Elaborato esercizio 4 - Laboratorio Algoritmi e Strutture dati

Taralli Giulio - Toure Ismaila

Scelte implementative

Sezione grafo

Per rappresentare in memoria il grafo abbiamo deciso di utilizzare la struttura dati HashMap che contiene al suo interno come chiave i vertici del grafo e come valore una classe Vertex contenente un'altra HashMap la quale rappresenterà gli archi collegati a quel vertice.
La seconda HashMap ha come chiave il nome del vertice di destinazione e come valore il peso dell'arco.

Grazie a questa struttura dati riusciamo a soddisfare tutte le complessità riportate in consegna.

Sezione algoritmo di Kruskal

Per eseguire al meglio l'algoritmo di Kruskal abbiamo deciso di rappresentare l'insieme di archi come un ArrayList di ArrayList, dove l'ArrayList più interno ha sempre lunghezza 2:

- il primo elemento rappresenta il vertice di partenza dell'arco
- il secondo arco rappresenta il vertice di arrivo dell'arco

Una volta inseriti tutti gli archi nella struttura dati, non resta che ordinarla in modo crescente in base al peso con il MergeSort.

La seconda struttura dati utilizzata è un ArrayList di oggetti di tipo Branch, essi rappresenteranno gli alberi (o un solo albero) di copertura minima del grafo.
Su di esso verranno quindi eseguite le operazioni di Union-Find.

- Quando verrà eseguita l'operazione di Union (la quale restituirà un nuovo oggetto Branch) essa verrà inserita nella posizione dove era presente il vertice di partenza dell'arco e verrà eliminato il Branch dove risiedeva il vertice di destinazione dell'arco.

Come supporto all'ArrayList di oggetti di tipo Branch implementeremo un ArrayList di HashMap, essa rappresenterà al suo interno tutti nomi dei vertici del branch in modo tale da eseguire più facilmente una ricerca dell'elemento specifico (su quale oggetto Branch si trova)
Ovviamente quando verrà usata la funzione Union dovremmo manualmente unire anche le due HashMap in modo tale da mantenere corretta la rappresentazione in base all'ArrayList di tipo Branch (quindi mantenere l'integrità della struttura)

Conclusioni

La tempistica complessiva dell'esecuzione dell'esercizio si attesta intorno ai 4-8 secondi.

```
run:
     [java] Loading the dataset...
     [java] Data loaded
     [java]
     [java] Graph's vertices number: 18640
     [java]
     [java] graph's arcs numbers: 18640
     [java] creating the list of ordered arcs...
     [java] sorting the list...
     [java] Kruskal's algorithm started...
     [java] Picked up _JAVA_OPTIONS: -Xmx512M
     [java]
     [java] Total weight of the minimal spanning tree: 8.993991258557318E7
     [java] MST-KRUSKAL TIME: 3993 millisec

BUILD SUCCESSFUL
Total time: 4 seconds
```

```
run:
     [java] Loading the dataset...
     [java] Data loaded
     [java]
     [java] Graph's vertices number: 18640
     [java]
     [java] graph's arcs numbers: 18640
     [java] creating the list of ordered arcs...
     [java] sorting the list...
     [java] Kruskal's algorithm started...
     [java]
     [java] Total weight of the minimal spanning tree: 8.9939912585573 18E7
     [java] MST-KRUSKAL TIME: 4343 millisec
     [java] Picked up _JAVA_OPTIONS: -Xmx512M

BUILD SUCCESSFUL
Total time: 5 seconds
```

```
run:
     [java] Loading the dataset...
     [java] Data loaded
     [java]
     [java] Graph's vertices number: 18640
     [java]
     [java] graph's arcs numbers: 18640
     [java] creating the list of ordered arcs...
     [java] sorting the list...
     [java] Kruskal's algorithm started...
     [java]
     [java] Total weight of the minimal spanning tree: 89939.91258557318 km
     [java] MST-KRUSKAL TIME: 7144 millisec
     [java] Picked up _JAVA_OPTIONS: -Xmx512M

BUILD SUCCESSFUL
Total time: 8 seconds
```