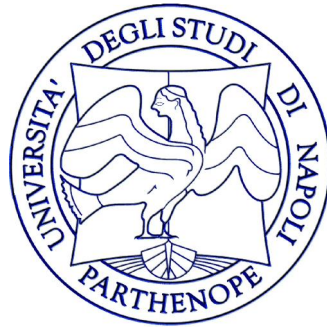


UNIVERSITÀ DEGLI STUDI DI NAPOLI "PARTHENOPE"
SCUOLA INTERDIPARTIMENTALE DELLE SCIENZE
DELL'INGEGNERIA E DELLA SALUTE

CORSO DI LAUREA IN INFORMATICA



DOCUMENTAZIONE PROGETTO ALGORITMI E STRUTTURE DATI
E
LABORATORIO DI ALGORITMI E STRUTTURE DATI

TRACCIA 3

DOCENTI

Camastra Francesco

Ferone Alessio

CANDIDATO

Iannucci Vincenzo

matricola: 0124002093

Anno Accademico 2021-2022

Indice

1	RBGraph	1
1.1	Descrizione problema	1
1.2	Descrizione strutture dati	1
1.3	Formato dati in input/output	4
1.4	Descrizione algoritmo	4
1.4.1	Creazione di un nuovo nodo del grafo	4
1.4.2	Aggiunta arco al grafo	5
1.4.3	Rimozione arco dal grafo	5
1.4.4	Ricerca di un arco del grafo	6
1.4.5	Elaborazione dei dati di input	6
1.5	Class diagram	8
1.5.1	Classi e il loro ruolo	8
1.5.2	Relazioni vigenti tra le classi	9
1.6	Studio complessità	9
1.7	Test/risultati	10
1.7.1	File inesistente	10
1.7.2	Aggiunta arco al grafo	10
1.7.3	Rimozione arco dal grafo	11
1.7.4	Ricerca di un arco nel grafo	14
1.7.5	BFS	15
2	Linee Notturme	17
2.1	Descrizione problema	17
2.2	Descrizione strutture dati	18
2.3	Formato dati in input/output	21
2.4	Descrizione algoritmo	21
2.5	Class diagram	24

2.5.1	Classi e il loro ruolo	24
2.5.2	Relazioni vigenti tra le classi	25
2.6	Studio complessità	25
2.7	Test/risultati	25
2.7.1	Grafo input0_3_2.txt	26
2.7.2	Grafo traccia 3	27
2.7.3	Gestione input errati	28

Elenco delle figure

1.1	Grafo orientato	2
1.2	(a) Grafo orientato. (b) Grafo orientato con liste di adiacenza.(c) Grafo orientato con matrice di adiacenze.	2
1.3	Albero red black	3
1.4	Diagramma delle classi RBGraph	8
1.5	Grafo caricato nel file di input	10
1.6	File inesistente	11
1.7	Aggiunta arco al grafo.	12
1.8	Arco già esistente	12
1.9	Nodo inserito fuori dal range	13
1.10	Rimozione arco dal grafo	13
1.11	Rimozione arco dal grafo: Arco inesistente	14
1.12	Ricerca di un arco nel grafo	15
1.13	Ricerca di un arco nel grafo: Arco inesistente	15
1.14	BFS eseguita sui nodi del grafo	16
1.15	BFS: Sorgente inesistente	16
2.1	Grafo non orientato che rappresenta la piantina della città, con i costi per ogni tratta	17
2.2	Esempio di grafo non orientato	18
2.3	(a) Grafo non orientato. (b) Grafo non orientato con liste di adiacenza.(c) Grafo non orientato con matrice di adiacenze.	19
2.4	Foresta di insiemi disgiunti (a) Due insieme disgiunti (b) Gli insiemi disgiunti dopo l'operazione di UNION	20
2.5	Compressione del cammino durante l'operazione di FIND-SET (a) Prima della FIND-SET (b) Dopo la FIND-SET	20
2.6	Diagramma delle classi Linee Notturne	24

2.7	Input e output del secondo grafo	26
2.8	Test 1	26
2.9	Input e output del primo grafo	27
2.10	Test 2	27
2.11	N errato, P corretto	28
2.12	N e P errati	29
2.13	N corretto, P errato	29
2.14	Underflow del costo	30
2.15	Overflow del costo	30

List of Algorithms

1	Creazione di un nuovo nodo del grafo	5
2	Aggiunta al grafo di un nuovo arco	5
3	Rimozione di un arco dal grafo	6
4	Ricerca di un arco del grafo	6
5	Elaborazione dei dati di input	7
6	Algoritmo di Kruskal	21
7	Operazioni sugli insiemi disgiunti	22
8	Trova il primo e il secondo MST	23

Capitolo 1

RBGraph

1.1 Descrizione problema

Si vuole realizzare la struttura dati RBGraph che consenta di memorizzare un grafo orientato in cui la lista di adiacenza di ogni nodo è rappresentata da un albero Red Black. Progettare ed implementare una struttura dati che, dato un file di input contenente il grafo, costruisca RBGraph corrispondente e consenta di effettuare le seguenti operazioni: AddEdge(i,j), RemoveEdge(i,j), FindEdge(i,j) e BFS(s). Il file di input contiene nel primo rigo due numeri interi, $0 \leq N \leq 1000$ e $0 \leq M \leq 1000$, separati da uno spazio che rappresentano rispettivamente il numero di nodi ed il numero di archi. I successivi M righi contengono due numeri interi separati da uno spazio che rappresentano il nodo sorgente ed il nodo destinazione. Dotare il programma di un menu da cui sia possibile richiamare le suddette operazioni.

1.2 Descrizione strutture dati

La traccia richiede espressamente l'uso di un grafo orientato che abbia come lista di adiacenza un albero red black. Un grafo $G = (V, E)$ è definito come un insieme di nodi o **vertici** $v \in V$ e un insieme di **archi** $(v, w) \in E$, dove v e w sono una coppia ordinata di vertici presenti in V . In particolare, definiamo **grafo orientato** un grafo in cui per ogni arco $(v, w) \in E$ si ha che $(v, w) \neq (w, v)$.

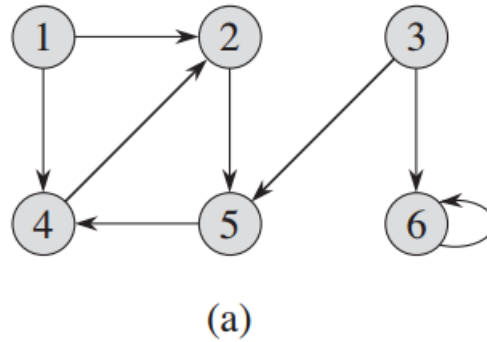


Figura 1.1: Grafo orientato

Un grafo può essere rappresentato in due modi:

- mediante lista di adiacenza
- mediante matrice di adiacenza

La traccia suggerisce di utilizzare il primo tipo di rappresentazione, che consiste nel far in modo che il vertice abbia una lista linkata in cui siano memorizzati i nodi a cui esso è collegato tramite arco. Come ben sappiamo però le liste linkate non

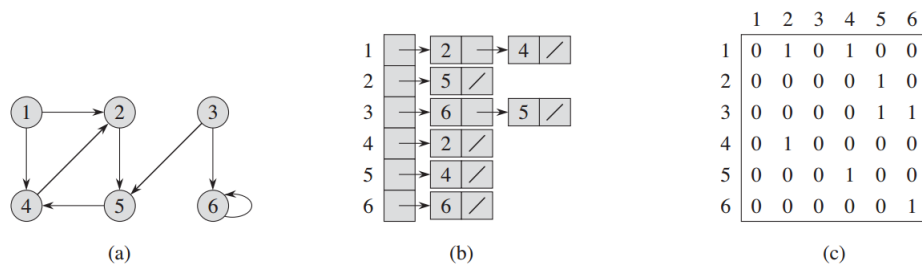


Figura 1.2: (a) Grafo orientato. (b) Grafo orientato con liste di adiacenza. (c) Grafo orientato con matrice di adiacenze.

offrono una buona complessità quando si tratta di visitare i dati presenti nella

lista in quanto bisogna scorrerla fintanto che non si trova il valore desiderato (si impiega un $O(n)$). Tuttavia, il loro uso trova giustificazione nel fatto che la quantità di memoria impiegata per la loro memorizzazione è $\Theta(V + E)$, inferiore rispetto alla matrice di adiacenza in quanto la matrice costruita avrà dimensione $|V| \times |V|$.

