without loss of generality that $z_i \neq 0$, since otherwise we can simply remove z_i, so that Z' remains an MWCS of X and Y. We have:

1. *If $x_m = y_n \leq 0$, then Z is an MWCS of X_{m-1} and Y_{n-1}.*
   Proof: Since Z contains only numbers greater than zero, it must be $j_k < m$ and $h_k < n$. Hence Z is a CS of X_{m-1} and Y_{n-1}. Clearly, it must be an MWCS of X_{m-1} and Y_{n-1}, or otherwise an MWCS of X_{m-1} and Y_{n-1} would also be a CS of X and Y heavier than Z.

2. *If $x_m = y_n > 0$, then $z_k = x_m$ and Z_{k-1} is an MWCS of X_{m-1} and Y_{n-1}.*
   Proof: If $z_k \neq x_m = y_n$, it must be $j_k < m$ and $h_k < n$. But then $Z' = \langle < Z, x_m \rangle$ is still a CS of X and Y with weight $x_m + \text{weight}(Z) > \text{weight}(Z)$, a contradiction. Also, Z_{k-1} is a CS of X_{m-1} and Y_{n-1}. By arguing as above we claim that it must be the one of maximum weight.

3. *If $x_m \neq y_n$ then Z is the sequence of maximum weight between an MWCS of X_m and Y_{n-1} and an MWCS of X_{m-1} and Y_n.*
   Proof: Since $x_m \neq y_n$, either $j_k < m$ or $h_k < n$. Hence Z must be either an MWCS of X_{m-1} and Y (first case) or an MWCS of X and Y_{n-1} (second case). Clearly, Z must be the sequence of largest weight among the two.

Let $W[i, j]$ be the weight of an MWCS of X_i and Y_j, with $0 \leq i \leq m$ and $0 \leq j \leq n$. The above property implies the following recurrence for $W[i, j]$.

$$W[i,j] = \begin{cases} 0 & i = 0 \text{ or } j = 0, \\ W[i-1, j-1] & \text{if } x_i = y_j \leq 0, \\ x_i + W[i-1, j-1] & \text{if } x_i = y_j > 0, \\ \max\{W[i-1, j], W[i, j-1]\} & \text{otherwise.} \end{cases}$$

(b) The bottom-up computation of the recurrence of Point **(a)** can be performed as follows:

```
MWCS(X, Y)
    m ← length(X)
    n ← length(Y)
    for i ← 0 to m do W[i, 0] ← 0
    for j ← 1 to n do W[0, j] ← 0
    for i ← 1 to m do
        for j ← 1 to n do
            if (x_i = y_j)
                then if (x_i ≤ 0)
                        then W[i, j] ← W[i − 1, j − 1]
                        else W[i, j] ← x_i + W[i − 1, j − 1]
                else W[i, j] ← MAX(W[i, j − 1], W[i − 1, j])
    return W[m, n]
```

The correctness of the above algorithm follows from Point **(a)** and from the fact that the values $W[i-1, j-1], W[i, j-1]$ and $W[i-1, j]$ have already been computed when $W[i, j]$ is being computed. The algorithm's running time is clearly $\Theta(mn)$. □

Esercizio 4 [8 punti] Si consideri il seguente problema decisionale:

CHS (**C**ONSTRAINED **H**ITTING **S**ET):
ISTANZA: $\langle n, k, E_1, E_2, \ldots E_m\rangle$, $1 \leq k \leq n$, $E_i \subseteq \{1, 2, \ldots n\}$, $|E_i| = 2$, $1 \leq i \leq m$.
DOMANDA: Esiste un sottoinsieme $V' \subseteq \{1, 2, \ldots, n\}$ con $|V'| = k$ e tale che $V' \cap E_i \neq \emptyset$, per $1 \leq i \leq m$?

Dimostrare che CHS è NP-Completo. (*Suggerimento:* Ridurre da VERTEX_COVER)

Answer: The proof that CHS is in NP is trivial and amounts to developing a polynomial verification algorithm for the problem. We skip it here for the sake of brevity.

In order to show that CHS is NP-Hard, we show that VERTEX_COVER (VC) $<_P$ CHS. Recall that an instance of VC is $\langle G = (V, E), k\rangle$, with $V \subseteq \mathbf{N}$ finite and $E \subseteq \{\{u, v\} : u \neq v, u, v \in V\}$, and the question is whether V contains a subset V' of size k such that each edge in E has at least one of its endpoints incident on V'. Let $\pi : V \to \{1, 2, \ldots, |V|\}$ be an arbitrary one-to-one function from V to $\{1, 2, \ldots, |V|\}$. Our reduction function is the following:

$$f\left(\langle G = (V, E), k\rangle\right) = \langle |V|, k, E_1, E_2, \ldots E_{|E|}\rangle,$$

where $E_i = \{\pi(u), \pi(v)\}$ if the i-th edge in E is $e_i = \{u, v\}$.

Clearly, f is computable in polynomial (in fact, linear) time. To show that f reduces VC to CHS, observe that if V' is a vertex cover of G of cardinality k, then each edge in E (seen as a subset of two elements) contains at least one element in V'. This means that if we consider the set $\pi(V') = \{i : \exists v \in V' : i = \pi(v)\}$, then $\pi(V') \cap E_i \neq \emptyset$ for $1 \leq i \leq |E|$. Vice versa, if $f(\langle G, \rangle) \in$ CHS), consider the cardinality k subset $V' \subseteq \{1, 2, \ldots, |V|\}$ such that $V' \cap E_i \neq \emptyset$ for $1 \leq i \leq |E|$. Then, $\pi^{-1}(V') \subseteq V$ contains at least one node for each edge, hence $\langle G, k\rangle \in$ VC. □

Attenzione: Risposte immotivate e prive di prova di correttezza non saranno considerate. Inoltre, gli algoritmi da sviluppare negli Esercizi 2 e 3 vanno descritti utilizzando lo **pseudocodice** usato in classe.

