

# Progetto - “Archivio città e mappa”

## Relazione di Frasca Luigi

Matricola 232345

Lo scopo del progetto è riuscire a progettare in C++ un programma in grado di rappresentare un Archivio contenente i nomi delle città e le relative informazioni su di esse, oltre ad una Mappa contenente i possibili collegamenti tra le stesse precedentemente aggiunte in Archivio. Il programma interagisce con l'utente attraverso menu' a più livelli, per poter gestire la Mappa è necessario aver caricato le città in Archivio.

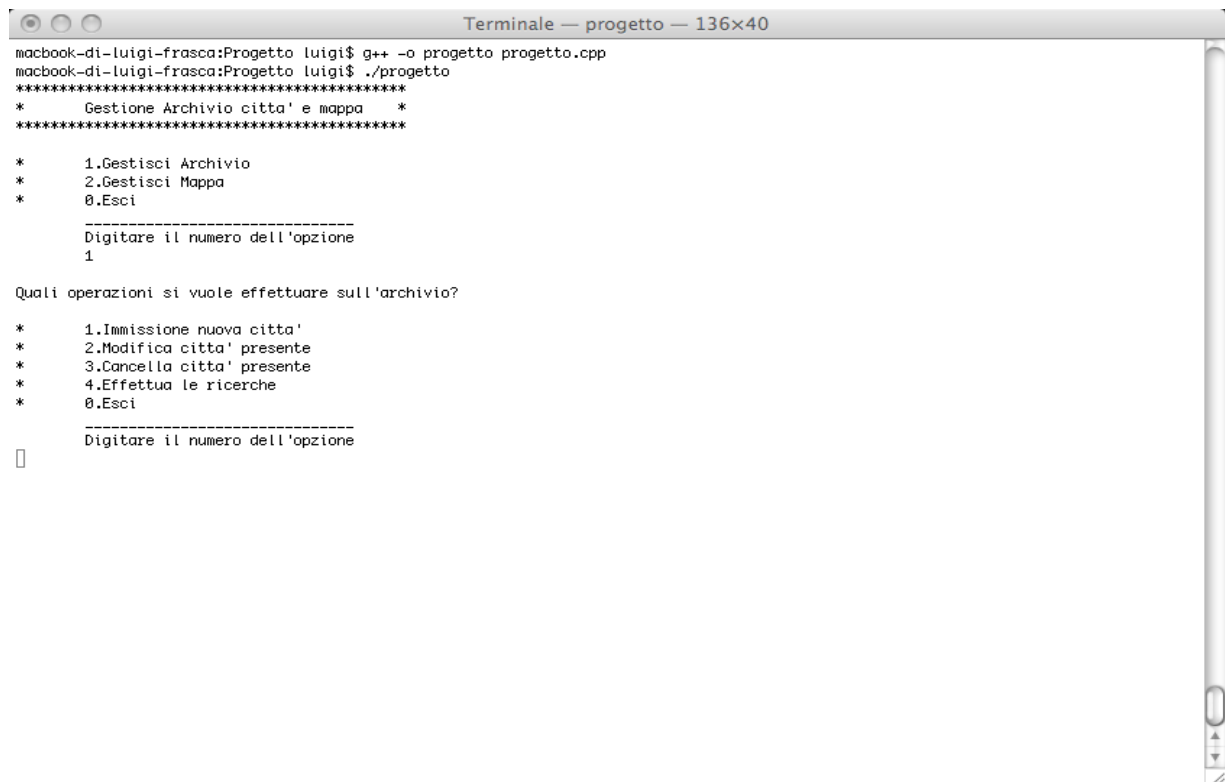
### Gestione Archivio

Archivio è costituito da due classi :

Classe Citta astrae il modello della città carpando informazioni sotto forma di attributi, in particolare nome, numero abitanti e le maggiori attrazioni turistiche. I metodi della classe si occupano di effettuare controlli sulle relative caratteristiche della stessa.

