03MNO Algoritmi e Programmazione

Appello del 28/01/2020 - Prova di teoria (12 punti)

1. (2 punti)

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

12 83 29 1 41 24 35 13 45 9 65 17

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.

2. (2.5 punti)

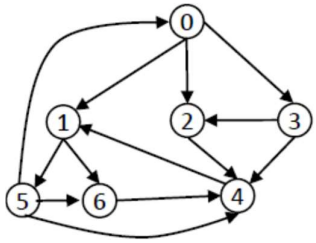
Data la catena di matrici (A_1, A_2, A_3, A_4) di dimensioni (2x5), (5x4), (4x6) e (6x3) rispettivamente, si determini mediante un algoritmo di programmazione dinamica la parentesizzazione ottima del prodotto di matrici che minimizza il numero di moltiplicazioni. Si indichino i passaggi.

3. (2 punti)

Sia dato un albero binario con 11 nodi. Nella visita in preorder si ottiene `preorder` = (100, 10, 55, 25, 13, 3, 30, 14, 0, 20, 90) e in quella `inorder` = (10, 100, 13, 25, 3, 55, 0, 14, 20, 30, 90). Si ricostruisca l'albero binario di partenza, verificando che la visita postorder dà come risultato `postorder` = (10, 13, 3, 25, 0, 20, 14, 90, 30, 55, 100).

4. (2 punti + 1.5 punti)

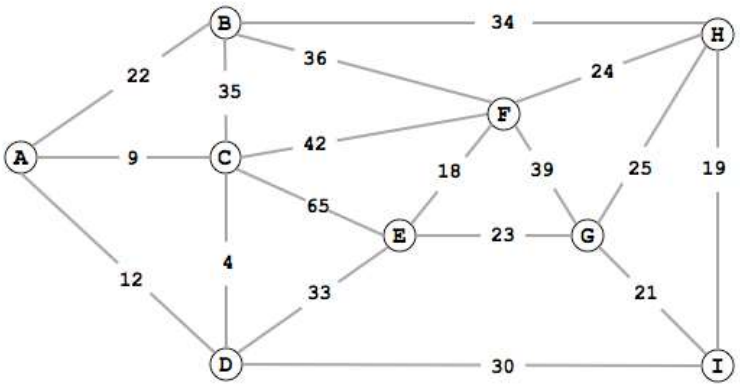
Sia dato il seguente grafo orientato:



se ne effettui una visita in profondità, considerando **0** come vertice di partenza etichettando i vertici con tempo di scoperta/tempo di fine elaborazione (**2 punti**) e gli archi con T, B, F, C (**1.5 punti**). Qualora necessario, si trattino i vertici secondo l'ordine numerico. Si indichino i passaggi.

5. (2 punti)

Dato il seguente grafo non orientato, connesso e pesato, se ne determini un minimum spanning tree applicando l'algoritmo di Prim a partire dal vertice **A**, disegnando l'albero e ritornando come risultato il valore del peso minimo. Si esplicitino i passi intermedi.



03MNO Algoritmi e Programmazione

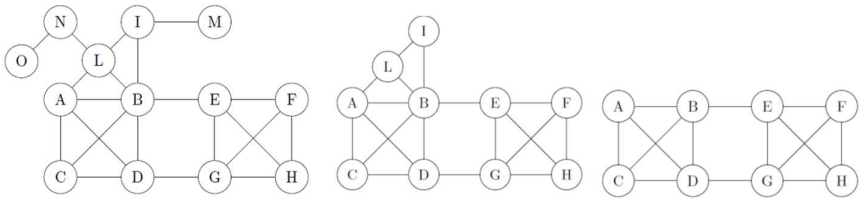
Appello del 28/01/2020 - Prova di programmazione (18 punti)

1. (**18 punti**)

Nella Teoria dei Grafi, dato un grafo G = (V, E) non orientato, semplice e connesso:

- si definisce **k-core** un sottografo massimale di vertici di G tale che ognuno di questi abbia grado ≥ k. Vanno considerati unicamente gli archi appartenenti al sottografo.

Esempio:



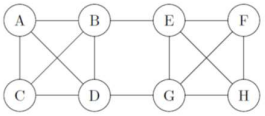
G 2-core di G 3-core di G

Algoritmo:

per calcolare il k-core di G si cancellino tutti i suoi vertici con grado < k. Si noti che la rimozione di un vertice comporta la rimozione di tutti gli archi su di esso incidenti e questo, a sua volta, diminuisce il grado di altri vertici, che quindi possono a loro volta dover essere cancellati. Nell'esempio, se k=2, la rimozione del vertice O di grado 1 rende di grado 1 anche il vertice N, originariamente di grado 2, che quindi deve a sua volta essere rimosso. La rimozione del vertice M di grado 1 rende di grado 2 il vertice I, originariamente di grado 3, che quindi non deve a sua volta essere rimosso dal 2-core, ma che sarà rimosso dal 3-core.

- il grafo G si dice **j-edge-connected** se e solo se è necessario rimuovere almeno j archi per sconnetterlo.

