ESERCIZI SULLA TECNICA *Greedy*

1. **[FILE]** Si supponga di avere n files di lunghezze $l_1, \ldots, l_n$ (interi positivi) che bisogna memorizzare su un disco di capacità data D. Si assuma che la somma delle lunghezze di questi files ecceda la capacità del disco. Il problema sta nel selezionare un sottoinsieme degli n files che abbia cardinalità massima e che possa essere memorizzato sul disco. Descrivere un algoritmo *greedy* che risolve il problema, provarne la correttezza e valutare la complessità di una sua efficiente implementazione.

2. **[TAPPE]** Si supponga di dover effettuare un viaggio dalla località A alla località B con un'auto che ha un'autonomia di k chilometri. Lungo il percorso, a partire da A sono presenti n distributori di benzina ciascuno distante dal precedente meno di k chilometri e l'ultimo dista meno di k chilometri da B. Sia d_i la distanza che separa il distributore i dal distributore $i+1$ per $i = 1, 2, \ldots n-1$ e sia d_n la distanza da B dell'ultimo distributore. Descrivere un algoritmo greedy che seleziona un numero minimo di distributori in cui far tappa durante il viaggio. Descrivere un algoritmo greedy che risolve il problema, provarne la correttezza e valutare la complessità di una sua efficiente implementazione.

3. **[INSIEME UNIVOCO]** In un grafo orientato G un sottoinsieme A degli archi è detto **univoco** se non contiene 2 o più archi uscenti dallo stesso nodo. Descrivere un algoritmo *greedy* che preso in input un grafo G orientato con pesi positivi sugli archi, trovi un insieme univoco A di archi di G di peso massimo. Provare la correttezza dell'algoritmo proposto e valutare la complessità di una sua efficiente implementazione.

4. **[INTERVALLI]** Dato un vettore V di n numeri reali, bisogna calcolare il numero minimo di intervalli di lunghezza unitaria in grado di ricoprire tutti i valori di V. Ad esempio se $n = 5$ e $V[1] = 2.5, V[2] = 3.8, V[3] = 1.5, V[4] = 3.1, V[5] = 1.8$, allora il numero minimo di intervalli unitari è 2 e una possibile soluzione è $\{[1.5, 2.5], [3, 4]\}$. Descrivere un algoritmo greedy che risolve il problema, provarne la correttezza e valutare la complessità di una sua efficiente implementazione.

5. **[AULE]** Si assuma di dover assegnare aule in cui tenere n lezioni e che ciascuna lezione i è caratterizzata da un tempo di inizio s_i ed un tempo di fine f_i (dove naturalmente si ha $s_i < f_i$). Tenendo conto che in una stessa aula non possono tenersi più lezioni contemporaneamente, si trovi un algoritmo *greedy* per determinare un assegnazione delle lezioni alle aule che minimizzi il numero di aule utilizzate. Provare la correttezza dell'algoritmo proposto e valutare la complessità di una sua efficiente implementazione.

6. **[UTILIZZAZIONE AULA]** Date $n$attività (lezioni, seminari, ecc.) ognuna delle quali caratterizzata da un tempo di inizio s_i ed un tempo di fine f_i (dove naturalmente si ha $s_i < f_i$), si vuole trovare un insieme di attività che possono essere svolte in un unica aula senza sovrapposizioni e che massimizzi il tempo totale di utilizzo dell'aula. Proporre un algoritmo *greedy* per questo problema. Discutere la correttezza dell'algoritmo: se non risolve il problema valutarne il rapporto d'approssimazione.

7. **[RESTO]** Consideriamo il problema di dare un resto di n euro con il minor numero possibile di monete.

   a. Si descriva un algoritmo *greedy* che dia un resto di n euro usando monete del valore di $0, 100, 501$ e 2 euro. Provare la correttezza dell'algoritmo proposto e valutare la complessità di una sua efficiente implementazione.

   b. Si generalizzi l'algoritmo del punto precedente al caso in cui i valori di monete disponibili siano $c_1, c_2, \ldots, c_k$ tali che c_i divide c_{i+1} per $1 \leq i < k$. Provare la correttezza dell'algoritmo proposto e valutare la complessità di una sua efficiente implementazione.

   c. Si mostri un insieme di valori di monete per cui l'algoritmo proposto al punto precedente non è corretto.

