02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 06/02/2014 - Prova di teoria (12 punti)

1. (**2 punti**)
Si risolva la seguente equazione alle ricorrenze mediante il metodo dello sviluppo (unfolding):

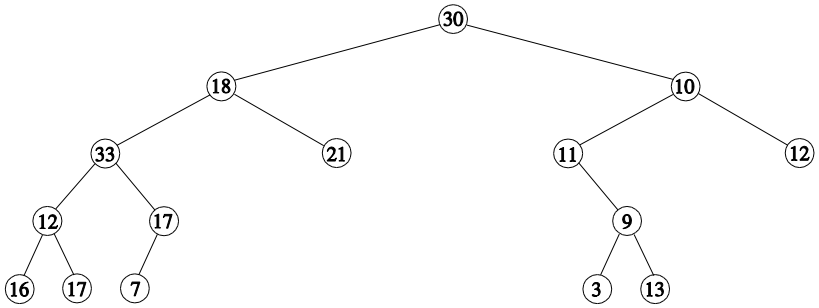
$$T(n) = 8T(n/2) + n^3 \qquad n \geq 2$$
$$T(1) = 1$$

2. (**2 punti**)
Sia data la sequenza di interi, supposta memorizzata in un vettore:

12 29 9 11 21 33 22 13 15 19 25 30

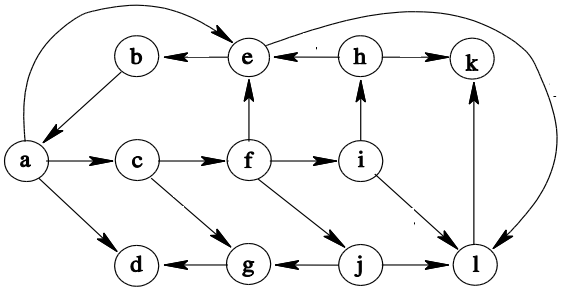
si eseguano i primi 2 passi dell'algoritmo di quicksort per ottenere un ordinamento ascendente, indicando ogni volta il pivot scelto. NB: i passi sono da intendersi, impropriamente, come in ampiezza sull'albero della ricorsione, non in profondità. Si chiede, pertanto, che siano ritornate le 2 partizioni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

3. (**3 x 0.5 punti**)
Si visiti in pre-order, in-order e post-order il seguente albero binario. In caso di chiavi ripetute, distinguere le instanziazioni con pedici.



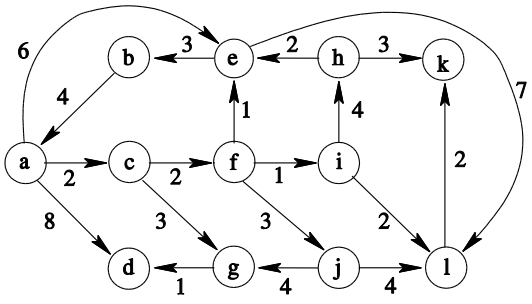
4. (**2 punti**)
Sia data la sequenza di chiavi $S_1IDE_1E_2F_1F_2E_3CTS_2$, dove ciascun carattere è individuato dal suo ordine progressivo nell'alfabeto (A=1, ..., Z=26) con eventuale pedice. Si riporti la struttura di una tabella di hash di dimensione 23, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si supponga di utilizzare l'open addressing con double hashing utilizzando le funzioni di hash $h_1(k) = k \bmod 23$, $h_2(k) = 1 + (k \bmod 21)$.

5. **(2.5 punti)**
Dato il seguente grafo orientato:



se ne determinino mediante l'algoritmo di Kosaraju le componenti fortemente connesse. Si consideri **a** come vertice di partenza e, qualora necessario, si trattino i vertici secondo l'ordine alfabetico.

6. **(2 punti)**
Sul seguente grafo orientato e 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.



02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 06/02/2014 - Prova di programmazione (18 punti)

L'**aritmetica verbale** è un gioco matematico pubblicato per la prima volta da Henry Dudeney nel numero del luglio 1924 della rivista Strand Magazine. Si supponga di avere dei numeri interi le cui cifre sono "criptate" sotto forma di lettere e di conoscere, sempre in forma criptata, il risultato di un'operazione aritmetica tra 2 interi criptati, come nel seguente esempio:

	S	E	N	D	+
	M	O	R	E	=
M	O	N	E	Y	

Il gioco consiste nell'identificare per ciascuna lettera la cifra decimale tale per cui sia soddisfatta l'operazione. Nell'esempio considerato con la seguente corrispondenza

O = 0, M = 1, Y = 2, E = 5, N = 6, D = 7, R = 8 e S = 9

si ottiene infatti

	9	5	6	7	+
	1	0	8	5	=
1	0	6	5	2	

Ogni lettera presente è associata a una e una sola cifra decimale. Per semplicità si ipotizzi che l'unica operazione ammessa sia quella di somma tra 2 operandi interi criptati.

Assunzioni:

- le stringhe contengono solo caratteri alfabetici tutti maiuscoli o tutti minuscoli
- la cifra più significativa non può corrispondere allo 0
- le prime due stringhe non hanno necessariamente la stessa lunghezza, la terza stringa ha lunghezza coerente con quella delle prime 2, data l'operazione di somma
- le prime 2 stringhe hanno una lunghezza massima pari a 8, la terza coerente con quelle delle prime due
- nelle stringhe non compaiono più di 10 lettere distinte.

Si scriva un programma C che, dopo aver letto da tastiera le tre stringhe di caratteri, funzioni alternativamente come:

- verificatore di una soluzione fornita dall'utente da tastiera
- generatore automatico di una soluzione.

Nel primo caso, acquisita da tastiera in un formato a scelta la soluzione dell'utente, ne verifichi la correttezza. Nel secondo caso il programma calcoli e visualizzi una corrispondenza lettere-cifre decimali corretta.

Suggerimenti:

- il problema di identificare una corrispondenza lettere-cifre che soddisfa l'operazione si può risolvere con ricorsione con backtrack
- la condizione di accettazione della soluzione può essere verificata (senza pruning preliminare) una volta raggiunta la foglia dell'albero delle ricorsioni
- è possibile, in fase di accettazione, realizzare l'operazione di somma lavorando su singole cifre e propagando riporti oppure mediante conversione da vettore di cifre simboliche a intero e somma tra interi.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 06/02/2014 - Prova di programmazione (12 punti)

1. (**2 punti**)
Sia dati 3 vettori ordinati di interi. Sia nota la lunghezza di ciascuno di essi. Si realizzi una funzione C `merge3` che ritorni come risultato il vettore ordinato di interi che deriva dalla fusione dei 3 vettori di ingresso. Il prototipo della funzione è:

```
int *merge3 (int *a, int *b, int *c, int na, int nb, int nc);
```

dove `a`, `b` e `c` sono i 3 vettori già ordinati, `na`, `nb` e `nc` le loro lunghezze. Il vettore ritornato, di dimensione `na+nb+nc`, va allocato dinamicamente e generato mediante un'unica operazione di merge simultanea sui tre vettori. In altri termini è vietato fondere i primi 2 e poi il risultato con il terzo.

2. (**4 punti**)
Si scriva una funzione C in grado di inserire in una lista ordinata alcuni dati anagrafici di una persona, dati da una coppia cognome e nome, entrambe stringhe di caratteri di lunghezza massima 20. La chiave di ordinamento sia il cognome e, in caso di cognomi identici, il nome. La funzione, di cui il prototipo è:

```
int inserisciInOrdine (lista_t *lista, char *cognome, char *nome);
```

riceve una lista (tipo `lista_t`, corrispondente a un ADT di prima categoria), le due stringhe corrispondenti a cognome e nome, ritorna, come intero, un valore vero nel caso di inserimento corretto, falso nel caso di dato già presente. Si definiscano il tipo `lista_t` e il tipo per il generico dato in lista, tenendo conto che, per ogni persona, vanno generati per la lista duplicati allocati dinamicamente (e separati) per cognome e nome. Il codice deve essere scritto esplicitamente, in altri termini è vietato ricorrere a funzioni di libreria.

3. **(6 punti**)
Sia dato un vettore di float di lunghezza nota n. Ogni float rappresenta un movimento su un conto bancario: se positivo, è un'entrata, se negativo un'uscita. Si assuma che tutti i movimenti siano distinti. Dato un ordine per i movimenti, per ogni movimento, si definisce saldo corrente il valore ottenuto sommando algebricamente al saldo precedente (inizialmente 0) l'importo dell'operazione. Per ogni ordine di movimenti esisterà un saldo corrente massimo e un saldo corrente minimo, mentre il saldo finale sarà ovviamente lo stesso, qualunque sia l'ordine.
Si scriva una funzione C che, utilizzando un algoritmo ricorsivo, determini l'ordinamento del vettore tale da minimizzare la differenza tra saldo corrente massimo e saldo corrente minimo. La funzione deve ritornare la sequenza ordinata sul vettore di partenza.

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).***
- ***É consentito utilizzare chiamate a funzioni <u>standard</u>, quali ordinamento per vettori, inserzione/estrazione/ricerca relative a FIFO, LIFO, liste, BST, tabelle di hash e altre strutture dati, considerate come librerie esterne. Gli header file delle librerie utilizzate devono essere allegati all'elaborato. Le funzioni richiamate, inoltre, dovranno essere incluse nella versione del programma allegata alla relazione.***
- ***Consegna delle relazioni (per entrambe le tipologie di prova di programmazione): entro martedì 11/02/2014, alle ore 24:00, via e-mail all'indirizzo: danilo.vendraminetto@polito.it, usando come subject (oggetto) la stringa APA#<m>, essendo <m> il proprio numero di matricola. L'allegato alla mail deve essere costituito da <u>un unico file</u>: un archivio compresso, contenente sia il codice corretto, sia la relazione (<u>NO eseguibili</u>). QUALORA IL CODICE SPEDITO 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.***

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 24/02/2014 - 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:

3-10, 5-3, 1-5, 7-2, 3-8, 5-4, 0-9, 1-5, 8-9, 10-4

si applichi un algoritmo di on-line connectivity con 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. (1 punto)

Si ordini in maniera ascendente mediante counting-sort il seguente vettore di interi:

3 4 5 2 6 5 4 2 8 1 4 1 8 0 4

Si indichino le strutture dati usate nei passi intermedi.

3. (2 punti)

Sia data una coda a priorità inizialmente vuota implementata mediante uno heap. Sia data la sequenza di interi e carattere *: 10 22 29 71 * * 58 * 34 9 31 * * 14 * 41 27 18 dove ad ogni intero corrisponde un inserimento nella coda a priorità e al carattere * un'estrazione con cancellazione del massimo. Si riporti la configurazione della coda a priorità dopo ogni operazione e la sequenza dei valori restituiti dalle estrazioni con cancellazione del massimo.

4. (**2 punti**)

Sia dato un albero binario con 12 nodi. Nella visita si ottengono le 3 seguenti sequenze di chiavi:

preorder	33	41	23	1	7	11	17	19	13	3	27	5
Inorder	23	41	7	1	11	33	13	19	17	27	3	5
postorder	23	7	11	1	41	13	19	27	5	3	17	33

Si disegni l'albero binario di partenza.

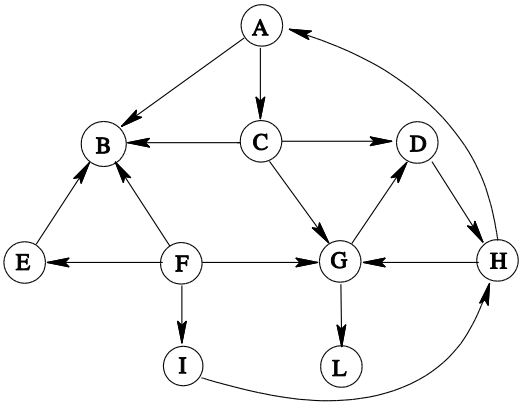
5. (1 punto)

Si supponga di aver memorizzato tutti i numeri compresi tra 1 e 1000 in un albero di ricerca binario e che si stia cercando il numero 531. Quali tra queste **non** possono essere le sequenze esaminate durante la ricerca? Perché?

500	550	510	540	533	539	532	531				
500	600	580	570	510	520	530	540	535	531		
570	100	200	300	400	500	660	550	520	525	530	531
610	30	600	60	588	117	519	299	301	477	531	

6.

Sia dato il seguente grafo orientato:



- se ne effettui una visita in profondità, considerando **A** come vertice di partenza. Si etichettino indicando per ognuno di essi i tempi di scoperta e di fine elaborazione nel formato tempo1/tempo2 (**2 punti**)
- se ne effettui una visita in ampiezza, considerando **a** come vertice di partenza (**1.5 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**).

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

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

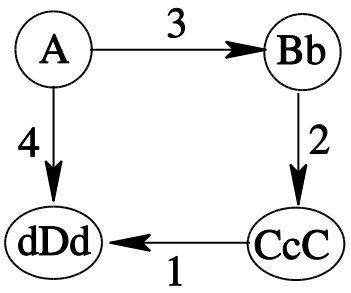
Appello del 24/02/2014 - Prova di programmazione (18 punti)

Un grafo orientato pesato è memorizzato su file mediante l'elenco dei suoi archi, con il seguente formato:

idV1 val idV2

che indica che il vertice con identificatore idV1 è connesso con un arco orientato di peso val (valore intero e positivo) al vertice idV2.
Si osservi che il numero di vertici del grafo è indefinito e che ciascun vertice è individuato mediante un identificatore alfanumerico di lunghezza massima uguale a 20 caratteri. Si può assumere che non esistano archi duplicati.
La figura seguente riporta un esempio di file e della relativa rappresentazione grafica del grafo memorizzato:

```
A 3 Bb
Bb 2 CcC
A 4 dDd
CcC 1 dDd
```



Si vuole scrivere un programma C in grado di:

- ricevere il nome di un file contenente la descrizione del grafo sulla riga di comando
- leggere il grafo e memorizzarlo in un'opportuna struttura dati
- verificare se il grafo è regolare o meno. Un grafo orientato si dice regolare se e solo se tutti i vertici hanno lo stesso in_degree e out_degree e questi sono uguali tra di loro
- effettuare un'iterazione all'interno della quale viene letto l'identificatore di un vertice:
  - se l'identificatore letto è uguale a "`fine`" il programma deve terminare
  - in caso contrario il programma deve determinare il cammino semplice con sorgente nel vertice indicato per il quale la somma dei pesi degli archi è massima. Per tale cammino si devono visualizzare gli archi attraversati (come coppia di identificatori alfanumerici dei vertici su cui insistono e peso dell'arco) e il peso totale.

Suggerimento:
Poiché il numero di vertici non è noto a priori, si suggerisce di calcolarlo tramite lettura preliminare del file e/o caricamento in una tabella di simboli.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 24/02/2014 - Prova di programmazione (12 punti)

1. (**2 punti**)

Sia dato un vettore di N interi. Si consideri il vettore rappresentato orizzontalmente con indice 0 a sinistra e indice N-1 a destra. Una rotazione elementare del vettore verso destra (sinistra) consiste nello spostare tutti i dati di una posizione verso destra (sinistra), portando il dato di indice N-1 (0) in posizione 0 (N-1). Una rotazione di D posizioni può essere definita concettualmente come l'effetto di D rotazioni elementari, Si realizzi una funzione C di rotazione a destra o a sinistra di D posizioni, dove D è un intero (positivo per rotazione a destra, negativo a sinistra).

Ad esempio, che una rotazione, con N=10 e D=3, sul vettore: 5,4,10,-1,7,20,11,-3,6,34, produrrebbe il vettore: -3,6,34,5,4,10,-1,7,20,11.

E' vietato realizzare la rotazione di D posizioni come la sequenza di D rotazioni da 1 posizione ciascuna, che avrebbe costo O(N*D). Si richiede invece un algoritmo O(N), che utilizzi un vettore ausiliario di D posizioni, allocato dinamicamente, in cui depositare temporaneamente D elementi, Dopo aver spostato gli altri N-D elementi, ciascuno nella sua casella destinazione, anche i D elementi salvati vanno collocati nella loro destinazione finale.

2. (**4 punti**)

Una matrice si dice "sparsa" quando la maggior parte delle caselle contiene un valore nullo. Nel caso di una matrice sparsa, può' risultare conveniente l'utilizzo di una struttura dati che allochi dinamicamente memoria per le sole caselle non nulle.

Si realizzi in C, per una matrice sparsa di numeri (tipo float), gestita come ADT di prima categoria, la funzione di prototipo

```
MATRwrite (matr_t *M, int r, int c, float val);
```

che inserisca alla riga r e alla colonna c il valore val. Il tipo matr_t è una struct che, oltre a contenere, come interi, le due dimensioni della matrice (NR, numero di righe, e NC, numero di colonne) punta a un vettore (di dimensione NR) di puntatori a righe, ognuna realizzata mediante una lista, contenente gli elementi non nulli della riga corrispondente). Si definisca matr_t, si definisca poi il tipo struct usato per le liste e si scriva infine la funzione MATRwrite.

3. (**6 punti**)

Si scriva in C una funzione ricorsiva in grado di generare tutte le stringhe di lunghezza N costituite dalle 5 vocali maiuscole 'A', 'E', 'I', 'O', 'U' con i seguenti vincoli:

- il valore di N sia un parametro della funzione e sia $N \geq 5$
- nella stringa generata ogni vocale compaia almeno una volta.

All'i-esimo livello di ricorsione:

- la funzione gestisca l'i-esima cella della stringa della soluzione
- le scelte siano le 5 vocali maiuscole.

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).***
- ***È consentito utilizzare chiamate a funzioni <u>standard</u>, quali ordinamento per vettori, inserzione/estrazione/ricerca relative a FIFO, LIFO, liste, BST, tabelle di hash e altre strutture dati, considerate come librerie esterne. Gli header file delle librerie utilizzate devono essere allegati all'elaborato. Le funzioni richiamate, inoltre, dovranno essere incluse nella versione del programma allegata alla relazione.***
- ***Consegna delle relazioni (per entrambe le tipologie di prova di programmazione): entro giovedì 27/02/2014, alle ore 23:59, via e-mail all'indirizzo: danilo.vendraminetto@polito.it, usando come subject (oggetto) la stringa APA#<m>, essendo <m> il proprio numero di matricola. L'allegato alla mail deve essere costituito da <u>un unico file</u>: un archivio compresso, contenente sia il codice corretto, sia la relazione (<u>NO eseguibili</u>). QUALORA IL CODICE SPEDITO 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.***

PER I LAUREANDI): per il calendario di invio relazioni ed esami orali vedere avviso sula pagina del corso 02MNO AP sul Portale.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 27/06/2014 - Prova di teoria (12 punti)

1. (1 punto)

Si ordini in maniera ascendente mediante merge-sort il seguente vettore di interi:

11 9 12 1 13 3 39 23 29 17 21 10 18 16 8 100

Si indichino i passaggi rilevanti.

2. (2 punti)

Si determini mediante un algoritmo greedy un codice di Huffman ottimo per i seguenti caratteri con le frequenze specificate:

a: 6 b: 16 c: 4 d: 3 e: 22 f: 11 g: 14 h: 9 i: 8 l: 10

3. (1 punto)

Si esprima in notazione prefissa e postfissa la seguente espressione aritmetica mediante visita dell'albero binario corrispondente: A + ((B – C) / D) * E

4. (2 punti)

Sia data la sequenza di chiavi T con eventuale $T_1HELA_1S_1T_2S_2T_3A_2ND$, dove ciascun carattere è individuato dal suo ordine progressivo nell'alfabeto (A=1, ..., Z=26) con eventuale pedice. Si riporti la struttura di una tabella di hash di dimensione 19, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si supponga di utilizzare il linear chaining con la funzione di hash $h(k) = k \bmod 19$.

5. (1 punto)

Si effettuino, secondo l'ordine specificato, le seguenti operazioni su un BST supposto inizialmente vuoto (+ indica una inserzione in foglia, − una cancellazione):

+10 +4 +3 +16 +21 +7 +11 +1 +8 +19 +12 +15 -4 -10 -7

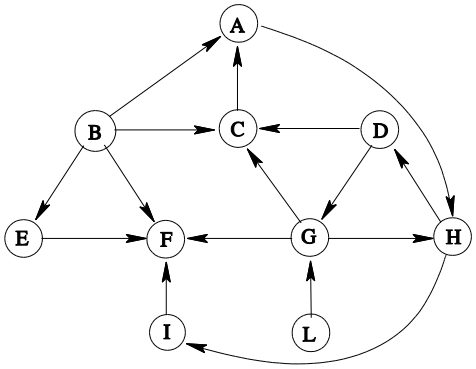
6. (2 punti)

Si inseriscano in sequenza nella radice di un BST, inizialmente supposto vuoto, le chiavi:

10 4 3 16 21 7 11 1

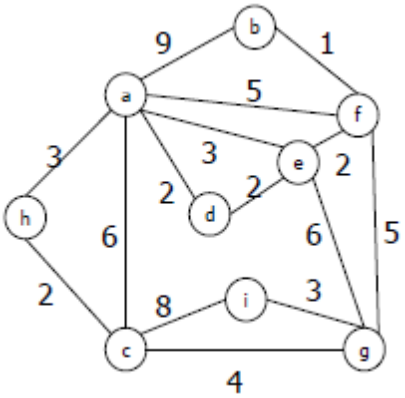
7. (2 x 0.5 punti)

Si rappresenti il seguente grafo orientato come lista delle adiacenze (**0.5 punti**) e come matrice delle adiacenze (**0.5 punti**).



8. (2 punti)

Sia dato il seguente grafo non orientato pesato:



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

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 27/06/2014 - Prova di programmazione (12 punti)

1. (2 punti)

Siano dati 2 vettori di interi `vet1` e `vet2` di dimensioni note `d1` e `d2`, rispettivamente. Si realizzi in C la funzione di prototipo

```
int ricerca(int *vet1, int *vet2, int d1, int d2);
```

che ricerchi se nel vettore `vet1` compare almeno una volta il vettore `vet2` come sottovettore. In caso affermativo, la funzione ritorni l'indice della prima cella della prima occorrenza di `vet2`, in caso negativo ritorni -1.

