

UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTENOPE

SCUOLA INTERDIPARTIMENTALE DELLE SCIENZE, DELL'INGEGNERIA E DELLA SALUTE

INFORMATICA

PROGETTO ALGORITMI E STRUTTURE DATI

Traccia III



Proponente:
Ferraro Dominick 0124002048

Data di Consegnna:
05/12/2022

Anno Accademico:
2022 – 2023

Docenti:
Prof. F. Camastra
Prof. A. Ferone

Indice

DESCRIZIONE DEL PROBLEMA	3
DESCRIZIONE STRUTTURE DATI	3 - 4
FORMATO DATI IN INPUT / OUTPUT	4
DESCRIZIONE ALGORITMO	5 - 6
CLASS DIAGRAM	7 - 8
STUDIO COMPLESSITA'	8
TEST & RISULTATI	8 - 10

LINEE NOTTURNE



DESCRIZIONE DEL PROBLEMA

Il primo ministro dello stato di *Grapha-Nui* ha necessità di risparmiare sulle spese di gestione. Per ottenere questo risultato, decide che di notte solo alcune tratte di autobus restino attive, con il vincolo che tutte le città debbano essere servite. A tal fine viene convocato un famoso informatico a cui viene fornita la piantina delle città con l'indicazione del costo di ogni tratta, con il compito di indicare quali percorsi mantenere attivi. Per prevenire eventuali interruzioni del servizio, viene richiesto di indicare l'insieme delle tratte di costo minimo e quello di costo immediatamente successivo.

Stiamo parlando quindi di un problema da risolvere con una soluzione Greedy, ci viene in aiuto quindi l'algoritmo di Kruskal dove la *greedy choose* è prendere a ogni iterazione l'arco più leggero esistente.

Una volta calcolato il primo Minimum Spanning Tree calcoleremo quello immediatamente successivo, tutta la trattazione riguardo questo passo verrà descritta in avanti.

DESCRIZIONE STRUTTURE DATI **INSIEMI DISGIUNTI**

Un struttura per insiemi disgiunti mantiene una collezione $S = \{S_1, \dots, S_k\}$ di insiemi dinamici disgiunti. Ciascun insieme è identificato da un elemento detto **rappresentante**.

Le operazioni che vengono effettuate sono diverse:

1. **Make-Set(x)**: crea un insieme con un unico elemento dato in input.
2. **Find-Set(x)**: restituisce un puntatore al rappresentante dell'insieme contenente x .
3. **Union(x, y)**: effettua l'unione degli insiemi che contengono x e y .

Ciascun insieme è rappresentato da una lista concatenata o da un albero radicato in cui ogni nodo contiene un elemento dove:

- a. *Ogni elemento punta solo al padre*
 - b. *La radice contiene il rappresentante*
 - c. *La radice è padre di sé stessa*

MINIMUM SPANNING TREE

Dato un grafo connesso non orientato pesato $G = (V, E)$ con pesi $w(u, v)$, vogliamo trovare un sottoinsieme aciclico $T \subseteq E$ che collega tutti i vertici V , il cui peso totale $w(T) = \sum_{(u,v) \in T} w(u, v)$ sia minimo.

Poiché T è aciclico e collega tutti i vertici, forma un albero chiamato albero di connessione minimo o MST.

Ci sono diversi algoritmi per individuare , dato un grafo, un MST, in particolare:

- i. Algoritmo di Prim
 - ii. Algoritmo di Kruskal
 - iii. Algoritmo di Boruvka

Gli MST sono usati nella progettazione di reti di computer, telecomunicazione, rete elettrica e idrica e **trasporto**.

L'algoritmo che utilizzeremo è Kruskal.

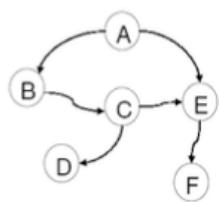
GRAFO

Un grafo $G = (V,E)$ è un insieme di vertici o nodi $v \in V$ e un insieme di archi $(v,w) \in E$.

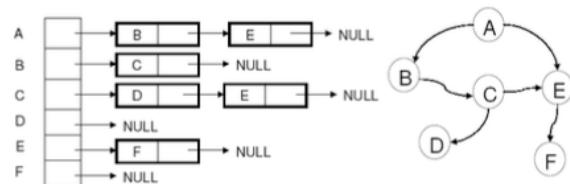
In un grafo **non orientato**, un arco è una coppia non ordinata (v,w) di nodi.

