Esercitazione 7
Grafi Diretti e DFS

Corso di Fondamenti di Informatica II

Algoritmi e strutture dati

A.A. 2017/2018

18 Maggio 2018

Sommario

Scopo di questa esercitazione è sperimentare gli utilizzi della visita DFS sui grafi diretti.

Per svolgere l'esercitazione è necessario collegarsi al sito:
`www.sites.google.com/diag.uniroma1.it/crocefederico/materiale`
e scaricare i file contenuti nella cartella dedicata a questa esercitazione.

Attenzione: L'esercitazione può 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 è altamente consigliato di far girare le soluzioni in C sotto `valgrind`.

1 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 classificare 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.
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

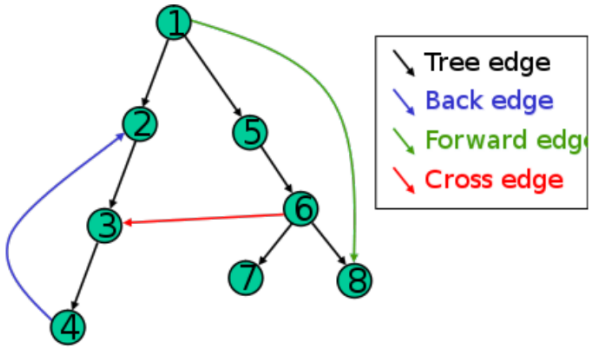

Figura 1: Esempio di esecuzione della funzione `sweep`. Visita DFS del grafo indotto dal nodo `1` con classificazione degli archi.

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. Realizzare la funzione `sweep` descritta nel header `graph_services.h` (`GraphServices.java`) che, preso in ingresso un grafo, stampa a video la classificazione di tutti i suoi archi nel formato:

```
v1 -> v2 : {TREE, FORWARD, BACK, CROSS}
```

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

```java
public static <V> void sweep(Graph<V> g);
```

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'esecuzione 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`.

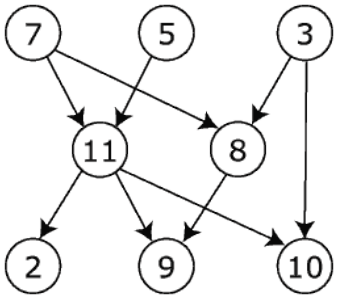

Figura 2: Esempio di grafo diretto aciclico.

2 Ordine Topologico

Dato un grafo diretto aciclico (DAG), un ordine topologico del grafo è una lista ordinata dei nodi tale che ogni nodo viene prima di tutti quelli collegati ai suoi archi uscenti. Si noti che, nel caso peggiore, esistono $n!$ diversi ordinamenti. In questa esercitazione siamo interessati a risolvere il problema dell'ordinamento topologico facendo uso di una visita DFS. La Fig.2 mostra un'esempio di un grafo diretto aciclico i cui diversi ordinamenti topologici sono (lista non completa):

- 7, 5, 3, 11, 8, 2, 9, 10;
- 3, 5, 7, 8, 11, 2, 9, 10;
- 3, 7, 8, 5, 11, 10, 2, 9;
- ...

Specifiche. Realizzare la funzione `topological_sort` descritta nel header `graph_services.h` (`GraphServices.java`) che, preso in ingresso un grafo, stampa a video un possibile ordinamento topologico dello stesso. Si ricorda che, nel caso il grafo passato alla funzione non sia diretto ed aciclico, non è possibile derivare alcun ordinamento topologico.

```
void topological_sort(graph *g);

public static <V> void topologicalSort(Graph<V> g);
```

3 Componenti fortemente connesse

Un grafo diretto si dice fortemente connesso se esiste un percorso per ogni coppia dei suoi vertici. Una componente fortemente connessa di un grafo diretto, è un sottografo (diretto) fortemente connesso con il maggior numero possibile di vertici. In questa esercitazione, siamo interessati ad identificare tutte le componenti fortemente connesse di un grafo diretto, facendo uso della visita DFS. La Fig.3 mostra un esempio di grafo diretto, con evidenziate le sue componenti fortemente connesse.

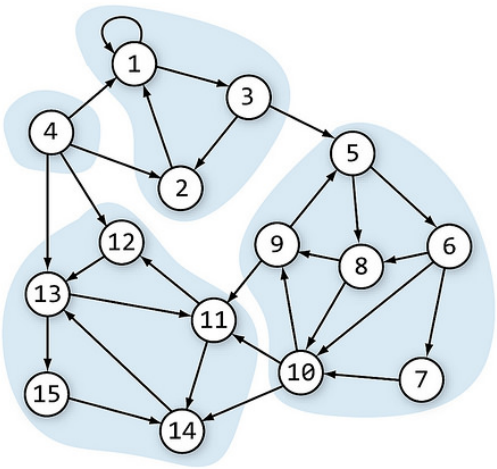

Figura 3: Esempio di grafo diretto aciclico con evidenziate le sue componenti fortemente connesse.

Specifiche. Realizzare la funzione `strong_connected_components` descritta nel header `graph_services.h` (`GraphServices.java`) che, preso in ingresso un grafo, stampa a video tutte le componenti fortemente connesse del grafo nel formato:

```
Sottografo fortemente connesso
V1, V2, V3, ..., Vn
.
.
Sottografo fortemente connesso
Vk, Vk+1, ..., Vm
```

```
void strong_connected_components(graph *g);

public static <V> void strongConnectedComponents(Graph<V> g);
```

Suggerimento: le componenti fortemente connesse di un grafo diretto, e quelle del suo grafo trasposto (cioè il grafo in cui ogni arco $V_m \to V_n$ è sostituito con l'arco $V_n \to V_m$) sono uguali.

Riferimenti bibliografici

[1] M. T. Goodrich, R. Tamassia and M. H. Goldwasser. *Algoritmi e strutture dati in Java*. Apogeo Education - Maggioli Editore, 2015.