Fondamenti di Informatica II – Ingegneria Informatica
Compito, 17/3/2004 (Durata: 3h)

Esercizio 1 [8 punti] Utilizzando l'induzione parametrica, si individui la minima costante $c > 0$ per cui risulti essere $T(n) \leq cn$, per $n \geq 1$, dove

$$T(n) = \begin{cases} 5 & n \leq 4 \\ T(\lfloor n/3 \rfloor) + T(\lfloor n/5 \rfloor) + 7n & n > 4. \end{cases}$$

Answer: From the (parametric) induction base, we obtain $c \geq 5$. When going from inductive hypothesis to thesis, we get:

$$T(n) \leq c(n/3) + c(n/5) + 7n \boxed{\leq} cn \Leftrightarrow (7/15)cn \geq 7n \Leftrightarrow c \geq 15$$

Hence $c \geq \max\{5, 15\} = 15$. □

Esercizio 2 [8 punti] Sia n una potenza di due. Una matrice intera $n \times n$ A si dice *supertriangolare di ordine* n se $n = 1$ oppure

$$A = \begin{bmatrix} A & B \\ 0 & C \end{bmatrix},$$

con A, B e C matrici supertriangolari di ordine $n/2$.

1. Dimostrare che la somma di due matrici supertriangolari è una matrice supetriangolare e sviluppare e analizzare un algoritmo di somma che esegue un numero $T_{\text{sum}}(n)$ ottimo di operazioni aritmetiche tra scalari interi.
2. Dimostrare che il prodotto di due matrici supertriangolari è una matrice supertriangolare e sviluppare e analizzare un algoritmo di moltiplicazione che esegue $T_{\text{mult}}(n) = O(n^2)$ operazioni tra scalari interi.

Answer: Point 1 The proof proceeds by induction. The base $n = 1$ is trivial. Assume that the hypothesis holds for $k < n$. For $k = n$, let

$$A = \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix} \text{ and } B = \begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix}$$

be two supertriangular matrices of order n. Then,

$$C = A + B = \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix} + \begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix} = \begin{bmatrix} A_1 + B_1 & A_2 + B_2 \\ 0 & A_3 + B_3 \end{bmatrix} = \begin{bmatrix} C_1 & C_2 \\ 0 & C_3 \end{bmatrix}.$$

By the inductive hypothesis, $C_1 = A_1 + B_1$, $C_2 = A_2 + B_2$, and $C_3 = A_3 + B_3$, are supertriangular matrices of order $n/2$, hence C is supertriangular of order n. The algorithm for summing two supertriangular matrices is the following:

ST_SUM(A, B)
$n \leftarrow$ rows(A)
if n=1 **then return** $a_{1,1} + b_{1,1}$
$\star$ let $A = \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix}$ and $B = \begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix}$ $\star$
$C_1 \leftarrow$ ST_SUM(A_1, B_1)
$C_2 \leftarrow$ ST_SUM(A_2, B_2)
$C_3 \leftarrow$ ST_SUM(A_3, B_3)
return $\begin{bmatrix} C_1 & C_2 \\ 0 & C_3 \end{bmatrix}$

The correctness of the above algorithm follows from the fact that the sum of two supertriangular matrices is still supertriangular. It running time, in terms of scalar operations (sums) is regulated by the following recurrence:

$$T(n) = \begin{cases} 1 & n = 1, \\ 3T(n/2) & n > 1 \end{cases}$$

whose solution, by the Master Theorem, is $T(n) = \Theta\left(n^{\log_2 3}\right)$ (indeed, $T(n) = n^{\log_2 3}$ as can be easily proved by induction). This is clearly optimal, since $T(n)$ is also the maximum number of distinct nonzero entries of a supetriangular matrix. Therefore, in the worst case, we need to add at least those many scalars to obtain the sum of two supetriangular matrices.

Point 2 As before, we use induction. The base $n = 1$ is trivial. Assume that the hypothesis holds for $k < n$. For $k = n$, let

$$A = \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix} \text{ and } B = \begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix}$$

be two supertriangular matrices of order n. Then,

$$C = A \times B = \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix} \times \begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix} =$$

$$= \begin{bmatrix} A_1 \times B_1 & A_1 \times B_2 + A_2 \times B_3 \\ 0 & A_3 \times B_3 \end{bmatrix} = \begin{bmatrix} C_1 & C_2 \\ 0 & C_3 \end{bmatrix}.$$

By the inductive hypothesis and the fact that the sum of supertriangular matrices is still supertriangular, C_1, C_2 and C_3 are supertriangular matrices of order $n/2$, hence C is supertriangular of order n. The algorithm for multiplying two supertriangular matrices is the following:

ST_MUL(A, B)
$n \leftarrow$ rows(A)
if n=1 **then return** $a_{1,1}b_{1,1}$
$\star$ let $A = \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix}$ and $B = \begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix}$ $\star$
$C_1 \leftarrow$ ST_MUL(A_1, B_1)
$C_2 \leftarrow$ ST_SUM(ST_MUL(A_1, B_2), ST_MUL(A_2, B_3))
$C_3 \leftarrow$ ST_MUL(A_3, B_3)
return $\begin{bmatrix} C_1 & C_2 \\ 0 & C_3 \end{bmatrix}$

