Progetto Totale n. 2 di Laboratorio di Algoritmi e Strutture Dati

Anno Accademico 2018/19

Corso di Laurea in Informatica

Università di Camerino

Prof. Luca Tesei

Obiettivi del Progetto

1. Realizzare in Java due diverse implementazioni di una coda con priorità generica utilizzando in un caso uno heap binario e nell'altro caso uno heap di Fibonacci.
2. Realizzare in Java una implementazione di un grafo generico orientato con liste di adiacenza.
3. Realizzare in Java una implementazione dell'algoritmo di Dijkstra per i cammini minimi con sorgente singola che usi sia una coda con priorità non ottima sia le implementazioni delle code con priorità realizzate.
4. Realizzare in Java una implementazione dell'algoritmo di Bellman-Ford per i cammini minimi con sorgente singola.
5. Realizzare una suite di test JUnit 4 che controlli le funzionalità implementate.
6. Valutare numericamente e confrontare le prestazioni delle due implementazioni della coda con priorità.
7. Scrivere una relazione sul progetto e fare una presentazione orale dello stesso utilizzando delle slides e il codice scritto.

Realizzazione in Gruppo

Questo progetto può essere realizzato singolarmente o in un gruppo di massimo 3 studenti.

Descrizione

Heap e code con priorità

Una coda con priorità è una struttura dati che colleziona un insieme di elementi a cui è associata una priorità che può variare nel tempo. Essenzialmente la coda deve permettere le seguenti operazioni:

- inserire un elemento, con associata una priorità;
- aggiornare la priorità di un elemento già presente;
- restituire l'elemento in testa, che corrisponde sempre all'elemento con il valore corrente della priorità minimo o massimo (la coda può essere min o max);

Operazione	**Heap Binario**	**Heap di Fibonacci**
Inserimento di un elemento	$\mathcal{O}(\log n)$	$\Theta(1)$
Ricerca del Min/Max	$\Theta(1)$	$\Theta(1)$
Estrazione del Min/Max	$\Theta(\log n)$	$\mathcal{O}(\log n)$
Decremento/Incremento di priorità	$\mathcal{O}(\log n)$	$\Theta(1)$

Tabella 1: Costo asintotico delle operazioni di base su uno heap binario e su uno heap di Fibonacci, riferito a strutture che contengono n elementi.

- estrarre dalla coda l'elemento in testa, che corrisponde sempre all'elemento con il valore corrente della priorità minimo o massimo (la coda può essere min o max).

. Una coda con priorità può essere realizzata in maniera molto semplice tramite un array, una list o un set in cui vengono inseriti gli elementi in coda o in testa e ogni volta che si deve cercare l'elemento con priorità minima o massima si scorrono tutti gli elementi. Stessa cosa quando la priorità di un elemento deve essere aggiornata: si scorrono gli elementi fino a trovare quello che ci interessa e si aggiorna il campo priorità. In questo modo le operazioni (tranne l'inserimento) sono tutte di costo lineare nel numero degli elementi della coda.

Tuttavià è possibile fare di meglio. Uno heap binario, visto a lezione e spiegato nel Capitolo 6 di [1], permette di realizzare tutte le operazioni di una coda con priorità con i costi asintotici indicati in Tabella 1. Uno heap di Fibonacci, spiegato nel Capitolo 19 di [1], permette di fare ancora meglio, migliorando, rispetto allo heap binario, il costo delle operazioni di inserimento e aggiornamento della priorità (si veda sempre la Tabella 1).

Cammini minimi su grafi con sorgente singola

Dato un grafo orientato e pesato, un classico problema algoritmico che si può definire è quello del calcolo dei cammini minimi a partire da un certo nodo, detto sorgente, verso tutti gli altri nodi del grafo. In ogni cammino minimo la somma dei pesi associati agli archi attraversati dal cammino è quella più piccola possibile considerando tutte le possibili strade che possono essere intraprese tramite gli archi orientati del grafo.

Esistono diversi algoritmi che risolvono il problema, si veda il Capitolo 24 di [1]. In particolare, il classico algoritmo di Dijkstra opera su grafi diretti con archi pesati e pesi non negativi. L'algoritmo di Dijkstra fa uso di una coda con priorità per inserire i nodi durante la visita del grafo. La complessità di estrazione del minimo e di aggiornamento della priorità nella coda influenza la complessità totale dell'algoritmo.

