Strutture Dati Algoritmi e Complessità **Name:** _____
A.A. 2023/2024
Compito B
20/06/2024
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] **Ordinamento topologico.** In questo problema si fa riferimento a grafi semplici *diretti*. In particolare, ogni nodo ha associata la lista dei vicini uscenti (vicini ai quali il nodo è connesso da archi uscenti) e dei vicini entranti (nodi che hanno archi diretti verso il nodo considerato). Il grafo è descritto nella classe `Graph.java` (Java) e nel modulo `graph.c` (C). Il generico nodo è descritto nella classe emphinterna `Graph.GraphNode` (Java) e in `struct graph_node` (C).

   Sono inoltre già disponibili le primitive di manipolazione del grafo: creazione di grafo vuoto, lista dei vicini uscenti ed entranti di ciascun nodo (quest'ultima non necessaria per lo svolgimento di questo esercizio) 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. In particolare, per Java si rimanda alle classi al file `Graph.java`, per C si rimanda all'header `graph.h`. 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.

   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:

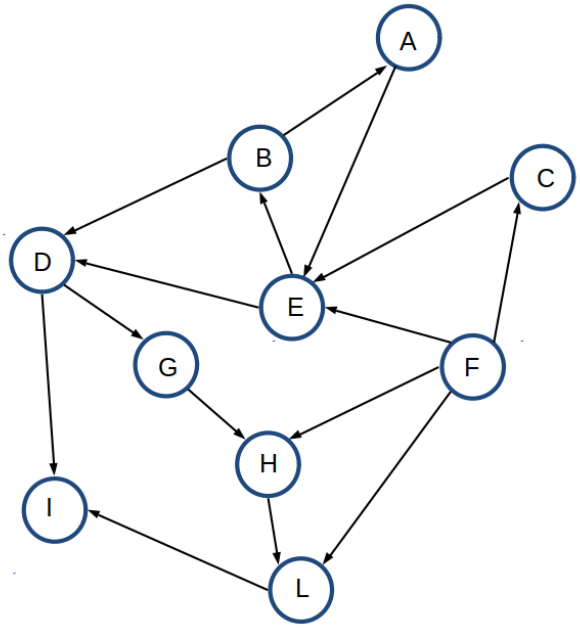

Figure 1: Esempio di grafo diretto. Il grafo in figura *non ammette* un ordinamento topologico, per la presenza del ciclo (A, E, B). Invece, il grafo ottenuto rimuovendo l'arco (B, A) *ammette* un ordinamento topologico. In tal caso, un possibile ordinamento topologico è "F C A E B D G H L I".

Implementare la funzione/metodo `public static <V> void tsort(Graph<V> g)` della classe `GraphServices` (o la funzione `void tsort(graph* g)` del modulo `graph_services`

in C) che, dato un grafo diretto `g`, verifica se `g` ammetta un ordinamento topologico o meno. Nel primo caso, la funzione/metodo deve stampare un possibile ordinamento topologico dei nodi, mentre nel secondo caso deve stampare un messaggio (ad esempio, "Il grafo dato non ammette un ordinamento topologico"). Si faccia riferimento alla Figura 1 per due esempi.

Algoritmi e Strutture Dati - nozioni di base

2. **[No studenti con DSA] Algoritmi di Ordinamento.** Si consideri il seguente array **A**:

7	2	9	13	15	1	5	3

   (a) [4 points] Si descriva l'evoluzione dell'algoritmo MERGE SORT a partire dall'array **A**.

   Non è necessario scrivere molto e non è necessario scrivere l'algoritmo, ma occorre descrivere in modo chiaro e non ambiguo l'evoluzione dell'algoritmo.