8. **[COMPITI UNITARI]** Si considerino n compiti ciascuno rappresentato dalla coppia (p_i, d_i) con $p_i \geq 0$ e $d_i \geq 1$, dove p_i è il guadagno che si ottiene qualora il compito i sia svolto entro la scadenza d_i. Assumendo che i compiti richiedono tempo unitario e che sia disponibile una macchina in grado di eseguirne uno per volta a partire dal tempo 0:

   a. Provare che un sottoinsieme degli n compiti può essere eseguito dalla macchina rispettando tutte le scadenze se e solo se, ordinandone gli elementi per tempo di scadenza, l'h-esimo elemento ha un tempo di scadenza non inferiore ad h.

   b. Descrivere un algoritmo *greedy* che produce un sottoinsieme dei compiti che è possibile eseguire sulla macchina rispettando le scadenze e che massimizza il guadagno totale.

   c. Provare la correttezza dell'algoritmo proposto.

   d. Proporre un'implementazione efficiente dell'algoritmo e valutarne la complessità.

9. **[ZAINO A VARIABILI FRAZIONARIE]** Si considerino n oggetti ed uno zaino. L'oggetto i ha peso w_i e valore v_i mentre la capacità dello zaino è C. Se nello zaino viene inserita una frazione x_i dell'oggetto i, ove $0 \leq x_i \leq 1$, allora si ottiene un profitto $v_i \cdot x_i$ e la capacità dello zaino diminuisce di $w_i \cdot x_i$. Si desidera riempire lo zaino massimizzando il profitto totale.

   a. Descrivere un algoritmo *greedy* che risolva il problema.

   b. Provare la correttezza dell'algoritmo proposto e valutarne la complessità (la complessità non dovrebbe superare $O(n \log n)$).

10. **[SOTTOSEQUENZA ZIG-ZAG]** Una sequenza di interi $X = x_1 x_2 \ldots x_m$ si definisce ZIG-ZAG se, per $1 \leq i \leq m-1$,

$$x_i < x_{i+1} \quad \text{se } i \text{ dispari}$$

$$x_i > x_{i+1} \quad \text{se } i \text{ pari}$$

    Ad esempio $X = (3, 8, 1, 5, 2)$ è una sequenza ZIG-ZAG, mentre $X = (3, 8, 10, 5, 2)$ non lo è. Descrivere ed analizzare un algoritmo *greedy* che data una sequenza $Y = y_1 y_2 \ldots y_n$ determini la lunghezza della più lunga sottosequenza ZIG-ZAG di Y. Ad esempio, se $Y = (3, 4, 8, 8, 5, 6, 2)$ allora la lunghezza massima è 5 (ossia la sottosequenza è $3, 8, 5, 6, 2$ o anche $4, 8, 5, 6, 2$). Provare la correttezza dell'algoritmo proposto e valutare la complessità di una sua efficiente implementazione.

11. **[ASSEGNAMENTO DI LAVORI]** Bisogna eseguire n lavori, si dispone di m operai, $m \geq n$, e di una matrice M di $n \times m$ interi dove $M[i, j]$ indica il tempo necessario ad eseguire l'i-esimo

lavoro quando alla sua esecuzione sono assegnati j operai. Una distribuzione degli m operai agli n lavori è una sequenza $j_1, j_2, \dots j_n$ di n interi positivi con $j_1 + j_2 + \dots + j_n = m$ e determina un tempo di terminazione per l'esecuzione dei lavori pari a $\max\{M[1, j_1], M[2, j_2], \dots, M[n, j_n]\}$. Il tempo di terminazione per l'esecuzione degli n lavori dipende ovviamente dalla distribuzione che si fa degli operai ai lavori. Si vuole determinare il minimo tempo di terminazione possibile per l'esecuzione degli n lavori.

Ad esempio: dato $n = 3$, $m = 6$ ed $M = \begin{pmatrix} 14 & 10 & 9 & 8 & 4 & 1 \\ 20 & 15 & 8 & 6 & 5 & 3 \\ 30 & 28 & 27 & 25 & 20 & 15 \end{pmatrix}$
la distribuzione $4, 1, 1$ determina un tempo di terminazione pari a 30 mentre la distribuzione $2, 2, 2$ determina un tempo di terminazione pari a 28. Il tempo di terminazione minimo è 25 ed è determinato dalla distribuzione $1, 1, 4$.

(a) Si descriva un algoritmo greedy che calcola una distribuzione con tempo minimo di terminazione. e provarne la correttezza. La complessit dell'algoritmo deve essere $O(mn)$.

(b) Risolvere il problema con un algoritmo greedy a complessit $O(m \log n)$.

12. **[PROPRIETÀ DELL'ALBERO MINIMO DI COPERTURA]** Sia $G = (V, E)$ un grafo non orientato connesso e con pesi sugli archi. Sia T un minimo albero di copertura per G.

a. Provare che T è ancora un minimo albero di copertura anche per il grafo che si ottiene da G incrementando di una stessa costante c il peso degli archi.

b. Provare che T deve contenere un arco di peso minimo.

c. Provare che qualora i pesi di G siano tutti distinti, allora T è l'unico minimo albero di copertura. Mostrare che che la condizione sul fatto che i pesi siano tutti distinti non è necessaria per l'unicità del minimo albero di copertura.

