Strutture Dati Algoritmi e Complessità **Name:** _____
A.A. 2024/2025
Compito A
27/05/2025
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. **Distanze minime** In questo problema si fa riferimento a grafi semplici *diretti*, con pesi positivi 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 uscenti dal 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. Per dettagli sulle segnature di tali primitive si rimanda ai file sorgente distribuiti. 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.

   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.

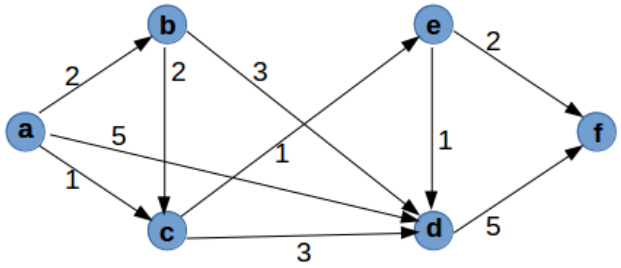

Figure 1: Esempio di grafo diretto pesato. Le distanze minime dal nodo a sono [a:0, b:2, d:3, c:1, e:2, f:4], mentre le distanze minime dal nodo c sono [a:100000, b:100000, d:2, c:0, e:1, f:3], assumendo di rappresentare la distanza infinita attraverso un valore maggiore della somma dei pesi di tutti archi (in questo caso si è scelto il valore 100000).

   (a) [10 points] Implementare la funzione/metodo `public static <V> void distances(Graph<V> g)` della classe `GraphServices` (o `void distances(graph* g)` del modulo `graph_services.c` in C) che, dato un grafo `g` stampa, *per ogni* nodo `source`, le distanze minime da `source` a tutti gli altri nodi raggiungibili nel grafo. Per i nodi non raggiungibili, la distanza andrà impostata a un valore di riferimento (lo studente può ad esempio usare il valore 100000). Ad esempio, per il nodo `a` si dovrebbe stampare la riga (l'ordine di stampa non è importante) `[a:0, b:2, d:3, c:1, e:2, f:4]`.[1]

   Suggerimenti. i) si consiglia di definire una funzione/metodo privato che calcola le distanze minime dal generico nodo del grafo; ii) In Java: si consiglia di usare una `HashMap` per associare i nodi del grafo a corrispondenti oggetti `HeapEntry` se necessario.

[1] Per il formato di stampa, si faccia riferimento alla stampa della risposta corretta del driver.

Algoritmi e Strutture Dati - casi semplici

2. **Alberi Binari di Ricerca** Si consideri l'albero binario di ricerca (BST) a chiavi intere rappresentato nella Figura 2.

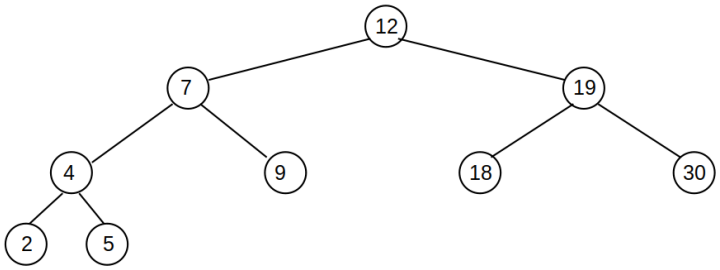

Figure 2: Un albero binario di ricerca.

(a) [4 points] Si descriva l'evoluzione del BST a seguito dell'inserimento di una nuova coppia con chiave **15**. In particolare, occorre i) descrivere ogni passo di aggiornamento del BST a seguito dell'inserimento della chiave e del mantenimento della condizione di BST, ii) descrivere l'albero finale risultante.

3. **Alberi Minimi Ricoprenti** Si consideri il grafo non diretto e pesato in Figura 3

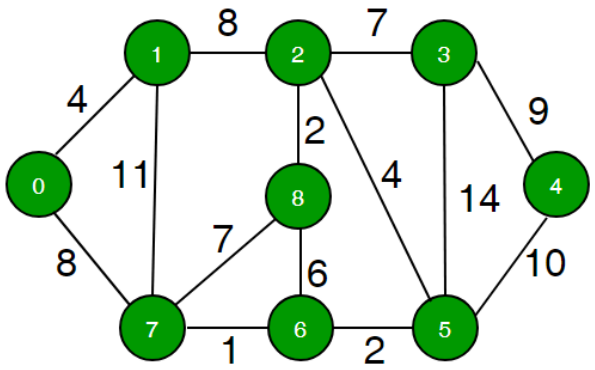

Figure 3: Grafo non diretto pesato.

