Algoritmi per Bioinformatica **Anno Acc. 2014-15**

Esercizi 3

1. Ricordiamo il problema della ricerca di un elemento in un array ordinato: L'input è un array $A[1 \dots n]$ di n elementi in ordine crescente, ed un valore x. L'output deve essere la posizione di x in A se A contiene un elemento del valore x ed altrimenti l'output deve essere "non trovato".

   Una soluzione nota è quella della ricerca binaria. Considerate la seguente variante: ad ogni iterazione l'algoritmo sceglie 3 posizioni $1 \leq i < j < k \leq n$ e confronta x con $A[i]$, con $A[j]$ e con $A[k]$. Se nessuna di queste posizioni è quella giusta continua nel sotto-intervallo appropriato.

   Si scriva con precisione l'algoritmo–completando i dettagli relativi a come scegliere efficientemente le 3 posizioni—e se ne analizzi la complessità, usando la notazione O. Si provi anche a valutare il caso pessimo per l'algoritmo usando la notazione Θ.

2. Dobbiamo organizzare una gita per n amici. Abbiamo a disposizione due date possibili. Sappiamo che alcuni di loro hanno litigato e non vogliono andare insieme. Scrivere un algoritmo per risolvere il problema. Immaginiamo di rappresentare gli amici con i numeri da 1 a n. L'algoritmo prende in input per ogni partecipante i la lista $A[i]$ degli amici con cui i ha litigato. L'output dell'algoritmo deve essere: o la suddivisione degli n amici in due gruppi, in modo tale che nello stesso gruppo non ci sono due che hanno litigato tra loro; oppure se tale suddivisione non è possibile, l'algoritmo risponde "nessuna soluzione".

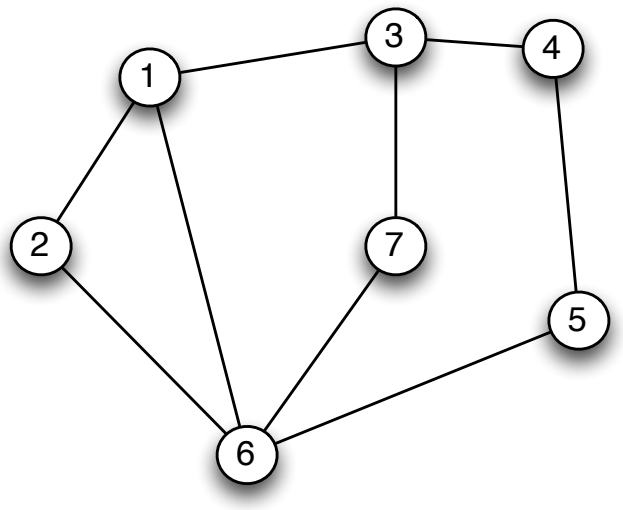
   Per un bonus di punti addizionale, nel caso di output "nessuna soluzione", l'algoritmo deve fornire anche una qualche prova—motivata—che non esiste una soluzione.

3. A lezione abbiamo visto che l'algoritmo del cassiere non funziona sempre. Per esempio, con i seguenti tagli (francobolli italiani): 70c, 85c, 90c, 1Euro, 1.40Euro, 2Euro, 2.20Euro; la somma 2.70 non può essere cambiata usando l'algoritmo del cassiere, mentre esiste una soluzione: 2Euro + 70c—che è anche ottimale.

   Un nostro amico ci ha suggerito la seguente modifica dell'algoritmo del cassiere, sostenendo che riesca a fornire sempre una soluzione—e, secondo lui, quella ottima:

   1. procedi con le scelte dell'algoritmo del cassiere
   2. se si riesce a cambiare l'intera somma, dai in output la soluzione trovata
   3. se l'algoritmo del cassiere termina perchè nessuno dei tagli è più piccolo della somma da cambiare, allora non esiste una soluzione
   3. se l'algoritmo del cassiere riesce a costruire solo una soluzione parziale, cioè non riesce a cambiare l'intera somma, allora ricomincia da 1. utilizzando solo i tagli $\{c_1, \dots, c_{k-1}\}$, dove c_k è il taglio di moneta più grande utilizzato dalla soluzione parziale costruita al precedente punto 1.

   Esempio: con l'istanza di cui sopra, procederemmo come segue: Lanciamo l'algoritmo del cassiere con i tagli (espressi in centesimi) $\{70, 85, 90, 100, 140, 200, 220\}$. L'algoritmo fallisce dopo avere scelto 220c perché la somma rimanente 50c non può essere cambiata. Quindi,



i\j	1	2	3	4	5	6	7
1	-	2	3	3	3	6	6
2	1	-	1	1	1	6	6
3	1	1	-	4	4	7	7
4	3	3	3	-	5	5	3
5	6	6	4	4	-	6	6
6	1	2	7	5	5	-	7
7	6	6	3	3	6	6	-

Il grafo G La matrice **PRIMOPASSO**

Figura 1: Esempio per l'esercizio 4

secondo l'algoritmo del nostro amico, dovremmo rilanciare l'algoritmo del cassiere dopo avere eliminato dall'insieme dei tagli quello da 220c e tutti quelli superiori. Quindi dobbiamo ripetere usando solo i francobolli $\{70, 85, 90, 100, 140, 200\}$. Ora la soluzione costruita sarà $200 + 70$ che è quella ottima.

Siamo convinti che questo sia solo un caso e che l'algoritmo del nostro amico sia ugualmente fallimentare. Vanno mostrati due controesempi: esiste un caso in cui l'algoritmo del nostro amico fornisce la soluzione e non è quella ottima; (ii) esiste un caso in cui l'algoritmo del nostro amico non fornisce una soluzione anche se ne esiste una.

