Progetti di Algoritmi e Strutture Dati

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

Anno Accademico 2021/2022, sessione estiva

Ultimo aggiornamento 09/05/2022

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 06/06/2022**. La prova orale potrà essere sostenuta in uno qualunque degli appelli della sessione estiva (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 gli esercizi del progetto 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

La società Real Time System (RTS) intende progettare e implementare un algoritmo per la gestione di dati provenienti da un sensore ambientale.

I dati prodotti dal sensore consistono in coppie <info, priorità>;

Tipo info: carattere;

Tipo priorità: un intero compreso nell'intervallo [1,100].

La produzione di coppie da parte del sensore può essere 'simulata' utilizzando opportunamente i generatori pseudocasuali di Java, (java.util.Random; come seme utilizzare il proprio numero di matricola per generare sia i valori interi nell'intervallo [1,100], sia i valori della variabile info)

Inizialmente il sensore produce **K** coppie che vengono memorizzate in una struttura dati **S**.

Il valore di **K** è calcolato nel seguente modo: **K** = 8 + ultima cifra del numero di matricola.

Le operazioni previste su **S**:

a) selezionare ed eliminare la coppia avente priorità più alta;

b) dopo aver eliminato la coppia di cui al punto a), richiedere al sensore una nuova coppia e inserire quest'ultima in **S**. Ovviamente i valori di questa nuova coppia potranno essere diversi da quelli della coppia appena eliminata;

Anche in questo caso, vedi punto b), la produzione di coppie da parte del sensore può essere 'simulata' utilizzando opportunamente i generatori pseudocasuali di Java, (come seme iniziale utilizzare il valore **3131123**)

RST richiede che:

1) la struttura **S** sia ad accesso diretto;

2) il massimo numero di accessi ad **S** per identificare la coppia avente priorità più alta sia pari ad 1;

3) l'inserimento di una coppia all'interno di **S** deve prevedere un numero di confronti tra elementi non superiore a 2* [log(**K**)]

Discutere brevemente la soluzione proposta considerando le specifiche (requisiti soddisfacibili?) di cui 1), 2) e 3).

Per verificare la correttezza dell'implementazione proposta, si richiede di effettuare alcuni test che prevedano la modifica dello stato della struttura **S** effettuando operazioni di cancellazione e inserimento. In particolare, il programma Java deve:

stampare lo stato iniziale della struttura dati S rappresentata dal valore di **K**, dal seme utilizzato per generare le coppie e dalle **K** coppie generate inizialmente;

supponendo di effettuare **K** operazioni di cancellazione/inserimento di nuovi elementi, per ognuna di tali operazioni il programma deve:

identificare ed eliminare da **S** l'elemento con priorità più alta;

generare una nuova coppia e inserirla in S; stampare la nuova coppia generata e stampare lo stato della struttura dati **S**;

Esempio:

Stato iniziale della struttura (in questo esempio **S** viene rappresentata come insieme di coppie)

K=9, seme = --------;

S = <q,90>, <r,90>, <l,77>, <w,59> , <a,40>, <b,25>, <a,15>, <a,12>, <s,12>

identificare ed eliminare da S l'elemento con priorità più alta: <q,90>.

Si noti che l'algoritmo avrebbe potuto eliminare invece che <q,90> la coppia <r,90>: dipende da come vengono inserite le coppie nella struttura **S.**

generare una nuova coppia e inserirla in S: **stampare la nuova coppia generata e stampare lo stato della struttura dati S**;

viene generata la coppia <w, 1>;

si inserisce in S la coppia appena generata:

S = <r,90>, <w,59>, <l,77>, <a,12>, <a,40>, <b,25>, <a,15>, <w,1>, <s,12>

Esercizio 2

La società Area51 gestisce un dizionario (D) di N elementi. Il singolo elemento di D è costituito da una tripla <chiave, info, hs>:

Tipo chiave: stringa;

Tipo info: carattere;

Tipo hs: intero nell'intervallo [0, K-1], con K intero positivo.

