# 03MNO Algoritmi e Programmazione

## Appello del 18/09/2017 - Prova di teoria (12 punti)

**1. (1 punto)**

Sia data la seguente sequenza di coppie, dove la relazione i-j indica che il nodo i è adiacente al nodo j:

1-7, 4-3, 4-7, 6-2, 5-10, 5-6, 0-9, 3-5, 6-9, 10-1

si applichi un algoritmo di on-line connectivity con **weighted** quickunion, riportando a ogni passo il contenuto del vettore e la foresta di alberi al passo finale. I nodi sono denominati con interi tra 0 e 10.

**2. (2 punti)**

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

31 2 21 3 7 43 13 50 16 9 71 12

- la si trasformi in un heap, ipotizzando di usare un vettore come struttura dati. Si riportino graficamente i diversi passi della costruzione dell'heap ed il risultato finale. Si ipotizzi che, alla fine, nella radice dell'heap sia memorizzato il valore massimo
- si eseguano su tale heap i primi 2 passi dell'algoritmo di heapsort.

NB: la sequenza è già memorizzata nel vettore e rappresenta una configurazione intermedia per cui la proprietà di heap non è ancora soddisfatta.

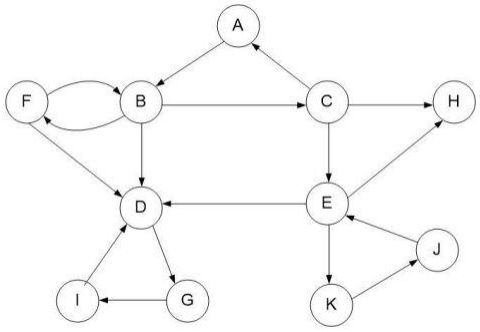
**3. (2 punti)**

Si effettuino, secondo l'ordine specificato, le seguenti inserzioni in foglia su un Interval BST supposto inizialmente vuoto:

[3,11] [4, 7] [1,4] [16,22] [7,12] [2,9] [12,20] [5,16] [9,13]

**4. (2 + 1.5 + 1.5 punti)**

Sia dato il seguente grafo orientato:

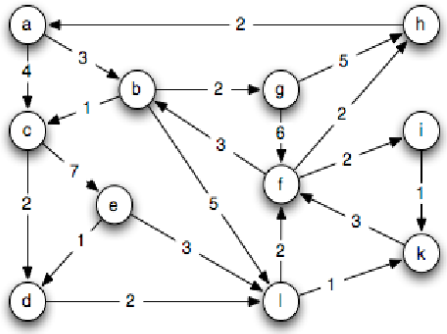



- se ne effettui una visita in profondità, considerando **A** come vertice di partenza (**2 punti**)
- lo si ridisegni, etichettando ogni suo arco come T (tree), B (back), F (forward), C (cross), considerando **A** come vertice di partenza (**1.5 punti**)
- se ne effettui una visita in ampiezza, considerando **A** come vertice di partenza (**1.5 punti**)

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

**5. (2 punti)**

Sia dato il seguente grafo pesato (lo si consideri non orientato trascurando i versi degli archi):




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 18/09/2017 - Prova di programmazione (12 punti)

**1.** (**2 punti**)

Si scriva una funzione caratterizzata dal seguente prototipo:

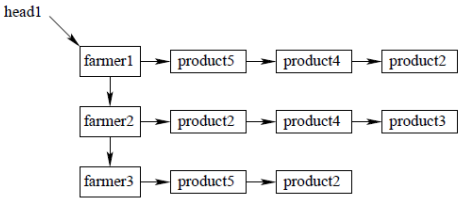
```
int mat_search (char **mat, int r, int c, char *s);
```

La funzione riceve una matrice di caratteri `mat` (con `r` righe e `c` colonne) e una stringa `s`. La funzione conti quante volte la stringa appare in orizzontale o in verticale nella matrice. Nell'esempio seguente la striga `foo` compare 3 volte. Si osservi che sono lecite sovrapposizioni parziali tra stringhe.

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | x | f | o | o | x |
| 1 | y | o | x | z | f |
| 2 | x | o | 2 | f | o |
| 3 | g | 4 | x | a | o |

**2.** (**4 punti**)

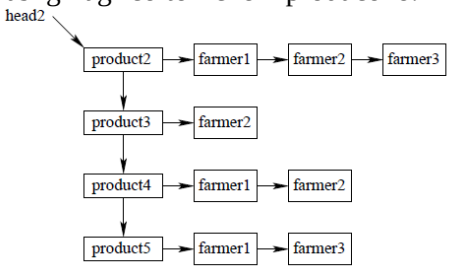
Una database di agricoltori e prodotti è organizzato come lista di liste. La lista principale memorizza in ordine qualsiasi gli agricoltori e per ciascuno di essi in ordine qualsiasi i suoi prodotti. Agricoltori e prodotti sono identificati da stringhe di al massimo 20 caratteri. Alla lista principale si accede mediante il puntatore `head1` e i suoi nodi sono di tipo `node1_t`. I nodi delle liste secondarie sono di tipo `node2_t`.




