Progetti di Algoritmi e Strutture Dati

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

Anno Accademico 2020/2021, sessione invernale

Ultimo aggiornamento 6/12/2021

Istruzioni

Il progetto consiste in cinque 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 **23:59 del 14/1/2022**. La prova orale deve essere sostenuta nell'unico appello della sessione autunnale (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 `EsercizioN.java`, (`Esercizio1.java`, `Esercizio2.java` eccetera). Il file deve avere una classe pubblica chiamata `EsercizioN`, 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 EsercizioN.java
```

ed eseguiti sempre dalla riga di comando con

```
java -cp . EsercizioN 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 file 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 obbiettivi 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 **almeno quattro esercizi in modo corretto**.

La discussione includerà 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 `EsercizioN.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 i file di input forniti?

Esercizio 1: Portafogli Finanziari

Una società di investimenti gestisce diversi prodotti finanziari ognuno dei quali è associato ad un certo valore (modificabile in base agli andamenti del mercato) ed identificato univocamente da un codice. Un cliente, può costruire un "portafoglio", ovvero può acquistare uno o più prodotti finanziari tra quelli proposti dalla società. In ogni portafoglio uno stesso prodotto può comparire più volte. Ogni cliente è identificato univocamente da una matricola a cui viene associata la collezione di prodotti acquistati che ne definisce il portafoglio.

Si implementino le strutture dati che permettano di gestire i prodotti e i portafogli dei clienti di tale società. Tali strutture dovranno permettere di aggiungere e rimuovere prodotti, modificare o visualizzare il valore di un certo prodotto, inserire o rimuovere clienti, modificare o visualizzare il contenuto del portafoglio di un certo cliente (in termini di codici identificativi dei prodotti).

Codici dei prodotti e matricole dei clienti sono vincolati ad essere di tipo stinga (sequenza di caratteri alfabetici maiuscoli o minuscoli, e caratteri numerici; si fa differenza tra maiuscole e minuscole); i codici prodotto devono iniziare con la lettera 'P' mentre i codici dei clienti devono iniziare con la lettera 'C'. L'inserimento di identificativi non validi rispetto a queste regole non è permesso.

Il software dovrà permettere le operazioni di inserimento, modifica ecc.. tramite sequenze di comandi la cui sintassi è riportata di seguito.

Comando	**Descrizione**
m *ID_prod val*	Modifica il valore del prodotto identificato dal codice *ID_prod* al nuovo valore *val*
p *ID_prod val*	Inserisce un nuovo prodotto identificato dal codice *ID_prod* di valore *val*
c *matr*	Inserisce un nuovo cliente identificato dalla matricola *matr*
a *matr ID_prod n*	Aggiunge al portafoglio del cliente con matricola *matr* una quantità *n* del prodotto *ID_prod*
r *matr ID_prod n*	Toglie al portafoglio del cliente con matricola *matr* una quantità *n* del prodotto *ID_prod*
k *cod*	Cancella un prodotto identificato da *cod* o un cliente identificato da *cod*
v *matr*	Elenca codici e valori dei prodotti presenti nel portafogli del cliente identificato dalla matricola *matr*

Quando un prodotto o un cliente vengono aggiunti è necessario controllare che il codice identificativo o la matricola non siano già presenti nella struttura dati. Qualora il codice o la matricola fossero già stati utilizzati si stampi il messaggio *identificativo già presente*. Per gli altri tipi di comando, controllare che *matr* o *ID_prod* sia effettivamente presente nella struttura dati. Qualora un identificativo (codice prodotto o matricola) non siano presenti si stampi il messaggio *identificativo non trovato.*

I comandi devono essere letti da un file (un comando per riga) il cui nome viene passato sulla riga di comando che porta all'esecuzione del programma:

```
java -cp . Esercizio1 nome_file_comandi
```

Esercizio 2: Ceste natalizie

Alla fine di ogni anno, una cooperativa di consumatori decide di preparare delle ceste natalizie per i soci, distribuendo in modo equo i prodotti invenduti che ha in magazzino.