Un grafo può essere rappresentato da una matrice di adiacenza o una lista di adiacenza.

Matrice di adiacenza



Liste di Adiacenza



FORMATO DATI IN INPUT / OUTPUT

I dati di input sono estratti da file con una lettura statica del file: bisognerà cambiare nel main il file che si desidera processare.

Nel file input il primo rigo contiene due numeri interi separati da uno spazio che sono il numero di nodi e il numero di archi.

I successivi righi contengono tre numeri divisi da uno spazio che rappresentano il nodo sorgente, quello di destinazione e il loro costo.

I dati in input del file devono essere numeri positivi.

In output avremo il calcolo del primo e secondo MST e la stampa dei due grafi.

Le assunzioni fatte in input sono:

$$\begin{aligned}1 &\leq N \leq 1000 \\0 &\leq P \leq 10000 \\1 &\leq C_3 \leq 10000\end{aligned}$$

Dove N nodo sorgente, P nodo destinazione e C costo.

DESCRIZIONE ALGORITMI

KRUSKAL

L'algoritmo di Kruskal trova un **arco sicuro** da aggiungere alla foresta scegliendo tra tutti gli archi che collegano due alberi della foresta quello con peso *minimo*. I costi degli archi sono valori positivi. Kruskal è un algoritmo Greedy, la scelta greedy in questo caso è scegliere ad ogni passo l'arco con minor peso esistente.

Siano A1 e A2 due alberi collegati da (u,v) , l'arco (u,v) è un arco leggero, quindi un arco sicuro.

Si definisce **foresta** un grafo non orientato dove due vertici sono connessi al più da un cammino. Kruskal usa una struttura dati per insiemi disgiunti dove ogni insieme contiene i vertici di un albero della foresta attuale.

Con l'operazione **Find-Set()** verifichiamo se due nodi appartengono alla stessa componente连通的 e con l'operazione **Union()** uniamo i due alberi.

L'idea di Kruskal è semplice: ordiniamo gli archi in ordine crescente in base al costo e li ispezioniamo singolarmente, inserendo l'arco se non forma un ciclo con gli archi già inseriti.

Un MST non è unico: a ogni passo di Kruskal, se abbiamo diversi archi con lo stesso costo, possiamo sceglierne uno qualsiasi a nostro piacimento, purché non crei un ciclo.

KRUSKAL (G, w)

1. $A \leftarrow \emptyset$
2. $\forall v \in G.V$
3. $Make-Set(v)$
4. $Sort(E) \quad //ordinamento crescente$
5. $\forall (u,v) \in G.E$
6. *If* $Find-Set(u) \neq Find-Set(v)$
7. $A = A \cup \{(u,v)\}$
8. $Union(u,v)$
9. *Return* A

SEARCH BEST MST

Search Best MST si occupa di risolvere il problema presentato: calcola il primo MST e quello subito successivo.

