ALGORITMI E STRUTTURE DATI

Autore: GIORGIO D'ANGELO

1) Minimum Spanning Tree

Dato un grafo non orientato e pesato quindi ad ogni arco è associato un peso vogliamo trovare il minimo albero ricoprente ossia creare un nuovo sottografo che ha gli stessi vertice ma un sottoinsieme degli archi e tra tutti i possibili spanning tree vogliamo quello il cui peso totale degli archi sia minimo.

1.1) Descrizione strutture dati utilizzate

Userò come struttura dati per costruire il grafo, una lista di adiacenza inserita all'interno di un vector.
L'idea è quella di costruire un grafo dove si conosce la posizione dei sui vertici, nel senso che se ho costruito la lista di adiacenza del nodo 6 , inserirò nell'indice 6 del grafo la lista.

Poiché il grafo non è orientato significa che un nodo A può andare in B ma anche B può andare in A.

Quindi mi sono creato lista di adiacenza delle isole 1, ma anche la lista di adiacenza delle isole 2, quindi avremo all'interno del grafo un po' di ridondanza però necessaria.

In quanto potrebbe succedere che quando estraggo un vertice ad esempio 1 e vado a vedere chi sono i nodi adiacenti ad esempio 2 e vado a verificare tutto l'algoritmo di prim, quando poi andrò estrarre 2 praticamente sto andando nella posizione 2 del grafo, quindi grafo[2] però in realtà non esiste nessuna lista se non vado a creare anche la lista di adiacenza nel senso opposto.

L'altra struttura dati che ho utilizzati è stato il minHeap creato con un vector per la coda di priorità.

Una coda di priorità è una struttura dati che serve a mantenere un insieme S di elementi, ciascuno con un valore associato detto chiave(priorità).

E' molto utilizzata in quanto il costo per l'estrazione del minimo è O(1) perché il minimo si trova sempre nella radice.

Ho usato l'heap in modo che i vertici adiacenti fossero ordinati in base al peso degli archi.

1.2) Formato dati in input/output

Leggerò un file in input ,dove il primo rigo conterrà 2 interi uno che rappresenta il numero totale di ponti e l'altro intero rappresenta il numero totale di archi.

Dopo il primo rigo avrò sempre righi composti 3 interi che rappresentano l'isola 1 , l'isola2 e l'arco.

1.3) Diagramm class

1.4) Descrizione algoritmo

L'algoritmo funziona in questo modo:

Avrò 3 vettori oltre al grafo su cui lavorare, key che indica i pesi , parent indica i padri e un vector booleano che mi indica se un vertice è stato scoperto o meno.

Il primo passo di Prim è che devo inizializzare il vector key a infinito, parent a null e il vector booleano false.

In realtà non inizializzerò tutti a massimo questo perché, avendo un grafo io inserisco il vertice nella posizione di indice vertice cioè grafo[vertice]=vertice.

Adesso il problema è questo è vero che io conosco il numero massimo di ponti ma non conosco il range dei valori , quindi se ad esempio c'è un vertice 2000 io dovrò avere un vettore di size 2000+1.

Quindi non tutti i valori del vector key verranno controllati, quindi vado a inizializzare a infinito solo i valori che esistono nel grafo.

Successivamente prendo il nodo sorgente da cui voglio iniziare, qualunque esso sia dovrebbe restituirmi lo stesso risultato con la condizione che tutti i nodi siano collegati.

Inizializzo la key della sorgente a 0 cioè il peso e inizio con quel vertice.

Estraggo ad esempio il vertice 1 ,setterò nel vector booleano nell'indice 1 true cioè significa che il nodo 1 è stato scoperto, entro in un ciclo while dove mi scorro la lista di adiacenza di 1 e vedrò chi sono i vertici adiacenti e controllerò se questi non sono stati scoperti e il peso del vertice adiacente è minore del peso settato in key(che ricordo inizializzazione è infinito) se sono vere le condizioni aggiornerò il valore di key ,il valore di parent e inserirò il vertice adiacente nell'min heap i quali verranno ordinati in base al peso degli archi e avremo sempre nella radice dell'heap l'elemento più piccolo.

In realtà prima ancora di inserire l'elemento nell'heap vado a verificare se il vertice che sto inserendo già l'avevo inserito e controllo se il nuovo vertice che sto inserendo abbia un peso minore di quello che già avevo se è cosi aggiorno il peso.