13. **[MASSIMO ALBERO DI COPERTURA]** Si consideri l'algoritmo di Prim modificato in modo che venga scelto ogni volta l'arco di peso massimo anziché quello di peso minimo. Provare che:

a. L'algoritmo trova l'albero di copertura di peso massimo.

b. Se il peso di un albero è dato dal prodotto dei pesi (anziché dalla somma) l'algoritmo trova l'albero di copertura di peso massimo ma solo nell'ipotesi in cui i pesi del grafo sono tutti positivi.

14. **[ALBERO DI COPERTURA]** Si consideri il seguente problema. Dato un grafo non diretto e connesso $G = (V, E)$ trovare un albero di copertura di G. Si propone il seguente algoritmo:

INPUT un grafo non orientato e connesso $G = (V, E)$
    $SOL \leftarrow \emptyset$
    $R \leftarrow \{s\}$ dove s è un qualsiasi vertice di V
    $A \leftarrow E$
    WHILE $A \neq \emptyset$ DO
        estrai un arco e da A

IF $e = \{u, v\}$ con $u \in R$ e $v \notin R$ THEN
$SOL \leftarrow SOL \cup \{\{u, v\}\}$
$R \leftarrow R \cup \{v\}$
ENDIF
ENDWHILE
OUTPUT SOL

Dire se l' algoritmo proposto risolve il problema e in caso affermativo spiegare perché l'algoritmo è corretto (meglio ancora dimostrarne la correttezza) mentre in caso negativo fornire un controesempio.

15. [**MINIMO ALBERO DI COPERTURA CON PESI LIMITATI**] Si deve calcolare il minimo albero di copertura per grafi non orientati e connessi $G = (V, E)$ con pesi sugli archi appartenenti all'insieme $\{1, 2, \ldots, k\}$, dove k è un intero fissato. Implementare l'algoritmo di Prim sfruttando la particolarità di questi grafi in modo che il minimo albero di copertura venga trovato in tempo $O(k|V| + |E|)$.

16. [**SOTTOGRAFO k-COPRENTE**] Si consideri il seguente problema: dato un grafo $G = (V, E)$ non orientato e connesso con pesi non negativi sugli archi, un suo nodo s ed un intero k, trovare il sottografo di peso minimo di G che contiene s ed ha esattamente k nodi. Quando k è uguale al numero di nodi di G, il problema diventa equivalente a trovare il minimo albero di copertura di G. Così, per risolvere il problema, in analogia con l'algoritmo di Prim, viene proposto il seguente algoritmo:

INPUT un grafo non orientato e connesso $G = (V, E)$ con archi pesati con pesi non negativi, un suo nodo s ed un intero k
$SOL \leftarrow \emptyset$
$A \leftarrow \{s\}$
WHILE $|A| < k$ DO
sia $\{a, b\}$ l'arco di peso minimo tra quelli incidenti tra nodi $a \in A$ e $b \notin A$
$SOL \leftarrow SOL \cup \{ \{a, b\} \}$
$A \leftarrow A \cup \{b\}$
ENDWHILE
OUTPUT SOL.

a. Provare che l'algoritmo produce un'albero di k nodi.

b. Descrivere un'implementazione dell'algoritmo che abbia complessità $O(k \cdot |V|)$.

c. Provare che l'algoritmo non è corretto.

d. Dire se l'algoritmo ha un rapporto d'approssimazione limitato da 5.

17. [**ALBERO CROMATICO Prim**] Si consideri il seguente problema: dato un grafo $G = (V, E)$ non orientato e connesso con archi colorati, trovare un albero di copertura per G tale che il numero di colori distinti che compaiono sui suoi archi sia minimo. Per risolvere il problema si propone il seguente algoritmo greedy (ispirato dall'algoritmo di Prim):

INPUT un grafo non orientato e connesso $G = (V, E)$ con archi colorati
$SOL \leftarrow \emptyset$

$R \leftarrow \{s\}$, dove s è un qualsiasi vertice in V
WHILE $R \neq V$ DO
IF c' è un arco $\{a,b\}$ con $a \in R$ e $b \notin R$ con un colore di un arco in SOL THEN
$SOL \leftarrow SOL \cup \{\{a,b\}\}$
ELSE
$SOL \leftarrow SOL \cup \{\{a,b\}\}$ dove $\{a,b\}$ è un qualsiasi arco con $a \in R$ e $b \notin R$
ENDIF
$R \leftarrow R \cup \{b\}$
ENDWHILE
OUTPUT SOL.

a. Provare che l'algoritmo produce un'albero di copertura.

b. Descrivere un'implementazione dell'algoritmo che abbia complessità $O(|V|^2)$.

c. Provare che l'algoritmo non è corretto.

d. Dire se l'algoritmo ha un rapporto d'approssimazione limitato da 5.