Classe Archivio può contenere tutte le informazioni relative ad una città. Inoltre, mette a disposizione dei metodi che permettano di aggiungerne delle altre, modificarne le caratteristiche, ricercarne una in particolare in base a determinati criteri, cancellarle dall'archivio. I dati vengono letti e salvati su file(“dati.txt”). La lettura è affidata al costruttore di classe mentre la scrittura è gestita dal distruttore.

screenshot\_archivio#1#apertura\_eseguibile



```
macbook-di-luigi-frasca:Progetto luigi$ g++ -o progetto progetto.cpp
macbook-di-luigi-frasca:Progetto luigi$ ./progetto
*****
*      Gestione Archivio città e mappa      *
*****

*      1.Gestisci Archivio
*      2.Gestisci Mappa
*      0.Esci
*
-----
Digitare il numero dell'opzione
1

Quali operazioni si vuole effettuare sull'archivio?

*      1.Immissione nuova città
*      2.Modifica città presente
*      3.Cancella città presente
*      4.Effettua le ricerche
*      0.Esci
*
-----
Digitare il numero dell'opzione
1
```

Ecco come si presenta il programma appena aperto.

Di seguito illustro alcuni metodi di particolare interesse da me utilizzati.

L'inserimento di una città in archivio è gestito dalla funzione riportata qui di seguito.

```
void Archivio::immissione(){ //public
    bool nuova;
    do{
        Citta temp;
        string attrazione;
        bool risp;
        cout<<"Inserisci nome della città da inserire"<<endl;
        cin.clear();
        cin.ignore(256, '\n');
        temp.setcitta(cin);
        cout<<"Inserisci la provincia di appartenenza della città (es.PR)"<<endl;
        temp.setprovincia(cin);
        cout<<"Inserisci il numero abitanti della città"<<endl;
        temp.setabitanti(cin);
        cout<<"Inserisci l'attrazione di questa città"<<endl;
        do{
            cin.ignore(256, '\n');
            getline(cin, attrazione);
            int i = temp.setatt(attrazione);
            //i contiene il numero di attrazioni inserite
            temp.setcount(i);
            cin.clear();
            if(i!=1){
                cout<<"Ci sono altre attrazioni che vuoi aggiungere?(1-si/0-no)"<<endl;
                cin>>risp;
                while(!ctrlInt()){
                    cout<<"Errore immissione,prego ripetere"<<endl;
                    cin>>risp;
                }
            }else{ break; }
        }while(risp);
        cout<<"sono state inserite:"<<temp.getcount()<<"attrazioni"<<endl;
        elencocitta.push_back(temp);
        cout<<"Vuoi inserire un'altra città?(1-si/0-no)\t"<<endl;
        cin>>nuova;
    }while(nuova);
}
```

La funzione immissione(), si occupa di inserire una nuova città nell'archivio con le relative informazioni. Vengono effettuati i controlli sull'input passando direttamente lo stream ai metodi set della classe Citta. E' possibile inserire fino a un massimo di 10 attrazioni per città. Ad ogni immissione sia delle attrazioni che delle città, viene chiesto se si vuole continuare con l'inserimento di altre città/attrazioni, l'utente deve digitare '1' in caso di risposta affermativa o '0' in caso di risposta negativa.

Di importante rilievo è la funzione di ricerca sottostringa.

```
int Archivio::ric_sottostringa(const string x){ //private
    dat_trovati.clear();
    ind_trovati.clear();
    int n_citta = elencocitta.size();
    cout << "città in archivio:\t" <<n_citta<<endl;
    int c=0;
    string app;
    size_t found;
    for(int i=0;i<n_citta;i++){
        app = elencocitta[i].getcitta();
        found=app.find(x);
        if(found!=string::npos) {
            c++;
            dat_trovati.push_back(app);
            ind_trovati.push_back(i);
            int k = dat_trovati.size();
            int j = ind_trovati.size();
        }
    }
    return c;
}
```

Il metodo ric\_sottostringa(const string x) prende come parametro una variabile di tipo stringa.

Usa due vector 'dat\_trovati' e 'ind\_trovati' dove andrà a memorizzare rispettivamente il nome della città trovata e l'indice che occupa all'interno dell'elenco.

