Querying and Traversing Graphs

Part 2: Using real data; building, querying, and traversing the graph.

For this part of the project, you will choose an input dataset among the options listed below and you will build a labeled graph (using the same labeled graph code from part 1) with the data in the dataset.

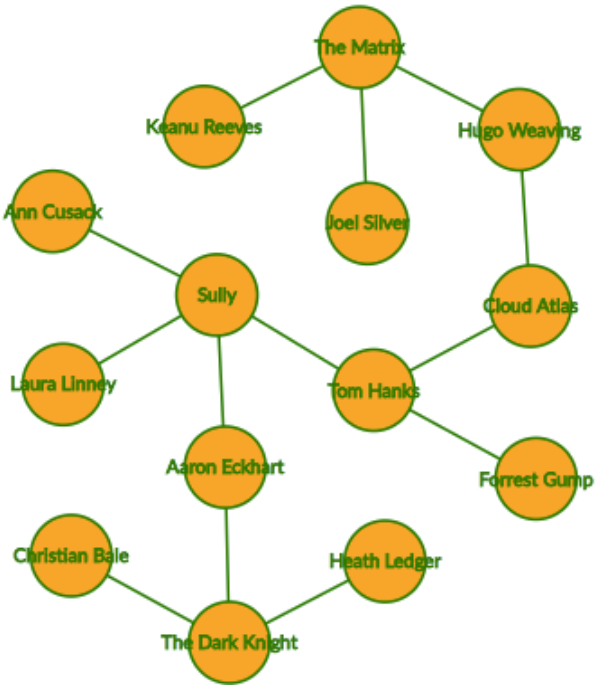
Datasets

Movies and Actors – This dataset, extracted from the Internet Movie Database (IMDB: www.imdb.com), contains a list of movies and their performers (actors and actresses). Each line in the *movies.txt* file contains a movie title followed by the list of the performers in the movie. Entries are separated by a forward slash "/". Each line in the file looks as follows:

```
Matrix, The (1999)/Arahanga, Julian/Aston, David/Ball, Jeremy/Brown, Tamara/Butcher,
Michael/Chong, Marcus/Dodd, Steve/Doran, Matt/Fishburne, Laurence/Foster, Gloria/Goddard,
Paul/Gordon, Denni/Gray, Marc/Harbach, Nigel/Johnson, Fiona/Lawrence, Harry/Ledger,
Bernard/McClory, Belinda/Moss, Carrie-Anne/Nicodemou, Ada/O'Connor, David/Pantoliano,
Joe/Parker, Anthony Ray/Pender, Janaya/Quinton, Luke/Reeves, Keanu/Scott, Chris/Simper,
Robert/Taylor, Robert/Tjen, Natalie/Weaving, Hugo/White, Adryn/Witt, Elenor/Witt,
Rowan/Woodward, Lawrence/Young, Bill
```

The file *movies_short.txt* has a snapshot of *movies.txt* so that you can work with a small file while developing and testing and not take too long every time you run your program. Work with the short file until you are done and ready to test it with the large one.

In the graph that you will build from this dataset, the vertices correspond to actors and movies, and the edges connect actors with the movies they have appeared in. For example:



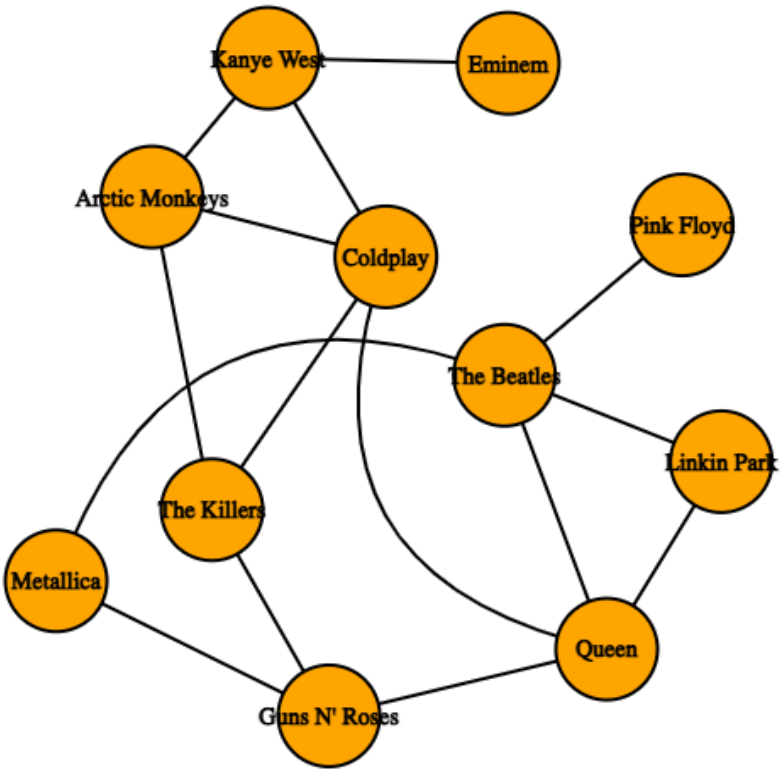
Favorite Musical Artists – This dataset, extracted from Spotify (www.spotify.com), contains the favorite musical artists of 1000 users. Each line in the original file corresponds to the preferences of one user and contains the list of his/her favorite musical artists, for example:

```
R.E.M., The Beatles, My Bloody Valentine, Arcade Fire, David Bowie, Rammstein, Pavement,
Pixies, The Smiths, Daft Punk, The Rolling Stones, Tool, Radiohead, The Verve, Led Zeppelin,
The Clash, Metallica, AC/DC, Guns N' Roses, Queen & David Bowie, Elton John, Clint Mansell,
Neil Norman and His Cosmic Orchestra, The Black Keys, The Who
```

