Strutture Dati Algoritmi e Complessità **Name:** _____
A.A. 2023/2024
Compito B
12/07/2024
Time Limit: 180 minuti (3 ore) [Studenti DSA: no quesiti 2, 4 e 6]

Questo compito contiene 7 pagine (inclusa questa) e 7 quesiti.
Il numero totale di punti è **32**.

Tabella dei punteggi (per i docenti)

Question	Points	Score
1	10	
2	4	
3	4	
4	4	
5	3	
6	4	
7	3	
Total:	32	

Vi sono 7 quesiti diversi, i cui punteggi ne identificano l'importanza relativa.

Si prega di scrivere in modo chiaro. Completezza (non lungaggine) e rigore nell'esposizione delle vostre argomentazioni sono fattori decisivi ai fini della valutazione. Non occorrse scrivere tanto, ma scrivere bene il necessario.

Istruzioni

Lanciare la macchina virtuale Oracle VirtualBox e lavorare all'interno della cartella `Esame`. Per prima cosa compilare il file `studente.txt` con le informazioni richieste. In particolare, cognome, nome, matricola, linguaggio in cui si svolge l'esercizio 1 (Java o C).

Nota bene. Per ritirarsi è sufficiente rinominare il file `studente.txt` in `studenteritirato.txt`, **senza** cancellarlo.

Come procedere. Nella cartella `Esame` trovate i) il testo del compito, ii) il file `studente.txt` sopra citato, iii) due sottocartelle `C-aux` e `java-aux`, contenenti il codice di supporto e lo scheletro delle soluzioni per il quesito di programmazione (quesito 1), iv) infine tre file testo in cui scrivere le vostre risposte ai quesiti di teoria. Svolgere il compito nel modo seguente:

- Per il quesito 1, Alla fine la cartella `C-aux` (o java-aux, a seconda del linguaggio usato) dovrà contenere le implementazioni delle soluzioni al quesito 1, ottenute completando i relativi metodi (o le relative funzioni) contenuti nei file forniti dai docenti; si ricorda ancora una volta che la cartella `java-aux` (o `C-aux`) deve trovarsi all'interno della cartella `Esame`.
- Usare invece i file di testo `teoria_2-3.txt`, `teoria_4-5.txt` e `teoria_6-7.txt` forniti con la distribuzione per le risposte ai quesiti di teoria che corrispondono alla numerazione di ogni singolo file. Si prega di indicare chiaramente l'inizio dello svoglimento di ogni quesito a cui si risponde.

Nota bene: È possibile integrare i contenuti di tali file con eventuale materiale cartaceo *unicamente per disegni, grafici, tabelle, calcoli.* Come spiegato sopra, i file da completare sono già forniti con la distribuzione, quindi in generale non è necessario creare nuovi file.

Avviso importante 1. Per svolgere il quesito di programmazione *si consiglia caldamente l'uso di un editor di testo (ad esempio Geany) e la compilazione a riga di comando*, strumenti più che sufficienti per lo svolgimento del compito. La macchina virtuale mette a disposizione diversi ambienti di sviluppo, quali Eclipse e Javabeans. *Gli studenti che li usano lo fanno a proprio rischio.* In particolare, *se ne sconsiglia l'uso qualora non se ne abbia il pieno controllo e certamente se non si è già in grado di sviluppare servendosi unicamente di un editor e della compilazione a riga di comando.* Qualora lo studente intenda comunque usare un ambiente di sviluppo integrato, si raccomanda di controllare che i file vengano effettivamente salvati nella cartella `java-aux` (o `C-aux`, a seconda del linguaggio usato), in quanto vi è il rischio concreto di perdere il proprio lavoro. In generale, file salvati esternamente alla cartella `Esame` andranno persi al termine della prova e quindi non saranno corretti.

Avviso importante 2. Il codice *deve compilare* correttamente (warning possono andare, ma niente errori di compilazione), altrimenti riceverà **0** punti. Si raccomanda quindi di scrivere il programma in modo incrementale, ricompilando il tutto di volta in volta.