Questi vector vengono puliti ad ogni chiamata di funzione per eliminare dati di ricerche precedenti. La funzione scorre tutto l'elenco

delle città presenti nell'archivio e quando trova un'occorrenza, salva il nome e l'indice della città incrementandone il contatore. Questo sarà restituito alla fine dello scorrimento, in modo da fornire il numero di occorrenze trovate.

## Un esempio di ricerca tramite sottostringa ...

screenshot\_archivio#2#ricerca\_sottostringa

```
*      1.Immissione nuova città'
*      2.Modifica città' presente
*      3.Cancella città' presente
*      4.Effettua le ricerche
*      0.Esci

-----
Digitare il numero dell'opzione
4
*      1.visualizza dati di una città(individuata tramite sottostringa)
*      2.visualizza città appartenenti ad una provincia
*      3.visualizza città con meno di n abitanti
*      0.Menu' precedente

-----
Digitare il numero dell'opzione
1
Inserisci il nome della città di cui si vogliono avere le informazioni
a
città in archivio:      3
                    0parma
                    1milano
                    2la spezia
Inserisci scelta
[]
```

qui l'utente deve scegliere una città digitandone l'indice.

Ecco come si presenta il programma nel caso l'utente volesse modificare relative informazioni riguardanti una città.

```
Inserisci il nome(o la sottostringa) della città che si vuole modificare
a
città in archivio:      4
                    0milano
                    1la spezia
                    2ancona
                    3bolzano
Inserisci scelta
1
Per la città' scelta,cosa si vuole modificare?
    1. numero abitanti della città
    2. attrazioni turistiche della città
    0. Esci
[]
```

L'utente può solamente modificare il numero di abitanti e le attrazioni turistiche di una città. Infine la classe Archivio permette di cancellare una città se e solo se non è presente un collegamento in Mappa che contenga quella città.

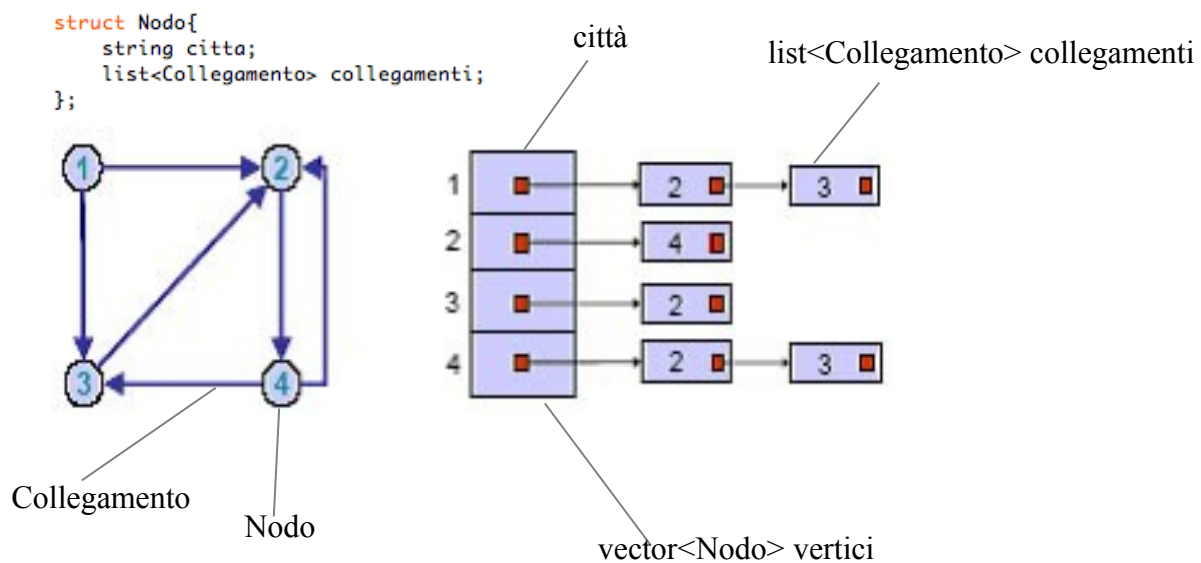
## Gestione Mappa

E' stata creata la classe Collegamento che si occupa di contenere al suo interno il nome della città collegata, il tipo di collegamento ed una variabile di tipo peso dove andiamo a memorizzare altre informazioni riguardanti il collegamento.

