UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

SCUOLA INTERDIPARTIMENTALE DELLE SCIENZE, DELL'INGEGNERIA E DELLA SALUTE

INFORMATICA

PROGETTO ALGORITMI E STRUTTURE DATI

Traccia III

Proponente:
Ferraro Dominick **0124002048**

Data di Consegna:
05/12/2022

Anno Accademico:
2022 – 2023

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

Indice

RBGRAPH

DESCRIZIONE DEL 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. Dato un file di input contenente il grafo, costruire un RBGraph corrispondente ed effettuare le seguenti operazioni:

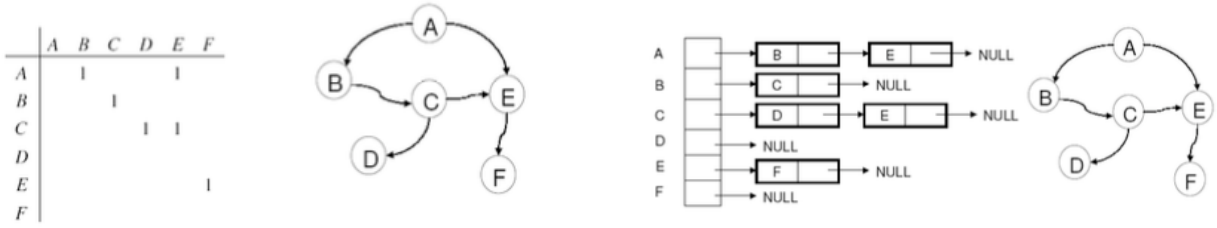
a. **AddEdge(i,j)**
b. **RemoveEdge(i,j)**
c. **FindEdge(i,j)**
d. **BFS(s).**

Il file di input contiene nel primo rigo due numeri interi, *0<=N<=1000 e 0<=M<=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 menù da cui sia possibile richiamare le suddette operazioni.

DESCRIZIONE STRUTTURE DATI

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 **orientato,** un arco è una coppia ordinata (v,w) di nodi.
Un grafo puó essere rappresentato da una matrice di adiacenza o una lista di adiacenza.

Matrice di adiacenza **Liste di Adiacenza**



In questo progetto una lista di adiacenza di un nodo è rappresentato da un Albero Red & Black.

ALBERO RED & BLACK

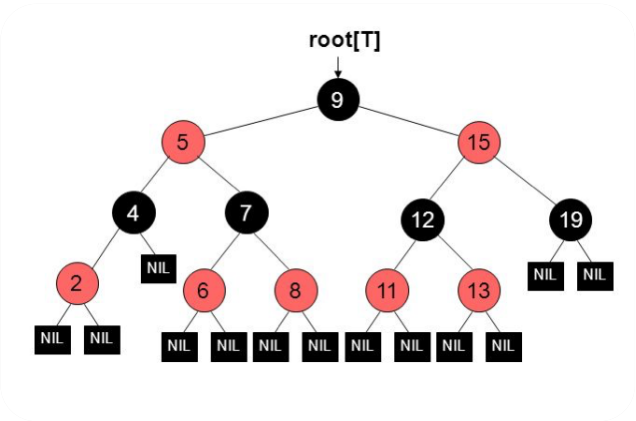
Un albero Red-Black è un particolare tipo di albero binario di ricerca **self-balancing.**
Le operazioni di ricerca, inserimento e cancellazione hanno complessità O(log n), dove n è il numero degli elementi dell'albero. Questa complessità rende molto diffuso il loro utilizzo in applicazioni realtime e in videogiochi.

Un Albero Red & Black ha diverse caratteristiche:

a. Le foglie sono costituite da nodi **null.**
b. Le chiavi sono mantenute solo nei nodi interni dell'albero.