Quesito di programmazione

1. [10 points] **Alberi minimi ricoprenti.** In questo problema si fa riferimento a grafi semplici non diretti, con pesi sugli archi, rappresentati con liste di incidenza. In particolare, il grafo è una lista di coppie (nodo, lista di incidenza). La lista di incidenza di un nodo contiene tutti gli archi incidenti nel nodo, nonché i loro pesi. Più in dettaglio, ogni arco di una lista di incidenza è una terna `(source, target, weight)`, con `source` e `target` rispettivamente i nodi origine e destinazione dell'arco e `weight` il suo peso (positivo). La rappresentazione degli archi è data dalla classe `Edge` (Java) o dalla `struct graph_edge` in `graph.h` . Sono inoltre già disponibili le primitive di manipolazione del grafo (get lista nodi, get lista archi aventi origine da un nodo dato ecc.. Per dettagli sulle segnature di tali primitive, così come per dettagli su quali porzioni di memoria allocata vengono liberate dalle varie primitive di cancellazione, si rimanda ai file sorgente distribuiti e ai commenti in essi contenuti. Si noti che le primitive forniscono un insieme base per la manipolazione di grafi, ma i problemi proposti possono essere risolti usandone solo un sottoinsieme.

   Viene fornita l'implementazione delle primitive Union - Find per operare su insiemi disgiunti (classe `Partition` in Java e modulo `partition.c` in C). Si noti che tale implementazione gestisce insiemi di numeri. E' possibile mappare ogni nodo del grafo ad un numero facendo uso del campo int map presente in ogni nodo del grafo.

   Infine, le classi `MinHeap` e `HeapEntry` (Java) o il modulo `min_heap.c` (in C) implementano un heap minimale che gestisce coppie con chiavi intere e valori di tipo generico.

   Tutto ciò premesso, risolvere al calcolatore quanto segue, in Java o C, con l'avvertenza che gli esempi usati nei programmi di prova `Driver.java` e `driver.c` fanno riferimento alla figura seguente:

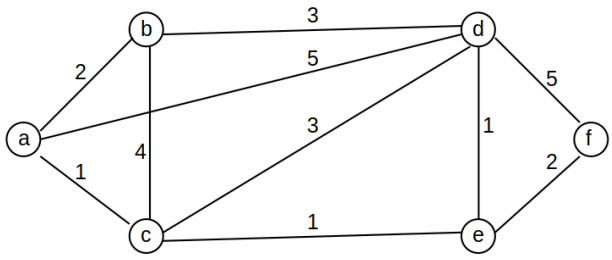

Figure 1: Esempio di grafo non diretto connesso e pesato. In questo caso, l'unico albero minimo ricoprente è quello costituito dall'insieme di archi {`(a, b)`, `(a, c)`, `(c, e)`, `(e, d)`, `(e, f)`}. L'ordine tra gli archi o tra i nodi estremi dello stesso arco non è ovviamente importante.

   Implementare la funzione/metodo `public static <V> void mst(Graph<V> G)` della classe `GraphServices` (o la funzione `void mst(graph* g)` del modulo `graph_services` in C) che, dato un grafo non diretto connesso e pesato `g`, stampa a video un sottoinsieme degli archi del grafo che compongono un *albero minimo ricoprente*, usando l'algoritmo di Kruskal. Si faccia riferimento alla Figura 1 per un esempio.

Suggerimenti. Come visto nelle esercitazioni, si consiglia di usare il campo `int map` della classe `Node` (o di `struct graph_node` in C) per assegnare a ogni vertice un intero da usare con le primitive della classe `Partition` (o del modulo `partition.c` in C).

Algoritmi e Strutture Dati - nozioni di base

2. [**No studenti con DSA**] **Hashing.** Si consideri una tabella hash di dimensione $m = 10 + x$, dove x è l'ultima cifra del vostro numero di matricola. Ad esempio, se il vostro numero di matricola fosse 1234567 allora $m = 17$. Usare un valore diverso per m comporta una penalizzazione.

