Progetti di Algoritmi e Strutture Dati

Università di Bologna, corso di laurea in Informatica per il Management

Anno Accademico 2021/2022, sessione invernale

Ultimo aggiornamento 05/12/2022

Istruzioni
Il progetto consiste di quattro esercizi di programmazione da realizzare in Java. Lo svolgimento del progetto è obbligatorio per poter sostenere l'orale nella sessione cui il progetto si riferisce.

I progetti dovranno essere consegnati entro le ore **23:59 del 18/01/2023**. La prova orale potrà essere sostenuta in uno dei due appelli della sessione invernale (è obbligatoria l'iscrizione tramite AlmaEsami).

L'esito dell'esame dipende sia dalla correttezza ed efficienza dei programmi consegnati, sia dal risultato della discussione orale, durante la quale verrà verificata la conoscenza della teoria spiegata in tutto il corso (quindi non solo quella necessaria allo svolgimento dei progetti). L'orale è importante: **una discussione insufficiente comporterà il non superamento della prova**.

Modalità di svolgimento dei progetti
I progetti devono essere **esclusivamente frutto del lavoro individuale di chi li consegna; è vietato discutere i progetti e le soluzioni con altri** (sia che si tratti di studenti del corso o persone terze). La similitudine tra progetti verrà verificata con strumenti automatici e, se confermata, comporterà l'immediato annullamento della prova per TUTTI gli studenti coinvolti senza ulteriori valutazioni dei progetti.

È consentito l'uso di algoritmi e strutture dati definite nella libreria standard Java, nonché di codice messo a disposizione dai docenti sulla pagina del corso o sulla piattaforma "Virtuale"; è responsabilità di ciascuno verificare che il codice sia corretto (anche e soprattutto quello fornito dai docenti!). **Non è consentito fare uso di altro codice, anche se liberamente disponibile in rete**.

I programmi devono essere realizzati come applicazioni a riga di comando. Ciascun esercizio deve essere implementato in un singolo file sorgente chiamato Esercizio*N*.java, (Esercizio1.java, Esercizio2.java eccetera). Il file deve avere una classe pubblica chiamata Esercizio*N*, contenente il metodo statico main(); altre classi, se necessarie, possono essere definite all'interno dello stesso file. I programmi **non devono specificare il package** (quindi **non devono contenere** l'intestazione "package Esercizio1;" o simili).

I programmi devono iniziare con un blocco di commento contenente nome, cognome, numero di matricola e indirizzo mail (@studio.unibo.it) dell'autore. Nel commento iniziale è possibile indicare per iscritto eventuali informazioni utili alla valutazione del programma (ad esempio, considerazioni sull'uso di strutture dati particolari, costi asintotici eccetera).

I programmi verranno compilati dalla riga di comando utilizzando Java 11 (OpenJDK 11) con il comando:

javac Esercizio*N*.java

ed eseguiti sempre dalla riga di comando con

java -cp . Esercizio*N* *eventuali_parametri_di_input*

I programmi non devono richiedere nessun input ulteriore da parte dell'utente. Il risultato deve essere stampato a video rispettando **scrupolosamente** il formato indicato in questo documento, perché i programmi subiranno una prima fase di controlli semiautomatici. **Non verranno accettati programmi che producono un output non conforme alle specifiche**.

Si può assumere che i dati di input siano sempre corretti. Vengono forniti alcuni esempi di input per i vari esercizi, con i rispettivi output previsti. **Un programma che produce il risultato corretto con i dati di input forniti non è necessariamente corretto**. I programmi consegnati devono funzionare correttamente su *qualsiasi* input: a tale scopo verranno testati anche con input differenti da quelli forniti.

Alcuni problemi potrebbero ammettere più soluzioni corrette; in questi casi – salvo indicazione diversa data

nella specifica – il programma può restituirne una qualsiasi, anche se diversa da quella mostrata nel testo o fornita con i dati di input/output di esempio. Nel caso di esercizi che richiedano la stampa di risultati di operazioni in virgola mobile, i risultati che si ottengono possono variare leggermente in base all'ordine con cui vengono effettuate le operazioni, oppure in base al fatto che si usi il tipo di dato `float` o `double`; tali piccole differenze verranno ignorate.

