Progetti di Algoritmi e Strutture Dati—moduli 2 - 3

Anno Accademico 2015/2016

Dott. Ivan Lanese, Dott. Moreno Marzolla

Versione 1.0, 11 maggio 2016: prima versione.

Istruzioni

I progetti dei moduli 2-3 consistono in quattro esercizi di programmazione da realizzare in Java. Lo svolgimento del progetto non è obbligatorio; i progetti possono essere consegnati anche da coloro che non hanno consegnato (o non hanno superato) i progetti del modulo 1. In caso di superamento di uno dei due progetti (modulo 1 oppure moduli 2-3), sarà possibile recuperare la parte mancante durante uno qualsiasi degli appelli scritti svolgendo la parte di compito relativa al modulo non superato. Non verranno assegnati altri progetti durante il corrente Anno Accademico, né saranno possibili ulteriori consegne oltre le scadenze indicate nel seguito di questo documento.

L'esito della valutazione dei progetti dipende dalla correttezza ed efficienza dei programmi consegnati, e dal risultato della discussione orale durante la quale verrà verificata la conoscenza della teoria, e verrà anche verificato che il progetto sia stato effettivamente svolto dall'autore (in passato si sono verificati casi assai sgradevoli di persone che si sono fatte fare il progetto da altri). **Una discussione gravemente insufficiente comporterà il non superamento della prova**.

Modalità di svolgimento dei progetti

I progetti devono essere svolti **individualmente**: la similitudine tra progetti potrà essere verificata anche utilizzando strumenti automatici e, se confermata, comporterà l'immediato annullamento della prova per TUTTI gli studenti coinvolti. **E' vietato discutere le soluzioni dei progetti con gli altri studenti**; è pero' consentito (anzi, incoraggiato) confrontare l'output prodotto dalla propria implementazione con quella di altri, possibilmente su input diversi da quelli forniti, per verificare la correttezza dei risultati.

E' consentito l'uso di algoritmi e strutture dati definite nella libreria standard Java. **Non è consentito fare uso di altro codice, anche se liberamente disponibile in rete**.

Il codice deve poter essere compilato ed eseguito sulle macchine Linux del laboratorio studenti (Java 7). I programmi devono essere realizzati come applicazioni a riga di comando, senza alcun tipo di interfaccia grafica.

Ciascun esercizio deve essere implementato in un UNICO 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. Tutti 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). Uno schema di sorgente Java è presente all'interno dell'archivio con i dati del progetto sulla pagina Web del corso.

I sorgenti verranno compilati con il comando:

```
javac EsercizioN.java
```

ed eseguiti con il comando

```
java -cp . EsercizioN <eventuali parametri>
```

Si può assumere che i dati di input siano sempre corretti. Sulla pagina web del corso verranno forniti alcuni file di input per i vari esercizi. I programmi consegnati devono ovviamente funzionare correttamente su qualsiasi input; a tale scopo verranno testati anche con dati diversi da quelli forniti.

Per verificare la correttezza in modo (semi)automatico, i programmi devono produrre output a video rispettando il formato indicato nelle specifiche. Si noti che alcuni problemi potrebbero ammettere più

soluzioni corrette; in questi casi il programma può restituirne una qualsiasi, anche se diversa da quella mostrata nel testo dell'esercizio 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 differenze verranno ignorate. Siete pregati di segnalare eventuali errori significativi negli output associati ai file di input forniti, che potrebbero indicare errori da parte nostra.

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). Tutto questo dovrebbe essere noto, dal corso di Laboratorio di Programmazione Internet.

I docenti sono a disposizione per fornire chiarimenti sulle specifiche dei progetti, ma non per fare debug del vostro codice!

