Algoritmi e Strutture Dati

Consegna 4

# Organizzazione del codice

Il codice è organizzato come da standard nelle cartelle bin, build, lib, src, contenenti rispettivamente i file binari, oggetto, le librerie utilizzate (come junit e hamcrest) e sorgenti. Il codice sorgente è a sua volta organizzato in pacchetti contenenti le diverse strutture dati (disjointset, graph, kruskalmst) e l’applicazione main (kruskalusage). Oltre l’implementazione delle unit test per ciascuna struttura, con l’utilizzo di Unity, nella cartella troviamo un file build.xml utile a compilare, eseguire i test ed eseguire l’applicazione, il tutto rapidamente grazie ad Apache Ant.

Le istruzioni per l’esecuzione sono presenti nel file KruskalUsage.java e vengono stampate a schermo in caso di errore nel passaggio dei parametri richiesti.

# Struttura dati DisjointSet

La classe DisjointSet utilizza una HashMap che codifica ogni Node **radice** utilizzando come chiave proprio l’elemento del nodo. In questo modo, abbiamo accesso alle operazioni di base della struttura in tempi costanti. Inoltre le caratteristiche della HashMap si rivelano ottimali proprio per le implementazioni dei metodi della classe. La classe contiene i metodi makeSet, union, link, findSet, utili alla rappresentazione e gestione di foreste di insiemi, utilizzando poi le euristiche di unione per rango (metodo link richiamato da union) e compressione dei cammini (metodo findSet). Questo verrà poi sfruttando appieno per il calcolo della MST.

Da notare, come anticipato sopra, che la HashMap si rivela particolarmente adatta all’operazione link, che si occupa di unire due insiemi e decidere chi dei due diventerà l’insieme radice, posizionando quindi il nodo radice sia nella posizione codificata con la chiave del proprio elemento che nella posizione codificata con le chiavi degli elementi dei nodi figli. Questo permette di mantenere invariata la proprietà dell’HashMap di contenere solo nodi radice.

# Struttura dati Graph

La classe Graph utilizza una HashMap composta, che ad ogni chiave “vertice” associa una HashMap che ne rappresenta la lista di adiacenza. Questa sorta di lista di adiacenza a sua volta associa ad ogni chiave “vertice adiacente” la label dell’arco.

Inoltre nella classe troviamo un’altra HashMap usata per generare in maniera efficiente la LinkedList di ritorno contenente oggetti di tipo Edge, necessaria poi per il calcolo dell’MST. Questa HashMap contiene infatti oggetti di tipo Edge, usati per rappresentare tutti gli archi del grafo, usando come chiave l’output del metodo hashcode() di ciascun oggetto Edge. In questo modo è possibile effettuare sia eliminazione che creazione di un arco in tempo costante, sia sulla struttura principale che su quella di appoggio per il ritorno a KruskalMST.

# Struttura dati KruskalMST

La classe KruskalMST si occupa del calcolo della minima foresta ricoprente e del suo peso complessivo. Questo è possibile utilizzando una struttura DisjointSet nella quale quale vengono caricati tutti i vertici come radice. In seguito, richiamando il metodo MST è possibile fare un controllo tramite union sulla coppia di vertici di ciascun, per controllare se siano disgiunti o no, e unirli in caso di esito positivo. Per ogni insieme unito con successo, si caricherà quindi l’arco in mstEdges, LinkedList usata per calcolare il peso complessivo dell’MST. Inoltre, se richiesto dall’utente, la lista può essere anche usata per generare e ritornare il grafo dell’MST.