Esempio

Se `vet1` è

0	15	12	21	7	25	32	1

e `vet 2` è

21	7	25

la ricerca ha esito positivo e la funzione ritorna l'intero 3.

2. (4 punti)

Si implementi come ADT di I categoria una lista bi-linkata di interi con le seguenti 2 funzioni:

- `list_insert (list_t *l, int chiave, int estremo)` dove se il parametro `estremo` vale 0 l'inserzione avviene in testa, se vale 1 in coda
- `list_display (list_t *l, int modo)` dove se il parametro `modo` vale 0 la visualizzazione a video è fatta dalla testa alla coda, se vale 1 dalla coda alla testa.

dove

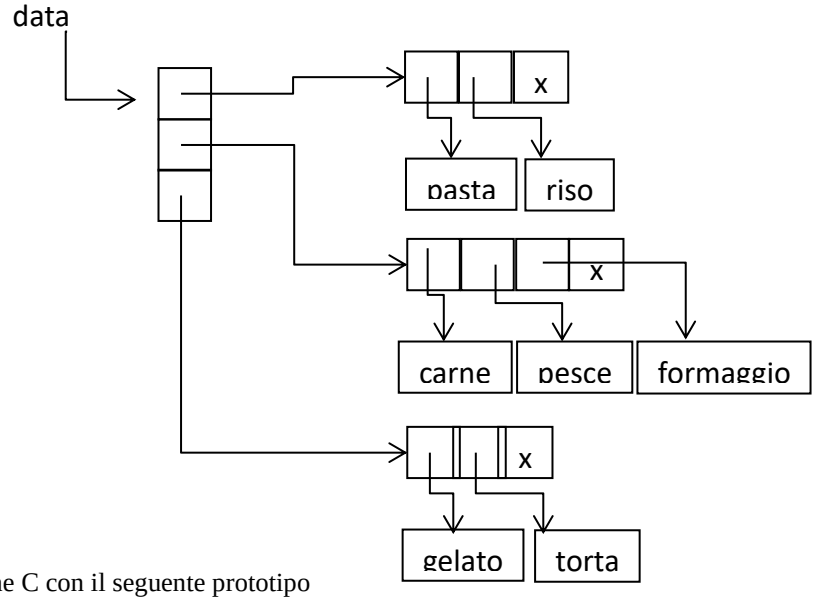
```
typedef struct {nodo_t *head; nodo_t *tail; } list_t;
```

mentre `nodo_t` è una `struct` la cui definizione è richiesta al candidato.

3. (6 punti)

Un ristorante serve un menù a prezzo fisso composto da `n` portate. Per ciascuna portata, il cliente sceglie obbligatoriamente un piatto tra un certo numero di piatti disponibili. Le informazioni sono memorizzate in un vettore `data` di `n` elementi, ciascuno dei quali è un puntatore a un vettore di puntatori a stringhe. Quest'ultimo vettore elenca i piatti disponibili per una certa portata. Il numero di questi piatti per portata non è dato esplicitamente, ma quando l'elemento contiene un puntatore a NULL, l'elenco dei piatti di quella portata si considera terminato. Ogni stringa rappresenta un piatto disponibile.

Esempio:



Si scriva una funzione C con il seguente prototipo

```
void build_menu(char **data[], int n);
```

che, utilizzando un algoritmo ricorsivo, determini tutte le composizioni possibili del menu e le visualizzi. Nel caso dell'esempio, l'output deve essere:

```
(pasta, carne, gelato), (pasta, carne, torta), (pasta, pesce, gelato),
(pasta, pesce, torta), (pasta, formaggio, gelato), (pasta, formaggio, torta)
(riso, carne, gelato), (riso, carne, torta), (riso, pesce, gelato),
(riso, pesce, torta), (riso, formaggio, gelato), (riso, formaggio, torta)
```

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 27/06/2014 - Prova di programmazione (18 punti)

Un archivio produttore-modello-accessori è organizzato come segue:

- il primo file memorizza l'elenco dei produttori, in ragione di un produttore per ciascuna riga. Il numero di produttori non è noto a priori, né è presente in alcun modo nel file. Per ciascun produttore viene riportato il suo nome e il nome del file che contiene i modelli da esso prodotti
- i file successivi, tanti quanti sono i produttori, riportano per ogni produttore quali modelli esso produce in ragione di un modello per ciascuna riga. Per ogni modello viene riportato il suo nome e il nome del file che contiene gli accessori relativi a quel modello
- i file successivi, tanti quanti sono i modelli, riportano per ogni modello la lista degli accessori di cui dispone in ragione di un accessorio per ciascuna riga. Per ogni accessorio viene riportato il suo nome e il suo costo in Euro.

Si osservi che tutti i nomi sono stringhe alfanumeriche nelle quali gli spazi sono sostituiti dal carattere di sottolineatura "_" e di lunghezza massima uguale a 100. Inoltre tutti i nomi del produttore, del modello e dell'accessorio, sono univoci all'interno dell'intera base dati.

Esempio

```
Produttori.txt
Audi  Audi_modelli.txt
BMW   BMW_modelli.txt
FIAT  FIAT_modelli.txt
Ford  Ford_modelli.txt
Renault Renault_modelli.txt
Toyota Toyota_modelli.txt
Volkswagen Volkswagen_modelli.txt

FIAT_modelli.txt
Cinquecento Cinquecento_accessori.txt
Bravo Bravo_accessori.txt
Doblò Doblò_accessori.txt
Freemont Freemont_accessori.txt
Panda Panda_accessori.txt
Punto Punto_accessori.txt

Bravo_accessori.txt
Barre_portasurf 150,00
Spoiler 220,50
Cerchi_lega 75,25
Vivavoce 450,50
Sensori_parcheggio 700,50
```

Si scriva un programma in grado di:

- ricevere il nome del file produttori sulla riga di comando
- leggere i file dei modelli e degli accessori e memorizzare tutti i dati relativi a produttori, modelli e accessori in un'opportuna struttura dati
- realizzare un menu con i seguenti comandi:
  - leggere da tastiera il nome di un produttore, visualizzando (a video) l'elenco dei modelli da esso prodotti con complessità al più logaritmica nel numero di produttori
  - leggere da tastiera il nome di un modello, visualizzando (a video) l'elenco degli accessori ad esso associati con complessità al più logaritmica nel numero totale di modelli
  - cancellare un produttore, i modelli ad esso relativi e gli accessori associati a quei modelli
  - cancellare un modello e gli accessori ad esso associati
  - cancellare un accessorio
  - incorporare il produttore2 nel produttore1: tutti i modelli ed i relativi accessori del produttore2 sono attribuiti al produttore1, il produttore2 viene cancellato dalla base dati
  - uscita.

Le strutture dati siano gestite come ADT di I categoria.

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).***
- ***È consentito utilizzare chiamate a funzioni standard, quali ordinamento per vettori, inserzione/estrazione/ricerca relative a FIFO, LIFO, liste, BST, tabelle di hash e altre strutture dati, considerate come librerie esterne. Gli header file delle librerie utilizzate devono essere allegati all'elaborato. Le funzioni richiamate, inoltre, dovranno essere incluse nella versione del programma allegata alla relazione.***
- ***Consegna delle relazioni (per entrambe le tipologie di prova di programmazione): entro mercoledì 02/07/2014, alle ore 23:59, via e-mail all'indirizzo: danilo.vendraminetto@polito.it, usando come subject (oggetto) la stringa APA#<m>, essendo <m> il proprio numero di matricola. L'allegato alla mail deve essere costituito da un unico file: un archivio compresso, contenente sia il codice corretto, sia la relazione (NO eseguibili). QUALORA IL CODICE SPEDITO CON LA RELAZIONE NON COMPILI CORRETTAMENTE, VERRÀ APPLICATA UNA PENALIZZAZIONE. Si ricorda che la valutazione del compito viene fatta senza discussione o esame orale, sulla base dell'elaborato svolto in aula. Non verranno corretti i compiti di cui non sarà stata inviata la relazione nei tempi stabiliti.***

PER I LAUREANDI DELLA SESSIONE DI LUGLIO 2014: per il calendario di invio relazioni ed esami orali vedere avviso sula pagina del corso 02MNO AP sul Portale.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 03/09/2014 - 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:

2-10, 4-2, 0-4, 6-1, 2-7, 4-3, 0-8, 0-4, 7-8, 10-3

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

2. (**2 punti**)
Sia data la sequenza di interi, supposta memorizzata in un vettore:

21 2 11 3 7 33 13 5 16 9 17 1

- 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.

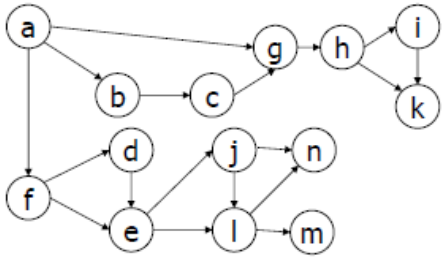
3. (2 punti)
Si inseriscano in sequenza in foglia in un BST, inizialmente supposto vuoto, le chiavi:

9 24 31 68 11 13 58 8 75

e al termine dell'inserzione si partizioni il BST secondo la chiave mediana.

4 (2 punti)
Sia data la sequenza di chiavi HUNGE$_1$RGAME$_2$S, dove ciascun carattere è individuato dal suo ordine progressivo nell'alfabeto (A=1, ..., Z=26) e da un eventuale pedice. Si riporti la struttura di una tabella di hash di dimensione 23, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si supponga di utilizzare l'open addressing con quadratic probing. Si selezionino opportuni valori per c_1 e c2.

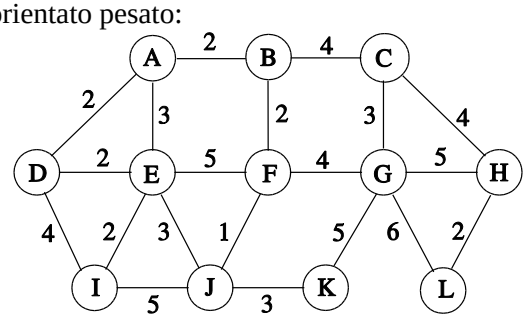
5. (2 punti)
Si ordini topologicamente il seguente DAG. Qualora necessario, si trattino i vertici secondo l'ordine alfabetico e si assuma che la lista delle adiacenze sia anch'essa ordinata alfabeticamente.



6. (1 punto)
Si trasformi il grafo dell'esercizio 5 nel corrispondente grafo non orientato e se ne determinino i punti di articolazione. Si consideri **a** come vertice di partenza e, qualora necessario, si trattino i vertici secondo l'ordine alfabetico.

7. (2 punti)

Sia dato il seguente grafo non orientato pesato:



se ne determini un minimum spanning tree applicando l'algoritmo di Kruskal, disegnando l'albero e ritornando come risultato il valore del peso minimo. Si esplicitino i passi intermedi.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 03/09/2014 - Prova di programmazione (12 punti)

1. (2 punti)

Si implementi la funzione

```
void searchStr (char *str, int *start, int *length);
```

che riceve in ingresso la stringa `str` e rintraccia in tale stringa la sequenza di caratteri uguali di lunghezza maggiore, ritornandone l'indice di inizio nella variabile `start` e la sua lunghezza nella variabile `length`.

Ad esempio, se la funzione ricevesse la stringa

abbcccddddeeeee

dovrebbe rintracciare la sequenza eeeee e ritornare i valori start = 10 e length = 5.

2. (4 punti)

Una stringa contiene delle sottostringhe, costituite da caratteri esclusivamente alfabetici e numerici, delimitate dal carattere punto "`.`". Si scriva la funzione:

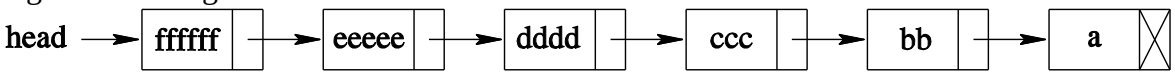
```
node_t *splitStr (char *str);
```

in grado di ricevere la stringa `str` con un tale formato e di restituire il puntatore a un lista in cui ciascun elemento contiene una delle sottostringhe della stringa originaria, definita dinamicamente. Si riporti inoltre la definizione del nodo della lista.

Ad esempio, se la funzione ricevesse la stringa

a.bb.ccc.dddd.eeeee.fffff

dovrebbe generare la seguente lista:



nella quale le stringhe sono allocate dinamicamente, restituendo il puntatore `head`.

3. (6 punti)

Gli alberi di grado *n* possono essere rappresentati mediante nodi ciascuno contenente *n* puntatori (Fig. 1(a)) oppure, per evitare di appesantire ciascun nodo di un albero con un numero eccessivo di puntatori, mediante la tecnica del *left-child right-sibling.* Con questa tecnica in ogni nodo vi sono un puntatore al figlio più a sinistra e al fratello del nodo immediatamente a destra. La Fig. 1(b) mostra lo stesso albero della Fig. 1(a) rappresentato con la tecnica del *left-child right-sibling.*

Si supponga che ciascun nodo dell'albero rappresentato come *left-child right-sibling* sia di tipo `node_t` e che includa due stringhe dinamiche (cognome e nome) e il voto di un esame (valore intero). Si definisca la struttura C relativa a un nodo dell'albero. Si scriva inoltre la funzione ricorsiva in grado di visualizzare tutto il contenuto dell'albero stesso una volta ricevuta la radice dell'albero quale parametro.

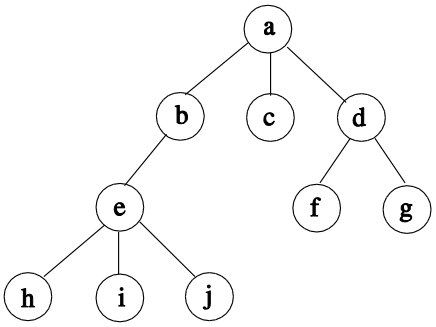


Fig. 1(a)

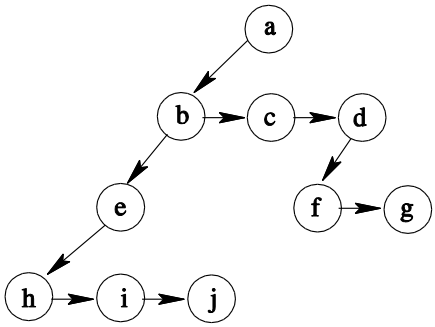


Fig. 1(b)

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 03/09/2014 - Prova di programmazione (18 punti)

Un progettista deve configurare una rete, della quale inizialmente sono dati solo i nodi. L'obiettivo del programma sarà quindi di determinare gli archi:

- o verificando una soluzione generata manualmente
- o generandola automaticamente.

Un file contiene l'elenco dei nodi uno per riga come stringa di caratteri univoca di lunghezza pari al massimo a 20 caratteri. Il numero di nodi non è noto a priori. Il nome del file compare sulla riga di comando.

Esempio di file

`Nodi.txt`

```
nodoA
nodoB
nodoC
nodoD
```

Il progettista deve stabilire quali sono gli archi (bidirezionali) tra i nodi per ottenere una rete connessa che rispetti tutti i seguenti vincoli:

- il numero di archi deve essere minimo
- per ogni coppia di vertici (v_i, v_j) la lunghezza del cammino minimo che li connette non deve superare un valore intero *k* che compare sulla riga di comando
- ogni nodo ha grado non superiore a *m*,, valore intero che compare sulla riga di comando.

Si ricordi che per rete connessa si intende una rete in cui per ogni coppia di vertici (v_i, v_j) esiste un cammino che li connette).

Si scriva un programma C che:

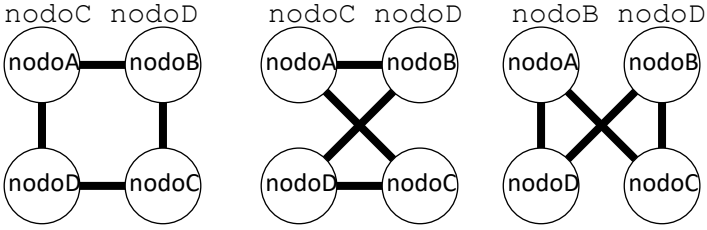
- legga il file contenente i nodi e memorizzi le sue informazioni in un'opportuna struttura dati
- generi gli archi in una delle seguenti 2 modalità alternative con scelta da input:
  - legga un secondo file contenente un elenco di archi, uno per riga nel formato v_i v_j, e verifichi se questo insieme soddisfa le seguenti condizioni:
    - ogni coppia di vertici (v_i, v_j) è connessa da un cammino lungo al massimo *k*
    - ogni vertice ha grado non superiore a *m*

    Si assuma che tutti gli archi contenuti nel file siano coerenti con l'elenco dei nodi del primo file.
  - generi automaticamente una qualsiasi soluzione ottima che soddisfi i vincoli e la scriva sotto forma di elenco di archi, uno per riga, in un file di output il cui nome compare sulla riga di comando.

Esempio:

per `Nodi.txt` con *k*=2 e *m*=2 le seguenti 3 soluzioni rispettano i vincoli:

```
nodoA nodoB     nodoA nodoB   nodoA nodoC
nodoA nodoD     nodoA nodoC   nodoA nodoD
nodoB nodoC     nodoB nodoD   nodoB nodoC
nodoC nodoD     nodoC nodoD   nodoB nodoD
```



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).***
- ***È consentito utilizzare chiamate a funzioni <u>standard</u>, quali ordinamento per vettori, inserzione/estrazione/ricerca relative a FIFO, LIFO, liste, BST, tabelle di hash e altre strutture dati, considerate come librerie esterne. Gli header file delle librerie utilizzate devono essere allegati all'elaborato. Le funzioni richiamate, inoltre, dovranno essere incluse nella versione del programma allegata alla relazione.***
- ***Consegna delle relazioni (per entrambe le tipologie di prova di programmazione): entro lunedì 08/09/2014, alle ore 23:59, via e-mail all'indirizzo: danilo.vendraminetto@polito.it, usando come subject (oggetto) la stringa APA#<m>, essendo <m> il proprio numero di matricola. L'allegato alla mail deve essere costituito da <u>un unico file</u>: un archivio compresso, contenente sia il codice corretto, sia la relazione (<u>NO eseguibili</u>). QUALORA IL CODICE SPEDITO 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.***

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 02/02/2015 - 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:

3-9, 4-3, 2-5, 6-2, 3-8, 5-6, 0-10, 3-5, 8-9, 10-3

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. (**1 punto**)
Si ordini in maniera ascendente il seguente vettore di interi mediante Shellsort con la sequenza di Knuth:

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

Si indichino i passaggi principali.

3. **(2 punti**)
Sia data una coda a priorità inizialmente vuota implementata mediante uno heap. Sia data la sequenza di interi e carattere *:

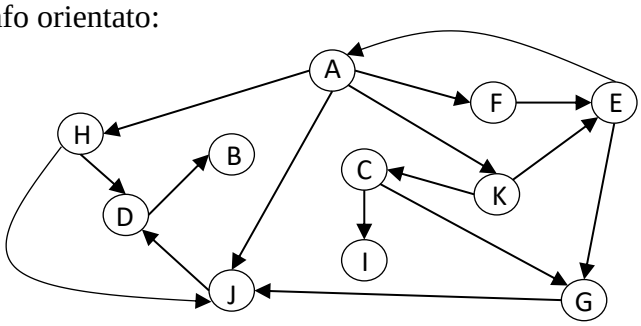
20 32 19 51 * * 28 * 74 9 81 * * 17 * 41 33 18

dove ad ogni intero corrisponde un inserimento nella coda a priorità e al carattere * un'estrazione con cancellazione del massimo. Si riporti la configurazione della coda a priorità dopo ogni operazione e la sequenza dei valori restituiti dalle estrazioni con cancellazione del massimo. Al termine si cambi la priorità di 41 in 11 e si disegni la configurazione risultante della coda a priorità.

4. (**1 punto**)
Si esprima in notazione prefissa e postfissa la seguente espressione aritmetica mediante visita dell'albero binario corrispondente: ((A +B) * (C*(D-E))) /((E-F) + G*H)

5. **(2 punti**)
Sia data la sequenza di chiavi FEB_1B_2RAIO, dove ciascun carattere è individuato dal suo ordine progressivo nell'alfabeto (A=1, ..., Z=26) e da un eventuale pedice. Si riporti la struttura di una tabella di hash di dimensione 19, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si supponga di utilizzare l'open addressing con quadratic probing. Si selezionino opportuni valori per c_1 e c_2.

6.
Sia dato il seguente grafo orientato:



- se ne effettui una visita in profondità, considerando **A** come vertice di partenza. Si etichettino indicando per ognuno di essi i tempi di scoperta e di fine elaborazione nel formato tempo1/tempo2 (**2 punti**)
- se ne effettui una visita in ampiezza, considerando **A** come vertice di partenza (**1.5 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**).

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

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 02/02/2015 - Prova di programmazione (18 punti)

Formulazione del problema: uno studente X impegnato a preparare gli esami della sessione invernale si è scordato di fare i regali di Natale ai suoi amici. Alcuni di questi amici sono a loro volta amici tra loro e la loro relazione di amicizia è descritta mediante un grafo non orientato. Ogni amico dello studente X deve ricevere da X uno ed un solo regalo, più amici di X possono ricevere da X regali dello stesso tipo alla condizione di non essere amici tra di loro.

Dati in ingresso:

- le relazioni di amicizia tra gli amici di X sono memorizzate in un primo file di testo, dove per ogni riga compare una coppia di nomi di 2 amici di X che sono a loro volta amici tra di loro. Il numero di righe del file non è noto a priori, né sovrastimabile
- lo studente X ha prodotto manualmente una soluzione scritta su di un secondo file di testo. Nella prima riga si trova il numero N di tipologie di regali che X ha determinato. Di seguito compaiono tanti gruppi di righe quante sono le N tipologie di regali. Ciascun gruppo consta di una prima riga con un intero k_i ($1 <= k_i <=$ numero_amici) che indica il numero di amici che riceveranno un regalo appartenente alla tipologia corrente i ($0 <= i < N$), seguito nelle righe k_i successive dai nomi degli amici uno per riga.