Tuttavia, invece di utilizzare le liste linkate si richiede di utilizzare un **albero red black** per memorizzare la lista di adiacenza di ciascun vertice del grafo. Un albero red black è un albero binario di ricerca che ha l'interessante proprietà di essere un albero auto-bilanciante. Ciò è reso possibile da cinque proprietà che ogni albero red black deve rispettare:

1. Ogni nodo può essere **rosso** o **nero**;
2. La radice dell'albero è **nera**;
3. Ogni foglia (NIL) è **nera**;
4. Un nodo **rosso** deve avere i figli **neri**;
5. Ogni cammino semplice da un nodo alle sue foglie discendenti contiene lo stesso numero di nodi **neri**.

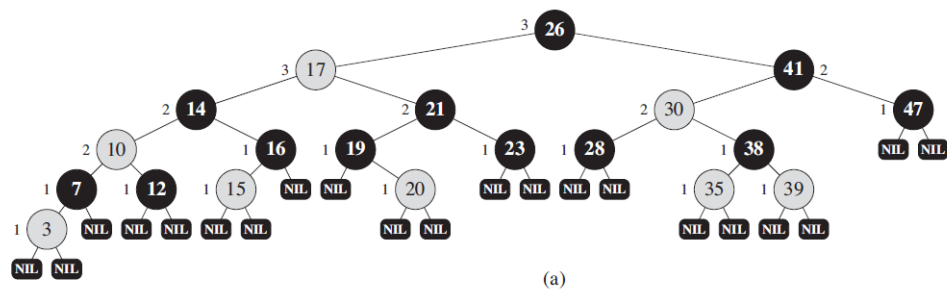


Figura 1.3: Albero red black

Il rispetto di queste proprietà fa sì che la complessità di un albero red black per inserire, cancellare e ricercare un nodo sia pari ad $O(\log n)$. Dunque la complessità di queste operazioni non è influenzata dall'altezza dell'albero, come accadeva per l'albero binario di ricerca.

1.3 Formato dati in input/output

I dati di input vengono letti da un file. Nel file di input i dati sono riportati come una coppia di valori separati da uno spazio. La prima coppia identifica rispettivamente il numero di nodi e il numero di archi del grafo. I restanti valori rappresentano il nodo di partenza e il nodo di destinazione dell'arco. L'output del programma dipende dalla scelta dell'utente, dato che si interagisce con esso tramite un menù composto dalle seguenti voci:

1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(src)

1.4 Descrizione algoritmo

In questa sezione viene effettuata una descrizione degli algoritmi principali della classe **RBGraph**. Si andranno a descrivere l'algoritmo per la creazione dei nodi del grafo, per l'aggiunta dei nodi del grafo, per la loro rimozione e per la loro ricerca. L'algoritmo della BFS non viene descritto in quanto è stato ampiamente trattato a lezione e il suo pseudocodice si trova nel Cormen. La descrizione è effettuata tramite pseudocodice ispirato a quello del Cormen ma in certi casi è più discorsivo rispetto a quello in uso nel libro di testo.

1.4.1 Creazione di un nuovo nodo del grafo

L'algoritmo di creazione del nodo riceve in input la chiave che il nuovo nodo dovrà avere. Prima di crearlo però, controlla se tale nodo è già stato creato precedentemente: in caso affermativo lo restituisce immediatamente. Altrimenti procede con la normale creazione del nodo. Restituisce in output il nuovo nodo, con chiave key.

Algorithm 1 Creazione di un nuovo nodo del grafo

```

function CREATE-NEW-GRAPH-NODE( $G$ ,  $key$ )
  if esiste un vertice  $v \in G.V$  con chiave uguale a  $key$  then
    return  $v$ 
  else
    crea il nodo  $node$ 
     $node.key = key$ 
    return  $node$ 
  end if
end function

```

1.4.2 Aggiunta arco al grafo

L'algoritmo di aggiunta di un arco al grafo ha il compito di creare i due nodi del grafo, ricevendo in input il nodo sorgente e il nodo destinazione. Dopodiché, l'algoritmo controlla se il nodo destinazione è già presente nella lista di adiacenza del nodo sorgente. In caso affermativo termina, altrimenti il nodo di destinazione viene aggiunto alla lista di adiacenza del nodo sorgente. Restituisce true se l'arco viene creato correttamente, false altrimenti.

Algorithm 2 Aggiunta al grafo di un nuovo arco

```

function ADD-EDGE( $G$ ,  $src$ ,  $dest$ )
   $srcNode = \text{CREATE-NEW-GRAPH-NODE}(G, src)$ 
   $destNode = \text{CREATE-NEW-GRAPH-NODE}(G, dest)$ 
  if  $destNode$  si trova nel RBTREE di adiacenze di  $srcNode$  then
    return false
  end if
  Inserisci  $destNode$  nell'albero red black delle adiacenze di  $srcNode$ 
  return true
end function

```

1.4.3 Rimozione arco dal grafo

L'algoritmo di rimozione di un arco dal grafo riceve in input il nodo sorgente e il nodo destinazione. L'algoritmo richiama la funzione FIND-EDGE per cercare se il nodo è presente nel grafo. Qualora il nodo non fosse presente, la funzione

ritorna immediatamente, dato che non è possibile cancellare un nodo che non esiste. Nel caso il nodo fosse presente, procede con la sua eliminazione.

Algorithm 3 Rimozione di un arco dal grafo

```
function REMOVE-EDGE(G, src, dest)  
    result = FIND-EDGE(G, src, dest)  
    if result == true then  
        rimuovi dest dall'albero red black delle adiacenze di src  
        return true  
    else  
        return false  
    end if  
end function
```

1.4.4 Ricerca di un arco del grafo

L'algoritmo di ricerca di un arco nel grafo riceve in input il nodo sorgente e il nodo destinazione. L'algoritmo controlla se il nodo di destinazione è già presente nella lista di adiacenza del nodo sorgente. In caso affermativo ritorna true, altrimenti ritorna false.

Algorithm 4 Ricerca di un arco del grafo

```
function FIND-EDGE(G, src, dest)  
    cerca il nodo dest nell'albero red black delle adiacenze di src  
    if il nodo dest si trova nell'albero red black delle adiacenze di src then  
        return true  
    else  
        return false  
    end if  
end function
```

1.4.5 Elaborazione dei dati di input

L'algoritmo seguente descrive nel dettaglio tutto il processo che parte dall'acquisizione del file fino al caricamento dei dati contenuti in quest'ultimo all'interno del grafo. La funzione ritorna 0 se i dati sono stati acquisiti correttamente, -1 in caso contrario.

Algorithm 5 Elaborazione dei dati di input

```
function INPUT-PROCESSING(G, file-name)
    apri il file con nome file-name
    if non è possibile aprire il file then
        print "Errore nome file"
        return -1
    else
        leggi il primo rigo del file
        if i valori letti non sono in formato corretto then
            print "Valori riportati in formato non corretto"
            return - 1
        else
            converti il contenuto in due numeri interi che rappresentano N e M
            if N e M non sono compresi tra 0 e 1000 then
                print "N e M non soddisfano i requisiti"
                return -1
            else
                while ci sono ancora righe del file da leggere do
                    leggi un rigo
                    controlla che i valori sono nel formato corretto
                    if i valori letti non sono in formato corretto then
                        print "Valori riportati in formato non corretto"
                        return -1
                    else
                        converti il rigo in due valori: src e dest
                        ADD-EDGE(G, src, dest)
                    end if
                end while
            end if
        end if
    return 0
end function
```

