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 5-7-2019 – a.a. 2018-19 – Tempo: 4 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.zip` e `java-aux.zip`, contenenti il codice di supporto e lo scheletro delle soluzioni per il quesito di programmazione (quesito 2). Svolgere il compito nel modo seguente:

- Per il quesito 2, estrarre la cartella `c-aux` o `java-aux` (a seconda del linguaggio che si intende usare) all'interno della cartella `Esame`, estraendola dal corrispondente file `.zip` . Alla fine la cartella `c-aux` (o `java-aux`, a seconda del linguaggio usato) conterrà le implementazioni delle soluzioni al quesito 2, 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 1, 3 e 4, creare tre file `prob1.<matricola>.txt`, `prob3.<matricola>.txt` e `prob4.<matricola>.txt`, contenenti, rispettivamente, gli svolgimenti dei problemi 1, 3 e 4; 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. 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.

Quesito 1: Analisi algoritmo

Si consideri l'algoritmo descritto dai metodi Java di seguito illustrati.

```
static int f(int q) {
    if(q <= 1) return q;
    if((q % 2) == 0) return g(q>>1);
    else return f(q-1);
}

static int g(int r) {
    if(r <= 1) return f(r);
    if((r % 3) == 0) return f(r/3);
    else return f(r+1);
}
```

Con riferimento a tali algoritmi si sviluppi quanto segue

1.1 Analisi costo temporale

Determinare il costo asintotico di caso peggiore di `f(int)` in funzione della dimensione dell'input. Risposte non motivate non saranno prese in considerazione.

Punteggio: [4/30]

1.2 Analisi di variante

Come cambia l'analisi precedente se l'ultima riga del metodo `f(int)` viene riscritta come segue?

```
  else return f(q+1); // invece di return f(q-1)
}
```

Anche in questo caso occorre motivare la risposta.

Punteggio: [3/30]

Quesito 2: Progetto algoritmi C/Java [soglia minima: 5/30]

In questo problema si fa riferimento a grafi semplici *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 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: creazione di grafo vuoto, get lista nodi, get lista archi aventi origine da un nodo dato, 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. 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:

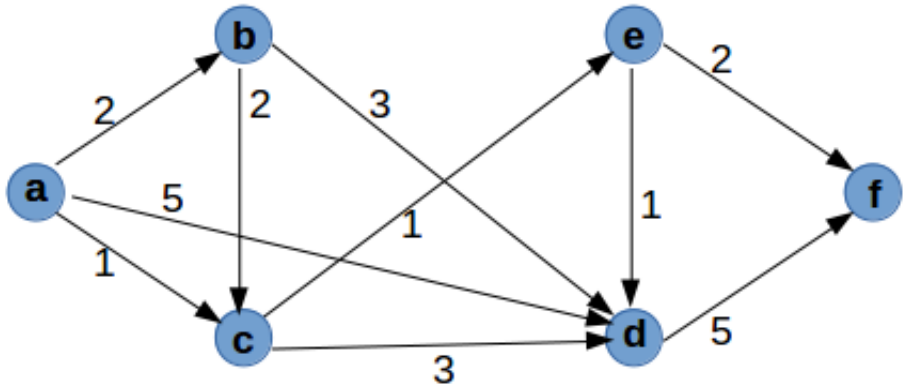

Figura 1. Esempio di grafo diretto pesato. Le distanze minime dal nodo **a** sono [**a**:0, **b**:2, **d**:3, **c**:1, **e**:2, **f**:4], mentre le distanze minime dal nodo **c** sono [**a**:100000, **b**:100000, **d**:2, **c**:0, **e**:1, **f**:3], **assumendo di rappresentare la distanza infinita attraverso un valore maggiore della somma dei pesi di tutti archi** (in questo caso si è scelto il valore 10000).

1. Implementare la funzione/metodo `bfs(Graph<V> g, Node<V> source)` della classe `GraphServices` (o `bfs(graph*, graph_node*)` in `graph_services.c` 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

```
BFS da a:
Livello a: 0
Livello b: 1
Livello c: 1
Livello d: 1
Livello e: 2
Livello f: 2
```

Nota: Per la rappresentazione del livello di un nodo si consiglia di usare il campo `timestamp` della classe `Nodo`.

Punteggio: [3/30]

2. Implementare la funzione/metodo `sssp(Graph<V> g, Node<V> source)` della classe `GraphServices` (o `sssp(graph*, graph_node*)` in `graph_services.c` in C) che, dato un grafo `g` e un (oggetto) nodo `source`, esegue il calcolo dei cammini minimi da `source` a tutti gli altri nodi raggiungibili nel grafo. In particolare, la classe deve restituire una stringa costituita da più righe. La i-esima riga deve dare le distanze di tutti i nodi dall'i-esimo nodo del grafo secondo il seguente formato:

   `Distanze dal nodo <identificatore i-esimo nodo> [<identificatore nodo>:distanza, ...]`.

   Qualora un nodo `v` non sia raggiungibile da `source`, la sua distanza sarà pari a un valore predefinito (preimpostato a 10000 nel codice di supporto fornito agli studenti).