Vincoli:

- ogni stringa è lunga al massimo 15 caratteri
- i nomi dei file sono passati sulla riga di comando.

Richieste:

Si scriva un programma C che:

1. legga dal primo file le relazioni di amicizia tra gli amici di X, legga dal secondo file la soluzione presunta, e memorizzi le informazioni in opportune strutture dati in memoria
2. verifichi che la soluzione presunta soddisfi le condizioni (ogni amico ha ricevuto uno ed un solo regalo e che due amici di X amici tra di loro non abbiano ricevuto regali dello stesso tipo). Non è richiesto di verificare che il numero tipologie di regalo sia minimo
3. calcoli in modo ricorsivo il numero minimo N di tipologie di regalo tali da soddisfare le condizioni di cui sopra e generi UNA PARTIZIONE degli amici di X in N sottoinsiemi corrispondenti ognuno ad una tipologia di regalo.

Esempio:

I file
Diego Emilia
Claudia Emilia
Bartolo Claudia
Claudia Diego
Bartolo Diego
Andrea Claudia
Claudia Fabio
Andrea Bartolo
Diego Fabio

II file
4
1
Claudia
2
Andrea
Diego
2
Bartolo
Emilia
1
Fabio

Il programma con questo esempio deve fornire i seguenti risultati con formato a scelta del candidato:

- punto 2: la soluzione manuale soddisfa le condizioni di cui è richiesta la verifica
- punto 3: il numero minimo di tipologie di regali è N=3 e un possibile partizionamento è {Claudia}, {Andrea, Diego}, {Bartolo, Emilia, Fabio}.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 02/02/2015 - Prova di programmazione (12 punti)

1. (**2 punti**)

Si realizzi una funzione C `eraseDuplicate` che, data una stringa `str` ritorni la stessa stringa in cui sia mantenuta solo la prima occorrenza di tutti i caratteri che appaiono più di una volta nella stringa originale. Il prototipo della funzione è:

```
void eraseDuplicate (char *str);
```

Ad esempio, se inizialmente `str` contiene `"aa;;;bbbab;"` la funzione deve ritornare la stringa `"a;b"`.

2. (**4 punti**)

Sia dato un albero di grado N, i cui nodi sono definiti dalla seguente struttura C:

```
struct node {
  int key;
  struct node *children[N];
};
```

Si realizzi una funzione C

```
void visitLevelByLevel (struct node *root, int l1, int l2);
```

che visiti l'albero da profondità `l1` a profondità `l2` e che visualizzi tutte le chiavi livello per livello, cioè prima tutte quelle a profondità `l1`, seguite da quelle a profondità `l1+1`, etc. fino a quelle a profondità `l2`. Si assuma che `l1 < l2`. Si noti che è possibile che l'albero sia visitato più di una volta per ottenere il risultato desiderato. **Non è consentito l'uso di funzioni di libreria**.

3. **(6 punti**)

Una sigla alfanumerica di lunghezza `N` è composta selezionando per ognuna delle `N` posizioni `sigla[i]` un carattere che appartiene all'insieme `set[j]`. Ogni insieme `set[j]` è dato come stringa di caratteri alfanumerici e la sua cardinalità è al massimo 10. Un file contiene le informazioni relative alla lunghezza della sigla `N` (intero sulla prima riga) e alle stringhe che rappresentano i `set[j]` (`N` stringhe una per riga sulle `N` righe successive). Si realizzi una funzione ricorsiva in C che, letto il file, generi tutte le possibili sigle e le memorizzi su di un secondo file. I nomi dei file siano passati come parametri alla funzione.

Esempio : se il primo file ha il seguente contenuto :
3
A
Xy
123
occorre scrivere nel secondo file le sigle: AX1, AX2, AX3, Ay1, Ay2, Ay3.

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).***
- ***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 delle librerie utilizzate devono essere allegati all'elaborato. 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 05/02/2015, alle ore 24:00, mediante caricamento su Portale. Le istruzioni per il caricamento sono pubblicate sul Portale nella sezione Materiale). QUALORA IL CODICE SPEDITO CON LA RELAZIONE NON COMPILI CORRETTAMENTE, VERRÀ APPLICATA UNA PENALIZZAZIONE. Si ricorda che la valutazione del compito viene fatta senza discussione o esame orale, sulla base dell'elaborato svolto in aula. Non verranno corretti i compiti di cui non sarà stata inviata la relazione nei tempi stabiliti.***

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 23/02/2015 - Prova di teoria (12 punti)

1. (**2 punti**)
Si risolva la seguente equazione alle ricorrenze mediante il metodo dello sviluppo (unfolding):

$$T(n) = 4T(n/3) + n \qquad n \geq 2$$
$$T(1) = 1$$

2. (1 punto)
Si ordini in maniera ascendente mediante counting-sort il seguente vettore di interi:

5 2 7 0 8 3 6 0 10 1 2 1 10 0 4

Si indichino le strutture dati usate nei passi intermedi.

3. (2 punti)
Sia data la sequenza di interi, supposta memorizzata in un vettore:

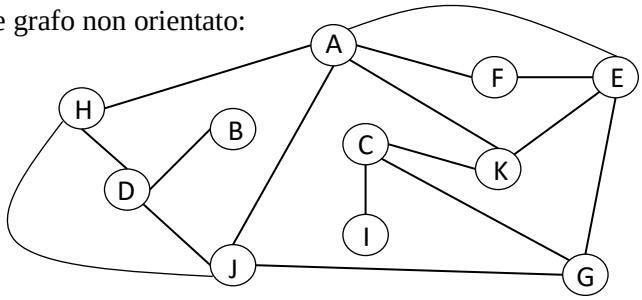
22 39 19 21 31 43 32 23 25 29 35 40

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.

4. (**2 punti**)
Si effettuino, secondo l'ordine specificato, le seguenti inserzioni in foglia su un Interval BST supposto inizialmente vuoto:

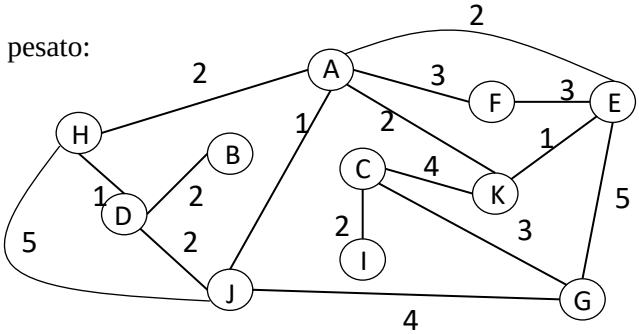
[3,10] [4, 6] [1,3] [16,21] [7,11] [2,8] [12,19] [5,15] [9,10]

5. (1 punto)
Si determinino i punti di articolazione del seguente grafo non orientato:



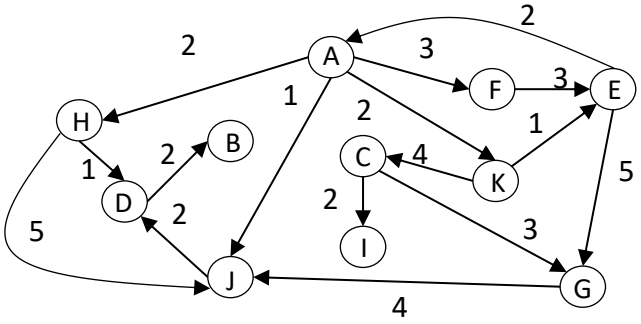
Si consideri **A** come vertice di partenza e, qualora necessario, si trattino i vertici secondo l'ordine alfabetico.

6. (2 punti)
Sia dato il seguente grafo non orientato pesato:



se ne determini un minimum spanning tree applicando l'algoritmo di Kruskal, disegnando l'albero e ritornando come risultato il valore del peso minimo. Si esplicitino i passi intermedi.

7. (2 punti)
Sul seguente grafo orientato e 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.



02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 23/02/2015 - Prova di programmazione (18 punti)

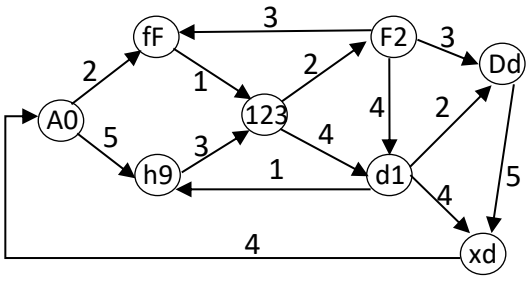
Un grafo orientato pesato è memorizzato su un primo file mediante l'elenco dei suoi archi, con il seguente formato:

idV_1 val idV_2

che indica che il vertice con identificatore idV1 è connesso con un arco orientato di peso val (valore intero e positivo) al vertice idV2.
Poiché il numero di vertici del grafo non è noto a priori, si suggerisce di calcolarlo tramite lettura preliminare del file e/o caricamento in una tabella di simboli. 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.

Esempio: contenuto del primo file e grafo corrispondente:

```
fF 1 123
A0 2 fF
A0 5 h9
h9 3 123
123 2 F2
123 4 d1
F2 3 Dd
F2 4 d1
d1 2 Dd
d1 4 xd
d1 1 h9
Dd 5 xd
xd 4 A0
F2 3 fF
```



Su un secondo file sono memorizzati 2 cammini **semplici** con il seguente formato: sulla prima riga compare la lunghezza del primo cammino, seguita dai vertici che lo compongono separati da spazi o caratteri a-capo. Analogamente per il secondo cammino.

Esempio: contenuto del secondo file

```
4
A0 fF 123 d1 xd
5
A0 h9 123 F2 d1 Dd
```

Si scriva un programma C in grado di:

- ricevere i nomi dei due file sulla riga di comando
- leggere il grafo dal primo file e memorizzarlo in un'opportuna struttura dati
- leggere i 2 cammini semplici dal secondo file, identificare i vertici in comune (intersezione dei 2 cammini) e visualizzare ciascuno dei 2 cammini decomposto in sottocammini determinati dai vertici in comune
- data una coppia di vertici sorgente/destinazione letta da tastiera, dati 2 interi k e p letti da tastiera ($k \leq p$), stampare il **cammino ottimo non necessariamente semplici** tra i vertici sorgente e destinazione che soddisfi i seguenti vincoli:
  - è massima la somma dei pesi degli archi del cammino
  - sono riattraversati al più k vertici
  - il numero complessivo di riattraversamenti è al massimo p
  - una volta raggiunto il nodo di destinazione, il cammino è da considerarsi terminato.

In riferimento all'esempio, i 2 cammini hanno in comune i vertice `A0`, `123 e d1`, quindi si dovrà visualizzare:

```
i vertici in comune sono:
A0
123
d1
```

```
il cammino 1 si decompone in 3 sottocammini:
sottocammino 1.1: A0, fF, 123
sottocammino 1.2: 123, d1
sottocammino 1.3: d1, xd

il cammino 2 si decompone in 3 sottocammini:
sottocammino 2.1: A0, h9, 123
sottocammino 2.2: 123, F2, d1
sottocammino 2.3: d1, Dd
```

Ad esempio, indicando `A0` e `fF` come sorgente e destinazione, rispettivamente, con k=1 e p=1, il cammino massimo vale 27 e corrisponde a: `A0 h9 123 F2 d1 Dd xd A0 fF` (si può riattraversare solo un nodo e una sola volta).
Mantenendo i medesimi nodi come sorgente e destinazione, ma utilizzando k=6 e p=7, si ottiene un cammino di valore 50, corrispondente a: `A0 h9 123 F2 d1 Dd xd A0 h9 123 d1 Dd xd A0 fF`

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 23/02/2015 - Prova di programmazione (12 punti)

1. (**2 punti**)

Si realizzi una funzione C `invertSequence` con il seguente prototipo:

```
void invertSequence (int *v1, int n, int *v2);
```

La funzione riceve come parametri 2 vettori di interi `v1` e `v2` e la loro dimensione `n` e memorizza in `v2` lo stesso insieme di numeri presente in `v1` ma con tutte le sottosequenze ascendenti in `v1` trasformate in sottosequenze discendenti in `v2`.

Ad esempio, se inizialmente `v1` contiene `1 2 3 4 5 0 12 13 14 2`, `v2` al termine dovrà contenere `5 4 3 2 1 14 13 12 0 2`.

2. (**4 punti**)

Sia dato un albero binario i cui nodi sono definiti dalla seguente struttura C:

```
struct node {
  int  key;
  struct node *left;
  struct node *right;
};
```

Si realizzi una funzione C

```
void printPaths (struct node *root, int h, ...);
```

che, dato un albero di altezza `h`, visualizzi tutti i cammini radice-foglie elencandone le chiavi. Si noti che ogni cammino deve essere visualizzato completamente da radice a foglia, quindi il cammino percorso in ogni momento deve essere passato tra chiamate ricorsive. Se necessario, è possibile aggiungere parametri alla funzione `printPath`.

3. (**6 punti**)

Una password è una stringa di 5 elementi:

- i primi 3 sono lettere dell'alfabeto inglese maiuscolo e gli ultimi 2 sono cifre della base 10
- la stessa lettera o cifra può comparire al più k volte, dove k è un intero letto da tastiera

Si scriva una funzione ricorsiva in C in grado di generare tutte le password secondo le regole di cui sopra e di scriverle su un file di uscita, il cui nome è passato come parametro nella riga di comandi.

Esempio: con k=2 alcune delle password generate sono: AAB11, ZDZ09, ABC34

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).***
- ***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 delle librerie utilizzate devono essere allegati all'elaborato. 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 26/02/2015, alle ore 23:59, mediante caricamento su Portale. Le istruzioni per il caricamento sono pubblicate sul Portale nella sezione Materiale). QUALORA IL CODICE SPEDITO 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.***

SCADENZE PER LAUREANDI Viste le scadenze per superare gli esami e presentare domanda di laurea, per I SOLI LAUREANDI le tempistiche del II appello sono le seguenti: - martedì 24 febbraio ore 23.59: termine per la consegna della relazione mediante caricamento su Portale. L'archivio deve avere nome sXXXXXX_230215_LAUREANDO - venerdì 27 febbraio ore 9.00 aula 23: esami orali.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 16/06/2015 - Prova di teoria (12 punti)

1. (1+1 punti)

Si ordini in maniera ascendente il seguente vettore di interi:

12 91 2 41 13 3 39 43 29 17 71 10 18 6 8 100

- mediante selection-sort
- mediante merge-sort.

Si indichino i passaggi rilevanti.

2. (2 punti)

Sia data una coda a priorità inizialmente vuota implementata mediante uno heap. Sia data la sequenza di interi e carattere *: 12 23 29 61 * * 18 * 34 9 13 * * 14 * 21 27 28 dove ad ogni intero corrisponde un inserimento nella coda a priorità e al carattere * un'estrazione con cancellazione del minimo. Si riporti la configurazione della coda a priorità dopo ogni operazione e la sequenza dei valori restituiti dalle estrazioni con cancellazione del minimo.

3. (2 punti)

Si determini mediante un algoritmo greedy un codice di Huffman ottimo per i seguenti caratteri con le frequenze specificate:

a: 26 b: 6 c: 4 d: 3 e: 22 f: 13 g: 14 h: 9 i: 18 l: 10

4. (1.5 punti)

Si esprima in notazione prefissa, postfissa e infissa senza parentesi la seguente espressione aritmetica mediante visita dell'albero binario corrispondente:

(((A + B) – (C – D)) * E) /((F * G) / H – I)

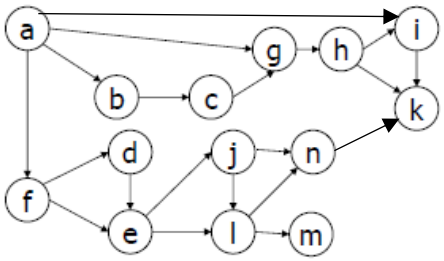
5. (1 punto)

Si supponga di aver memorizzato tutti i numeri compresi tra 1 e 500 in un albero di ricerca binario e che si stia cercando il numero 231. Quali tra queste non possono essere le sequenze esaminate durante la ricerca? Perché?

250	200	240	260	255	220	230	231				
450	100	400	150	350	200	250	231				
270	100	200	300	400	450	260	350	220	225	230	231

6.

Sia dato il seguente DAG: considerando **a** come vertice di partenza



- se ne effettui una visita in ampiezza (**1.5 punti**)
- lo si ordini in modo topologico inverso **(2 punti).**

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

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 16/06/2015 - Prova di programmazione (12 punti)

1. (**4 punti**)

Si realizzi una funzione C `mul` con il seguente prototipo:

```
void mul (int *v1, int *v2, int n, int **pv);
```

che moltiplica due numeri interi di *n* cifre, il primo memorizzato nel vettore `v1`, il secondo nel vettore `v2`.
La funzione restituisce il risultato della moltiplicazione nel vettore individuato dal puntatore `pv`. I numeri sono memorizzati nei vettori `v1` e `v2` in ragione di una cifra decimale per ciascun elemento. La funzione implementa l'algoritmo classico di moltiplicazione, operando per somma e scalamento e tenendo conto dei riporti, come illustrato dal seguente esempio:

```
    0 3 2 x
    2 4 3 =
--------------
0 0 0 0 9 6 +
0 0 1 2 8  =
0 0 6 4
-------------
0 0 7 7 7 6
```

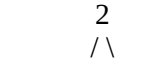
Si osservi che, per evitare overflow, il prodotto dovrà essere rappresentato su *2n* cifre. La funzione `mul` si occupi di allocare dinamicamente il vettore individuato da `pv` e di memorizzare il risultato in tale vettore.

2. (**4 punti**)

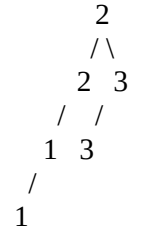
Si realizzi una funzione C `doubleTree` con il seguente prototipo:

```
void doubleTree(struct node *root);
```

che, per ciascun nodo di un albero binario, crei un nodo duplicato e inserisca tale nodo come figlio sinistro del nodo originale. Ad esempio l'albero:



diventa:



Si scriva inoltre la struttura del nodo dell'albero `struct node` in grado di memorizzare chiavi intere.

3. (**4 punti**)

Un insieme di produttori-prodotti è memorizzato in una lista di liste. La lista principale specifica il nome dei produttori (stringa di al massimo 30 caratteri). Le liste secondarie riportano, per ciascun produttore, i nomi dei prodotti (stringa di al massimo 30 caratteri) e il relativo prezzo (valore reale). Tanto i nomi dei produttori quanto quelli dei prodotti sono univoci all'interno dell'intera base dati. Tanto la lista principale quanto quelle secondarie sono ordinate in ordine lessicografico crescente per nome produttore e prodotto, rispettivamente.

Si scriva una (singola) funzione ricorsiva in C in grado di ricevere quali parametri il nome di un produttore, il nome di un prodotto e gli ulteriori parametri che il candidato ritiene necessari.
Essa sia in grado di visualizzare il prezzo del prodotto del produttore indicato se esso esiste, altrimenti dia un'indicazione di errore.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 16/06/2015 - Prova di programmazione (18 punti)

Si richiede lo sviluppo di un'applicazione in grado di gestire un impianto sciistico secondo le seguenti modalità:

- ogni sciatore ha una tessera caratterizzata da un identificatore (`cardId`) costituito da un numero intero
- ogni skilift ha un identificatore di skilift (`skiliftId`) costituito da una stringa di 10 caratteri. Ogni skilift ha un lettore di tessere per abilitare uno sciatore all'utilizzo dello skilift stesso.

Il sistema riceve da tastiera le letture degli skilift con formato:

```
skiliftId cardId time
```

verifica e rilascia o meno l'autorizzazione al passaggio dello sciatore. Il tempo viene introdotto quale valore intero rappresentante il numero di minuti trascorsi a partire dalle ore `00:00` del giorno stesso.

L'obbiettivo è di evitare che persone diverse usino la stessa tessera. L'autorizzazione viene quindi data solo se la tessera non è stata letta dallo stesso skilift per un certo intervallo di tempo. Tale intervallo di tempo dipende dalla posizione e lunghezza dello skilift.

I tempi dai vari skilift vengono letti da un file. Il nome del file viene ricevuto dall'applicazione come parametro sulla linea di comando e ha il seguente formato:

```
skiliftId   timeInterval
```

dove `timeInterval` è un valore espresso in minuti.

Il sistema, una volta inizializzato, deve:

- fornire una funzione di autorizzazione del tipo

  ```
  int authorize (long cardId, char *skiliftId, int time);
  ```

  che permetta allo sciatore `cardId` di utilizzare lo skilift `skiliftId` oppure gli neghi l'accesso se il vincolo temporale, valutato in base all'ultimo passaggio dello sciatore su tale skilift e l'ora attuale `time`, non è rispettato
- mantenere, in memoria centrale, l'elenco di tutti gli sciatori abilitati da ciascuno skilift e, per ciascuno skilift, il numero di volte per cui lo sciatore è stato abilitato
- mantenere, in memoria centrale, l'elenco di tutti gli skilift utilizzati da ciascuno sciatore e per ciascuno skilift l'ora dell'ultimo utilizzo (abilitazione).

Si osservi che:

- essendo il numero di sciatori relativamente elevato e variabile durante la giornata, il costo asintotico di tutte le operazioni effettuate deve essere al più logaritmico nel numero di sciatori
- essendo il numero di skilift limitato e deducibile dalla lettura del file riportante i ritardi temporali, il costo asintotico di tutte le operazioni non ha alcun vincolo in relazione al numero di skilift.