Alcune buone pratiche di programmazione

Usare nomi appropriati per variabili, classi e metodi. L'uso di nomi inappropriati rende il codice difficile da comprendere e da valutare). Per rendersi conto di ciò, chi direbbe che questo algoritmo (espresso in pseudocodice):

```
integer cancella( double trovato, double tentativo[1..n], integer terzo, integer quarto )
        integer box ← (terzo + quarto) / 2;
        if ( tentativo[box] == trovato ) then
                return box;
        elseif ( tentativo[box] > trovato ) then
                return cancella( trovato, tentativo, terzo, box – 1);
        else
                return cancella( trovato, tentativo, box + 1, quarto );
        endif
```

in realtà non cancella nulla ma implementa la ricerca binaria?

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.

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

Usare strutture dati adeguate. In questi progetti è consentito l'utilizzo di qualsiasi struttura dati o algoritmo già implementato nella JDK. Decidere quale struttura dati e algoritmo risultano più efficienti per risolvere un determinato problema è tra gli obbiettivi di questo corso, e pertanto avrà un impatto significativo sulla valutazione.

Scadenza e modalità di consegna

I progetti vanno consegnati in un archivio in formato .zip (altri formati non saranno accettati). L'archivio deve essere denominato con il nome e cognome dell'autore (es., `MarcoRossi.zip`), e deve contenere una directory con lo stesso nome (`MarcoRossi/`) contenente a sua volta tutti i sorgenti. Ad esempio, l'archivio potrebbe essere scompattato nei file:

```
MarcoRossi/Esercizio1.java
MarcoRossi/Esercizio2.java
MarcoRossi/Esercizio3.java
MarcoRossi/Esercizio4.java
```

L'archivio può contenere eventuali altri file, purché di dimensioni contenute. **Nota per gli utenti Apple**: gli archivi .zip prodotti sotto MacOSX spesso includono una directory `.DS_Store/` contenente metadati specifici per MacOSX. L'archivio che viene consegnato deve essere privo di tale directory.

La consegna deve avvenire entro le **12:00 (mezzogiorno) di venerdì 10 giugno 2016 (prima consegna)** oppure entro le **12:00 (mezzogiorno) di venerdì 1 luglio 2016 (seconda consegna)** inviando una mail dal proprio indirizzo istituzionale a Tong Liu (t.liu@unibo.it) con oggetto

Consegna Progetto ASD 2015-2016

allegando l'archivio .zip. Nella mail vanno indicati il proprio cognome, nome, e numero di matricola. Riceverete una mail di conferma (questo potrà richiedere alcuni giorni). Ciascuno studente puo' consegnare una sola volta.

Valutazione dei progetti

A partire da **lunedì 20 giugno 2016** per la prima consegna e da **lunedì 11 luglio 2016** per la seconda consegna, gli studenti ammessi all'orale verranno convocati per discutere i progetti, secondo un calendario che verrà comunicato sulla pagina web del corso. La discussione includerà anche domande sulla parte teorica svolta a lezione. **Una discussione gravemente insufficiente comporterà il non superamento della prova**.

La valutazione dei progetti sarà determinata dai parametri seguenti (i primi sono particolarmente importanti):

- Correttezza dei programmi implementati;
- Efficienza dei programmi implementati;
- Chiarezza del codice; codice poco comprensibile, ridondante o inefficiente comporterà penalizzazioni, indipendentemente dalla sua correttezza.
- Capacità dell'autore 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 sorgenti sono compilabili sulle macchine Linux del laboratorio studenti?
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. L'archivio con i sorgenti spedito ai docenti è in formato .zip, denominato `NomeCognome.zip` (ove NomeCognome deve essere sostituito con il nome e cognome dell'autore)?
5. I programmi consegnati producono l'output atteso sui file di input presenti sulla pagina Web del corso?

Esercizio 1.

La pianta stradale di una città è rappresentata da un grafo orientato $G = (V, E)$ composto da $n \geq 2$ nodi etichettati come $\{0, 1, \ldots n - 1\}$, che rappresentano incroci stradali, e $m \geq 1$ 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. E' 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.
Ad esempio, considerando una ipotetica implementazione della funzione `attesa()` che restituisce sempre il valore 5.0 (costante), l'algoritmo applicato all'esempio precedente stamperà a video:

```
99.14
0 3 2 4
```

L'archivio dei test presente sulla pagina web del corso conterrà una implementazione del metodo `attesa()` che deve essere utilizzata per verificare gli output inclusi nell'archivio. Si presti attenzione che l'algoritmo implementato deve comunque funzionare correttamente con qualsiasi implementazione della funzione `attesa()`, naturalmente producendo risultati diversi da quelli forniti.