1.5 Class diagram

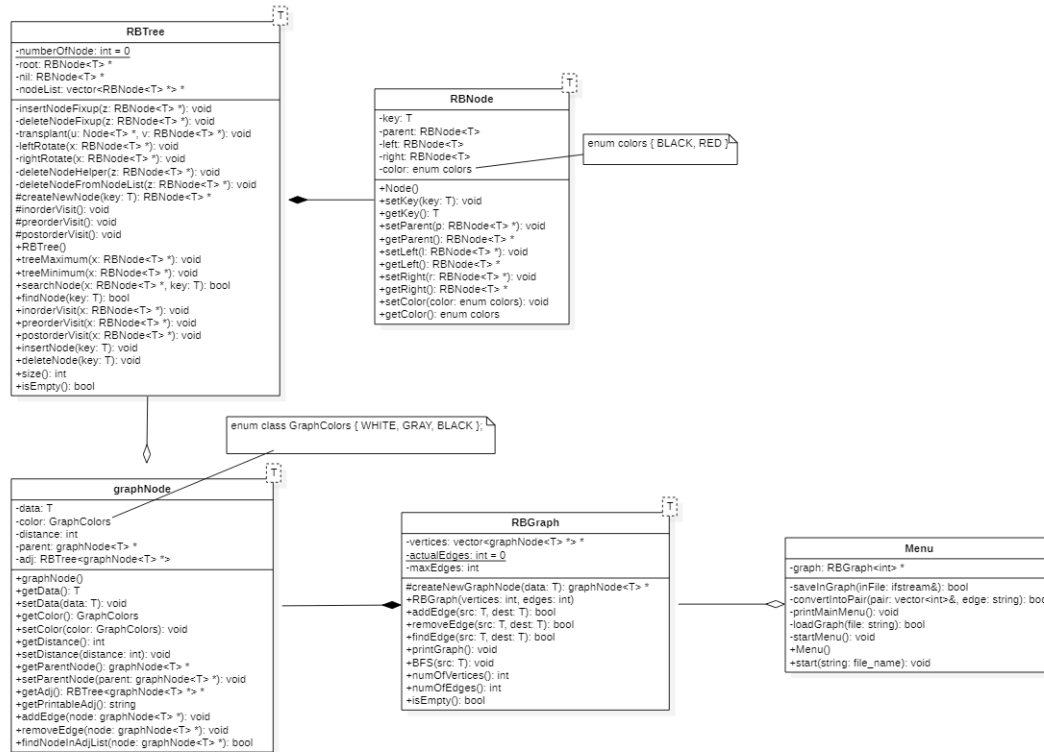


Figura 1.4: Diagramma delle classi RBGraph

1.5.1 Classi e il loro ruolo

Tutte le classi (tranne **Menu**) sono template. Questo perché si vuole rendere le classi riutilizzabili con diversi parametri che differiscano dall'<int> usato per il progetto. La classe **RBTree** rappresenta un albero red black ed è composto da **RBNode**. Mentre la classe **RBGraph** rappresenta il nostro RBGraph ed è composta da **graphNode**. La classe **Menu** contiene un'istanza dell'**RBGraph** ed è la classe che si occupa di tutte le operazioni necessarie a caricare i dati nel grafo da file, permette la visualizzazione del menù e gestisce l'interazione dell'utente con quest'ultimo attraverso una serie di appositi metodi.

1.5.2 Relazioni vigenti tra le classi

Tra le classi **RBTree** e **RBNode** esiste una relazione di **composizione**, così come tra **RBGraph** e **graphNode**. Il motivo va ricercato nel fatto che le classi **RBNode** e **graphNode** sono strettamente correlate alle classi **RBTree** e **RBGraph**, nel senso che la loro esistenza perde di significato senza l'esistenza di quest'ultime. Tra **graphNode** e **RBTree** esiste una relazione di **aggregazione**, poiché l'**RBTree** può esistere autonomamente anche senza **graphNode**, le due classi non sono strettamente correlate. Discorso analogo per la relazione di aggregazione tra **Menu** e **RBGraph**.

1.6 Studio complessità

Per tutta la trattazione N indicherà il numero di nodi del grafo e con M il numero di archi del grafo. La complessità delle operazioni di un grafo dipendono da quale struttura dati si utilizza per la memorizzazione delle liste di adiacenze. Facciamo dunque un parallelismo tra la complessità delle operazioni in un grafo normale e un grafo red black, che cioè utilizza alberi red black per rappresentare la lista di adiacenza di ogni nodo.

In un grafo, l'operazione $\text{AddEdge}(i, j)$ prevede l'inserimento del nodo j nella lista di adiacenza del nodo i . Questa operazione ha costo pari ad $O(1)$, dato che è il costo per inserire un nodo in una lista concatenata (sia che sia singolarmente concatenata o doppiamente concatenata). Nel caso del grafo red black, l'operazione $\text{AddEdge}(i, j)$ fa uso dell'inserimento dell'albero red black, che dunque costa $O(\log M)$.

L'operazione $\text{FindEdge}(i, j)$ nel grafo richiama l'operazione di ricerca su una lista, che impiega un tempo $O(M)$ nel caso peggiore. L'operazione $\text{RemoveEdge}(i, j)$ richiama l'operazione di eliminazione di un nodo dalla lista, che a sua volta richiama l'operazione di ricerca per trovare il nodo da eliminare. Per cui il costo di queste operazioni è $O(M)$. Per quanto riguarda il grafo red black, $\text{FindEdge}(i, j)$ e $\text{RemoveEdge}(i, j)$ richiamano rispettivamente l'operazione di ricerca e di inserimento di un nodo nell'albero red black. Il tempo impiegato da queste operazioni è pertanto $O(\log M)$.

La complessità della BFS (Breadth First Search) sia che si utilizzi un albero red black o una lista di adiacenza è sempre $O(N + M)$, poiché questa dipende dalla

lunghezza di tutte le liste di adiacenza (o alberi red black) dei vertici del grafo.

1.7 Test/risultati

Nei test che sono stati svolti utilizzando la classe **RBGraph** è stato usato come grafo di test il seguente grafo:

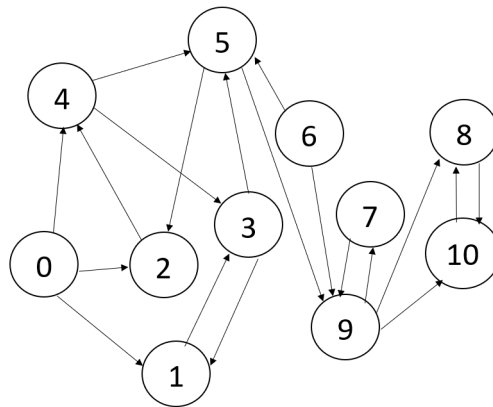


Figura 1.5: Grafo caricato nel file di input

1.7.1 File inesistente

Il menù offre dei controlli sulla correttezza del file che si vuole caricare. Se per caso viene fornito il nome di un file inesistente o che non si trova nella cartella dell'IDE, il menu lo segnala con un messaggio di errore.

1.7.2 Aggiunta arco al grafo

La scelta 1 del menù corrisponde all'aggiunta di un arco al grafo. I nodi del grafo non possono essere aggiunti in nessun modo, bisogna lavorare con quelli che vengono caricati all'inizio dal file. Una volta premuto 1 il menù richiede di inserire nodo sorgente e nodo destinazione dell'arco. Dopo il suo inserimento, il programma stampa le liste di adiacenza (in questo caso sono alberi RB) di tutti i nodi del grafo.

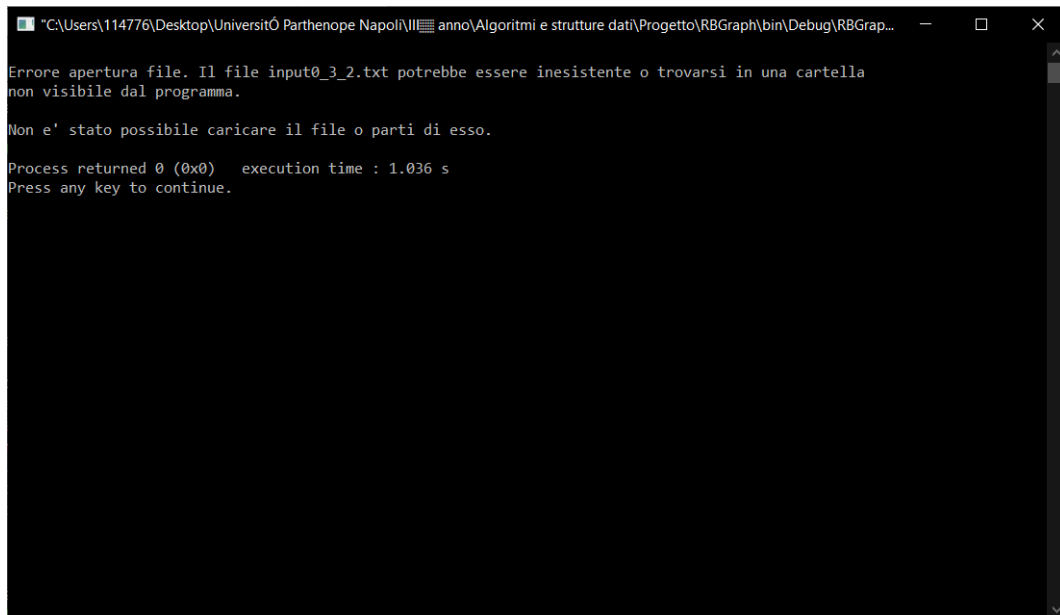


Figura 1.6: File inesistente

Arco gi  esistente

Se viene inserito un arco che gi  si trova memorizzato nel grafo, questo viene segnalato con un messaggio di errore, evitando l'inserimento a vuoto dello stesso arco.