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)***
- ***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 delle librerie utilizzate devono essere allegati all'elaborato. 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 19/06/2015, alle ore 24:00, mediante caricamento su Portale. Le istruzioni per il caricamento sono pubblicate sul Portale nella sezione Materiale). QUALORA IL CODICE SPEDITO 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.***

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 02/09/2015 - Prova di teoria (12 punti)

1. (**1 punto**)

Si ordini in maniera ascendente il seguente vettore di interi:

22 11 12 51 23 3 19 43 29 17 81 100 18 60 8 10

mediante insertion-sort, Si riportino i passaggi rilevanti.

2. (**1 punto**)

Sia dato il seguente insieme di attività, dove la *i*-esima attività è identificata dalla coppia tempo di inizio (s_i) e tempo di terminazione (f_i):

i	s_i	f_i	*i*	s_i	f_i
1	0	6	5	1	9
2	7	10	6	11	15
3	2	4	7	3	7
4	5	9	8	4	10

Si determini, mediante un algoritmo greedy, il massimo numero di attività mutuamente compatibili.

3. **(2 punti)**

Sia dato un albero binario con 15 nodi. Nella visita si ottengono le 3 seguenti sequenze di chiavi:

preorder	30	18	33	12	16	17	19	7	21	10	11	9	3	13	15
inorder	16	12	17	33	7	19	18	21	30	11	3	9	13	10	15
postorder	16	17	12	7	19	33	21	18	3	13	9	11	15	10	30

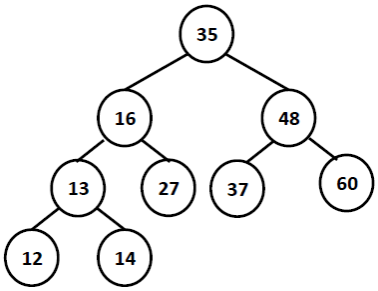
Si disegni l'albero binario di partenza.

4. **(2 punti**)

Sia data la sequenza di chiavi I B U X E G Z H Y, dove ciascun carattere è individuato dal suo ordine progressivo nell'alfabeto (A=1, ..., Z=26). Si riporti la struttura 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.

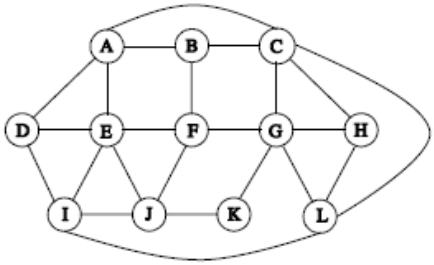
5. **(2 punti)**

Si partizioni il seguente BST intorno alla chiave mediana:



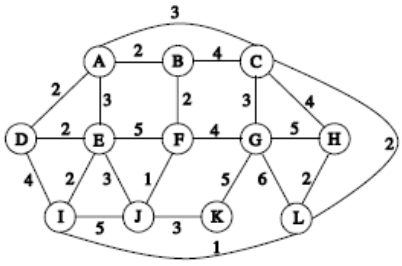
6. **(2 punti)**

Si visiti in profondità il seguente grafo non orientato, considerando **A** come vertice di partenza. Si etichettino indicando per ognuno di essi i tempi di scoperta e di fine elaborazione nel formato tempo1/tempo2. Qualora necessario, si considerino i nodi in ordine alfabetico.



7. **(2 punti)**

Sia dato il seguente grafo non orientato pesato:



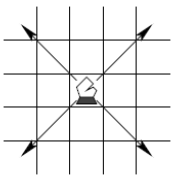
se ne determini un minimum spanning tree applicando l'algoritmo di Prim a partire dal vertice **A**, disegnando l'albero e ritornando come risultato il valore del peso minimo. Si esplicitino i passi intermedi.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 02/09/2015 - Prova di programmazione (12 punti)

1. (2 punti)

Nel gioco degli scacchi l'alfiere si muove di un numero di caselle arbitrario lungo la diagonale e la diagonale inversa come rappresentato in figura:



Negli scacchi inoltre ogni pezzo assume idealmente un certo valore, ad esempio 1 per il pedone, 3 per il cavallo, etc. Con questo principio le caselle vuote assumono valore uguale a 0. Si realizzi un funzione in C che data una matrice di interi di dimensione NxN nota e contenente i pezzi degli scacchi con i loro pesi nonché celle vuote, calcoli e visualizzi le coordinate delle casella vuota della scacchiera nella quale occorrerebbe disporre un alfiere in modo che la somma dei valori dei pezzi disposti sulla stessa diagonale e diagonale inversa sia massima. Sia dato il seguente esempio di scacchiera 4x4:

	1	2	3	4
1	0	3	4	0
2	1	0	6	6
3	1	3	9	0
4	0	0	3	1

In questo caso occorre posizionare l'alfiere nella cella di riga 4 e colonna 2 (con somma 16).

2. (4 punti)

Siano dati i puntatori t1 e t2 a 2 alberi binari. Si scriva una funzione C

```
int TREEisomorph(struct node *t1, struct node *t2);
```

che verifichi se i 2 alberi sono o meno isomorfi. Per isomorfi si intendono alberi strutturalmente identici, cioè che contengono gli stessi valori disposti allo stesso modo. Si ipotizzi di avere a disposizione una funzione di comparazione tra chiavi

```
int KEYcmp(Key k1, Key k2);
```

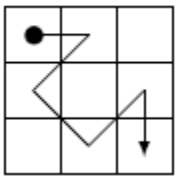
3. (6 punti)

Una mappa NxM contiene in ogni cella un valore intero. Nella mappa la posizione di partenza è sempre l'angolo in alto a sinistra, mentre quella di arrivo è sempre l'angolo in basso a destra. Sono ammessi spostamenti in tutte le celle adiacenti a quella corrente, inclusi quelli in diagonale, purché non si ripassi sulla medesima casella.

Si scriva una funzione C che, ricevuto come parametro la matrice contenente la mappa, individui un cammino che permetta di spostarsi dalla partenza all'arrivo massimizzando la somma dei valori delle celle attraversate. A parità di valore complessivo, inoltre, l'algoritmo deve prediligere soluzioni che richiedano il minor numero possibile di passi.

Esempio: per la mappa della figura a sinistra, una possibile soluzione è quella della figura a destra con peso 24:

1	2	-3
9	-9	7
0	1	4

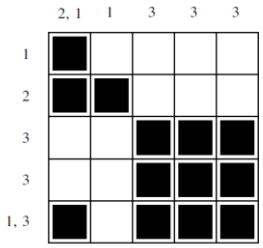


02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 02/09/2015 - Prova di programmazione (18 punti)

I **Nonogram** o **Paint by Numbers** (dipingere con i numeri) o **griddlers**, sono dei rompicapi logici grafici in cui le celle di una griglia NxM devono essere colorate o lasciate in bianco in base a dei numeri che compaiono a lato delle righe e delle colonne della griglia. In questo tipo di rompicapo, ogni numero indica quante celle consecutive devono essere riempite in una riga o in una colonna. Per esempio, un'etichetta di riga/colonna del tipo "4 8 3" significa che su quella riga/colonna ci sono 3 insiemi di 4, 8 e 3 celle nere rispettivamente da riempire in questo ordine, con almeno una cella bianca tra gruppi successivi di celle nere. Il gioco consiste nell'individuare una configurazione di celle nere congruente con l'etichettatura delle righe e delle colonne.

Esempio di versione originale del gioco dove il numero di gruppi di celle contigue colorate non è dato esplicitamente, ma si ricava.



Un primo file di testo il cui nome è passato sulla riga di comando contiene:

- nella prima riga del file il numero di righe N della griglia
- per ogni riga della griglia un gruppo di informazioni (una per riga del file):
  - numero di gruppi *ni* di celle nere contigue per quella riga della griglia (semplificazione rispetto alla versione originale del gioco)
  - *ni* interi che rappresentano la lunghezza di ciascuno di questi insiemi
- segue su una riga del file il numero di colonne M della scacchiera
- per ogni colonna della griglia un gruppo di informazioni (una per riga del file):
  - numero di gruppi *ni* di celle nere contigue per quella colonna della griglia (semplificazione rispetto alla versione originale del gioco)
  - *ni* interi che rappresentano la lunghezza di ciascuno di questi insiemi.

Per l'esempio di cui sopra il file conterrebbe:

```
5
1 1
1 2
1 3
1 3
2 1 3
5
2 2 1
1 1
1 3
1 3
1 3
```

Un secondo file di testo il cui nome è passato sulla riga di comando contiene una possibile soluzione da verificare nella forma di N righe ciascuna contenente M interi cha valgono 0 se la cella della griglia a quella riga e quella colonna è vuota, 1 se è piena.

Si scriva un programma C che:

1. legga i vincoli dal primo file di testo e li memorizzi in un'opportuna struttura dati in memoria
2. verifichi che la soluzione calcolata a mano e letta dal secondo file soddisfi le condizioni del nonogram
3. calcoli in modo ricorsivo una soluzione del gioco e la scriva su un terzo file di testo il cui nome è passato sulla riga di comandi. Il formato di questo terzo file sia lo stesso del secondo file.

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).***
- ***È consentito utilizzare chiamate a funzioni <u>standard</u>, quali ordinamento per vettori, inserzione/estrazione/ricerca relative a FIFO, LIFO, liste, BST, tabelle di hash e altre strutture dati, considerate come librerie esterne. Gli header file delle librerie utilizzate devono essere allegati all'elaborato. Le funzioni richiamate, inoltre, dovranno essere incluse nella versione del programma allegata alla relazione.***
- ***consegna delle relazioni (per entrambe le tipologie di prova di programmazione): entro lunedì 07/09/2015, alle ore 24:00, mediante caricamento su Portale. Le istruzioni per il caricamento sono pubblicate sul Portale nella sezione Materiale). QUALORA IL CODICE SPEDITO 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.***

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 26/01/2016 - Prova di teoria (12 punti)

1. (2 punti)
Sia data la sequenza di interi, supposta memorizzata in un vettore:

4 53 92 52 32 34 13 12 91 93 22

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)
Sia data una coda a priorità implementata mediante uno heap di dati. I dati siano formati da una coppia di interi 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:

(1,20) (4,32) (5,19) (3,51) (7, 28) (8,74) (9,9) (0,81) (10,17) (6,41) (2,37)

Si riporti la configurazione della coda a priorità dopo ogni inserzione. Al termine si cambi la priorità del dato in posizione 4 in 101 e si disegni la configurazione risultante della coda a priorità.

3. (2 punti)
Sia data la sequenza di chiavi "alpha", "beta", "delta", "epsilon", "zeta", "eta", "theta", "iota", "kappa". Si supponga che h("alpha")=2, h("beta")=14, h("delta")=18, h("epsilon")=17, h("zeta")=17, h("eta")=16, h("theta")=18, h("iota")=2, h("kappa")=14. Si riporti la struttura di una tabella di hash di dimensione 19, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si supponga di utilizzare l'open addressing con linear probing.

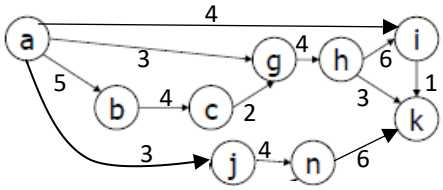
4. (2 punti)
Si determini mediante un algoritmo greedy un codice di Huffman ottimo per i seguenti simboli con le frequenze specificate:

a: 46 c: 20 g: 14 k: 3 i: 35 n: 13 o: 24 r: 19 t: 12 u: 17

5. (1 punto)
Si inseriscano in sequenza nella radice di un albero di ricerca binario supposto inizialmente vuoto le seguenti chiavi:

12 34 7 89 23

6. (2.5 punti)
Sia dato il seguente DAG pesato: considerando **a** come vertice di partenza, si determinino i cammini **massimi** tra **a** e tutti gli altri vertici. Qualora necessario, si trattino i vertici secondo l'ordine alfabetico e si assuma che la lista delle adiacenze sia anch'essa ordinata alfabeticamente.



02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 26/01/2016 - Prova di programmazione (18 punti)

N città sorgono lungo una strada rettilinea. Ogni città è caratterizzata da un nome, dal numero di abitanti e dalla sua distanza rispetto all'inizio della strada. La distanza tra due città è data dalla differenza, in valore assoluto, tra le relative distanze dall'inizio della strada. Le informazioni sulle città sono lette da un file, il cui nome è passato come parametro sulla riga di comando. Il file ha il seguente formato:

- la prima riga contiene N (numero intero)
- in ognuna delle N righe successive compaiono una stringa di al massimo 20 caratteri per il nome della città, un intero per la popolazione (in migliaia) e un intero per la distanza rispetto all'inizio della strada.

Esempio: contenuto del file

```
11
Piacenza 102 0
Fidenza 27 35
Parma 190 57
ReggioEmilia 171 83
Modena 185 115
Bologna 386 144
Imola 69 177
Faenza 58 193
Forlì 118 207
Cesena 97 225
Rimini 147 253
```

Si vuole pianificare in quali città localizzare K<N Autorità di Ambito Territoriale Ottimale (ATO).. Sia K passato come parametro sulla riga di comando. Il sottoinsieme di K città va scelto in modo tale da minimizzare la distanza media, per ogni abitante, dalla sede di Autorità ATO più vicina. Questo risultato si può ottenere minimizzando la sommatoria SD delle distanze di ogni abitante dall'Autorità ATO più vicina:

$$\mathrm{SD} = \sum_{i=0}^{num.citt\grave{a}-1} popolazione_i * distMinDaATO_i$$

Nella formula precedente *popolazione*$_i$ indica la popolazione dell'i-esima città, mentre *distMinDaAto*$_i$ ne indica la distanza dalla sede di Autorità ATO più vicina.

Si scriva in C un programma che, ricevuti come argomenti al main in nome del file e K:

1. legga il file e generi una opportuna struttura dati
2. calcoli la mutua distanza tra ogni città e tutte le altre, memorizzando tali informazioni in una struttura dati appropriata. La complessità dell'algoritmo deve essere O(N^2)
3. calcoli, mediante un algoritmo ricorsivo, la soluzione ottima richiesta (un sottoinsieme di K città in cui collocare le sedi ATO, in modo da minimizzare SD)
4. date due soluzioni, lette da due file (ognuno contenente l'elenco di K città, una per riga), determini quale delle due soluzioni è migliore.

Nei punti 3 e 4, al fine di determinare SD, si realizzi una opportuna funzione SommaDistanze, la quale, a partire dal vettore delle popolazioni e da un insieme di città selezionate come sedi dell'Autorità ATO, calcoli il relativo SD. La complessità di questa funzione è oggetto di valutazione. Si noti che è possibile realizzare SommaDistanze con un algoritmo O(N).

Indicare esplicitamente sulla prima pagina del foglio il modello di Calcolo Combinatorio utilizzato motivando la scelta in al massimo 3 righe.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 26/01/2016 - Prova di programmazione (12 punti)

1. (**2 punti**)

Scrivere la funzione `char *charErase (char *str, int *pos);` che riceve una stringa `str` e un vettore di interi `pos` e restituisce la stringa ottenuta da `str` cancellando i caratteri nelle posizioni indicate dal vettore `pos`. La stringa ritornata va opportunamente allocata. Il vettore `pos` ha dimensione ignota, ma il suo ultimo elemento contiene il valore -1.

Ad esempio, se la stringa `str` è `T h i s I s A S t r i n g` e il vettore `pos` contiene `7 4 2 0 11 -1` occorre cancellare dalla stringa `str` i caratteri `S, I, i, T, n` e restituire la stringa contenente

`h s s A t r i g`

2. (**4 punti**)

Sia data una matrice sparsa di `float` realizzata mediante un ADT (tipo `matr_t`) di prima categoria, che internamente gestisce i dati mediante una lista di liste. Si ricordi che una matrice si dice *sparsa* se viene realizzata con una struttura dati che alloca memoria per i soli dati non nulli. La lista di liste è così composta:

- una lista di primo livello contiene un elemento per ogni riga della matrice contenente almeno un dato non nullo,
- ad ogni elemento di questa prima lista corrisponde una sotto-lista, contenente un elemento per ogni dato non nullo della riga.

Al tipo `matr_t` corrisponde una `struct` wrapper contenente le dimensioni della matrice e il puntatore al primo elemento della lista di righe.

Si scriva la funzione `MatrWrite`, di prototipo

```
void MatrWrite (matr_t *M, float d, int r, int c);
```

che scrive nella matrice il dato `d` nella casella di indici `r`, `c`. Sono possibili quattro casi:

- `d != 0.0` e la matrice sparsa non contiene dati in riga `r` e colonna `c`. Il dato va aggiunto
- `d != 0.0` e la matrice sparsa contiene già un dato in riga `r` e colonna `c`. Il vecchio dato viene sostituito da quello nuovo
- `d == 0.0` e la matrice sparsa non contiene dati in riga `r` e colonna `c`. Non si fa nulla
- `d == 0.0` e la matrice sparsa contiene un dato in riga `r` e colonna `c`. Il dato va rimosso.

Si noti che NON si può far riferimento a funzioni di libreria sulle liste. Si richiede la realizzazione della funzione, nonché la definizione dei tipi `struct` (compreso `matr_t`) utilizzati.

3. (6 punti)

Sia data una stringa `str` di al massimo 30 caratteri e un vettore `lungh` di `num` interi distinti che rappresentano la lunghezza delle sottostringhe in cui si vuole decomporre la stringa `str`. Si scriva una funzione ricorsiva in C `void decomponi(char *str, int num, int *lungh);`

che visualizzi **una** delle possibili decomposizioni di `str` usando sottostringhe di lunghezza specificata nel vettore `lungh`.

Esempi:

- se `str= "tentativo", num = 3, lungh` contiene `2, 5, 7`, una delle possibili decomposizioni è `"te" "nt" "ativo"`
- se `str= "tentativo", num = 2, lungh` contiene `2, 4,` non vi sono decomposizioni.

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 venerdì 29/01/2016, alle ore 24:00, 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 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.***

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 22/02/2016 - 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, 2-1, 0-3, 4-0, 1-6, 3-4, 0-8, 1-3, 6-7, 10-3

si applichi un algoritmo di on-line connectivity con quickfind, 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. (2.5 punti)

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

$T(n) = T(n-1) + n^2 \quad n>1$

$T(1) = 1 \quad n=1$

ricordando che $\sum_{i=0}^{n-1} i^2 = \frac{n(n+1)(2n+1)}{6}$

3. (1 punto)

Si ordini in maniera ascendente mediante counting-sort il seguente vettore di interi:

3 0 5 0 6 1 4 0 8 1 2 1 8 0 2

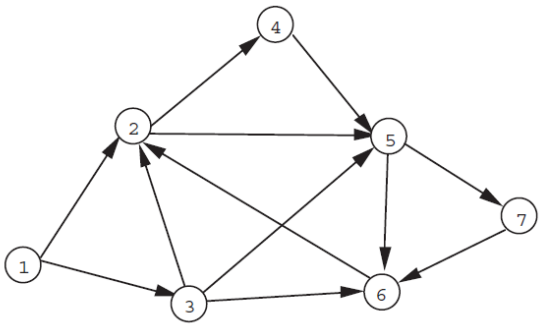
Si indichino le strutture dati usate nei passi intermedi.

4. (2 punti)

Sia data la sequenza di stringhe **I**n **B**ob **U**ntil **X**eno **E**dge **G**raph **Z**eta **H**otel **Y**ou, dove ogni stringa ha come chiave il suo primo carattere, individuato dal suo ordine progressivo nell'alfabeto (A=1, ..., Z=26). Si riporti la struttura 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.

5. (1.5 + 1.5 punti)

Sia dato il seguente grafo orientato:

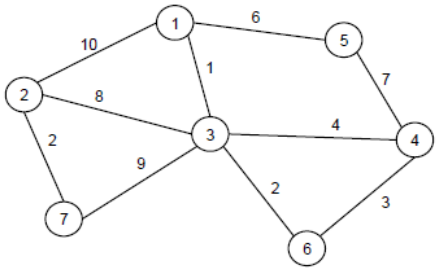


- se ne effettui una visita in ampiezza, considerando **1** come vertice di partenza (**1.5 punti**)
- lo si ridisegni, etichettando ogni suo arco come T (tree), B (back), F (forward), C (cross), considerando **1** come vertice di partenza (**1.5 punti**).

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

6. (1.5 + 0.5 + 0.5 punti)

Si determini mediante l'algoritmo di Kruskal l'albero ricoprente minimo per il grafo in figura illustrando i passaggi intermedi del procedimento adottato (**1.5 punti**).



Dopo aver determinato la soluzione, si consideri l'aggiunta di un nuovo arco (1-4) di costo 5. Questo arco è utile per migliorare il costo della soluzione? Perché (**0.5 punti**)? Si consideri anche l'arco (1-7) di costo 5. Questo arco è utile per migliorare il costo della soluzione? Perché (**0.5 punti**)?

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 22/02/2016 - Prova di programmazione (18 punti)

Un grafo non orientato, pesato e **colorato** è memorizzato su un file mediante l'elenco dei suoi archi, con il seguente formato:

$$idV_1\ colV_1\ val\ idV_2\ colV_2$$

che indica che il vertice con identificatore idV_1 e colore $colV_1$ è connesso con un arco non orientato di peso val al vertice con identificatore idV_2 e colore $colV_2$. Gli identificatori alfanumerici sono di lunghezza massima 20 caratteri. I colori sono 2 e sono identificati dalle stringhe ROSSO e NERO. I pesi sono valori interi e positivi.

Una volta acquisito il grafo in una opportuna struttura dati, implementare le seguenti funzionalità che risolvono 2 problemi distinti:

- verifica di congruenza della colorazione dei vertici: non deve succedere che lo stesso vertice compaia con colori diversi in archi diversi
- calcolo del cammino semplice a peso massimo all'interno del grafo che soddisfi il seguente vincolo:

  nel cammino un vertice NERO può essere seguito da un vertice sia NERO che ROSSO, mentre un vertice ROSSO può solo essere seguito da un vertice NERO

  Una volta individuato tale cammino si provveda a stamparlo a video elencando i nomi dei nodi attraversati dalla sorgente alla destinazione
- identificazione del sottografo che soddisfa i seguenti vincoli :
  - è connesso
  - la somma dei pesi dei suoi archi è massima
  - il numero di vertici neri e rossi che lo compongono differisce al più di 2.

