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

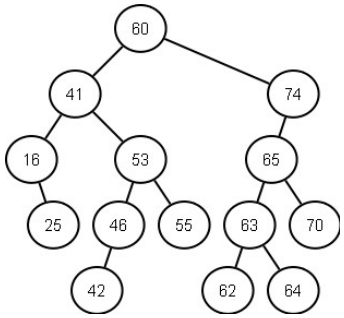
si eseguano i primi 2 passi dell'algoritmo di quicksort per ottenere un ordinamento ascendente, indicando ogni volta il pivot scelto. NB: i passi sono da intendersi, impropriamente, come in ampiezza sull'albero della ricorsione, non in profondità. Si chiede, pertanto, che siano ritornate le 2 partizioni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

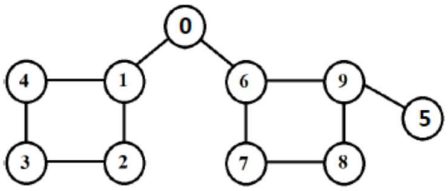
4. (2 punti)

Si partizioni il seguente BST attorno alla chiave 55:



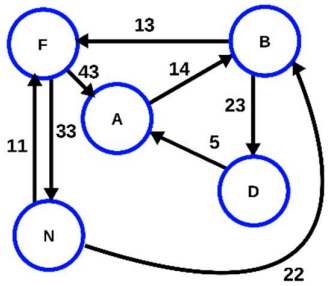
5. (1 punto)

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)

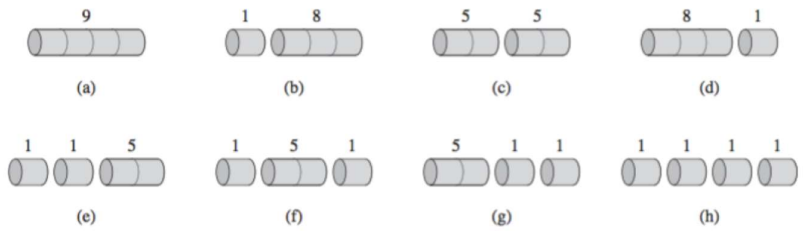
Un nastro lungo `n` (intero) può essere tagliato in pezzi di `m` (intero) diverse lunghezze intere `lun` dove 1 ≤`lun`≤`n`. Ciascuno dei pezzi viene venduto ad un determinato prezzo. Si possono tagliare più pezzi della stessa lunghezza. Un vettore `prezzo` di `m` interi contiene i prezzi per ciascuna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

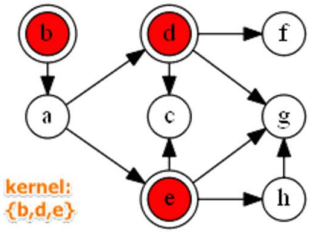
Nella Teoria dei Grafi, dato un grafo orientato G = (V, E), si definisce **Kernel** o **Nucleo** un insieme K $\subseteq$V di vertici di G tale per cui valgono entrambe le seguenti condizioni:

- 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.***

03MNO Algoritmi e Programmazione

Appello del 22/02/2019 - Prova di teoria (12 punti)

1. (2 punti)

Sia data una coda a priorità implementata mediante uno heap di dati. I dati siano formati da una coppia di stringa-intero dove il secondo rappresenta la priorità. Nella radice dello heap si trovi il dato a priorità **minima**. Si inserisca la seguente sequenza di dati nella coda a priorità supposta inizialmente vuota:

(ab,20) (cfd,32) (FF,19) (aAd,51) (rt, 28) (PUy,74) (S,9) (ttt,81) (mn,17) (qwr,41) (ws,37)

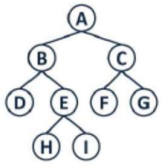
Si riporti la configurazione della coda a priorità dopo ogni inserzione. Al termine si cambi la priorità del dato in posizione 4 in 11 e si disegni la configurazione risultante della coda a priorità.

2. (2.5 punti)

Data la catena di matrici (A_1, A_2, A_3, A_4) di dimensioni (3x7), (7x4), (4x2) e (2x5) 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. (3 x 0.5 punti)

Si visiti il seguente albero binario in preorder, inorder e postorder:

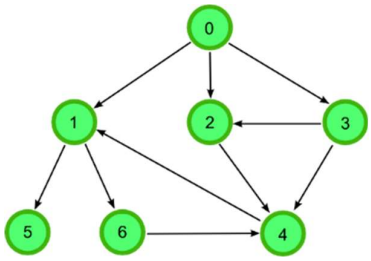


4. (2 punti)

Sia data la sequenza di chiavi intere: 31 124 167 102 98 10 75 107 130 99 17. Si riporti il contenuto di una tabella di hash di dimensione 19, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si usi l'open addressing con double hashing, definendo le opportune funzioni di hash. Si indichino i passaggi.

5. (2 punti)

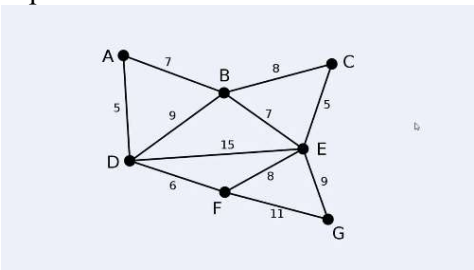
Sia dato il seguente grafo orientato:



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

6. (2punti)

Si determini mediante l'algoritmo di Kruskal l'albero ricoprente minimo per il grafo non orientato, pesato e connesso in figura illustrando i passaggi intermedi del procedimento adottato. Si disegni l'albero e si ritorni come risultato il valore del peso minimo.



