Esame di

Algoritmi e strutture dati (parte di Fondamenti di informatica II 12 CFU)
Algoritmi e strutture dati (V.O., 5 CFU)
Algoritmi e strutture dati (Nettuno, 6 CFU)

Appello del 19-07-2021 – a.a. 2020-21 – Tempo: 2 ore – somma punti: 32

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, email, esame (vecchio o nuovo), linguaggio in cui si svolge l'esercizio 2 (Java o C). Per quanto riguarda il campo esame (vecchio o nuovo), possono optare per il vecchio esame gli studenti che nel periodo che va dal 2014-15 al 2017-18 (estremi inclusi) sono stati iscritti al II anno.

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 compresse `C-aux` e `java-aux`, contenenti il codice di supporto e lo scheletro delle soluzioni per il quesito di programmazione (quesito 1). 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`.
- Per i quesiti 2 e 3, creare due file `probl2.txt` e `probl.txt` contenenti, rispettivamente, gli svolgimenti dei problemi proposti nei quesiti 2 e 3; i tre file devono trovarsi nella cartella `Esame`. È possibile integrare i contenuti di tali file con eventuale materiale cartaceo *unicamente per disegni, grafici, tabelle, calcoli*.

Attenzione: i simboli < e > usati nei nomi dei file fanno parte del linguaggio dei metadati e non debbono essere inclusi nei nomi reali.

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 1: Progetto algoritmi C/Java [soglia minima per superare l'esame: 5/30]

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 `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), inserimento di un nuovo nodo, inserimento di un nuovo arco, get label/valore (stringa) di un dato nodo, cancellazione arco, cancellazione nodo (e relativi archi incidenti), cancellazione grafo, stampa grafo. 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.

In particolare, per Java si rimanda alle classi `Graph` e `GraphNode` (quest'ultima classe interna di e contenuta nel file `Graph.java`) e ai commenti contenuti in `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 dati nel testo che segue fanno riferimento alla figura seguente. che corrisponde al grafo usato nel programma di prova (`Driver.java` o `driver`):

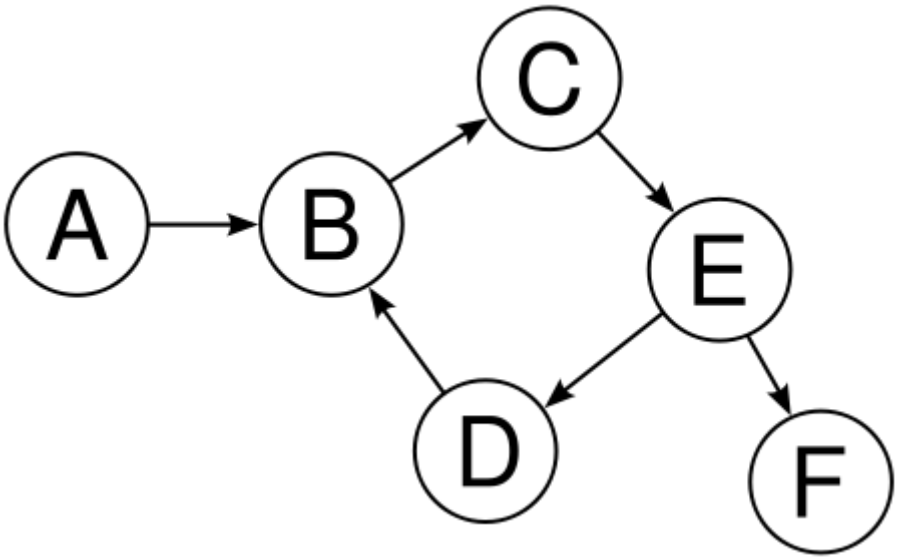

Figura 1. Esempio di grafo diretto pesato. Le distanze minime dal nodo **A** sono [**A**:0, **B**:1, **C**:2, **E**:3, **D**:4, **F**:4], mentre le distanze minime dal nodo **c** sono [**A**:100000, **B**:3, **C**:0, **D**: 2, **E**:1, **F**:2], **assumendo di rappresentare la distanza infinita attraverso un valore maggiore della somma dei pesi di tutti archi** (in questo caso si è scelto il valore 10000, che può anche essere usato dagli studenti).