   Le collisioni sono risolte usando l'indirizzamento aperto, in particolare l'*ispezione lineare*, e la prima posizione in cui si cerca di inserire la generica chiave k è $h(k) = k \mod m$.

   (a) [4 points] Descrivere la sequenza degli stati della tabella (ossia le chiavi contenute nella tabella e le loro posizioni) durante l'inserimento della seguente sequenza di chiavi: $12, 22, 31, 4, 15, 28, 17, 88, 59$. Occorre segnalare gli inserimenti che danno luogo a collisione, specificando le posizioni coinvolte.

   Il punteggio dipenderà dalla completezza e dalla chiarezza espositiva. Non occorre scrivere tanto ma scrivere bene.

3. **Cammini minimi.** Si consideri il grafo non diretto e pesato in Figura 2

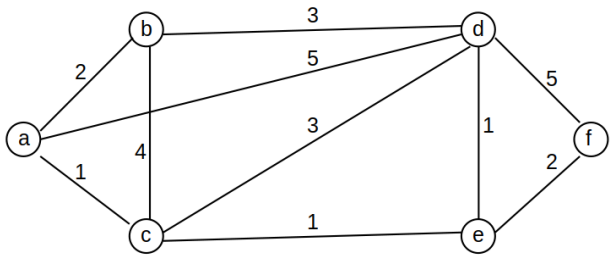

Figure 2: Grafo non diretto pesato.

   (a) [4 points] Si mostri l'evoluzione dell'algoritmo di Dijkstra per il grafo non orientato e pesato in Figura 2 (lo stesso della Figura 1, riportato qui per comodità) a partire dal nodo a. In particolare, per ogni iterazione i, occorre indicare i) la distanza stimata di ogni nodo da a e il nodo la cui distanza da a diviene quella definitiva nell'iterazione i. Per descrivere l'evoluzione dell'algoritmo usare una tabella con il formato di quella sottostante (o equivalente):

Iterazione	a	b	c	d	e	f
1	**0**	∞	∞	∞	∞	∞
2	**0**	...	...	...	...	...
⋮	...	...	...	...	...	...

   A titolo esemplificativo la distanza stimata di a (da se stesso) diviene definitiva nella prima iterazione, e ciò è indicato scrivendola in grassetto in quella iterazione e nelle successive (gli studenti possono semplicemente cerchiarla).

Nota bene: descrizioni incomplete o confuse saranno penalizzate.

Algoritmi e Strutture Dati - proprietà di algoritmi e problemi

4. **[No studenti con DSA] Visita in profondità (DFS).** Si consideri un grafo $G = (V, E)$ orientato. Si consideri il *sottografo dei predecessori* (o *foresta di visità in profondità*) risultante da una visita in profondità di G. Come visto a lezione, per ogni vertice $v \in V$ indichiamo con $v.d$ l'istante in cui v è scoperto e con $v.f$ quello in cui la DFS completa l'ispezione della lista di adiacenza di v e quindi la visita di v. Rispondere alle domande seguenti:

   (a) [2 points] 1) Si definisca formalmente il sottografo dei predecessori di una visita in profondità di G; 2) considerati due vertici u e v appartenenti a V, si spieghi chiaramente quando diciamo che v è un *discendente* di u nel sottografo dei predecessori.

   (b) [2 points] Dimostrare che se $u.d < v.d < v.f < u.f$ allora v è un discendente di u.

   Il punteggio dipenderà molto dal rigore, dalla chiarezza espositiva e dalla solidità delle argomentazioni proposte. Non occorre scrivere tanto ma scrivere bene.

5. **Correttezza di algoritmi.** Si consideri il problema di calcolare il valore di un polinomio di grado n nella variabile x:

$$P(x) = \sum_{i=0}^{n} a_k x^k.$$

   Si supponga che l'array $\mathbf{A}[0 \ldots n]$ contenga gli $n + 1$ coefficienti di $P(x)$. Il seguente algoritmo, noto come regola di Horner, calcola $P(x)$:

```
HORNER(A, x)
Input: Array A[0 ... n], x

p = 0
for i = n downto 0 do
|   p = A[i] + x · p
end
return p
```

Algorithm 1: Regola di Horner.