03MNO Algoritmi e Programmazione

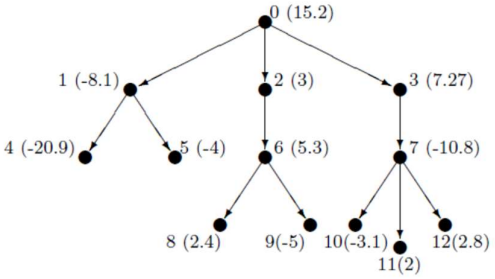
Appello del 22/02/2019 - Prova di programmazione (12 punti)

1. (**2 punti**)
Sia dato un vettore *V* di *N* interi. Si scriva una funzione C che visualizzi un qualsiasi sottovettore proprio di celle contigue la somma dei cui valori sia massima. Un sottovettore si dice proprio se esso contiene un numero di celle strettamente inferiore a quello del vettore da cui deriva.
Esempio: se V= -1, 2, 3, -6, 1, 3, 1 i 2 sottovettori a somma massima 5 sono 2,3 e 1,3,1 entrambi ammissibili come soluzione.

2. (**4 punti**)
Sia dato un albero T di grado `k`a cui si accede tramite il puntatore `root` alla radice. I nodi dell'albero constano di un identificatore intero `i` e di un peso `float` qualsiasi `wi` e dei puntatori di tipo `link` ai sottoalberi. Il peso di un sottoalbero è definito come la somma dei pesi dei suoi nodi. Si definisca opportunamente il tipo `nodo_t` e si scriva una funzione C con il seguente prototipo che ritorni l'identificatore della radice del sottoalbero di peso massimo ed il peso massimo:

```
int maxSum(link root, float *maxwt);
```

Esempio: dato l'albero T di grado `k`=3 in figura, il sottoalbero di peso massimo è radicato in 2 ed ha peso 3 + 5.3 + 2.4 − 5 = 5.7:



3. (**6 punti**)
Sia dato un vettore `V` di `N` interi. Si definisce **sottosequenza** di `V` di lunghezza `k` (`k`≤`N`) un qualsiasi n-upla `Y` di `k` elementi di `V`non necessariamente contigui.Si ricordi che nella n-upla l'ordine conta.
Esempio: se `N`=9 e `V` = [8, 9, 6, 4, 5, 7, 3, 2, 4], una sottosequenza con `k`=4 è `Y` = [9, 6, 7, 2]

La sottosequenza si dice **alternante minore-maggiore** se e solo se valgono entrambe le seguenti condizioni:

- ogni elemento `Y[i]` di indice `i`(nella sottosequenza `Y`) pari è seguito, se esiste, da un elemento `Y[i+1]` di indice dispari e di valore maggiore, cioè tale che `Y[i]<Y[i+1]`
- ogni elemento `Y[i]` di indice `i` dispari è seguito, se esiste, da un elemento `Y[i+1]` di indice pari e di valore minore, cioè tale che `Y[i] > Y[i+1]`.

Esempio: la sequenza 4, 11, 3, 8, 5, 6, 2, 10, 1 è alternante minore-maggiore.

Si scriva una funzione C che, ricevuti come parametri il vettore `V` e l'intero `N` calcoli e visualizzi una qualsisasi sottosequenza alternante minore-maggioredi lunghezza massima.

Esempio: se `N`=9 e `V` = [8, 9, 6, 4, 5, 7, 3, 2, 4] una sottosequenza alternante minore-maggiore di lunghezza massima `k`=6 è `V` = [8, 9, 6, 7, 3, 4].

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 lunedì 25/02/2019, 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, 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 22/02/2019 - Prova di programmazione (18 punti)

1. (**18 punti**)

Un'azienda impiega dipendenti che possono svolgere più di una **tipologia** di lavoro(**o**peraio, **a**mministrativo, **t**ecnico e **i**nformatico) con un certo grado di competenza. Un primo file (*dei dipendenti*) contiene le informazioni sui dipendenti:

- la prima riga riporta il numero N dei dipendenti
- le `N` righe successive contengono i dati sui dipendenti, ognuno descritto (in campi separati da spazi) da:
  - matricola (intero su 4 cifre)
  - nome e cognome (stringhe di al massimo 20 caratteri)
  - competenze (4 interi tra 0 e 100) nelle 4 tipologie (operaio, amministrativo, tecnico o informatico).

Esempio: `1234 Rossi Mario 10 60 30 80` è un dipendente molto adatto al ruolo di informatico (80 su 100), abbastanza al ruolo di amministrativo (60 su 100), poco al ruolo di tecnico (30 su 100), per nulla al ruolo di operaio (10 su 100).

L'azienda è organizzata in `D`divisioni che la Direzione Generale sta ristrutturando. Gli `N` dipendenti vanno ripartiti tra le `D`divisioni, dove svolgerà una sola delle 4 tipologie di lavoro con la relativa competenza. A tal fine, un secondo file (*delle divisioni*) contiene le richieste delle divisioni: la prima riga contiene il numero `D` di divisioni, seguono `D` gruppi di 5 righe: l'`i`-esimo gruppo, corrispondente alla divisione `i`, identificata, su una riga, dalla sigla (una stringa la massimo di 10 caratteri). Seguono 4 righe, associate ognuna a una delle 4 tipologie per le quali la divisione fa una richiesta: detta `j` la riga (la tipologia), questa contiene una terna di interi m_{ij},$cmin_{ij}$ e r_{ij}. (separati da spazi), che rappresentano, rispettivamente (per la divisione `i` e la tipologia `j`), il numero minimo di addetti, la competenza minima totale e competenza ottimale totale richieste.