18. [**ALBERO CROMATICO Kruskal**] Si consideri il seguente problema: dato un grafo $G = (V, E)$ non orientato e connesso con archi colorati, trovare un albero di copertura per G tale che il numero di colori distinti che compaiono sui suoi archi sia minimo. Per risolvere il problema si propone il seguente algoritmo (ispirato dall'algoritmo di Kruskal):

INPUT un grafo non orientato e connesso $G = (V, E)$ con archi colorati
$SOL \leftarrow \emptyset$
$A \leftarrow E$
WHILE $A \neq \emptyset$ DO
sia c il colore maggioritario in A
WHILE in A ci sono archi colorati c DO
estrai da A un arco $\{u,v\}$ colorato c
IF $\{u,v\}$ non forma cicli con archi in SOL THEN $SOL \leftarrow SOL \cup \{u,v\}$
ENDWHILE
ENDWHILE
OUTPUT SOL.

a. Provare che l'algoritmo produce un'albero di copertura.

b. Descrivere un'implementazione dell'algoritmo che abbia complessità $O(|E| \log |E|)$.

c. Provare che l'algoritmo non è corretto.

d. Dire se l'algoritmo ha un rapporto d'approssimazione limitato da 5.

19. [**ALBERO CROMATICO EQUILIBRATO**] Si consideri il seguente problema. Dato un grafo non diretto e connesso G con gli archi colorati trovare un albero di copertura di G che usa il minor numero di colori. Il problema in generale è difficile ma se ci si restringe a grafi *equilibrati* esiste un semplice ed efficiente algoritmo che lo rispolve. Un grafo è equilibrato se per ogni ciclo C e per ogni arco e del ciclo C esiste almeno un altro arco in C con lo stesso colore di e.

a. Descrivere un algoritmo che risolve il problema per grafi equilibrati in $\mathrm{O}(|V|^2)$.

b. Dimostrare che l'algoritmo proposto è corretto (per grafi equilibrati.

20. **[NUMERO DI CAMMINI MINIMI]** Come è noto, dato un grafo orientato G con pesi positivi sugli archi ed un nodo s di G, l'algoritmo di Dijkstra calcola l'albero dei cammini minimi da s ad un qualsiasi altro nodo di G che è raggiungibile da s. In generale, tra il nodo s e un altro nodo u di G può esserci più di un cammino minimo.

    a. Descrivere un algoritmo che calcoli per ogni nodo u il numero di tutti i possibili cammini di peso minimo da s a u. Suggerimento: modificare l'algoritmo di Dijkstra.

    b. Discutere la complessità dell'algoritmo.

21. **[CAMMINI MINIMI PER PESI TRASLATI]** Sia $G = (V, E)$ un qualsiasi grafo orientato con pesi sugli archi, pesi che possono essere anche negativi. Modifichiamo i pesi di G sommando ad essi un intero fissato M abbastanza grande da renderli tutti positivi. Al grafo che si ottiene G' (che ha pesi positivi) applichiamo l'algoritmo di Dijkstra. I cammini minimi che vengono così calcolati sono anche cammini minimi per il grafo originale G? Motivare la risposta.

22. **[CAMMINI MINIMI CON PESI LIMITATI]** Si devono calcolare le distanze minime da un nodo sorgente s per grafi non orientati e connessi $G = (V, E)$ con pesi sugli archi appartenenti all'insieme $\{1, 2, \ldots, k\}$, dove k è un intero fissato. Implementare l'algoritmo di Dijkstra sfruttando la particolarità di questi grafi in modo che le distanze minime vengano calcolate in tempo $O(k|V| + |E|)$.

23. **[CAMMINI COLORATI]** Si descriva un algoritmo che determini, in un grafo colorato con vertici rossi e neri, se esiste un cammino dal nodo i al nodo j contenente al più k vertici neri interni al cammino (ossia esclusi i e j). Valutare la complessità dell'algoritmo.

24. **[INSIEME INDIPENDENTE]** Dato un grafo $G = (V, E)$ non orientato chiamiamo *insieme indipendente* di G un sottoinsieme dei nodi di G in cui non ci sono due nodi adiacenti. Si assuma che ad ogni vertice del grafo è associato un valore intero positivo. Si desidera trovare un insieme indipendente di G di valore massimo (cioè, la cui somma dei valori dei vertici sia massima). Per risolvere il problema viene proposto il seguente algoritmo greedy:

    INPUT un grafo non orientato $G = (V, E)$ con nodi pesati con pesi positivi
        $SOL \leftarrow \emptyset$
        WHILE $V \neq \emptyset$ DO
            estrai da V un nodo u di valore massimo
            IF nessuno dei nodi adiacenti a u è in SOL THEN $SOL \leftarrow SOL \cup \{u\}$
        ENDWHILE
    OUTPUT SOL.

    a. Provare che l'algoritmo produce un insieme indipendente.

    b. Descrivere un'implementazione dell'algoritmo che abbia complessità $O(|V| \log |V| + |E|)$.

    c. Provare che l'algoritmo non è corretto.

    d. Dire se l'algoritmo ha un rapporto d'approssimazione limitato da 5.