The correctness of the above algorithm follows from the fact that the sum and product of two supertriangular matrices is still supertriangular. Its running time, in terms of scalar operations (sums and products) is regulated by the following recurrence:

$$T(n) = \begin{cases} 1 & n = 1, \\ 4T(n/2) + n^{\log_2 3}/3 & n > 1 \end{cases}$$

(the additive term $n^{\log_2 3}/3$ comes from the call to ST_SUM on supertriangular matrices of order $n/2$) whose solution, by the Master Theorem, is $T(n) = \Theta(n^2)$. □

Esercizio 3 [8 punti] I *numeri euleriani del secondo ordine* $E(n, k)$, per $0 \leq k \leq n$, sono caratterizzati dalla seguente ricorrenza:

$$E(n,k) = \begin{cases} 1 & k = 0 \\ 0 & k > 0, k = n \\ (k+1)E(n-1,k) + (2n-1-k)E(n-1,k-1) & 0 < k < n. \end{cases}$$

1. Si fornisca il codice di un algoritmo memoizzato (diviso in due procedure) che, dati in ingresso n e k, restituisca $E(n, k)$.

2. Si analizzi la complessità $T(n, k)$ dell'algoritmo associando costo unitario alla somma e sottrazione di interi.

Answer: Point 1 The code of the memoized algorithm that computes $E(n, k)$ is divided into two subroutines. The first, INIT_E(n, k), solves base cases and initializes an $(n + 1) \times (k + 1)$ table. Finally, for non-base values, INIT_E(n, k) makes the call REC_E$(n.k)$ to the second subroutine, which computes $E(n, k)$ using the table to avoid replicating subproblems. We will use -1 as a default value in the table (indicating that the subproblem has not been solved yet) since an easy induction suffices to show that for all relevant values, $E(n, k)$ is non negative. Finally, particular care must be taken when initializing the table for base cases, which correspond to all pairs $(i, 0)$, $0 \leq i \leq n$ and (j, j), $1 \leq j \leq k$. The codes follow

```
INIT_E(n, k)
if k = 0 then return 1
if k = n then return 0
for i = 0 to n do
    E[i, 0] ← 1
for i = 1 to n do
    for j = 1 to min{k, i} do
    E[i, j] ← −1
for j = 1 to k do
    E[j, j] ← 0
return REC_E(n, k)
```

```
REC_E(i, j)
if E[i, j] = −1 then
    a ← REC_E(i − 1, j)
    b ← REC_E(i − 1, j − 1)
    E[i, j] ← (j + 1) · a + (2i − 1 − j) · b
return E[i, j]
```

Point 2 To evaluate the running time of the algorithm, note that the initialization routine does not perform either sums or subtractions, hence its cost is null. As for REC_E, observe that in its associated recursion tree we have a single internal node for each subproblem (i, j) for which we had $E(i, j) = -1$. These nodes are $\Theta(nk)$. Since each internal node contributes a constant number of sums and subtractions to the overall running time, we have $T(n, k) = \Theta(nk)$ □

Esercizio 4 [8 punti] Si consideri il seguente problema decisionale:

DSS (DISJOINT SUBSETS SUM):
INSTANCE: $\langle S, t_1, t_2\rangle$, $S \subseteq \boldsymbol{N}$, $t_1, t_2 \in \boldsymbol{N}$
QUESTION: Esistono $S_1, S_2 \subseteq S$, con $S_1, S_2 \neq \emptyset$ e $S_1 \cap S_2 = \emptyset$ per cui

$$t_1 = \sum_{s_1 \in S_1} s_1 \quad \text{e} \quad t_2 = \sum_{s_2 \in S_2} s_2 \ ?$$

Dimostrare che DSS è NP-Hard.

Answer: We show that SUBSET SUM (SS) $<_P$ DSS. Recall that an instance of the former problem is simply $\langle S, t\rangle$, with $S \subseteq \boldsymbol{N}$ and $t \in \boldsymbol{N}$. The question is whether there exists a subset $S_1 \subseteq S$ such that $\sum_{s_1 \in S_1} s_1 = t$. Let

$$\bar{s} = 1 + \sum_{s \in S} s$$

and define the following reduction function.

$$f(\langle S, t\rangle) = \langle S \cup \{\bar{s}\}, t, \bar{s}\rangle.$$

Function f is clearly computable in polynomial time, since $\bar{s}$ can be computed in time linear in $|S|$, and the encoding $\langle S \cup \{\bar{s}\}, t, \bar{s}\rangle$ can be computed in time polynomial in $|\langle S, t\rangle|$. Let us now show that f reduces SS to DSS.

Let $x = \langle S, t\rangle \in$ SS. Then, there exists $S_1 \subseteq S$ such that $\sum_{s_1 \in S_1} s_1 = t$. Let $S_2 = \{\bar{s}\}$. Clearly, $S_1, S_2 \neq \emptyset$ and $S_1 \cap S_2 = \emptyset$ (since $\bar{s} \notin S$). Moreover, $\sum_{s_2 \in S_2} s_2 = \bar{s}$, hence $f(x) \in$ DSS.