(a) [4 points] Si mostri l'evoluzione dell'algoritmo di Prim per il grafo in Figura 3 a partire dal nodo **3**. In particolare, indicato con T l'insieme degli archi dell'albero minimo ricoprente (inizialmente vuoto), per ogni iterazione i dell'algoritmo occorre specificare i) l'arco che verrà aggiunto a T nell'iterazione i-esima e ii) il peso complessivo dell'albero parziale ottenuto al termine dell'iterazione. Per descrivere l'evoluzione dell'algoritmo usare una tabella con il formato di quella sottostante seguente (o equivalente):

Iterazione	**Arco aggiunto a T**	**Peso complessivo**
1	...	...
2	...	...

Algoritmi e Strutture Dati - problemi e loro proprietà

4. **Ordinamento topologico** Si considerino grafi *orientati*.
   (a) [2 points] Si definisca *formalmente* il problema dell'ordinamento topologico di un grafo orientato $G = (V, E)$.
   (b) [2 points] i) Spiegare brevemente il significato di arco all'indietro (Back) in una visita in profondità (DFS); ii) *Dimostrare* che se G è un DAG (grafo orientato aciclico) allora una visita in profondità di G non può generare archi all'indietro.

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

5. **Costruzione di Alberi Binari di Ricerca** Il Professor Knowitall sostiene di aver progettato un algoritmo basato esclusivamente su confronti tra chiavi intere che, dato un array **A** *disordinato* contenente n interi, costruisce un albero binario di ricerca (BST) avente per chiavi gli elementi di **A** con costo computazionale $\mathcal{O}(n)$.
   (a) [3 points] Dimostrare che il Prof. Knowitall ha torto, facendo vedere che se quanto afferma fosse vero, il suo risultato contraddirebbe noti risultati sull'ordinamento basato su confronti visti a lezione.

   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. **Metodo Avido (Greedy)** Si consideri un insieme $\mathbf{A} = \{x_1, \ldots, x_m\}$ di n punti sull'asse reale, dove $x_1 < x_2 < \cdots < x_n$.

   (a) [2 points] Progettare e scrivere lo *pseudo-codice* di un algoritmo *efficiente* che, dato $\mathbf{A}$, calcola il *più piccolo*[2] insieme di intervalli chiusi di lunghezza unitaria che contiene tutti i punti dati.

   (b) [2 points] Dimostrare che l'algoritmo è corretto, ossia che l'insieme di intervalli unitari calcolato i) copre tutti i punti e ii) ha cardinalità minima.

   Il punteggio dipenderà molto dal rigore, dalla chiarezza espositiva e dalla solidità delle argomentazioni proposte. Non occorre scrivere tanto ma scrivere bene. Gli algoritmi devono essere descritti in pseudo-codice e comprensibili (ad esempio: variabili definite ecc.).

7. **Programmazione dinamica** Mario sta seguendo una dieta. Ha un elenco di n tipi di cibo da mangiare durante il giorno (può consumare fino a una porzione al giorno di ciascun tipo di cibo). Ogni tipo di cibo i ha una quantità di calorie c_i e un grado di soddisfazione s_i che dà a Mario quando ne mangia una porzione.

   Il nutrizionista ha detto che può mangiare quello che vuole da quella lista, purché non superi il limite consentito di K calorie. L'obiettivo di Mario è individuare i cibi che complessivamente rispettano il vincolo delle calorie e massimizzano il grado di soddisfazione complessivo (assumiamo che il grado di soddisfazione di un insieme di cibi sia dato dalla somma dei gradi di soddisfazione dei singoli cibi dell'insieme).

   (a) [3 points] Si proponga un algoritmo che utilizzi la programmazione dinamica per aiutare Mario a scegliere i pasti della giornata, in modo che il menù corrispondente sia il più *soddisfacente* possibile (cioè il grado di soddisfazione ricevuto dall'insieme dei cibi mangiati sia massimo). Si calcoli la complessità asintotica e si dimostri la correttezza dell'algoritmo proposto.

[2] Ossia, contenente il minimo numero di intervalli.