Si scriva un programma C che, ricevuti sulla linea di comando i nomi di tre file:

- acquisisca in opportune strutture dati le informazioni contenute nei 2 file dei dipendenti e delle divisioni. Si richiede di utilizzare un quasi ADT (`dipendente_t`) per un dipendente e un ADT di prima classe (`divisione_t`) per una divisione. Le collezioni di dipendenti e di divisioni siano realizzate come vettori dinamici(`dipendenti` e `divisioni`) di tali strutture dati. L'ADT `divisione_t` deve contenere la collezione dei dipendenti assegnati alla divisione stessa (a scelta come collezione di item `dipendente_t` o di riferimenti a elementi del vettore `dipendenti`)e la tipologia di lavoro che ogni dipendente svolgerà. Per questo ADT si implementino esplicitamente le funzioni di *creazione, distruzione, acquisizione* della divisione*, inserimento* di un dipendente *e stampa/salvataggio* su file
- legga un terzo file (*delle associazioni*) contenente, su `N` righe, associazioni dipendente-tipologia-divisione, mediante terne `<matricola><tipologia><sigla>` che specificano a quale divisione e con quale tipologia è stato assegnato il dipendente con quella matricola, verifichi se sono soddisfatti i vincoli di numero minimo di addetti e di competenza totale minima richiesti per ogni tipologia di ogni divisione. La tipologia è identificata tramite la lettera iniziale (`o`, `a`, `t`, `i`). Si calcoli inoltre lo scostamento complessivo medio, rispetto alla competenza richiesta (somma degli scostamenti per divisione e per tipologia diviso il numero di divisioni)

$$\Delta_{avg} = \frac{\sum_{i=1}^{D}\sum_{j=1}^{4}|r_{ij} - c_{ij}|}{D}$$

  Per competenza totale c_{ij} della divisione `i` nella tipologia `j` si intende la somma delle competenze in quella tipologia dei singoli dipendenti assegnati alla divisione `i` con competenza `j`.
- calcoli un'attribuzione ottima di dipendenti alle divisioni tale da:
  - soddisfare i vincoli minimi per numero di dipendenti e competenza totale per ogni tipologia di ogni divisione
  - minimizzare lo scostamento complessivo medio Δ_{avg} (si veda definizione al punto precedente).

  Tale attribuzione sia memorizzata su un file di uscita, il cui nome è passato come parametro sulla riga di comando, nel formato una coppia matricola-sigla per ogni riga.
- dopo aver generato la soluzione ottima del punto precedente, la elabori e la memorizzi nel vettore di collezioni (`divisioni`): si ricorda che il tipo`divisione_t` prevede la possibilità di collezionare i dipendenti di una divisione. Si salvi, usando la funzione di *stampa/salvataggio* (di cui al punto 1) dell'ADT `divisione_t`, l'attribuzione ottima su un file di uscita, nello stesso formato del file delle associazioni (il terzo file, usato al punto 2).

03MNO Algoritmi e Programmazione

Appello del 02/07/2019 - 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: 3-6, 6-2, 0-8, 5-3, 4-8, 4-7, 9-8, 3-5, 6-8, 9-0. 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 9.

2. (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:

`"ab" 20  "cfd" 32  "FF" 19  "aAd" 51  "rt" 28`

Si ipotizzi di usare uno heap (priorità massima in radice, vettore di 5 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
aAD					FF			cfd	ab	rt
51					19			32	20	28
3					2			1	0	4

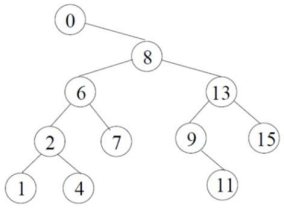
Si disegnino lo heap e il vettore `qp` ai diversi passi dell'inserzione. Al termine si cambi la priorità di `"rt"`da 28 a 60 e si disegnino i corrispondenti heap e vettore `qp`.

3. (3 x 0.5 punti)

Si disegni l'albero corrispondente alla seguente espressione in forma prefissa * A + – B C / + D E F e si trasformi l'espressione in forma infissa e postfissa.

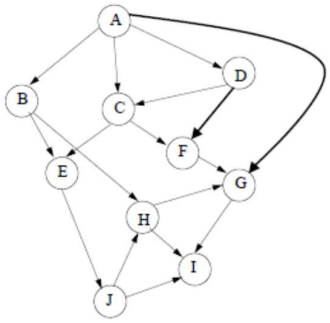
4. (2 punti)

Si aggiungano al seguente BST le informazioni che permettono di realizzate l'operazione di select e, indicando i passaggi, si identifichi la chiave di rango 8:



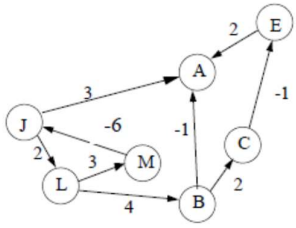
5. (2 punti)

Si ordini topologicamente il seguente DAG. Qualora necessario, si trattino i vertici secondo l'ordine alfabetico e si assuma che la lista delle adiacenze sia anch'essa ordinata alfabeticamente.



6. (2+1 punti)

Si determinino per il seguente grafo orientato pesato mediante l'algoritmo di Bellman-Ford i valori di tutti i cammini minimi che collegano il vertice **J** con ogni altro vertice (**2punti**). Si verifichi l'eventuale presenza di un ciclo a peso negativo (**1 punto**).Si assuma, qualora necessario, un ordine alfabetico per i vertici e gli archi.



03MNO Algoritmi e Programmazione

Appello del 02/07/2019 - Prova di programmazione (12 punti)

1. (2 punti)

Si scriva un programma C che, dato un intero `N`, generi una matrice quadrata `NxN` con tutti 0 sulla diagonale principale, tutti 1 sulle 2 diagonali immediatamente sopra e sotto, tutti 2 su quelle sopra e sotto le precedenti, etc, come nel seguente esempio per `N=5`

0	1	2	3	4
1	0	1	2	3
2	1	0	1	2
3	2	1	0	1
4	3	2	1	0

2. (4 punti)

Una lista linkata, accessibile mediante il puntatore `head1` alla testa, contiene una sequenza di caratteri alfanumerici. Si definisca il tipo link/nodo utilizzati e scriva una funzione C che generi una seconda lista, accessibile mediante il puntatore `head2` alla testa, in cui sono state eliminate le eventuali sottosequenze di elementi delimitati da coppie di parentesi tonde (aperta all'inizio della sequenza, chiusa alla fine della sequenza), sostituendole con un solo elemento asterisco "*". Si assuma che la lista sia corretta, cioè che non possa contenere coppie di parentesi che si "intersecano" e che per ogni parentesi aperta esista una parentesi chiusa. Il prototipo della funzione sia:

```
link purgeList(link head1);
```