Inoltre ci sono 5 proprietà essenziali da rispettare quando si usa un Red & Black:

1. Ogni nodo è colorato di **rosso** o **nero.**
2. La radice è **nera.**
3. Le foglie sono **nere.**
4. Se un nodo è **rosso** allora i suoi figli sono **neri.**
5. Ogni percorso da un nodo a una foglia ha la stessa **black-altezza (b-altezza).**



Ogni **nodo Red & Black** ha:

i. Puntatore al padre.
ii. Puntatore al figlio destro.
iii. Puntatore al figlio sinistro.
iiii. **Colore.**
iiiii. Chiave.
iiiiii. Dati satellite.

FORMATO DATI IN INPUT / OUTPUT

I dati di input sono estrapolati 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 due numeri divisi da uno spazio che rappresentano il nodo sorgente e quello di destinazione.
I dati in input del file devono essere numeri positivi.

In output avremo le elaborazioni delle funzioni offerte all'utente.

DESCRIZIONE ALGORITMO

Per quanta riguarda i codici di RBTree, GNode, RBNode, la maggior parte sono implementazioni giá affrontate nel corso e quasi tutte sono ispirati agli pseudocodici forniti dal Docente.
Particolarmente interessanti sono le funzioni sviluppate in RBGraph, che è il cuore del progetto.

La prima procedura che andiamo ad analizzare è AddEdge(i,j).
Come suggerisce il nome della function, questa ha il compito di creare un nuovo arco, aggiungendo quindi 2 nuovi nodi. Riceve in input un nodo sorgente e un nodo destinazione.
Se uno dei due nodi è nullo, allora non è possibile aggiungere un arco poiché supererebbe le dimensioni del grafo. Fatto questo controllo, si verifica se il nodo destinazione è giá nella lista di adiacenza del nodo sorgente, in tal caso non verrà effettuata l'operazione, in caso contrario viene creato un nuovo arco.

ADD_EDGE(source, dest)

1. *sourceNode = CREATE-NEW-GNODE(source)*
2. *destNode = CREATE-NEW-GNODE(dest)*
3. *IF destNode = null OR sourceNode = null*
4. *Return false*
5. *IF destNode ∈ ADJ(sourceNode)*
6. *Return false*
7. *Aggiungi destNode nella Lista di Adiacenza di sourceNode*
8. *Return true*

La prossima procedura che analizziamo è FindEdge(i,j). Prende in input un nodo sorgente e un nodo destinazione. Verifica che i 2 nodi non superino il range max del grafo e controlla se il nodo destinazione è presente nella lista di adiacenza del nodo sorgente. Se c'è ritorna true, oppure false.

FIND_EDGE(source, dest)

1. *Il dest >= size[nodes] OR source >= size[nodes]*
2. *Return false*
3. *Se il nodo dest* $\in$ *ADJ(sourceNode)*
4. *Return true*
5. *Else*
6. *Return false*

Analizziamo ora la procedura RemoveEdge(i,j).
La function prende in input un nodo sorgente e uno destinazione. Verifica se esistono i due nodi, se non esistono allora non possono essere cancellati e ritorna false, oppure li elimina e ritorna true.

REMOVE_EDGE(source, dest)

1. *IF source AND dest exists*
2. *Rimuovi l'arco dal grafo*
3. *Return true*
4. *Else*
5. *Return false*

L'ultima procedura che analizziamo è BFS(s).
La BFS o algoritmo di ricerca in ampiezza, è un algoritmo di ricerca per i grafi che partendo dalla sorgente *s*, scopre tutti i vertici raggiungibili da essa. L'algoritmo scopre tutti i vertici a distanza *k* da *s* prima di scoprire quelli a distanza *k+1*.