Nodo inserito fuori dal range

Se viene inserito un arco in cui uno dei due nodi (o entrambi) sono fuori dal range fissato in input, questo viene segnalato con un messaggio di errore.

1.7.3 Rimozione arco dal grafo

La scelta 2 del men  corrisponde alla rimozione di un arco dal grafo. I nodi del grafo non possono essere rimossi in nessun modo, bisogna lavorare con quelli che vengono caricati all'inizio dal file. Una volta premuto 2 il men  richiede di inserire nodo sorgente e nodo destinazione dell'arco da eliminare. Dopo la rimozione dell'arco, il programma stampa le liste di adiacenza (in questo caso sono alberi RB) di tutti i nodi del grafo.

```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGraph\bin\Debug\RBGraph...
File input0_3_1.txt caricato con successo.

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta: 1

Scelta 1
Nodo sorgente: 0
Nodo destinazione: 10
0 -> 1
0 -> 4
0 -> 2
0 -> 10
1 -> 3
1 -> 6
2 -> 4
3 -> 1
3 -> 5
4 -> 3
4 -> 5
5 -> 2
5 -> 9
6 -> 5
6 -> 9
7 -> 5
7 -> 9
8 -> 10
9 -> 7
9 -> 8
9 -> 10
10 -> 8
Arco 0 10 aggiunto con successo

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta:
```

Figura 1.7: Aggiunta arco al grafo.

```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGraph\bin\Debug\RBGrap...
File input0_3_1.txt caricato con successo.

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta: 1

Scelta 1
Nodo sorgente: 0
Nodo destinazione: 1

Non e' stato possibile creare l'arco 0 1
L'arco gia' esiste.

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta:
```

Figura 1.8: Arco gi  esistente

```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGraph\bin\Debug\RBGrap...
Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta: 1

Scelta 1
Nodo sorgente: 0
Nodo destinazione: 12

Non e' stato possibile creare l'arco 0 12
Il numero di vertici del grafo e' 12 (da 0 a 11)

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta:
```

Figura 1.9: Nodo inserito fuori dal range

```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGraph\bin\Debug\...
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta: 2

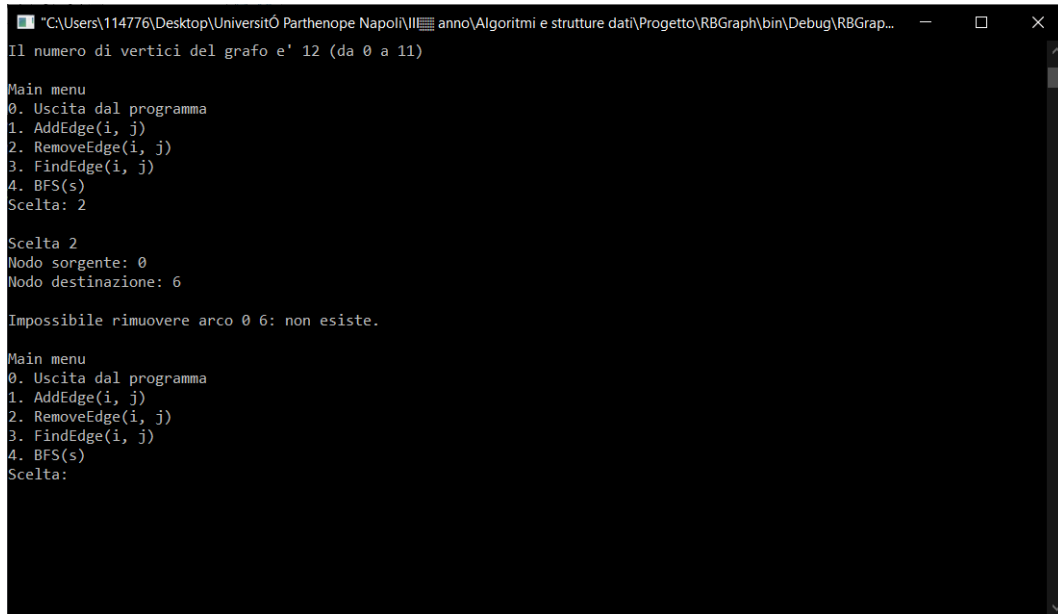
Scelta 2
Nodo sorgente: 0
Nodo destinazione: 10
0 -> 1
0 -> 4
0 -> 2
1 -> 3
1 -> 6
2 -> 4
3 -> 1
3 -> 5
4 -> 3
4 -> 5
5 -> 2
5 -> 9
6 -> 5
6 -> 9
7 -> 5
7 -> 9
8 -> 10
9 -> 7
9 -> 8
9 -> 10
10 -> 8
Arco 0 10 rimosso con successo

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta:
```

Figura 1.10: Rimozione arco dal grafo

Arco inesistente

Se viene inserito un arco che non è memorizzato nel grafo, questo viene segnalato con un messaggio di errore, evitando la rimozione di un arco che non esiste.



```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGraph\bin\Debug\RBGrap...
Il numero di vertici del grafo e' 12 (da 0 a 11)

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta: 2

Scelta 2
Nodo sorgente: 0
Nodo destinazione: 6

Impossibile rimuovere arco 0 6: non esiste.

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta:
```

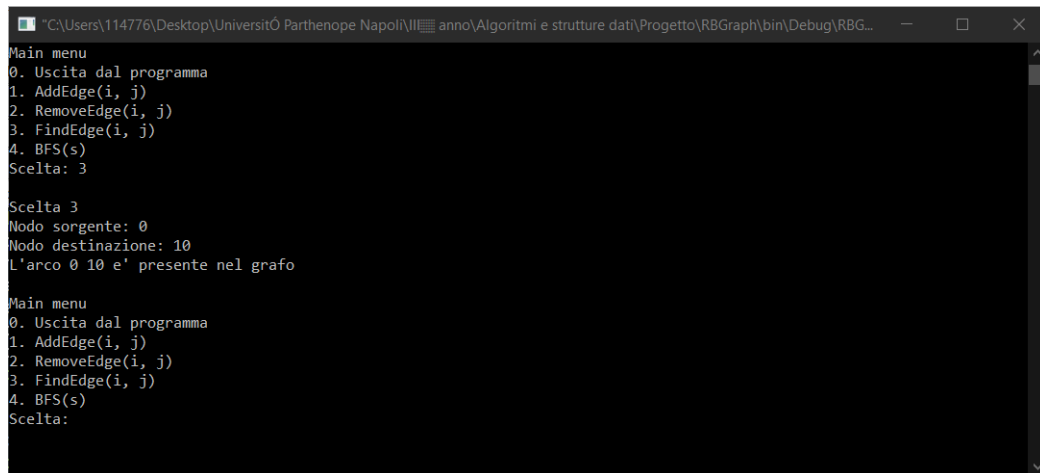
Figura 1.11: Rimozione arco dal grafo: Arco inesistente

1.7.4 Ricerca di un arco nel grafo

La scelta 3 del men  corrisponde alla ricerca di un arco al grafo. Una volta premuto 3 il men  richiede di inserire nodo sorgente e nodo destinazione dell'arco da ricercare. Il programma risponde alla richiesta mostrando un messaggio che attesta l'esistenza dell'arco all'interno del grafo.