Il valore di una cesta corrisponde alla somma del valore dei prodotti contenuti.

La cooperativa segue lo schema descritto di seguito per dividere equamente i prodotti tra le ceste, in modo da cercare di far rientrare la differenza tra la cesta di valore massimo e quella di valore minimo all'interno di un valore di soglia dato. Ogni cesta deve avere almeno un prodotto. Per riempire le ceste, fino ad esaurimento dei prodotti, si seleziona il prodotto di valore massimo e lo si mette nella cesta di valore minimo.

Vengono dati in input il numero n di soci, il valore di soglia s e la lista di prodotti $l[0..m - 1]$ rappresentati dal loro valore.

La cooperativa vuole un algoritmo efficiente che, seguendo lo schema sopra, indichi se è possibile assegnare almeno un prodotto ad ogni cesta in modo tale che la differenza tra il valore della cesta di valore massimo e quella di valore minimo sia minore o uguale alla soglia. In caso affermativo, l'algoritmo stampa due valori "X Y" dove X e Y sono rispettivamente i valori della cesta di valore minimo e quella di valore massimo. In caso contrario, l'algoritmo stampa "NO".

Si noti che se il numero di prodotti m è minore del numero di soci n, il programma deve stampare "NO" in quanto non è possibile inserire almeno un prodotto in ognuna delle n ceste.

Il programma accetta come parametro a riga di comando il nome di un file di testo nel seguente formato:

- la prima riga contiene il numero n di soci ($n > 1$);
- la seconda riga contiene un intero $s > 0$ corrispondente al valore di soglia;
- la terza riga contiene una lista non vuota di interi positivi separati da spazi, corrispondente alla lista dei valori dei prodotti in magazzino.

Esempio.

Input:	*Output:*
5 4 2 5 3 11 4 3 1 15 7 8 10	13 15
Input:	*Output:*
5 4 1 1 1 11 1 3 1 2 7 8	NO
Input:	*Output:*
5 10 1 1 1 1	NO

Esercizio 3: Il cavallo degli scacchi

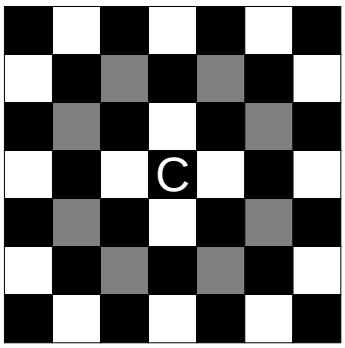
Consideriamo una scacchiera rettangolare con *n* righe e *m* colonne, in cui le celle sono identificate come gli elementi di una matrice: la cella (0, 0) si trova in alto a sinistra, mentre la cella (*n* - 1, *m* - 1) si trova in basso a destra. Un cavallo degli scacchi, più un numero qualsiasi di altri pezzi (ad esempio, pedoni), sono disposti sulla scacchiera. Scopo del programma è di determinare se il cavallo può raggiungere qualunque cella libera in una o più mosse, senza mai dover "mangiare" un altro pezzo.

Una mossa del cavallo consiste in:

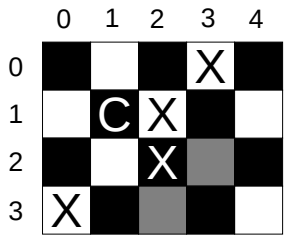
- uno spostamento orizzontale di due caselle seguito da uno spostamento verticale di una; oppure
- uno spostamento orizzontale di una casella seguito da uno spostamento verticale di due

in modo che il tragitto percorso formi idealmente una "L".

Ad esempio, nella figura seguente le otto caselle in grigio indicano le posizioni raggiungibili da un cavallo che si trovi inizialmente nella casella *C*. È ovviamente necessario che la casella di destinazione ricada all'interno della scacchiera.



Il cavallo può "saltare" sopra altri pezzi. Nell'esempio seguente, se *C* indica il cavallo e *X* indica una casella occupata da un altro pezzo, lo spostamento del cavallo dalla casella (1, 1) alla casella (2, 3) è ammesso, mentre la mossa che porta da (1, 1) a (0, 3) comporta la cattura del pezzo che si trova in (0, 3) e quindi non va considerata ai fini del nostro problema.



