Programming Assignment 1

EECS 311 Data Structures
Fall 2014
Instructor: Peter Scheuermann

Assigned: October 10, 2014 Due: October 24 (midnight)

Background

A common approach to the wire routing problem for electrical circuits is to impose a grid over the wire routing region. The grid divides the routing region into an m x m array of squares. A wire runs from midpoint of one square S to the midpoint of another T. In doing so, wire may take right angle turns. Grid squares which already have wire through them are blocked. To minimize the signal delay we wish to route the wire using shortest path between S and T. The shortest path between grid positions *S* and *T* is found in two passes

1. Distance-labeling pass (i.e., labeling grids)
2. Path-identification pass (i.e., finding the shortest path)

In rest of this document we would use terms coordinate and square interchangeably.

Input & Output

The input to your program is one file. This file contains tab separated values which can be either 1, 0, S or T. These values represent following:

'1' represents that this coordinate is open to be used in a path.
'0' represents that this coordinate is blocked and cannot be used in a path.
'S' represents that this coordinate is starting point.
'T' represents that this coordinate is the target.
Your program should be able to take 5 arguments. For example,

<program_name> <filename> <start_x> <start_y> <target_x> <target_y>

<filename> Sample Grid File
<start_x> Start Column number
<start_y> Start Row number
<target_x> Target Column number
<target_y> Target Row number

Important: Your program should work for any combination of S, T, 0's and 1's in the sample file. Your program will be tested with a couple of different input files. You will be provided with some more sample files in a few days.

Example Input: One input file named ‘sample_grid’ is provided with the assignment. An example 4x4 grid has been copied below.

S 1 1 1
1 1 1 1
1 1 1 1
1 1 1 T

After Finding the Path:

S	**1**	**1**	**1**		**S**	1	1	1
1	1	1	**1**		**1**	1	1	1
1	1	1	**1**	**OR**	**1**	1	1	1
1	1	1	**T**		**1**	**1**	**1**	**T**

In above example starting point is coordinate (0,0). So, <start_x> =0 and <start_y> = 0.

The required output of your program should be the following:
--Report whether you found a path from S to T
-- If finding a path was successful then print the path using the following notation:

Integer ‘Path length’: 6
Array Path: (1,0) (2,0) (3,0) (3,1) (3,2) (3,3)

OR

Array Path: (0,1) (0,2) (0,3) (1,3) (2,3) (3,3)

Any path is acceptable as long as your path doesn’t have redundant coordinates. Note, in above example, none of the value is ‘0’. If during labeling you find a ‘0’, you cannot use that in your path. For example, in following case path you have to avoid ‘0’. Now, path has been changed.

S	**1**	**1**	1		**S**	**1**	1	1
0	1	**1**	0		0	**1**	1	0
1	1	**1**	**1**	**OR**	1	**1**	1	1
1	1	1	**T**		1	**1**	**1**	**T**

Also, you cannot jump from one coordinate to another diagonally.

Suggested Plan-of-attack

1. Determine your starting point and target. You can do “cat sample_grid” to find out on which coordinate ‘S’ and ‘T’ placed. Give that information on input.

2. Read the input file into a 2-dimensional integer grid array. You may want to store ‘0’ instead of S or T in the array.

3. Starting from position *S*, label its reachable neighbors 1. Next, the reachable neighbors of coordinates labeled 1 are labeled 2. This labeling process continues until we either reach coordinates of T or have no more reachable coordinates.

4. At this point, all the members in the array which has been labeled contain the distance from the start location. Note, that you cannot update a coordinate with value 0, as that is a blocked coordinate and cannot be labeled. You will update the member which already has ‘1’. Also, you need to distinguish between the label ‘1’ and the ‘1’ read from sample file.
   Suggestion: You can avoid this by labeling the start point to be ‘10’ and label its immediate neighbors to be ‘11’ and so on. At the end you have to subtract 10 from the path length you have found.

5. If you have reached the target coordinate, starting from position *T*, move to any one of its neighbors labeled one less than *T*’s label. Such a neighbor must exist as each grid’s label is one more than that of at least one of its neighbors. From here, we move to one of its neighbors whose label is one less, and so on until we reach *S*.

6. NOTE: you may not utilize existing C++ STL templates. All data structures must be implemented on your own.

7. You need to implement a queue class similar to given bellow. Using this queue class, you will push all the neighbors of current coordinate into the queue. If T is not one of them, then pop the front and repeat it for new front. Until you reach T. A class implementing this interface has to implement these methods.

```
class queue
{
    public:
       queue(); {}
     ~queue();
        bool IsEmpty() const;
        int size() const;
        front() const;
        back() const;
        void pop() ;
        void push(const T& theElement);
};
```

Submission & Grading

Submission is done through the Northwestern Blackboard system. After you login to the system, you will find the link that allows you to upload your assignment.
We request all programs to be compiled on the T-lab machines using g++.

The files to be submitted include the following:

1. all source code files, .h, .cpp/ .cc files
2. the make file you used to compile your program
3. a brief (less than a page) description of any important design decisions you made while implementing the project.

To submit your project:

Before submission, you must use the following command to create a zip ball that includes all the files required above. You should not submit the executable! You should submit this zip file only. To do that, cd into your working directory on the T-lab machines and then type:

```
tar czf your_name.tgz
```

where `your_name` is your name.

A make rule that will do the trick:

```
submission:
    ${TAR} ${NAME}-${PROJ}.tar $DELIVERY$ $COMPRESS$
    ${TEAM}-${PROJ}.tar
```

You would then only have to type

```
make NAME=<your_name> submission
```

Late submission policy: programs will be accepted for at most two days later with a penalty of 10% off for each day. After submission, if you discover a mistake and want to submit a revised copy, you can simply redo the whole procedure and overwrite your old file.

We will recompile and test your program on the T-lab machines. We will use an input file other than the one supplied to you to test whether your program runs correctly. We will also be checking the quality of your solution, efficiency and programming style. Please refer to the first handout for more detailed grading policy.

Good luck!