L'algoritmo di Bellman-Ford accetta grafi orientati con archi pesati anche con pesi negativi. Oltre a calcolare i cammini minimi, durante l'esecuzione determina se ci sono cicli di peso negativo nel grafo raggiungibili dal nodo sorgente. In quest'ultimo caso termina segnalando il problema: infatti, se ci sono cicli con peso negativo, la definizione di cammino minimo cade (percorrendo il ciclo con peso negativo il numero di volte necessario si può scendere sotto qualsiasi valore minimo prefissato).

Realizzazione

Per realizzare il progetto devono essere completati i seguenti task.

Importazione del codice della traccia

Scaricare il progetto maven/eclipse fornito nel wiki (`http://didattica.cs.unicam.it/doku.php?id=didattica:triennale:asd:ay_1819:lab`) e importarlo in Eclipse (o altro IDE di programmazione) tramite le opportune funzioni di importazione per i progetti Maven.

Studio delle strutture dati e degli algoritmi

Come prima cosa bisogna acquisire le conoscenze necessarie per realizzare le strutture dati e le operazioni sugli gli heap binari e gli heap di Fibonacci. Poi bisogna anche comprendere come operano gli algoritmi di Dijkstra e di Bellman-Ford.

Scrittura del codice

Implementare le parti mancanti delle classi fornite, in particolare:

- `BinaryHeapMinPriorityQueue`, che implementa l'interfaccia fornita `MinPriorityQueue` in cui sono specificate le API dei metodi;
- `FibonacciHeapMinPriorityQueue`, che implementa `MinPriorityQueue`;
- `AdjacentListDirectedGraph`, che implementa l'interfaccia fornita `Graph` in cui sono specificate le API dei metodi. Le interfacce `GraphNode` e `GraphEdge` contengono le API per generici nodi e archi e le classi `DefaultGraphNode` e `DefaultGraphEdge` sono implementazioni di default per le due interfacce corrispondenti. Nella classe `AdjacentListDirectedGraph` i metodi che dipendono da indici associati ai nodi non devono essere implementati;
- `SimpleDijkstraShortestPathComputer`, che implementa l'interfaccia `SingleSourceShortestPathComputer` in cui sono specificate le API dei metodi. Si noti che i nodi del grafo devono essere del tipo generico `PriorityGraphNode` che implementa l'interfaccia `PriorityQueueElement` e che eredita tutte le funzionalità di `GraphNode`. La classe di default che implementa `PriorityGraphNode` è `DefaultPriorityGraphNode`, fornita, in cui viene fatta coincidere la priorità associata al nodo con la distanza floating point associata al nodo. Nella classe `SimpleDijkstraShortestPathComputer` l'algoritmo di Dijkstra usa una coda con priorità implementata con una semplice list (o set o array) che ha costo di estrazione dell'elemento con priorità minima lineare;
- `PriorityQueueDijkstraShortestPathComputer`, che implementa l'interfaccia `SingleSourceShortestPathComputer`. In questa classe l'algoritmo di Dijkstra usa una coda con priorità implementata con uno heap binario o con uno heap di Fibonacci (si usino le classi `BinaryHeapMinPriorityQueue` o `FibonacciHeapMinPriorityQueue`), il che aumenta le prestazioni dell'algoritmo;
- `BellmanFordShortestPathComputer`, che implementa l'interfaccia `SingleSourceShortestPathComputer`. In questa classe deve essere implementato l'algoritmo di Bellman-Ford per il calcolo dei cammini minimi, quindi sono accettati grafi orientati anche con pesi negativi sugli archi. Si noti comunque che quando l'algoritmo trova che esiste un ciclo di peso negativo deve lanciare una eccezione (si vedano le API).

Ogni punto da completare è segnalato da un tag `\\ TODO`. E' possibile aggiungere variabili istanza e metodi pubblici o privati, ma non è possibile cancellare o modificare le variabili istanza e i metodi che sono specificati.

Fase di test

Implementare uno o più test JUnit 4 per tutti i metodi implementati. In particolare, se un metodo ha più valori di ritorno possibili oppure può lanciare eccezioni, un test apposito deve essere previsto per ogni caso. I test vanno implementati nelle classi `<NomeClasse>Test` che sono state già create con lo stub del codice per ogni metodo. In caso di più test per lo stesso metodo cambiare i nomi. Se il test controlla il lancio di una eccezione va inserita una condizione come segue:

```
@Test(timeout = 10000, expected = IllegalArgumentException.class)
    public final void testMethod1() {
```

```
        // Codice che dovrebbe far lanciare l'eccezione IllegalArgumentException
    }
```

Il test avrà successo se verrà lanciata l'eccezione indicata, fallirà altrimenti. Il tag `timeout = 10000` serve a far fallire il test se il tempo di esecuzione eccede i 10000 millisecondi, cioè 10 secondi. In questo modo se un metodo va in ciclo il test fallisce in un tempo ragionevole.