Il valore della variabile hs del generico elemento viene calcolato in fase di creazione di ogni singolo elemento del dizionario, ed è ottenuto utilizzando una funzione F, che ha come input il valore della chiave e produce come output un valore intero compreso nell'intervallo [0, K-1].

F non è iniettiva ma distribuisce uniformemente i valori, hs, associati alle chiavi in [0, K-1].

In base a quanto specificato NON è possibile che vi siano due elementi aventi la stessa chiave (e info) ma valore di hs diverso. Ovviamente lo stesso valore della variabile hs può essere associato a chiavi diverse.

Le operazioni previste su D:

a) ricerca di un elemento <chiave, info, hs> e stampa dello stesso se presente;

b) dato un valore di hs, stampare il numero di elementi aventi lo stesso valore per hs.

c) dato un valore di hs, stampare la lista degli elementi con lo stesso valore di hs.

Area51 richiede di progettare e implementare D in modo che, in base ai valori di N e K sia in grado di garantire che:

1)numero **medio** di accessi per a) sia $\Theta(1+ N/K)$;

2)Il numero di accessi per b) sia pari a 1

Discutere la soluzione proposta considerando le specifiche (requisiti soddisfacibili?) di cui 1), e 2).

Scrivere un programma Java che

1) legga da un file di input i valori di N e K e le triple per la creazione di D, come nell'esempio seguente:

10 4

Chiave1 a 1

Chiave2 b 2

Chiave3 c 1

Chiave4 c 0

Chiave5 a 3

Chiave6 x 3

Chiave7 y 2

Chiave8 a 0

Chiave9 q 1

Chiave10 s 1

2) a fronte di una richiesta, come specificato in a), b) e c), e opportunamente definita in termini di input, stampi a video il risultato della ricerca, ad esempio:

verifica presenza elemento: input <Chiave1, a, 1> ; output ‘elemento presente’

verifica presenza elemento: input <Chiave13, a, 1> output ‘elemento non presente’

ricerca in base al valore hs: input 0; output: 2

stampa lista in base al valore di hs: input 0 output < Chiave4, c, 0>, < Chiave8, a, 0>

Esercizio 3

Una compagnia per l'energia elettrica, tipo Enel Energia, Hera Comm, eDF, etc., vuole pianificare la costruzione di una nuova rete di distribuzione che collega tra loro *n* località. Disponiamo delle coordinate geografiche *x*[*i*], *y*[*i*] di ciascuna località, *i* = 0, … *n* - 1, espresse in Km usando un sistema cartesiano di riferimento. La compagnia può costruire delle linee elettriche che collegano in linea retta qualsiasi coppia di località. Il costo di una linea elettrica che collega le località *i* e *j* è proporzionale alla distanza d_{ij} tra di esse, calcolata con la consueta formula per la distanza euclidea:

$$d_{ij} = \sqrt{(x[i]-x[j])^2+(y[i]-y[j])^2}$$

La compagnia intende pianificare accuratamente i collegamenti in modo da coprire tutte le *n* località e allo stesso tempo minimizzare il costo complessivo di pianificazione.

Scrivere un programma Java che legga da un file di input il valore di *n* e le coordinate geografiche delle località, come nell'esempio seguente:

```
7           numero n di località presenti in questo file
1 1         x[0] y[0]
2 1
3 1
1.5 1.5
1 2
2 2
3 2         x[n - 1] y[n - 1]
```

che corrisponde alla situazione nella parte sinistra della figura:

Il programma deve stampare a video *n* - 1 righe di testo, ciascuna delle quali indica una delle *n* - 1 linee elettriche che devono essere costruite e la relativa lunghezza. Infine, deve stampare una ulteriore riga di testo indicando la lunghezza totale di tutti i collegamenti realizzati. Nell'esempio precedente, il programma puo' stampare:

```
0 3 0.7071067811865476
1 3 0.7071067811865476
3 4 0.7071067811865476
3 5 0.7071067811865476
1 2 1.0
2 6 1.0
4.82842712474619
```

che corrisponde all'albero mostrato nella figura a destra (l'elenco degli archi puo' essere dato in qualsiasi ordine). Nel caso esista più di una soluzione ottima è sufficiente visualizzarne una qualsiasi.