You will not need the original file. Instead, you are provided with the *artists-pairs.txt* file, which was produced from the 1000 preferences lists in the original file and contains a list of pairs of artists that appear together in at least two of the preferences lists (and so, there is some implicit connection between them since at least 2 users have both on their preferred list). The names in each line are separated by a comma ",".

The file artists*_pairs_short.txt* has a snapshot of *artists_pairs.txt* so that you can work with a small file while developing and testing and not take too long every time you run your program. Work with the short file until you are done and ready to test it with the large one.

In the graph that you will build from this dataset, the vertices are musical artists, and the edges connect musical artists that appear together in at least two user's preferences list. For example:



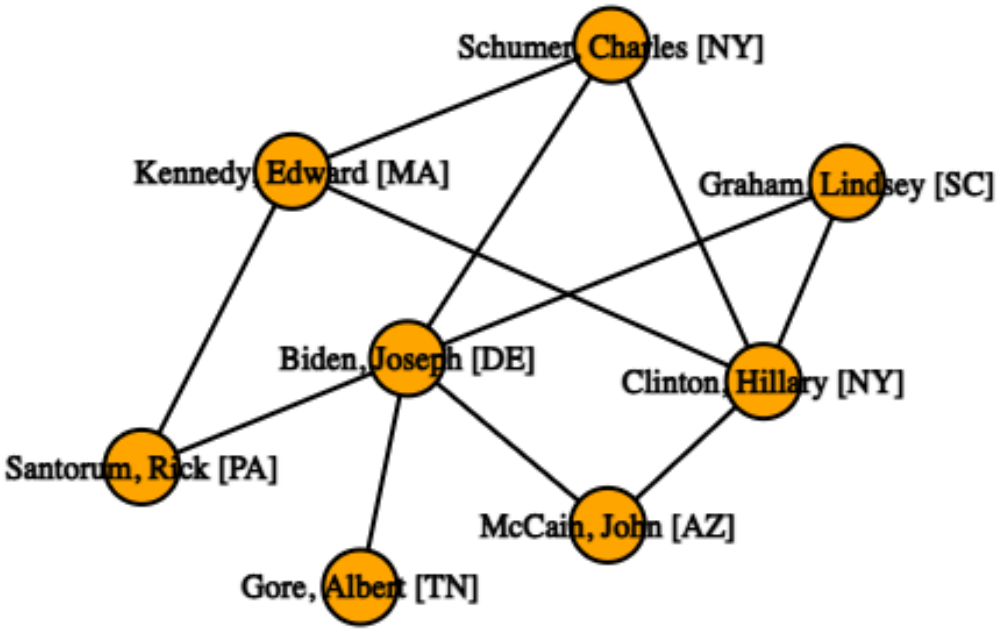
Legislative Co-sponsorship in the U.S. House and Senate – This dataset contains a list of sponsors and co-sponsors of legislative bills put forth in both the House of Representatives and the Senate. The dataset was produced from James Fowler's data (https://www.cs.cornell.edu/~arb/data/congress-bills). Each line in the original file corresponds to a legislative bill and contains the list of the congress people who worked together on that bill, for example:

```
Biden Jr., Joseph R. [DE]/Kennedy, Edward M. [MA]/Kerry, John F. [MA]/Feinstein, Dianne
[CA]/Corzine, Jon [NJ]/Reed, John F. [RI]/Murray, Patty [WA]/Talent, Jim [MO]/Durbin,
Richard J. [IL]/Specter, Arlen [PA]/Kerry, John F. [MA]/Hutchison, Kay Bailey
[TX]/Schumer, Charles E. [NY]
```

You will not need the original file, instead you will use the *congresspeople_pairs.txt* file, which contains a list of pairs of congresspeople that appear together in at least one legislative bill. The names in each line are separated by a forward slash "/".