Se il test controlla che il metodo calcoli valori corretti, allora dovrà essere strutturato come segue:

```
@Test(timeout = 10000)
    public final void testMethod1() {
        // Codice che prepara i valori per chiamare il metodo da testare
        assertEquals(15, obj.method1());
```

La clausola `assertEquals(valore atteso, valore calcolato)` fallisce se il valore atteso e il valore calcolato non sono uguali. Nel caso i valori siano oggetti è preferibile usare il metodo equals e testare il risultato di quest'ultimo: `assertEquals(true, obj.method1().equals(testValueObj))`. Se non ci sono eccezioni e nessuna clausola `assertEquals` fallisce allora il test avrà successo.

Valutazione numerica delle prestazioni

Eseguire il framework di valutazione numerica delle code con priorità (fornito nel codice) acquisendo i dati per la coda con priorità realizzata con heap binari e per quella realizzata con heap di Fibonacci. I dati prodotti dal framework di valutazione (file .csv, comma-separated values `https://it.wikipedia.org/wiki/Comma-separated_values`) dovranno essere elaborati con un foglio elettronico (ad esempio Microsoft Excel o OpenOffice) o con un framework che permette l'uso di tabelle di dati e calcoli statistici (ad esempio R, MatLab o Mathematica) per calcolare, per valori di lunghezza n crescenti delle sequenze di elementi delle code generati casualmente, i seguenti valori:

- il minimo (caso ottimo);
- il massimo (caso pessimo);
- la media aritmetica (caso medio);
- la deviazione standard (caso medio)

per le seguenti grandezze:

- tempo in nanosecondi di inserimento di n elementi in una coda vuota;
- tempo in nanosecondi di decremento della priorità di un elemento in una coda con n elementi;
- tempo in nanosecondi di estrazione del minimo da una coda con n elementi;

per entrambe le classi `BinaryHeapMinPriorityQueue` e`FibonacciHeapMinPriorityQueue`.

I valori elaborati dovranno poi essere usati per creare dei grafici che permettano di valutare e comparare visivamente le prestazioni dei due tipi di code nelle varie operazioni valutate. Tale valutazione e comparazione dovrà essere riportata, insieme ai grafici prodotti, nella relazione scritta del progetto. In particolare i grafici delle prestazioni nei vari casi dovranno essere comparati con il grafico della funzione $f(n) = \log n$.

Per i casi medi i grafici devono riportare la linea spezzata che, per valori crescenti di n, corrisponde al tempo medio di esecuzione e due linee spezzate che corrispondono al valore medio più la deviazione standard e al valore medio meno la deviazione standard.

Per una corretta esecuzione del framework di valutazione si devono evitare tutte le possibili interferenze di altri processi in esecuzione nel computer che si sta utilizzando. Quindi si consiglia di:

- chiudere tutte le applicazioni e tutti i servizi del sistema operativo non necessari;
- staccare la rete;

- eseguire la classe main del framework da riga di comando invece che dall'interno di Eclipse. Per far questo, salvare tutto e chiudere Eclipse. Poi aprire una finestra terminale (dipende dal sistema operativo, ad esempio su Windows bisogna eseguire `cmd.exe` oppure `Prompt dei comandi`). Una volta aperto il terminale scrivere:

  ```
  > cd <path del workspace>/project2/target/classes + INVIO
  ```

  sostituendo a `<path del workspace>` il percorso della cartella workspace di Eclipse (oppure il percorso della cartella dove si trova il progetto se non è nel workspace di Eclipse) e poi scrivere:

  ```
  > java -cp . it.unicam.cs.asdl1819.project2.MinPriorityQueueEvaluationFramework + INVIO
  ```

Scrittura della relazione

La relazione scritta deve illustrare le parti salienti del progetto, in particolare le principali soluzioni implementative e i risultati del framework di valutazione. Nella relazione possono essere riportati e commentati dei pezzi di codice che sono significativi, ad esempio il core dell'implementazione dei vari algoritmi per i cammini minimi e delle operazioni principali delle code con priorità.

