Esercitazione 7
DFS

Corso di Fondamenti di Informatica II

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

A.A. 2013/2014

29 Novembre 2013

Sommario

Scopo di questa esercitazione è di realizzare una struttura dati per gestire un grafo diretto e implementare l'algoritmo di DFS.

1 Rappresentazione di un grafo diretto

Un grafo è un TDA che contiene un insieme V di valori, chiamati nodi, e un insieme E di coppie di nodi, chiamati archi. Se il grafo è non diretto, la generica coppia $\langle$u, v$\rangle \in E$ rappresenta un arco che può essere percorso in entrambe le direzioni, da u a v e viceversa. Se il grafo è diretto, $\langle$u, v$\rangle$ rappresenta un arco diretto, che può essere percorso solo da u a v. In questa esercitazione assumeremo che il grafo sia diretto, e che i valori dei nodi siano di un tipo generico `V`.

Utilizzeremo una implementazione basata su lista di adiacenze. In questa rappresentazione, il grafo memorizza una `HashMap<Node<V>, List<Node<V>>>` che contiene, per ogni nodo del grafo (di tipo `Node<V>`) una lista di riferimenti ai nodi adiacenti. Dei nodi adiacenti, per implementare un grafo diretto, ci interessano solo quelli collegati da un arco uscente.

Specifiche Scrivere una classe Java parametrica `Graph<V>` con la seguente interfaccia. Si consiglia di implementare i metodi nell'ordine indicato.

```
public Graph()
```

Costruisce un grafo senza nodi.

```
public Collection<Node<V>> getNodes()
```

Restituisce i nodi del grafo.

```
public Collection<Node<V>> getOutEdges(Node<V> source)
```

Rstituisce la lista di adiacenza (nodi collegati da archi uscenti) di un nodo

```
public void insertNode(Node<V> v)
```

Aggiunge un nuovo nodo al grafo

```
public void insertEdge(Node<V> src, Node<V> u) throws Exception
```

Aggiunge un nuovo arco al grafo. Se uno dei due nodi non è presente nel grafo, lancia un'eccezione.

```
public void dfs()
```

Esegue una dfs sul grafo `this`. La funzione, usa come metodo ausiliario `sweep`.

```
private int sweep(Node<V> source,
                  HashMap<Node<V>, Status> statovisite,
                  HashMap<Node<V>, Integer> tempivisite,
                  int tempocorrente)
```

Esegue una dfs a partire dal nodo `source` stampando, per ogni arco che visita, il relativo tipo. Si vedano i dettagli nel seguito

Si proceda a testare `Graph<V>` attraverso il driver `Test.java` fornito nella cartella dell'esercitazione e la classe ausiliaria `GraphUtil<V>`.

2 La funzione `sweep`

Il metodo `sweep` consiste nel visitare, dato un nodo `source` del grafo, tutti in nodi "raggiungibili" da `source`. Più precisamente, `sweep` esegue una DFS sul sottografo G_s raggiungibile da `source`. In funzione dell'esecuzione del metodo, inoltre, si possono dividere gli archi di G_s in 4 tipi, in modo che:

- gli archi *tree* formino un albero
- gli archi *back* colleghino un nodo ad un suo antenato nell'albero
- gli archi *forward* colleghino un nodo ad un suo discendente nell'albero
- gli archi *cross* colleghino nodi senza relazione antenato-discendente

La Figura 1 mostra un esempio con valori di tipo intero. Qui il nodo `source` è il nodo `1` e l'ordine della visita è `1 2 3 4 5 6 7 8`.

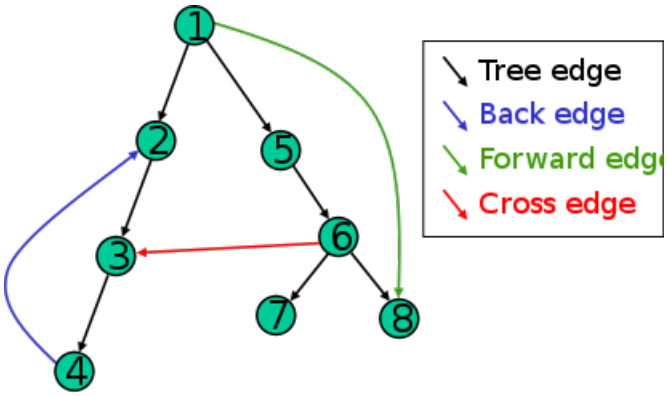

Figura 1: I 4 tipi di archi definiti dall'esecuzione di `sweep`.

Per far sì che `sweep` stampi, durante l'esecuzione, di che tipo è il prossimo arco da percorrere, si può fare come segue. Sia `u` (un nodo nella lista di adiacenza di `source`) il prossimo nodo da visitare (su cui chiamare ricorsivamente `sweep`):

- Se `u` non è mai stato visitato, l'arco da `source` a `u` è un arco *tree*
- Se `u` è ancora "in visita", l'arco da `source` a `u` è un arco *back*
- Se `u` è stato visitato
  - dopo `source`, l'arco da `source` a `u` è un arco *forward*
  - prima di `source`, l'arco da `source` a `u` è un arco *cross*

Si sfruttino i parametri `HashMap<Node<V>, Status> statovisite` , `HashMap<Node<V>, Integer> tempivisite` e `int tempocorrente` per implementare il meccanismo.