Si proseguirà finchè non verranno estratti tutti i vertici.

1.5) Complessità

Il tempo di esecuzione deriva dall'uso della coda di priorità la quale prevede come operazioni l'inserimento estrazione del minimo e ogni volta andare a controllare se è violata la proprietà dell'heap e in tal caso ripristinarla.

Avremo un ciclo for che mi scorre tutto il grafo , la prima operazione è l'extract min che richiede un tempo di esecuzione O(1) però quando cambiamo la radice con l'ultimo elemento dobbiamo controllare se è violata la proprietà dell'heap in tal caso il costo è O(logV).Quindi il costo di tutte le chiamate extract minimo avrò un costo computazionale pari a O(V logV).Il ciclo for verrà eseguito O(E) volte per tutte le liste di adiacenza, dato che nel caso peggiore vengono estratti tutti i vertici uno alla volta, significa che l'algoritmo andrà ad ispezionare ed aggiornare tutte le liste di adiacenza con cui è rappresentato il grafo e infatti la loro è il numero di relazioni del grafo, ovvero E. Quindi in questo ciclo si eseguono E aggiornamenti con costo logaritmico, quindi O(ElogV). In totale, sommando i costi ricavati da questi due cicli, si ha: O(V logV + ElogV) = O(ElogV).

1.6) Test/risultati

Test con il file dell'esempio:

11 13

1 3 0

1 4 7

2 5 0

2 6 0

3 7 0

4 10 10

5 11 0

7 11 12

7 6 8

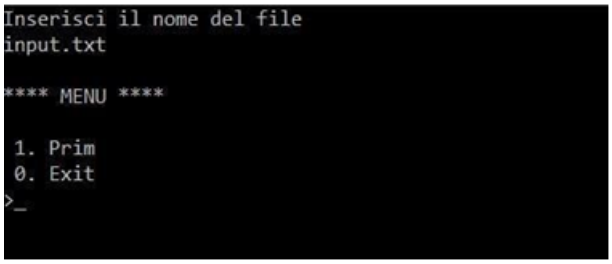
7 8 0

8 10 2

9 10 0

9 11 4

Inseriamo il nome del file txt



Inserendo 1 nel menu accediamo all'algoritmo di prim che vuole in input un nodo da dove iniziare per generare il minimum spanning tree.

Partiamo dal nodo sorgente 1.

Altro test:
Questo ci fa vedere se alcuni nodi non si possono collegare cosa succede inserendo come nodo sorgente 1 ok.txt

9 8

4 7 0

1 3 2

2 4 5

5 1 2

9 2 3

8 2 1

3 2 3

10 11 30

```
ok.txt

**** MENU ****

 1. Prim
 0. Exit
>_ 1
PRIM
INSERISCI IL NODO SORGENTE :2

PONTI RICOSTRUITI
Strada 1-3 peso 2
Strada 2-2 peso 0
Strada 3-2 peso 3
Strada 4-2 peso 5
Strada 5-1 peso 2
Strada 7-4 peso 0
Strada 8-2 peso 1
Strada 9-2 peso 3
10 non ricostruito
11 non ricostruito

=
Costo totale 16

**** MENU ****

 1. Prim
 0. Exit
>
```

Altro testo Prof.txt

```
PONTI RICOSTRUITI
Strada 3-59 peso 16
Strada 6-34 peso 16
Strada 7-18 peso 14
Strada 9-21 peso 19
Strada 12-30 peso 12
Strada 14-55 peso 11
Strada 16-12 peso 24
Strada 18-21 peso 11
Strada 20-15 peso 11
Strada 25-38 peso 19
Strada 26-60 peso 14
Strada 29-54 peso 12
Strada 34-15 peso 18
Strada 35-21 peso 12
Strada 36-12 peso 16
Strada 37-10 peso 18
Strada 39-7 peso 13
Strada 40-24 peso 18
Strada 41-21 peso 20
Strada 43-7 peso 12
Strada 44-18 peso 11
Strada 47-50 peso 12
Strada 48-38 peso 11
Strada 52-6 peso 20
Strada 55-5 peso 11
Strada 57-1 peso 12

=
Costo totale 383
```

.