I file di input assumono che i numeri reali siano rappresentati usando il punto ('.') come separatore tra la parte intera e quella decimale. Questa impostazione potrebbe non essere il default nella vostra installazione di Java, ma è sufficiente inserire all'inizio del metodo `main()` la chiamata:

```
Locale.setDefault(Locale.US);
```

per impostare il separatore in modo corretto (importare `java.util.Locale` per rendere disponibile il metodo).

Ulteriori requisiti

La correttezza delle soluzioni proposte deve essere dimostrabile. In sede di discussione dei progetti potrà essere richiesta la dimostrazione che il programma sia corretto. Per "dimostrazione" si intende una dimostrazione formale, del tipo di quelle descritte nel libro o viste a lezione per garantire la correttezza degli algoritmi. Argomentazioni fumose che si limitano a descrivere il programma riga per riga e altro non sono considerate dimostrazioni.

Il codice deve essere leggibile. Programmi incomprensibili e mal strutturati (ad es., contenenti metodi troppo lunghi, oppure un eccessivo livello di annidamento di cicli/condizioni – "if" dentro "if" dentro "while" dentro "if"…) verranno fortemente penalizzati o, nei casi più gravi, rifiutati.

Usare nomi appropriati per variabili, classi e metodi. L'uso di nomi inappropriati rende il codice difficile da comprendere e da valutare. L'uso di nomi di identificatori deliberatamente fuorviante potrà essere pesantemente penalizzato in sede di valutazione degli elaborati.

Commentare il codice in modo adeguato. I commenti devono essere usati per descrivere in modo sintetico i punti critici del codice, non per parafrasarlo riga per riga.

Esempio di commenti inutili	***Esempio di commento appropriato***

`v = v + 1;  // incrementa v`
`if ( v>10 ) { // se v e' maggiore di 10`
`  v = 0;  // setta v a zero`
`}`
`G.Kruskal(v); // esegui l'algoritmo di Kruskal`
	`// Individua la posizione i del primo valore`
	`// negativo nell'array a[]; al termine si ha`
	`// i == a.length se non esiste alcun`
	`// valore negativo.`
	`int i = 0;`
	`while ( i < a.length && a[i] >= 0 ) {`
	`  i++;`
	`}`

Ogni metodo deve essere preceduto da un blocco di commento che spieghi in maniera sintetica lo scopo di quel metodo.

Lunghezza delle righe di codice. Le righe dei sorgenti devono avere lunghezza contenuta (indicativamente minore o uguale a 80 caratteri). Righe troppo lunghe rendono il sorgente difficile da leggere e da valutare.

Usare strutture dati adeguate. Salvo dove diversamente indicato, è consentito l'utilizzo di strutture dati e algoritmi già implementato nella JDK. Decidere quale struttura dati o algoritmo siano più adeguati per un determinato problema è tra gli obiettivi di questo corso, e pertanto avrà un impatto significativo sulla valutazione.

Modalità di consegna
I sorgenti vanno consegnati tramite la piattaforma “Virtuale” caricando i singoli file .java (Esercizio1.java, Esercizio2.java, eccetera). Tutto il codice necessario a ciascun esercizio deve essere incluso nel relativo sorgente; non sono quindi ammessi sorgenti multipli relativi allo stesso esercizio.

Forum di discussione
È stato creato un forum di discussione sulla piattaforma "Virtuale". Le richieste di chiarimenti sulle specifiche degli esercizi (cioè sul contenuto di questo documento) **vanno poste esclusivamente sul forum** e non via mail ai docenti. Non verrà data risposta a richieste di fare debug del codice, o altre domande di programmazione: queste competenze devono essere già state acquisite, e verranno valutate come parte dell'esame.