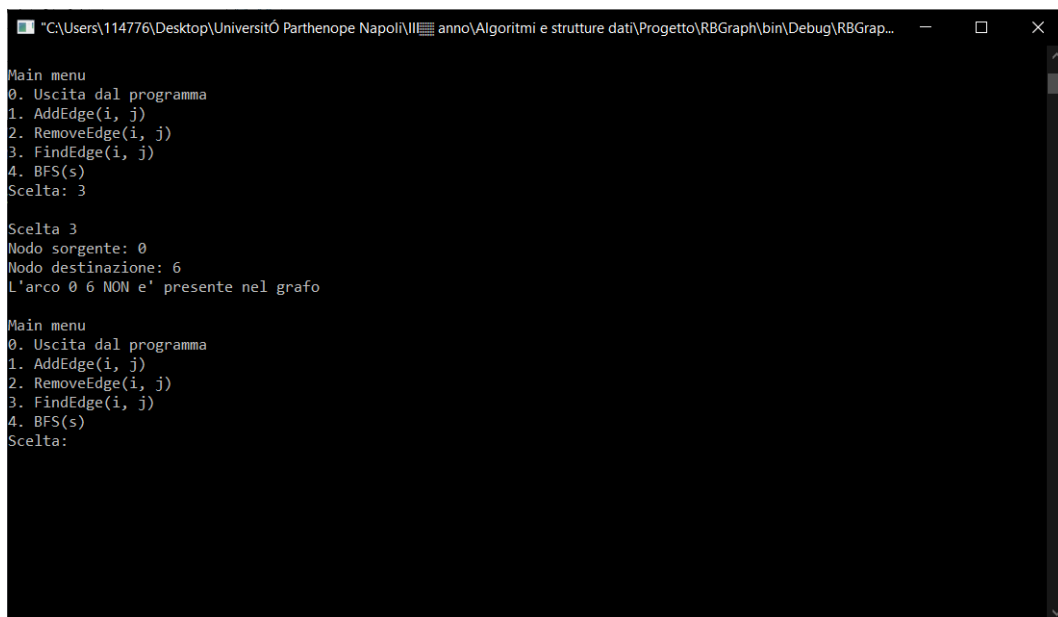
Arco inesistente

Se viene inserito un arco che non   memorizzato nel grafo, questo viene segnalato con un messaggio che segna all'utente il fatto che l'arco ricercato non   memorizzato nel grafo.



```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGraph\bin\Debug\RBG...  
Main menu  
0. Uscita dal programma  
1. AddEdge(i, j)  
2. RemoveEdge(i, j)  
3. FindEdge(i, j)  
4. BFS(s)  
Scelta: 3  
  
Scelta 3  
Nodo sorgente: 0  
Nodo destinazione: 10  
L'arco 0 10 e' presente nel grafo  
  
Main menu  
0. Uscita dal programma  
1. AddEdge(i, j)  
2. RemoveEdge(i, j)  
3. FindEdge(i, j)  
4. BFS(s)  
Scelta:
```

Figura 1.12: Ricerca di un arco nel grafo

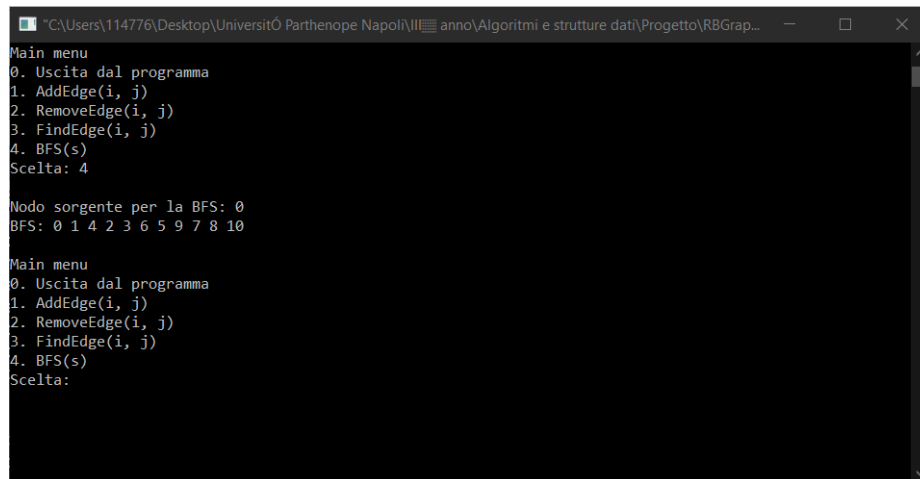


```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGraph\bin\Debug\RBGrap...  
Main menu  
0. Uscita dal programma  
1. AddEdge(i, j)  
2. RemoveEdge(i, j)  
3. FindEdge(i, j)  
4. BFS(s)  
Scelta: 3  
  
Scelta 3  
Nodo sorgente: 0  
Nodo destinazione: 6  
L'arco 0 6 NON e' presente nel grafo  
  
Main menu  
0. Uscita dal programma  
1. AddEdge(i, j)  
2. RemoveEdge(i, j)  
3. FindEdge(i, j)  
4. BFS(s)  
Scelta:
```

Figura 1.13: Ricerca di un arco nel grafo: Arco inesistente

1.7.5 BFS

La scelta 4 del men  mostra la visita BFS. Si   scelto di mostrare l'ordine in cui i nodi vengono visitati senza applicarea alcuna formattazione particolare.



```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGrap...
Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta: 4

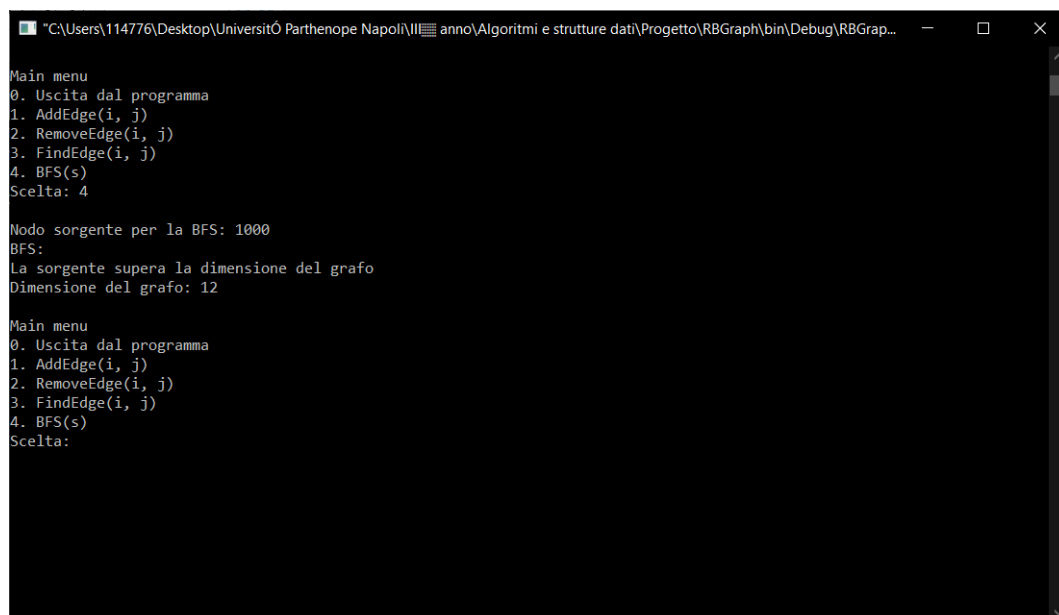
Nodo sorgente per la BFS: 0
BFS: 0 1 4 2 3 6 5 9 7 8 10

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta:
```

Figura 1.14: BFS eseguita sui nodi del grafo

Sorgente inesistente

Se la sorgente inserita per la BFS non esiste, il programma lo segnale con un apposito messaggio di errore.



```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\RBGraph\bin\Debug\RBGrap...
Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta: 4

Nodo sorgente per la BFS: 1000
BFS:
La sorgente supera la dimensione del grafo
Dimensione del grafo: 12

Main menu
0. Uscita dal programma
1. AddEdge(i, j)
2. RemoveEdge(i, j)
3. FindEdge(i, j)
4. BFS(s)
Scelta:
```

Figura 1.15: BFS: Sorgente inesistente

Capitolo 2

Linee Notturne

2.1 Descrizione problema

La piantina della città può essere vista come un **grafo non orientato**, dove le città sono i vertici, le tratte sono gli archi del grafo e i costi delle tratte sono i costi degli archi del grafo. Viene richiesto di trovare l'insieme delle tratte di costo minimo e quelle di costo immediatamente successivo, il che si traduce nel trovare il **Minimum Spanning Tree (MST)** e il secondo MST del grafo.

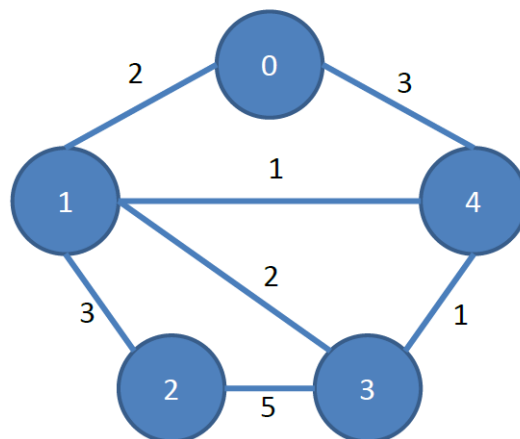


Figura 2.1: Grafo non orientato che rappresenta la piantina della città, con i costi per ogni tratta

2.2 Descrizione strutture dati

Un grafo $G = (V, E)$ è definito come un insieme di nodi o **vertici** $v \in V$ e un insieme di **archi** $(v, w) \in E$, dove v e w sono una coppia ordinata di vertici presenti in V . In particolare, definiamo **grafo non orientato** un grafo in cui per ogni arco $(v, w) \in E$ si ha che $(v, w) = (w, v)$.