25. **[TRIANGOLO]** Dato un grafo $G = (V, E)$ non orientato chiamiamo *triangolo* un insieme di tre nodi distinti che sono mutuamente adiacenti. Si assuma che ad ogni vertice del grafo G è associato

un costo intero positivo e che nel grafo siano presenti triangoli. Si desidera trovare un triangolo di costo minimo (il costo di un triangolo è dato dal costo dei tre nodi che lo compongono). Per risolvere il problema viene proposto il seguente algoritmo greedy:

INPUT un grafo non orientato $G = (V, E)$ con nodi pesati e che contiene triangoli
  $SOL \leftarrow \emptyset$
  WHILE $V \neq \emptyset$ AND $SOL = \emptyset$ DO
    estrai da V un vertice u di costo minimo
    IF esiste almeno un triangolo incidente in u THEN
      $SOL \leftarrow \{u, a, b\}$ dove $\{u, a, b\}$ è il triangolo di costo minimo incidente in u
    ENDIF
  ENDWHILE
OUTPUT SOL.

a. Provare che l'algoritmo produce un triangolo del grafo.

b. Descrivere un'implementazione dell'algoritmo che abbia complessità $O(|V| \log |V| + |V| \cdot |E|)$.

c. Provare che l'algoritmo non è corretto.

d. Dire se l'algoritmo ha un rapporto d'approssimazione limitato da 7.

26. **[COLORAZIONE]** Si consideri il seguente problema COLORAZIONE: Dato un grafo non diretto $G = (\{1, 2, \ldots, n\}, E)$, assegnare ad ogni nodo un valore in $\{1, 2, \ldots, n\}$ in modo tale che nodi adiacenti ricevono valori distinti e che il numero di valori distinti usati sia minimo. Una numerazione è ammissibile se nodi adiacenti ricevono numeri distinti. Viene proposto il seguente algoritmo greedy:

INPUT un grafo non orientato $G = (\{1, 2, \ldots, n\}, E)$
  FOR $i = 1$ TO n DO
    $C \leftarrow \{1, 2, \ldots, n\}$
    FOR $j = 1$ TO $i - 1$ DO
      IF $\{i, j\} \in E$ THEN $C \leftarrow C - \{SOL[j]\}$
    ENDFOR
    $SOL[i] \leftarrow \min\{c | \ c \text{ è in } C\}$
  ENDFOR
OUTPUT SOL.

Dire se l'algoritmo risolve il problema. In caso affermativo dare una dimostrazione di correttezza. In caso negativo esibire un controesempio.

27. **[PARTIZIONE EQUILIBRATA]** Dato un insieme A di $2n$ interi $x_1, x_2, \ldots x_{2n}$ si vuole trovare un sottoinsieme SOL di A con n interi tale che $\left|\sum_{x \in SOL} x - \sum_{x \notin SOL} x\right|$ sia minimo. Si propone il seguente algoritmo *greedy*:

INPUT Un insieme $A = \{x_1, x_2, \ldots x_{2n}\}$ con $2n$ interi
  calcola la somma M dei $2n$ interi in A
  $SOL \leftarrow \emptyset$
  $val \leftarrow 0$
  $B \leftarrow A$

WHILE $|SOL| < n$ DO
Sia x l'intero in B per cui risulta minimo il valore $|val + x - \frac{M}{2}|$
$SOL \leftarrow SOL \cup \{x\}$
$B \leftarrow B - \{x\}$
ENDWHILE
OUTPUT SOL.

Provare che l'algoritmo proposto non risolve il problema.

Viene allora proposto il seguente raffinamento :

INPUT Un insieme $A = \{x_1, x_2, \dots x_{2n}\}$ con $2n$ interi
calcola la somma M dei $2n$ interi in A
$t \leftarrow +\infty$
FOR $i = 1$ TO $2n$ DO
$SOL \leftarrow \{x_i\}$
$val \leftarrow x_i$
$B \leftarrow A - \{x_i\}$
WHILE $|SOL| < n$ DO
Sia x l'intero in B per cui risulta minimo il valore $|val + x - \frac{M}{2}|$
$SOL \leftarrow SOL \cup \{x\}$
$B \leftarrow B - \{x\}$
ENDWHILE
IF $|val - \frac{m}{2}| < |t - \frac{m}{2}|$ THEN
$SOL^* \leftarrow SOL$
$t \leftarrow val$
ENDIF
ENDFOR
OUTPUT SOL^*.

