

# Fondamenti di Informatica II

## Esercitazione del 19 maggio 2020 - canale I

Scopo di questa esercitazione è implementare le operazioni di single-source shortest path e minimum spanning tree.

**Attenzione.** Per realizzare questa esercitazione è necessario implementare funzioni (metodi) contenuti nel file `graph_services.c` ( `GraphServices.java` ). A tale scopo viene fornito materiale di supporto che realizza funzionalità di base utili alla soluzione dei quesiti posti (si vedano i paragrafi successivi per maggiori dettagli).

### Task 1. Classificazione degli archi

Dato un grafo diretto  $G$ , l'esercizio richiede di sviluppare un algoritmo (BFS) che faccia una visita in ampiezza dei nodi di  $G$ . L'output della procedura sarà una stampa a video della sequenza dei nodi visitati, ordinata secondo l'ordine di visita. Ad esempio, invocando la procedura sul grafo rappresentato in Fig. 1 si può ottenere la stampa:

```
1
2
3
4
```

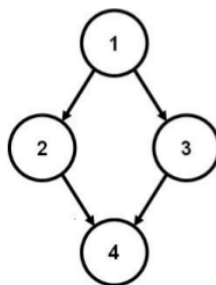


Figura 1: Un esempio di grafo diretto.

**Materiale di supporto.** Il modulo `graph.h` ( `Graph<V>` contenuto in `Graph.java` ) che rappresenta un grafo semplice orientato, a pesi interi. `V` è il tipo di dato usato per etichettare i nodi del grafo. Il grafo è rappresentato mediante lista di incidenze: ogni nodo ha associata una lista di tutti i suoi **archi** (esplicitati nel tipo `Edge<V>` ).

**Specifiche.** Realizzare la funzione `bfs` descritta nello header `graph_services.h` (`GraphServices.java`) che, preso in ingresso un grafo, stampa a video l'ordine di accesso dei nodi del grafo secondo una visita in ampiezza.

```
void bfs ( graph * g );  
  
public static <V> void bfs(Graph <V> g);
```

Si ricorda che il grafo potrebbe non essere fortemente connesso.

**Suggerimento.** Si consiglia di utilizzare una struttura **Queue** per gestire l'ordine di visita dei nodi del grafo. Si proceda a testare il codice sviluppato utilizzando `driver.c` oppure `Driver.java`.

## Task 2. Single Source Shortest Path

L'algoritmo di Dijkstra risolve il problema *single source shortest path* per grafi pesati, se tutti i pesi degli archi sono positivi. Si chiede di realizzare una funzione (`sssp`) che, preso in ingresso un grafo ed un nodo *sorgente*, stampi a video la distanza minima dal nodo indicato a tutti gli altri nodi del grafo. Un nodo non raggiungibile da quello *sorgente* può essere indicato con distanza *infinita*. Ad esempio, invocando la procedura sul grafo in Fig. 2 avendo *S* come nodo *sorgente*, si ottiene la stampa:

```
S 0  
A 1  
C 2  
B 7  
D 5  
E 6
```

*L'ordine non è importante.*

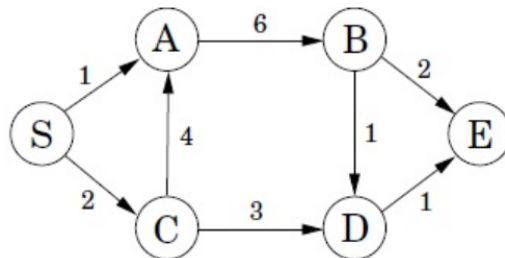


Figura 2: Un grafo diretto con gli archi a pesi positivi.

**Materiale di supporto.** viene fornita l'implementazione di un **MinHeap** per la gestione di una coda con priorità, sia nel linguaggio C che in Java.

**Specifiche.** Realizzare la funzione con segnatura indicata di seguito, descritta nello header

`graph_services.h` ( `GraphServices.java` ) che, preso in ingresso un grafo ed un nodo, stampa a video le distanze di tutti i nodi del grafo da quello dato in input, secondo la formattazione dell'esempio.

```
void single_source_shortest_path(graph * g, graph_node * source);  
  
public static void <V> sssp(Graph <V> g, Node <V> source);
```

**Suggerimento.** È possibile tenere traccia della distanza di ciascun nodo dal nodo `source` dato in input alla funzione, facendo uso del campo `int dist`, presente in ogni nodo del grafo. In Java, è possibile fare uso del Java Collection Framework per ottenere un risultato simile.

Si proceda a testare il codice sviluppato utilizzando `driver.c` oppure `Driver.java`.

### Task 3. Minimum Spanning Tree

Il problema del minimum spanning tree consiste nel calcolare, per un dato grafo pesato, un sottografo aciclico a peso minimo che ricopra tutti i nodi del grafo originale.

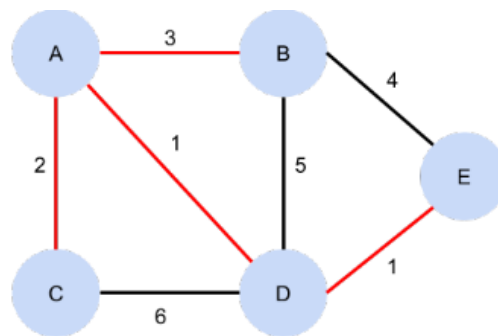


Figura 3: Un grafo diretto con gli archi a pesati. In rosso viene indicato il suo minimum spanning tree.

L'algoritmo di Kruskal risolve questo problema facendo uso delle primitive di **Union – Find**. Si faccia riferimento alle slide del corso per i dettagli su tali primitive e sulle possibili implementazioni ed ottimizzazioni. Si chiede di realizzare una funzione ( `mst` ) che, preso in ingresso un grafo con gli archi pesati  $G$ , stampi a video il sottoinsieme degli archi di  $G$  che costituiscono un minimum spanning tree del grafo. L'ordine degli archi stampati non è rilevante. Ad esempio, invocando la procedura sul grafo in Fig. 3 si ottiene la stampa:

```
a b  
a c  
a d  
d e
```

*L'ordine non è importante.*

**Materiale di supporto.** viene fornita l'implementazione delle primitive **Union – Find** per operare su insiemi disgiunti. Si noti che tale implementazione gestisce insiemi di numeri. E' possibile mappare ogni nodo del grafo ad un numero facendo uso del campo `int map` presente in ogni nodo del grafo.

**Specifiche.** Realizzare la funzione `mst` descritta nel header `graph_services.h` ( `GraphServices.java` ) che, preso in ingresso un grafo, stampa a video un sottoinsieme degli archi del grafo che compongono un *minimum spanning tree*.

```
void mst(graph * g);  
  
public static void <V> mst(Graph <V> g);
```

**Suggerimento.** si consiglia di utilizzare una coda con priorità per rappresentare gli archi del grafo da scegliere.

Si proceda a testare il codice sviluppato utilizzando `driver.c` oppure `Driver.java` .