   (a) [3 points] *Dimostrare* che HORNER($\mathbf{A}$, x) è corretto, ossia che l'algoritmo calcola correttamente il valore di $P(x)$.

   Suggerimento: Si osservi che per $i = n$ abbiamo $p = a_n$ e per $i = n - 1$ abbiamo $p = a_n x + a_{n-1}$. Si dimostri quanto richiesto generalizzando questi due esempi, ricavandone un invariante di ciclo.

   Il punteggio dipenderà molto dal rigore, dalla chiarezza espositiva e dalla solidità delle argomentazioni proposte. Non occorre scrivere tanto ma scrivere bene.

Problem Solving e Tecniche Algoritmiche

6. **[No studenti con DSA] Problem solving.** Si consideri un array $\mathbf{A}[1 \dots n]$ contenente n valori (non necessariamente interi), *distinti e ordinati in senso crescente*.

   (a) [2 points] Progettare e scrivere lo *pseudo-codice* di un algoritmo *efficiente* MinLarger($\mathbf{A}$, x) che, dato un array definito come sopra e un numero reale arbitrario x, restituisce l'indice del *più piccolo* elemento di $\mathbf{A}$ che è strettamente maggiore di x. Qualora x fosse maggiore o uguale del massimo di $\mathbf{A}$ l'algoritmo deve restituire il valore $n + 1$. *Mostrare in modo convincente perché l'algoritmo è corretto.*

   Ad esempio, se $\mathbf{A} = [2.5, 4, 7.2]$ e $x = 3$ l'algoritmo deve restituire il valore 2, mentre se $x = 9.1$ l'algoritmo deve restituire il valore 4 (si noti che gli indici di $\mathbf{A}$ partono da 1).

   (b) [2 points] Calcolare la complessità computazionale dell'algoritmo proposto rispetto a n.

   Il punteggio dipenderà molto dal rigore, dalla chiarezza espositiva e dalla solidità delle argomentazioni proposte. Non occorre scrivere tanto ma scrivere bene.

7. **Programmazione dinamica** È dato un array X di interi positivi di dimensione M; gli interi $X[i]$ sono tutti diversi tra loro. Vogliamo la lunghezza della sottosequenza *decrescente* più lunga possibile di X, i cui elementi non sono necessariamente adiacenti (ma appaiono in ordine decrescente). Ad esempio

   Input: $\mathbf{A} = \{30, 10, 2, 20, 1\}$; Output: 4 (la sottosequenza decrescente più lunga è $\{30, 10, 2, 1\}$)

   Input: $\mathbf{A} = \{50, 3, 10, 7, 40, 80\}$; Output: 3 (la sottosequenza decrescente più lunga è $\{50, 10, 7\}$)

   Input: $\mathbf{A} = \{25, 1, 20, 4, 5\}$; Output: 3 (sono presenti due sottosequenze decrescenti di lunghezza 3)

   (a) [2 points] Scrivere un algoritmo che risolve il problema della sottosequenza decrescente più lunga.

   (b) [1 point] Calcolare la complessità dell'algoritmo proposto.

   Suggerimento: Si suggerisce di attraversare l'array di input da sinistra a destra e, per ogni indice i, trovare la lunghezza della sottosequenza decrescente più lunga di elementi (anche non adiacenti) che termina esattamente con $X[i]$ e include un sottoinsieme degli elementi in $\{X[1], \dots, X[i-1]\}$. Sia $D[i]$ la lunghezza trovata per $D[i]$. Per calcolare $D[i]$ si inizia da $i = 1$ e si procede; $D[i]$ può essere calcolata utilizzando $X[i]$ e il valore $D[j]$, $j < i$, per tutti i valori di j per cui $X[j] > X[i]$. Alla fine, l'output richiesto è il massimo di tutti i valori $D[i]$.