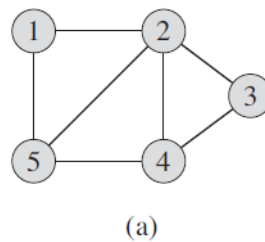


Figura 2.2: Esempio di grafo non orientato

Un grafo può essere rappresentato in due modi:

- mediante lista di adiacenza
- mediante matrice di adiacenza

La memorizzazione tramite **lista di adiacenza** consiste nel far in modo che il vertice abbia una lista linkata in cui siano memorizzati i nodi a cui esso è collegato tramite arco. Poiché nel nostro caso abbiamo un grafo non orientato, dato l'arco (u, v) bisogna assicurarsi che u sia presente nella lista di adiacenza di v ma anche che v sia presente la lista di adiacenza di u .

Per quanto riguarda la **matrice delle adiacenze**, bisogna creare una matrice $|V| \times |V|$ dove in corrispondenza della cella $[u, v]$ viene memorizzato 1 se c'è un arco che unisce i due vertici, 0 in caso contrario. Nel caso di un grafo non orientato, la matrice A è **simmetrica rispetto la diagonale principale**, cioè è tale che $A = A^T$. Quindi si può scegliere di memorizzare solo gli elementi della diagonale principale e al di sopra di essa, per risparmiare una quantità di memoria pari quasi alla metà di quella impiegata per memorizzare la matrice.

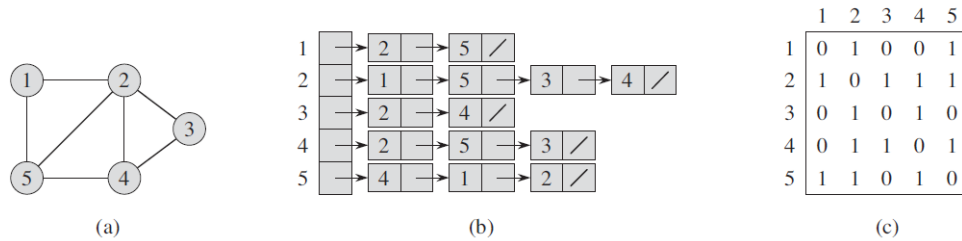


Figura 2.3: (a) Grafo non orientato. (b) Grafo non orientato con liste di adiacenza. (c) Grafo non orientato con matrice di adiacenze.

La traccia richiede di trovare l'MST e il secondo MST del grafo. Sia G un grafo **non orientato connesso**. Si definisce **spanning tree** un sottografo connesso di G che è un albero e connette tutti i nodi del grafo. Si definisce invece **Minimum Spanning Tree**, abbreviato in MST, uno spanning tree la cui somma dei costi degli archi è la più piccola possibile.

Per trovare l'MST ricorriamo all'algoritmo di Kruskal. Tale algoritmo fa uso al suo interno di una particolare struttura dati chiamata **insieme disgiunto**. In particolare, scegliamo di rappresentare l'insieme disgiunto come un albero radicato costituito da vari elementi e dove ogni elemento punta al proprio padre. La radice punta a sé stessa. Le tre operazioni fondamentali per gli insiemi disgiunti sono:

- **MAKE-SET(x)**: crea un nuovo albero in cui x è l'unico nodo
- **UNION(x, y)**: unisce gli insiemi che contengono x e y in un unico insieme (Figura 2.4).
- **FIND-SET(x)**: ritorna il puntatore al rappresentante dell'insieme (in questo caso la radice dell'albero). (Figura 2.5)

Per rendere tutto più efficiente si utilizzano le euristiche di **unione per rango** e **compressione del cammino** per le operazioni di UNION e FIND-SET.

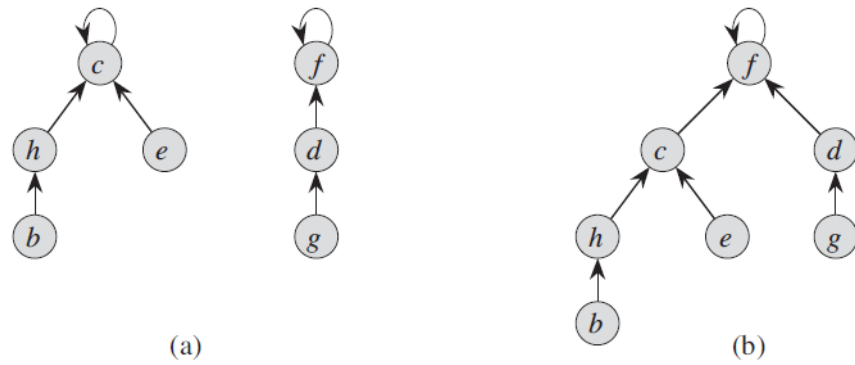


Figura 2.4: Foresta di insiemi disgiunti (a) Due insieme disgiunti (b) Gli insiemi disgiunti dopo l'operazione di UNION

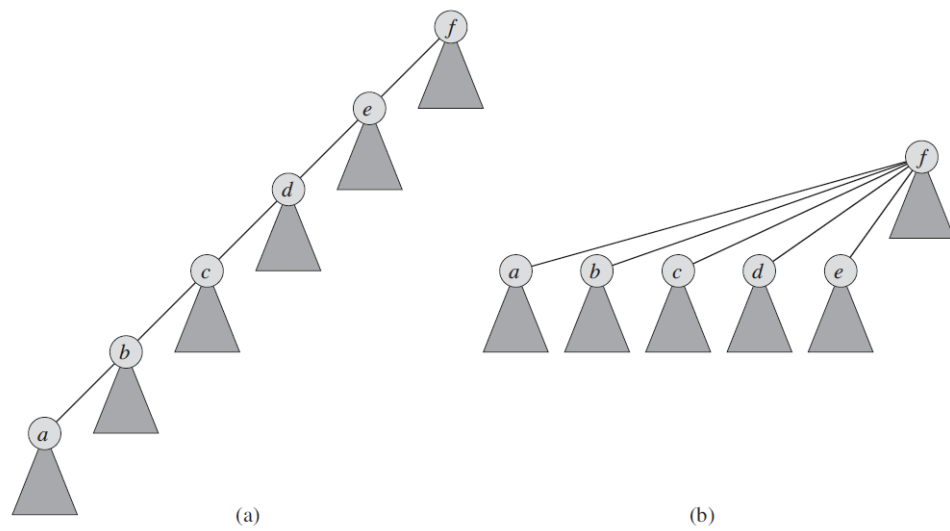


Figura 2.5: Compressione del cammino durante l'operazione di FIND-SET (a) Prima della FIND-SET (b) Dopo la FIND-SET

2.3 Formato dati in input/output

I dati di input vengono letti da un file. Nel file di input i dati sono riportati come una tripla di valori separati da uno spazio. La prima coppia identifica rispettivamente il numero di nodi N e il numero di archi P del grafo. I restanti valori rappresentano il nodo di partenza, il nodo di destinazione e il costo dell'arco. Il programma restituisce il costo del primo MST e gli archi che lo compongono, con i relativi costi e il costo del secondo MST e gli archi che lo compongono, con i relativi costi.

2.4 Descrizione algoritmo

L'algoritmo FIND-FIRST-AND-SECOND-MST si occupa di trovare l'insieme delle tratte di costo minimo e quello di costo immediatamente successivo, che coincide con trovare l'MST del grafo fornito in input ed il secondo MST. L'algoritmo riceve in input un grafo G e la funzione peso w e restituisce una lista contenente i due MST. Si assume che gli MST siano un insieme di archi e che contengano un attributo *cost* che rappresenta il costo complessivo dell'MST. Questo perché per evitare computazioni ulteriori, il costo complessivo dell'MST viene calcolato dall'algoritmo di Kruskal. Si noti che apportando questa piccola modifica all'algoritmo di Kruskal il suo costo rimane invariato.