Si scriva una funzione C caratterizzata dal seguente prototipo:

```
node1_t *list_of_list_invert(node1_t *head1);
```

che crei e ritorni una lista di liste in cui nella lista principale sono memorizzati in ordine qualsiasi i prodotti e nelle liste secondarie in ordine qualsiasi gli agricoltori che li producono.




Si definiscano esplicitamente i tipi dei nodi `node1_t` e `node2_t`. Non è consentito l'uso né di ADT, né di funzioni di libreria sulle liste.

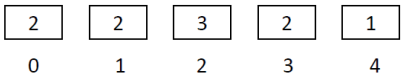
**3.** **(6 punti**)

P piattaforme, disposte in sequenza in linea retta, sono caratterizzate ognuna da un intero positivo. Ognuno di essi indica la lunghezza massima del salto verso destra che può fare il giocatore che si trova su quella piattaforma. Non sono permessi salti all'indietro (verso sinistra).

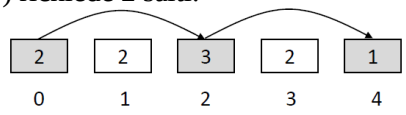
Si scriva una funzione in grado di determinare, se esiste, una sequenza ottima di salti che il giocatore deve fare per spostarsi dalla piattaforma di partenza (indice 0, estremo sinistro) a quella di arrivo (indice P-1, estremo destro). Criterio di ottimalità: minimo numero di salti.

Esempio

Con P = 5 e la seguente configurazione di piattaforme:




la soluzione ottima (da 0 a 2 e da 2 a 4) richiede 2 salti:



# 03MNO Algoritmi e Programmazione

## Appello del 18/09/2017 - Prova di programmazione (18 punti)

In una foresta di eucalipti vivono dei koala, appartenenti a famiglie diverse, che possono essere tra di loro nemiche. Per ogni koala è dato l'insieme degli alberi su cui può vivere. Sono altresì note le coppie di famiglie nemiche. Ogni eucalipto può ospitare un numero massimo noto di koala. Koala di famiglie nemiche non possono vivere sullo stesso eucalipto. I koala e le famiglie sono identificati da stringhe K<intero>, F<intero>, rispettivamente, gli eucalipti da interi non negativi:

- i koala, sono *N*
- gli eucalipti sono *T*
- le famiglie sono *S*
- il numero massimo di koala che un eucalipto può ospitare è *m* ed è unico per tutti gli eucalipti
- c'è certamente posto per tutti i koala ($T \geq N$)
- ogni koala appartiene a 1 e 1 sola famiglia.

Vi sono 4 file di dati di ingresso (assunti corretti):

1. `habitat.txt`: vi sono *N* blocchi. Per ciascuno di essi sulla prima riga compare l'identificativo del koala e il numero di alberi su cui può vivere, poi, uno per riga, gli identificativi degli alberi su cui il koala può vivere.
2. `families.txt`: vi sono *S* blocchi. Per ciascuno di essi sulla prima riga compare l'identificativo della famiglia e il numero di koala che la compongono, poi, uno per riga, gli identificativi dei koala
3. `enemies.txt`: su righe separate compaiono gli identificativi delle coppie di famiglie incompatibili.
4. `solution.txt`: soluzione di cui verificare la validità: su righe separate compaiono le coppie (koala, albero).

Esempio:

| habitat.txt | families.txt | enemies.txt | solution.txt |
|---|---|---|---|
| K7 3 | F2 2 | F2 F4 | K3 2 |
| 2 | K2 | F0 F1 | K0 1 |
| 4 | K1 | ….. | ….. |
| 1 | F0 3 | | |
| K2 2 | K4 | | |
| 0 | K6 | | |
| 6 | K3 | | |
| ….. | ….. | | |

Si scriva un programma C che, ricevuti sulla riga di comando i valori di *N, T, S* e *m*,

- verifichi se la soluzione proposta nel file `solution.txt` rispetti i vincoli del problema
- determini, se esiste, un'allocazione dei koala tra gli alberi che:
  - rispetti i vincoli del problema
  - minimizzi il numero di alberi complessivamente occupati

e la visualizzi a video con lo stesso formato del file `solution.txt`.

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

- ***indicare nell'elaborato e nella relazione (oltre a nome, cognome e numero di matricola) anche il nome del corso per cui si sta sostenendo l'esame ( AP, APA I+II).***
- ***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 giovedì 21/09/2017, alle ore 23:59, 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 discussione o esame orale, <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.***