Provare la correttezza dell'algoritmo o in alternativa produrre un controesempio.

28. **[INSCATOLAMENTO MASSIMO]** Si supponga di avere un contenitore di capacità C ed n oggetti di peso $p_1, p_2, \dots, p_n$, con $1 \leq p_i \leq C$ e $\sum_{i=1}^{n} p_i \geq C$. Il contenitore può contenere un qualsiasi sottoinsieme di oggetti il cui peso non eccede C. Fra tutti i sottoinsiemi di oggetti che possono andare nel contenitore si vuole quello di peso totale massimo. Per risolvere il problema si propone il seguente algoritmo.

INPUT i pesi $p_1, p_2, \dots, p_n$ degli oggetti e la capacità C del contenitore
$SOL \leftarrow \emptyset$
$peso \leftarrow 0$
$A \leftarrow \{1, 2, \dots n\}$
WHILE $A \neq \emptyset$ DO
sia x l'oggetto di peso minimo in A
IF $peso + p_x \leq C$ THEN
$SOL \leftarrow SOL \cup \{x\}$

$peso \leftarrow peso + p_x$
ENDIF
$A \leftarrow A - \{x\}$
ENDWHILE
OUTPUT SOL

a. Provare che l'algoritmo non risolve il problema.

b. Provare che l'algoritmo ha un rapporto d'approssimazione limitato da 2.

c. Provare che l'algoritmo non ha un rapporto d'approssimazione inferiore a 2.

29. [**SOTTOSEQUENZA COMUNE**] Si consideri il seguente problema. Date due sequenze $X = (x_1, x_2, \ldots, x_n)$ e $Y = (y_1, y_2, \ldots, y_m)$ sull'alfabeto $\{0, 1\}$, determinare la lunghezza della più lunga sottosequenza comune ad X e Y. Ad esempio, se $X = (1, 0, 1, 0, 1)$ e $Y = (0, 0, 1, 1, 1)$ allora una sottosequenza comune di lunghezza massima è $(0, 1, 1)$. Viene proposto il seguente algoritmo:

INPUT due sequenze $X = (x_1, x_2, \ldots, x_n)$ e $Y = (y_1, y_2, \ldots, y_m)$
$NX \leftarrow 0$
FOR $i \leftarrow 1$ TO n DO
IF $x_i = 0$ THEN $NX \leftarrow NX + 1$
ENDFOR
$NY \leftarrow 0$
FOR $i \leftarrow 1$ TO m DO
IF $y_i = 0$ THEN $NY \leftarrow NY + 1$
ENDFOR
$L \leftarrow \mathsf{max}\{\mathsf{min}\{NX, NY\}, \mathsf{min}\{n - NX, m - NY\}\}$
OUTPUT L.

a. Provare che la lunghezza prodotta in output dall'algoritmo è la lunghezza di una sottosequenza comune alle due sequenze di input.

b. Far veder che l'algoritmo non risolve il problema.

c. Dimostrare che l'algoritmo garantisce un rapporto di approssimazione limitato da 2.

d. Esibire un'istanza per cui il rapporto di approssimazione è esattamente 2.

30. [**COPERTURA 1**] In un grafo non orientato G un sottoinsieme A dei suoi nodi è detto *copertura* se per ogni arco del grafo almeno uno degli estremi dell'arco è in A. Si vuole trovare una copertura per il grafo G che sia di minima cardinalità. Per risolvere il problema si propone il seguente algoritmo

INPUT un grafo non orientato $G = (\{1, 2, \ldots, n\}, E)$
$SOL \leftarrow \emptyset$
FOR $i \leftarrow 1$ TO n DO
IF $i \notin SOL$ THEN VISITA(i, G, SOL)
ENDFOR
OUTPUT SOL

VISITA:
INPUT un nodo i, un grafo non orientato G, un insieme di nodi SOL
    IF i è adiacente ad un nodo non in SOL THEN
        $SOL \leftarrow SOL \cup \{i\}$
        FOR ogni nodo j adiacente ad i DO
            IF j non è in SOL THEN VISITA(j, G, SOL)
        ENDFOR
    ENDIF
END

a. Provare che l'algoritmo produce una copertura per G.

b. Provare che l'algoritmo non è corretto.

c. Provare che l'algoritmo non ha un rapporto d'approssimazione costante inferiore a 2.

c. Provare che l'algoritmo ha un rapporto d'approssimazione limitato da 2.

