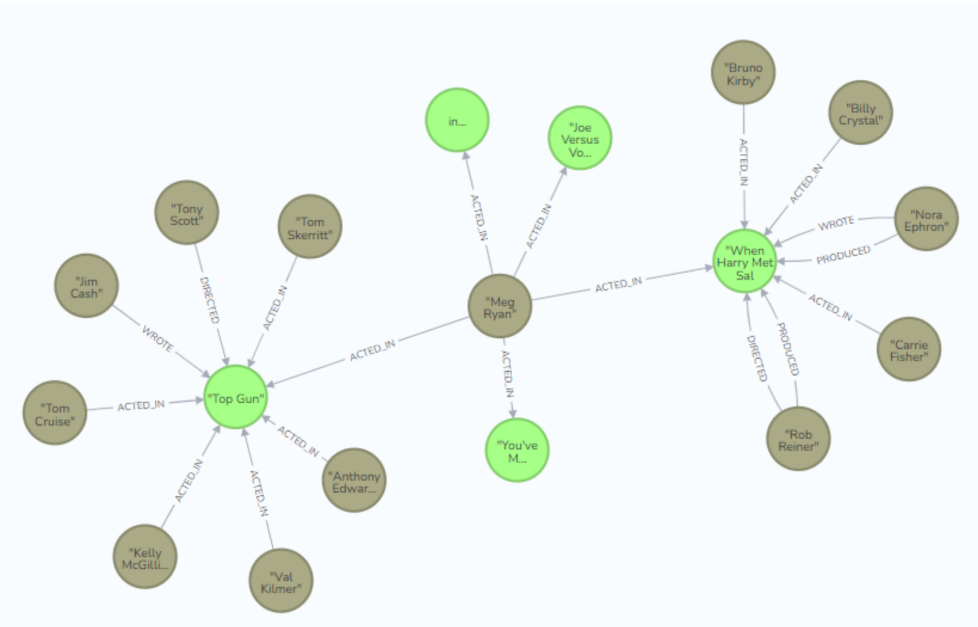
Laboratorio

Il Tipo di dato che è stato scelto per questo laboratorio è il TDD Graph, dopo un'attenta analisi delle informazioni che sono state fornite dal testo di laboratorio.



La struttura dati utilizzata sono le liste di adiacenza, dove i vertici sono memorizzati in una lista, dal momento in cui i nodi non sono etichettati con numeri naturali ma con stringhe.

Implementazione

`List`

Per le liste si è utilizzata una list-array viste antecedentemente a lezione, in più implementando una funzione booleana `isPresent` che ritorna vero se un elemento è presente nella lista.

`movies`

Per il file movies sono state utilizzate due struct:

```
// Mezzo arco, non tiene il nodo sorgente
struct halfEdgeNode{
  VertexNode *vertPtr;
  EdgeType role; // etichetta dell'arco
  halfEdgeNode* next; // puntatore al mezzo arco successivo
};
```

```
struct movies::VertexNode{
  Label label;
  NodeType type; // tipo : ACTOR o MOVIE
  halfEdgeNode *adjList;
  VertexNode *next;
};
```

Le liste di adiacenza sono state utilizzate con i puntatori al vertice, poiché per la specifica di alcune funzioni, come `coActors`, `actorsProducedBy`, `actorsDirectedBy` l'uso dei puntatori risultava più efficiente. Inoltre si è scelto di togliere la variabile `bool visited` da VertexNode, perchè la visita dei nodi è stata trattata differentemente nelle funzioni.

Oltre alle funzioni fornite nel file `movies.h` sono state aggiunte delle funzioni ausiliarie, al fine di rendere più efficienti le operazioni:

- `isEmpty(const MovieDB& mdb)` : restituisce vero se la funzione è vuota
- `getVertex(Label l, const MovieDB& mdb)` : restituisce il puntatore al vertice
- `isVertexInMovie(Label l, const MovieDB& mdb)` : restituisce vero se il vertice esiste nel database film
- `isEdgeInGraph(Label from, Label to, const MovieDB& mdb, const EdgeType t)` : ritorna vero se l'arco e' gia' presente nel grafo
- `addHalfEdge(Label from, Label to, const EdgeType t, MovieDB &mdb)`: aggiunge il "mezzo edge" alla lista di adiacenza
- `printadj(Label l, const MovieDB& mdb)`: Funzione ausiliaria che stampa lista di adiacenza
- `void PersonActor(const MovieDB &mdb, Label l, EdgeType t, lista::list & lst)`: Funzione ausiliaria per creare la lista degli attori che sono stati diretti-prodotti-coattori di un film
- `BaconNumberRecur(const MovieDB &mdb, lista::List &listaAttoriLivello, lista::List &listaAttorVisitati, int &count, bool &isFound)`: Funzione ausiliaria che mi ritorna la visita alle liste dei coattori

Complessità Funzioni

La complessità delle funzioni in `list` , considerando i casi peggiori, è la seguente:

dove:

- n è il numero degli elementi

clear(List& l)	$\Theta(1)$
set(int pos, Elem e, List& l)	$\Theta(1)$
add(int pos, Elem e, List& l)	$\Theta(n)$
addBack(Elem e, List& l)	$\Theta(n)$

addFront(Elem e, List& l)	$\Theta(n)$
removePos(int pos, List& l)	$\Theta(n)$
createEmpty()	$\Theta(1)$
get(int pos, const List& l)	$\Theta(1)$
isEmpty(const List& l)	$\Theta(1)$
size(const List& l)	$\Theta(1)$
isPresent(Elem e, const List& l)	$\Theta(n)$

La complessità di tutte funzioni di `movies` , considerando i casi peggiori, è la seguente:

dove:

- n è il numero dei nodi
- m il numero degli archi

isEmpty(const MovieDB& mdb)	$\Theta(1)$
getVertex(Label l, const MovieDB& mdb)	$\Theta(n)$
isVertexInMovie(Label l, const MovieDB& mdb)	$\Theta(n)$
isEdgeInGraph(Label from, Label to, const MovieDB& mdb, const EdgeType t)	$\Theta(n)$
addHalfEdge(Label from, Label to, const EdgeType t, MovieDB &mdb)	$\Theta(n)$
printadj(Label l, const MovieDB& mdb)	$\Theta(n)$
void PersonActor(const MovieDB &mdb, Label l, EdgeType t, lista::list & lst)	$\Theta(n+m)$
BaconNumberRecur(const MovieDB &mdb, lista::List &listaAttoriLivello, lista::List &listaAttorVisitati, int &count, bool &isFound)	$\Theta(n+m)$
createEmpty()	$\Theta(1)$
addVertex(MovieDB &mdb, const Label l, const NodeType t)	$\Theta(n)$
addEdge(MovieDB &mdb, const Label from, const Label to, const EdgeType t)	$\Theta(n)$
numNodesPerType(const MovieDB &mdb, NodeType t)	$\Theta(n)$
numEdgesPerType(const MovieDB &mdb, EdgeType t)	$\Theta(n)$
coActors(const MovieDB &mdb, Label l)	$\Theta(n+m)$
actorsProducedBy(const MovieDB &mdb, Label l)	$\Theta(n+m)$
actorsDirectedBy(const MovieDB &mdb, Label l)	$\Theta(n+m)$
baconNumber(const MovieDB &mdb, Label l)	$\Theta(n+m)$
printDB(const movies::MovieDB &mdb)	$\Theta(n)$

Complessità funzioni implementate

Le query richieste ovvero le liste di `coActor`, `ActorProducedBy`, `ActorDirectedBy` sono state tutte implementate dalla funzione `PersonActor`, che ha una complessità di $\Theta(n+m)$ ciò deriva dal fatto che:

- n consegue dalla complessità di `getVertex` che è la funzione che richiamo per puntare al vertice
- m è dato da $\Theta(\sum(u_i\ adiacent)\delta(u_i)+\delta(v))$ che è ≤m

La complessità temporale delle funzioni `BaconNumber` e `BaconNumberRecur` è $\Theta(n+m)$, dove *n* è il numero di attori e attrici nel grafo, e *m* il numero degli archi nel grafo; questa è una visita BFS (spiegato nell'implementazione di Bacon Number che è leggermente diversa) implementata con le liste di adiacenza dei soli co-attori.

Implementazione di BaconNumber

Per implementare la funzione `BaconNumber` è stata utilizzata la funzione ausiliaria `BaconNumberRecur` che ha come parametri:

- il database movie
- la lista di Attori del Livello → la lista dei coattori del livello i-esimo
- la lista di Attori Visitati → la lista degli attori visitati
- contatore → che ritornerà il baconNumber
- booleano found (trovato)

La funzione `BaconNumberRecur` prende in input il nome di una persona indicato nella funzione `BaconNumer` e restituisce il suo Bacon number. Prima ancora di chiamare la funzione `BaconNumberRecur`all'interno della funzione `BaconNumer` se la persona in input è 'Kevin Bacon' viene restituito il valore 0 mentre se la persona in input non è fra gli attori presenti viene restituito il valore 'No Bacon Number'. In caso contrario `BaconNumer` richiama la funzione `BaconNumberRecur` passando in input il nome di una persona.

La funzione `BaconNumberRecur` cerca il Bacon Number della persone indicata nella chiamata originale di `BaconNumber` in modo ricorsivo attraverso una lista di attori, inizialmente composta dalla sola persona in input; all'inizio il Bacon Number vale 0 e viene incrementato progressivamente con le chiamate ricorsive. Se all'i-esima chiamata di `BaconNumberRecur` nella lista degli attori trovati è presente Kevin Bacon viene restituito `i` come "Bacon number", altrimenti la funzione cerca i `coActors` che hanno recitato in un film con tutte gli attori presenti nella lista in input ed effettua la chiamata ricorsiva passando una nuova lista contenente i coActors individuati: se fra i coActors esistono degli attori già presenti nelle liste utilizzate nelle chiamate precedenti a `BaconNumberRecur` questi non vengono inseriti nella nuova lista; questo processo continua fino a quando viene trovato Kevin Bacon fra gli attori, oppure si esaurisce la lista degli attori da visitare, in quest'ultimo caso la funzione restituisce il valore 'No Bacon Number'.

La funzione `BaconNumber` prende in input il nome di una persona e restituisce il suo Bacon number, essa crea inizialmente un grafo di attori e attrici che hanno recitato insieme in un film. Successivamente, utilizza un algoritmo di ricerca in ampiezza simile al BFS, anche se non si vanno ad eliminare gli elementi visitati ma si aggiornano le liste. Questo algoritmo permette di trovare il percorso più breve dalla persona di input a Kevin Bacon. Il Bacon number della persona di input è la lunghezza del percorso più breve.

> **Note:**
>
> Nel codice a riga 162 del file `movies.cpp` sono state commentate le stampe, nel caso si volesse vedere quali attori sono stati utilizzati per calcolare il bacon number.

File Modificati

Nel file `main.cpp` poiché il codice è stato implementato su un sistema operativo macOS, il programma non compilava, allora si è modificata la stampa sul terminale. Non sono state apportate ulteriori modifiche ad altri file.

Compilazione codice:

Scrivere il percorso della cartella su cui si è salvato il file `cd`…..

Compilare con il seguente comando: `g++ -Wall *.cpp`

Per eseguire i test : `python3 run_test.py ./a.out`