Esercizio 2.

Disponiamo di $n \geq 1$ scatole a forma di parallelepipedo, etichettate come 0, 1, … n - 1; la scatola i ha larghezza $x[i]$, altezza $y[i]$ e profondità $z[i]$. Vogliamo inserire il maggior numero di scatole una dentro l'altra, in modo simile ad una "matrioska", nell'ipotesi in cui sia possibile inserire la scatola i dentro la scatola j se le dimensioni della scatola i sono strettamente minori di quelle della scatola j (ossia $x[i] < x[j]$, $y[i] < y[j]$ e $z[i] < z[j]$). Le scatole non possono essere ruotate. Non c'è nessun altro vincolo sulle scatole che possono essere inserite l'una nell'altra; in particolare, le scatole possono essere considerate in ordine qualsiasi.
Scrivere un algoritmo efficiente che, date in input le dimensioni delle scatole, determina quali scatole è possibile inserire l'una nell'altra in modo da ottenere il numero massimo di "inscatolamenti" possibile, rispettando il vincolo sulla dimensione dei lati di cui sopra. Nel caso esistano più soluzioni ottime, è sufficiente stamparne una qualunque; come caso particolare, se nessuna scatola puo' essere inserita in un'altra, è possibile stampare i dati di una scatola qualsiasi.

Il programma accetta sulla riga di comando un unico parametro che rappresenta il nome del file di input, il cui contenuto ha la struttura seguente:

```
10                          numero n di scatole
10  6.9 12.1                x[0] y[0] z[0]
2   2   3
3   2.5 3
4.4 5   6
7   5.1 7.4
9   8.7 5.6
8.7 6.5 9.5
2.5 6.5 7.3
5.7 8.7 9.8
7.6 5.1 6.2                 x[n-1] y[n-1] z[n-1]
```

L'output deve elencare le scatole che fanno parte della soluzione ottima, a cominciare da quella più esterna; per ciascuna di esse devono essere riportate le dimensioni (come da file di input). Ad esempio, nel caso del file precedente il programma deve produrre il seguente output:

```
scatola 0: 10.0 6.9 12.1
scatola 6: 8.7 6.5 9.5
scatola 4: 7.0 5.1 7.4
scatola 3: 4.4 5.0 6.0
scatola 1: 2.0 2.0 3.0
```

Suggerimento: una possibilità (non l'unica) per risolvere questo problema consiste nel costruire un grafo orientato non pesato, in cui ogni nodo corrisponda ad una scatola, ed esiste un arco orientato (i, j) se e solo se la scatola j puo' contenere la scatola i. A questo punto si deve individuare un cammino di lunghezza massima (anziché minima) sul grafo.

Esercizio 3.

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

Osserviamo 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` puo' essere decodificata in 5 modi diversi, ossia come `ADA`, `AF`, `CAA`, `CB` oppure `EA`.
Lo scopo di questo progetto 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 di caratteri `0` e `1`:

```
00100010010010010001100010000010001010000101000000010010100
```

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:

```
153408
```

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

Esercizio 4.

Un risparmiatore deve decidere come investire una certa somma di denaro (capitale) per T mesi, etichettati come $\{0, 1, \ldots, T-1\}$, in modo da massimizzare il guadagno finale. All'inizio del mese t ($0 \leq t < T$) il risparmiatore puo' scegliere se:

1. Investire tutto il capitale in obbligazioni Alfa S.p.A.; tali obbligazioni devono essere conservate per due mesi, e produrranno un guadagno di $A[t]$ euro al termine del mese $t + 1$;
2. Investire tutto il capitale in obbligazioni Beta S.p.A.; tali obbligazioni devono essere conservate per tre mesi, e produrranno un guadagno di $B[t]$ euro al termine del mese $t + 2$;
3. Investire tutto il capitale in titoli di stato della Cappadocia; tali titoli devono essere conservati per sei mesi, e produrranno un guadagno di $C[t]$ euro al termine del mese $t + 5$;
4. Non investire il capitale per il mese corrente; in questo caso, al termine del mese t il guadagno sarà pari a zero.