31. **[COPERTURA 2]** In un grafo non orientato G un sottoinsieme A dei suoi nodi è detto *copertura* se per ogni arco del grafo almeno uno degli estremi dell'arco è in A. Si vuole trovare una copertura per il grafo G che sia di minima cardinalità. Per risolvere il problema si propone il seguente algoritmo greedy

INPUT un grafo non orientato $G = (V, E)$
    $SOL \leftarrow \emptyset$
    $A \leftarrow V$
    $B \leftarrow E$
    WHILE $B \neq \emptyset$ DO
        partiziona i nodi del grafo $G' = (A, B)$ in base al grado
        sia S una della partizioni per cui è minimo il valore $|S|+$ grado dei nodi in S
        sia v un nodo in S che ha minima la somma dei gradi dei nodi adiacenti in G'
        $SOL \leftarrow SOL \cup \{v\}$
        $A \leftarrow A - \{v\}$
        $B \leftarrow B - \{\{v, u\} | \{v, u\} \in B\}$
    ENDWHILE
OUTPUT SOL

a. Provare che l'algoritmo produce una copertura per G.

b. Provare che l'algoritmo non è corretto.

c. Provare che l'algoritmo non ha un rapporto d'approssimazione costante.

32. **[COPERTURA IN GRAFI ACICLICI]** In un grafo non orientato $G = (\{1, 2, \ldots, n\}, E$ un sottoinsieme A dei suoi nodi è detto *copertura* se per ogni arco del grafo almeno uno degli estremi dell'arco è in A. Si vuole trovare una copertura per il grafo non orientato e aciclico G che sia di minima cardinalità. Per risolvere il problema si propone il seguente algoritmo greedy descritto in modo ricorsivo e da invocare con i e j ugauali ad 1 e $SOL = \emptyset$.

COPERTURA:

INPUT due nodi i e j, un grafo non orientato aciclico $G = (\{1, 2, \dots, n\}, E)$ e un insieme di nodi SOL

$X \leftarrow$ i nodi adiacenti al nodo j diversi da i

FOR ogni nodo t in X DO COPERTURA(j, t, G, SOL)

IF$|X \cap SOL| < |X|$ THEN $SOL \leftarrow SOL \cup \{j\}$ END

a. Provare che l'algoritmo produce una copertura per G.

b. Descrivere una implementazione dell'algoritmo che abbia complessità $O(n)$.

c. Provare che l'algoritmo è corretto.

33. **[INSIEME ACICLICO 1]** In un grafo orientato $G = (V, E)$ un sottoinsieme A degli archi è detto *aciclico* se il grafo $G' = (V, A)$ non contiene cicli. Si vuole trovare un sottoinsieme aciclico per G di massima cardinalità. Per risolvere il problema si propone il seguente algoritmo.

```
INPUT un grafo orientato G = ({1, 2, ..., n}, E)
    SOL ← ∅
    VISITATI ← ∅
    FOR i ← 1 TO n DO
        IF i ∉ VISITATI THEN VISITA(i, VISITATI, G, SOL)
    ENDFOR
OUTPUT SOL

VISITA:
INPUT un nodo i, un insieme di nodi VISITATI, un grafo G, un insieme di archi SOL
    FOR< i, j >∈ E DO
        IF j ∉ VISITATI THEN
            SOL ← SOL ∪ {< i, j >}
            VISITATI ← VISITATI ∪ {j}
            VISITA(j, VISITATI, G, SOL)
        ENDIF
    ENDFOR
END
```

a. Provare che l'algoritmo produce un insieme aciclico per G.

b. Provare che l'algoritmo non è corretto.

c. Provare che l'algoritmo non ha un rapporto d'approssimazione costante.

34. **[INSIEME ACICLICO 2]** In un grafo orientato $G = (V, E)$ un sottoinsieme A degli archi è detto *aciclico* se il grafo $G' = (V, A)$ non contiene cicli. Si vuole trovare un sottoinsieme aciclico per G di massima cardinalità. Per risolvere il problema si propone il seguente algoritmo.

```
INPUT un grafo orientato G = ({1, 2, ... n}, E)
    SOL ← ∅
    A ← E
    FOR i ← 1 TO n DO
        IN ← gli archi in A entranti nel nodo i
```

```
        OUT ← gli archi in A uscenti dal nodo i
        IF |IN| ≥ |OUT| THEN
            SOL ← SOL ∪ IN
        ELSE
            SOL ← SOL ∪ OUT
        ENDIF
        A ← A − (IN ∪ OUT)
    ENDFOR
OUTPUT SOL
```

a. Provare che l'algoritmo produce un insieme aciclico per G.

b. Provare che l'algoritmo non è corretto.

c. Provare che l'algoritmo ha un rapporto d'approssimazione limitato da 2.

d. Provare che l'algoritmo non ha un rapporto d'approssimazione inferiore a 2.

e. Descrivere un'implementazione dell'algoritmo con complessità $O(V + E)$.