Il programma accetta su riga di comando il nome di un file di input avente il contenuto simile all'esempio seguente:

```
8 10
...XXXX..X
..XXX....X
...XC..XX.
.X....X..X
XX.X..X...
.XX.X..XX.
..XX...X.X
...XXX....
```

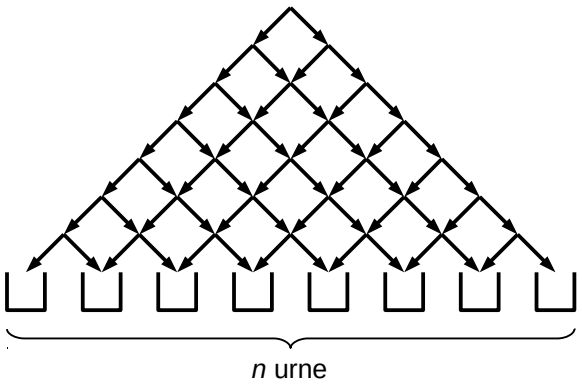
La prima riga contiene il numero di righe *n* (intero, $n \geq 1$) e di colonne *m* (intero, $m \geq 1$), separate da spazi o tab; seguono *n* righe composte da *m* caratteri ciascuna. I caratteri ammessi sono la lettera `C` maiuscola, che indica la posizione iniziale del cavallo, il punto (`.`) che indica una casella vuota, e la `X` maiuscola che indica la casella occupata da un altro pezzo. Si garantisce che esista sempre una unica casella etichettata con `C`.
Il programma deve stampare a video un output come quello che segue:

```
CCCXXXXCCX
CCXXXCCCCX
CCCXCCCXXC
CXCCCCXCCX
XXCXCCXCCC
CXXCXCCXXC
CCXXCCCXCX
.CCXXXCCC.
false
```

L'output deve contenere la scacchiera di input, in cui tutte le caselle raggiungibili dal cavallo sono etichettate con il carattere `C` maiuscolo (le caselle occupate restano etichettate con una `X` maiuscola). Segue una riga che contiene la stringa `true` se il cavallo è in grado di raggiungere tutte le caselle libere, `false` altrimenti. Nell'esempio sopra il cavallo non è in grado di raggiungere due caselle, quindi il programma ha stampato `false`.
Attenzione: non è necessario che il cavallo raggiunga tutte le celle libere con una unica sequenza di mosse (questo sarebbe un altro problema detto "*knight's tour*" e richiederebbe l'impiego di tecniche algoritmiche non viste a lezione); è sufficiente che, data una qualunque cella libera, esista una sequenza di mosse che porti il cavallo dalla sua posizione iniziale a quella cella.

Esercizio 4: Biglie

Si considerino *n* urne sopra alle quali è disposta una griglia di scivoli orientati come segue:



Una biglia viene posizionata in cima e scende verso il basso; ad ogni biforcazione, la biglia cade a destra con probabilità p e a sinistra con probabilità $(1 - p)$, dove p è un parametro reale compreso tra 0 e 1, fissato inizialmente.

Scrivere un programma che simula la caduta di *m* biglie, e al termine stampa la percentuale di biglie che si trovano su ciascuno degli *n* contenitori, da sinistra verso destra.

Il programma riceve tre parametri sulla riga di comando:

- Il numero *n* di urne (intero positivo);
- La probabilità *p* che ogni biglia cada verso destra (reale compreso tra 0 e 1, estremi esclusi);
- Il numero *m* di biglie da fare cadere (intero positivo).