Come la DFS, anche la BFS colora i vertici:

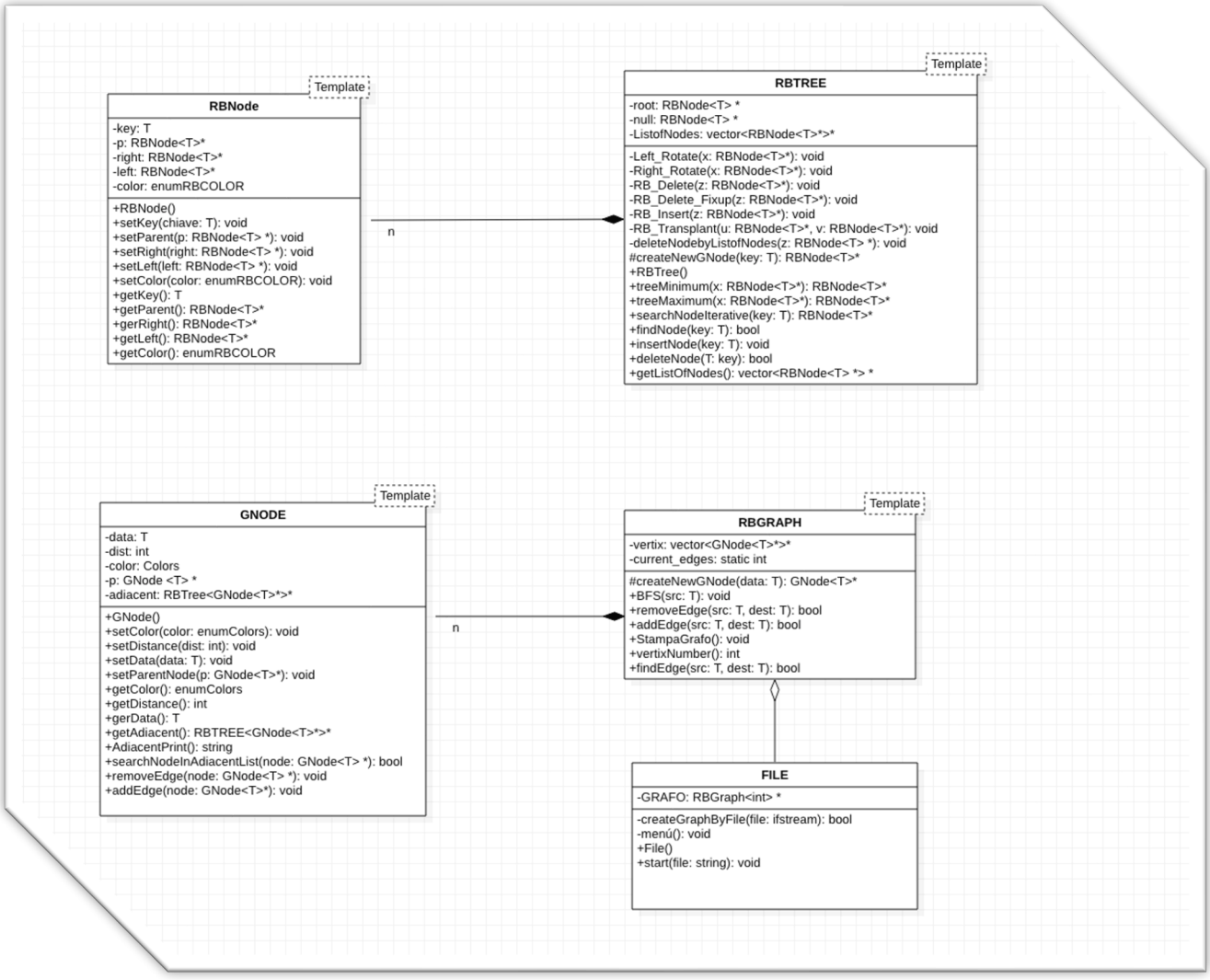
a. Inizialmente tutti i vertici sono bianchi.
b. Un vertice scoperto viene colorato di grigio
c. Quando tutti i vertici adiacenti di un nodo sono stati scoperti, questo diventa nero.