Valutazione dei progetti
Gli studenti ammessi all'orale verranno convocati per discutere i progetti, secondo un calendario che verrà comunicato sulla pagina del corso. Di norma, **potranno accedere all'orale solo coloro che avranno svolto gli esercizi del progetto in modo corretto.**

La discussione include domande sugli esercizi consegnati e sulla teoria svolta a lezione. Chi non sarà in grado di fornire spiegazioni esaurienti sul funzionamento dei programmi consegnati durante la prova orale riceverà una valutazione insufficiente con conseguente necessità di rifare l’esame da zero in una sessione d’esame successiva su nuovi progetti. Analogamente, una conoscenza non sufficiente degli argomenti di teoria, anche relativi a temi non trattati nei progetti, comporterà il non superamento della prova.

La valutazione dei progetti sarà determinata dai parametri seguenti:

- Correttezza dei programmi implementati;
- Efficienza dei programmi implementati;
- Chiarezza del codice: codice poco comprensibile, ridondante o inefficiente comporterà penalizzazioni, indipendentemente dalla sua correttezza. **L'uso di nomi di identificatori fuorvianti o a casaccio verrà fortemente penalizzato**.
- Capacità dell'autore/autrice di spiegare e giustificare le scelte fatte, di argomentare sulla correttezza e sul costo computazionale del codice e in generale di rispondere in modo esauriente alle richieste di chiarimento e/o approfondimento da parte dei docenti.

Checklist

Viene riportata in seguito un elenco di punti da controllare prima della consegna:

1. Ogni esercizio è stato implementato in un UNICO file sorgente Esercizio*N*.java?
2. I programmi compilano dalla riga di comando come indicato in questo documento?
3. I sorgenti includono all'inizio un blocco di commento che riporta cognome, nome, numero di matricola e indirizzo di posta (@studio.unibo.it) dell'autore?
4. I programmi consegnati producono il risultato corretto usando gli esempi di input forniti?

Esercizio 1.

La società MAGIC intende progettare e implementare in Java un algoritmo per la gestione di dati provenienti da un sensore ambientale. I dati prodotti devono essere memorizzati in una struttura dati **S**. L'i-esimo dato prodotto dal sensore è costituito dalla tripla

<i, TSI(i), TSE(i)>, TSI(i) e TSE(i) interi positivi.

Per ogni tripla:

a) TSE(i) > TSI(i) per i= 1,2, 3, ….

b) TSI(i) > TSI(i-1) per i = 2, 3, 4…

Esempio:

{primo dato generato:<1, 12, 16>}, {secondo dato generato<2, 14, 15>}, {terzo dato generato<3, 16, 19>} … {i-esimo dato generato <i, X, Y>: Y>X e X > TSI(i-1)}

Inizialmente il sensore produce K triple {i= 1, 2,..K} che vengono opportunamente memorizzate in **S**.

Le operazioni previste su **S**:

a) Selezionare la tripla presente in S caratterizzata dal valore minimo della differenza tra TSE(i) – TSI(i). MIN((TSE(i) – TSI(i), i = 1,2,..K}. Nel caso ci fossero più triple che soddisfino la condizione, si seleziona la tripla con valore dell'indice i minore;

b) Eliminare la tripla selezionata al punto a);

c) Richiedere al sensore una nuova tripla e inserirla opportunamente in S;

Si ipotizza che il ciclo di operazioni descritte in a), b) e c) venga eseguito continuativamente.