Esercizio 4

Consideriamo la seguente codifica dei caratteri `A, B, C, D, E, F, G, H`:

`A`	`0`
`B`	`00`
`C`	`001`
`D`	`010`
`E`	`0010`
`F`	`0100`
`G`	`0110`
`H`	`0001`

Osserviamo che questa codifica non gode della proprietà del codice di Huffman visto a lezione; il codice deve garantire che la decodifica di un messaggio sia univoca, e che nessun codice è un prefisso di un altro codice (ad esempio, il codice `00` di `B` è un prefisso del codice `001` di `C`); pertanto potrebbero esistere sequenze di bit che possono essere decodificate in modo diverso e quindi danno luogo ad ambiguità. Ad esempio, la sequenza `00100` può essere decodificata in 5 modi diversi, ossia come `ADA`, `AF`, `CAA`, `CB` oppure `EA`.
Lo scopo di questo esercizio è il seguente: data una stringa binaria *S*, determinare il **numero** di sequenze di caratteri che hanno la stessa codifica *S* utilizzando il codice nella tabella precedente; non è richiesto di stampare anche le sequenze di caratteri con la stessa codifica. Si presti attenzione al fatto che esistono stringhe che non possono essere decodificate in alcuna sequenza di caratteri (esempio: *S* = 1111); in tal caso l'algoritmo deve restituire il valore zero.

Il programma accetta sulla riga di comando un unico parametro che rappresenta il nome del file di input, che contiene (su una unica riga) la stringa *S* composta esclusivamente dai caratteri `0` e `1`:

```
00010010000100100001001000001001100
```

L'output è composto da un intero che rappresenta il numero di possibili decodifiche di *S*, utilizzando il codice indicato nella tabella precedente. Nel caso dell'esempio sopra, il programma stampa:

```
5567
```

Suggerimento: questo esercizio si risolve con la programmazione dinamica. I sottoproblemi corrispondono ai prefissi della stringa S...

Esercizio 5 – Vacanza in montagna

E arrivata l'estate e uno studente vuole fare una vacanza in montagna. Ci sono diverse tappe/punti di visita, che possiamo identificare con un numero intero (0, 1, 2, …, n-1), che lo studente vuole visitare. Lo studente ha in generale due alternative per andare da una tappa *u* ad un'altra *v*: passeggia a piedi o prende la teleferica. Questo problema può' essere modelizzato con un grafo orientato pesato particolare dove fra due nodi (o tappe) possono esistere più di un arco orientato di tipo diverso (nel nostro caso, al massimo due: andare a piedi o in teleferica).
Ciascun arco è etichettato con un tipo (*p,t*), che indica se l'arco rappresenta una passeggiata a piedi oppure un spostamento con la teleferica. Ad ogni arco (*u*, *v*) che rappresenta una passeggiata a piedi è associata la lunghezza *d*(*u*, *v*) in km. Ad ogni arco (*u*, *v*) che rappresenta uno spostamento con la teleferica sono associati la lunghezza *d*(*u*, *v*) in km e il costo di utilizzo della teleferica *c*(*u*, *v*) in euro. Tutte le lunghezze e i costi sono valori reali positivi.
Per ogni coppia di tappe/punti di visita (*u*, *v)* possono esistere zero, uno oppure due archi orientati che partono in *u* e terminano in *v*. Se è presente un solo arco, tale arco può rappresentare un tragitto a piedi oppure in teleferica, in base al tipo. Se sono presenti due archi, uno di essi rappresenta un tragitto a piedi e l'altro in teleferica. Non possono esistere due tragitti a piedi o due in teleferica che collegano direttamente la stessa coppia di nodi nella stessa direzione.