Poiché il numero di vertici del grafo non è noto a priori, si suggerisce di calcolarlo tramite lettura preliminare del file e/o caricamento in una tabella di simboli. Si può assumere che non esistano archi duplicati. Non è lecito assumere nessuna forma di ordinamento degli archi. E' richiesta la scrittura della funzione di verifica di connettività, nella quale è ammesso richiamare una funzione standard di libreria di visita in profondità.

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):***
  - ***LAUREANDI: entro martedì 24/02/2016 ore 24:00***
  - ***NON LAUREANDI: entro giovedì 25/02/2016, alle ore 24:00***

  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 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.

NB: per i LAUREANDI:

- ***indicare di essere laureandi sul compito scritto consegnato***
- ***gli orali si svolgeranno lunedì 29 in orario e aula che verranno pubblicati successivamente sul Portale.***

Queste modalità, concordate con la Segreteria, consentono a chi supera l'esame di fare domanda di laurea entro i tempi stabiliti.

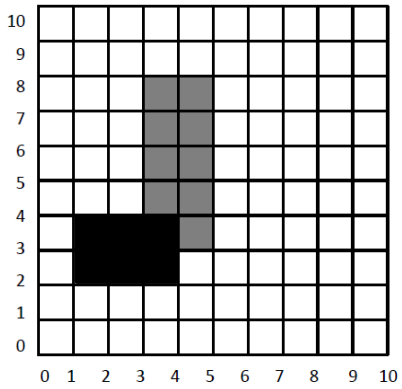
02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 22/02/2016 - Prova di programmazione (12 punti)

1. (**2 punti**)

Nel primo quadrante del piano cartesiano è individuata una regione quadrata i cui vertici in basso a sinistra e in alto a destra hanno coordinate (0,0) e (100,100). In un file è descritta una sequenza di rettangoli con i lati paralleli agli assi cartesiani. Ogni rettangolo è individuato dalle coordinate dei vertici in basso a sinistra e in alto a destra. Le coordinate sono interi compresi tra 0 e 100, estremi inclusi. Scrivere la funzione `int areaTot(FILE *fp);` che riceve come parametro un puntatore al file (già aperto) e restituisce il valore totale dell'area coperta dai rettangoli. Nel caso di intersezione di rettangoli l'area è contata una sola volta.

Esempio (semplificato con coordinate x e y nell'intervallo (0..10): se il file contiene sulla prima riga 1 2 4 4 e sulla seconda 3 3 5 8, l'area coperta vale 15.



2. (**4 punti**)

Sia data una lista non ordinata di interi e un valore intero inteso come soglia. Si scriva una funzione C che divida la lista in 2 : nella prima lista compaiono gli elementi della lista originaria minori della soglia, nella seconda quelli maggiori o uguali alla soglia. Sia mantenuto l'ordine relativo della lista originaria nelle 2 liste così create. **Si noti che NON si può far riferimento a funzioni di libreria sulle liste.** Si usi un ADT di I categoria per le liste con una `struct` wrapper di tipo `lista_t`. La funzione C sia compatibile con la chiamata effettuata dal seguente frammento del `main`:

```
lista_t *L0, *L1, *L2;
…
// acquisizione della lista L0
…
L1 = split_list(n, L0, &L2);
```

La funzione riceve come parametro la lista originale L0, ritorna la prima lista L1 e, by reference, la seconda L2.

Esempio: se la lista L0 in ingresso contiene i valori 7, 8, 25, 2, 9, -5, 10, 37 e la soglia è n=18, la prima lista L1 dovrà contenere 7, 8, 2, 9, -5, 10 e la seconda L2 25,37.

3. (**6 punti**)

Sia dato un insieme di n interruttori e un insieme di m lampadine, inizialmente tutte spente. Ogni interruttore comanda un sottoinsieme delle lampadine: se lo si preme, ogni lampadina da esso controllata commuta di stato, cioè se è accesa si spegne, se è spenta si accende. L'informazione su quale sottoinsieme di lampadine agisce ogni singolo interruttore è memorizzata in una matrice di interi di dimensioni n x m. L'elemento [i,j] della matrice vale 1 se l'interruttore i controlla la lampadina j, 0 altrimenti. Si scriva una funzione ricorsiva in C in grado di determinare l'insieme minimo di interruttori per accendere tutte le lampadine. Si noti che una lampadina è accesa se e solo se è dispari il numero di interruttori premuti che la controllano.

Esempio: se per n=4 e m=5 la matrice contiene

1	1	0	0	1
1	0	1	0	0
0	1	1	1	0
1	0	0	1	0

il minimo numero di interruttori da premere è 3 e questi interruttori sono gli interruttori 0, 1 e 3.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II
Appello del 14/06/2016 - 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:

1-9, 2-3, 0-5, 4-2, 0-8, 3-6, 0-10, 1-5, 6-9, 9-4

si applichi un algoritmo di on-line connectivity con 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. (0.5 + 0.5 punti)
Si ordini in maniera ascendente il seguente vettore di interi:

10 23 6 78 54 6 90 20

mediante merge-sort (**0.5 punti**) e bottom-up merge-sort (**0.5 punti**). Si indichino le strutture dati usate nei passi intermedi.

3. (2 punti)
Sia dato un albero binario con 13 nodi. Nella visita si ottengono le 3 seguenti sequenze di chiavi:

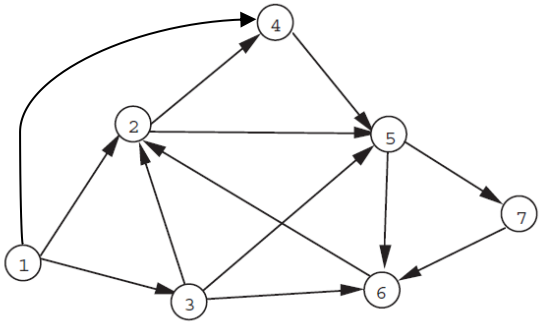
preorder	10	5	16	7	20	17	4	2	1	7	8	13	11
inorder	16	7	17	20	4	5	10	1	2	8	13	7	11
postorder	17	4	20	7	16	5	1	13	8	11	7	2	10

Si disegni l'albero binario di partenza.

4. (1 punto)
Si supponga di aver memorizzato tutti i numeri compresi tra 1 e 300 in un BST e che si stia cercando il numero 231. Quali tra queste non possono essere le sequenze esaminate durante la ricerca? Perché?

250	200	240	260	255	220	230	231
150	300	200	250	220	240	235	231
270	100	200	300	260	190	220	231

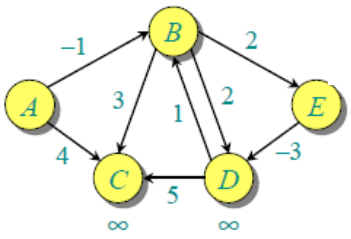
5. (2.0 + 1.5 + 1.5 punti)
Sia dato il seguente grafo orientato:



- se ne effettui una visita in profondità, considerando **1** come vertice di partenza (**2 punti**)
- lo si ridisegni, etichettando ogni suo arco come T (tree), B (back), F (forward), C (cross), considerando **1** come vertice di partenza (**1.5 punti**)
- se ne effettui una visita in ampiezza, considerando **1** come vertice di partenza (**1.5 punti**).

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

6. (1.5 + 0.5 punti)
Considerando **A** come vertice di partenza, si determinino mediante l'algoritmo di Bellman-Ford i valori di tutti i cammini minimi che collegano **A** con ogni altro vertice (**1.5 punti**). Si riportino i passaggi rilevanti. L'algoritmo di Dijkstra sarebbe pervenuto allo stesso risultato? Si giustifichi la risposta (**0.5 punti**).



02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 14/06/2016 - Prova di programmazione (18 punti)

Un negozio online ha la necessità di gestire il proprio catalogo di prodotti contenuto in un file con il seguente formato:

productID productName productPrice #availability category

dove:

- productID è il codice univoco alfanumerico di 10 caratteri
- productName è il nome (stringa di dimensione variabile di massimo 255 caratteri)
- productPrice è il prezzo (valore reale positivo)
- #availability è il numero di unità disponibili (valore intero non negativo)
- Category: è un codice univoco alfanumerico di 10 caratteri.

Per quanto riguarda le categorie si noti che ogni prodotto appartiene a una specifica categoria merceologica. Le categorie sono al massimo 100. Non si acquisisce in modo esplicito un elenco di codici di categoria. L'elenco va costruito durante l'inserimento dei dati

Il catalogo deve poter permettere le seguenti operazioni:

1. inserimento di un nuovo prodotto o aggiunta di unità disponibili per un prodotto già esistente
2. ricerca di un prodotto nel catalogo dato il codice
3. ricerca di un prodotto dato codice e categoria. Occorre localizzare prima la categoria e limitare quindi la ricerca ai prodotti della categoria
4. stampa dei prodotti di una data categoria ordinati per codice o nome a scelta dell'utente
5. ricerca di un prodotto per nome: il nome può essere parziale (dato solo dalla parte iniziale terminata col carattere asterisco '*'): in tal caso è possibile che la ricerca dia più risultati
6. valutazione di soddisfacibilità di un ordine.

Per i punti 2. e 3. la ricerca per prodotti deve essere al peggio di costo logaritmico nel numero di prodotti. Per il punto 3. la ricerca per categorie deve essere al peggio di costo lineare nel numero di categorie. Per il punto 5. la ricerca deve ritornare TUTTI gli elementi che rispettino i criteri di ricerca, ossia tutti i prodotti col medesimo nome. Per il punto 6. il programma deve acquisire da file un elenco di prodotti desiderati in quantità indicate e valutare se questi siano disponibili. Sia tale file caratterizzato dal seguente formato:

1. sulla prima riga è presente un unico intero N ad indicare il numero di prodotti d'interesse
2. sulle N righe successive sono presenti N coppie <codice_prodotto> <quantità>, in ragione di una per riga, a indicare i contenuti dell'ordine.

In caso l'ordine sia soddisfacibile, si provveda anche a calcolare l'ammontare complessivo dell'ordine stesso e ad aggiornare le disponibilità dei prodotti in magazzino.

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 venerdì 17/06/2016 ore 24:00 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 discussione o esame orale, sulla base dell'elaborato svolto in aula. Non verranno corretti i compiti di cui non sarà stata inviata la relazione nei tempi stabiliti.***

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 14/06/2016 - Prova di programmazione (12 punti)

1. (2 punti)

Scrivere una funzione

```
void matMax (int **m, int r, int c);
```

che riceve una matrice m di interi di r righe e c colonne e visualizza (a video) la posizione di tutti gli elementi che sono strettamente maggiori di tutti gli elementi adiacenti. Ad esempio, data la matrice seguente:

	0	1	2	3
0	5	2	3	1
1	4	1	6	4
2	3	0	5	2

occorre visualizzare: (0, 0) e (1, 2).

2. (4 punti)

Si scriva una funzione `separaParole` che, a partire da una stringa contenente parole separate da spazi, generi un vettore di stringhe (vettore di puntatori a caratteri) contenente le singole parole, prive di spazi e duplicate mediante allocazione dinamica. La funzione deve essere richiamabile, ad esempio, nel modo seguente:

```
char frase[1000], **parole;
int n, i;
…
fgets(frase,1000,stdin);
n = separaParole(frase,&parole);
for (i=0; i<n; i++)
  printf("parola %d -> %s\n", parole[i]);
…
```

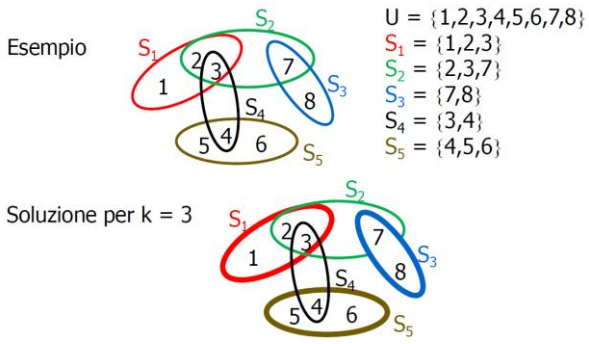
Si scriva il prototipo e il contenuto della funzione. Si noti che il puntatore parole viene utilizzato per un vettore (allocato dinamicamente) di puntatori a `char`: ogni casella punterà a una delle parole generate dalla stringa originale. Il vettore di stringhe viene ritornato per riferimento (o meglio, se ne passa per valore il puntatore). Per semplicità è lecito ipotizzare che gli spazi che separano le parole non siano multipli.

3. (6 punti)

Sia U l'insieme degli interi compresi 1 e 8: U= {1, 2, 3, 4, 5, 6, 7, 8}. Sia S una matrice n x 9 di interi che su ciascuna delle n righe contiene un sottoinsieme di U terminato dal valore 0. La matrice viene ricevuta da una funzione come parametro formale con numero di righe ignoto. Il numero di righe compare come ulteriore parametro. Dato un intero k, ulteriore parametro della funzione, si scriva una funzione ricorsiva in C che visualizzi, se esiste, una collezione di k sottoinsiemi la cui unione sia U. La funzione abbia il seguente prototipo:

```
void cover(int S[][9], int n, int k);
```

Esempio:



02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II
Appello del 09/09/2016 - Prova di teoria (12 punti)

1. (1 punto)
Si ordini in maniera discendente il seguente vettore di interi mediante selection sort:

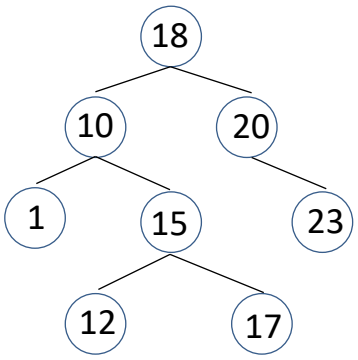
5 4 10 7 6 4 0 1 6 5 0 2 7 5 0 3 0 4 9

Si indichino i passaggi principali.

2. (1 punto)
Si esprima in notazione prefissa e postfissa la seguente espressione aritmetica mediante visita dell'albero binario corrispondente: ((A - B) / (C+(D*E))) * ((F-G) / (H*I))

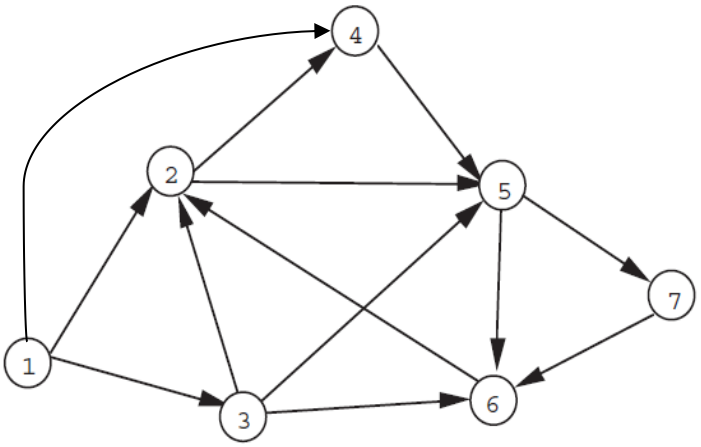
3. (2 punti)
Sia data la sequenza di chiavi RAWFZIEV dove ciascun carattere è individuato dal suo ordine progressivo nell'alfabeto (A=1, ..., Z=26). Si riporti la struttura di una tabella di hash di dimensione 17, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si supponga di utilizzare l'open addressing con double hashing. Si definiscano opportune funzioni di hash h_1 e h_2.

4 (1 punto)
Si inseriscano in foglia nel BST di figura in sequenza le chiavi: 11, 4, poi si cancelli la chiave 15, riportando a ogni passo l'albero risultante.



5 (0.5+0.5+0.5 punti)
Si visiti in pre-order, in-order e post-order l'albero binario dell'esercizio 4.

6 (1.5 + 1.5 + 2.5 punti)
Sia dato il seguente grafo orientato:



- lo si rappresenti come lista delle adiacenze e matrice delle adiacenze (**1.5 punti**)
- se ne effettui una visita in ampiezza, considerando **1** come vertice di partenza (**1.5 punti)**
- se ne determinino le componenti fortemente connesse mediante l'algoritmo di Kosaraju (**2.5 punti).**

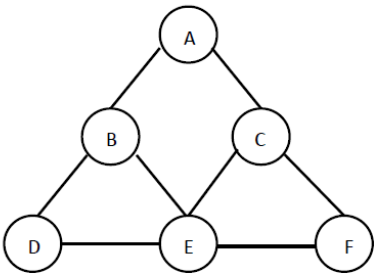
Qualora necessario, si trattino i vertici secondo l'ordine numerico.

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 09/09/2016 - Prova di programmazione (18 punti)

Nella Teoria dei Grafi si definisce **Insieme Indipendente** o **Insieme Stabile** un insieme IS di vertici di un grafo G = (V, E) tale per cui, qualunque coppia di vertici $v_1 \in$ IS e $v_2 \in$ IS si consideri, non esiste un arco che li renda adiacenti $(v_1, v_2) \notin$ E.
Dato un grafo G, un **Insieme Indipendente Massimale** è un insieme indipendente che non è sottoinsieme di alcun altro Insieme Indipendente.

Esempio: dato il seguente grafo non orientato {A,D,E,F} non è un insieme indipendente, {D,F} è un insieme indipendente ma non massimale, {A,D,F} è un insieme indipendente massimale.



Un grafo non 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. Poiché il numero di vertici del grafo non è noto a priori, si suggerisce di calcolarlo tramite lettura preliminare del file e/o caricamento in una tabella di simboli. 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 vuole scrivere un programma C in grado di:

- ricevere 3 nomi di file parametri sulla riga di comando:
  - il primo file contenente la descrizione del grafo
  - il secondo file contenente un elenco di vertici uno per riga
  - il terzo file su cui memorizzare il risultato
- leggere il grafo e memorizzarlo in un'opportuna struttura dati
- verificare se il grafo letto dal secondo file è davvero un insieme indipendente
- identificare un insieme indipendente massimale e memorizzarlo su un terzo file. A video si visualizzi la cardinalità dell'insieme così identificato, detta numero di indipendenza.

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 lunedì 12/09/2016 ore 24:00 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 discussione o esame orale, sulla base dell'elaborato svolto in aula. Non verranno corretti i compiti di cui non sarà stata inviata la relazione nei tempi stabiliti.***

02MNO Algoritmi e Programmazione 01JKE APA I / 01JKF APA II

Appello del 09/09/2016 - Prova di programmazione (12 punti)

1. (2 punti)

Scrivere la funzione

```
int subMatMax (int **mat, int r, int c, int n);
```

che ricevuta una matrice di interi `mat` con `r` righe e `c` colonne, ricerchi la sotto-matrice quadrata di dimensione `n` la cui somma degli elementi è massima.

Esempio: data la matrice seguente

	0	1	2	3
0	5	2	3	1
1	3	1	6	4
2	3	0	5	2

con `r=3` e `c=4`, se `n=2` la sottomatrice con somma dei suoi elementi massima, pari a 17, è riportata nelle celle con sfondo grigio.

2. (4 punti)

Scrivere la funzione:

```
int distance (nnode_t *root, int key1, int key2);
```

che, ricevuto un BST di radice root (di chiave intera) e due valori interi `key1` e `key2`, restituisce il numero di archi che è necessario attraversare per raggiungere il nodo di chiave `key1` da quello di chiave `key2` (o viceversa).

Suggerimento: partendo dalla radice, la funzione `distance` proceda ricorsivamente lungo il BST fintanto che il percorso per le chiavi `key1` e `key2` è comune e dia origine a due visite separate quando il percorso di ricerca per le due chiavi si suddivide.

3. (6 punti)

Come regalo di compleanno u ragazzo vuole comprare alla sua ragazza *k* libri. In libreria sono disponibili *n* libri, ciascuno dei quali appartiene a 1 e 1 solo tra *m* generi letterari. Vale $k \leq m$. I *k* libri devono essere di generi diversi. Sia dato un vettore `int *vet` di `n` elementi che registra nella posizione *i* il genere del libro i-esimo ($1 \leq$ `vet[i]` $\leq m$). Si scriva la funzione:

```
int birthday (int *vet, int n, int m, int k);
```

che stampa tutte le possibili maniere diverse di scegliere *k* libri di generi diversi.

Esempio 1: se

`vet = (2 1 1 4 3) n=5` (libri numerati da 0 a 4) `m=4` (generi numerati da 1 a 4) `k=3`

le soluzioni sono:

`(0,1,3), (0,1,4), (0,2,3), (0,2,4), (0,3,4), (1,3,4), (2,3,4)`

Esempio 2: se

`vet = (1 2 3 1 2 3) n=6` (libri numerati da 0 a 5) `m=3` (generi numerati da 1 a 3) `k=2`

le soluzioni sono:

```
(0,1), (0,2), (0,4), (0,5), (1,2), (1,3),
(1,5), (2,3), (2,4), (3,4), (3,5), (4,5)
```

03MNO Algoritmi e Programmazione

Appello del 04/02/2017 - 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:

`"si" 10  "inserisca" 15  "in" 5  "sequenza" 18  "il" 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
	il		si	in		inserisca	sequenza			
	4		10	5		15	18			

Si disegnino lo heap e il vettore `qp` ai diversi passi dell'inserzione. Al termine si cambi la priorità di `"il"` 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: 24 B: 12 C: 10 D: 8 E: 6 M: 19 O: 14 Q: 3 T: 11 U: 7

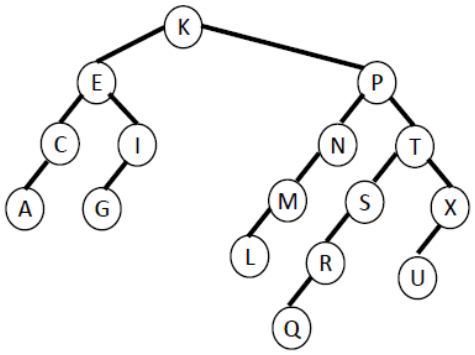
3. (2 punti)

Sia data la sequenza di chiavi intere: 11 144 267 312 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 double hashing, definendo le opportune funzioni di hash.