Algorithm 6 Algoritmo di Kruskal

```

function MST-KRUSKAL( $G, w$ )
   $A = \emptyset$ 
   $A.cost = 0$ 
  for ogni vertice  $v \in G.V$  do
    MAKE-SET( $v$ )
  end for
  Ordina gli archi  $G.E$  in ordine crescente di peso  $w$ 
  for ogni arco  $(u, v) \in G.E$ , preso in ordine crescente di peso do
    if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ) then
       $A = A \cup \{(u, v)\}$ 
       $A.cost = A.cost + w(u, v)$ 
      UNION( $u, v$ )

```

```
    end if
  end for
  return A
end function
```

Algorithm 7 Operazioni sugli insiemi disgiunti

```
function MAKE-SET(x)
  x.p = x
  x.rank = 0
end function

function FIND-SET(x)
  if x  $\neq$  x.p then
    x.p = FIND-SET(x.p)
  end if
  return x.p
end function

function UNION(x, y)
  LINK(FIND-SET(x), FIND-SET(y))
end function

function LINK(x)
  if x.rank > y.rank then
    y.p = x
  else
    x.p = y
    if x.rank == y.rank then
      y.rank = y.rank + 1
    end if
  end if
end function
```

Algorithm 8 Trova il primo e il secondo MST

```
1: function FIND-FIRST-AND-SECOND-MST( $G, w$ )
2:    $MST1 = MST\text{-}KRUSKAL(G, w)$ 
3:    $MST2 = \emptyset$ 
4:    $min\text{-}cost = \infty$ 
5:   Sia  $(s, t)$  un nuovo arco temporaneo
6:   for ogni arco  $(u, v) \in MST1$ .E do
7:      $(s, t) = (u, v)$   $\triangleright$  Salviamo  $(u, v)$  prima di rimuoverlo
8:     rimuovi arco  $(u, v)$  da  $G.E$ 
9:      $MST\text{-}temp = MST\text{-}KRUSKAL(G, w)$ 
10:     $mst\text{-}temp\text{-}cost = MST\text{-}temp.cost$ 
11:    if  $mst\text{-}temp\text{-}cost < min\text{-}cost$  then
12:       $min\text{-}cost = mst\text{-}temp\text{-}cost$ 
13:       $MST2 = MST\text{-}temp$ 
14:    end if
15:    aggiungi arco  $(s, t)$  a  $G.E$ 
16:  end for
17:  Sia  $L$  una nuova lista di MST
18:   $INSERT(L, MST1)$ 
19:   $INSERT(L, MST2)$ 
20:  return  $L$ 
21: end function
```

2.5 Class diagram

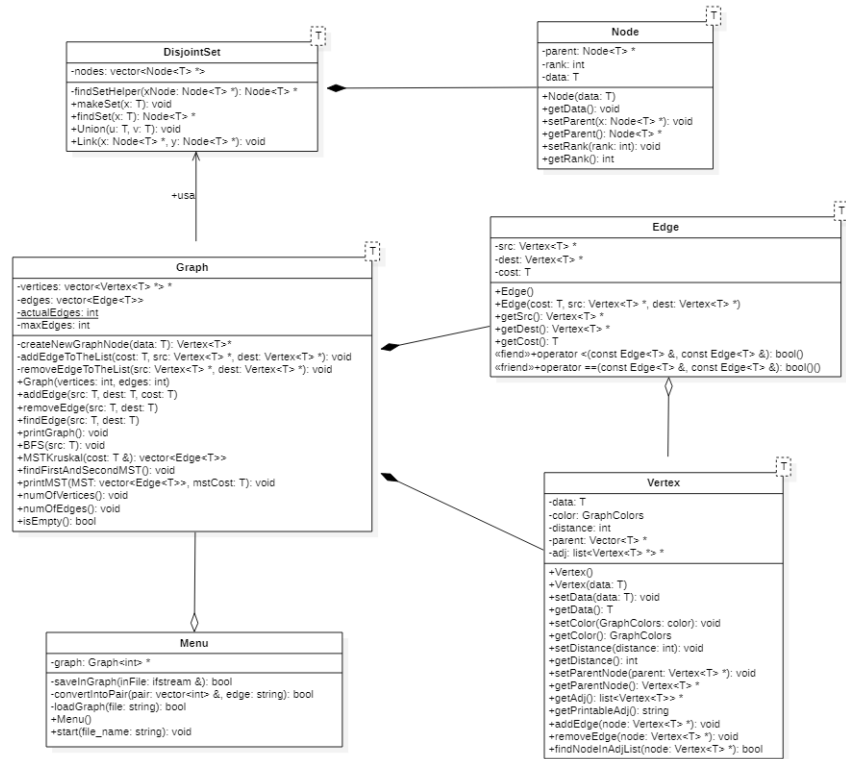


Figura 2.6: Diagramma delle classi Linee Notturne

2.5.1 Classi e il loro ruolo

Tutte le classi (tranne Menu) sono template. Questo perché si vuole rendere le classi riutilizzabili con diversi parametri che differiscano dall'<int> usato per il progetto. La classe **Graph** rappresenta il grafo ed è composta da **Vertex** e **Edge** che rappresentano rispettivamente vertici e archi del grafo. La classe **DisjointSet** rappresenta l'insieme disgiunto come albero radicato, composto da **Node**. La classe **Menu** contiene un'istanza di **Graph**. È responsabile del caricamento dei dati da file all'interno del grafo e dell'esecuzione della funzione che stampa a video i due MST, con i rispettivi archi e il loro peso.

2.5.2 Relazioni vigenti tra le classi

Tra le classi **Graph** e **Vertex** esiste una relazione di **composizione**, così come tra **Graph** e **Edge**. Questo perché Vertex e Edge sono strettamente correlate alla classe Graph, ossia la loro esistenza perderebbe di significato se la classe Graph non esistesse. La stessa relazione di composizione vige anche tra le classi **DisjointSet** e **Node**, per la stessa motivazione. Tra la classe **Menu** e la class Graph vi è un'aggregazione, perché la classe Graph può avere esistenza indipendente rispetto alla classe Menu, le due classi non sono strettamente correlate.

2.6 Studio complessità

Di seguito viene riportata l'analisi della complessità dell'algoritmo FIND-FIRST-AND-SECOND-MST(G, w). Alla riga 2 viene calcolato l'MST in un tempo $O(P \log N)$ utilizzando l'algoritmo di Kruskal. Il ciclo for righe 6-16 viene eseguite un numero di volte pari al numero di archi dell'MST, che ha $N - 1$ archi. Si conclude che il ciclo for viene eseguito in un $O(N)$. Il salvataggio dell'arco (u, v) supponiamo impieghi un tempo costante. La rimozione dell'arco riga 8 impiega, poiché rappresentiamo il grafo mediante liste di adiacenza, un tempo $O(P)$. Il calcolo dell'MST alla riga 9 impiega sempre un $O(P \log N)$. In totale, il ciclo for righe 6-16 costa $O(NP \log N)$. Poiché l'inserimento dei due MST nella lista righe 18-19 impiega un tempo costante, concludiamo che il tempo di esecuzione dell'algoritmo è $O(NP \log N)$.

2.7 Test/risultati

I test sono stati effettuati con due grafi. Il primo è quello fornito insieme alla traccia e il secondo è quello che si trova nel file `input0_3_2.txt`

2.7.1 Grafo input0_3_2.txt

input.txt	Output
15 40	best: 4290
11 8 530	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
11 9 653	14-10->403 1-6->507 11-8->530
11 12 411	second best: 4323
11 6 682	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
11 10 344	14-12->436 1-6->507 11-8->530
8 3 966	
8 2 769	
3 4 143	
3 9 211	
3 10 955	
3 7 208	
4 9 69	
4 2 254	
4 12 462	
4 5 608	
4 6 549	
4 13 148	
9 12 919	
2 14 687	
2 12 139	
2 6 446	
2 13 476	
14 12 436	
14 15 681	
14 6 266	
14 10 403	
12 15 382	
12 5 357	
12 1 867	
12 10 327	
15 5 635	
5 10 750	
1 6 507	
1 13 761	
1 0 213	
6 10 622	
6 13 853	
10 13 824	
13 0 934	
0 7 607	

Figura 2.7: Input e output del secondo grafo

```

"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\Linee...
File input0_3_2.txt caricato con successo.

cost = 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

cost = 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

Process returned 0 (0x0) execution time : 0.198 s
Press any key to continue.

```

Figura 2.8: Test 1

2.7.2 Grafo traccia 3

input.txt	Output
4 7	0-1, 1-4, 1-2, 3-4 = 7
0 1 2	0-1, 1-2, 1-3, 3-4 = 8
0 4 3	
1 4 1	
1 2 3	
1 3 2	
2 3 5	
3 4 1	

Figura 2.9: Input e output del primo grafo

```

"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\LineeNo...
File input0_3_2.txt caricato con successo.

cost = 7
1-4->1  3-4->1  0-1->2  1-2->3

cost = 8
3-4->1  0-1->2  1-3->2  1-2->3

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.