Viceversa, let $f(x) \in$ DSS. Then, there exists $S_1, S_2 \subseteq S \cup \{\bar{s}\}$ such that $\sum_{s_1 \in S_1} s_1 = t$, $\sum_{s_2 \in S_2} s_2 = \bar{s}$. Since it must be $S_2 = \{\bar{s}\}$ (there is no other choice of elements in $S \cup \{\bar{s}\}$ which sum to $\bar{s}$), and $S_1 \cap S_2 = \emptyset$, we have $S_1 \subseteq S$, hence $x \in$ SS. □

Fondamenti di Informatica II – Ingegneria Informatica
Compito, 6/9/2004 (Durata: 2h30m)

Esercizio 1 [8 punti] Sia n una potenza di due. Utilizzando l'induzione parametrica, si individui la minima costante $c > 0$ per cui risulti $T(n) \leq cn \log_2 n$, per $n \geq 1$, dove

$$T(n) = \begin{cases} 0, & n = 1, 2, 4 \\ T(n/2) + T(n/4) + 2T(n/8) + 7n, & n \geq 8 \end{cases}$$

Answer: From the (parametric) induction base, we obtain, for $n = 1, 2, 4$,

$$0 = T(n) \boxed{\leq} cn \log_2 n \Leftrightarrow c \geq 0.$$

When going from inductive hypothesis to thesis, for $n \geq 8$ a power of two, we get:

$$\begin{array}{rcl} T(n) & \leq & c(n/2)\log_2(n/2) + c(n/4)\log_2(n/4) + 2c(n/8)\log_2(n/8) + 7n \\ & = & cn\log_2 n - c(n/2 + 2 \cdot n/4 + 2 \cdot 3 \cdot n/8) + 7n \\ & = & cn\log_2 n - (7/4)cn + 7n \boxed{\leq} cn \log_2 n \\ & \Leftrightarrow & (7/4)cn \geq 7n \Leftrightarrow c \geq 4. \end{array}$$

Hence $c \geq \max\{0, 4\} = 4$. □

Esercizio 2 [8 punti] Per $n > 0$, sia $\boldsymbol{x} \in \boldsymbol{C}^n$ un vettore di n incognite complesse. Si consideri il sistema di equazioni espresso in forma vettoriale come segue:

$$(\boldsymbol{x} \circledast \boldsymbol{x}) - (2 \cdot \boldsymbol{x}) + (1, 0, \ldots, 0) = \mathbf{0}_n,$$

dove '$\circledast$' denota la convoluzione ciclica; '$-$','$+$' sono operazioni tra vettori, '$\cdot$' è il prodotto tra uno scalare e un vettore, e $\mathbf{0}_n$ è il vettore con n componenti nulle.

1. Si calcoli esplicitamente una soluzione del suddetto sistema.
2. Si determini il numero di soluzioni distinte del sistema.

Answer: Point 1 Let $\boldsymbol{X} = DFT_n(\boldsymbol{x}) = F_n\boldsymbol{x}$, and recall that $DFT_n(1, 0, \ldots, 0) = (1, 1, \ldots, 1)$. By multiplying both sides of the system by F_n and applying the cyclic convolution theorem, we obtain the equivalent system in the unknown vector $\boldsymbol{X}$:

$$(\boldsymbol{X} \odot \boldsymbol{X}) - (2 \cdot \boldsymbol{X}) + (1, 1, \ldots, 1) = \mathbf{0}_n,$$

where '$\odot$' denotes the component-wise multiplication of two vectors. The above system is much easier to solve than the original one, since it is made of the following n scalar equations:

$$X_i^2 - 2X_i + 1 = (X_i - 1)^2 = 0, \text{ for } 0 \leq i \leq n - 1.$$

Clearly vector $\boldsymbol{X} = (1, 1, \ldots, 1)$ is a solution of the transformed system. Hence $\boldsymbol{x} = DFT_n^{-1}(1, 1, \ldots, 1) = (1, 0, \ldots, 0)$ is a solution of the original system, as it can be easily checked by direct computation.

[Point 2] Since each scalar equation has a unique solution of multiplicity 2, vector $(1, 1, \ldots, 1)$ is the unique solution of the transformed system. Therefore, by the equivalence with the original system, vector $(1, 0, \ldots, 0)$ is the unique solution of the latter system. □

Esercizio 3 [8 punti] Per $n > 0$, siano dati due vettori a componenti intere $\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{Z}^n$. Si consideri la quantità $c(i, j)$, definita per ricorrenza su coppie di indici (i, j), con $0 \leq i \leq j \leq n-1$, come segue:

$$c(i,j) = \begin{cases} a_i, & 0 < i \leq n-1 \text{ e } j = n-1, \\ b_j, & i = 0 \text{ e } 0 \leq j \leq n-1, \\ c(i-1, j-1) \cdot c(i, j+1), & 0 < i \leq j < n-1. \end{cases}$$

Si voglia calcolare la quantità $m = \min\{c(i, j) : 0 \leq i \leq j \leq n-1\}$.

1. Si individui il tipo di scansione da effettuare per calcolare m utilizzando un algoritmo iterativo *bottom-up* che riceva in ingresso i vettori $\boldsymbol{a}$ e $\boldsymbol{b}$. Si fornisca inoltre lo pseudocodice dell'algoritmo.
2. Si valuti la *complessità esatta* dell'algoritmo sviluppato associando costo unitario ai prodotti tra numeri interi e costo nullo a tutte le altre operazioni.