Esempio: il seguente grafo è 2-edge-connected: rimuovendo gli archi (B, E) e (D, G) il grafo diventa non connesso. Rimuovendo un solo arco (qualsiasi) il grafo resta connesso.



Un grafo non orientato, semplice e connesso G = (V, E) è memorizzato in un file mediante l'elenco dei suoi archi con il seguente formato: sulla prima riga compare un intero che indica il numero $|V|$ di vertici del grafo, seguono $|V|$ righe ciascuna delle quali riporta l'identificatore alfanumerico di un vertice (max 10 caratteri), segue un numero indefinito di coppie di identificatori che rappresentano gli archi del grafo. Si assuma corretto il formato del grafo su file.

Si scriva un programma C che, ricevuto il nome di un file con la descrizione del grafo G come argomento sulla riga di comando:

1. legga il grafo G e lo memorizzi in un'opportuna struttura dati
2. legga da tastiera un intero k ≥ 0 e calcoli, se esiste, il **k-core** di G, visualizzando l'elenco dei vertici che vi appartengono
3. legga da tastiera un intero j ≥ 1 e verifichi se G sia **j-edge-connected,** escludendo l'esistenza di insiemi di archi di cardinalità < j in grado di sconnetterlo e individuando, se esiste, almeno un sottoinsieme di archi di cardinalità j che lo sconnetta**.**

Osservazioni e suggerimenti:

si suggerisce, qualora sia necessario cancellare vertici, di marcare nel grafo originale i vertici rimossi, piuttosto che eliminarli dalla struttura dati. Di conseguenza si suggerisce di modificare l'ADT di I classe del grafo per tener conto di questa cancellazione "logica".

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 venerdì 31/01/2020, 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.***

03MNO Algoritmi e Programmazione

Appello del 28/01/2020 - Prova di programmazione (12 punti)

1. (2 punti)

Sia dato un vettore `V` di `N` interi i cui elementi rappresentano, in formato compresso, una sequenza di numeri che deve essere inserita in una matrice `M` di interi di `r` righe e `c` colonne, secondo la strategia row-major, cioè per righe. Per decodificare la sequenza, gli interi del vettore devono essere considerati a coppie (`v[0],v[1]`), (`v[2],v[3]`), (`v[4],v[5]`), etc. Il primo elemento della coppia rappresenta il numero di volte con cui il secondo deve essere inserito in celle adiacenti sulla stessa riga della matrice `M`.

Si scriva una funzione C con il seguente prototipo che riempia la matrice e ne stampi il contenuto:

```
void buildAndPrint(int *V, int N, int **M, int r, int c);
```

Si assuma corretto il contenuto del vettore `V`, quindi compatibile con le dimensioni della matrice `M` già esistente.

Esempio. Siano `r` = 3, `c` = 5, `N` = 14 e `V` = (2, 1, 2, 17, 1, 3, 4, 8, 1, 6, 3, 7, 2, 5): la matrice `M` avrà il seguente contenuto:

$$M = \begin{pmatrix} 1 & 1 & 17 & 17 & 3 \\ 8 & 8 & 8 & 8 & 6 \\ 7 & 7 & 7 & 5 & 5 \end{pmatrix}$$

2. (4 punti)

Due vettori di interi `preorder` e `inorder` di lunghezza `N` uguale e nota contengono il risultato della visita in preorder e inorder di un albero binario. Si scriva una funzione C `buildTree` che, ricevuti come parametri i 2 vettori e la loro lunghezza, costruisca l'albero binario che, se visitato rispettivamente in preorder e inorder, dà come risultati i contenuti dei 2 vettori. Dell'albero binario la funzione restituisca il puntatore di tipo `link` alla radice. Il prototipo della funzione wrapper sia:

```
link buildTree(int *inorder, int *preorder, int N);
```

Si suppongano disponibili:

```
typedef struct node* link;
struct node { int item; link left; link  right; } ;

link NEW(int chiave, link left, link  right) {
  link x = malloc(sizeof *x);
  x->item = chiave; x->left = left; x->right = right;
  return x;
}
```

Suggerimento: si realizzi in C l'algoritmo usato nella soluzione dell'esercizio 3 del compito di Teoria. Si tratta di percorrere in parallelo i 2 vettori, ricordando che nel vettore `preorder` la radice compare per prima rispetto ai suoi sottoalberi e deve essere cercata nel vettore `inorder`, nel quale compare tra i 2 sottoalberi.

3. (6 punti)

Un sistema di monetazione ha `n` tipi di monete, il cui valore è contenuto in un vettore di `n` interi `val`. Un vettore di `n` interi `disp` registra per ogni tipo di moneta quanti sono i pezzi disponibili. Sia dato un resto intero `r`. Si scriva un programma C che, noti `r`, `n`, `val` e `disp`, calcoli, se possibile, il numero monete minimo necessario per erogare tale resto e visualizzi la soluzione. **Non sono ammesse soluzioni greedy.**

Esempio: `n`=3, `val`=(1, 10, 25), `disp`=(10, 3, 2), `r`=30 soluzione ottima 3 monete da 10.