3. **Alberi Minimi Ricoprenti** Si consideri il grafo non diretto e pesato in Figura 2

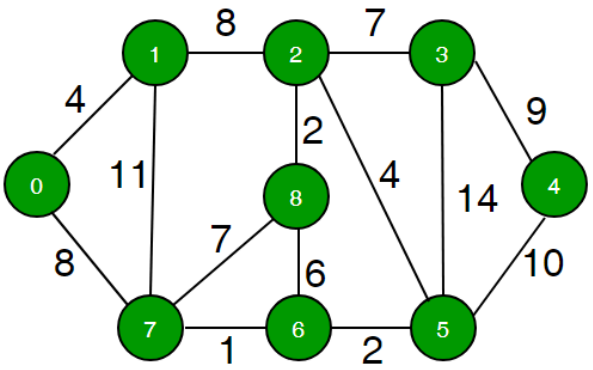

Figure 2: Grafo non diretto pesato.

   (a) [4 points] Si mostri l'evoluzione dell'algoritmo di Prim per il grafo in Figura 2 a partire dal nodo **1**. 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 (o equivalente):

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

Nota bene: descrizioni incomplete o confuse saranno penalizzate.

Algoritmi e Strutture Dati - proprietà di algoritmi e problemi

4. **[No studenti con DSA] Cammini Minimi.** Si considerino grafi orientati.
   (a) [2 points] Si definisca *formalmente* il problema dei cammini minimi su un grafo $G = (V, E)$ a partire da una sorgente s (con $s \in V$).
   (b) [2 points] Ricordando e richiamando esplicitamente la/le necessaria/e proprietà della visita in ampiezza (BFS) viste a lezione, *dimostrare* che se gli archi di G hanno tutti lo stesso peso positivo w (il valore di w non è importante ai fini della risposta) allora una BFS con sorgente il nodo s calcola i cammini minimi da s a tutti gli altri nodi del grafo. Per semplificare l'esposizione si assuma che G sia *non orientato e connesso*.

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

5. **Tabelle hash.** Si consideri una tabella hash T avente m posizioni, nella quale le collisioni sono gestite mediante concatenazione/liste di trabocco. La funzione hash h usata per mappare le chiavi nelle diverse posizioni della tabella si comporta in modo ideale, ossia è *uniforme e indipendente*, come visto a lezione.
   (a) [3 points] Si supponga ora che vengano inserite n chiavi *distinte* $k_1, \ldots, k_n$ usando la funzione h descritta sopra. Calcolare il *valore atteso* atteso del numero totale di collisioni. Precisamente, calcolare il valore atteso della cardinalità dell'insieme $\{(k_i, k_j) : i, j \in \{1, \ldots, n\}, i \neq j \text{ e } h(k_i) = h(k_j)\}$.

   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.** Un array bitonico $\mathbf{A}[1, \ldots, n]$ di lunghezza n è definito come un array contenente elementi *distinti*, nel quale esiste un indice k tale che: i) la sequenza $\mathbf{A}[1], \ldots, \mathbf{A}[k]$ è *crescente*; ii) la sequenza $\mathbf{A}[k], \ldots, \mathbf{A}[n]$ è decrescente.

   (a) [2 points] Progettare e scrivere lo *pseudo-codice* di un algoritmo *efficiente* che, dato un array bitonico $\mathbf{A}$, restituisce l'indice dell'elemento *massimo* contenuto in $\mathbf{A}$.

   (b) [2 points] Calcolare il costo computazionale dell'algoritmo proposto.

   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** È dato un insieme I di m interi compresi fra 1 e M. Sia W la somma di tutti gli elementi in I. Dovete trovare un sottoinsieme I_1 di I tale che la somma W_1 degli elementi di I_1 verifichi le seguenti condizioni: i) $W_1 \leq W/2$ (la somma degli elementi non è maggiore della metà di W); ii) W_1 è il valore più grande possibile fra tutti i sottoinsiemi di I che verificano la condizione $W_1 \leq W/2$.

   (a) [2 points] Scrivere lo pseudo-codice algoritmo di un efficiente che, dato in input I restituisce un sottoinsieme I_1 che verifichi le condizioni richieste.

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