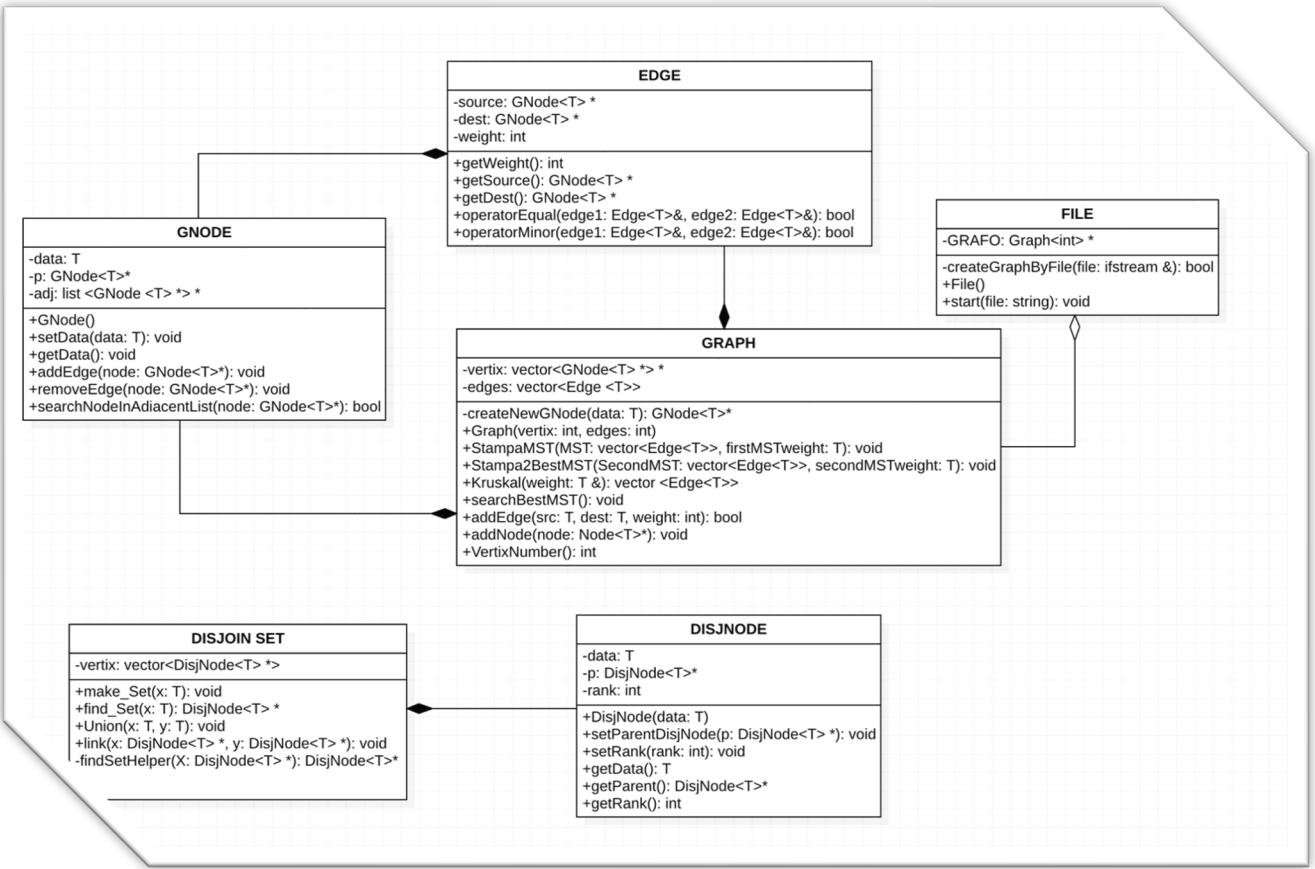
Questa procedura si serve dell'algoritmo di Kruskal calcolando prima il miglior MST esistente e poi ripete questa operazione per trovare quello subito dopo:

- Dopo aver calcolato il primo MST lo salviamo.
- Per ogni arco u,v appartenente al MST, lo rimuoviamo temporaneamente.
- Calcoliamo il nuovo MST con un arco in meno.
- Ricerchiamo l'MST minimo salvando il risultato e aggiornando la variabile.
- Aggiungiamo nuovamente l'arco temporaneamente eliminato.
- Ripetiamo l'operazione finché non abbiamo eliminato e calcolato tutti gli MST possibili, alla fine il nostro algoritmo ritornerà il costo del miglior MST e del secondo miglior MST.

SEARCH-BEST-MST()

1. $MST = Kruskal()$
 2. $MST2 = \emptyset$
 3. $MST-TMP = \emptyset$
 4. $MST2-COST = inf$
 5. $MST-TMP-COST = 0$
 6. $\forall (u,v) \in MST$
 7. $tmp = (u,v)$ *//salvo arco prima di eliminarlo*
 8. *Elimina (u,v) dall'insieme degli archi*
 9. $MST-TMP = Kruskal()$
 10. *IF* ($MST-TMP-COST < MST2-COST$)
 11. $MST2-COST = MST-TMP-COST$
 12. $MST2 = MST-TMP$
 13. *INSERISCI tmp >>> insieme degli archi*
 14. *Return MST, MST2*
-

CLASS DIAGRAM



Il progetto è composto da 6 classi:

- 1. File:** la classe file dispone le operazioni per caricare il grafo, legge da file la prima riga (numero di nodi e numero di archi del grafo) e inizializza il grafo con le successive righe lette. Inoltre viene richiamata un'istanza nel main per lanciare il calcolo del primo e secondo MST.
 - 2. Graph:** è la struttura dati portante che consente di memorizzare un grafo, contiene diverse procedure come la stampa degli MST, aggiunta di un nodo, Kruskal e searchBestMST.
 - 3. GNode:** è la classe dei singoli nodi del grafo.
 - 4. DisjointSet:** classe che rappresenta gli insiemi disgiunti con tutte le classiche operazioni.
 - 5. DisjNode:** classe che rappresenta un nodo degli insieme disgiunti.
 - 6. Edge:** rappresenta la classe degli archi del grafo Graph.