```

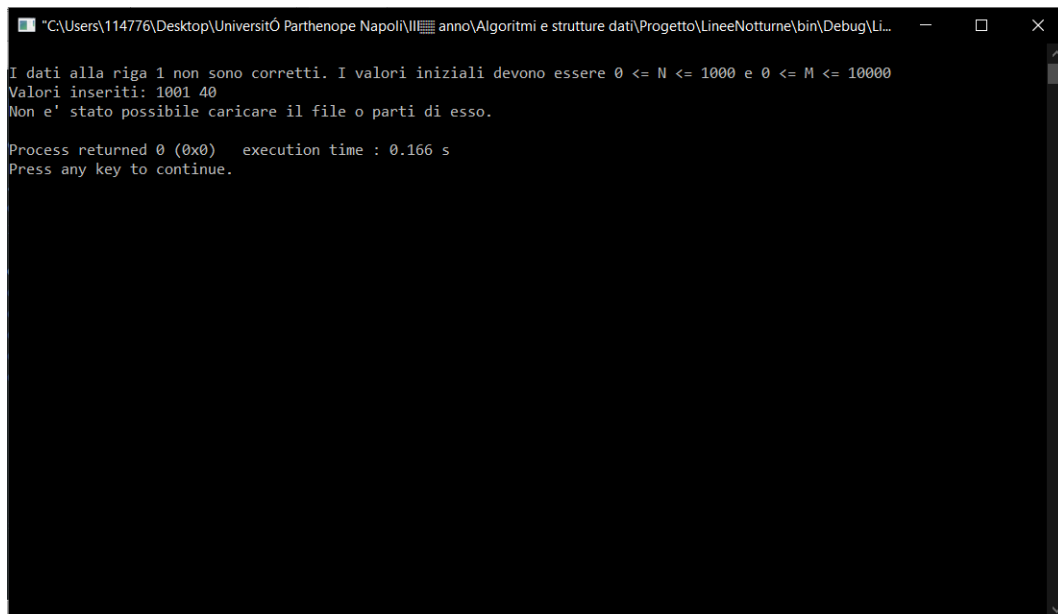
Figura 2.10: Test 2

2.7.3 Gestione input errati

Il programma gestisce gli eventuali errori che possono coinvolgere i dati di input. Infatti, come riporta la traccia

- $1 \leq N \leq 1000$
- $0 \leq P \leq 10000$
- $1 \leq C_3 \leq 10000$

Dove N è il numero di vertici, P è il numero di archi e C_3 è il costo dell'arco. Di seguito sono riportate una serie di schermate che rappresentano la gestione degli input errati.



```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\LineeNotturme\bin\Debug\Li...
I dati alla riga 1 non sono corretti. I valori iniziali devono essere 0 <= N <= 1000 e 0 <= M <= 10000
Valori inseriti: 1001 40
Non e' stato possibile caricare il file o parti di esso.

Process returned 0 (0x0)   execution time : 0.166 s
Press any key to continue.
```

Figura 2.11: N errato, P corretto

```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\LineeNottur \bin\Debug\Li...
I dati alla riga 1 non sono corretti. I valori iniziali devono essere 0 <= N <= 1000 e 0 <= M <= 10000
Valori inseriti: 1001 10001
Non e' stato possibile caricare il file o parti di esso.

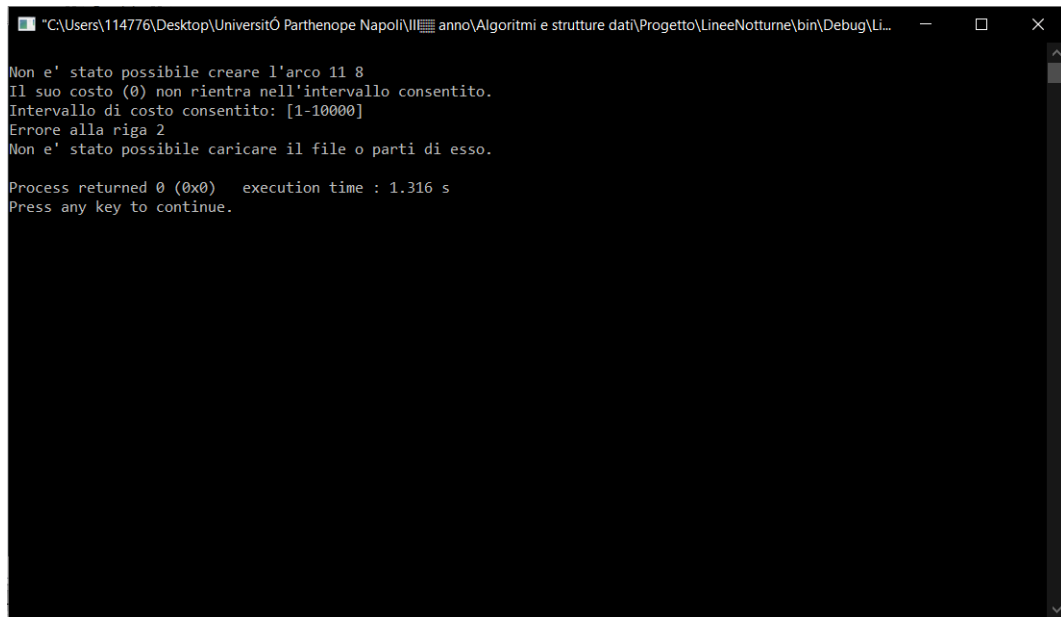
Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

Figura 2.12: N e P errati

```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\LineeNottur \bin\Debug\Li...
I dati alla riga 1 non sono corretti. I valori iniziali devono essere 0 <= N <= 1000 e 0 <= M <= 10000
Valori inseriti: 15 10001
Non e' stato possibile caricare il file o parti di esso.

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

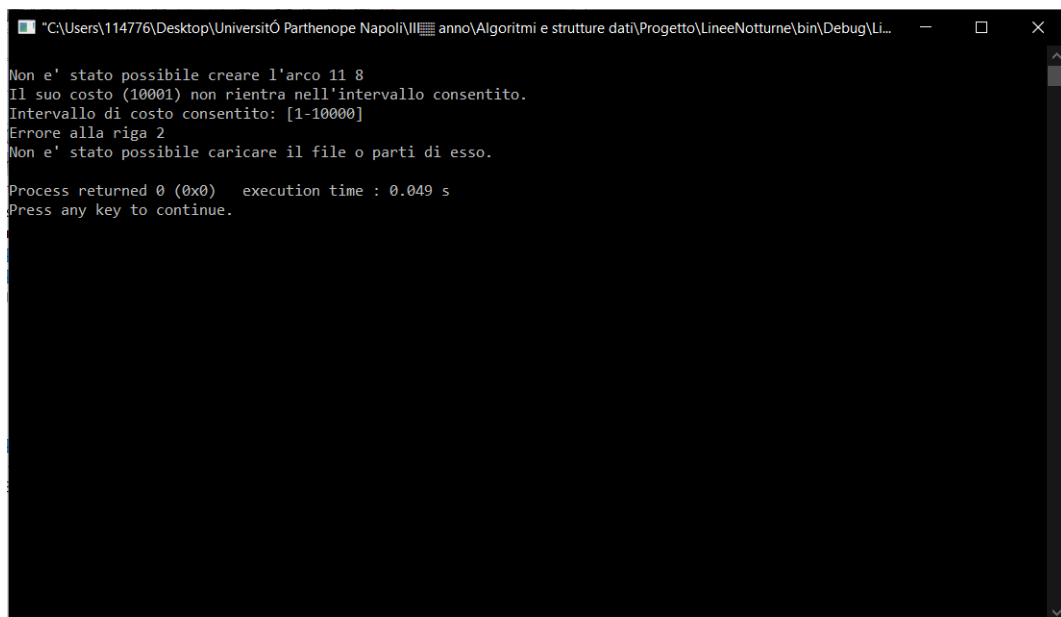
Figura 2.13: N corretto, P errato



```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\LineeNotturme\bin\Debug\Li...
Non e' stato possibile creare l'arco 11 8
Il suo costo (0) non rientra nell'intervallo consentito.
Intervallo di costo consentito: [1-10000]
Errore alla riga 2
Non e' stato possibile caricare il file o parti di esso.

Process returned 0 (0x0)   execution time : 1.316 s
Press any key to continue.
```

Figura 2.14: Underflow del costo



```
"C:\Users\114776\Desktop\Universit  Parthenope Napoli\III anno\Algoritmi e strutture dati\Progetto\LineeNotturme\bin\Debug\Li...
Non e' stato possibile creare l'arco 11 8
Il suo costo (10001) non rientra nell'intervallo consentito.
Intervallo di costo consentito: [1-10000]
Errore alla riga 2
Non e' stato possibile caricare il file o parti di esso.

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

Figura 2.15: Overflow del costo