BREADTH-FIRST-SEARCH(V, E, s)

1. $\forall\ u \in V[G] - \{s\}$
2. $color[u] \leftarrow white$
3. $\pi[u] \leftarrow null$
4. $d[u] \leftarrow \infty$
5. $color[s] \leftarrow gray$
6. $\pi[s] \leftarrow null$
7. $d[s] \leftarrow 0$
8. $Q \leftarrow \emptyset$

9. Enqueue(Q, s)
10. WHILE (Q != ∅)
11. u <- Dequeue(Q)
12. ∀ v ∈ Adj[u]
13. If color[v] = white
14. color[v] ←gray
15. π[v] ← u
16. d[v] ← d[u] + 1
17. Enqueue(Q, v)
18. color[u] ← black

CLASS DIAGRAM



Il progetto è composto da 5 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. Oltre a queste operazioni, dispone dell'interfaccia che verrà visualizzata dall'utente per permettergli di fare diverse operazioni.

2. RBGraph: è la struttura dati portante che consente di memorizzare un grafo orientato, la cui lista di adiacenza di ogni nodo è rappresentata da un albero Red & Black. Tra le operazioni principali abbiamo *BFS, addEdge, FindEdge, removeEdge.*

3. GNode: è la classe dei singoli nodi del grafo orientato.

4. RBTREE: classe che rappresenta un albero Red & Black con tutte le operazioni classiche.

5. RBNode: classe che rappresenta un nodo dell'albero Red & Black.

RELAZIONE TRA LE CLASSI

La classe **RBTree** e **RBNode** hanno una relazione di **composizione** poiché se rimuovessimo la classe **RBTree**, allora **RBNode** non avrebbe molto senso di esistere.

Lo stesso vale per le due classi **RBGraph** e **GNode**, anche qui vi è una relazione di **composizione**. Per quanto riguarda la classe **File,** ha una relazione di **aggregazione** con **RBGraph** poiché usa quest'ultima per alcune operazioni offerte.

STUDIO COMPLESSITA'

Poiché stiamo utilizzando come base un albero Red & Black abbiamo una complessità favorevole, come giá descritto in precedenza.
Abbiamo detto che un albero Red & Black ha complessità O(*log* n).
Analizzando le funzioni portanti allora possiamo tirare le somme dei nostri algoritmi:

- AddEdge(i,j) inserisce il nodo destinazione nella lista di adiacenza del nodo sorgente, facendo uso dell'albero Red & Black allora il costo sarà O(*M)* dove M è l'insieme di archi del grafo.

- RemoveEdge(i,j) utilizza un'operazione di cancellazione di un albero Red & Black, quindi il costo sará O(*log* M). [ricordando che O(lg n) è il costo per la cancellazione]

- FindEdge(i,j) utilizza un'operazione di ricerca di un albero Red & Black, quindi il costo sará O(*log M).* [ricordando che O(lg n) è il costo per la cancellazione]

- BFS(s) ha una complessitá *O*(N+M) dove N è l'insieme dei vertici e M l'insieme di archi del grafo.

Le altre procedure di "contorno" non hanno una complessitá 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 notificherá 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, ecco la schermata iniziale dove l'utente puó fare le sue scelte, basterà inserire un numero tra quelli a schermo.

```
                ++ INSERIRE UN NUMERO PER UN'OPERAZIONE ++
[0] EXIT
[1] AddEdge(i,j)
[2] RemoveEdge(i,j)
[3] FindEdge(i,j)
[4] Visita in Ampiezza [BFS(s)]
[5] Stampa del Grafo
[DIGIT]:
```

SCELTA [1]

Con la scelta 1 l'utente puó aggiungere un arco, scegliendo il nodo di partenza e quello di destinazione, se nel caso l'arco è già nel grafo, il programma stamperà un avviso e non inserirà l'arco.

```
Hai scelto: 1
              ****** ADD EDGE *****
Digita il nodo sorgente per aggiungere un collegamento: 10
Digita il nodo destinazione per aggiungere un collegamento: 7
[SUCCESS] Arco (10 7) aggiunto.
```