Esempio: data la lista

ab (a c g) b e ( ) a (x x) f

la lista di output sarà:

a b (*) b e (*) a (*) f

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

3. (6 punti)

Sia dato un vettore `V` di `N` interi. Si definisce **sottosequenza** di `V` di lunghezza `k` (`k`≤`N`) un qualsiasi n-upla `Y` di `k` elementi di `V` con indici crescenti, ma non necessariamente contigui $i_0, i_1, \cdots, i_{k-1}$. La sottosequenza si dice **bitonica crescente/decrescente** se i suoi elementi sono ordinati prima in modo crescente e poi decrescente. Esempio: la sequenza 4, 5, 9, 7, 6, 3, 1 è bitonica.

Si scriva un programma C che, ricevuti come parametri il vettore V e l'intero N calcoli e visualizzi una qualsisasi sottosequenza bitonica di lunghezza massima.

Esempio: se N=9 e V = [4, 2, 5, 9, 7, 6, 10, 3, 1] una sottosequenza bitonica di lunghezza massima 7 è 4 5 9 7 6 3 1.

Si osservi che una sequenza crescente è bitonica di lunghezza massima, come pure una sequenza decrescente.
Esempio: se N=5 e V = [1, 2, 3, 4, 5], essa stessa è una sequenza bitonica di lunghezza massima 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 <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 05/07/2019, alle ore 23:59, mediante caricamento su Portale. Le istruzioni per il caricamento sono pubblicate sul Portale nella sezione Materiale). <u>QUALORA IL CODICE CARICATO CON LA RELAZIONE NON COMPILI CORRETTAMENTE, VERRÀ APPLICATA UNA PENALIZZAZIONE.</u> 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 02/07/2019 - Prova di programmazione (18 punti)

Un'Agenzia della Mobilità deve pianificare in quali dei Comuni del suo territorio posizionare delle stazioni di ricarica pubbliche per veicoli elettrici. I dati noti sono:

- i Comuni del territorio coinvolto sono `N` e sono identificati dagli interi tra 0 e `N-1`
- il loro numero di abitanti è memorizzato in un vettore `pop` di `N` interi
- le loro distanze reciproche sono memorizzate in una matrice `dist NxN` di interi.

Si vuole ottimizzare la localizzazione delle stazioni di ricarica in base a funzioni obiettivo diverse e indipendenti:

- **funzione obiettivo 1**: data una distanza intera massima `distMax`, determinare il numero minimo `stazMin` di stazioni di ricarica e la loro localizzazione in modo che tutti i Comuni distino meno di `distMax` dalla stazione di ricarica più vicina. In ogni Comune è localizzata al più una stazione
- **funzione obiettivo 2**: dato un numero fisso intero di stazioni di ricarica posizionabili sull'insieme del territorio `numStaz`, dato il numero massimo di stazioni di ricarica localizzabili in ciascun Comune memorizzato in un vettore `stazComune` di interi ≥1, determinare una qualsiasi localizzazione che minimizzi la distanza, pesata in base alla popolazione e al numero di stazioni localizzate in un dato Comune, tra ogni Comune e la stazione di ricarica ad esso più vicina

$$\textbf{min} \sum_{i=0}^{N-1} pop_i * distMinDaStazione_i \,/numStazioniAdistMin_i$$

Esempio: vi siano 5 Comuni caratterizzati da:

dist	0	1	2	3	4
0	0	8	10	7	12
1	8	0	7	9	11
2	10	7	0	10	9
3	7	9	10	0	8
4	12	11	9	8	0

- **funzione obiettivo 1**: se `distMax=8,` il numero minimo di stazioni di ricarica da localizzare è `stazMin=2` e una soluzione è `sol=(1,3)`. La matrice sottostante riporta le distanze minime di ogni Comune dalla soluzione:

dist	0	1	2	3	4
0	0	8	10	7	12
1	8	0	7	9	11
2	10	7	0	10	9
3	7	9	10	0	8
4	12	11	9	8	0

- **funzione obiettivo 2**: se `numStaz=2,` se i vettori `stazComune` e `pop` sono quelli riportati sotto

stazComune	1	1	4	3	2
	0	1	2	3	4

pop	15	5	50	30	25
	0	1	2	3	4

la soluzione `sol=(2,3)` porta ad un valore minimo della funzione obiettivo 2 di:

340 = 7x15/1+7x5/1+0x50/1+0x30/1+8x25/1

La soluzione `sol=(2,2)` porta ad un valore **non** minimo della funzione obiettivo 2 di:

355 = 10x15/2+7x5/2+0x50/2+0x30/2+9x25/2

Si scriva un programma in C che, una volta acquisite le informazioni del problema:

- legga da file, con formato libero, una proposta di allocazione di risorse e verifichi se essa rispetti o meno le condizioni imposte dalla **funzione obiettivo 1**
- individui una soluzione ottima che rispetti i criteri della **funzione obiettivo 1** mediante un algoritmo ricorsivo
- individui una soluzione ottima che rispetti i criteri della **funzione obiettivo 2** mediante un algoritmo ricorsivo

L'introduzione di tecniche di pruning sarà oggetto di valutazione.

