Lecture Review

Graph data type

In lecture, we discuss graph data type and its implementation. A graph is a collection of *vertices* and *edges*. See `https://introcs.cs.princeton.edu/java/45graph/` for more details on graphs. The `Graph` class is defined in `graph.hpp` file that represents an undirected graph data type. Here's a brief description of its API:

- `Graph()`: Default constructor that initializes an empty graph with no vertices or edges.
- `Graph(string filename, char delimiter)`: Constructor that initializes a graph from the specified file, using the specified delimiter.
- `int V()`: Returns the number of vertices in this graph.
- `int E()`: Returns the number of edges in this graph.
- `int degree(string v)`: Returns the degree of vertex `v` in this graph.
- `bool hasVertex(string v)`: Returns true if `v` is a vertex in this graph.
- `void addVertex(string v)`: Adds vertex `v` to this graph (if it is not already a vertex).
- `bool hasEdge(string v, string w)`: Returns true if `v-w` is an edge in this graph.
- `void addEdge(string v, string w)`: Adds the edge `v-w` to this graph (if it is not already an edge).
- `to_string()`: Returns a string representation of this graph.
- `const unordered_set<string>& adjacentTo(string v)`: Returns a reference to the set of vertices adjacent to `v`.

The `vertex_iterator` class is a nested class within the `Graph` class. It is used to iterate over the vertices of the graph. The following two methods are provided by the `Graph` class to create iterators over the vertices:

- `vertex_iterator vbegin()`: Returns an iterator pointing to the beginning of the vertices.
- `vertex_iterator vend()`: Returns an iterator pointing to the end of the vertices.

Shortest paths in graphs

The `PathFinder` class in your `path_finder.hpp` file represents a path finder in a graph using *Breadth-First Search (BFS)*. Here's a brief description of its API:

- `PathFinder(Graph G, string s)`: Constructor that runs BFS in graph `G` from the given source vertex `s`.
- `bool hasPathTo(string v)`: Returns true if vertex `v` is reachable from the source `s`.

- `int distanceTo(string v)`: Returns the length of the shortest path from `v` to `s`. If `v` is not reachable from `s`, it returns the maximum possible integer value.
- `std::vector<string> pathTo(string v)`: Returns the shortest path from `v` to `s` as a vector of strings. Each string in the vector represents a vertex in the path. The path is constructed by backtracking from `v` to `s` using the `prev` map, which stores the previous vertex on the shortest path from `s` to each vertex.

Lab Exercises

Exercise 1 ....

Word ladder. Write a program that takes filename (containing list of 5-letter words) and two 5-letter strings as command-line arguments, and prints out a shortest *word ladder* (`http://en.wikipedia.org/wiki/Word_ladder`) using the words from the provided list connecting the two strings (if it exists). Two words are adjacent in a word ladder chain if they differ in exactly one letter. As an example, the following word ladder connects green and brown:

```
green greet great groat groan grown brown
```

This game, originally known as *doublet*, was invented by *Lewis Carroll.* You can also try out your program on this list of 6 letter words.

The world lists are attached as `words5.txt` and `words6.txt`.

Hint: Read the words file and create a appropriate `Graph` object. Then use the PathFinder class to find the shortest path between the two words.

Exercise 2 ....

The *Bacon Number* is a way of measuring the closeness of an actor to *Kevin Bacon* (`https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon`). The Bacon number of an actor is the length of the shortest path between the actor and Kevin Bacon in the *movie-performer* graph. Two actors are connected in the graph if they have appeared in a movie together. Write a program that takes a filename (containing list of movies and actors) and two actor names as command-line arguments, and prints out the Bacon number of the first actor with respect to the second actor.

Various movie-performer data files are available on the web. See `https://introcs.cs.princeton.edu/java/45graph/`.

FILE	DESCRIPTION	MOVIES	ACTORS	EDGES
`cast.06.txt`	movies release in 2006	8780	84236	103815
`cast.00-06.txt`	movies release since 2000	52195	348497	606531
`cast.G.txt`	movies rated G by MPAA	1288	21177	28485
`cast.PG.txt`	movies rated PG by MPAA	3687	74724	116715
`cast.PG13.txt`	movies rated PG-13 by MPAA	2538	70325	103265
`cast.mpaa.txt`	movies rated by MPAA	21861	280624	627061
`cast.action.txt`	action movies	14938	139861	266485
`cast.rated.txt`	popular movies	4527	122406	217603
`cast.all.txt`	over 250,000 movies	285462	933864	3317266

Note: These input files are as of November 2, 2006. The data is taken from the *Internet Movie Database*.

Exercise 3 ....

Histogram. Write a program that prints a histogram of Kevin Bacon numbers, indicating how many performers from `movies.txt` (attached) have a Bacon number of $0, 1, 2, 3, \ldots$. Include a category for those who have an infinite number (not connected at all to Kevin Bacon).

```
  #     Freq
-----------
  0        1
  1     2083
  2   187072
  3   515582
  4   113741
  5     8269
  6      772
  7       93
  8        7
Inf    28942
```