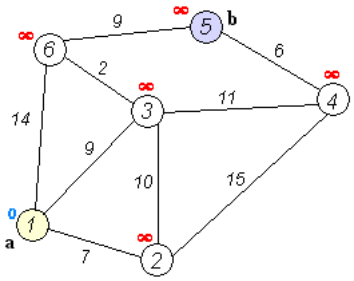
Assignment 5

For this assignment you'll implement Dijkstra's algorithm for finding the shortest path between nodes in a graph.



To achieve this goal you'll need two important data structures a `Graph` and a `Priority Queue`, which you will do here in part 1. Actually implementing the Dijkstra algorithm will happen in part 2.

For the `Graph` object, you will implement your own. I have provided to you `TemplateGraph.h` which includes the public facing aspects of the `Graph` class. You should modify it (adding private fields only) and change the name to `Graph.h` It is up to you how to implement the underlying, private details. You may use an adjacency list, adjacency matrix, or some other scheme. I have also provided to you a lightly redacted `GraphTests.cpp` program, which includes some of the tests I will run on your `Graph`. You should fill in more tests and add test functions as you see necessary.

For the `Priority Queue` object, it would be nice to be able to use the standard library `std::priority_queue` Unfortunately, the interface is somewhat limited (http://www.cplusplus.com/reference/queue/priority_queue/). To implement Dijkstra's algorithm we need an `Update()` method to change the priority of items in the queue. Additionally, it may not be necessary, but it would might be nice to have a `Contains()` method. **Both** of these are missing. I guess we can write a letter to Bjarne Stroustrup and ask him what the hell his problem is. For now we're just going to have to fix the problem for our-selves by extending (writing a child-class) of `std::priority_queue.`

1. Create a BetterPriorityQueue class that extends `std::priority_queue` thusly:

```
class BetterPriorityQueue: public priority_queue<DNode, vector<DNode>,
greater<DNode>>::priority_queue {

// ... your code goes here ...

};
```

Oh yeah, that's a big one! Specifying the `priority_queue` this way makes three distinctions. First, the items inside this queue will be `DNode` which is a struct. The code for for `DNode` is provided to you in `Dnode_struct.txt`, but you will have to decide where to incorporate it into your project. The second parameter specifies that a `vector` of `DNode` will be used as the underlying mechanism to

implement the priority queue (fine). The third parameter specifies that this queue will place the minimum item first instead of the largest (also fine).

Your `BetterPriorityQueue` should add **four** new public methods.

- `Contains()` takes a `DNode` and returns `true` if that `DNode` is in the queue.
- `Update()` takes a `DNode` and uses the values in that `DNode` to change the priority of a matching `DNode`. For example,

  suppose `BetterPriorityQueue bpq = [(a: 1), (b: 2), (f: 9), (d: 13), (e: 14)]`
  `DNode n = (f: 0);`
  calling `bpq.Update(n)` results in `[(f:0), (a: 1), (b: 2), (d: 13), (e: 14)]`

  If the `DNode` doesn't match any existing in the queue, `Update()` should have no effect on the queue. The example above is not literal C++ code, it is written to be human understandable. Please note, the order of the nodes is determined by an underlying binary heap.
- `ToString()` returns a string representation of the queue (like those shown above)
- `DnodeToString()` static, takes a `DNode struct` and returns a string representation (like those shown above, in the queue).

Extending `std::priority_queue` provides you access to the protected member `c`, which is the `vector<DNode>` containing the items of the queue. You can use `c` to implement the above methods.
https://en.cppreference.com/w/cpp/container/priority_queue

An incomplete test file `BPQTests.cpp` has been provided to you. You should fill in more tests and add test functions as you see necessary.

You're done when you have a well tested `Graph` and `BetterPriorityQueue`. Or not, I suppose. I can't really stop you from simply giving up. I'll leave it up to you to decide how much testing is enough testing for your own piece-of-mind. Just be sure to remember that your success and happiness for the rest of your professional life depends on your exhaustive completion of this specific homework assignment.

For part one you should have a Makefile that creates two executables:

- `Graph.cpp GraphTests.cpp → graph-tests`
- `BetterPriorityQueue.cpp BPQTests.cpp → bpq-tests`

Note: This assignment is complicated on my end. I might have (very probably) made a technical mistake that makes the assignment very difficult or perhaps impossible! Sorry in advance for that. If you notice something seems missing, undefined, or otherwise confusing please don't hesitate to reach out to me! For those of you wondering how I could make a mistake just know that I am in fact perfect and any mistake is likely the result of the imperfect society that raised me.

Submission: Your program will be submitted using canvas and git + github.

1) You should create a private `github.com` repository (repo) with your code in it. You may need to create a github user account and a github repository. To achieve this. The name you choose for the repo and your username are arbitrary but I suggest "HW5" as the repo name.

2) Your repository on github.com should be **private** and you should add my account as a collaborator by going to "settings" → "manage access" → "invite collaborator" and entering my username: `fmresearchnovak`

3) On canvas you should upload a link to this github repository
`https://github.com/fmresearchnovak/HW5.git`

I will grade the final commit made before the due-date. This means that you can actually submit on canvas at any time before the deadline, and still make changes (new commits) to the code.