Stanford | ENGINEERING
Computer Science

CS 106B: Programming Abstractions in C++
Summer 2014

Homework 7 (Trailblazer) FAQ

Q: The spec says I am not supposed to modify the `.h` files. But I want to use a helper function. Don't I need to modify the `.h` file to add a function prototype declaration for my helpers? Can I still use helper functions even if I don't modify the `.h` file?
A: Do not modify the provided .h file. Just declare your function prototypes in your .cpp file (near the top, above any code that tries to call those functions) and it'll work fine. You can declare a function prototype anywhere: in a .cpp file, in a .h file, wherever you want. The idea of putting them in a .h file is just a convention. When you `#include` a file, the compiler literally just copy/pastes the contents of that file into the current file. We have already done this on hw1, hw2, and others.

Q: Why am I getting a compiler error of, "no type named 'iterator' in 'class BasicGraph'"?
A: You are probably trying to do a "for-each" loop over the graph. You can't do that. You *can*, however, loop over the vertices or edges of the graph by calling `getVertexSet` or `getEdgeSet` on it.

Q: Why am I getting a runtime error of, "Non-adjacent locations passed into cost function"?
A: Your path-finding function is returning an invalid path that contains two or more vertices that are not actually neighbors of each other.

Q: My depth-first search crashes on very large graphs. Why?
A: If you implemented it recursively, it crashes if the call stack has too many recursive calls piled up. This is okay and you don't need to worry about it, as long as it works for most graphs other than really large ones.

Q: I implemented the algorithms and I get the same path length/cost, but my "Locations visited" is slightly different on some inputs. And my solution seems to visit the possible paths in a different order than the screenshot. Is this okay? Is my algorithm wrong? Do I need to change/fix it?
A: If you have implemented each path-searching algorithm correctly, for DFS you should get any valid path from the start to the end; for BFS you should get the same path lengths as shown in the expected outputs posted on the class web site. For Dijkstra's and A* you should get the same path costs as shown in the

expected outputs. But you do not need to exactly match our path itself, nor its "locations visited", so long as your path is a correct one.

One common reason for output differences comes from how you loop over neighbors. Our solution loops over them by calling `getNeighbors` on the graph and passing in the vertex of interest. Some students loop over them by accessing the `edges` field of the vertex of interest and looking at all the vertices that are endpoints of those edges. Neither way is "right" or "wrong", but they sometimes produce different orders.

Q: How should we handle the case when a path is not found?
A: The spec answers this question, under the "Required Functions:" section. "If no path is found, ..."

Q: How should we handle the case when the start and end vertices are the same?
A: The spec answers this question, under the "Required Functions:" section.

Q: How do I test unusual cases, like impossible paths?
A: Some of the later higher-numbered provided input files test such cases, like the last few maze/terrain files.

Q: The A* algorithm produces results where the cost is too high / incorrect. What might be causing this?
A: Check the lecture slides on A* carefully. There is a difference between a vertex's cost and its priority. The former doesn't incorporate heuristics and the latter does. Make sure you are properly maintaining these two values separately.

Q: My Dijkstra's and A* work for mazes but the costs are often a little bit too high for terrains. Why could this be?
A: Are you storing costs as an `int`, by chance? They *must* be a `double` for the algorithm to work properly on terrains.

Q: Is there an efficient way to determine whether or not a value already exists in the priority queue (aka a contains() function analogy)?
A: You are supposed to be storing various information associated with each vertex. When you look at a vertex, you can look at that information and based on its value you can take different actions. At the start of Dijkstra's algorithm, you are supposed to set every vertex's associated information/state to certain values. If you haven't ever added a given vertex to the PQ, its state will still be set to those same values. But if you do enqueue it, presumably you change the state to different values. So you should be able to look at the state and figure out what if anything has been done to that vertex in the past.

Q: My Kruskal's algorithm is very slow. Why? How can I speed it up?
A: This part of the assignment really stretches your understanding of Big-Oh. Note that seemingly simple operations can be very expensive, such as:

- looping over all vertices (O(V))
- looping over all edges (O(E))
- looping over all vertices once for every edge in the graph (O(V * E))

- repeatedly calling a method that makes a deep copy of a collection, like `getEdgeSet` (O(E log E)), or `getVertexSet` (O(V log V))
- setting a Vector variable equal to another Vector variable (O(N))
- setting a Set variable equal to another Set variable (O(N log N))
- ...

Operations like the above can really add up. Carefully audit your code and reduce the number of unnecessary bulk operations you are performing.

Q: The spec mentions the possibility of doing a bi-directional search. How would I go about adding this to the program in a way that can be tested?
A: If you want to enable bidirectional search, in the `trailblazer.h` file, add the following line:

```
#define BIDIRECTIONAL_SEARCH_ALGORITHM_ENABLED true
```

Then in your `trailblazer.cpp` file, write a function with exactly the following heading that contains your algorithm:

```
Vector<Vertex*> bidirectionalSearch(BasicGraph& graph, Vertex* start, Vertex* end) {
    ...
}
```

You'll need to do a full rebuild of your project.