

Algoritmi e Strutture Dati - Prova d'esame

06/06/11

Esercizio 1

Poichè questa è la funzione di complessità dell'algoritmo deterministico per la selezione che funziona in tempo lineare, utilizziamo $O(n)$ come guess.

- Limite superiore $O(n)$, dimostrato per sostituzione con induzione. Partendo dal caso base, si ha che

$$T(0) = 1 \leq c \cdot 0 \Leftrightarrow 1 \leq 0 \quad \text{Falso!}$$

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

$$T(2) = 22/5 + T(\lfloor 2/5 \rfloor) + T(\lfloor 14/10 \rfloor) = 22/5 + T(0) + T(1) = 32/5 \leq c \cdot 2 \Leftrightarrow c \geq 32/10$$

$$T(3) = 33/5 + T(\lfloor 3/5 \rfloor) + T(\lfloor 21/10 \rfloor) = 33/5 + T(0) + T(2) = 70/5 \leq c \cdot 3 \Leftrightarrow c \geq 70/15$$

$$T(4) = 44/5 + T(\lfloor 4/5 \rfloor) + T(\lfloor 28/10 \rfloor) = 44/5 + T(0) + T(2) = 81/5 \leq c \cdot 4 \Leftrightarrow c \geq 81/20$$

$$T(5) = 55/5 + T(\lfloor 5/5 \rfloor) + T(\lfloor 35/10 \rfloor) = 55/5 + T(1) + T(3)$$

Abbiamo calcolato tutti questi casi perché la condizione non viene rispettata per il caso $T(0)$, che poi si ripresenta nei casi $T(2)$, $T(3)$ e $T(4)$. A partire da $T(5)$, tuttavia, la ricorrenza utilizza solo casi già dimostrati; possiamo quindi interrompere il calcolo e utilizzare l'induzione.

Dimostriamo ora il caso induttivo, assumendo che $T(n') \leq cn'$ per tutti i valori $n' < n$, e volendo dimostrare che $T(n) \leq cn$. Sostituendo abbiamo che

$$\begin{aligned} T(n) &\leq 11/5n + c\lfloor n/5 \rfloor + c\lfloor 7n/10 \rfloor \\ &\leq 11/5n + 1/5cn + 7/10cn \\ &= 9/10cn + 22/10n \leq cn \end{aligned}$$

L'ultima disequazione è vera per $c \geq 22$. Quindi, per $m = 1$, $c = \max\{22, 1, \frac{32}{10}, \frac{70}{15}, \frac{81}{20}\} = 22$ abbiamo che $T(n) \leq cn, \forall n \geq m$.

- Limite inferiore $\Omega(n)$, dimostrato per sostituzione con induzione: Partendo dal caso base, si ha che

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

Dimostriamo ora il caso induttivo, assumendo che $T(n') \geq cn'$ per tutti i valori $n' < n$, e volendo dimostrare che $T(n) \geq cn$. Sostituendo abbiamo che:

$$\begin{aligned} T(n) &\geq 11/5n + c\lfloor n/5 \rfloor + c\lfloor 7n/10 \rfloor \\ &\geq 11/5n \geq cn \end{aligned}$$

L'ultima disequazione è vera per $c \leq 11/5$. Quindi, per $m = 1$, $c = 1$, abbiamo che $T(n) \geq cn, \forall n \geq m$.

Esercizio 2

Tramite uno degli algoritmi che risolvono il problema della selezione in tempo lineare, è possibile individuare la mediana m . Utilizzando un vettore di appoggio B , si calcola la differenza assoluta

$$B[i] = |V[i] - m|, 1 \leq i \leq n$$

Nuovamente tramite il problema della selezione, è possibile trovare la k -esima distanza assoluta d dalla mediana. A questo punto si scorre nuovamente il vettore stampando tutti i valori $V[i]$ la cui distanza $|V[i] - m| < d$ e infine stampando i valori con distanza pari a d fino a raggiungere il valore k . Fra l'altro, questa versione è in grado di gestire anche il caso in cui i valori di input non siano distinti.

```
nearest(int[] V, int n, int k)
    int mediana = selezione(V, 1, n, ⌈n/2⌉)
    int[] B = new int[1...n]
    for i = 1 to n do
        B[i] = |V[i] - mediana|
    int d = selezione(V, 1, n, k)
    int count = 0
    for i = 1 to n do
        if B[i] < d and V[i] ≠ mediana then
            print V[i]
            count = count + 1
    for i = 1 to n do
        if B[i] = d and count < k then
            print V[i]
            count = count + 1
```

Esercizio 3

Una semplice soluzione $O((m+n)n)$ consiste nel fare partire una visita BFS a partire da ogni nodo in V_1 ; poiché $V_1 \subseteq (V)$, è possibile che vengano effettuate $O(n)$ visite di costo $O(m+n)$.