Se riproviamo a inserire lo stesso arco, o un altro giá esistente, ecco l'alert:

```
[WARNING] Non é possibile creare l'arco (10 7), giá esiste.
```

SCELTA [2]

Con la scelta 2 l'utente puó rimuovere un arco, scegliendo il nodo di partenza e quello di destinazione, se l'arco non esiste nel grafo, il programma stamperà un avviso e non effettuerà l'eliminazione.

```
Hai scelto: 2
              ****** REMOVE EDGE *****
Digita il nodo sorgente per rimuovere un collegamento: 0
Digita il nodo destinazione per rimuovere un collegamento: 1
[SUCCESS] Arco (0 1) rimosso.
```

Se proviamo a cancellare un arco che non esiste, ecco l'alert:

```
[WARNING] Impossibile rimuovere arco (0 6): non esiste.
```

SCELTA [3]

Con la scelta 3 l'utente puó verificare se esiste un arco, scegliendo il nodo di partenza e quello di destinazione, se l'arco non esiste nel grafo, il programma stamperà un avviso e notifica che l'arco non esiste.

```
Hai scelto: 3
              ****** FIND EDGE *****
Digita il nodo sorgente per la ricerca: 10
Digita il nodo destinazione per la ricerca: 8
[SUCCESS] L'arco (10 8) é presente nel grafo.
```

Nel caso l'arco non esista, ecco l'alert:

```
              ****** FIND EDGE *****
Digita il nodo sorgente per la ricerca: 10
Digita il nodo destinazione per la ricerca: 9
[WARNING] L'arco 10 9 non é presente nel grafo.
```

SCELTA [4]

Con la scelta 4 l'utente puó effettuare la visita in ampiezza, BFS. Basta inserire la sorgente e viene stampata la soluzione. Se viene inserita una sorgente maggiore della dimensione del grafo, viene notificato e si stoppa l'operazione.

```
[DIGIT]: 4
              ****** BREADTH-FIRST SEARCH *****

Inserire il nodo sorgente per la Visita in Ampiezza [BFS]: 4
BFS in corso...
BFS: 4 3 5 1 2 9 6 7 8 10
```

Ecco cosa stampa il programma se la sorgente è maggiore:

```
La sorgente é maggiore della dimensione del grafo
```

SCELTA [5]

Con la scelta 5 l'utente puó stampare il grafo. Ovviamente la struttura dati viene aggiornata a ogni operazione effettuata: l'utente potrebbe eliminare un arco e dopo puó stampare il nuovo grafo.

```
[DIGIT]: 5
                ****** STAMPA GRAFO *****
0 ~~> 1
0 ~~> 4
0 ~~> 2
1 ~~> 3
1 ~~> 6
2 ~~> 4
3 ~~> 1
3 ~~> 5
```

Quando viene effettuata un'operazione, il programma chiede all'utente se vuole continuare effettuando una nuova operazione oppure stopparsi:

```
Vuoi fare un'altra operazione? [Y/y per continuare. Qualsiasi lettera per uscire.]
y

                  ++ INSERIRE UN NUMERO PER UN'OPERAZIONE ++
```

```
Vuoi fare un'altra operazione? [Y/y per continuare. Qualsiasi lettera per uscire.]
n
[EXIT] Grazie per aver usato questo programma
```

Glossario

Il glossario ha lo scopo fondamentale di chiarire il gergo tecnico usato e di evidenziare eventuali sinonimie e omonimie.

Termine	Descrizione	Sinonimi	Omonimi
Nodo	Unità fondamentale di cu i grafi sono costituiti.	Vertice	-
Self Balancing	Albero la cui altezza, grazie a particolari operazioni di ripristino proprietà, è sempre bilanciata.	-	-
B-Altezza	Il cammino da un nodo a una foglia in un albero Red & Black ha sempre lo stesso numero di nodi neri.	Black-H	-
Alert	Notifica di sistema per avvisare una data situazione verificata.	Notifica	-