Per decidere se la pallina deve cadere a sinistra oppure a destra occorre usare un generatore di valori pseudocasuali. A tale scopo si usi la classe `java.util.Random`, creando un oggetto `Random` usando le ultime otto cifre del proprio numero di matricola come seme. Ad esempio, se il proprio numero di matricola è 900012345678

```
import java.util.Random;
Random rnd = new Random(12345678L);
```

(la lettera L finale indica che il letterale 12345678 deve essere interpretata come un long). È ora possibile estrarre una sequenza di valori casuali uniformemente distribuiti in [0, 1] con il metodo `nextDouble()`. Per andare a destra con probabilità *p* si può scrivere:

```
if (rnd.nextDouble() < p) {
    vai a destra
} else {
    vai a sinistra
}
```

Si deve usare sempre lo stesso oggetto `Random`, che va quindi creato **una sola volta** all'inizio; non bisogna crearne uno nuovo per ogni pallina!

Ad esempio:

```
java Esercizio4 10 0.3 100000
```

potrebbe stampare qualcosa di simile a:

```
0 0.040620
1 0.156770
2 0.267110
3 0.266240
```

```
4 0.170520
5 0.073040
6 0.021210
7 0.004120
8 0.000340
9 0.000030
```

Ogni riga contiene un intero che indica l'urna (iniziando da 0, che indica l'urna più a sinistra), e il valore reale indica la frazione delle biglie che è caduta in quell'urna.

Nota: Il risultato del programma dipende dalla sequenza di numeri casuali generati durante la sua esecuzione, che verosimilmente sarà diversa per ciascuno (dipende dal numero di matricola). Pertanto, non vengono forniti file di output perché non sarebbero comunque rappresentativi di ciò che si dovrebbe ottenere.

Esercizio 5: Disuguaglianze

Si considerino m disuguaglianze tra n variabili $x_0, x_1, \ldots x_{n-1}$; le disuguaglianze sono tutte del tipo:

$x_i \leq x_j$

con i e j compresi tra 0 e n - 1 (estremi inclusi). Progettare e realizzare un algoritmo che determina se, dalle disuguaglianze fornite, è possibile dedurre che due variabili date sono uguali. Ad esempio, con l'input seguente (i numeri all'inizio servono semplicemente per fare riferimento alle singole diseguaglianze, ma non compaiono nell'input):

1. $x_0 \leq x_3$
2. $x_3 \leq x_5$
3. $x_5 \leq x_4$
4. $x_4 \leq x_2$
5. $x_2 \leq x_3$

possiamo dedurre che l'uguaglianza $x_2 = x_5$ è sempre vera, perché

- da 5 e 2 possiamo dedurre che $x_2 \leq x_5$;
- da 3 e 4 possiamo dedurre che $x_5 \leq x_2$

da cui $x_2 = x_5$. Invece, dalle disuguaglianze fornite sopra non è possibile dedurre che $x_0 = x_3$ sia sempre vera.

Il programma accetta come parametro sulla riga di comando il nome di un file di testo, la cui struttura è la seguente:

- la prima riga contiene due interi positivi n ed m, separati da spazi o tab, che indicano rispettivamente il numero di variabili e il numero di disuguaglianze;
- segue una riga vuota;
- seguono m righe, ciascuna contenente una coppia interi i, j separati da spazi o tab; ogni riga rappresenta una disuguaglianza $x_i \leq x_j$, con $0 \leq i, j < n$; può capitare che una o più variabili non compaiano in alcuna disuguaglianza, così come può capitare che la stessa disuguaglianza compaia più volte;
- segue una riga vuota
- seguono un certo numero di righe (ce ne sarà sempre almeno una) contenenti coppie di interi i, j separati da spazi o tab, che indicano uguaglianze tipo $x_i = x_j$. Per ciascuna di esse il programma stampa a video `true` se, dalle disuguaglianze indicate precedentemente, si può stabilire che l'uguaglianza è sicuramente vera, `false` altrimenti. Nota: uguaglianze del tipo $x_i = x_i$ vanno sempre considerate vere, a prescindere dalle disuguaglianze presenti. Le uguaglianze da verificare terminano quando si arriva alla fine del file di input.

Esempio.

Input:	*Output:*

`6 5`
 	
`0 3`
`2 3`
`3 5`
`5 4`
`4 2`
 	
`2 5`
`0 5`
`2 2`
	`true`
	`false`
	`true`