Corso di Algoritmi e strutture dati
A.A. 2017/2018

Progetto

Regole per lo svolgimento:

- Il progetto deve essere svolto **singolarmente** o **in gruppi di 2 persone**.
- Il progetto deve essere consegnato **almeno 7 giorni prima** della prova orale di Algoritmi e strutture dati in cui si discute il progetto.
- Il progetto deve contenere le classi, i campi e i metodi richiesti **rispettandone esattamente il nome, il tipo e l'ordine dei parametri formali, ed il tipo di ritorno**.
- Si è naturalmente liberi di sviluppare metodi aggiuntivi di supporto ai metodi richiesti, laddove lo si ritenga utile.
- Deve essere consegnato via email all'indirizzo **luca.moscardelli@unich.it** un file compresso contenente tutti i file sorgenti e avente come nome
  [ASD] progetto 2017_2018 Cognome1 Cognome2.zip
  l'email deve avere come oggetto
  [ASD] consegna progetto 2017-2018 Cognome1 Cognome2
  dove Cognome1 e Cognome2 sono i cognomi degli studenti che consegnano; chiaramente Cognome2 va omesso se la consegna è effettuata da uno studente singolo.
  In caso di consegna in gruppo la consegna deve essere effettuata da uno dei componenti del gruppo, mettendo in CC nell'email anche l'indirizzo dell'altro componente.

Testo del progetto:

Sviluppare in **Java** un package **algoritmiGrafi** per l'implementazione di grafi e dei seguenti algoritmi:

- Visita in ampiezza
- Visita in profondità
- Algoritmo di Kruskal
- Algoritmo di Prim
- Algoritmo di Dijkstra
- Algoritmo di Bellman-Ford
- Algoritmo di Floyd-Warshall

Il package deve contenere le seguenti classi pubbliche:

1. `Grafo`

   Questa classe gestisce un grafo diretto o non diretto (sia pesato sugli archi che non pesato). Deve prevedere (anche) i metodi seguenti:

   - Costruttore che prende in input un file di testo contenente la rappresentazione di un grafo come descritto nel seguito.
   - `public boolean isUnweighted()` che restituisce true se e solo se il grafo è non pesato (tutti i pesi degli archi sono uguali ad 1)
   - `public boolean hasNegativeEdges()` che restituisce true se e solo se il grafo contiene almeno un arco di peso negativo
   - `public boolean isSymmetric()` che restituisce true se e solo se il grafo è non diretto (per ogni arco dal nodo i al nodo j c'è anche l'arco da j a i)
   - `public void toFile(…)` che crea un file con il nome opportunamente passato nei parametri, contenente la rappresentazione del grafo come descritto nel seguito.
   - `public Grafo bFS(int sorgente)` che restituisce il grafo (l'albero in particolare) risultante dalla visita in ampiezza effettuata a partire dal nodo indicato.
   - `public Grafo dFS(int sorgente)` che restituisce il grafo (l'albero in particolare) risultante dalla visita in profondità effettuata a partire dal nodo indicato.
   - `public Grafo dFS()` che restituisce il grafo (la foresta di alberi) risultante dalla visita in profondità effettuata a partire da nodi arbitrari.

2. `AlgKruskal`
3. `AlgPrim`
4. `AlgDijkstra (astratta)`
5. `AlgDijkstraList extends AlgDijkstra`
6. `AlgDijkstraHeap extends AlgDijkstra`
7. `AlgBellmanFord`
8. `AlgFloydWarshall`

   Queste classi si preoccupano di implementare i vari algoritmi a cui si riferiscono. Si noti che la class AlgDijkstra è astratta perché deve prevedere due implementazioni diverse. Devono comprendere almeno i seguenti metodi:

   - Costruttore che prende in input un Grafo
   - Metodo `esegui` che esegue l'algoritmo e memorizza il o i grafi in output in opportune variabili di istanza
   - Metodo `Grafo getOutput` che restituisce l'output calcolato. Se l'output dipende anche da un modo sorgente, questo è preso come parametro in ingresso dal metodo getOutput.

