Esercitazione 5
DFS

Corso di Fondamenti di Informatica II

Algoritmi e strutture dati

A.A. 2016/2017

23 Maggio 2017

Sommario

Scopo di questa esercitazione è realizzare una struttura dati per rappresentare un grafo ed implementare una visita DFS.

Per svolgere l'esercitazione e' necessario scaricare i file ausialiari `c-aux.tar.gz` e `java-aux.tar.gz`.

Attenzione: L'esercitazione puo' essere svolta sia in linguaggio C che in Java. E' altamente consigliato di sviluppare le soluzioni in laboratorio usando il linguaggio C e di svolgere l'esercitazione in Java come esercizio per casa. Inoltre e' altamente consigliato di far girare le soluzioni in C sotto `valgrind`.

1 Rappresentazione di grafo

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 (`UNDIRECTED`), la generica coppia (u, v) $\in$ E rappresenta un arco che può essere percorso in entrambe le direzioni, da u a v e viceversa. Se il grafo è diretto (`DIRECTED`), (u, v) rappresenta un arco diretto, che può essere percorso solo da u a v. In questo esercizio, occorre implementare una struttura dati per rappresentare un grafo di entrambi i tipi (`UNDIRECTED` o `DIRECTED`). Ogni nodo ha associato un valore di tipo generico (se in C, il valore è un `void *`)

Per mantenere traccia degli archi in un grafo, sono possibili diverse implementazioni. In questa esercitazione, è consigliato di utilizzare liste di adiacenza: ogni nodo mantiene una lista dei suoi nodi vicini. Per chi sviluppa l'esercizio in C, il codice ausiliario fornisce una semplice implementazione di una linked list. Per chi sviluppa l'esercizio in Java, si può far uso di una qualunque implementazione di una struttura collegata fornita dal Java Collection Framework (ad esempio, `LinkedList`).

Specifiche. Scrivere un modulo C con intestazione `graph.h` (rispettivamente una classe generica Java `Graph.java`) contenente le seguenti funzioni (metodi di classe se Java):

- una funzione che construisce un grafo vuoto:

```
graph * graph_new(GRAPH_TYPE type);

public Graph(GRAPH_TYPE type);
```

  Dove `type` specifica il tipo di grafo (`UNDIRECTED` o `DIRECTED`).

- una funzione che restituisce i nodi presenti nel grafo:

```
linked_list * graph_get_nodes(graph * g);

public List<GraphNode<V>> getNodes();
```

  La lista collegata che viene ritornata contiene i nodi del grafo (se C `graph_node`, se Java `GraphNode`.

- una funzione che restituisce la lista dei nodi vicini di un nodo:

```
linked_list * graph_get_neighbors(graph * g, graph_node * n);

public List<GraphNode<V>> getNeighbors(GraphNode<V> n);
```

- una funzione che aggiunge un nodo al grafo:

```
graph_node * graph_add_node(graph * g, void * value);

public GraphNode<V> addNode(V value);
```

  che ritorna il nodo del grafo creato.

- una funzione che aggiunge un arco al grafo:

```
void graph_add_edge(graph * g, graph_node * from, graph_node * to);

public void addEdge(GraphNode<V> from, GraphNode<V> to);
```

  Se il grafo è di tipo `UNDIRECTED`, la funzione deve inserire un arco navigabile in entrambi i versi.

- una funzione che ritorna il valore associato ad un nodo:

```
void * graph_get_node_value(graph * g, graph_node * n);

public V getNodeValue(GraphNode<V> n);
```

- (solo in C) una funzione che elimina il grafo:

```
void graph_delete(graph * h);
```

Si proceda a testare il codice sviluppato utilizzando `driver.c` (rispettivamente `Driver.java`) passando come argomento `graph`.

2 Grafo non diretto: percorso tra due nodi

In questo esercizio, occorre sviluppare un algoritmo, basato su una visita DFS di un grafo non diretto, che ritorni un percorso esistente nel grafo tra due nodi (source, target).

Specifiche. Estendere il codice sviluppato nel precedente esercizio aggiungendo la funzione (metodo):

```
linked_list * graph_get_path(graph * g, graph_node * s, graph_node * t);

public List<V> getPath(GraphNode<V> s, GraphNode<V> t);
```

che restituisce la lista dei valori memorizzati nei nodi lungo il percorso (da `s` fino ad arrivare a `t`) che è stato identificato dalla funzione nel grafo.

Si proceda a testare il codice sviluppato utilizzando `driver.c` (rispettivamente `Driver.java`) passando come argomento `path`.

3 Grafo diretto: classificazione degli archi

Dato un grafo diretto G, l'esercizio richiede di sviluppare un algoritmo (`sweep`) che visita tutti i nodi effettuando una DFS. Durante la visita degli archi, tale funzione deve ogni arco (u,v) in base a queste tipologie:

1. se v è visitato per la prima volta quando attraversiamo l'arco (u, v), allora l'arco (u,v) è un *tree* edge.

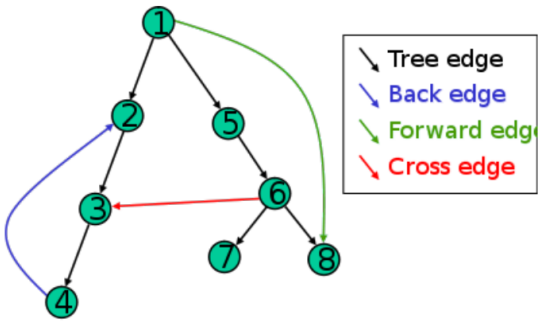

Figura 1: Esempio di esecuzione della funzione `sweep`. Visita DFS del grafo indotto dal nodo `1` con classificazione degli archi.

2. altrimenti:
    - se v è un antenato di u, allora l'arco (u, v) è un *back* edge.
    - se v è un discendente di u, allora l'arco (u, v) è un *forward* edge.
    - se v non è né antenato né discendente di u, allora l'arco (u, v) è un *cross* edge

Figura 1 mostra un esempio con valori di tipo intero. La DFS viene fatta iniziare dal nodo `1` e l'ordine della visita è `1 2 3 4 5 6 7 8`.

Per far sı che `sweep` stampi, durante l'esecuzione, la tipologia di un arco, si può fare come segue. Sia (u, v) un arco del grafo:

- Se v non è mai stato visitato, l'arco da u a v è un tree edge
- Se v è ancora "in visita", l'arco da u a v è un back edge
- Se v è stato visitato:
    - dopo u, l'arco (u,v) è un forward edge
    - prima di u, l'arco (u,v) è un cross edge

Specifiche. Estendere il codice presente in `graph.c` (`Graph.java`) implementando la funzione (metodo):

```
void graph_sweep(graph * g, char * format_string);

public void sweep();
```

In C, `format_string` può essere utilizzata dalla funzione per stampare il valore memorizzato in un nodo (ad esempio, `printf(format_string, node_value)`). Tale funzione visita il grafo, effettuando una DFS su ogni nodo del grafo che non è stato ancora visitato, e stampa in `stdout` il tipo di arco {tree, back, cross, forward} per ogni arco presente nel grafo. Alla fine dell'esecuzioni tutti gli archi devono essere stati classificati. Si ricorda che il grafo potrebbe non essere completamente connesso.

Si proceda a testare il codice sviluppato utilizzando `driver.c` (rispettivamente `Driver.java`) passando come argomento `sweep`.

Riferimenti bibliografici

[1] M. T. Goodrich and R. Tamassia. *Strutture dati e algoritmi in Java*. Zanichelli, 2007.