La relazione può essere scritta con un editor di testi WYSIWYG (What You See Is What You Get) come Microsoft Word o OpenOffice oppure con il sistema LaTeX. In quest'ultimo caso un mini corso è disponibile qui: `http://didattica.cs.unicam.it/doku.php?id=informazioni:documentiutile:main`.

La relazione deve essere fornita come file PDF.

Consegna

Va compilato il file `DACOMPILARE.txt` presente nella cartella principale del progetto, inserendo i dati richiesti. Ogni studente ha un proprio spazio Google Drive corrispondente all'indirizzo istituzionale:

`nome.cognome@studenti.unicam.it`

(o una variante di questo schema). Il primo componente del gruppo (in ordine alfabetico per cognome) dovrà creare nel suo spazio Google Drive una cartella che si chiami esattamente così:

`ASDL1819-NOME-COGNOME-PT2`

ad esempio `ASDL1819-MARIO-ROSSI-PT2`.

All'interno della cartella Google Drive creata va caricata la cartella `project2` (con esattamente questo nome) contenente il progetto implementato con la stessa struttura di file e cartelle fornita con la traccia. Si tratta di caricare direttamente così com'è la cartella `project2` all'interno del workspace di Eclipse (o nella posizione in cui si trova se non è nel workspace). Tutti i file devono essere caricati, compresi quelli di progetto di Eclipse (o altro IDE) e quelli di Maven.

I risultati del framework di valutazione, cioè i file .csv generati e i file (Microsof Excel o altro) usati per l'elaborazione devono essere caricati nella cartella

`project2/src/main/evaluationFrameworkData`

La relazione in formato PDF deve essere caricata nella cartella

`project2/src/main/RelazioneScritta`

Una volta caricati tutti i file, il primo componente del gruppo (in ordine alfabetico per cognome) deve condividere la cartella `ASDL1819-NOME-COGNOME-PT2` e tutti i file che contiene con la possibilità di

modificare (flag "Può Modificare" in fase di condivisione) con `luca.tesei@unicam.it` e con tutti gli altri componenti del gruppo (usando l'email istituzionale unicam).

La data entro cui consegnare il progetto (entro le 23.59 di quel giorno) è fissata per ogni appello in ESSE3 (Prova Parziale con nome "XXX Consegna Progetto Tot Lab aa 2018-19"). Farà fede la data di caricamento dei file su Google Drive. Entro *il giorno prima della data di scadenza* (scadenza iscrizioni in ESSE3) **tutti gli studenti del gruppo** devono registrarsi su ESSE3 alla prova parziale corrispondente.

Il giorno successivo alla consegna sarà fissato il calendario degli orali nei giorni successivi. Tutti i componenti del gruppo saranno convocati all'orale tramite un invito di Google Calendar sulla loro email istituzionale unicam. In caso di problemi per la data e l'ora indicate contattare subito il docente via email.

Preparazione della presentazione orale

La prova orale consiste, nella fase iniziale, in una presentazione del progetto da parte di tutti i componenti del gruppo. Questa presentazione dovrà essere supportata da slides e dall'esposizione del codice implementato. Può essere realizzata anche una piccola demo.

Tutti i componenti del gruppo dovranno esporre la presentazione, dividendosi il tempo necessario, che comunque non può eccedere i **20 minuti**.

La presentazione deve esporre le fasi salienti del progetto e i risultati ottenuti, permettendo di avere un panoramica completa del lavoro svolto.

Le slides della presentazione devono essere sintetiche e possibilmente contenere immagini o parole chiave per punti. Non fare delle slides che contengono frasi lunghe; le slide devono essere comprese a colpo d'occhio.

Valutazione

La valutazione della prova di Laboratorio di Algoritmi si baserà sui seguenti criteri:

- Qualità della realizzazione delle strutture dati e degli algoritmi e sua esposizione nella relazione scritta e nella presentazione orale.
- Contenuto e qualità della relazione scritta e dei grafici in essa contenuti riguardanti i risultati del framework di valutazione.
- Qualità e successo dei test JUnit implementati.
- Codice chiaro e ben commentato.
- Qualità delle slides, della presentazione orale e delle risposte alle domande.

Durante la presentazione orale o successivamente ci saranno domande specifiche ai componenti del gruppo. Il voto finale consiste di una parte comune riguardante il progetto realizzato e di una parte individuale riguardante la prova orale di ogni componente del gruppo. Alla fine ogni componente del gruppo riceverà il suo voto in trentesimi.

Riferimenti bibliografici

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduzione agli algoritmi 3/ED*. McGraw-Hill, 2010.