The file *congresspeople_pairs_short.txt* has a snapshot of *congresspeople_pairs.txt* so that you can work with a small file while developing and testing and not take too long every time you run your program. Work with the short file until you are done and ready to test it with the large one.

In the graph that you will build from this dataset, the vertices are congresspersons and edges connect congresspersons that have worked together in at least one legislative bill. For example:



Programming assignment: building and querying the graph

TASK 1. Add a constructor to *LabeledGraph* that will build the graph from an input file (with the format of the input file you chose). The name of the input file should be passed as an argument. Notice that *LabeledGraph* already has a constructor that takes no arguments. You will create a constructor that takes a *string* argument. You will need to modify both:

- the header file (*LabeledGraph.h*) – add the header of the constructor
- the implementation file (*LabeledGraph.cpp*) – write the implementation of the constructor

Depending on the dataset you chose, the function should process the input file as follows:

Movies and Actors – for each line in the file, create a vertex for the movie and one vertex for each actor/actress in the movie. Create edges to connect the movie to each actor in the movie. Remember that entries in each line are separated by "/".

Favorite Musical Artists – for each line in the file, create a vertex for each of the two musical artists and an edge to connect them. Remember that entries in each line are separated by ",".

Legislative Co-sponsorship in the U.S. House and Senate – for each line in the file, create a vertex for each congressperson in the pair as well as an edge to connect them. Remember that entries in each line are separated by "/".

The name of the input file should be passed as an argument to the constructor so that a *LabeledGraph* object can be created in a client program using **one** of the following instructions, depending on which dataset you chose:

```
LabeledGraph g("movies.txt");
LabeledGraph g("artists_pairs.txt");
LabeledGraph g("congresspeople_pairs.txt");
```

TASK 2. Write a client program to build a *LabeledGraph* from the input file (using one of the instructions above). You should initially use the short version of the file and test with the larger file once everything is working. After building the graph, your client program should answer simple queries where the user enters a vertex label and gets the list of vertices adjacent to that vertex (the neighbors). When you run the program:

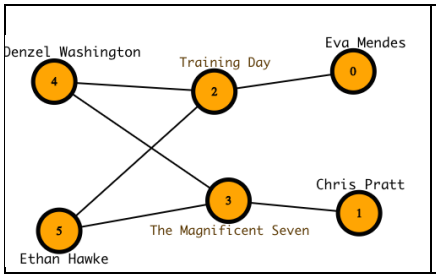
- if you chose the *Movies and Actors* dataset, when you type the name of an actor, you should get the list of the movies in the database in which that actor appeared, and when you type the name of a movie, you should get the list of actors that appear in that movie.
- if you chose the *Favorite Musical Artists* dataset, when you type the name of a musical artist, you should get the list of musical artists that appear together in at least one user's preferences list with the artist you typed.
- if you chose the *Legislative Co-sponsorship in the U.S. House and Senate* dataset, when you type the name of a congressperson, you should get the list of congresspersons that worked together in at least one legislative bill with the one you typed.

Notice that you can make use of the *index()* function to convert a vertex label to an index for use in graph processing and the *label()* function to convert an index into a label for use in the context of the application (such as displaying).

Quick Recap of Concepts (2)

This section is a quick recap of the concepts that you need to know for the next part of the project. If you don't understand something or can't answer the questions and exercises, you should review the textbook and/or materials provided for this topic.

To find the single-source shortest paths in a graph, we use *Breadth First Search (BFS)*. Given a graph and a source vertex `s` in that graph, BFS builds a table (parent-*map*) containing a parent-link representation of a *tree* rooted at `s`, which defines the shortest paths from `s` to every vertex that is reachable from `s`. From that table, we can obtain the path from `s` to any vertex that is connected to it. Example:



Graph G

Adjacency list

0	→	2
1	→	3
2	→	0, 4, 5
3	→	1, 4, 5
4	→	2, 3
5	→	2, 3

Label-Index map

Chris Pratt	1
Ethan Hawke	5
Eva Mendes	0
Denzel Washington	4
The Magnificent Seven	3
Training Day	2