Non è possibile incassare anticipatamente il capitale,. Questo significa che l'ultimo investimento (o non-investimento, nel caso 4) deve tassativamente scadere al termine del mese $T - 1$. Si presti inoltre attenzione al fatto che i rendimenti potrebbero anche essere negativi (e quindi produrre una perdita alla scadenza). Non è consentito di reinvestire gli eventuali guadagni, ma solo e sempre il capitale iniziale (anche se nel frattempo si è ottenuta una perdita). Il rendimento complessivo al termine del periodo di T mesi è dato dalla somma dei rendimenti di tutti gli investimenti che sono stati effettuati.

Ad esempio, consideriamo un periodo di 6 mesi con i rendimenti indicati nella tabella seguente:

	$A[t]$	$B[t]$	$C[t]$
$t = 0$	3.5	12.1	12.3
$t = 1$	15.8	28.9	33.1
$t = 2$	19.3	11.3	8.8
$t = 3$	-1.5	2.4	-1.1
$t = 4$	13.1	1.2	3.1
$t = 5$	8.3	11.7	21.0

Nell'ipotesi in cui il risparmiatore decida di acquistare obbligazioni Alfa per i primi due mesi, obbligazioni Beta per i successivi tre e conservare il capitale non investito per l'ultimo mese, il rendimento sarà pari a:

$$3.5 + 11.3 + 0 = 16.8 \text{ Euro}$$

Se il cliente decide di acquistare esclusivamente obbligazioni Alfa all'inizio dei mesi $t = 0$, $t = 2$ e $t = 4$, il rendimento sarà

$$3.5 + 19.3 + 13.1 = 35.9 \text{ Euro}$$

Nell'ipotesi in cui il cliente decida di acquistare titoli della Cappadocia all'inizio del mese $t = 0$, il rendimento finale sarà 12.3 Euro. Naturalmente esistono anche altre possibilità di investimento. Si noti che non sarebbe possibile acquistare, ad esempio, obbligazioni Beta all'inizio del mese $t = 4$, in quanto scadrebbero oltre il periodo di investimento.

Scopo del progetto è la progettazione e implementazione di un algoritmo efficiente che, dati in input la durata complessiva T e gli array dei rendimenti $A[0..T-1]$, $B[0..T-1]$ e $C[0..T-1]$, determini la sequenza di investimenti che al termine del mese $T - 1$ massimizzi il guadagno complessivo.

Il programma accetta come unico parametro sulla riga di comando il nome di un file di testo contenente i dati di input. La struttura del file è descritta nell'esempio seguente:

```
6                               Numero di mesi T
3.5     12.1    12.3            A[0], B[0], C[0]
15.8    28.9    33.1            A[1], B[1], C[1]
19.3    11.3    8.8
```

```
-1.5    2.4     -1.1
13.1    1.2     3.1
8.3     11.7    21.0            A[T-1], B[T-1], C[T-1]
```

Il programma deve stampare la sequenza di investimenti da effettuare per ottenere il massimo rendimento totale, seguita dal rendimento stesso. Nel caso precedente, il programma stamperà

```
K
B
A
42.0
```

dove `K` indica che per un mese si tiene il capitale non investito, `A`, `B`, `C` indicano l'acquisto di azioni Alfa, Beta e titoli della Cappadocia, rispettivamente. Nel nostro caso, l'investimento che fornisce il rendimento ottimo si ottiene mantenendo il capitale non investito per il primo mese, acquistando poi obbligazioni Beta da conservare per tre mesi, e infine obbligazioni Alfa da conservare per i restanti due mesi.