RELAZIONE TRA LE CLASSI

Le classi **Graph**, **Edge** e **GNode** hanno una relazione di **composizione** poiché se rimuovessimo la classe **Graph**, allora **Edge** e **GNode** non avrebbero molto senso di esistere.

Lo stesso vale per le due classi **DisjointSet** e **DisjNode**, anche qui vi è una relazione di **composizione**.

Un'altra relazione di composizione vi è tra **GNode** e **Edge**.

Per quanto riguarda la classe **FILE**, ha una relazione di **aggregazione** con **Graph** poiché usa quest'ultima per alcune operazioni offerte.

STUDIO COMPLESSITA'

Sappiamo che l'algoritmo di Kruskal impiega un tempo, nel caso peggiore di $O(E \log V)$.

Analizziamo la complessità di Search Best MST.

Calcoliamo la prima volta Kruskal per il primo MST: il costo sarà quindi $O(E \log V)$.

L'eliminazione dell'arco, così come il salvataggio in una variabile hanno un tempo costante di $O(1)$.

In un ciclo for, per n volte calcoliamo il nuovo MST avendo una complessità di $O(E \log V n)$.

La ricerca del minimo impiega un tempo costante $O(1)$, lo stesso vale per inserire nuovamente l'arco all'insieme degli archi.

Alla fine il nostro algoritmo avrà la stessa complessità dell'algoritmo di Kruskal, con l'unica differenza che questo verrà effettuato n volte, quindi la complessità totale è

$$O(n E \log V) + O(1) + O(1) = O(n E \log V).$$

Dove n è il numero degli archi del MST.

Le altre procedure di "contorno" non hanno una complessità particolare da analizzare in quanto sono semplici operazioni.

TEST & RISULTATI

Una volta caricato il file, se l'operazione è andata a buon fine visualizzeremo un messaggio di conferma lettura file:

[SUCCESS] File inputProf.txt caricato correttamente.

Nel caso il file non dovesse esistere oppure si trova in un path differente del progetto, il programma notificherà il problema:

[WARNING] Errore apertura file. Il file inputPrsof.txt potrebbe essere inesistente o trovarsi nel path diverso del programma

[in questo caso è stato caricato un file inesistente]

Dopo aver caricato il file, verranno processati i dati e a schermo stamperà il primo e secondo MST.

Peso Primo MST = 4290

4 ~~> 9 ~~> 69
2 ~~> 12 ~~> 139
3 ~~> 4 ~~> 143
4 ~~> 13 ~~> 148
3 ~~> 7 ~~> 208
1 ~~> 0 ~~> 213
4 ~~> 2 ~~> 254
14 ~~> 6 ~~> 266
12 ~~> 10 ~~> 327
11 ~~> 10 ~~> 344
12 ~~> 5 ~~> 357
12 ~~> 15 ~~> 382
14 ~~> 10 ~~> 403
1 ~~> 6 ~~> 507
11 ~~> 8 ~~> 530

Peso Secondo Miglior MST = 4323

4 ~~> 9 ~~> 69
2 ~~> 12 ~~> 139
3 ~~> 4 ~~> 143
4 ~~> 13 ~~> 148
3 ~~> 7 ~~> 208
1 ~~> 0 ~~> 213
4 ~~> 2 ~~> 254
14 ~~> 6 ~~> 266
12 ~~> 10 ~~> 327
11 ~~> 10 ~~> 344
12 ~~> 5 ~~> 357
12 ~~> 15 ~~> 382
14 ~~> 12 ~~> 436
1 ~~> 6 ~~> 507
11 ~~> 8 ~~> 530

Se i primi due numeri letti da file non rispettano i limiti della traccia, ecco l'alert:

```
[WARNING] I dati alla riga principale sono errati. I valori iniziali devono essere  
1 <= N <= 1000 e 0 <= P <= 10000.
```

Stessa cosa vale per i successivi input: ecco l'alert se il nodo è maggiore del numero di nodi totali:

```
OVERFLOW: la dimensione del nodo e' maggiore del numero dei vertici
```

Nel caso in cui il peso superi la soglia stabilità, ecco l'alert:

```
[WARNING] Non e' possibile aggiungere l'arco (11 8) perche' il suo peso  
non rispetta la soglia stabilita di [0 <= PESO <= 10000]
```

Nel caso di un arco già esistente, ecco l'alert:

```
[WARNING] Non e' possibile aggiungere l'arco (11 8), gia' esiste.
```