Nella classe Mappa è stata creata una struct Nodo che può contenere un vertice e tutti i suoi collegamenti.

Per rappresentare quindi una mappa con più vertici è stato creato un vector di tipo Nodo all'interno di mappa che si chiama vertici.

Ecco una rappresentazione pittorica di quanto è stato detto sopra



di seguito è riportato uno screenshot delle varie operazioni possibili nella gestione della mappa.

```
Quali operazioni si vuole effettuare sulla mappa?

*      1. Aggiunta alla mappa di una nuova città '*'
*      2. Aggiunta alla mappa di un nuovo collegamento '*'
*      3. Ricerca cammino minimo tra due città?
*      4. Cancellazione di una città dalla mappa
*      5. Cancellazione di un collegamento dalla mappa
*      0. Esci

-----
Digitare il numero dell'opzione
```

Una città può essere aggiunta alla mappa solo se è presente già nell'archivio.

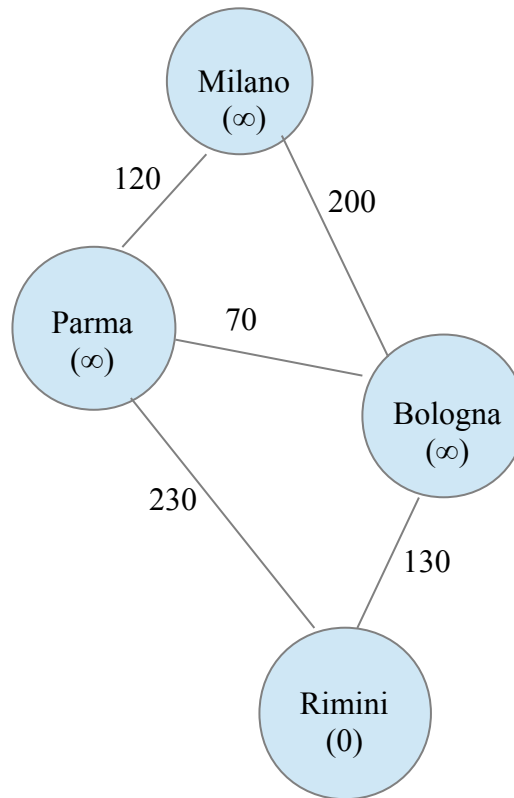
E' possibile aggiungere un collegamento tra due città, solo se entrambe sono state aggiunte in mappa.

Molto importante è il terzo punto, che permette di ricercare il cammino minimo tra due città e quindi di poter scegliere il percorso con il minor numero di chilometri possibili.

I cammini minimi sono particolarmente utili e vengono utilizzati in molti campi non solo in quelli informatici.

Il problema dei cammini minimi è stato risolto attraverso l'algoritmo di Dijkstra.

## Funzionamento dell'algoritmo di Dijkstra



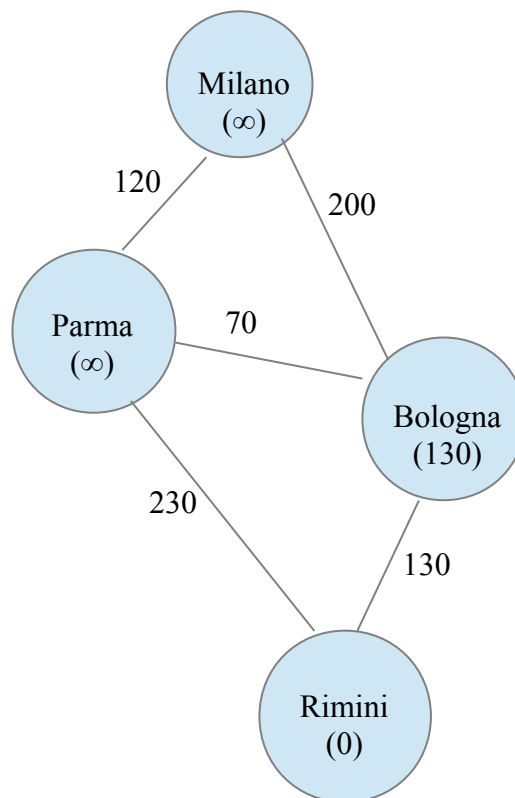
Tutti i nodi hanno un peso e tutti gli archi hanno un valore che equivale ai chilometri del collegamento tra le città.