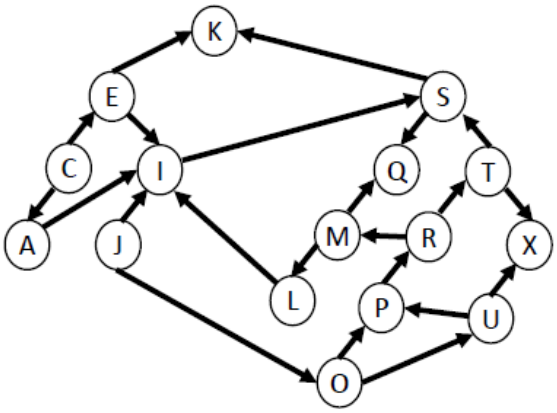
4. (2 punti)

Si partizioni il seguente BST attorno alla chiave M, evidenziando i passaggi rilevanti:



5. (2 + 1.5 punti)

Sia dato il seguente grafo orientato:



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

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

03MNO Algoritmi e Programmazione

Appello del 04/02/2017 - Prova di programmazione (18 punti)

Un *social network* vuole focalizzare meglio le sue attività rivolte a gruppi fortemente omogenei. Per raggiungere tale scopo deve individuare questi gruppi all'interno della rete. Essa è costituita da amici tra i quali esistono relazioni di amicizia più o meno intense ed è modellata mediante un grafo non orientato e pesato G = (V, E) dove i vertici sono gli amici, gli archi le relazioni di amicizia e i pesi interi e positivi le intensità di queste relazioni.

Un gruppo omogeneo di amici corrisponde a un sottografo **completo** di G, detto **cricca** (o **clique**) in Teoria dei Grafi. Una cricca è **massimale** se e solo se non esistono altri sottografi completi di cui essa è un sottoinsieme proprio. Il grafo è memorizzato in un file mediante l'elenco dei suoi archi pesati con il seguente formato:

idV_1 idV_2 w

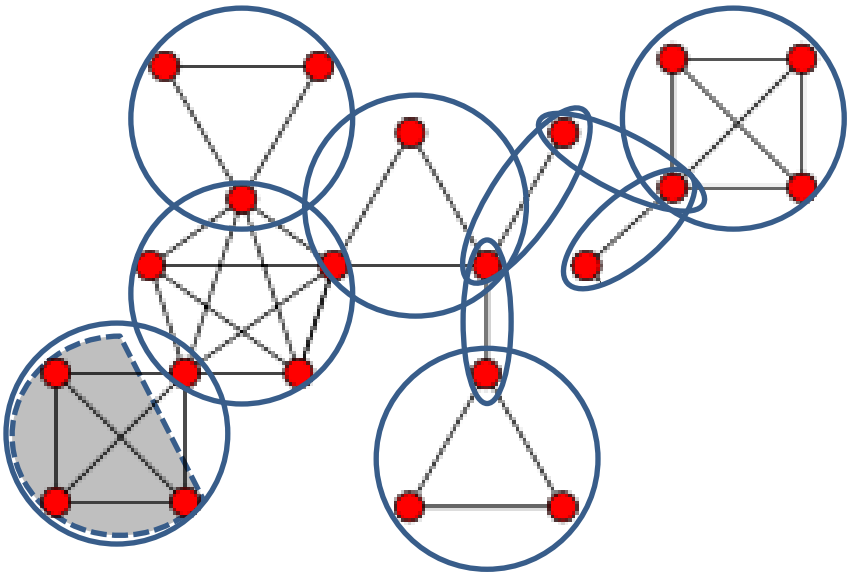
che indica che l'arco (idV_1, idV_2) ∈ E, dove idV1∈ V e idV_2∈ V e w è il peso dell'arco. Poiché il numero di vertici del grafo non è noto a priori, si suggerisce di calcolarlo tramite lettura preliminare del file e/o caricamento in una tabella di simboli. Ogni vertice è individuato mediante un identificatore alfanumerico idV 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 che, ricevuti sulla riga di comando

- come primo argomento il nome del file contenente la descrizione del grafo
- come secondo argomento il nome di un file che contiene una possibile cricca (vertici uno per riga)

- verifichi se la soluzione proposta sia davvero una cricca massimale, cioè se è un sottografo completo massimale. *Si osservi che un sottografo completo non è una cricca massimale se esiste almeno un vertice ad esso esterno adiacente a tutti i suoi vertici.*
- determini tra tutte le cricche massimali quella con massimo numero di vertici
- determini tutte le cricche massimali, memorizzandole in un'opportuna struttura dati (vettore di liste). *Si noti che il massimo numero di cricche massimali è |V|*
- per ogni cricca massimale tra quelle individuate al punto precedente, identifichi il ciclo hamiltoniano (cammino semplice che tocca tutti i vertici richiudendosi su quello di partenza) a peso massimo.

Esempio:



La figura riporta cricche massimali con 2, 3, 4 e 5 vertici. Il sottografo completo di 3 vertici con sfondo grigio non è una cricca massimale, in quanto è incluso (come sottoinsieme proprio) nella cricca massimale di 4 vertici.

03MNO Algoritmi e Programmazione

Appello del 04/02/2017 - Prova di programmazione (12 punti)

1. (**2 punti**)

Data una matrice quadrata `N` x `N`, con `N` dispari, scrivere una funzione caratterizzata dal seguente prototipo:

```
void sommaCornici(int **mat, int N, int **vet);
```

che memorizzi nel vettore `vet`, allocato della dimensione opportuna dentro alla funzione, la somma degli elementi su ogni cornice della matrice.

A titolo di esempio, la seguente matrice 5x5 è caratterizzata dalla presenza di 3 cornici, evidenziate per facilità di comprensione da toni diversi (bianco, grigio chiaro, grigio scuro). Si osservi che esiste una cornice formata dal solo elemento centrale.

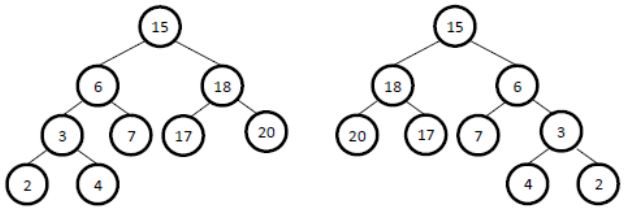
1	2	3	4	5
6	7	8	9	0
1	2	3	4	5
6	7	8	9	0
1	1	1	1	1

2. (**4 punti**)

Dato un albero binario cui si accede tramite un puntatore alla radice, si scriva una funzione C con il seguente prototipo:

```
link mirror(link root);
```

che costruisca l'albero binario "speculare", cioè quello in cui sono scambiati figlio sinistro e figlio destro, ritornando come risultato il puntatore alla sua radice, come nell'esempio seguente:



I nodi contengono interi. Non è ammesso l'uso di funzioni di libreria sugli alberi.

3. (6 punti)

Una matrice di interi `n` x `n` contiene celle bianche (1) e nere (0). Esiste un comando che, data una riga o una colonna, cambia di colore tutte le celle di quella riga o colonna.

Si scriva una funzione in grado di individuare un insieme **minimo** di comandi che, se attivati, permettano di rendere la matrice interamente bianca.

Esempio: data la matrice

0	1	0
1	0	1
0	1	0

i comandi cambia riga 0, riga 2 e colonna 1 la trasformano in una matrice di tutti 1.

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 martedì 07/02/2017, alle ore 24:00, 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 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.***

3MNO Algoritmi e Programmazione

Appello del 22/02/2017 - Prova di teoria (12 punti)

1. (2.5 punti)

Sia data la generica equazione alle ricorrenze

$$T(n) = aT(n/b) + cn \qquad n>1$$
$$T(1) = 1 \qquad n=1$$

dove a, b e c sono coefficienti interi, mediante il metodo dello sviluppo (unfolding), se ne determinino le 3 soluzioni per i casi a=b, a<b e a>b.

2. (2 punti)

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

10 1 27 14 17 0 8 55 19 41 23 91 31 37 7 3 13

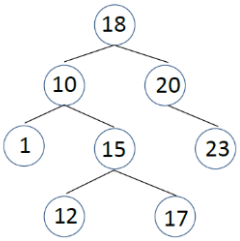
Si eseguano i primi 2 passi dell'algoritmo di quicksort per ottenere un ordinamento ascendente**.** NB: I passi sono da intendersi, impropriamente, come in ampiezza sull'albero della ricorsione, non in profondità. Si chiede, pertanto, che siano ritornate le 2 partizioni del vettore originale e le due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

3. (1 punto)

Si converta la seguente espressione da forma infissa a forma postfissa: A * ((B * C) + (D * (E + F)))

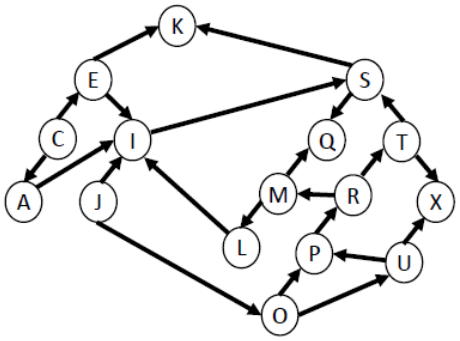
4. (2 punti)

Si inseriscano in radice nel BST di figura in sequenza le chiavi: 16, 19 e 22 e poi si cancelli la chiave 19. Si disegni l'albero ai passi significativi.



5. (2.5 punti)

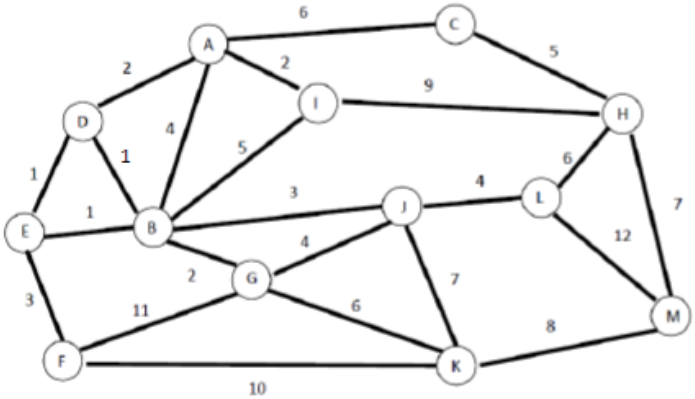
Dato il seguente grafo orientato:



se ne determinino mediante l'algoritmo di Kosaraju le componenti fortemente connesse. Si consideri **M** come vertice di partenza e, qualora necessario, si trattino i vertici secondo l'ordine alfabetico. Si riportino i passaggi rilevanti:

6. (2 punti)

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.



03MNO Algoritmi e Programmazione

Appello del 22/02/2017 - Prova di programmazione (18 punti)

Un supermercato offre prodotti acquistabili singolarmente a prezzo pieno e offerte speciali a prezzo ridotto. In ogni offerta sono raggruppati almeno due prodotti. Tutti i prodotti di un'offerta sono venduti congiuntamente. Il cliente può sfruttare ogni offerta al massimo una volta.

I prodotti e le offerte del supermercato sono riportati in un primo file (`catalogo.txt`), organizzato come segue:

- sulla prima riga sono presenti due interi `N e O`, rappresentanti il numero di prodotti disponibili e il numero di offerte
- sulle `N` righe successive sono riportati i prodotti, ognuno mediante un identificativo alfanumerico e il prezzo unitario (intero)
- seguono `O` blocchi di righe, in cui sono riportati i dettagli di ogni offerta, una per blocco, organizzata come segue:
  - una terna con l'identificativo alfanumerico per l'offerta, il prezzo (intero) e il numero `X` di oggetti distinti che l'offerta comprende
  - seguono `X` coppie, una per riga, a rappresentare l'identificativo di ogni prodotto incluso nell'offerta e la sua quantità

La lista della spesa è contenuta in un secondo file (`spesa.txt`), organizzato come segue:

- sulla prima riga presente un intero `M`, rappresentante il numero di prodotti distinti da acquistare
- sulle `M` righe successive sono riportati l'identificativo di ogni prodotto voluto e la quantità da acquistare (al massimo 9 unità per pezzo).

Esempio di `catalogo.txt` (indentato per leggibilità):

```
6 4
      Latte 10
      Pane 6
      Biscotti 16
      Nutella 12
      Zucchero 3
      Caffe 3
OFF1 19 2
      Latte 2
      Caffe 1
OFF2 28 2
      Nutella 2
      Pane 1
OFF3 10 2
      Pane 2
      Zucchero 1
OFF4 18 2
      Latte 2
      Zucchero 1
```

Esempio di `spesa.txt`:

```
3
Latte 4
Nutella 1
Pane 3
```

Si scriva un programma in C che, a partire dai due file sopra descritti:

- generi una struttura dati, in cui acquisire i dati del primo file, realizzata mediante due vettori dinamici, uno per i prodotti e uno per le offerte. Ogni offerta contiene una collezione di ***riferimenti*** (puntatori o indici) ai prodotti compresi: la si realizzi a scelta come vettore o lista. Occorre inoltre realizzare, per ogni prodotto, la lista delle offerte che includono tale prodotto (si consiglia una lista di ***riferimenti*** mediante indici, nel caso di vettore di offerte, o puntatori a offerte)
- fornisca una funzione che, dato l'identificativo di un prodotto, elenchi tutte le offerte in cui il prodotto compare
- fornisca una funzione che, dati gli identificativi di due offerte, elenchi gli eventuali prodotti in comune
- una volta letto il secondo file, determini l'insieme ottimo di acquisti di prodotti singoli e/o di offerte, che permetta di soddisfare la lista della spesa. Si noti che è possibile acquistare più oggetti del necessario, purché il costo sia complessivamente minimo. Si ricordi che ogni offerta può essere sfruttata al massimo una volta.

Con i dati degli esempi precedenti, la soluzione ottima include OFF1, OFF3 e OFF4 + 1 Pane e 1 Nutella presi singolarmente, per un totale di 65.

03MNO Algoritmi e Programmazione

Appello del 22/02/2017 - Prova di programmazione (12 punti)

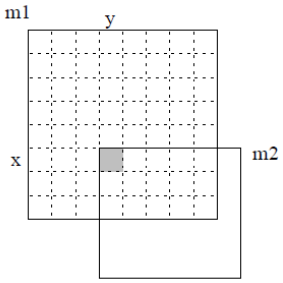
1. (**2 punti**)

Si scriva una funzione caratterizzata dal seguente prototipo

```
void f(int **m1, int r1, int c1, int **m2, int r2, int c2, int x, int y);
```

che riceve due matrici `m1` e `m2` di dimensioni `r1` x `c1` e `r2` x `c2`, rispettivamente, e una coppia di coordinate `x` e `y`.

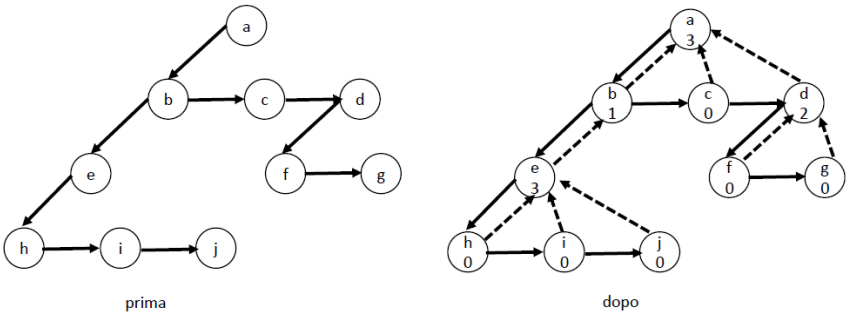
La funzione deve sovrascrivere la porzione di matrice `m1`, a partire dall'angolo in alto a sinistra identificato da `x` e `y`, con i contenuti di `m2`. Poiché non c'è garanzia circa le relative dimensioni delle matrici, la funzione deve effettuare tutti i controlli del caso per non sforare le dimensioni di `m1`, come illustrato nella seguente figura:



2. (**4 punti**)

Dato un albero in formato left-child right-sibling:

- si definisca con una struct il nodo, in cui, oltre alle altre informazioni legate alla rappresentazione left-child right-sibling, vi siano un campo intero per memorizzare il numero di figli e un campo di puntatore al padr
- si scriva una funzione ricorsiva `void processTree(link root);`
- in grado di memorizzare in ogni nodo dell'albero il numero dei suoi figli e il puntatore al padre:



3. **(6 punti**)

Si consideri un insieme di `n` elementi distinti identificati univocamente con un intero da `0` a `n-1`. Per ogni coppia di elementi è noto se questi possono essere separati o no. Tale informazione è riportata in una matrice quadrata di dimensione `n` x `n`. Ogni cella `m[i][j]` della matrice riporta il valore `1` (`0`) se due elementi `i` e `j` sono (non sono) separabili.

Si scriva una funzione dal prototipo:

```
int *f(int n, int k, int **m)
```

il cui obiettivo sia di trovare una partizione dell'insieme di partenza in due sottoinsiemi, il secondo dei quali contenga almeno `k` oggetti, tali per cui nessuna coppia di elementi indivisibili venga separata.

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 sabato 25/02/2017, 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.***

03MNO Algoritmi e Programmazione

Appello del 26/06/2017 - 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:

0-5, 1-3, 0-2, 4-0, 7-6, 3-4, 9-8, 1-3, 6-7, 10-2

si applichi un algoritmo di on-line connectivity con quickfind, 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. (1 punto)

Si ordini in maniera ascendente mediante merge-sort il seguente vettore di interi:

31 19 2 14 43 3 89 23 29 17 51 10 18 16 8 100

Si indichino i passaggi rilevanti.

3. (2 punti)

Sia data una coda a priorità inizialmente vuota implementata mediante uno heap. Sia data la sequenza di interi e carattere *: 20 12 9 17 * * 85 * 71 29 31 * * 14 * 41 27 38 dove ad ogni intero corrisponde un inserimento nella coda a priorità e al carattere * un'estrazione con cancellazione del massimo. Si riporti la configurazione della coda a priorità dopo ogni operazione e la sequenza dei valori restituiti dalle estrazioni con cancellazione del massimo. Al termine si cambi la priorità di 38 in 11 e si disegni la configurazione risultante della coda a priorità.

4. (1.5 punti)

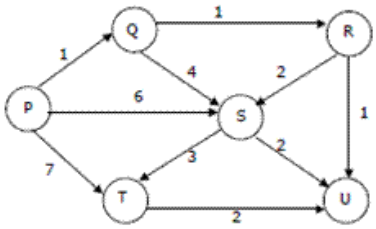
Si esprima in notazione infissa, prefissa e postfissa la seguente espressione aritmetica mediante visita dell'albero binario corrispondente: A * ((B / C) - (D * (E - F)))

5. (2 punti)

Sia data la sequenza di chiavi GFERBAOI, dove ciascun carattere è individuato dal suo ordine progressivo nell'alfabeto (A=1, ..., Z=26). Si riporti la struttura di una tabella di hash di dimensione 17, inizialmente supposta vuota, in cui avvenga l'inserimento della sequenza indicata. Si supponga di utilizzare l'open addressing con quadratic probing. Si selezionino opportuni valori per c_1 e c_2.

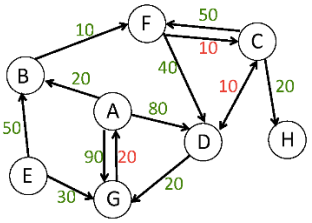
6. (2.5 punti)

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



7. (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 26/06/2017 - Prova di programmazione (18 punti)

Una catena di alberghi gestisce *N* hotel in una città dove per un numero noto *X* di giorni si svolge un importante evento. Ciascun albergo H_i ha un numero di camere M_i. Tramite il suo sito Internet la catena riceve le prenotazioni dei clienti, trascritte su di un file una per riga. Il numero di prenotazioni non è noto a priori. Il formato del file è

```
id_cliente arrivo pernottamenti
```

dove `id_cliente` è una stringa di al massimo 20 caratteri, `arrivo` è il giorno di arrivo (intero tra 0 e *X-1*) e `pernottamenti` è il numero di notti trascorse in quell'albergo (intero tra 1 e *X*). Si assuma che i clienti siano distinti e che i dati del file siano corretti.

La catena vuole allocare i clienti tra i suoi alberghi:

- tenendo conto delle richieste di ciascun cliente
- rispettando per ogni giorno il vincolo di capacità M_i di ciascun albergo
- ottimizzando l'utilizzo delle strutture sulla base delle formule seguenti:

il riempimento medio r_i dell'albergo H_i nel periodo in questione è definito come l'occupazione delle camere su camere totali disponibili per i giorni ($0 \leq r_i \leq 1$):

$$r_i = \frac{1}{M_i \cdot X} \sum_{j=0}^{X-1} camere\_occupate_j$$

siano r_{max} e r_{min} rispettivamente il massimo e il minimo riempimento tra i valori di r_i di tutti gli alberghi. Si massimizzi la funzione:

$$f = a \sum_{i=0}^{N-1} r_i - b \cdot \frac{r_{max} - r_{min}}{r_{max}}$$

dove a e b sono valori interi positivi ricevuti in input.