Una soluzione più efficiente consiste nel modificare la BFS in modo che tutti i nodi in V_1 abbiano distanza 0 e siano inseriti nella coda. A questo punto, l'algoritmo estrarrà dalla coda tutti i vicini di nodi in V_1 che non appartengano a V_1 , assegnerà loro distanza 1 e li inserirà in coda. Al termine di questa prima "passata", avremo analizzato solo e unicamente i nodi a distanza 1 da V_1 . Se fra essi esiste un nodo di V_2 , si ritornerà 1 e si terminerà. Altrimenti, si procederà alla seconda passata, in cui vengono estratti dalla coda tutti i nodi a distanza 1 da V_1 e si inseriranno in coda tutti i nodi a distanza 2 da V_1 . In questo modo, vengono individuati tutti i nodi a distanza 1, 2, ..., i , dai nodi in V_1 . Il primo nodo in V_2 che si incontra avrà distanza minima. Se non viene trovato alcun nodo in V_2 , vuole dire che non sono raggiungibili e si può ritornare $+\infty$. Non essendo altro che una visita BFS, l'algoritmo ha complessità $O(m+n)$.

```
int mindist(Graph G, Set V1, Set V2)
```

```
    Queue Q = Queue()
    int[] dist = new int[1...V.n]
    foreach u ∈ G.V() do
        if V1.contains(u) then
            Q.enqueue(u)
            dist[u] = 0
            if V2.contains(u) then
                return 0
        else
            dist[u] = ∞
    while not Q.isEmpty() do
        Node u = Q.dequeue()
        foreach v ∈ G.adj(u) do
            if dist[v] = ∞ then
                dist[v] = dist[u] + 1
                if V2.contains(v) then
                    return dist[v]
            Q.enqueue(v)
    return +∞
```

% Esamina l'arco (u, v)
 % Il nodo u non è già stato scoperto

Soluzione alternativa Una soluzione alternativa costruisce un nuovo grafo $G' = (V', E')$ dove V' è ottenuto sostituendo tutti i nodi in V_1 con un unico nodo \mathbf{v}_1 e tutti i nodi in V_2 con un unico nodo \mathbf{v}_2 ; l'insieme degli archi è ottenuto sostituendo tutti gli archi uscenti da un nodo in V_1 o entranti in un nodo in V_2 con archi che escono da \mathbf{v}_1 o entrano in \mathbf{v}_2 , rispettivamente. Più formalmente,

$$\begin{aligned} V' &= V - (V_1 \cup V_2) \cup \{\mathbf{v}_1, \mathbf{v}_2\} \\ E' &= E - \{(u, v) : (u, v) \in E \wedge (u \in V_1 \vee v \in V_2)\} \cup \\ &\quad \{(\mathbf{v}_1, v) : \exists u, u \in V_1 \wedge v \notin V_1 \wedge (u, v) \in E\} \cup \\ &\quad \{(u, \mathbf{v}_2) : \exists v, v \in V_2 \wedge u \notin V_2 \wedge (u, v) \in E\} \end{aligned}$$

La costruzione di questo grafo richiede tempo $O(m + n)$, in quanto le operazioni di inserimento nel nuovo grafo possono essere effettuate in tempo $O(1)$ (come avevamo fatto per la costruzione del grafo trasposto per determinare le componenti fortemente connesse) e le operazioni di verifica di appartenenza sugli insiemi richiedono $O(1)$ (come da definizione del problema), sia con liste di adiacenza ma ancor più facilmente con matrici di adiacenza.

A questo punto, è sufficiente fare una visita BFS sul grafo G' a partire dal nodo \mathbf{v}_1 e misurare la distanza con \mathbf{v}_2 ; avendo eliminato tutti gli archi interni a V_1 e V_2 , questa distanza rappresenta la minima distanza da un nodo in V_1 ad un nodo in V_2 .

Esercizio 4

È possibile utilizzare la programmazione dinamica, in quanto il problema gode della sottostruttura ottima.

Definiamo con $N[j]$ il massimo numero di scatole inseribili l'una nell'altra scegliendo fra le prime j scatole. Supponiamo di aggiungere una scatola fittizia S_0 con dimensioni $(0, 0, 0)$. $N[j]$ può essere così definita:

$$N[j] = \begin{cases} 0 & j = 0 \\ \max\{N[j-1], N[\max\{i : 0 \leq i < j \wedge S_i \subset S_j\}] + 1\} & j > 0 \end{cases}$$

Ovvero, se la scatola j viene presa, ci si riduce al più grande sottoproblema dato da i scatole tale per cui la scatola S_i è inseribile in S_j (e si aumenta di 1 il numero di scatole prese), oppure la scatola j non viene presa, e si considera il sottoproblema dato da $j-1$.

Il codice basato su programmazione dinamica è il seguente:

```

int scatole(BOX[]  $S$ , int  $n$ )


---


int[]  $N$  = new int[ $0 \dots n$ ]
int[]  $prev$  = new int[ $1 \dots n$ ]
int  $i, j$ 
 $N[0] = 0$                                 % Per ogni scatola  $j$ , individua la scatola con indice più alto e inseribile in  $S[j]$ 
for  $j = 1$  to  $n$  do
     $i = j - 1$ 
    while  $i > 0$  and not  $S[i] \subset S[j]$  do
         $i = i - 1$ 
     $prev[j] = i$ 

for  $j = 1$  to  $n$  do                                % Calcola il vettore  $N$ 
     $N[j] = \max(N[j-1], N[prev[j]])$ 

 $j = n$                                             % Stampa un sottoinsieme di scatole, dalla più grande alla più piccola
while  $j > 0$  do
    if  $N[j] \neq N[j-1]$  then
        print  $j$ 
         $j = prev[j]$ 
    else
         $j = j - 1$ 

```

La complessità è $O(n^2)$, dominata dalla costruzione del vettore $prev$.