03MNO Algoritmi e Programmazione

Appello del 13/02/2018 - Prova di teoria (12 punti)

1. (2.5 punti)

Si inseriscano in sequenza i seguenti dati, composti da una stringa e da un intero che ne rappresenta la priorità, in una coda a priorità di indici inizialmente supposta vuota:

`"Nel" 5  "mezzo" 10  "del" 7  "cammin" 12  "di" 4`

Si ipotizzi di usare uno heap (priorità massima in radice, vettore di 7 celle) per la coda a priorità e che la tabella di hash di dimensione 11 per la tabella di simboli abbia il seguente contenuto:

0	1	2	3	4	5	6	7	8	9	10
	di	Nel	del	mezzo				cammin		
	4	5	7	10				12		

Si disegnino lo heap e il vettore `qp` ai diversi passi dell'inserzione. Al termine si cambi la priorità di `"di"` da 4 a 20 e si disegnino i corrispondenti heap e vettore `qp`.

2. (2 punti)

Si determini mediante un algoritmo greedy un codice di Huffman ottimo per i seguenti simboli con le frequenze specificate: A: 4 B: 12 C: 1 D: 8 E: 16 M: 2 O: 14 Q: 13 T: 11

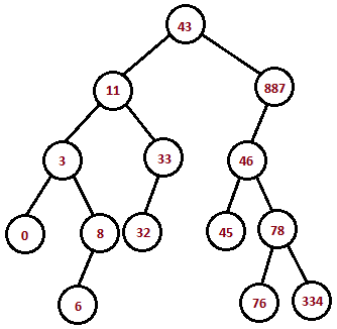
3. (2 punti)

Sia data la sequenza di chiavi intere: 21 94 267 302 98 100 45 207 13 99 181

Si riporti il contenuto di una tabella di hash di dimensione 23, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si usi l'open addressing con quadratic probing. Si definiscano gli opportuni coefficienti.

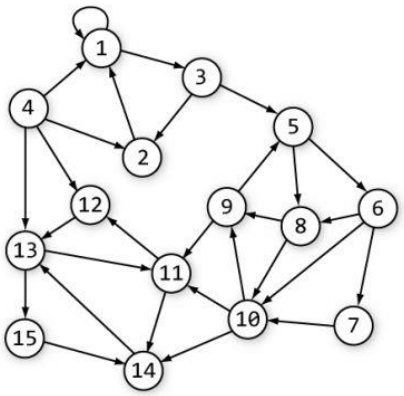
4. (2 punti)

Si cancelli la chiave 43 nel seguente BST:



5. (2 + 1.5 punti)

Sia dato il seguente grafo orientato:



- se ne effettui una visita in profondità, considerando **9** come vertice di partenza (**2 punti**) ed etichettando i vertici con tempo di scoperta/tempo di fine elaborazione
- lo si ridisegni, etichettando ogni suo arco come T (tree), B (back), F (forward), C (cross), considerando **9** come vertice di partenza (**1.5 punti**).

Qualora necessario, si trattino i vertici secondo l'ordine numerico.

03MNO Algoritmi e Programmazione

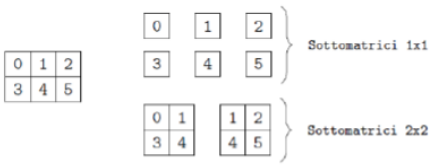
Appello del 13/02/2018 - Prova di programmazione (12 punti)

1. (**2 punti**)

Sia data una matrice *A* di numeri interi con *n* righe ed *m* colonne. Si scriva una funzione C con il seguente prototipo che visualizzi tutte le matrici quadrate contenute in *A*:

```
void displSquare(int **A, int n, int m);
```

Esempio: con la seguente matrice di 2 righe e 3 colonne, l'output deve elencare le seguenti matrici (il formato di stampa è libero):

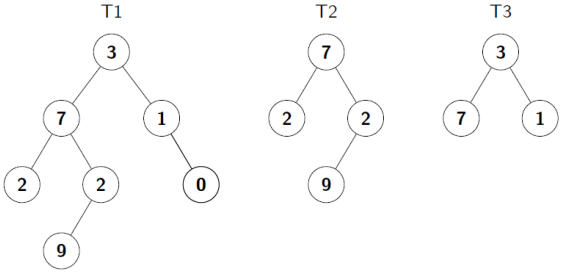


2. (**4 punti**)

Dati 2 alberi binari cui si accede mediante il puntatore alla radice, si scriva una funzione C con il seguente prototipo che decida se il secondo è uguale a un sottoalbero del primo o meno.

```
int subtree(link root1, link root2);
```