Il grafo è descritto in un file di testo, avente la struttura seguente. La prima riga contiene un intero *n* che indica il numero di tappe/nodi; i nodi sono identificati dagli interi 0, … *n* - 1. La seconda riga contiene un intero *m* che indica il numero totale di archi. Seguono *m* righe, ciascuna rappresentante un arco (*u*, *v*), composta dai seguenti campi separati da spazi o tab:

1. un singolo carattere che indica se si tratta di un tragitto a piedi (`p`) oppure in teleferica (`t`);
2. l'id del nodo di origine *u*;
3. l'id del nodo di destinazione *v*;
4. un valore reale positivo che indica la lunghezza *d*(*u*, *v*) del tratto da percorrere;
5. un valore reale positivo che indica il costo *c*(*u*, *v*) (solo per archi di tipo *t*).

Ad esempio, il file di testo dal contenuto seguente (le parole in corsivo sono solo commenti descrittivi e non fanno parte del file di input) descrive il grafo:

```
7                       Numero di nodi n (intero positivo)
13                      Numero di archi m (intero positivo)
p 0 1 10.2              tipo sorgente destinazione lunghezza
p 1 0 13.8
t 0 2 27.0 13           tipo sorgente destinazione lunghezza costo
t 1 0 9.2 10
p 1 2 18.0
p 1 4 32.1
p 1 3 11.7
p 4 2 8.2
t 2 5 12.2 18
p 4 5 31.1
p 5 6 21.1
p 3 6 82.3
t 3 6 54.1 22.5
```

E' richiesta la progettazione e realizzazione di un programma efficiente che accetta come unico parametro della riga di comando il nome di un file di input avente la struttura sopra indicata. Il programma deve quindi stampare:

1. il cammino di lunghezza minima che parte nel nodo 0 e termina nel nodo *n* - 1, e non faccia uso della teleferica ma solo di archi che rappresentano tratti a piedi ; se non esiste alcun cammino stampa `non raggiungibile`. Oltre al cammino, deve essere stampata anche la lunghezza complessiva del cammino minimo.
2. il cammino di lunghezza minima che parte nel nodo 0 e termina nel nodo *n* - 1, potendo usare sia archi "a piedi" che "in teleferica"; oltre alla lunghezza del cammino, il programma deve stampare il

costo totale dei tratti fatti usando la teleferica. Se non esiste alcun cammino dal nodo 0 al nodo *n* - 1, stampa `non raggiungibile`.

Il cammino va stampato elencando gli archi attraversati (ciascuno su una riga di output) nel formato:

tipo head_node tail_node

dove *tipo* è il carattere `p` (arco che rappresenta un tratto a piedi) oppure `t` (arco che rappresenta uno spostamento/tratto in teleferica) e *head_node, tail_node* sono gli indici dei nodi sorgente e destinazione dell'arco. Gli archi vanno elencati in ordine di attraversamento, a partire dall'arco uscente dal nodo iniziale (nodo 0). Di seguito alla lista di archi attraversati va stampata la lunghezza del cammino (e l'eventuale costo). Nell'esempio sopra il programma stamperà quanto segue:

```
p 0 1
p 1 4
p 4 5
p 5 6
94.5
t 0 2
t 2 5
p 5 6
60.3
31.0
```

L'output indica che:

- il cammino minimo per andare dal nodo 0 al nodo 6 senza usare la teleferica ha lunghezza 94.5; il cammino è composto dagli archi (0, 1) (1, 4) (4, 5) (5, 6), che rappresentano tutti tratti a piedi.
- il cammino minimo per andare dal nodo 0 al nodo 6 potendo usufruire anche della teleferica ha lunghezza 60.3 e comporta un costo di 31.0; il cammino è composto dagli archi (0, 2) (2, 5) (entrambi usando la teleferica) e dal tratto a piedi (5, 6). Si noti che la distanza e il costo che vengono stampati in questo caso non sono i valori esatti; ciò è normale (e non costituisce un errore) dato che internamente i numeri reali vengono rappresentati in forma approssimata come numeri in virgola mobile.