Si scriva un programma C che, ricevuti sulla riga di comando:

- come primo argomento il nome del file contenente le prenotazioni
- come secondo argomento il nome di un file che contiene una possibile allocazione dei clienti uno per riga nella forma
  `id_cliente albergo`
  dove *albergo* è un intero tra 0 e *N-1*
- come terzo argomento il nome di un file di output

letti da input gli interi *N*, *X*, *a* e *b* e il vettore *M*

- verifichi se l'allocazione proposta nel secondo file rispetti i vincoli di capacità giornaliera (non tenendo conto quindi dei criteri di ottimalità)
- determini, se esiste, un'allocazione dei clienti tra gli alberghi
  - tenendo conto delle richieste di ciascun cliente
  - rispettando per ogni giorno il vincolo di capacità M_i di ciascun albergo
  - massimizzando la funzione *f*

  e la scriva sul file di output nel formato (per ogni riga) `id_cliente albergo`.

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 29/06/2017, 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.***

03MNO Algoritmi e Programmazione

Appello del 26/06/2017 - Prova di programmazione (12 punti)

NON E' AMMESSO L'USO DI FUNZIONI DI LIBRERIA

1. (**2 punti**)

Si scriva una funzione con il seguente prototipo

```
float my_atof(char *s);
```

che riceve una stringa di caratteri che rappresenta un numero reale e ritorna il corrispondente valore `float`. Nella stringa sono ammessi solo i caratteri '+', '-', '. ' e quelli tra '0' e '9' secondo il seguente formato:

- il primo carattere è il segno positivo o negativo
- segue la parte intera del numero come numero imprecisato di caratteri tra '0' e '9' terminato da '. '
- segue infine la parte decimale del numero come numero imprecisato di caratteri tra '0' e '9'.

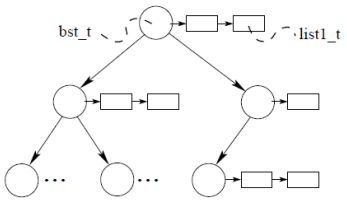
Si assuma che la stringa in ingresso soddisfi sempre il formato sopra descritto. Esempi di stringhe corrette sono: +123.45, −3.5678, −678.2.

2. (**4 punti**)

Un database di modelli di automobile è organizzato come BST. Ogni nodo del BST contiene le seguenti informazioni:

- nome del modello (stringa di caratteri che funge da chiave per il BST)
- puntatore a una lista non ordinata i cui nodi contengono le seguenti informazioni:
  - nome del concessionario (stringa di caratteri)
  - prezzo in Euro (`float`).

I nodi del BST sono di tipo `bst_t` quelli della lista di tipo `list1_t`. La figura seguente rappresenta graficamente un esempio di database:



Si scriva una funzione con il seguente prototipo

```
void bst2list(bst_t *bst, list2_t **head, list2_t **tail);
```

che riceve come parametri un puntatore alla radice del BST e i puntatori alla testa e alla coda di una seconda lista. Si crei una lista ordinata per modelli (ordine alfabetico ascendente), i cui nodi contengono il nome del modello, il nome del concessionario che lo vende a prezzo minimo e il prezzo minimo. L'inserzione in detta lista avvenga in coda. Si riportino esplicitamente le definizioni dei tipi `bst`, `list1` e `list2`.

3. **(6 punti**)

Siano una matrice di interi `mat` , le sue dimensioni `r` e `c` e un intero `k` i parametri della funzione `f` con il seguente prototipo:

```
void f(int **mat, int r, int c, int k);
```

La funzione deve trovare e stampare (se esiste) uno qualsiasi dei sottoinsiemi dei valori della matrice:

- di cardinalità minima
- di cardinalità massima

la somma dei cui elementi sia `k`.

Esempio:

`r = 3, c = 4, k = 10` e `mat`

	0	1	2	3
0	2	3	1	2
1	6	4	2	5
2	2	4	3	2

uno dei sottoinsiemi a cardinalità minima la cui somma vale 10 è (6, 4), uno dei sottoinsiemi a cardinalità massima la cui somma vale 10 è (2, 3, 1, 2, 2).

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.

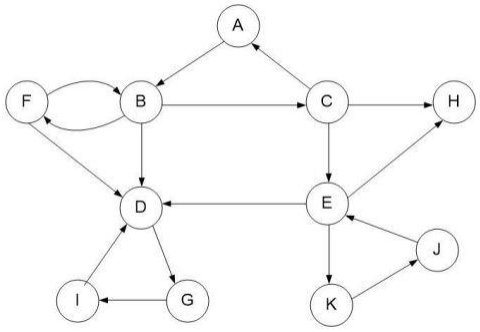
3. (2 punti)

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:

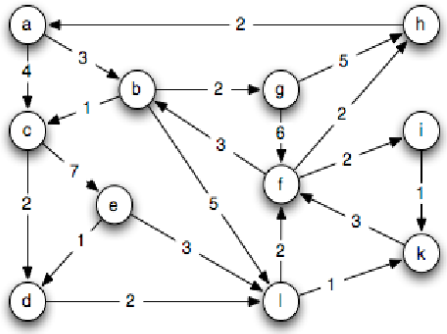


- se ne effettui una visita in profondità, considerando **A** come vertice di partenza (**2 punti**)
- lo si ridisegni, etichettando ogni suo arco come T (tree), B (back), F (forward), C (cross), considerando **A** come vertice di partenza (**1.5 punti**)
- se ne effettui una visita in ampiezza, considerando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

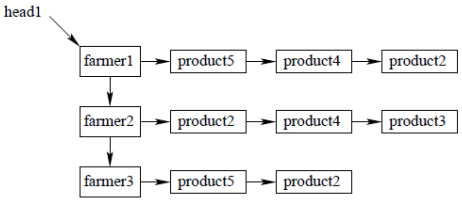
```
int mat_search (char **mat, int r, int c, char *s);
```

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**)

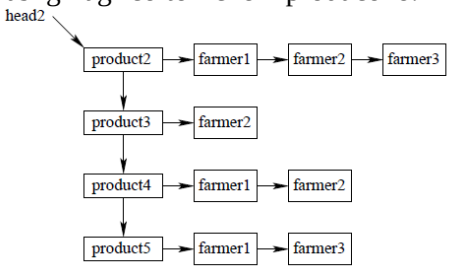
Una database di agricoltori e prodotti è organizzato come lista di liste. La lista principale memorizza in ordine qualsiasi gli agricoltori e per ciascuno di essi in ordine qualsiasi i suoi prodotti. Agricoltori e prodotti sono identificati da stringhe di al massimo 20 caratteri. Alla lista principale 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.

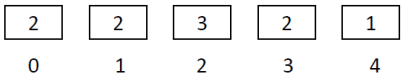
3. **(6 punti**)

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).

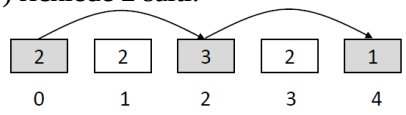
Si scriva una funzione in grado di determinare, se esiste, una sequenza ottima di salti che il giocatore deve fare per spostarsi dalla piattaforma di partenza (indice 0, estremo sinistro) a quella di arrivo (indice P-1, estremo destro). Criterio di ottimalità: minimo numero di salti.

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.***

03MNO Algoritmi e Programmazione

Appello del 29/01/2018 - Prova di teoria (12 punti)

1. **(2.5 punti)**
Si risolva la seguente equazione alle ricorrenze mediante il metodo dello sviluppo (unfolding):

$$T(n) = 3T(n/4) + 1 \qquad n>1$$
$$T(1) = 1 \qquad n=1$$

2. **(2 punti)**
Sia data la sequenza di interi, supposta memorizzata in un vettore:

1 10 72 41 71 0 8 55 91 14 32 19 13 73 7 3 31

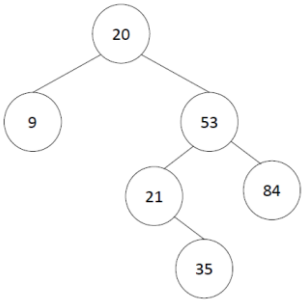
Si eseguano i primi 2 passi dell'algoritmo di quicksort per ottenere un ordinamento **discendente.** NB: I passi sono da intendersi, impropriamente, come in ampiezza sull'albero della ricorsione, non in profondità. Si chiede, pertanto, che siano ritornate le 2 partizioni del vettore originale e le due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

3. **(2.5 punti)**
Data la catena di matrici (A_1, A_2, A_3, A_4) di dimensioni (5x7), (7x6), (6x3) e (3x5) rispettivamente, si determini mediante un algoritmo di programmazione dinamica la parentesizzazione ottima del prodotto di matrici che minimizza il numero di moltiplicazioni.

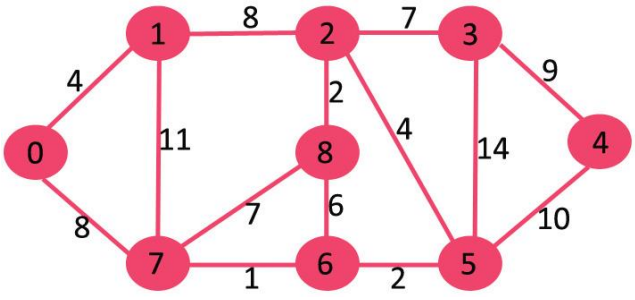
4. **(1 punto)**
Si converta la seguente espressione da forma infissa a forma postfissa:

(A – B) / ((C / D) + (E / (F - G)))

5. **(2 punti)**
Si inseriscano in **radice** nel BST di figura in sequenza le chiavi: 16, 19 e 22 e poi si cancelli la chiave 19. Si disegni l'albero ai passi significativi.



6. **(2 punti)**
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 29/01/2018 - Prova di programmazione (18 punti)

1. (**18 punti**)

Ad un'agenzia per il lavoro si rivolgono *N* persone in cerca di impiego e *N* aziende ciascuna delle quali è alla ricerca di una persona da assumere. L'agenzia organizza gli N^2 colloqui delle *N* persone in cerca di impiego presso le *N* aziende ed al termine chiede ad ogni persona di elencare le aziende in ordine di preferenza decrescente e ad ogni azienda di elencare le persone in ordine di preferenza decrescente. Una volta acquisiti questi dati, l'agenzia cerca di assegnare ogni persona a una e una sola azienda (matching tra persone e aziende). Un matching è un vettore di *N* coppie (persona, azienda).

Dato un matching, una qualsiasi delle *N(N-1)* coppie persona/azienda (*p, a*) **alternative** al matching è una **fonte di instabilità** per quel matching se e solo se sono vere entrambe le seguenti condizioni (AND logico):

- la persona *p* preferisce l'azienda *a* all'azienda a cui è stata assegnata nel matching
- l'azienda *a* preferisce la persona *p* alla persona che le è stata assegnata nel matching.

Un matching è **perfetto** se e solo se nessuna delle *N(N-1)* coppie persona/azienda **alternative** al matching è una **fonte di instabilità.** In altri termini, in un matching perfetto nessuna delle coppie **alternative** al matching è tale per cui, per entrambi i membri, l'assegnazione proposta da questa coppia sarebbe stata migliore di quella assegnata dal matching ad ognuno dei membri.

Esempio: se con *N* =4 le persone p0, p1, p2 e p3 hanno espresso le preferenze per le aziende a0, a1, a2 e a3 riportate nella matrice P e le aziende a0, a1, a2 e a3 hanno espresso le preferenze per le persone p0, p1, p2 e p3 riportate nella matrice A

P

a1	a3	a0	a2
a2	a0	a3	a1
a1	a2	a0	a3
a3	a0	a2	a1

A

p1	p0	p3	p2
p3	p2	p0	p1
p0	p3	p2	p1
p1	p0	p3	p2

Il matching `match` = (p0, a0), (p1, a2), (p2, a1), (p3, a3) non è perfetto in quanto la coppia (p0, a3) è fonte di instabilità: la persona p0 preferisce l'azienda a3 all'azienda a0 cui è stato assegnata && l'azienda a3 preferisce la persona p0 alla persona p3 che le è stata assegnata.
I `match` = (p0, a3), (p1, a2), (p2, a1), (p3, a0) e (p0, a3), (p1, a0), (p2, a1), (p3, a2) sono perfetti perché nessuna coppia tra le *N(N-1)* coppie alternative al matching è fonte di instabilità rispetto al matching.

Si scriva un programma C che, ricevuti sulla riga di comando:

- come primo argomento un intero *N*
- come secondo argomento il nome di un file che contiene *N* gruppi di 2 righe, dove la prima contiene il nome della persona e la seconda N nomi di aziende in ordine di preferenza decrescente
- come terzo argomento il nome di un file che contiene *N* gruppi di 2 righe, dove la prima contiene il nome dell'azienda e la seconda N nomi di persone in ordine di preferenza decrescente
- come quarto argomento il nome di un file che contiene un possibile matching persone/aziende (*N* coppie di nomi persona/azienda)

- acquisisca in opportune strutture dati le preferenze delle persone e delle aziende
- verifichi se il matching sia perfetto o meno. Sarà valutata anche la complessità dell'algoritmo di verifica
- determini e visualizzi un matching perfetto mediante un algoritmo completo.

Le stringhe sono di al massimo 10 caratteri. Si può supporre corretto il file.

03MNO Algoritmi e Programmazione

Appello del 29/01/2018 - Prova di programmazione (12 punti)

1. (**2 punti**)
Sia dato un vettore *V* di *N* interi. Si scriva una funzione C che visualizzi tutti i sottovettori di dimensione massima formati da celle contigue contenenti dati non nulli.

Esempio: dato il vettore 1 3 4 0 1 0 9 4 2 0, i 2 sottovettori di dimensione massima contenenti dati non nulli sono 1 3 4 e 9 4 2.

2. (**4 punti**)
Sia data una lista di interi ordinati in ordine ascendente. Alla lista si accede mediante puntatore alla testa. Si scriva una funzione C che, senza usare array o liste di appoggio, completi la sequenza di interi, inserendo nella posizione corretta all'interno della lista tutti i numeri mancanti e visualizzi la lista così generata. La chiamata alla funzione può essere alternativamente:

```
1. n = aggiungi(&head);
2. n = aggiungi(head);
```

dove `head` rappresenta la testa della lista. Si realizzi una delle 2 alternative motivando la scelta. La funzione ritorna come valore intero il numero di nodi aggiunti.

Esempio: se la lista in input contiene 4, 7, 10, quella in output deve contenere 4,5,6,7,8, 9,10

3. (6 punti)
Un intervallo aperto (l_i, h_i) è caratterizzato da un estremo inferiore l_i, un estremo superiore h_i e una durata $d_i = h_i - l_i$. Estremi e durata sono interi. Una collezione di intervalli *S* è memorizzata in un vettore `v` di *N* `struct interv` aventi come campi estremo inferiore ed estremo superiore. Si suppongano corretti i dati contenuti nel vettore ($\forall\ i\ l_i \leq h_i$). Due intervalli *i* e *j* sono **incompatibili** se e solo se si intersecano o sovrappongono:

$$(l_i < h_j)\ \&\&\ (l_j < h_i) \Leftrightarrow i \cap j \neq \varnothing$$

Si scrivano la funzione wrapper e una funzione ricorsiva in C in grado di determinare e visualizzare il sottoinsieme *S* di intervalli **compatibili** che massimizza la somma delle durate:

$$\sum_{k \in S} d_k = \text{MAX} \quad \&\& \quad \forall (k_1, k_2) \in S\ k_1 \cap k_2 = \varnothing$$

Il prototipo della funzione wrapper sia:

```
void intervSel(int N, interv *v);
```

Si identifichi, giustificando la scelta, il modello combinatorio sottostante. L'algoritmo deve essere completo (non sono ammesse soluzioni greedy).

Esempio: se S =((1,2), (2,4), (2,5), (3,5), (5,7), (6,8)), uno dei sottoinsiemi di *S* di intervalli **compatibili** che massimizza la somma delle durate è (1,2), (2,5), (6,8) per una somma delle durate pari a 6.

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 giovedì 01/02/2018, 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, <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 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

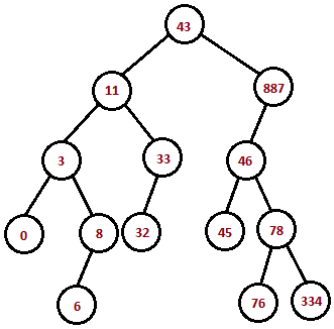
3. (2 punti)

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.

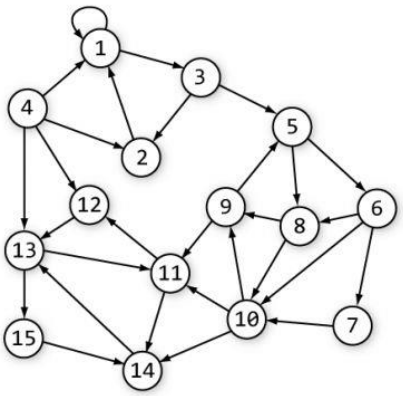
4. (2 punti)

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

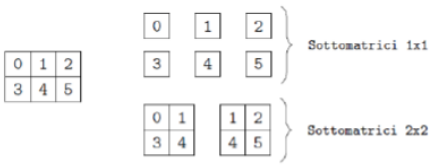
Appello del 13/02/2018 - Prova di programmazione (12 punti)

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):

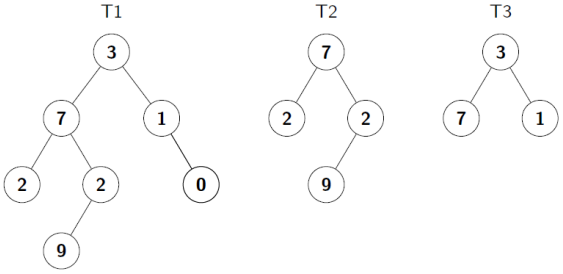


2. (**4 punti**)

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);
```

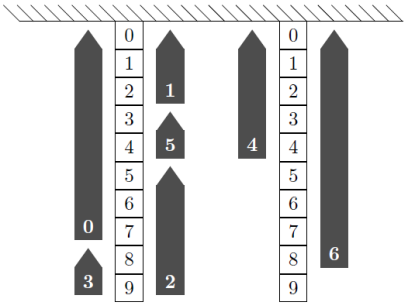
Si supponga la disponibilità di una funzione `KEYcmp(KEY k1, KEY k2)` per comparare le chiavi. Un sottoalbero di un albero T è un albero composto da un nodo che appartiene a T e da **tutti** i suoi discendenti. Esempio: T2 è un sottoalbero di T1, T3 non è un sottoalbero di T1:



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.

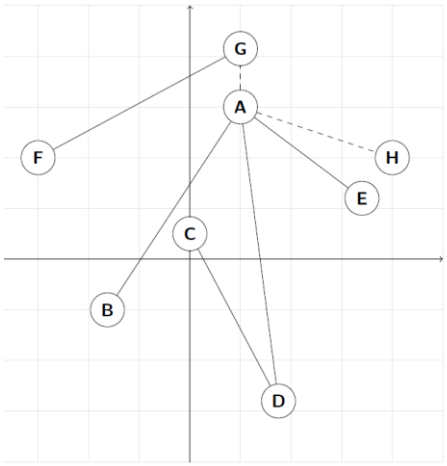
Il grafo in input è **non connesso**, per cui esistono almeno due componenti connesse.

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.

03MNO Algoritmi e Programmazione

Appello del 21/06/2018 - 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: 2-7, 5-3, 1-7, 6-2, 5-9, 5-6, 10-9, 3-5, 6-8, 10-0. 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 10.

2. (1 punto)

Si ordini in maniera ascendente mediante **bottom-up** merge-sort il seguente vettore di interi:

21 19 2 14 43 3 79 23 29 17 51 10 15 16 8 101

Si indichino i passaggi rilevanti.

3. (2 punti)

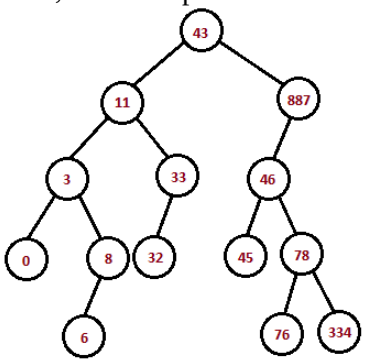
Sia data una coda a priorità implementata mediante uno heap di dati. I dati siano formati da una coppia di interi dove il secondo rappresenta la priorità. Nella radice dello heap si trovi il dato a priorità **minima**. Si inserisca la seguente sequenza di dati nella coda a priorità supposta inizialmente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

(1,20) (4,32) (5,19) (3,51) (7, 28) (8,74) (9,9) (0,81) (10,17) (6,41) (2,37)

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.

4. (1.5 punti)

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

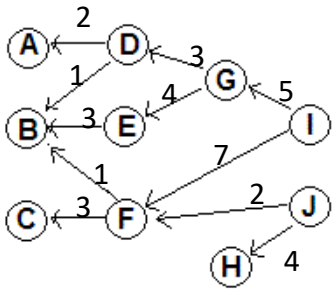


5. (2 punti)

Sia data la sequenza di chiavi intere: 12 145 267 31 98 100 4 207 13 99 181. 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.

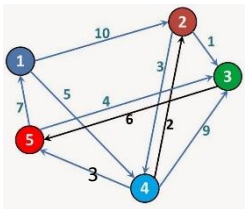
6. (2.5 punti)

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



7. (2 punti)

Sia dato il seguente grafo orientato pesato:



Si determinino i valori di tutti i cammini minimi che collegano il vertice **1** con ogni altro vertice mediante l'algoritmo di Dijkstra. Si assuma, qualora necessario, un ordine numerico per i vertici e gli archi.

03MNO Algoritmi e Programmazione

Appello del 21/06/2018 - Prova di programmazione (12 punti)

1. (**2 punti**)

Sia data una stringa *S* di caratteri. Si scriva una funzione C che conti quante sottostringhe di *n* caratteri in essa contenute hanno 2 vocali.

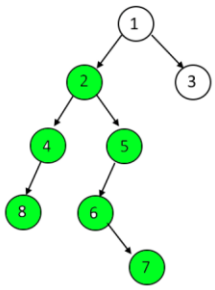
Esempio: se S=”forExample” e n= 4, le sottostringhe di lunghezza 4 con 2 vocali sono 4 e sono “forE”, “orEx”, “rExa” e “Exam”.

2. (**4 punti**)

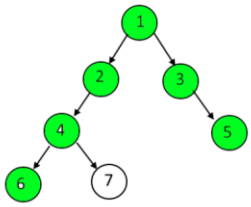
Il diametro di un albero binario è definito come la lunghezza del cammino più lungo tra una qualsiasi coppia di nodi. Si scriva una funzione C che, dato il puntatore alla radice dell’albero, ne determini il diametro:

