03MNO Algoritmi e Programmazione

Appello del 31/01/2019 - Prova di teoria (12 punti)

1. (2.5 punti)

Si risolva la seguente equazione alle ricorrenze mediante il metodo dello sviluppo (unfolding):

$$T(n) = 16T(n/4) + 5n^3 \quad n>1$$
$$T(1) = 1 \quad n=1$$

2. (2 punti)

Sia data la sequenza di interi, supposta memorizzata in un vettore:

2 93 19 11 31 34 25 23 35 9 75 27

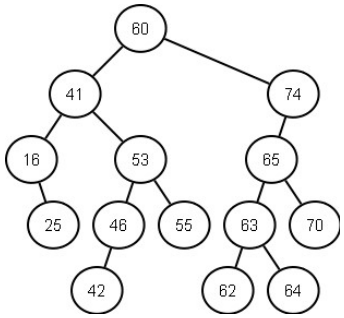
si eseguano i primi 2 passi dell'algoritmo di quicksort per ottenere un ordinamento ascendente, indicando ogni volta il pivot scelto. NB: i passi sono da intendersi, impropriamente, come in ampiezza sull'albero della ricorsione, non in profondità. Si chiede, pertanto, che siano ritornate le 2 partizioni del vettore originale e le due partizioni delle partizioni trovate al punto precedente.

3. (2.5 punti)

Si determini una Longest Increasing Sequence della sequenza 7, 4, 6, 3, 8, 9, 2 mediante un algoritmo di programmazione dinamica. Si evidenzino i passaggi intermedi.

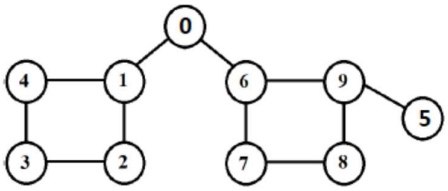
4. (2 punti)

Si partizioni il seguente BST attorno alla chiave 55:



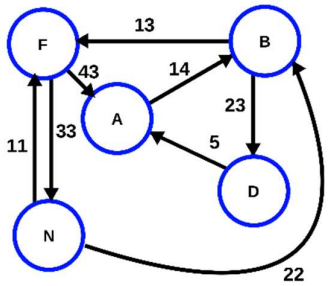
5. (1 punto)

Si determinino i punti di articolazione del seguente grafo non orientato. Si assuma, qualora necessario, un ordine numerico per i vertici



6. (2 punti)

Sia dato il seguente grafo orientato pesato:



Si determinino i valori di tutti i cammini minimi che collegano il vertice **B** con ogni altro vertice mediante l'algoritmo di Dijkstra. Si assuma, qualora necessario, un ordine alfabetico per i vertici e gli archi.

03MNO Algoritmi e Programmazione

Appello del 31/01/2019 - Prova di programmazione (12 punti)

1. (2 punti)

Sia data una matrice `An × m` di interi. Si scriva una funzione C che ritorni l'indice di una colonna qualsiasi fra quelle in cui la massima differenza (in valore assoluto) tra due elementi consecutivi sia minima.

Esempio: data la seguente matrice `A` di `n` = 4 righe e `m` = 3 colonne:

$$A = \begin{pmatrix} 15 & 13 & 7 \\ 6 & 18 & 4 \\ 11 & 4 & 12 \\ 13 & 9 & 5 \end{pmatrix}$$

La massima differenza tra gli elementi consecutivi della prima colonna è 9 = 15−6; per la seconda colonna è 14 = 18−4; per la terza è 8 = |4−12|. Quindi la funzione deve stampare l'indice 2 della terza colonna.

La funzione ha il seguente prototipo:

```
int minmaxdiff(int **A, int n, int m);
```

2. (4 punti)

Si implementi una funzione caratterizzata dal seguente prototipo:

```
void splice (list L1, list L2, int start, int num);
```

Tale funzione rimuove `num` elementi a partire dalla testa di `L2` e li sposta nel medesimo ordine in `L1`, posizionandoli immediatamente a seguire del nodo che occupa la posizione `start`. Si assuma che le posizioni nelle liste siano indicizzate da zero. Oltre che l'implementazione della funzione `splice`, si richiede anche l'esplicita definizione del tipo `list` e dei nodi usati all'interno delle liste. La lista sia un ADT di I classe.

Ai fini dell'esercizio, non si può fare uso di funzioni di libreria.

Esempio:

se `L1 = [1, 3, 5, 7]` e `L2 = [7, 4, 9]`, `splice(L1, L2, 1, 2)` darà

`L1 = [1, 3, 7, 4, 5, 7]` e `L2 = [9]`

3. (6 punti)

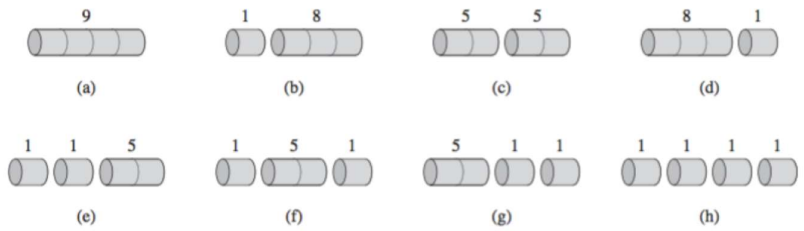
Un nastro lungo `n` (intero) può essere tagliato in pezzi di `m` (intero) diverse lunghezze intere `lun` dove 1 ≤`lun`≤`n`. Ciascuno dei pezzi viene venduto ad un determinato prezzo. Si possono tagliare più pezzi della stessa lunghezza. Un vettore `prezzo` di `m` interi contiene i prezzi per ciascuna lunghezza e un vettore `lunghezza` di `m` interi contiene le lunghezze ammesse.

