Project 3: Queue and Its Applications

Due: 03/09/2017

Educational Objectives: Understand and experience the implementation of Queue ADT and its applications. Understand and implement the breadth first search algorithm for route search in the flight reservation application. Analysis of algorithm complexity.

Statement of Work: Implement a generic Queue class template, and use it to implement the bread search algorithm to find the shortest route between two nodes in a graph. Analyze the time complexity of a member function of Queue.

Requirements:

This project contains two parts. In the first part of the project you need to implement a Queue class template as an adaptor class. You have the freedom to use any C++/STL containers (and algorithms) in the project, except the STL queue container. Note also that, you must not use the Vector class template you have implemented in the previous project in the current project. If you need to use a vector, use the one provided in C++/STL. In the second part of this project, you will implement the breadth first search algorithm to find the shortest route between two nodes in a graph.

Task 1: Implement Queue as a generic adaptor class template

- You must implement Queue as an adaptor class template. You can use any C++ STL container as the underlying adaptee class (of course, you cannot use C++ STL queue container).
- Your implementation of Queue must be in the namespace of cop4530.
- You must provide the template definition and implementation in two different files Queue.h (containing Queue class template definition) and Queue.hpp (containing the implementation of member functions). You must include Queue.hpp inside Queue.h as we have done in the previous project.
- You must implement all the interface specified below for the Queue class template.

  Queue interfaces (T is the template parameter, i.e., data type)

- **Queue()**: zero parameter constructor. Create an empty queue.
- **~Queue()**: destructor. De-allocate memory if necessary.
- **Queue(const Queue &rhs)**: copy constructor. Create the new queue with the elements of an existing queue rhs.
- **Queue(Queue &&rhs)**: move constructor. Create the new queue with the elements of an existing queue rhs.
- **Queue& operator=(const Queue &rhs)**: copy assignment operator.
- **Queue& operator=(Queue &&rhs)**: move assignment operator.
- **T& back()**: return a reference to the last element in the queue.
- **const T& back() const**: constant version of the above member function.
- **bool empty() const**: return true if there is no element in the queue; return false otherwise.
- **T& front()**: return a reference to the first element in the queue.
- **const T& front() const**: constant version of the above member function.
- **void pop()**: remove the first element in the queue and discard it.
- **void push(const T& val)**: add a new element val into the end of the current queue.
- **void push(T&& val)**: add a new element val into the end of the current queue. val should be moved instead of copied (if possible)

- **int size()**: return the number of elements in the current queue.

  Analyzing the worst-case time complexity of the member function pop() of Queue. Give the complexity in the form of Big-O. Your analysis can be informal; however, it must be clearly understandable by others. Name the file containing the complexity analysis as "analysis.txt".

Task 2: Implement the shortest route search algorithm for flight reservation using the breadth first search.

- In this part of the project, you will implement a (simplified) system to help airline customers buy tickets. The system loads the direct flight information from a file. It then asks users for the source/destination city of the travel, and compute the shortest route from the source city to the destination city. In this project, the shortest route is defined in terms of number of connections in a route. A shortest route is a route with the least number of connections. If there are multiple of such shortest routes, you can output any of them.
- The direct flight information is given in file`proj3.input`. The first line of this file is the number of cities (num) that the airline services. After this line, the list of the names of the cities is given. Each name occupies one line. Finally, the direct flight information is given in a `num x num` matrix. In the matrix, a positive number is the price of the direct flight between the two cities. The number -1 indicates that there is no direct flight between the two cities. Following is an example of proj3.input. Your first task is to load the information in proj3.input and store it in an internal data structure to be used later.

-
  ```
  8
  Atlanta
  Denver
  New York
  Orlando
  Pittsburgh
  San Diego
  Seattle
  ```

```
Tallahassee
  0 100   -1   50   -1   -1    -1   100
100   0   -1   -1   50   10    -1    -1
 -1  -1    0 1000  200   -1  1000    -1
 50  -1 1000    0   -1   -1    -1    -1
 -1  50  200   -1    0   -1   100    -1
 -1  10   -1   -1   -1    0    50  2000
 -1  -1 1000   -1  100   50     0    -1
100  -1   -1   -1   -1 2000    -1     0
```

The matrix should be interpreted as follows.

	ATL	Den.	NY	Orlando	Pitt.	SD	Seattle	TLH
Atlanta	0	100	-1	50	-1	-1	-1	100
Denver	100	0	-1	-1	50	10	-1	-1
New York	-1	-1	0	1000	200	-1	1000	-1
Orlando	50	-1	1000	0	-1	-1	-1	-1
Pitt.	-1	50	200	-1	0	-1	100	-1
San Diego	-1	10	-1	-1	-1	0	50	2000
Seattle	-1	-1	1000	-1	100	50	0	-1
TLH	100	-1	-1	-1	-1	2000	-1	0

- With the direct flight information, you will implement the shortest route airline ticket search algorithm, which will find a route with the least number of connections (minimum hop path).
- The I/O of your program must follow the sample executable.

Downloads

Click here to download the tar file, which contains the following files: proj3.x and proj3.input. The sample executable program proj3.x was compiled on a linprog machine.

Note: The first person to find a programming error in our program will get a bonus point!

Hints:

- Write some test files to make sure your implementation of Queue is correct.
- You can use the breadth first search algorithm to determine the shortest route from a source city to a destination city. In order to keep the path information (what cities a route traverses), you have a number of choice. For example, you keep a "parent" pointer to the parent city (how the current city is reached), or you can simply keep the partial route in the queue.
- The complexity of the pop() function depends on your underlying adaptee class. Different choices of the underlying adaptee class will present different complexity of pop().

Deliverables: Turn in all your source program files, makefile, and analysis.txt in a tar file via the blackboard system.