3MNO Algoritmi e Programmazione

Appello del 18/09/2019 - Prova di teoria (12 punti)

1. (2 punti)

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

3 31 72 41 71 0 73 1 10 32 19 13 55 91 14 7

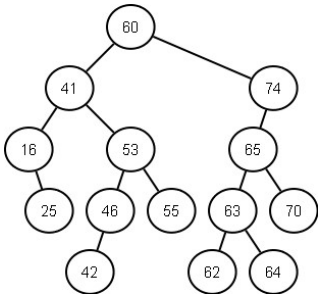
Si eseguano i primi 2 passi dell'algoritmo di quicksort per ottenere un ordinamento **ascendente.** NB: I passi sono da intendersi, impropriamente, come in ampiezza sull'albero della ricorsione, non in profondità. Si chiede, pertanto, che siano ritornate le due partizioni del vettore originale e le due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

2. (2 punti)

Sia data la sequenza di chiavi intere 13 181 267 302 98 110 45 207. Si riporti il contenuto di una tabella di hash di dimensione 17, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si usi l'open addressing con quadratic probing. Si definiscano gli opportuni coefficienti.

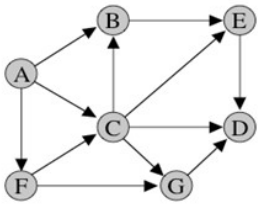
3. (2 punti)

Si inseriscano in **radice** nel BST di figura in sequenza le chiavi 6, 19 e 22 e poi si cancelli la chiave 19. Si disegni l'albero ai passi significativi.



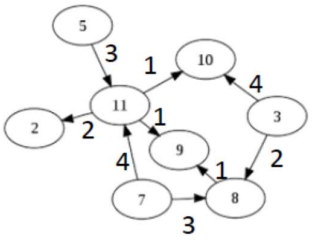
4. (1.5 punti)

Si effettui una visita in ampiezza del seguente grafo orientato, considerando **A** come vertice di partenza. Qualora necessario, si trattino i vertici secondo l'ordine alfabetico.



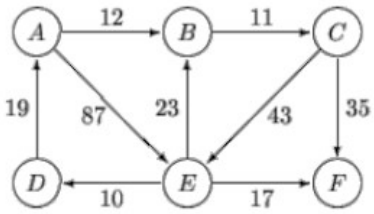
5. (2.5 punti)

Sia dato il seguente DAG pesato: considerando **5** come vertice di partenza, si determinino i cammini **massimi** tra **5** e tutti gli altri vertici. Qualora necessario, si trattino i vertici secondo l'ordine numerico si assuma che la lista delle adiacenze sia anch'essa ordinata numericamente.



6. (**2 punti**)

Sia dato il seguente grafo orientato pesato:



Si determinino i valori di tutti i cammini minimi che collegano il vertice **A** 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 18/09/2019 - Prova di programmazione (12 punti)

1. (**2 punti**)
Sia dato un vettore `V` di `N` interi non negativi. Si supponga di avere a disposizione le seguenti funzioni:

- `int findBound(int *V, int N, int *lp, int *rp);` che ritorna mediante puntatore gli indici sinistro (`lp`) destro (`rp`) del sottovettore di `V` di lunghezza massima contenente soli valori positivi. Il valore intero ritornato è 1 se l'intervallo esiste, 0 in caso contrario (il vettore contiene tutti 0)
- `void dec(int *V, int l, int r);` che decrementa di 1 tutte le celle del vettore `V` comprese tra gli indici di sinistra `l` e di destra `r`, inclusi

Si scriva una funzione C che, con il minimo numero di chiamate a `findBound` e `dec`, trasformi `V` nel vettore di tutti 0. ATTENZIONE: Non occorre realizzare `findBound` e `dec`. Non è un problema di ottimizzazione, la soluzione è univoca.

Esempio: se `V= {0,1,2,0,0,3,4,5}` sono necessarie 7 chiamate (a entrambe le funzioni), se `V={3,1,2,0,0,3,0,5}` ne servono 12, se `V={3,1,2,0,0,3,4,5}` ne servono 9 che danno come risultato a ciascuna chiamata {**2,0,1**,0,0,3,4,5}, {2,0,1,0,0,**2,3,4**}, {2,0,1,0,0,**1,2,3**}, {2,0,1,0,0,**0,1,2**}, {2,0,1,0,0,0,**0,1**}, {**1**,0,1,0,0,0,0,1}, {**0**,0,1,0,0,0,0,1}, {0,0,**0**,0,0,0,0,1}, {0,0,0,0,0,0,0,**0**}.

2. (**4 punti**)
Sia dato un albero binario T, cui si accede tramite il puntatore `root` alla radice. I nodi dell'albero hanno come chiavi stringhe di lunghezza massima `maxC`. Si scriva una funzione C con prototipo

```
linkL tree2List(linkT root, int visit);
```

che visiti l'albero secondo la strategia specificata nel parametro `visit` (1: inorder, 2: preorder, 3: postorder), memorizzando le chiavi in una lista concatenata, di cui ritorna il puntatore alla testa.
Si definiscano il nodo dell'albero e relativo puntatore (tipo `linkT`), il nodo della lista e il relativo puntatore (tipo `linkL`). **Si scriva esplicitamente la funzione di inserzione in lista senza fare uso di funzioni di libreria.**

3. (**6 punti**)
Un grafo non orientato, connesso e pesato è rappresentato come insieme di archi. Gli `nV` vertici del grafo sono identificati come interi tra 0 e `nV-1`. Gli archi sono rappresentati mediante il tipo `Edge`:

```
typedef struct { int v; int w; int wt;} Edge;
```

in cui `v` e `w` sono indici di vertici e `wt` è un peso. Gli `nE` archi del grafo sono memorizzati in un vettore dinamico `Edge *edges` (già allocato e contenente i dati). Un albero ricoprente minimo (minimum-spanning tree) è un albero che tocca tutti i vertici e per cui è minima la somma dei pesi degli archi che vi appartengono. Usando i **modelli del Calcolo Combinatorio** si scriva una funzione C che calcoli il peso di un albero ricoprente minimo (**4 punti**). Si scriva una funzione C che, dato un insieme di archi, verifichi se esso rappresenta un albero ricoprente, non necessariamente minimo (**2 punti**). Si osservi che la seconda funzione è usata nella prima in fase di verifica di accettabilità della soluzione. **È vietato l'uso degli algoritmi classici di Kruskal e Prim.**