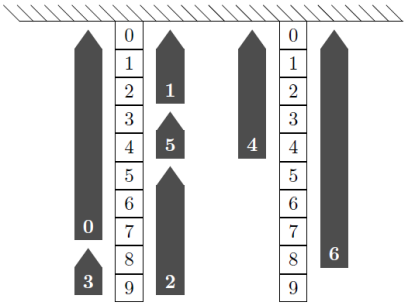
Si supponga la disponibilità di una funzione `KEYcmp(KEY k1, KEY k2)` per comparare le chiavi. Un sottoalbero di un albero T è un albero composto da un nodo che appartiene a T e da **tutti** i suoi discendenti. Esempio: T2 è un sottoalbero di T1, T3 non è un sottoalbero di T1:



3. **(6 punti**)

Un porto ha *n* moli di ugual lunghezza intera *lun*. A ogni molo si accede da entrambi i lati. Una nave è caratterizzata da un intero che ne indica lo spazio necessario per attraccare a un molo. Le navi che chiedono di attraccare sono *k* e lo spazio richiesto da ciascuna nave è un intero contenuto in una cella del vettore *navi*. Si scriva in C una funzione ricorsiva che faccia attraccare, se possibile, le *k* navi agli *n* moli minimizzando il numero di moli utilizzati.

Esempio: con 4 moli di lunghezza 10 utilizzabili sia a sinistra sia a destra e 6 navi (di dimensioni 8, 3, 5, 2, 5, 2 e 9), la soluzione che utilizza il minimo numero di moli (2) è:



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 <u>standard</u>, 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 venerdì 16/02/2018, alle ore 12:00, 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, <u>sulla base dell'elaborato svolto in aula</u>. Non verranno corretti i compiti di cui non sarà stata inviata la relazione nei tempi stabiliti.***

03MNO Algoritmi e Programmazione

Appello del 13/02/2018 - Prova di programmazione (18 punti)

1. (**18 punti**)

Un grafo pesato e non orientato è memorizzato in un file di testo `grafo.txt`. I nodi del grafo rappresentano punti del piano cartesiano Oxy. Il peso dell'arco che separa due nodi coincide con la distanza euclidea tra i 2 vertici su cui insiste l'arco stesso.

Il file ha il seguente formato:

- sulla prima riga un intero `N` rappresenta il numero di vertici del grafo
- seguono `N`righe ciascuna delle quali contiene una terna di valori `<id> <x> <y>` a rappresentare un identificatore alfanumerico (max 10 caratteri) e le coordinate reali di ogni nodo
- seguono un numero indefinito di coppie `<id_1> <id_2>` a rappresentare gli archi del grafo.

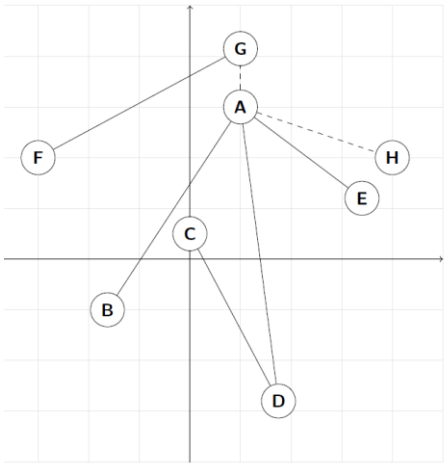
Il grafo in input è **non connesso**, per cui esistono almeno due componenti connesse.

Si definisce **diametro** di un (sotto)grafo il più lungo tra i cammini minimi tra ogni coppia di nodi del (sotto)grafo stesso.

Si chiede di implementare un programma C che:

- legga da file il grafo (il nome è passato sulla riga dei comandi) e lo memorizzi in un'apposita struttura dati (ADT di I classe)
- legga due file `sol1.txt` e `sol2.txt`, riportanti ognuno un insieme di archi da aggiungere al grafo originale, in ragione di una coppia `<id_1> <id_2>` per riga. Il programma stabilisca quale dei due insiemi porti ad avere il grafo finale ad essere connesso con diametro inferiore
- individui un insieme di archi a cardinalità minima da aggiungere al grafo originale così che il grafo finale sia connesso. Tra tutti gli eventuali insiemi di archi a cardinalità minima che soddisfano la condizione precedente, si ritorni come risultato ottimo quello per cui il grafo finale ha diametro minimo.

Nell'esempio seguente il numero minimo di archi da aggiungere per avere un grafo connesso e a diametro minimo è 2 e gli archi sono A-G e A-H.



Si suggerisce di utilizzare per il grafo l'ADT di I classe e gli algoritmi sui grafi presentati a lezione. Gli algoritmi di calcolo dei cammini minimi ritornino il vettore allocato dinamicamente delle distanze minime dalla sorgente corrente. Per la verifica di connettività si consiglia di utilizzare la funzione vista a lezione di calcolo delle componenti connesse modificata in modo da ritornare il vettore allocato dinamicamente delle componenti connesse. In alternativa si realizzi una funzione ad hoc di verifica di connettività. I nomi dei vertici siano gestiti mediante tabella di simboli, scegliendo se realizzarla con ADT o meno.