Si scriva una funzione C che, noti `n`, `m`, `lunghezze` e `prezzo`, determini come tagliare il nastro in modo da massimizzare il valore complessivo dei pezzi venduti.

Esempio: se `m`=8, `lunghezza`[8] = {7, 4, 8, 1, 5, 2, 6, 3} e `prezzo`[8] = {17, 9, 20, 1, 10, 5, 17, 8}, la lunghezza dei nastri tagliati varia tra 1 e 8, un pezzo di nastro di lunghezza 7 costa 17, di lunghezza 4 costa 9, di lunghezza 8 costa 20 etc.

Se il nastro è lungo `n`=4, la figura seguente mostra come tagliarlo con 0, 1, 2 o 3 tagli in pezzi di lunghezza variabile da 1 a 4. Si osservi come le soluzioni (b) e (d) siano equivalenti, come pure le soluzioni (e), (f) e (g).

La soluzione ottima è (c) con 1 tagli che genera 2 nastri di lunghezza 2 e valore complessivo 10.



03MNO Algoritmi e Programmazione

Appello del 31/01/2019 - Prova di programmazione (18 punti)

1. **(18 punti)**

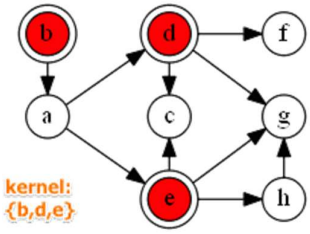
Nella Teoria dei Grafi, dato un grafo orientato G = (V, E), si definisce **Kernel** o **Nucleo** un insieme K $\subseteq$V di vertici di G tale per cui valgono entrambe le seguenti condizioni:

- qualunque coppia di vertici $k_i \in$ K e $k_j \in$K si consideri, non esiste un arco che li renda adiacenti

$$\forall i, j \text{ con } 0 \leq i, i \leq V\text{-}1 \ (k_i, k_j) \notin E \ \&\& \ (k_j, k_i) \notin E$$

- per ogni nodo u $\in$ V-K esiste un nodo k $\in$ K tale per cui l'arco (k, u)$\in$ E.

Esempio: dato il seguente grafo orientato, K = {b, d, e}. Infatti non esistono archi tra i nodi b, d ed e, per i nodi a, c, f, g, h esiste per ognuno un nodo in K da cui essi sono raggiunti mediante un arco.



Un grafo orientato G = (V, E) è memorizzato in un file mediante l'elenco dei suoi archi con il seguente formato:

$$idV_1 \ idV_2$$

che indica che (idV1, idV2)$\in$ E, dove idV1$\in$ V e idV2$\in$ V.Ogni vertice è individuato mediante un identificatore alfanumerico di lunghezza massima uguale a 20 caratteri. Si può assumere che non esistano archi duplicati. Non è lecito assumere nessuna forma di ordinamento degli archi.

Si scriva un programma C in grado di:

1. ricevere 3 nomi di file come parametri sulla riga di comando:
   - il primo file contenente la descrizione del grafo G
   - il secondo file contenente un elenco di vertici uno per riga
   - il terzo file su cui memorizzare il risultato
2. leggere il grafo e memorizzarlo in un'opportuna struttura dati
3. verificare se i vertici letti dal secondo file formano un kernel di G
4. identificare un **kernel minimo**di G e memorizzarlo sul terzo file. Per minimo si intende un kernel, tale per cui non ne esista un altro di cardinalità minore
5. calcolare la lunghezza del cammino semplice che attraversa il **maggior** numero di nodi del kernel. Non è richiesta la stampa del cammino, ma solo il numero di nodi del kernel che esso attraversa. **Non è ammesso per questo punto l'utilizzo di funzioni di libreria.**

Nota Bene: si osservi che non è necessario aver svolto il punto 4 per poter svolgere il 5. Basta infatti assumere di conoscere la soluzione del punto 4 e usarla come input del punto 5.

PER ENTRAMBE LE PROVE DI PROGRAMMAZIONE (18 o 12 punti):

- ***indicare nell'elaborato e nella relazione nome, cognome e numero di matricola.***
- ***se non indicato diversamente, è consentito utilizzare chiamate a funzioni standard, quali ordinamento per vettori, funzioni su FIFO, LIFO, liste, BST, tabelle di hash, grafi e altre strutture dati, considerate come librerie esterne.***
- ***gli header file devono essere allegati all'elaborato (il loro contenuto riportato nell'elaborato stesso). Le funzioni richiamate, inoltre, dovranno essere incluse nella versione del programma allegata alla relazione. I modelli delle funzioni ricorsive non sono considerati funzioni standard.***
- ***consegna delle relazioni (per entrambe le tipologie di prova di programmazione): entro domenica 03/02/2019, alle ore 14:00, mediante caricamento su Portale. Le istruzioni per il caricamento sono pubblicate sul Portale nella sezione Materiale). QUALORA IL CODICE CARICATO CON LA RELAZIONE NON COMPILI CORRETTAMENTE, VERRÀ APPLICATA UNA PENALIZZAZIONE. Si ricorda che la valutazione del compito viene fatta, senza la presenza del candidato, sulla base dell'elaborato svolto in aula. Non verranno corretti i compiti di cui non sarà stata inviata la relazione nei tempi stabiliti.***