```
int diameter(link root);
```

Esempi:



diametro = 5



diametro = 5

Si noti che il problema può essere ricondotto alla ricerca di un nodo per il quale sia massima la somma delle altezze del sotto-albero sinistro e di quello destro. È concesso estendere la struct del nodo per includere opportune informazioni aggiuntive utili.

3. **(6 punti**)

Sia data una sequenza di N interi memorizzata in un vettore $X = (x_0, x_1, \ldots x_{N-1})$. Si definisce **sottosequenza** di X di lunghezza k ($k \leq N$) un qualsiasi sottovettore Y degli elementi di X con indici crescenti $i_0, i_1, \cdots, i_{k-1}$ non necessariamente contigui.

Esempio: se X = 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15, Y = 0, 8, 12, 10, 14, 1, 7, 15 è una sottosequenza di lunghezza k=8 di X.

Si scriva in C una funzione che, ricevuti come parametri il vettore X e la sua lunghezza N, calcoli una qualsiasi delle sottosequenze **STRETTAMENTE CRESCENTI** di X a lunghezza massima e restituisca come risultato la sua lunghezza k:

```
int maxSubSeq(int *X, int N, int *Y);
```

Esempio: se X = 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15, le sottosequenze strettamente crescenti Y di X a lunghezza massima (k = 6) sono:

0, 2, 6, 9, 11, 15
0, 4, 6, 9, 11, 15
0, 2, 6, 9, 13, 15
0, 4, 6, 9, 13, 15

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 24/06/2018, 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 21/06/2018 - Prova di programmazione (18 punti)

1. (**18 punti**)

Un gioco si svolge su una scacchiera rettangolare di `RxC` caselle. In ogni casella deve essere posizionata una tessera su cui sono disegnati due segmenti di tubo, uno in orizzontale e uno in verticale. Le diagonali non sono contemplate. Ogni segmento è caratterizzato da un colore e da un punteggio (valore intero positivo).

Per ottenere i punti associati ai vari segmenti, è necessario allineare lungo un'intera riga (colonna) tubi in orizzontale (verticale) dello stesso colore. Le tessere possono essere ruotate di 90°. Le tessere sono disponibili in copia singola. Si assuma esistano abbastanza tessere per completare la scacchiera.

Lo scopo del gioco è ottenere il massimo punteggio possibile posizionando una tessera in ogni cella della scacchiera.

Su un primo file di testo (`tiles.txt`) è riportato l'elenco delle tessere disponibili nel seguente formato:

- sulla prima riga è presente il numero `T` di tessere
- seguono `T` quadruple nella forma `<coloreT1><valoreT1> <coloreT2> <valoreT2>` a descrivere la coppia di tubi presenti sulla tessera in termini di rispettivo colore e valore.

Si assuma che a ogni tessera sia associato un identificativo numerico nell'intervallo 0..`T`-1

Su un secondo file di testo (`board.txt`) è riportata una configurazione iniziale per la scacchiera di gioco. Il file ha il seguente formato:

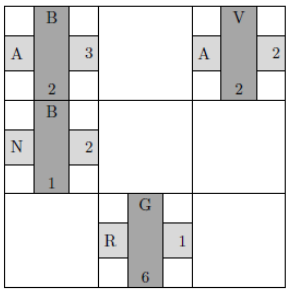
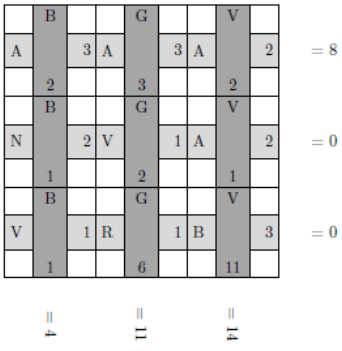
- sulla prima riga è presente una coppia di interi `R C` a rappresentare il numero di righe e colonne della superficie di gioco
- seguono `R` righe riportanti `C` elementi ciascuna a definire la configurazione di ogni cella
- ogni cella è descritta da una coppia t_i`/r` dove t_i è l'indice di una tessera tra quelle presenti in `tiles.txt` e `r` rappresenta l'eventuale sua rotazione (es: 7/0 oppure 3/1 per rappresentare rispettivamente una tessera non ruotata e una ruotata). Una cella vuota è rappresentata come -1/-1.

Si scriva un programma in C che, una volta acquisiti in opportune strutture dati l'elenco delle tessere disponibili e la configurazione iniziale della scacchiera:

- legga da file due possibili completamenti per la scacchiera letta in precedenza e valuti quale dei due porti a un punteggio maggiore. I file siano caratterizzati da un numero (non precisato, ma compatibile con le celle libere della scacchiera) di righe nella forma `<`t_i`>/<rotazione> <posR> <posC>` a indicare quale tessera e con quale orientamento vada posizionata in una data cella [`posR`][`posC`] della scacchiera.
- generi la soluzione a punteggio massimo possibili a partire dalla configurazione iniziale letta da file.

Esempio:

`tiles.txt`	`board.txt`	scacchiera iniziale	scacchiera finale

`9`
`A 3 B 2`
`A 2 V 1`
`A 2 V 2`
`B 1 N 2`
`A 3 G 3`
`V 1 G 2`
`R 1 G 6`
`V 1 B 1`
`V 11 B 3`
	`3 3`
	`0/0 -1/-1 2/0`
	`3/1 -1/-1 -1/-1`
	`-1/-1 6/0 -1/-1`
			

03MNO Algoritmi e Programmazione

Appello del 13/09/2018 - 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:

2-7, 5-3, 1-7, 6-2, 5-9, 5-6, 10-9, 3-5, 6-8, 10-0

si applichi un algoritmo di on-line connectivity con quickfind, 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:

21 19 2 14 43 3 79 23 29 17 51 10 15 16 8 101

- 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.

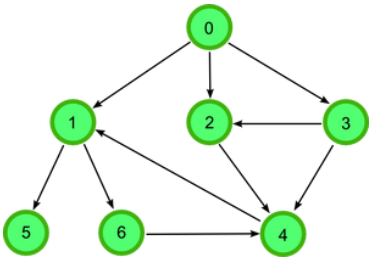
3. (2 punti)

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

[2,12] [3, 6] [0,5] [15,21] [8,13] [1,8] [11,23] [4,15] [7,12]

4. (2 + 1.5 + 1.5 punti)

Sia dato il seguente grafo orientato:

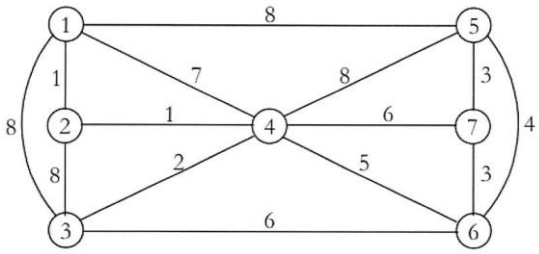


- se ne effettui una visita in profondità, considerando **0** come vertice di partenza (**2 punti**)
- lo si ridisegni, etichettando ogni suo arco come T (tree), B (back), F (forward), C (cross), considerando **0** come vertice di partenza (**1.5 punti**)
- se ne effettui una visita in ampiezza, considerando **0** come vertice di partenza (**1.5 punti**)

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

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 **1**, disegnando l'albero e ritornando come risultato il valore del peso minimo. Si esplicitino i passi intermedi.

03MNO Algoritmi e Programmazione

Appello del 13/09/2018 - Prova di programmazione (12 punti)

1. (**2 punti**)

In un campionato *n* squadre giocano per *m* giornate. Sia data una matrice *C* di $n \times m$ numeri interi, ognuno dei quali può valere soltanto 0, 1 o 3. Ogni riga della matrice rappresenta i punti acquisiti dalle *n* squadre nelle partite disputate nelle *m* giornate del campionato: 3 punti per le partite vinte, 1 punto per quelle pareggiate e 0 punti per le sconfitte. I risultati della giornata *k*-esima sono contenuti nelle righe della colonna di indice *k*. Si scriva una funzione C con il seguente prototipo

```
void displRanking(int **C, int n, int m);
```

che, per ogni giornata del campionato, stampi l'indice (il numero di riga corrispondente) della squadra capolista in quella giornata.

Esempio:

Con la seguente matrice di 4 righe e 3 colonne, l'output sarà:

3	1	0
0	1	1
1	1	1
1	1	3

0	0	3

2. (**4 punti**)

Sia data una lista di interi, cui si accede mediante puntatore alla testa `link1 head`. Si scriva una funzione C che costruisca una nuova lista "compressa": per ogni elemento della prima lista si memorizza nella seconda lista l'elemento stesso e il numero di ripetizioni consecutive nella prima. Si definisca opportunamente il nodo della seconda lista. Il prototipo sia:

```
link2 comprimi(link1 head);
```

Esempio: se la prima lista è (3, 3, 3, 3, 2, 2, 3, 5, 5, 5), la seconda lista sarà ((3, 4), (2, 2), (3, 1), (5, 3)).

3. (6 punti)

Un vettore *paste* di *n struct* descrive le tipologie e le disponibilità dei pasticcini realizzati da un laboratorio di pasticceria. Ogni *struct* consta di 3 campi interi non negativi:

- `codice` identifica la tipologia di pasticcino (ad esempio 3 bigné, 5 crostatina, etc.)
- `peso` indica il peso in grammi di ogni pasticcino
- `quantita` indica il numero di pasticcini di quella tipologia disponibili.

Si scriva un programma C che, ricevuti come parametro il vettore, la sua lunghezza e un intero che rappresenta il peso del vassoio di pasticcini da comporre, calcoli l'insieme di pasticcini il cui peso si avvicini il più possibile (al più eguagli) al peso richiesto, tenendo conto delle disponibilità di ogni tipologia. Si proceda quindi a stampare tale insieme, specificando per ogni tipologia di pasticcini la sua cardinalità, ed il peso finale del vassoio. Nel caso di più soluzioni equivalenti è sufficiente stamparne una. Si supponga inoltre che l'utente non abbia alcuna preferenza sulla tipologia di pasticcini selezionati.

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 16/09/2018, 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 13/09/2018 - Prova di programmazione (18 punti)

1. (**18 punti**)

I titoli azionari delle aziende quotate in Borsa possono essere oggetto di transazioni (vendita/acquisto di un certo numero di titoli in una certa data/ora ad un certo valore su una certa piazza) in una o più piazze finanziarie in tutto il mondo. Si immagini di voler creare un sistema globale, cioè che tenga conto di tutte le Borse, per la memorizzazione e gestione dei titoli azionari.

Ogni Borsa invia al sistema con regolarità un file con un elenco di transazioni raggruppate per titolo. Sulla prima riga compare il numero di titoli, di seguito i titoli e le relative transazioni con il seguente formato:

- una prima riga contiene il `<titolo>` (codice alfanumerico univoco di al più 20 caratteri) seguito dal numero di transazioni
- sulle righe successive, una per riga, le transazioni sotto forma di quaterne `<data> <ora> <valore> <numero>`. Le date sono nella forma `aaaa/mm/gg`, le ore sono nel formato su 24 ore `hh:mm` riferite al tempo di Greenwich (GMT), mentre i valori dei titoli sono rappresentati con numeri reali non negativi, la quantità è un intero. Non si presupponga nessuna forma di ordinamento né sui nomi dei titoli, né sulle transazioni.

Il sistema acquisisce i file e ne memorizza i contenuti in un'opportuna struttura dati a 2 livelli che prevede una collezione di titoli e per ogni titolo una collezione delle sue quotazioni giornaliere. La quotazione giornaliera $\mathbf{Q_i}$ di un titolo in una certa data **i** è la media di tutti i valori $\mathbf{v_{ij}}$ di quel titolo in quella data **i** pesati sul numero di titoli scambiati $\mathbf{n_{ij}}$

$$\boldsymbol{Q_i = \frac{\sum_j v_{ij} \cdot n_{ij}}{\sum_j n_{ij}}}$$

Si realizzino:

- un quasi ADT per la data, implementato come struct con campi interi per anno, mese e giorno
- un ADT di I classe per titolo e collezione di titoli
- un ADT di I classe per quotazioni giornaliere e collezione di quotazioni giornaliere
- un client che fornisca le seguenti funzionalità (l'eventuale menu e il tipo di I/O sono a scelta dello studente):
  1. acquisizione in qualsiasi momento del contenuto di un file contenente un insieme di transazioni (*si noti che la possibilità di acquisire un nuovo file in qualsiasi momento richiede una struttura dati completamente dinamica*)
  2. ricerca di un titolo azionario, con complessità al più logaritmica nel numero di titoli
  3. ricerca, dato un titolo precedentemente selezionato, della sua quotazione in una certa data, con complessità al più logaritmica nel numero di quotazioni per quel titolo
  4. ricerca, dato un titolo precedentemente selezionato, della sua quotazione minima e massima in un certo intervallo di date.
  5. ricerca, dato un titolo precedentemente selezionato, della quotazione minima e massima lungo tutto il periodo registrato.

La complessità dell'operazione nei punti 4 e 5 non è vincolata, ma sarà usata come metro di giudizio. Si richiede di indicare esplicitamente la complessità della soluzione sviluppata. Non è richiesta un'implementazione completa degli ADT. Per le collezioni si realizzino, oltre alla definizione della struttura dati, le funzioni di inserimento e ricerca richieste dai punti precedenti. Per l'ADT data si realizzino le funzioni di lettura, scrittura e confronto. Nel caso si utilizzino BST non si ponga il problema di bilanciarli.

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

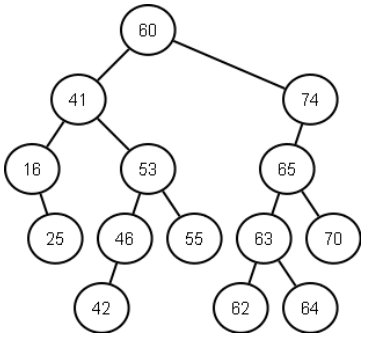
si eseguano i primi 2 passi dell'algoritmo di quicksort per ottenere un ordinamento ascendente, indicando ogni volta il pivot scelto. NB: i passi sono da intendersi, impropriamente, come in ampiezza sull'albero della ricorsione, non in profondità. Si chiede, pertanto, che siano ritornate le 2 partizioni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

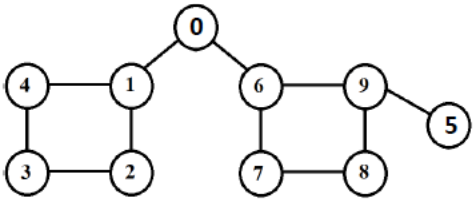
4. (2 punti)

Si partizioni il seguente BST attorno alla chiave 55:



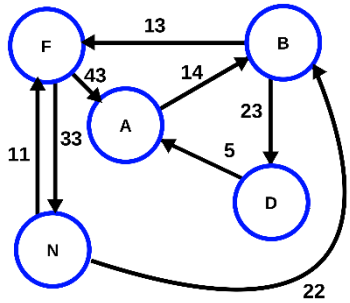
5. (1 punto)

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)

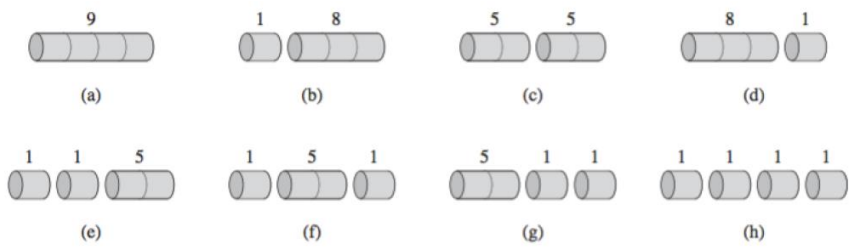
Un nastro lungo `n` (intero) può essere tagliato in pezzi di `m` (intero) diverse lunghezze intere `lun` dove 1 ≤`lun`≤`n`. Ciascuno dei pezzi viene venduto ad un determinato prezzo. Si possono tagliare più pezzi della stessa lunghezza. Un vettore `prezzo` di `m` interi contiene i prezzi per ciascuna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

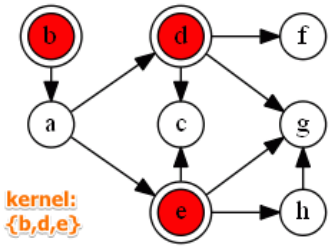
Nella Teoria dei Grafi, dato un grafo orientato G = (V, E), si definisce **Kernel** o **Nucleo** un insieme K $\subseteq$V di vertici di G tale per cui valgono entrambe le seguenti condizioni:

- 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)

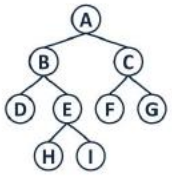
Si riporti la configurazione della coda a priorità dopo ogni inserzione. Al termine si cambi la priorità del dato in posizione 4 in 11 e si disegni la configurazione risultante della coda a priorità.

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:

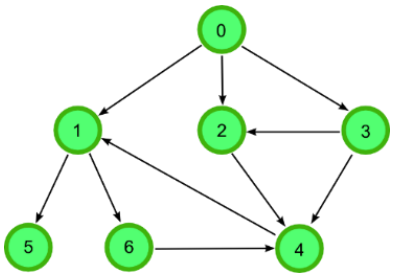


4. (2 punti)

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)

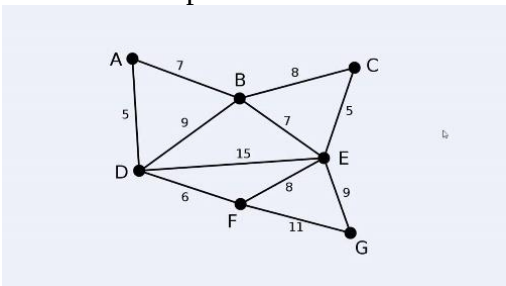
Sia dato il seguente grafo orientato:



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.

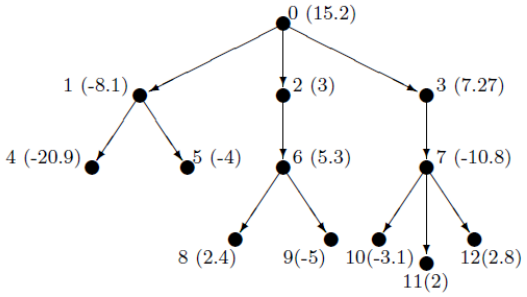
Esempio: se V= -1, 2, 3, -6, 1, 3, 1 i 2 sottovettori a somma massima 5 sono 2,3 e 1,3,1 entrambi ammissibili come soluzione.

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

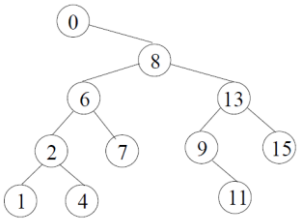
Si disegnino lo heap e il vettore `qp` ai diversi passi dell'inserzione. Al termine si cambi la priorità di `"rt"`da 28 a 60 e si disegnino i corrispondenti heap e vettore `qp`.

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.

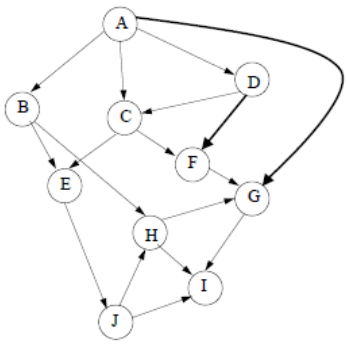
4. (2 punti)

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:



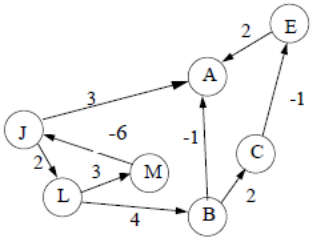
5. (2 punti)

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). 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 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

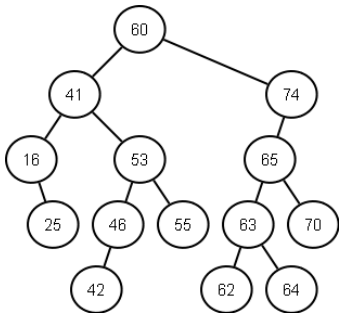
Si eseguano i primi 2 passi dell'algoritmo di quicksort per ottenere un ordinamento **ascendente.** NB: I passi sono da intendersi, impropriamente, come in ampiezza sull'albero della ricorsione, non in profondità. Si chiede, pertanto, che siano ritornate le due partizioni del vettore originale e le due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

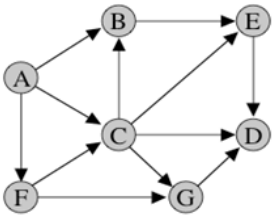
3. (2 punti)

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.



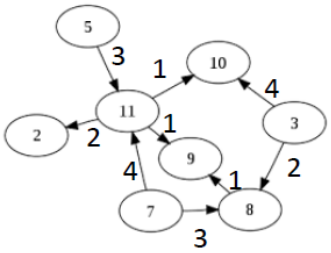
4. (1.5 punti)

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.



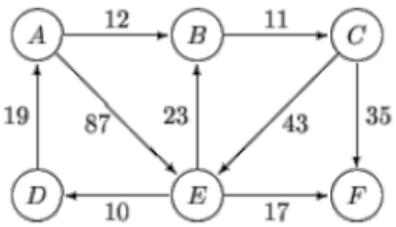
5. (2.5 punti)

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 <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 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, <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 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`

```
6 5
1 1
1 4
3 2
4 3
```

`proposta.txt`

```
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
```

Mappa corrispondente a `proposta.txt`

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.