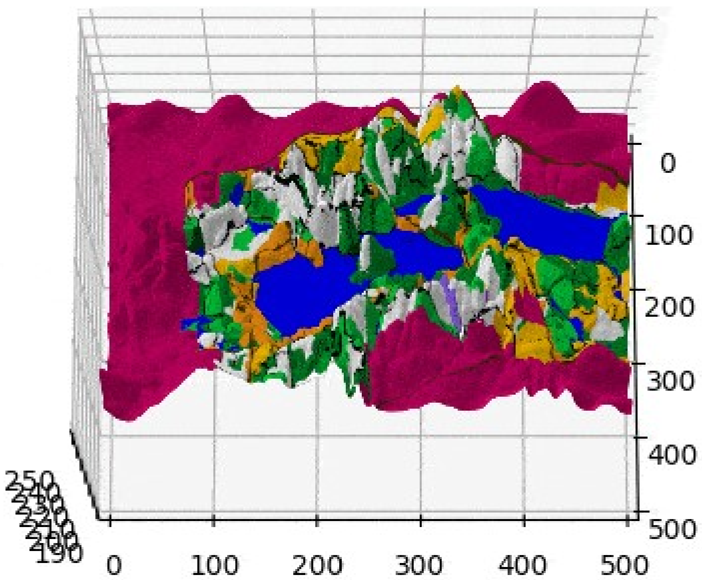
Lab 1 : Summer Orienteering

In this lab, you will be generating optimal paths for orienteering. In the sport (or "activity", depending on your level of fitness/competitiveness) of orienteering, you are given a map with terrain information, elevation contours, and a set or sequence of locations to visit ("controls"). There is a combination of athletic skills and planning skills required to succeed - a smarter competitor who can figure out the best way to get from point to point may beat out a more athletic competitor who makes poor choices! In this lab, you are given computer-friendly inputs so that you can use an algorithm to determine the best path for you to take depending on your ability.



Credit: Eric Dudley

The map(s)

In an ordinary orienteering event, the map you get will be quite detailed. Different background colors show the type of terrain (see the table below), while buildings, boulders, man-made objects and other notable features are shown with different symbols. For this assignment, you are given two separate inputs, both representing Mendon Ponds Park: a text representation of the elevations within an area (500 lines of 400 double values, each representing an elevation in meters) and a 395x500 simplified color-only terrain map (color legend below). To address the difference in width between the elevation and terrain files you should just ignore the last five values on each line of the elevation file. Also, the real-world pixel size is determined by that of the National Elevation Dataset, which in our area is one third of an arc-second, equivalent to 10.29 m in longitude (X) and 7.55 m in latitude (Y). **You must take these dimensions into your account of distance.**

The basic event

As for the points you will need to go visit, those will come in a simple text file, two integers per line, representing the (x,y) pixel (origin at upper left) in the terrain map containing the location. In the classic event type that we are considering, the sequence of points must be visited in the order given. One such classic event was the World Deaf Orienteering Championships, held in Mendon Ponds Park. The local club also had an event using the same courses, the results and maps of which can be seen here. I have done my best to convert the white, brown and red courses into the relevant text files.

The planning

So, you have to get to some controls. However, going in a straight line, even if possible, is often not advisable. First of all, you will be able to proceed at different speeds through different terrains. Rather than telling you how fast, you need to decide based on some representative photos how fast you can travel through these terrains:

Terrain type	Color on map	Photo (legend)
Open land	#F89412 (248,148,18)	A
Rough meadow	#FFC000 (255,192,0)	B
Easy movement forest	#FFFFFF (255,255,255)	C · D
Slow run forest	#02D03C (2,208,60)	E
Walk forest	#028828 (2,136,40)	F
Impassible vegetation	#054918 (5,73,24)	G
Lake/Swamp/Marsh	#0000FF (0,0,255)	H · I · J
Paved road	#473303 (71,51,3)	K · L
Footpath	#000000 (0,0,0)	M · N
Out of bounds	#CD0065 (205,0,101)	

Then we get to the planning. This is a large environment, so while breadth-first search might be acceptable for individual paths, it is much better (and not much harder!) to implement an A* search to handle complete events. However, consider your heuristic function carefully. Showing that it is admissible (or it is not quite admissible, but you can bound its error) is another important part of your writeup. Note that if any of these alterations would make your heuristic inadmissible, you should change it!

Input

Name your program 'lab1'. It should take 4 arguments, in order: terrain-image, elevation-file, path-file, output-image-filename.

Python would look like, $python3 lab1.py terrain.png mpp.txt red.txt redOut.png

Output

You should output an image of the input map with the optimal path drawn on top of it. You should also output the total path length in meters either to the terminal or drawn on the map itself. Here is an example path for the brown trail. Note that your solution may produce a different result and still be correct. This is especially true on the park map where our terrain penalties will differ. The test cases linked below have less variance and your output should match those more closely.

Writeup

As discussed above, your writeup should include all relevant decisions made in the implementation of your code. That is, how you implemented your cost function, how you implemented your heuristic, and justification for their correctness.

Some hints/tips

- You are welcome to write your solution in Python, or Java. If you would like to use a different language, please consult with me first. Regardless, your program should run on a Linux-based CS machine, and your code should operate there without modification.
- In Python, you can use the Python Image Library to both read an image in to an array of pixels as well as to modify those pixels to output your results. (I used `Image.open()`, `.load()` and `.save()` on the CS machines.)
- In Java, you can use the ImageIO class to read in an image into a BufferedImage object and get/set pixels from there.
- You are welcome to hard-code things which make sense to hard code, such as the color values. Be careful if you hard code file names that it will still work when downloaded and run on a different machine.
- Some overall suggestions to approach the work: First just get a single simple path planned. Here is a simple one, but try some others and some longer ones. Get the graphical output working first, it will help you debug everything.
- A pixel's neighbors can be defined as either the 4 pixels that share an edge (the cardinal directions) or the 8 pixels that share either an edge or a corner (the cardinal directions and north-east, north-west, etc). Either is acceptable and test cases have been generated for both.
- You do not have to do anything fancy like moving faster or slower depending on elevation - it is sufficient for the elevation to simply add to the 3d distance between pixels. For those who insist on doing something beyond that, get everything working with a simple handling of elevation first before trying anything special.

Grading

- Proper interpretation of input files: 5%
- Proper A* search algorithm and heuristic: 25%
- Testcases: 20%
- Efficiency of solution*: 15%
- Human-readable output (drawing path and reporting distance): 20%
- Writeup: 15%

* As a point of comparison, most solutions in Python produces a plan for the red course above in about 3 seconds with a conservative admissible heuristic. If your program takes longer than 20 seconds there is likely something wrong with your heuristic. Path finding on the testcases that serpentine often will take longer. Anything over a minute indicates you are using inefficient data structures.

Testcases

Check these tests cases out! If your solution runs in a reasonable amount of time and matches the supplied output then you are *probably* fine.