Considerando che vogliamo partire da Rimini e arrivare a Milano con il percorso minore, l'algoritmo di Dijkstra pone all'inizio il peso della città di partenza a 0 e tutti gli altri a infinito.

Fatto questo inizia a valutare le città adiacenti a quella di partenza e sceglie quella che ha il collegamento minore.

In seguito aggiorna il peso della città su cui si è spostato che sarebbe la somma dell'arco più il peso della città precedente.

Quindi avremmo una situazione del genere:



Adesso fa la stessa cosa valutando gli adiacenti di Bologna, scegliendo Parma poiché ha il collegamento minore.

Alla fine il programma stamperà questo risultato:

Quali operazioni si vuole effettuare sulla mappa?

- \* 1. Aggiunta alla mappa di una nuova città\*
- \* 2. Aggiunta alla mappa di un nuovo collegamento\*
- \* 3. Ricerca cammino minimo tra due città\*
- \* 4. Cancellazione di una città dalla mappa\*
- \* 5. Cancellazione di un collegamento dalla mappa\*
- \* 0. Esci

-----  
Digitare il numero dell'opzione

3

Inserisci nome della città di partenza

rimini

Inserisci nome della città di arrivo

milano

il cammino minimo tra rimini e milano è di 320 km

il tragitto è

rimini

bologna

parma

milano

Infine il programma permette di cancellare un nodo se questo non ha collegamenti con altri nodi ed cancellare un collegamenti già esistente tra due nodi.

Di seguito riporto le funzioni utile alla realizzazione della mappa

#### **void inserimento()**

permettere l'inserimento di un nodo nella mappa,controllando che sia presente nell'archivio utilizzando la funzione privata isPresent().

#### **bool insertcolleg()**

permette l'inserimento di un collegamento tra due nodi,controllando che non sia già presente e che le città inserite esistano in mappa.

#### **void canc\_nodo()**

Cancella il nodo nella mappa se e solo se esso non ha collegamenti con altri nodi.

#### **void canc\_coll()**

Permette di cancellare un collegamento tra due nodi che lo hanno.

#### **void salvafile()**

Salva tutte le informazioni sul file,questo metodo è chiamato dal distruttore di classe.

#### **vector<string> separa(string x,char fine)**

A questo metodi gli vengono passati due parametri.

parametro stringa e un carattere, il metodo si occupa di dividere la stringa ogni qualvolta incontra il carattere passato e ritorna un vector in cui abbiamo le singole informazioni. Molto utile nella lettura del file.

#### **int ricerca\_nodo(string citta\_nodo);**

Controlla se un nodo è presente,ritorna -1 in caso di fallimento.

#### **bool exist\_coll(int primonodo,string nsnodo)**

vengono passati due parametri, l'intero è l'indice di dove si trova il primo nodo all'interno del vector vertici e il secondo parametro è la stringa del secondo nodo.

Il metodo si posiziona all'indice passato in vertici e scorre la lista di collegamenti di quell'indice fino a trovare l'occorrenza,ritornando 0 in caso di fallimento e 1 in caso di successo.

#### **void cam\_min()**

E' la funzione che si occupa di trovare i percorsi minimi da un nodo di partenza a uno di arrivo utilizzando l'algoritmo di Dijkstra spiegato precedentemente.

Salvando tutti i percorsi tra il nodo di partenza e il nodo di arrivo.

### **Conclusione**

Le relative classi del programma sono controllate dal main() che si occupa di caricare il menuprincipale attraverso la funzione omonima, in base alla scelta che ritorna la funzione carica i rispettivi costruttori di Archivio e Mappa.