35. **[INSIEME ACICLICO 3]** In un grafo orientato $G = (V, E)$ un sottoinsieme A degli archi è detto *aciclico* se il grafo $G' = (V, A)$ non contiene cicli. Si vuole trovare un sottoinsieme aciclico per G di massima cardinalità. Per risolvere il problema si propone il seguente algoritmo.

```
INPUT un grafo orientato G = ({1, 2, . . . n}, E)
    SIN ← ∅
    DES ← ∅
    A ← E
    WHILE A ≠ ∅ DO
        estrai un arco < i, j > da A
        IF i > j THEN
            SIN ← SIN ∪ {< i, j >}
        ELSE
            DES ← DES ∪ {< i, j >}
        ENDIF
    ENDWHILE
    IF |SIN| ≥ |DES| THEN
        SOL ← SIN
    ELSE
        SOL ← DES
    ENDIF
OUTPUT SOL
```

a. Provare che l'algoritmo produce un insieme aciclico per G.

b. Provare che l'algoritmo non è corretto.

c. Provare che l'algoritmo ha un rapporto d'approssimazione limitato da 2.

d. Provare che l'algoritmo non ha un rapporto d'approssimazione inferiore a 2.

36. **[INSIEME CICLICO]** In un grafo orientato $G = (V, E)$ un sottoinsieme A degli archi è detto *ciclico* se il grafo $G' = (V, E - A)$ non contiene cicli. Si vuole trovare un sottoinsieme ciclico per G di minima cardinalità. Per risolvere il problema si propone il seguente algoritmo.

```
INPUT un grafo orientato G = ({1, 2, ... n}, E)
    SIN ← ∅
    DES ← ∅
    A ← E
    WHILE A ≠ ∅ DO
        estrai un arco < i, j > da A
        IF i > j THEN
            SIN ← SIN ∪ {< i, j >}
        ELSE
            DES ← DES ∪ {< i, j >}
        ENDIF
    ENDWHILE
    IF |SIN| ≤ |DES| THEN
        SOL ← SIN
    ELSE
        SOL ← DES
    ENDIF
OUTPUT SOL
```

a. Provare che l'algoritmo produce un insieme ciclico per G.

b. Provare che l'algoritmo non è corretto.

c. Provare che l'algoritmo non ha un rapporto d'approssimazione costante.

37. **[ORIENTAZIONE DI GRAFI]** Un *orientazione* di un grafo non orientato G è un grafo orientato G' ottenuto da G sostituendo ogni arco $\{a, b\}$ con un arco in $\{< a, b >, < b, a >\}$. Il grado entrante di un grafo orientato è il numero massimo di archi entranti in un nodo del grafo. Dato un grafo non orientato G si vuole trovare una sua orientazione che minimizzi il grado entrante. Per risolvere il problema si propone il seguente algoritmo greedy.

```
INPUT un grafo non orientato G = (V, E)
    A ← V
    B ← E
    SOL ← ∅
    WHILE A ≠ ∅ DO
        sia a un nodo di grado minimo in G' = (A, B)
        SOL ← SOL ∪ {< u, a > |{a, u} ∈ B}
        A ← A − {a}
        B ← B − {{u, a}|{u, a} ∈ B}
    ENDWHILE
OUTPUT G' = (V, SOL)
```

a. Provare che l'algoritmo produce un orientazione di G.

b. Provare che l'algoritmo ha un rapporto d'approssimazione pari a 2.

38. **[INSCATOLAMENTO]** Si supponga di avere n oggetti di peso $p_1, p_2, \ldots, p_n$, con $1 \leq p_i \leq C$, e di voler inserire tutti questi oggetti nel minor numero di contenitori. Ciascun contenitore può contenere un qualsiasi sottoinsieme di oggetti il cui peso non eccede C. Per risolvere il problema si propone il seguente algoritmo dove l'inscatolamento è dato da un vettore SOL ad n componenti dove nella locazione i-esima è indicato il contenitore in cui è stato inserito l'oggetto i-esimo.

```
INPUT i pesi p_1, p_2, ..., p_n degli oggetti e la capacità C dei contenitori
    contenitore ← 1
    peso ← 0
    FOR i ← 1 TO n DO
        IF peso + p_i ≤ C THEN
            SOL[i] ← contenitore
            peso ← peso + p_i
        ELSE
            contenitore ← contenitore + 1
            SOL[i] ← contenitore
            peso ← p_i
        ENDIF
    ENDFOR
OUTPUT SOL
```

a. Provare che l'algoritmo non risolve il problema.

b. Provare che l'algoritmo non ha un rapporto d'approssimazione costante inferiore a 2.

c. Provare che l'algoritmo ha un rapporto d'approssimazione costante limitato da 2.