Progettare e implementare S in modo che:

1) il massimo numero di accessi agli elementi di **S** per identificare la tripla di cui ai punti a) e b) sia pari ad **1**;

2) il costo dell'inserimento di cui al punto c) sia non superiore a $\lfloor \log(\mathbf{K}) \rfloor$

Discutere brevemente la soluzione proposta considerando i requisiti (soddisfacibili?) definiti in precedenza (punti 1) e 2))

Per verificare la correttezza dell'implementazione proposta, si richiede di effettuare alcuni test assumendo che i valori di K (15<K<27), TSI(1) e TSE(1) (TSE(1) > TSI(1)) siano dati in input (a video). Inoltre:

Per i = 2, 3, 4, ….:

TSI(i) = TSI(i-1) + Z(i);

TSE(i) = TSI(i) + T(i);

Z(i) intero uniformemente distribuito in [4, 10];

T(i) intero uniformemente distribuito in [2, 7];

Per la generazione dei valori Z(i) e T(i) utilizzare opportunamente i generatori pseudocasuali di Java, (java.util.Random); come seme utilizzare il proprio numero di matricola

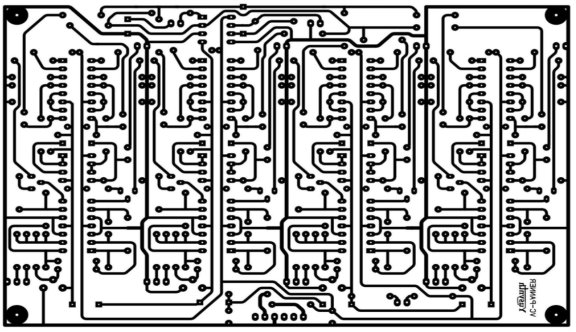
Eseguire il ciclo di operazioni a), b) e c) 2*K volte.

i) Stampare a video la lista delle K triple inizialmente inserite nella struttura S;

ii) Stampare a video la lista delle K triple in S dopo aver effettuato il ciclo di operazioni richiesto.

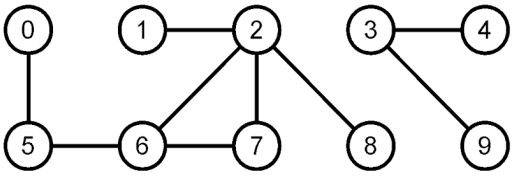
Esercizio 2.

Un circuito stampato, come illustrato nella seguente figura,



può essere rappresentato come un *grafo non orientato* $G=(V, E)$, con $V = \{0, 1, \ldots n\text{-}1\}$ l'insieme degli $n \geq 2$ "piedini" (pin) ed E l'insieme delle $m \geq 1$ connessioni che collegano tra loro coppie di pin. I segnali elettrici si propagano da un pin a tutti quelli ad esso collegati, sia direttamente che indirettamente tramite altri pin.

Ad esempio, il circuito seguente ha n=10 pin e m=9 connessioni:



I pin 0 e 8 sono tra di loro collegati, mentre 1 e 9 non lo sono.

Scrivere un programma Java che legge da un file la descrizione del circuito e da un secondo file una sequenza di coppie di pin. Il programma deve stampare a video, per ogni coppia nel secondo file, se i pin sono connessi oppure no. E' richiesto che il programma costruisca l'insieme delle componenti connesse del grafo mediante una struttura ***Union-Find***; per ogni arco $\{u, v\}$ del grafo, il programma unisce gli insiemi contenenti u e v. Terminata questa fase, due pin s e d sono connessi se e solo se `find(s) == find(d)`. E' lasciata libertà di decidere il tipo di struttura Union-Find da implementare (QuickUnion, QuickFind, con o senza euristiche), tenendo presente che l'implementazione corretta di una euristica comporterà un punteggio maggiore in sede di valutazione. Non è consentito fare ricorso a implementazioni già pronte delle strutture Union-Find, nemmeno se presenti nella libreria standard di Java.