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 sabato 21/09/2019, 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 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 18/09/2019 - Prova di programmazione (18 punti)

Sia data una mappa rettangolare di dimensione `NR` x `NC` caratterizzata dalla presenza di celle libere e celle *occupate (per rappresentare, in casi reali, ostacoli o muri)*. Due celle libere della mappa sono considerate adiacenti se condividono un lato; non sono adiacenti celle poste "in diagonale" l'una rispetto all'altra*: la nozione di adiacenza permette quindi di considerare la mappa come un grafo (implicito) nel quale le celle libere rappresentano i vertici, mentre le adiacenze (al massimo 4 per ogni vertice) rappresentano gli archi.*
Sulla mappa è possibile posizionare ***risorse*** nelle celle libere, in modo tale da *"coprire"* un insieme di caselle sufficientemente vicine. Ogni risorsa *"copre"* la cella da essa occupata e tutte le celle libere raggiungibili da questa in al più `k` passi (quindi tutte le celle a distanza $d \leq k$ nel grafo implicito).
Data una distribuzione delle risorse, ***non è possibile che una cella sia "coperta" da più di una risorsa*** (mentre è possibile che non sia coperta da alcuna risorsa). Lo scopo del programma è duplice: da un lato verificare una copertura, dall'altro individuare una copertura ***ottima*** della mappa.

Si scriva un programma in C che:

- legga un primo file di testo `mappa.txt` organizzato come segue:
  - la prima riga contiene una coppia di interi `NR NC`, che rappresentano le dimensioni della mappa
  - segue l'elenco delle caselle occupate, una per riga (al massimo `NR` x `NC` righe), rappresentate dalle coordinate `X Y`.
- legga un secondo file `proposta.txt` e valuti se esso rappresenta una copertura ammissibile rispetto alle regole di cui sopra, **senza** alcun criterio di ottimalità. Il file sia organizzato come segue:
  - la prima riga contiene il valore intero `k` (distanza massima di copertura) e il numero (intero) `Z` di risorse presenti
  - seguono `Z` coppie `X Y`, una per riga, a rappresentare le coordinate della posizione di ciascuna risorsa
  - seguono `NR` righe di `NC` interi separati da spazi a rappresentare la mappa con la copertura associata alla soluzione proposta. Gli interi sono nell'intervallo `[0..Z]`: 0 indica una cella *non coperta* (indifferentemente libera o occupata/ostacolo) mentre un numero `i` diverso da `0` indica la copertura della cella da parte dell'`i`-esima risorsa (risorse numerate a partire da 1). *Per le dimensioni della mappa e la collocazione delle caselle occupate occorre far riferimento al file `mappa.txt`.*
- nel caso la valutazione di ammissibilità del punto precedente fallisca ed esista una soluzione ammissibile con le stesse risorse, la generi e la memorizzi su di un file `corretto.txt` con lo stesso formato
- individui, se possibile, una soluzione ottima usando ***esattamente `Z` risorse***: `Z` sia ricevuto in input oppure come argomento al main. La soluzione è ottima se massimizza il numero di caselle coperte
- individui una soluzione ottima ***avente il minor numero possibile di risorse*** (in sostanza, a parità di copertura massima di caselle, si sceglie la soluzione che usa meno risorse)

NOTE: *Non sono ammissibili configurazioni in cui una risorsa copra una casella occupata (ostacolo/muro presente nella mappa) o già coperta da un'altra risorsa. La copertura include obbligatoriamente tutte le celle che rispettino il criterio di raggiungibilità: non è possibile escludere arbitrariamente (ad esempio per evitare sovrapposizioni) una o più celle dalla zona di copertura di una risorsa.*
A seguire, un esempio per i file `mappa.txt` e `proposta.txt`. Per rendere più immediato l'esempio, alla rappresentazione del secondo file si è aggiunta una visualizzazione della mappa con ulteriori dettagli grafici: le celle con sfondo grigio sono quelle occupate (quindi inutilizzabili a causa di ostacoli/muri). I numeri in grassetto e sottolineati (**i**) rappresentano la cella in cui ogni `i`-esima risorsa è posizionata.

mappa.txt	proposta.txt	Mappa corrispondente a proposta.txt
6 5 1 1 1 4 3 2 4 3	2 2 2 1 4 4 0 1 0 0 0 1 1 1 0 0 1 1 1 1 2 1 1 1 2 2 0 1 0 2 2 0 0 0 2 2	0 1 0 0 0 1 1 1 0 0 1 **1** 1 1 2 1 1 1 2 2 0 1 0 2 **2** 0 0 0 2 2

Si noti, che la configurazione d'esempio riportata in `proposta.txt` **NON** è valida. Le risorse occupano correttamente celle libere, e non vi sono sovrapposizioni nella copertura delle aree. Per via dell'ostacolo, la risorsa 1 non può coprire le celle in `(1,1)` e in `(3,2)` e la risorsa 2 non può coprire la cella in `(4,3)`. La risorsa `1` inoltre non può coprire la cella `(0,1)` in quanto troppo lontana (distanza 4). Per queste ragioni la configurazione è errata.

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)

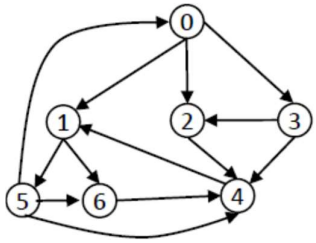
Data la catena di matrici (A_1, A_2, A_3, A_4) di dimensioni (2x5), (5x4), (4x6) e (6x3) rispettivamente, si determini mediante un algoritmo di programmazione dinamica la parentesizzazione ottima del prodotto di matrici che minimizza il numero di moltiplicazioni. Si indichino i passaggi.