Answer: Point 1 The problem requires computing the entries in the upper triangle of an $n \times n$ array C, with $C[i, j] = c(i, j)$, for $0 \leq i \leq j \leq n-1$, and accumulating the minimum m among these entries. Observe that the last column and the first row of the array are given in input, since $C[i, n-1] = a_i$, for $i > 0$, and $C[0, j] = b_j$, for $j \geq 0$, respectively. When $0 < i \leq j < n-1$, entry $C[i, j]$ can be computed if entry $C[i-1, j-1]$ (on the previous row and previous column) and entry $C[i, j+1]$ (on the same row and next column) have already been computed. This can be ensured by computing the elements of C by increasing row indices and, on each row, by decreasing column indices. Let $\text{MIN}(x, y)$ be a simple routine which returns $\min\{x, y\}$. The algorithm follows.

```
COMPUTE(a, b)
n ← length(a)
m ← +∞
for i ← 1 to n − 1 do
    C[i, n − 1] ← a_i
    m ← MIN(m, C[i, n − 1])
for j ← 0 to n − 1 do
    C[0, j] ← b_j
    m ← MIN(m, C[0, j])
for i ← 1 to n − 2 do
    for j ← n − 2 downto i do
        C[i, j] ← C[i − 1, j − 1] · C[i, j + 1]
        {C[i − 1, j − 1] and C[i, j + 1] already computed at this point}
        m ← MIN(m, C[i, j])
return m
```

Point 2 The running time $T(n)$ of the algorithm is determined by the products performed in its two nested loops. We have

$$\begin{aligned} T(n) &= \sum_{i=1}^{n-2} \sum_{j=i}^{n-2} 1 \\ &= \sum_{i=1}^{n-2} n - 1 - i = \sum_{k=1}^{n-2} k \\ &= (n-2)(n-1)/2. \end{aligned}$$

□

Esercizio 4 [8 punti] Si definiscano i problemi decisionali INDEPENDENT_SET e CLIQUE come coppia (ISTANZA, DOMANDA) e si dimostri che INDEPENDENT_SET $<_P$ CLIQUE.

Answer: Recall that given an undirected graph $G = (V, E)$, an *Independent Set* of G is a subset of nodes $V' \subseteq V$ such that no two distinct nodes in V' are connected by an edge in E. In contrast, a *Clique* of G is a subset of nodes $V'' \subseteq V$ such that each pair of distinct nodes in V' is connected by an edge in E. Then, the two decision problems can be defined as follows:

INDEPENDENT SET:
INSTANCE: $\langle G = (V, E), k\rangle$, G undirected graph, $k \in \boldsymbol{N}$
QUESTION: Is there an Independent Set V' of G of size $|V'| = k$?

CLIQUE:
INSTANCE: $\langle G = (V, E), k\rangle$, G undirected graph, $k \in \boldsymbol{N}$
QUESTION: Is there a Clique V'' of G of size $|V''| = k$?

Given $G = (V, E)$, define now $G^c = (V, E^c)$ as the graph with the same node set, and set of edges $E^c = \{\ \{u, v\} : u, v \in V, u \neq v, \{u, v\} \notin E\}$. In order to show that INDEPENDENT SET $<_P$ CLIQUE we resort to the following reduction function:

$$f(\langle G = (V, E), k\rangle) = \langle G^c = (V, E^c), k\rangle$$

The above function is clearly computable in polynomial time. Let $x = \langle G = (V, E), k\rangle$. All we are left to show is that

$$x = \langle G = (V, E), k\rangle \in \text{ INDEPENDENT SET } \Leftrightarrow f(x) = \langle G^c = (V, E^c), k\rangle \in \text{ CLIQUE.}$$

We can easily establish the above proposition as follows:

$$\begin{aligned}
&x = \langle G = (V, E), k\rangle \in \text{ INDEPENDENT SET} \\
&\quad \Leftrightarrow \quad \exists V' \subseteq V, |V'| = k : \forall u, v \in V, u \neq v : (u \in V') \vee (v \in V') \Rightarrow \{u, v\} \notin E \\
&\qquad \text{(setting } V' = V'') \\
&\quad \Leftrightarrow \quad \exists V'' \subseteq V, |V''| = k : \forall u, v \in V, u \neq v : (u \in V'') \vee (v \in V'') \Rightarrow \{u, v\} \in E^c \\
&\Leftrightarrow f(x) = \langle G^c = (V, E^c), k\rangle \in \text{ CLIQUE}
\end{aligned}$$

□

Fondamenti di Informatica II – Ingegneria Informatica
Compito, 14/12/2004 (Durata: 2h30m)

Esercizio 1 [8 punti] Si consideri la seguente equazione di ricorrenza definita per valori del parametro $n = 2^{2^i}$, con $i \geq 0$:

$$T(n) = \begin{cases} 2, & n = 2, \\ \frac{\sqrt{n}}{2} T\left(\sqrt{n}\right), & n > 2. \end{cases}$$

(a) (6 punti) Risolvere la ricorrenza, determinando la forma esplicita di $T(n)$ e **(b) (2 punti)** verificare la soluzione ottenuta usando l'induzione.

Answer: Let $n = 2^{2^i}$ and $0 < k \leq i$. We have:

$$\begin{aligned} T\left(2^{2^i}\right) &= \frac{2^{2^{i-1}}}{2} T\left(2^{2^{i-1}}\right) \\ &= \frac{2^{2^{i-1}}}{2} \frac{2^{2^{i-2}}}{2} T\left(2^{2^{i-2}}\right) \\ &\vdots \\ &= \frac{\left(\prod_{j=1}^{k} 2^{2^{i-j}}\right)}{2^k} T\left(2^{2^{i-k}}\right) \\ &= \frac{2^{\sum_{j=i-k}^{i-1} 2^j}}{2^k} T\left(2^{2^{i-k}}\right) \end{aligned}$$

When $k = i$, we obtain

$$\begin{aligned} T\left(2^{2^i}\right) &= \frac{2^{\sum_{j=0}^{i-1} 2^j}}{2^i} T(2) \\ &= \frac{2^{2^i - 1}}{2^i} \cdot 2 \\ &= \frac{2^{2^i}}{2^i} \end{aligned}$$

