CS302 -- Project 4 -- Path Finding

Overview

The fourth project is to implement a path finding component of a game, either online (web application) or on your computer.

You are allowed to work with a single partner (group of 2) but this will not required.

Big picture

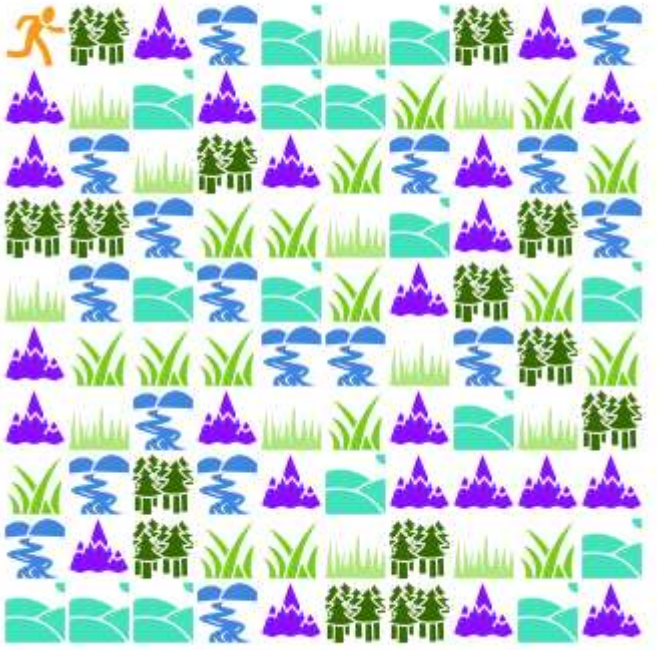
Consider the following text and graphical representation of a 2D landscape:

Text

```
hfmrhghfmr
mghmhhGgGm
mrgfmGrmrG
ffrGGghmfr
grhrhGmfGh
mGGGrrgrfG
mgrmgGmhgf
Grfrmhmmmm
rmfGGgfgGh
hhhrmffmhm
```

Tiles

Map



Each tile has a symbol (e.g., a forest has "f", river has "r") and can be assigned a specific cost, i.e., forest has a cost of 3 and river has a cost of 5.

Your mission, which you must choose to accept, is to help the runner (orange guy above) go from cell (x,y) to cell (i, j) with the lowest overall cost.

Dijkstras

For this project, you will need to implement Dijkstra's Algorithm (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm) in the `src/dijkstras.cpp` file.

The `dijkstras` path finding component will be given the specification of the tile information, the map layout, the runner's location, and the target location via standard input (https://en.wikipedia.org/wiki/Standard_streams#Standard_input_.28stdin.29) in the following format:

```
TILES_N
TILE_NAME_0 TILE_COST_0
...
TILE_NAME_N-1   TILE_COST_N-1

MAP_ROWS MAP_COLUMNS
TILE_0_0    ...
...

RUNNER_START_ROW RUNNER_START_COL
RUNNER_END_ROW   RUNNER_END_COL
```

The first `TILES_N` indicates the number of different tiles. This is followed by the specified number tiles, where each line has the tile name and then its corresponding cost.

Next, the number of rows and columns in the map is given in `MAP_ROWS` and `MAP_COLUMNS`. This is followed by all the tile symbols where each row is separated by a newline and each column is separated by a space.

Finally, you are given the runner's starting and ending map coordinates.

Here is an example of this input:

```
6
f 3
g 1
G 2
h 4
m 7
r 5
10 10
G r g g m m m r f m
G g r G G G G m f h
m r f m m f G r h h
G m G f h r g m g g
g g g m h m h f m f
h r g f f f g h r h
m G f r m m G r g f
m r h h h h G m m r
r r g f G r r m f r
G g r g g r h m m r
0 0
7 6
```

Once this information is read in and parsed, the `dijkstras` application should perform Dijkstra's Algorithm (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm) to determine the shortest path from the runner's starting location to the target ending position. The result of this computation should be printed to standard output (https://en.wikipedia.org/wiki/Standard_streams#Standard_output_.28stdout.29) in the following format:

```
Cost
ROW_0  COL_0
...
```

That is, the first item to be printed is the total path cost, followed by the tiles to travel starting from the runner's starting position to the target destination.

Here is the output your program should emit given the example input above:

```
27
0 0
0 1
0 2
0 3
1 3
1 4
1 5
1 6
2 6
3 6
4 6
5 6
6 6
7 6
```

To test your program, you can use `make test`, which will run `dijkstras` against 4 different maps.

Note, as we will discuss in class, and similar to SuperBall -- there may be multiple valid shortest paths if different paths have the same total weight. Because of this, if the tests fail, let us know and we will verify if it is a false negative or not.

Calculating Path Cost

To calculate the total cost of the path and to determine the tiles to travel, you will need to reconstruct the path backwards as discussed in class. In computing the total cost, remember to only include the cost of **leaving** a tile. This means, we should include the cost of leaving the starting tile but not out of the destination (because we never leave it).

Additional questions

When you are finished implementing your `dijkstras` implementation answer the following questions in your `README.md` :

1. Write a program, `generate_map`, that given `N` generates a `NxN` map of random tiles. Use this program to generate random maps with the following values of `N`:

   ```
   10, 20, 50, 100, 200, 500, 1000
   ```

   Benchmark your `dijkstras` path finding component on each of these map sizes and record the elapsed time and memory usage in a Markdown (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) table as demonstrated below:

```
| N              | Elapsed Time   | Memory Usage   |
|----------------|----------------|----------------|
| 10             | ...            | ...            |
| 20             | ...            | ...            |
| 50             | ...            | ...            |
| 100            | ...            | ...            |
| 200            | ...            | ...            |
| 500            | ...            | ...            |
| 1000           | ...            | ...            |
|----------------|----------------|----------------|
```

Note: You should run multiple trials (e.g `5`) and average the results.

Note: You should choose the top-left and bottom-right corners as the starting and ending points respectively.

In addition to the table above, each member must provide a **brief summary** of their **individual contributions** to the project and each member must independently answer the following questions based on the shared table above:

1. How did you **represent** the map as a graph?

   Explain which graph representation you used and how you determined the relationship between vertices include the edges and their weights.

2. What is **complexity** of your implementation of Dijkstra's Algorithm (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)?

   Explain this by describing which data structures you used and how you used them to implement path finding.

3. How well does your implementation **scale**?

   Explain what effect did `N` (ie. the size of the map) have on the performance of your `dijkstras` path finding component in terms of execution time and memory usage?

Submission instructions

Please have one team member submit an archive of the following files (not in a subdirectory of the archive) on Canvas to aid us in grading:

1. source files for dijkstras
2. source files for generate_map
3. README.md with required table
4. Text document summarizing the student who is uploading's contributions and their answers to the required questions
5. Makefile (even if you use the provided Makefile, please include it) that does the following:
   - compiles your code into an executable called “dijkstras”

- creates an executable called generate_map from your generate_map code (either by compiling it if you write in a compiled language or by running chmod to make it executable if you didn't write in a compiled language)

If you work in a pair, the team member who isn't submitting the above files should write up a separate document with their contributions/answers (see above) and submit only that on Canvas.

Rubric

Your project will be scored as follows:

```
+2    Code is well formatted, commented (inc. name, assignment, and overview), with reasonable variable names
+5    Passes memory tests
+20   Test cases successfully solved (5 points each)
+10   Generate_map works properly
+8    Table and responses, inc. individual contributions
```

We reserve the right to take off up to three points for submissions that do not follow the new requested format designed to streamline TA grading effort for the initial pass.