Grafo con le matrici

# Funzioni:

* Aggiungere arco.
* Rimuovere arco.
* Aggiungere nodo.
* Altro in futuro, ora memorizziamo solo i dati.

# Ulteriori consegne possibili:

* Confrontare tra C++ e Java le possibili librerie.
* Gli oggetti contengono dei dati.

# Cosa fare/Uso pratico d’esempio:

Fare una mappa sentieristica, dove i nodi sono le località, e gli archi le direzioni da prendere per arrivare a destinazione.

Per capire meglio, cercare una mappa sentieristica su internet.

## Inizializzazione:

Creo all’inizio una matrice delle adiacenze, ad ogni riga e colonna corrisponde un valore pari a 0 se non c’è un arco, e 1 se invece presente.

In Java può essere creato tramite un ArrayList con al suo interno altrettante ArrayList in modo da creare una lista a matrice.  
Oppure si può sia in Java che C++ creare un vettore a matrice, ma deve essere dichiarato in precedenza con una dimensione massima prevista.

Per capire ad ogni riga quale nodo corrisponde, uso una lista di oggetti.

## Esempio oggetto della lista:

Contiene:

* Delle informazioni sulla località per esempio:
  + Località.
  + Altitudine.
* A quale riga e colonna corrispondono l’oggetto.

## Aggiungere un arco:

Per aggiungere un arco, si va alla cella nella matrice delle adiacenze che corrisponde all’arco tra il primo e il secondo nodo, e si imposta il valore 1.

Per trovare la cella corretta, la posizione nella lista dell’elemento (in ordine con gli altri nodi) deve corrispondere con quello nella matrice, questo significa che l’elemento alla posizione 2 (a partire da 0 -> 1 -> 2) nella lista, in una matrice da 3 elementi, avrà come coordinate delle adiacenze:

V[2][0], V[2][1] e se stesso V[2][2].

Se viene aggiunto un arco tra l’elemento 2 e l’elemento 1, la casella V[2][0] sarà impostata a 1, ovviamente se necessario si dovrebbe fare la ricerca del nodo da modificare nella lista dei nodi e poi tramite posizione modificare la matrice opportuna.

## Rimuovere un arco:

Per rimuovere un arco, si va alla cella nella matrice delle adiacenze che corrisponde all’arco tra il primo e il secondo nodo, e si imposta il valore 0.

Questo procedimento è simile all’aggiunta di un arco, solamente che si imposta a 0 un valore e se tutti i valori nella matrice delle adiacenze sono a 0, il nodo diventa irraggiungibile.

## Aggiungere nodo:

Si aggiunge in caso di grafo non orientato sia una riga che una colonna alla matrice, incrementandola in modo da avere la possibilità di rappresentare tutti i nuovi possibili archi.

|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B | C |
| A | 0 | 1 | 0 |
| B | 1 | 0 | 1 |
| C | 1 | 1 | 0 |

Diventa con l’aggiunta di D:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0 | 1 | 0 | 0 |
| B | 1 | 0 | 1 | 0 |
| C | 1 | 1 | 0 | 0 |
| :D | 0 | 0 | 0 | 0 |

Dove sarà necessario aggiungere almeno una adiacenza per non lasciare il nodo separato dal grafo, magari potrebbero esserci due metodi per aggiungere il nodo, di cui uno che specifica un nodo adiacente, e uno non.

Il nodo sarà sempre aggiunto alla FINE della lista e la sua posizione nella matrice sarà la lunghezza della lista – 1 (.lenght o .size() ritorna un valore pari al numero di elementi presenti nella lista e deve essere convertito in una posizione binaria adatta ai vettori).

Questo nuovo nodo deve essere aggiunto anche alla lista dei nodi.

# RIMOZIONE DI UN NODO:

Eseguo in un doppio ciclo la copia della matrice delle adiacenze in una temporanea con dimensione più piccola di uno rispetto all’originale, ogni volta che si esegue un ciclo, si controlla se la colonna o la riga è quella da eliminare e nel caso NONA si copia il valore nel vettore temporaneo.

Alla fine dell’operazione si rende la matrice temporanea la nuova definitiva, e si rimuove il nodo dalla lista dei nodi.

# Oggetto/Classe/struct nodo:

Il nodo è composto da, come da esempio:

* Nome/Cartello/Paese.
* Altitudine.
* Opzionalmente le coordinate.

### Attributi:

string nome;  
int difficolta;  
int altitudine;

### Prototipo metodi:

string getNome();  
int getDifficolta();  
int getAltitudine();  
void setNome(string nome);  
void setDifficolta(int difficolta);  
void setAltitudine(int altitudine);

// COSTRUTTORE:  
Nodo(string nome, int difficolta, int altitudine);  
Nodo(string nome, int difficolta);  
Nodo(string nome);  
Nodo();

# Visite al grafo:

### Visita in ampiezza -> Breadth first search (BFS)

Consiste nel visitare prima tutti i nodi sullo stesso livello (riga) e solo successivamente passare ai nodi allo stesso livello del primo e così via, ogni nodo visitato viene marchiato come “visitato” in modo da non ripetere i cicli innutilmente.

Tramite l’ausilio di una LinkedList salvo i valori da visitare in un livello e man mano che li rimuovo in un ciclo li stampo.

Ogni volta che si scende ad un livello successivo, aggiungo alla lista tutti i nodi da visitare e successivamente saranno stampati (visitati) e aggiunti al vettore dei visitati, questi durante l’aggiunta saranno ignorati.

### visita in profondita -> depth first search (DFS)

Consiste nel visitare per primi i nodi scendendo fino a quello più in basso (sottonodo) stampando man mano il percorso.

Tramite un metodo ricorsivo, visito i nodi stampandoli, e per evitare di visitarli di nuovo li aggiungo ad un vettore dei nodi visitati, i nodi già visitati non ripeteranno l’operazione ricorsiva per scendere al sottonodo.