Since $n = 2^{2^i}$, we have that $\log n = 2^i$, therefore

$$T(n) = \frac{n}{\log n}.$$

Let us now verify our solution using induction on i such that $n = 2^{2^i}$. When $i = 0$ we have $n = 2$, and $2 = T(2) = (2/\log 2)$. Assume now that the property holds for $i - 1 \geq 0$. Note that if $n = 2^{2^i}$, then $2^{2^{i-1}} = \sqrt{n}$. Therefore we have

$$T(n) = \frac{\sqrt{n}}{2} T\left(\sqrt{n}\right) = \frac{\sqrt{n}}{2} \frac{\sqrt{n}}{\log\left(\sqrt{n}\right)} = \frac{\sqrt{n}}{2} \frac{\sqrt{n}}{\frac{\log n}{2}} = \frac{n}{\log n}$$

□

Esercizio 2 [8 punti] Sia n una potenza di due e siano dati due insiemi $A, B \subseteq \{0, 1, 2, \ldots n-1\}$ rappresentati in ingresso per mezzo dei loro vettori caratteristici (cioè vettori binari $\boldsymbol{a}$ e $\boldsymbol{b} \in \{0,1\}^n$ con $a_i = 1 \Leftrightarrow i \in A$ e $b_j = 1 \Leftrightarrow j \in B$). Si voglia calcolare,

per ogni intero i, $0 \leq i \leq 2n-2$, il numero z_i di coppie distinte $(a,b) \in A \times B$ tali che $a+b=i$.

(a) (5 punti) Si riconduca il problema al calcolo di una opportuna convoluzione lineare.

(b) (3 punti) Si fornisca lo pseudocodice dell'algoritmo CARTESIAN_SUM($\boldsymbol{a}, \boldsymbol{b}$) che risolve il problema in tempo $O(n \log n)$.

Answer:

Part (a) It is sufficient to compute

$$\boldsymbol{z} = (z_0, z_1, \ldots, z_{2n-2}) = \boldsymbol{a} \star \boldsymbol{b},$$

where $\boldsymbol{a} \star \boldsymbol{b}$ denotes the linear convolution of $\boldsymbol{a}$ and $\boldsymbol{b}$. Indeed, by definition of linear convolution we have:

$$\begin{aligned} z_i &= \sum_{\substack{(j,k):j+k=i \\ 0 \leq j,k \leq n-1}} a_j b_k \\ &= |\{(j,k) \in A \times B : j+k=i\}|, \end{aligned}$$

since the only nonzero terms in the summation are all and only those with indices in $A \times B$.

Part (b) The algorithm is as follows:

```
CARTESIAN_SUM(a, b)
n ← length(a)
fa ← FFT(a|0_n)
fb ← FFT(b|0_n))
for i ← 0 to 2n − 1 do
    zf_i ← fa_i · fb_i
z ← INV_FFT(zf)
return (z_0, z_1, ..., z_{2n−2})
```

The above algorithm executes in $O(n \log n)$ time if we use the FFT algorithm to compute the Discrete Fourier Transform and its inverse. □

Esercizio 3 [8 punti] Dato il seguente codice di programmazione dinamica

```
DP_SUM(n)
for i ← 0 to n do
    A[i, i] ← i
    A[0, i] ← 1
for ℓ ← 2 to n do
    for i ← 1 to n − ℓ + 1 do
        j ← i + ℓ − 1
        A[i, j] ← A[i − 1, j − 1] + A[i, j − 1] + A[i + 1, j]
return A[1, n]
```

(a) (5 punti) si fornisca un codice memoizzato equivalente e **(b) (3 punti)** se ne analizzi la complessità.

Answer: The given code computes the following recurrence:

$$A(i,j) = \begin{cases} 1 & i = 0, 0 \leq j \leq n \\ j & 1 \leq i = j \leq n \\ A(i-1,j-1) + A(i,j-1) + A(i+1,j) & 1 \leq i < j \leq n \end{cases}$$

The memoized code to compute this recurrence is given by the following pair of routines:

```
INIT_A(n)
if n = 1 then return 1
for j ← 0 to n do
    A[j, j] ← j
    A[0, j] ← 1
for i ← 1 to n do
    for j ← i + 1 to n do
        A[i, j] ← 0 { note that A(i, j) is always positive}
return REC_A(1, n)

REC_A(i, j)
if A[i, j] = 0
    then A[i, j] ← REC_A(i − 1, j − 1) + REC_A(i, j − 1) + REC_A(i + 1, j)
return A[i, j]
```

To obtain $A(1,n)$, we call INIT_A(n). Note that in REC_A, each entry is computed exactly once for every pair (i,j), with $1 \leq i < j \leq n$, hence INIT_A(n) triggers $\Theta(n^2)$ recursive calls to REC_A. Since each call performs constant work, the running time of the memoized code is $\Theta(n^2)$, which is also the running time of the corresponding dynamic programming code. □

Esercizio 4 [8 punti] Si consideri il seguente problema decisionale:

NCBC (**N**ON **C**ONSTANT **B**OOLEAN **C**IRCUIT):
ISTANZA: Circuito Booleano C a n ingressi $x_1, x_2, \ldots, x_n$.
DOMANDA: C realizza una funzione *non costante* ($C \not\equiv 0$ e $C \not\equiv 1$)?

