CMPSC 122.1
April 12, 2018

Homework 5

Due: Refer to last page for due dates

Overview

This assignment involves graphs, and using several different data structures together in order to implement a class graph search algorithm.

Dijkastra's Shortest Path Algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities

Source: [https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm]

Heap

A heap is an ADT that is very similar to the priority-queue from the last assignment. A heap is called a min-heap or a max-heap based on whether smaller or larger values determine the priority of a value. For this assignment, we will be using a min-heap to determine which vertices we want to visit next. You will be using the values in the distance array (see pseudocode) as the priority of a vertex. The min-heap will automatically keep its heap structure when new values are inserted or the min value is extracted from the top.

Resources

A visual goes a long way in explaining complex algorithms. This video goes through the entire process of Dijkstra's algorithm on the graph that is provided to you in the graph text files.

Source: [https://www.youtube.com/watch?v=5GT5hYzjNoo]

Take note of the process and how it related to the pseudocode below.

Pseudocode

Graph represents the adjacency matrix of a graph. You can determine all of the edges and vertex in the graph just with the adjacency matrix. Source and target are both vertices from the graph, and simply indices from the adjacency matrix.

Distance is an array that is used to keep track of the shortest path between source and all of the other vertices in the graph. The previous array is used to keep track of the vertices with the shortest path that leads to each vertex.

This pseudocode is modified version of the code from here:

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Using_a_priority_queue

```
function Dijkstra(Graph, source, target)
	distance[source] = 0;

	create min-heap Q

	for each vertex v in Graph:
		if v is not source
			distance[v] = infinity
			previous[v] = -1

		Q.add_with_priority(v, distance[v])
	end for

	while Q is not empty:
		u = Q.extract_min()

		if u is target
			break out of while
		for each neighbor v of u:		// only consider v that are still in Q
			alt = distance[u] + length(u, v)
			if alt < distance[v]
				distance[v] = alt
				previous[v] = u
				Q.decrease_priority(v, alt)
		end for
	end while

	// Determine the shortest path from source to target using previous array

	create stack S
	u = target

	while previous[u] is defined:
		push u onto S
		u = previous[u]
	push u onto S

	// Output path ordering with distance between each node
	// Output total distance of smallest path from source to target

end function
```

CMPSC 122.1
April 12, 2018

Homework 5

Due: Refer to last page for due dates

Program (100 points) **Dijkastra's Shortest Path Algorithm between two vertices on a graph**

Write a program (*dij.cpp)* that does the following:

- Leverages the provided includes to accomplish the following:
  - *min-heap.h*: determine the most desirable vertice to process next
  - *graph.h*: read in a graph from a text file and store its adjacency matrix for read-only access
  - *set.h*: keep track of vertices that have already been visited
  - *stack.h*: output the vertices in the shortest path

- You should accept a file name of the graph you want to test (assume the file is in the same directory as the cpp program)
- Load the graph into memory with the Graph class
- Accept two index values representing the source vertex and target vertex
- Run Dijkstra's Shortest Path Algorithm on the loaded graph using the source and target vertex
- Output the entirety of the determined shortest path, with the distance of each edge in the path and the total path between the two vertices

Input/Output Format:

```
Please enter location of graph file to load: graph_1_win.txt

Please enter the index of the starting vertex: 0
Please enter the index of the target vertex: 7

The shortest path from vertex 0 to vertex 7 is:

0 to 2: 2
2 to 3: 2
3 to 4: 1
4 to 6: 1
6 to 5: 2
5 to 7: 3

Total path length is 11
```

Template Program Files

There are nine (9) files in total that will be provided for download on Canvas.

- A starter file *dij.cpp* that shows how to load a graph from a text file into a graph class
- Six (6) header files with the complete implementations of all the ADTs you need to complete the assignment
- Two (2) graph text files. The file ending in nix contains Unix style line endings and the file ending with win contains Windows style line endings. The Graph class should be able to read in either file regardless of the operating system you are; however, it is recommended you use the text file that is appropriate for the system you are currently on. If you are using a Mac, use the text file with Unix style line endings.

Homework 5

Compiling the Program with g++

Use the following command to compile your classes. This is the command I personally use on the Sunlab machines to compile and grade your programs:

```
g++ -Wall -o <output_name> <program_name.cpp>
```

Example:

```
g++ -Wall -o dij dij.cpp
```

Remember: Your programs must successfully compile without any errors, or a zero will be given for that program.

Following Instructions

You are expected to follow the directives laid out in this assignment and the course syllabus. Points will be deducted for incorrectly named files, missing/incorrectly filed out banner comments, not supplying hardcopy submissions in a pocket folder with the appropriate information, and failing to implement features/functionality specified in the assignment. It is expected that you will utilize the provided template files for this assignment, and that you do not modify the names of class members/functions that are provided in the template.

If you have questions about any portions of the assignment or what is expected, contact me for clarification.

Submission

- Electronic Submission (Due: 5:00 PM, Monday, April 30, 2018)
  - All nine (9) files initially provided (your completed *dij.cpp*, all the header files, and the two graph files)
  - Submitted using the **mail122** command on the Sunlab machines
- Hardcopy Submission (Due: Beginning of class, Monday, April 30, 2018)
  - Printed hardcopy of your completed *dij.cpp* source code
  - The page of the program should be stapled together
  - Submitted in a pocket folder with your name, the course, and the section number on the front