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)

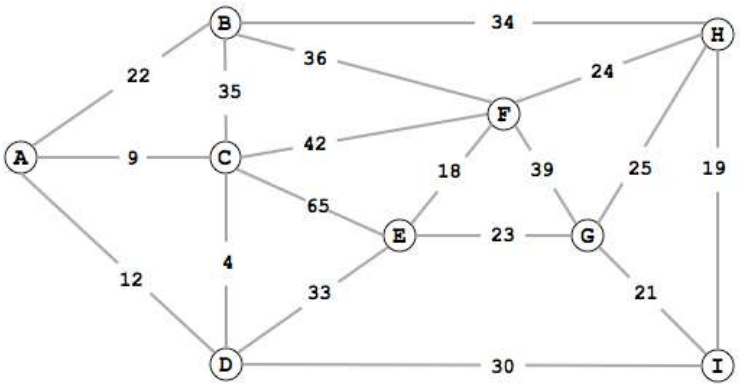
Sia dato il seguente grafo orientato:



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

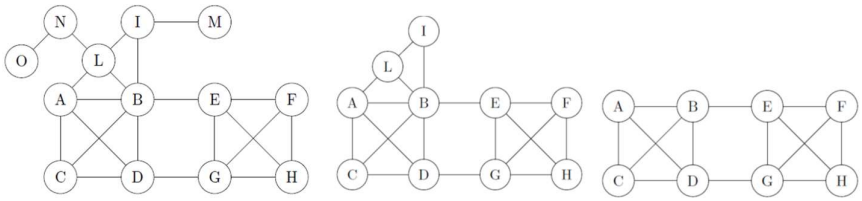
Appello del 28/01/2020 - Prova di programmazione (18 punti)

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:



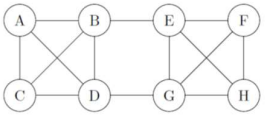
G 2-core di G 3-core di G

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.

03MNO Algoritmi e Programmazione

Appello del 21/02/2020 - 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: 2-6, 5-2, 0-7, 4-3, 3-8, 3-7, 9-8, 2-5, 5-8, 8-0. 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 9.

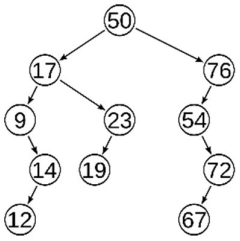
2. (2.5 punti)

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

$$T(n) = 3T(n/3) + n/3 \qquad n>1$$
$$T(1) = 1 \qquad n=1$$

3. (2 punti)

Si inseriscano in **radice** nel BST di figura in sequenza le chiavi 6, 29 e 22 e poi si cancelli la chiave 19. Si disegni l'albero ai passi significativi.



4. (2 punti)

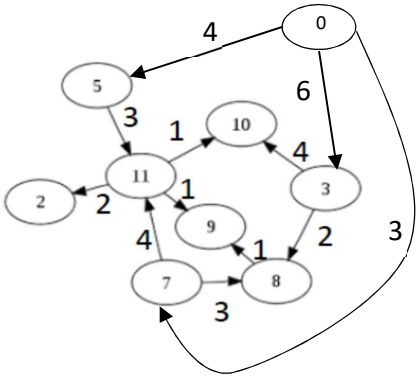
Sia data una coda a priorità implementata mediante uno heap di dati. I dati siano formati da una coppia di stringa-intero dove il secondo rappresenta la priorità. Nella radice dello heap si trovi il dato a priorità **massima**. Si inserisca la seguente sequenza di dati nella coda a priorità supposta inizialmente vuota:

(ab,24) (cfd,62) (FF,39) (aAd,51) (rt, 81) (PUy,74) (S,19) (ttt,18) (mn,27) (qwr,61) (ws,59)

Si riporti la configurazione della coda a priorità dopo ogni inserzione. Al termine si cambi la priorità del dato che si trova nello heap all'indice 4 in 11 e si disegni la configurazione risultante della coda a priorità.

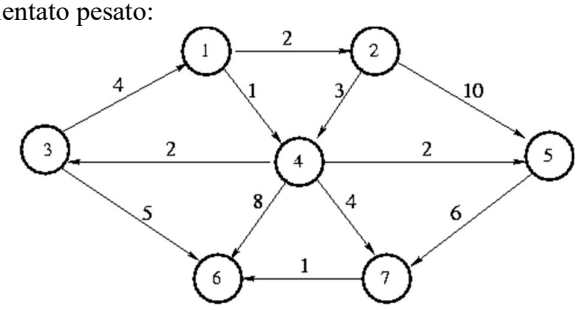
5. (2.5 punti)

Sia dato il seguente DAG pesato: considerando **0** come vertice di partenza, si determinino i cammini **massimi** tra **0** e tutti gli altri vertici. Qualora necessario, si trattino i vertici secondo l'ordine numerico si assuma che la lista delle adiacenze sia anch'essa ordinata numericamente.



6. (2 punti)

Sia dato il seguente grafo orientato pesato:



Si determinino i valori di tutti i cammini minimi che collegano il vertice **3** con ogni altro vertice mediante l'algoritmo di Dijkstra.

3MNO Algoritmi e Programmazione

Appello del 21/02/2020 - Prova di programmazione (18 punti)

1. (**18 punti**)

Un ristorante offre una selezione di piatti tra cui è possibile scegliere, riportati in un file testuale `piatti.txt`. Nella prima riga compare il numero `N` di piatti. Su ognuna delle `N` righe successive compare la descrizione di ogni piatto come quaterna `<nome> <portata> <tipologia> <costo>`. I primi tre campi sono stringhe senza spazi di massimo 100 caratteri. Il quarto campo è un numero positivo con al massimo 2 cifre decimali.