Dimostrare che NCBC è NP-Hard.
(*Suggerimento:* Si riduca da BOOLEAN CIRCUIT SATISFIABILITY (BC-SAT))

Answer: We show that BC-SAT $<_P$ NCBC using the following reduction function f: given a circuit C, $f(\langle C \rangle)$ produces the encoding $\langle C' \rangle$ of a circuit C' obtained from C by introducing a new input x_{n+1}, and an AND gate whose inputs are the (unique) output of C and x_{n+1}. The output of $f(C)$ is the output of this AND gate. Note that f is trivially computable in time polynomial in $| < C > |$.

Let us show that f reduces BCSAT to NCBC. If $C \in$ BCSAT, then there exists a satisfying assignment $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ such that $C(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) = 1$. Then $C'(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n, 1) = 1$ and $C'(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n, 0) = 0$, therefore $f(\langle C \rangle) \in$ NCBC. Vice versa, if $C \notin$ BCSAT, then for any assignment $(x_1, x_2, \ldots, x_n)$, it is $C(x_1, x_2, \ldots, x_n) = 0$, but then $C'(x_1, x_2, \ldots, x_n, x_{n+1}) = (0 \text{ AND } x_{n+1}) = 0$ regardless of the value of x_{n+1}. Therefore C' is a constant circuit ($C' \equiv 0$), hence $f(\langle C \rangle) \notin$ NCBC. □

Fondamenti di Informatica II – Ingegneria Informatica
Compito, 8/9/2005 (Durata: 3h)

Esercizio 1 [8 punti] Utilizzando l'induzione parametrica, si individui la minima costante $c > 0$ per cui risulti $T(n) \leq cn^2$, per $n \geq 1$, dove

$$T(n) = \begin{cases} 1, & 1 \leq n \leq 8 \\ 8T(\lfloor n/8 \rfloor) + 2T(\lfloor n/4 \rfloor) + (3n^2/4), & n > 8 \end{cases}$$

Answer: From the (parametric) induction base, we obtain $c \geq 1$. When going from inductive hypothesis to thesis, we get:

$$\begin{aligned} T(n) &\leq 8c(n^2/64) + 2c(n^2/16) + (3n^2/4) \\ &= c(n^2/4) + (3n^2/4) \boxed{\leq} cn^2 \\ &\Leftrightarrow c(3n^2/4) \geq (3n^2/4) \\ &\Leftrightarrow c \geq 1. \end{aligned}$$

Hence $c = \max\{1, 1\} = 1$. □

Esercizio 2 [8 punti] Per n potenza di due, sia $\boldsymbol{x}$ un vettore di n incognite complesse e $\boldsymbol{a} \in \boldsymbol{C}^n$ un vettore noto. Per un fissato valore $m \geq 2$, si consideri il sistema di equazioni espresso in forma vettoriale come segue:

$$\underbrace{\boldsymbol{x} \circledast \boldsymbol{x} \circledast \cdots \circledast \boldsymbol{x}}_{m \text{ volte}} = \boldsymbol{a},$$

dove $\circledast$ denota l'operatore di convoluzione ciclica. Si fornisca lo pseudocodice e si analizzi un algoritmo NUM_SOL($\boldsymbol{a}, m$) che, dati in ingresso $\boldsymbol{a}$ e m, ritorni il numero di soluzioni vettoriali distinte del sistema effettuando $O(n \log n)$ operazioni tra scalari complessi.

Answer: Let us transform the system by applying the DFT_n operator to both its sides. Let $\boldsymbol{X} = DFT_n(\boldsymbol{x})$ and $\boldsymbol{A} = DFT_n(\boldsymbol{a})$. By the convolution theorem we obtain the equivalent system:

$$\underbrace{\boldsymbol{X} \odot \boldsymbol{X} \odot \cdots \odot \boldsymbol{X}}_{m \text{ times}} = \boldsymbol{A},$$

where operator $\odot$ denotes component-wise product. The i-th scalar equation of the above system is:

$$X_i^m = A_i, \text{ for } 0 \leq i \leq n-1,$$

which is a simple m-th degree equation in the complex field in the isolated unknown X_i. Such equation admits either m *distinct* solutions if $A_i \neq 0$, or the single solution $X_i = 0$ if $A_i = 0$. We can assemble one scalar solution to each equation arbitrarily to obtain a vector solution for the transformed system. In turn, each distinct vector solution for the transformed system yields a distinct solution to the original system (since DFT_n is a bijection). In summary, let nz be the number of nonzero entries of $\boldsymbol{A} = DFT_n(\boldsymbol{a})$. Then the number of distinct solutions to the system is $s = m^{nz}$. The algorithm follows.

```
NUM_SOL(a, m)
n ← length(a)
A ← FFT(a)
s ← 1
for i ← 0 to n − 1 do
    if A_i ≠ 0
        then s ← m · s
return s
```

The correctness of the above algorithm follows from the previous observations. Its running time $T(n)$ is dominated by the call to FFT, hence $T(n) = \Theta(n \log n)$. □

Esercizio 3 [8 punti] Sia $n > 0$ e sia $\boldsymbol{x}$ un dato vettore di ingresso a n componenti intere. Si supponga che una certa strategia di programmazione dinamica conduca alla seguente ricorrenza, definita in funzione di $\boldsymbol{x}$ per tutti i valori di i e j con $1 \leq i, j \leq n$:

$$w(i,j) = \begin{cases} x_j & i = 1, 1 \leq j \leq n, \\ \sum_{r=1}^{i-1} w(r,j) + \sum_{s=j+1}^{n} w(i,s), & i \neq 1, 1 \leq j \leq n \end{cases}$$

Si fornisca un algoritmo iterativo *bottom-up* che, dato in ingresso $\boldsymbol{x}$, restituisca in una tabella W i valori $w(i,j)$, per $1 \leq i, j \leq n$. Si analizzi la correttezza e la complessità dell'algoritmo in funzione del numero di somme effettuate. Per ottenere il massimo punteggio basta un algoritmo corretto di complessità $O(n^3)$. Punti aggiuntivi verranno assegnati ad algoritmi corretti di complessità $O(n^2)$.