   Ad esempio, con riferimento alla Figura 1, se `source` fosse il nodo **c**, la funzione/metodo dovrebbe stampare

   ```
   Distanze dal nodo c [a:100000, b:100000, d:2, c:0, e:1, f:3]
   ```

   Punteggio: [4/30]

3. Implementare la funzione/metodo `apsp(Graph<V> g)` della classe `GraphServices` (o `apsp(graph*)` in `graph_services.c` in C) che, dato un grafo `g`, esegue il calcolo dei cammini minimi tra tutti i nodi. In particolare, per ogni vertice `v`, la funzione deve stampare a schermo una riga con le distanze minime di tutti gli altri nodi da `v`. Ad esempio, con riferimento alla Figura 1, la funzione/metodo dovrebbe stampare

   ```
   Distanze dal nodo a [a:0, b:2, d:3, c:1, e:2, f:4]
   Distanze dal nodo b [a:100000, b:0, d:3, c:2, e:3, f:5]
   Distanze dal nodo d [a:100000, b:100000, d:0, c:100000, e:100000, f:5]
   Distanze dal nodo c [a:100000, b:100000, d:2, c:0, e:1, f:3]
   Distanze dal nodo e [a:100000, b:100000, d:1, c:100000, e:0, f:2]
   Distanze dal nodo f [a:100000, b:100000, d:100000, c:100000, e:100000,
   f:0]
   ```

   Nota: ovviamente ci si può giovare della soluzione fornita per il punto 2

   Punteggio: [3/30]

Quesito 3: Algoritmi

1. Si supponga di avere una tabella hash di dimensione iniziale n (ossia, la tabella può inizialmente contenere fino a n coppie (chiave, valore)). La tabella è inizialmente vuota. Si supponga che la tabella hash sia implementata particolarmente bene e che per fattori di carico non superiori a 0.5 il suo comportamento in termini di complessità delle operazioni sia ideale (questa è ovviamente una semplificazione, ma abbstanza realistica, almeno in media). Si supponga ora che siano inserite in sequenza $\lfloor \frac{n}{2} \rfloor$ coppie (chiave, valore) e che a seguito di un ulteriore inserimento (l'$(\lfloor n/2 \rfloor + 1)$-esimo) la dimensione della tabella venga portata a $2n - 1$ (in modo da mantenere non superiore a 0.5 il fattore di carico). Ciò premesso: i) Si valuti il costo dell'operazione $\lfloor \frac{n}{2} \rfloor + 1$-esima; ii) si determini il costo medio di ciascun inserimento, pari a *(costo totale di tutte le operazioni)/(numero di operazioni)*.

   Punteggio: [3/30]

2. Si consideri un min-heap inizialmente vuoto. Si supponga che, in sequenza, vengano inserite nell'heap n coppie (chiave, valore), *in ordine crescente rispetto all'ordinamento delle chiavi*. Si valuti il costo complessivo delle n operazioni di inserimento, nel caso peggiore. *Giustificare la risposta*.

   [assumere che le chiavi siano usate come priorità]

   Punteggio: [3/30]

3. Mostrare un'istanza del problema dei cammini minimi s-t (sorgente-destinazione) su un grafo pesato *diretto* che non ammette una soluzione a causa della presenza di (almeno) un ciclo di lunghezza negativa. *Istanza significa che occorre specificare il grafo, i pesi sugli archi, i nodi sorgente e destinazione.*

   Punteggio: [3/30]

Quesito 4:

Si consideri il grafo *non diretto e pesato* della Figura 2:

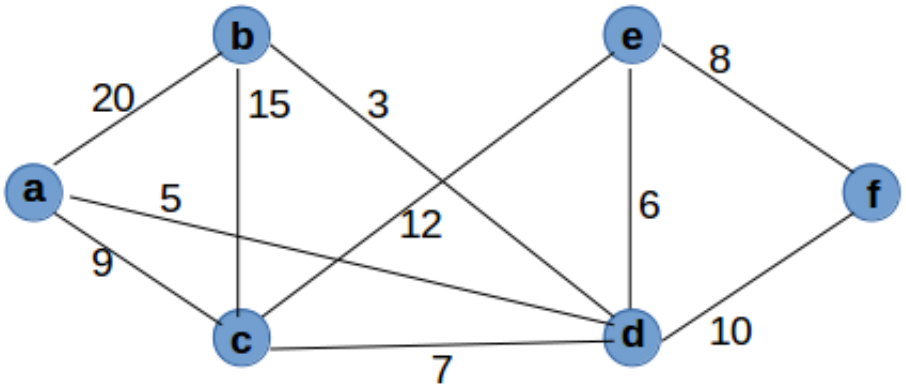

Figura 2. Esempio di grafo non diretto pesato.

Si risponda alle domande seguenti:

- Esiste un unico albero minimo ricoprente per il grafo in figura o può esisterne più di uno? *Non basta rispondere sì o no. Occorre dare una motivazione, anche senza dimostrazione.*

  Punteggio: [3/30]

- Si mostri l'evoluzione dell'algoritmo di Prim-Jarnik per il grafo in Figura 2 a partire dal nodo **b**. In particolare, per ogni iterazione *i* dell'algoritmo occorre specificare i) il sotto-insieme T degli *archi* dell'albero minimo ricoprente già identificati all'inizio dell'iterazione *i*-esima e ii) l'arco che verrà aggiunto a T nell'iterazione *i*-esima.

  Punteggio: [3/30]