Esercitazione 8
Grafi

Corso di Fondamenti di Informatica II

BIAR2 (Ing. Informatica e Automatica) e BSIR2 (Ing. dei Sistemi)

A.A. 2013/2014

06 Dicembre 2013

Sommario

Scopo di questa esercitazione è visitare un grafo in ampiezza e calcolare i cammini minimi di un grafo pesato.

1 Visita in ampiezza di grafi

La visita in ampiezza (*breadth-first search* o BFS) visita i nodi di un grafo per livelli. Basandosi sul materiale di supporto, si vuole aggiungere alla classe `GraphUtil` un metodo statico per la visita in ampiezza di un grafo.

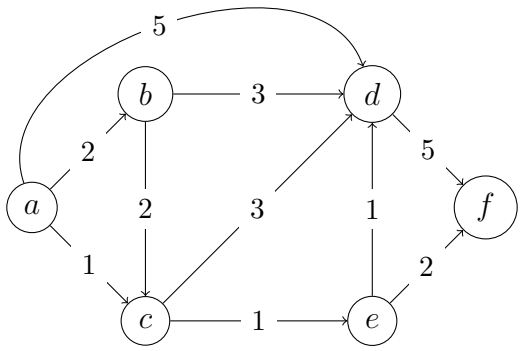
Materiale di supporto. Vengono fornite le classi `Graph<V>` che rappresenta un grafo semplice orientato, a pesi interi. `V` è il tipo di dato usato per etichettare i nodi del grafo ed `Integer` il tipo di dato usato per i pesi degli archi del grafo. Sono incluse le classi `Node<V>` ed `Edge<V>` per rappresentare nodi ed archi. Viene inoltre fornita la classe `GraphUtil` che contiene un `main` di prova, un metodo di stampa, e un metodo che realizza la visita in profondità `dfs(Graph<V> g, Node<V> source)`.

Programma Java. Realizzare il metodo statico

```
public static <V> void bfs(Graph<V> g, Node<V> source)
```

che stampa la sequenza dei nodi ottenuti visitando il grafo mediante *Breadth-First Search* a partire dal nodo `source`.

Suggerimento. Così come il metodo `dfs` utilizza uno `Stack` (o in alternativa la ricorsione), per implementare `bfs` si utilizzi una struttura `Queue` (ad esempio, `ArrayDeque`) per mantenere i nodi da visitare, e una struttura `Set`, come ad esempio `HashSet`, per mantenere i nodi già inseriti in coda.



	distanza da source (a)					
step (nodo)	a	b	c	d	e	f
0 (-)	0 (-)	∞	∞	∞	∞	∞
1 (a)	0 (-)	2 (a)	1 (a)	5 (a)	∞	∞
2 (c)	0 (-)	2 (a)	1 (a)	4 (a, c)	2 (a, c)	∞
3 (b)	0 (-)	2 (a)	1 (a)	4 (a, c)	2 (a, c)	∞
4 (e)	0 (-)	2 (a)	1 (a)	3 (a, c, e)	2 (a, c)	4 (a, c, e)
5 (d)	0 (-)	2 (a)	1 (a)	3 (a, c, e)	2 (a, c)	4 (a, c, e)
6 (f)	0 (-)	2 (a)	1 (a)	3 (a, c, e)	2 (a, c)	4 (a, c, e)

Figura 1: Esempio di esecuzione dell'algoritmo di Dijkstra con source il nodo a.

2 Cammini minimi

L'algoritmo di Dijkstra risolve il problema *single-source shortest path* per grafi pesati se tutti i pesi degli archi sono maggiori o uguali a zero. In Fig. 1 è riportato un esempio di esecuzione.

Programma Java. Aggiungere un metodo statico alla classe `GraphUtil`

```
public static void <V> sssp(Graph<V> g, Node<V> source)
```

che stampa la distanza (lunghezza del cammino minimo) dal nodo `source` a ogni altro nodo. Nel caso di Fig. 1, dato `source`=a, il metodo deve stampare (in ordine qualsiasi):

```
a 0
b 2
c 1
d 3
e 2
f 4
```

Materiale di supporto. Viene fornita la classe parametrica `MinHeap` già realizzata in una precedente esercitazione. Si consiglia di utilizzare una struttura di

tipo `MinHeap<Node<V>>` per mantenere in una coda di priorità i nodi da visitare nell'algoritmo, associandovi come priorità la loro distanza stimata.