Answer: Let us first begin with a simple $O(n^3)$ algorithm. Consider a table $W[i,j]$ which stores the value of $w(i,j)$, for $1 \leq i \leq j \leq n$. From the definition, it is clear that we can compute $W[i,j]$ if the table entries $W[r,j]$, for $1 \leq r < i$ and $W[i,s]$, for $j < s \leq n$ have already been computed. Observe that the needed values are either on the same column j as $w(i,j)$ but on previous rows, or on the same row but on columns of higher index $s > j$. Therefore, a reverse column-major scan (decreasing column index/increasing row index) will correctly compute all values. The code immediately follows.

```
COMPUTE_W(x)
n ← length(x)
for j ← n downto 1 do
    { decreasing column index first ...}
    W[1, j] ← x_j
    for i ← 2 to n do
        { increasing row index second ...}
        W[i, j] ← 0
        for r ← 1 to i − 1 do
            W[i, j] ← W[i, j] + W[r, j]
        for s ← j + 1 to n do
            W[i, j] ← W[i, j] + W[i, s]
return W
```

The correctness of the code follows from the fact that the values $W[r,j]$ and $W[i,s]$ have already been (correctly) computed when used. The running time of the above

algorithm is evaluated as follows:

$$\begin{aligned} T_{\text{COMP}}(n) &= \sum_{j=1}^{n}\sum_{i=2}^{n}[(i-1)+(n-j)] \\ &= \sum_{j=1}^{n}[(n-1)n/2+(n-1)(n-j)] \\ &= (n-1)n^2/2+(n-1)\sum_{k=1}^{n-1}k \\ &= (n-1)n^2/2+(n-1)^2n/2 = \Theta(n^3). \end{aligned}$$

A much more efficient algorithm can be obtained by first observing that $w(i,n) = 2^{i-2}x_n$, for $2 \leq i \leq n$ and that for $i > 1$ and $j < n$, each $w(i,j)$ is the sum of two terms. By keeping these two terms separate, storing them into two distinct tables, we can reduce the time to compute a single entry of W to a constant. Specifically, define the following two tables:

$$Up[i,j] = \begin{cases} 0 & i = 1, 1 \leq j < n, \\ \sum_{r=1}^{i-1} W[r,j] & i > 1, 1 \leq j < n; \end{cases}$$

and

$$Right[i,j] = \begin{cases} 0 & j = n, 2 \leq i \leq n, \\ \sum_{s=j+1}^{n} W[i,s], & j < n, 2 \leq i \leq n; \end{cases}$$

Note that from the definition of $w(i,j)$, it immediately follows that for $i > 1$ and $j < n$, $W[i,j] = Up[i,j] + Right[i,j]$. Moreover, the following recurrences hold:

$$Up[i,j] = \begin{cases} 0 & i = 1, 1 \leq j \leq n, \\ Up[i-1,j] + W[i-1,j] & i \neq 1, 1 \leq j \leq n; \end{cases}$$

and

$$Right[i,j] = \begin{cases} 0 & j = n, 1 \leq i \leq n, \\ Right[i,j+1] + W[i,j+1] & j \neq n, 1 < i \leq n. \end{cases}$$

Note that in order to compute the values at coordinates (i,j) for tables W, Up and $Right$, we only need values at coordinates $(i-1,j)$ and $(i,j+1)$. Hence we can keep the same type of scan used in the previous algorithm. We obtain the following code:

```
COMPUTE2_W(x)
n ← length(x)
{ we need to treat the last column separately}
W[1,n] ← W[2,n] ← x_n
Right[1,n] ← Right[2,n] ← 0
for i ← 3 to n do
        W[i,n] ← 2 · W[i−1,n]
        Right[i,n] ← 0
for j ← n−1 downto 1 do
    { decreasing column index first ...}
    Up[1,j] ← 0
    W[1,j] ← x_j
    for i ← 2 to n do
        { increasing row index second ...}
        Up[i,j] ← Up[i−1,j] + W[i−1,j]
        Right[i,j] ← Right[i,j+1] + W[i,j+1]
        W[i,j] ← Up[i,j] + Right[i,j]
return W
```

As before, all values needed in an iteration have been computed previously, hence the code is correct. The new algorithm performs a constant number of operations for each entry of W to be computed, hence its running time is $\Theta(n^2)$. □

Esercizio 4 [8 punti] Siano L_1 e L_2 due linguaggi NP-Hard e si supponga che esista una funzione di riduzione f da SAT a L_1 calcolabile in tempo polinomiale tale che, per ogni stringa $x \in \{0,1\}^\star$, si abbia inoltre che $f(x) \notin L_2$. Si dimostri che allora il linguaggio $L_1 \cup L_2$ è anch'esso NP-Hard.

Answer: It is sufficient to prove that under the stated hypotheses, f also reduces SAT to $L_1 \cup L_2$. Indeed, if $x \in$ SAT, then $f(x) \in L_1$, since f reduces SAT to L_1, hence $f(x) \in L_1 \cup L_2$. Vice versa, if $f(x) \in L_1 \cup L_2$, it must be that $f(x) \in L_1$ (since $f(x) \notin L_2$ from the hypothesis), hence $x \in$ SAT (again, since f reduces SAT to L_1). □