Il primo file di input, passato sulla riga di comando, contiene la descrizione del grafo (o del circuito). Ad esempio, il grafo in figura può essere descritto dal file di configurazione seguente (le parole in corsivo non fanno parte del file di configurazione):

```
10          Numero di nodi n
9           Numero di archi m
0 5         origine, destinazione arco 0
5 6
6 7
6 2
1 2
2 7
2 8
3 4
3 9         origine, destinazione arco m-1
```

Il secondo file contiene $q \geq 1$ coppie di pin da testare. Ad esempio:

```
6          Numero q di coppie da controllare
1 0        origine, destinazione coppia 0 da testare
5 0
3 1
8 1
8 2
8 9        origine, destinazione coppia q-1 da testare
```

A fronte degli input sopra, il programma deve produrre a video l'output seguente:

```
1 0 C
5 0 C
3 1 NC
8 1 C
8 2 C
8 9 NC
```

L'output è composto da q righe, ciascuna contenente gli identificatori dei pin, nell'ordine in cui compaiono nel secondo file di input, e la stringa C se sono connessi, NC se non lo sono.
Analizzare il costo asintotico della prima fase di costruzione delle strutture Union-Find, e della seconda fase di esecuzione di tutte le query.

Esercizio 3.

Un distributore automatico di souvenir al museo del Louvre a Parigi contiene n monete aventi valore rispettivamente $m[0]$, $m[1]$, ... $m[n - 1]$. I valori delle monete sono interi positivi; è possibile che esistano più monete aventi lo stesso valore, ed è ugualmente possibile che i valori delle monete siano diversi da quelli cui siamo abituati (ad esempio, potrebbero essere presenti monete con valore $m[...]=3$).
Dati i parametri di cui sopra, vogliamo **determinare il *minimo* valore del resto**, intero positivo, che il distributore *non* è in grado di erogare.

Il programma accetta come unico parametro il nome di un file contenente l'input nel formato seguente:

```
6          numero n di monete
1          m[0]
2
2
5
10
20         m[n-1]
```

La prima riga contiene il numero n di monete presenti nel serbatoio ($n \geq 1$); seguono n righe contenenti i valori delle monete, in un ordine qualsiasi. Il programma deve stampare a video il valore del minimo resto non erogabile. Nell'esempio sopra stamperà:

```
41
```

Se il file di input contenesse i seguenti valori:

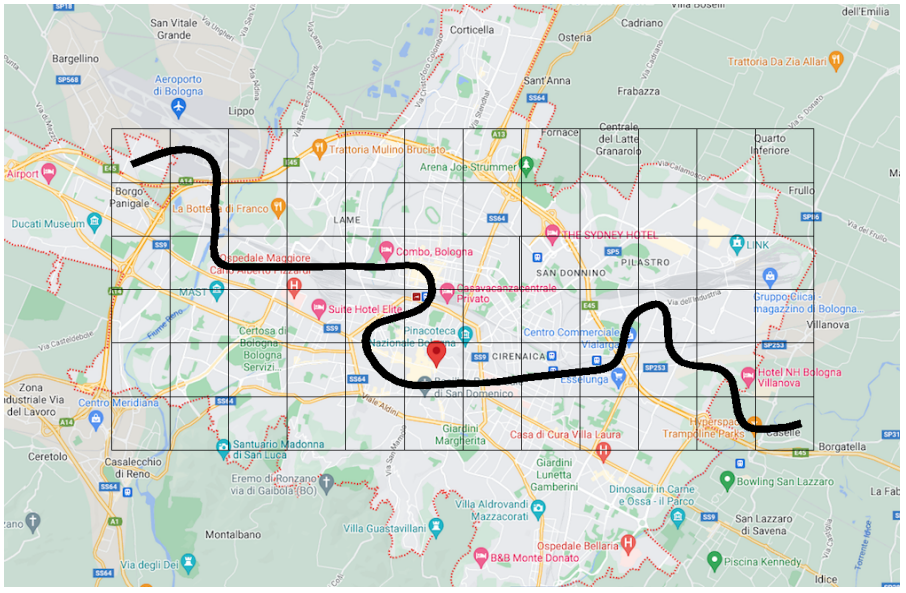
```
10          numero n di monete
2           m[0]
1
2
50
50
20
20
20
10
10          m[9]
```

il risultato sarà

```
6
```

Esercizio 4.