   Per ognuna di queste classi devono essere presenti nella classe Grafo dei metodi opportuni che si occupano di costruire l'oggetto corrispondente e di eseguire l'algoritmo e restituire l'output (ad esempio `public Grafo algDijkstra (int source)`)

Deve essere infine sviluppata **una classe MainClass con il metodo main** a cui deve essere passato da linea di comando un file contenente un grafo in input. Il metodo deve creare il grafo corrispondente e eseguire tutti gli algoritmi, producendo in output i seguenti file di testo:

- |V| files per la visita in ampiezza: `<file_input>_out_bfs_<n>.txt` dove <file_input> è il nome del grafo preso in input e <n> varia da 0 |V|-1, e rappresenta il nodo sorgente per l'esecuzione dell'algoritmo della visita in ampiezza.
- |V| files per la visita in profondità: `<file_input>_out_dfs_<n>.txt` dove <file_input> è il nome del grafo preso in input e <n> varia da 0 al numero dei nodi del grafo meno uno, e rappresenta il nodo sorgente per l'esecuzione dell'algoritmo della visita in profondità.
- 1 file per la visita in profondità con foresta: `<file_input>_out_dfs.txt` dove <file_input> è il nome del grafo preso in input.
- 1 file per l'algoritmo di Kruskal: `<file_input>_out_kruskal.txt` dove <file_input> è il nome del grafo preso in input.
- |V| files per l'algoritmo di Prim: `<file_input>_out_prim_<n>.txt` dove <file_input> è il nome del grafo preso in input e <n> varia da 0 al numero dei nodi del grafo meno uno, e rappresenta il nodo sorgente per l'esecuzione dell'algoritmo di Prim.
- |V| files per l'algoritmo di Dijkstra implementato con liste: `<file_input>_out_dijkstra_liste_<n>.txt` dove <file_input> è il nome del grafo preso in input e <n> varia da 0 al numero dei nodi del grafo meno uno, e rappresenta il nodo sorgente per l'esecuzione dell'algoritmo di Dijkstra.
- |V| files per l'algoritmo di Dijkstra implementato con heap: `<file_input>_out_dijkstra_heap_<n>.txt` dove <file_input> è il nome del grafo preso in input e <n> varia da 0 al numero dei nodi del grafo meno uno, e rappresenta il nodo sorgente per l'esecuzione dell'algoritmo di Dijkstra.
- 1 file per l'algoritmo di Bellman-Ford: `<file_input>_out_bellmanford.txt` dove <file_input> è il nome del grafo preso in input.
- |V| files per l'algoritmo di Floyd-Warshall: `<file_input>_out_floydwarshall_<n>.txt` dove <file_input> è il nome del grafo preso in input e <n> varia da 0 al numero dei nodi del grafo meno uno, e rappresenta il nodo sorgente dell'albero dei cammini minimi derivante dall'esecuzione dell'algoritmo di Floyd-Warshall.

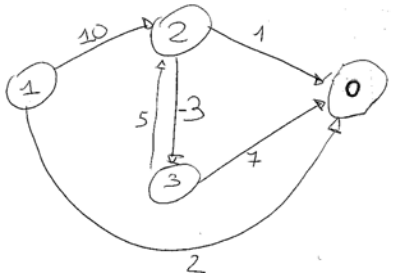
Il **formato dei file di testo** che rappresentano grafi (sia in input che in output è il seguente):

- la prima linea contiene il numero di nodi presenti nel grafo. Sia n tale numero
- si assuma che i nodi sono mumerati da 0 a n-1
- Seguono 2n righe in modo tale che ogni coppia di righe consecutive si riferisce alla lista di adiacenza di un nodo del grafo, a partire dal nodo 0 fino ad arrivare al nodo n-1.
- Ad ogni nodo i (i=0,…,n-1) corrispondono quindi 2 righe consecutive (la riga 2(i+1)-esima e la riga numero 2(i+1)+1-esima). La prima di queste contiene, separati da virgole, gli indici dei nodi adiacenti al nodo i. La seconda, sempre separati da virgole, il peso degli archi che connette il nodo i ai vari adiacenti, in corrispondenza all'ordine degli adiacenti dato nella prima riga.

Se ad esempio il grafo è il seguente, una sua possibile rappresentazione come file di testo è:

```
4


2,0
10,2
3,0
-3,1
0,2
7,5
```



Si noti che la seconda e la terza riga sono vuote in quanto il nodo 0 non ha archi uscenti.