Representation of G

parent-map

0	-1
1	3
2	0
3	5
4	2
5	2

Parent map returned by BFS with *Eva Mendes* as source vertex

BFS Algorithm

```
BFS (source_vertex)
  Q = empty queue
  visited = empty set
  parent_map = empty map

  parent_map[source_vertex] = NULL
  Q.enqueue(source_vertex)
  visited.add(source_vertex)

  while Q is not empty
    parent = Q.dequeue()
    for child in neighbors(parent)
      if child not in visited
        parent_map[child] = parent
        Q.enqueue(child)
        visited.add(child)

  return parent_map
```

declare variables

initialize variables with the source vertex

take the next vertex from the queue and visit all its unvisited neighbors

Programming assignment: Traversing the graph

In this part of the project, you will use the graph you built to query paths between vertices to see how they are connected. In fact, we want to find the shortest path (one with a minimal number of edges) between any pair of vertices. For example:

- in the *Movies and Actors* domain, you will find the shortest path between a pair of actors or between a pair of movies, or between a movie and an actor.
- in the *Favorite Musical Artists* domain, you will find the shortest path between two musical artists.
- in the *Legislative Co-sponsorship in the U.S. House and Senate* domain, you will find the shortest path between two congresspersons.

TASK 1. Extend *LabeledGraph* to add a member function that will perform **BFS** on the graph to find shortest paths from a source vertex. The function should receive the source vertex, v, as argument and return a *map* containing a parent-link representation of a tree rooted at v, which defines the shortest paths from v to every vertex that is reachable from it. The signature of the function should be as follows:

```
map<int, int> BFS(int v);
```

TASK 2. Add a member function to *LabeledGraph* that, given a *parent map*, it will return the path (from the source vertex for which the *parent map* was built) to a specific target vertex. The signature of the function should be:

```
vector<int> pathTo(map<int, int> & parent_map, int target);
```

TASK 3. Write a client program that will build the graph from the input file (using the constructor you added to *LabeledGraph*) and it then will allow querying paths from a source vertex to other vertices. The program should ask for the source vertex only once (at the beginning) and then it should repeatedly ask for a target vertex until the user decides to exit the program.

Sample runs for each domain are given in the next page.

Movies and Actors domain. Sample run:

Enter source vertex: Chris Pratt

All (shortest) paths from Chris Pratt to other vertices have been created.

Enter a target vertex and I will display the shortest path from Chris Pratt to that vertex (-1 to exit): Sam Worthington

Shortest Path from Chris Pratt to Sam Worthington:

Chris Pratt --> Guardians of the Galaxy --> Zoe Saldana --> Avatar --> Sam Worthington

Enter a target vertex and I will display the shortest path from Chris Pratt to that vertex (-1 to exit): Zoe Saldana

Shortest Path from Chris Pratt to Zoe Saldana:

Chris Pratt --> Guardians of the Galaxy --> Zoe Saldana

Enter a target vertex and I will display the shortest path from Chris Pratt to that vertex (-1 to exit): -1

Goodbye.

Notice that the input files provided (movies.txt and movies_short.txt) have the names in a different format, so you have to either enter the names as formatted on the file or convert them as you read the input file and build the graph.

Favorite Musical Artists domain. Sample run:

Enter source vertex: Kanye West

All (shortest) paths from Kanye West to other vertices have been created.

Enter a target vertex and I will display the shortest path from Kanye West to that vertex (-1 to exit): Queen

Shortest Path from Kanye West to Queen:

Kanye West --> Coldplay --> Queen

Enter a target vertex and I will display the shortest path from Kanye West to that vertex (-1 to exit): Metallica

Shortest Path from Kanye West to Metallica:

Kanye West --> Arctic Monkeys --> The Killers --> Guns N' Roses --> Metallica

Enter a target vertex and I will display the shortest path from Kanye West to that vertex (-1 to exit): -1

Goodbye.

Legislative Co-sponsorship in the U.S. House and Senate domain. Sample run:

Enter source vertex: Joseph Biden

All (shortest) paths from Joseph Biden to other vertices have been created.

Enter a target vertex and I will display the shortest path from Joseph Biden to that vertex (-1 to exit): Hillary Clinton

Shortest Path from Joseph Biden to Hillary Clinton:

Joseph Biden --> John McCain --> Hillary Clinton

Enter a target vertex and I will display the shortest path from Joseph Biden to that vertex (-1 to exit): Edward Kennedy

Shortest Path from Joseph Biden to Edward Kennedy:

Joseph Biden --> Charles Schumer --> Edward Kennedy

Enter a target vertex and I will display the shortest path from Joseph Biden to that vertex (-1 to exit): -1

Goodbye.

Notice that the input files provided (congresspeople_pairs.txt and congresspeople_pairs_short.txt) have the names in a different format, so you have to either enter the names as formatted on the file or convert them as you read the input file and build the graph.