Una certa città, come Bologna, può essere descritta da una mappa suddivisa in $n \times m$ celle. La matrice $H[0..n - 1, 0..m - 1]$ di n righe ed m colonne viene usata per codificare l'altezza media sul livello del mare di ciascuna cella, per cui l'elemento $H[i, j]$ indica l'altezza media sul livello del mare della cella (i, j) che si trova nella riga i e colonna j. Le altezze sono numeri reali arbitrari, anche negativi, in quanto una o più porzioni di terreno possono trovarsi sotto il livello del mare.
Si vuole costruire una strada che parta dalla cella di coordinate (0, 0), per esempio Aeroporto di Bologna, e termini nella cella di coordinate $(n - 1, m - 1)$, per esempio, una certa destinazione o un albergo. La strada deve passare attraverso celle adiacenti, cioè che hanno un lato in comune. Un esempio di strada valida è rappresentato nella figura seguente.



Supponiamo che la strada attraversi le k celle di coordinate (x_0, y_0), (x_1, y_1), … (x_{k-1}, y_{k-1}); è necessario rispettare i vincoli seguenti:

i. La prima cella (x_0, y_0) deve corrispondere a quella di coordinate (0, 0);
ii. L'ultima cella (x_{k-1}, y_{k-1}) deve corrispondere a quella di coordinate (n - 1, m - 1);
iii. Ogni cella deve essere adiacente (avere un lato in comune) alla successiva. Formalmente, per ogni $i = 0, \ldots k - 2$, la cella (x_i, y_i) deve avere un lato in comune con (x_{i+1}, y_{i+1});
iv. La strada non può mai attraversare più di una volta la stessa cella.

La costruzione della strada ha un costo, che si compone di due parti:

1. Si sostiene un costo fisso pari a C_{cell} per ogni cella attraversata;
2. Per ogni coppia di celle adiacenti attraversate, (x_i, y_i) e (x_{i+1}, y_{i+1}), si paga un ulteriore costo pari a C_{height} moltiplicato per il quadrato del dislivello tra le celle, cioè $C_{height} \times (H[x_i, y_i] - H[x_{i+1}, y_{i+1}])^2$. Si noti che tale costo potrebbe essere zero nel caso in cui non ci sia dislivello tra le celle (x_i, y_i) e (x_{i+1}, y_{i+1}).

Quindi, se la strada attraversa le k celle $(x_0, y_0), \ldots (x_{k-1}, y_{k-1})$, il suo costo totale sarà

$$C_{cell} \times k + C_{height} \times \sum_{i=0}^{k-2} \left(H\left[x_i, y_i\right] - H\left[x_{i+1}, y_{i+1}\right]\right)^2$$

Progettare e implementare un algoritmo efficiente che determina le celle su cui ***costruire la strada di costo complessivo minimo***, rispettando i vincoli di cui sopra (**i-iv**).

Il programma accetta sulla riga di comando il nome di un file di input avente la struttura seguente:

```
50.0                                    # costo C_cell (reale positivo)
1000.0                                  # costo C_height (reale positivo)
6                                       # numero di righe n (intero positivo)
5                                       # numero di colonne m (intero positivo)
100.0  123.9  129.3  98.3   100.4       # H[0, 0] ... H[0, m-1] (reali arbitrati separati da spazi o tab)
99.7   124.8  108.1  102.5  101.2       # H[1, 0] ... H[1, m-1]
98.6   99.7   127.3  105.4  104.4
102.9  112.2  100.9  102.3  103.3
105.9  107.2  182.9  198.3  104.3
105.9  106.7  170.3  207.3  100.1       # H[n-1, 0] ... H[n-1, m-1]
```

Il programma deve stampare a video le coordinate delle celle attraversate (una per riga), e il costo complessivo della strada. Le celle devono essere stampate in ordine, a partire da quella di coordinate (0, 0) fino all'ultima di coordinate (n - 1, m - 1). Ad esempio, nel caso precedente l'output sarà simile a:

```
0 0
1 0
2 0
3 0
4 0
5 0
5 1
4 1
3 1
3 2
3 3
3 4
4 4
5 4
204670.0
```