4. Abbiamo un grafo connesso $G = (V, E)$ con $|V| = n$ e $|E| = m$. Per ogni coppia di nodi i, j esiste un cammino minimo da i a j—minimo nel senso di minimo numero di archi da attraversare. Vorremmo sapere quando siamo in un nodo i e vogliamo dirigerci verso j quale è il primo passo da fare, cioè il primo vertice, tra i vicini di i su cui spostarci, in modo da stare su un cammino minimo.

   A questo scopo, vogliamo riempire una matrice $n \times n$, che chiamiamo PrimoPasso, in cui le righe e le colonne rappresentano i vertici del grafo, in modo tale che nella cella PrimoPasso$[i, j]$, all'incrocio della riga i e della colonna j, ci sia il primo nodo verso cui spostarci quando da i vogliamo andare a j su un cammino minimo.

   Fornire un algoritmo che risolve tale problema. Analizzarne la complessità.

   Esempio. La Figura 1 mostra un grafo con 7 nodi ed una possibile costruzione della matrice 7×7 PrimoPasso.

5. Si analizzi la complessità dei seguenti algoritmi. Detto $T(n)$ il running time dell'algoritmo, si determini una funzione $f(n)$ tale che $T(n) = O(f(n))$. Inoltre si dica se vale anche $T(n) = \Theta(f(n))$. Le risposte vanno motivate

---

ALGO1(n)

1: $j \leftarrow 1,\ k \leftarrow 0$
2: **while** $j \leq n$ **do**
3: **for** $\ell = 1$ **to** n-j **do**
4: $k \leftarrow k + j$
5: **end for**
6: $j \leftarrow j + 2$
7: **end while**

ALGO2(n)

1: $j \leftarrow 1,\ k \leftarrow 0$
2: **while** $j \leq n$ **do**
3: **for** $\ell = 1$ **to** n-j **do**
4: $k \leftarrow k + j$
5: **end for**
6: $j \leftarrow j \times 2$
7: **end while**

ALGO3(n)

1: $c \leftarrow 1$
2: **for** $i = 1$ **to** n **do**
3: $j \leftarrow i$
4: **while** $j \leq n/2$ **do**
5: **for** $k = n/2$ **to** $n - j$ **do**
6: $c \leftarrow c + 1$
7: **end for**
8: $j \leftarrow j + 1$
9: **end while**
10: **end for**

ALGO4($a_1, a_2, \ldots, a_n$)

1: **if** $n = 1$ **then**
2: **return** a_1
3: **else**
4: $x \leftarrow$ ALGO4$(a_1, a_2, \ldots, a_{n/2})$
5: $y \leftarrow$ ALGO4$(a_{n/2}, a_{n/2+1}, \ldots, a_n)$
6: $m \leftarrow x \times y$
7: **for** $i = 1$ **to** $n/8$ **do**
8: $m \leftarrow \min\{m, \max\{a_{4i}, a_{n/2+4i}\}\}$
9: **end for**
10: **return** m
11: **end if**

---

6. In un cassetto ci sono 20 guanti: 5 paia di guanti neri, 3 paia di guanti rossi, 2 paia di guanti bianchi. Possiamo prendere un certo di numero di guanti al buio ma possiamo vedere il loro colore e tipo (mano destra o mano sinistra) solo dopo averli estratti.

   Qual è il minimo numero di guanti che dobbiamo estrarre (senza guardare) per esser certi di estrarre almeno un paio dello stesso colore?

   Qual è il minimo numero di guanti che dobbiamo estrarre (senza guardare) per esser certi di estrarre almeno un paio di ogni colore?

   Qual è il minimo numero di guanti che dobbiamo estrarre (senza guardare) per esser certi di estrarre almeno un paio di colore diverso, cioè un guanto destro e un guanto sinistro di colore diverso?

   Si giustifichino con precisione le affermazioni fatte.

7. n amici verranno a farci visita a Verona ed ognuno di essi ci ha già comunicato le date del suo arrivo e della sua partenza. Vorremmo organizzare un party. Vogliamo sapere il massimo numero di amici che saranno a Verona contemporaneamente e quando questo accadrà. Immaginate di avere per ogni amico, $a = 1, \ldots, n$, il tempo s_a di arrivo di a ed il tempo f_a di partenza di a. Per semplicità possiamo pensare che i tempi siano espressi come interi, indicanti il numero di giorni a partire da oggi. Progettare un algoritmo che fornisce il massimo numero di amici che saranno qui a Verona contemporaneamente ed un momento in cui tale evento si verificher.

   Si analizzi la complessità dell'algoritmo proposto.

8. Provare le seguenti relazioni, usando la definizione della notazione asintotica Θ, cioè esibendo le opportune costanti c1,c2 ed n0, o in alternativa usando le proprietà di composizione delle notazioni asintotiche (somma, prodotto, transitività etc.) Si assuma che i logaritmi siano in base 2. Si giustifichino le affermazioni fatte.

   (a) $n \log \sqrt{n} = \Theta(n \log n)$
   (b) $1000n \log n^8 + 10n\sqrt{n} = \Theta(n\sqrt{n})$
   (c) $\log(n!) = \Theta(n \log n)$
   (d) $n^{3/2} - 2^{\frac{1}{2}\log n} \log n^{100} = \Theta(n\sqrt{n})$

9. Si supponga di avere un algoritmo che su di un input di una data taglia n impiega 4 minuti per terminare. Quanti minuti l'algoritmo impiegherà su di un input di taglia $n+1$ se il tempo di esecuzione $T(n)$ dell'algoritmo è

   (a) $T(n) = \log n$
   (b) $T(n) = n$
   (c) $T(n) = n^2$
   (d) $T(n) = 2^n$
   (e) $T(n) = 2^{2^n}$