Progetti di Algoritmi e Strutture Dati

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

Anno Accademico 2023/2024, sessione autunnale

Istruzioni

Il progetto consiste in **tre 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 lunedì 02/09/2024**. La prova orale potrà 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 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 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 fuorvianti 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

```
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
```

Esempio di commento appropriato

```
// 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 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

Si realizzi un programma che legga il contenuto di due alberi, i cui nodi contengono valori interi univoci, da due distinti file che utilizzano due rappresentazioni differenti. I nomi dei due file sono specificati come parametri di riga di comando.
Il primo file utilizza coppie di valori, una per riga, ad indicare che il primo rappresenta un nodo padre e l'altro un suo nodo figlio; il carattere ',' è utilizzato per separare i due valori di ogni riga. Il secondo file utilizza una rappresentazione in forma di liste annidate. In questo formato ogni nodo e i suoi figli sono rappresentati come elementi di una lista nella quale il primo elemento è il valore del nodo e gli elementi successivi sono liste rappresentanti i suoi figli. I caratteri '[' e ']' vengono usati per delimitare una lista e gli elementi all'interno della lista sono separati dal carattere ','. In entrambi i formati eventuali spaziature sono considerate ininfluenti.
A fronte della lettura di questi due file il programma deve costruire delle strutture dati per memorizzare i relativi alberi e deve poi confrontarli per verificare se sono uguali. Attenzione che il criterio di uguaglianza da utilizzare deve ignorare l'ordine in cui i nodi figli sono organizzati. Si includa nel commento in testa al file una analisi del costo computazionale dell'algoritmo di comparazione utilizzato.

Esempio:

Dato un file `parent_child_pairs.txt` contenente:

1, 2

4, 9

1, 3

6, 13

2, 6

2, 7

4, 10

6, 12

3, 8

8, 14

9, 15

5, 11

1, 5

1, 4

E un file `nested_list.txt` contenente:

[1, [2, [6, [12], [13]], [7]], [3, [8, [14]]], [4, [9, [15]], [10]], [5, [11]]]

Il programma, eseguito con `java -cp . Esercizio1 parent_child_pairs.txt nested_list.txt` dovrà visualizzare:
`I due alberi sono uguali.`

Esercizio 2

A lezione è stato illustrato l'algoritmo di Huffman per determinare il codice "ottimo" per

codificare un dato messaggio. Uno dei problemi a cui prestare attenzione è che il codice deve garantire che la decodifica di un messaggio sia univoca; per questo è necessario assicurarsi che *nessun codice sia un prefisso di un altro codice*. Consideriamo la seguente codifica dei caratteri A, B, C, D, E, F, G, H:

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

Potete osservare che la codifica non gode della proprietà descritta sopra (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 e stampare **1) il numero** di sequenze di caratteri e **2) le sequenze di caratteri** che hanno la stessa codifica ***S*** utilizzando il codice nella tabella precedente.
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 di caratteri 0 e 1.

Se il file di input contiene la stringa 00100, il programma deve stampare a video un numero intero (5 in questo esempio), che rappresenta il numero di possibili decodifiche di S, utilizzando il codice indicato nella tabella precedente, e le 5 sequenze di caratteri seguenti :
5,
ADA,
AF,
CAA,
CB,
EA,

Se il file di input contiene la stringa seguente:
00100010011000
il programma deve stampare a video “24” e tutte le possibili sequenze di caratteri che non sono elencate qui per mancanza di spazio.

Se il file contiene la stringa 10000, …, o la stringa 1000011000000110
il programma deve stampare a video 0.

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

Esercizio 3

La pianta stradale di una città è rappresentata da un grafo orientato G=(V,E) composto da n≥ 2 nodi etichettati come {0, 1, ...n-1}, che rappresentano incroci stradali, e m≥1 archi che rappresentano strade che collegano coppie di incroci. Ogni arco è etichettato con un valore reale positivo, che indica il tempo (in secondi) necessario per percorrere la strada corrispondente. Su ciascuno degli ***n*** incroci è posizionato un semaforo: prima di poter imboccare una qualsiasi delle strade che escono dall'incrocio (archi in uscita dal nodo corrispondente) è necessario aspettare che il semaforo diventi verde.
È data una funzione ***double attesa(int i, double t)***, che accetta come parametri l'identificativo di un nodo i (intero compreso tra 0 e n-1), e l'istante di tempo ***t*** in cui si arriva all'incrocio. La funzione restituisce un valore reale ≥0, che indica il tempo di attesa prima che il semaforo diventi verde. Quindi, supponendo di arrivare al nodo ***i*** al tempo ***T***, sarà possibile imboccare uno degli archi in uscita all'istante ***T+attesa(i,T)***.

Implementare un programma per calcolare il tempo minimo necessario per andare dal nodo 0 al nodo n-1, se possibile, supponendo di trovarsi al nodo 0 all'istante ***t=0***. Si presti attenzione al fatto che, da quanto detto sopra, si può imboccare una delle strade uscenti dal nodo 0 all' istante attesa(0, 0). Il programma accetta un unico parametro sulla riga di comando, che rappresenta un file contenente la descrizione della rete stradale.

Un esempio è il seguente:

5			numero di nodi n
7			numero di archi m
0	1	132.3	origine, destinazione e tempo di percorrenza arco 0
1	2	12.8	
0	3	23.81	
3	2	42.0	
2	4	18.33	
3	4	362.92	
3	1	75.9	origine, destinazione e tempo di percorrenza arco m-1

Se il nodo n-1 non è raggiungibile dal nodo 0, il programma stampa non raggiungibile e termina. In caso contrario il programma stampa due righe: la prima contiene un numero reale che indica il tempo minimo necessario per arrivare al nodo n-1 partendo dal nodo 0 all'istante 0; il tempo deve ovviamente includere le soste ai semafori. La seconda riga contiene la sequenza di nodi visitati, nell'ordine in cui vengono attraversati.

A questo punto consideriamo 2 casi:
Caso 1: una ipotetica implementazione della funzione ***attesa()*** che restituisce sempre il valore 5.0 (costante)

Caso 2: la funzione ***attesa()*** restituisce un numero pseudo random con l'utilizzo della classe Random con inizializzazione con un **SEED** (**numero di matricola**) per avere la garanzia di ottenere gli stessi risultati dopo ogni esecuzione del programma.

Caso 1: l'algoritmo stamperà a video:
99.14
0 3 2 4

Caso 2: l'algoritmo stamperà a video (con SEED = 10000):
84.71
0 3 2 4