Esempio di `piatti.txt`:

10

Carbonara	Primo	Pasta	8.50
Amatriciana	Primo	Pasta	7.80
Caprese	Contorno	Vegetariano	5.50
VerdureGrigliate	Contorno	Vegetariano	4
Nizzarda	Contorno	NonVegetariano	8.25
FilettoAllaWellington	Secondo	Carne	17
Opera	Dessert	Dolce	4.50
Tiramisu	Dessert	Dolce	4.00
Cotoletta	Secondo	Carne	6.50
VitelloTonnato	Antipasto	NonVegetariano	4.00

Il ristorante vuole generare l'elenco di tutti i possibili menu di `P` piatti, dove `P` è un numero fisso. Per menu si intende un elenco di `P` piatti in cui:

- l'ordine dei piatti non conta. In altri termini tutte le permutazioni di piatti dello stesso menu sono identiche e basta considerarne una sola
- un piatto può eventualmente comparire due volte (per indicare un "bis").

Si scriva un programma che, ricevuto come argomento sulla riga di comando il valore `P`:

- acquisisca dal file `piatti.txt` l'elenco dei piatti disponibili, caricandoli in un vettore `ElencoPiatti`. I piatti sono ordinati secondo l'ordine con cui compaiono nel vettore. Un piatto x è minore di un piatto y se x compare prima di y nel file e quindi nel vettore
- utilizzi un quasi ADT per il menu, contenente l'elenco dei `P` piatti del menu (interi, indici nel vettore di cui al punto precedente) e il prezzo complessivo del menu. I piatti vanno ordinati in base al criterio di cui sopra
- contenga una funzione di confronto tra menu `MENUcompare` che, dati due menu, li confronti in base al prezzo e, a parità di prezzo, in base all'elenco dei piatti: un menu A precede un menu B se il prezzo di A è minore del prezzo di B. A parità di prezzo si confrontano per ognuno dei due menu i piatti che compaiono per primi in essi. Se persiste la parità si procede con la seconda coppia di piatti e così via
- generi, mediante funzione ricorsiva basata su un opportuno modello del calcolo combinatorio, l'elenco di tutti i possibili menu, immagazzinandoli in un BST, ordinato secondo la funzione `MENUcompare`
- stampi i menu disponibili (secondo l'ordine crescente individuato dalla funzione `MENUcompare`), mediante una visita del BST. Per ogni menu si stampino le informazioni complete (quaterna) di ogni piatto.

Nota: le funzioni di creazioni e di inserimento nel BST possono essere considerate di libreria. E' richiesta la funzione di visita con stampa dei dati.

Esempio: se `P`=5, i due menu A e B seguenti presentano bis di piatto, i piatti sono ordinati secondo l'ordine del file, sono a pari prezzo (25,00) il primo precede il secondo nella `MENUcompare` in quanto il piatto `Tiramisu` precede il piatto `VitelloTonnato` in `piatti.txt` (i piatti precedenti sono identici):

Menu A	**Menu A**
Carbonara Primo Pasta 8.50	Carbonara Primo Pasta 8.50
VerdureGrigliate Contorno Vegetariano 4.00	VerdureGrigliate Contorno Vegetariano 4.00
VerdureGrigliate Contorno Vegetariano 4.00	VerdureGrigliate Contorno Vegetariano 4.00
Opera Dessert Dolce 4.50	Opera Dessert Dolce 4.50
Tiramisu Dessert Dolce 4.00	VitelloTonnato Antipasto NonVegetariano 4.00

03MNO Algoritmi e Programmazione

Appello del 21/02/2020 - Prova di programmazione (12 punti)

1. (**2 punti**)
Sia data una matrice di caratteri `A` di dimensioni `n` × `m` ed una stringa. Si scriva una funzione C che conti quante volte la stringa è contenuta nelle righe e nelle colonne della matrice. La stringa va considerata in orizzontale da sinistra a destra o in verticale dall'alto al basso. Il prototipo della funzione sia:

```
int conta(char **matrice, int n, int m, char *stringa);
```

Esempio: la stringa `cat` è contenuta 3 volte nella seguente matrice `A` di `n` = 4 righe e `m` = 5 colonne (nella seconda colonna, nella quarta colonna e nella quarta riga):

$$A = \begin{pmatrix} x & c & e & c & a \\ w & a & e & c & q \\ d & t & p & a & z \\ p & c & a & t & f \end{pmatrix}$$

2. (**4 punti**)
Sia data una lista concatenata `L`. Sia dato un intero `k`. Si scriva una funzione C che scambi il `k`-esimo nodo dalla testa della lista con il `k`-esimo nodo dalla coda della lista

```
void swap(list L, int k);
```

Oltre che l'implementazione della funzione `swap`, 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.** Negli scambi è lecito o scambiare i valori o scambiare i nodi

Esempio: se la lista contiene 10, 20, 30, 40, 50, 60, 70 e k=0 oppure k=6, dopo lo scambio conterrà **70**, 20, 30, 40, 50, 60, **10**. Se la lista contiene 10, 20, 30, 40, 50, 60, 70 e k=2, dopo lo scambio conterrà 10, 20, **50**, 40, **30**, 60, 70. Per k > 6 lo scambio non avviene.

3. **(6 punti**)
Sia data un insieme S = $\{S_0, S_1, \ldots, S_{n-1}\}$ di stringhe di lettere maiuscole. Si determini mediante un algoritmo ricorsivo il numero massimo di stringhe appartenenti ad S mutuamente disgiunte, cioè che non presentano nessuna lettera in comune.

Esempio: dato un insieme S che contiene le seguenti 7 stringhe
S_0 = ABGCIEF, S_1 = BA, S_2 = CD, S_3 = FE, S_4 = GHBD, S_5 = JKLGHI, S_6 = FK

il numero massimo di stringhe mutuamente disgiunte è 4. Anche se le specifiche non richiedono che esse siano elencate, in quanto basta il loro numero, per una migliore comprensione si osservi che esse sono S_1 = BA, S_2 = CD, S_3 = FE, S_5 = JKLGHI.

La funzione abbia come prototipo:

```
int disgiunte(char **stringhe, int n);
```

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 lunedì 24/02/2020, 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.***