1. Implementare la funzione/metodo `bfs(Graph<V> g, Node<V> source)` della classe `GraphServices` (o `graph_services` in C) che, dato un grafo `g` e un (oggetto) nodo `source` , esegue una visita in ampiezza (bfs) del grafo a partire dal nodo `source` . In particolare, per ogni nodo `v` raggiungibile da `source` , il metodo/funzione dovrà stampare il livello, ossia il numero minimo di archi per raggiungere `v` a partire da `source` . Ad esempio, rispetto alla Figura 1, se `source` fosse il nodo A la funzione/metodo dovrebbe stampare (l'ordine in cui il livello dei nodi viene stampato non è importante)

   ```
   BFS da A:
   Livello A: 0
   Livello B: 1
   Livello C: 2
   Livello E: 3
   Livello D: 4
   Livello F: 4
   ```

   Nota: Per la rappresentazione del livello di un nodo si consiglia di usare il campo `timestamp` della classe `Nodo` .
   Punteggio: [10/30]

Quesito 2: Algoritmi

1. Si consideri il problema di realizzare una funzione che associ hashcode a chiavi di tipo **Stringa**. Si supponga che la soluzione proposta preveda che l'hashcode di una stringa s sia ottenuto sommando gli interi che corrispondono ai codici ASCII dei primi 4 caratteri di s. Mostrare e discutere attraverso esempi i potenziali svantaggi di questa scelta se l'obiettivo è minimizzare il numero di collisioni tra chiavi diverse.

   Occorre dare argomentazioni convincenti. Il punteggio dipenderà anche dalla completezza e dalla chiarezza espositiva. Non occorre scrivere tanto ma scrivere bene.

   Punteggio: [8/30]

2. Si consideri una lista non ordinata rappresentata con un array. L'array è inizializzato a una dimensione pari a 1. Si consideri ora una successione di n inserimenti. Il generico inserimento ha costo costante *a meno che l'array non sia già pieno*. Se l'array è pieno l'inserimento determina i) l'allocazione di un *nuovo* array di dimensione $3x$ se x era la dimensione precedente dell'array, ii) la copia del contenuto del vecchio array nel nuovo (la copia del *singolo* valore dal vecchio array al nuovo ha costo costante), iii) l'inserimento del nuovo elemento (nel nuovo array).
   - a) Si calcoli il *costo temporale complessivo asintotico* della successione degli n inserimenti (si supponga che non vi siano mai rimozioni di elementi).
   - b) Si calcoli il costo ammortizzato per elemento.

   Occorre dare un'argomentazione quantitativa, rigorosa (prova) e chiara, giustificando adeguatamente la risposta. Il punteggio dipenderà soprattutto da questo.

   Punteggio: [7/30]

Quesito 3:

Si supponga di avere una lista non ordinata contenente n coppie `(key, e)`, dove `e` è il riferimento a un oggetto (ad esempio una pagina Web) e `key` è una chiave avente *valore numerico positivo*.

- Si proponga un algoritmo che sia il più efficiente possibile che, presa in input una lista come sopra, restituisca la sottolista ordinata contenente le k
  coppie il cui valore della chiave è più alto, in ordine decrescente rispetto al valore delle chiavi

  Punteggio: [4/30]

- Si calcoli il costo temporale asintotico dell'algoritmo proposto.

  Punteggio: [3/30]

Occorre descrivere chiaramente l'algoritmo proposto e dare un'argomentazione quantitativa e per il costo asintotico. Il punteggio dipenderà soprattutto da questo