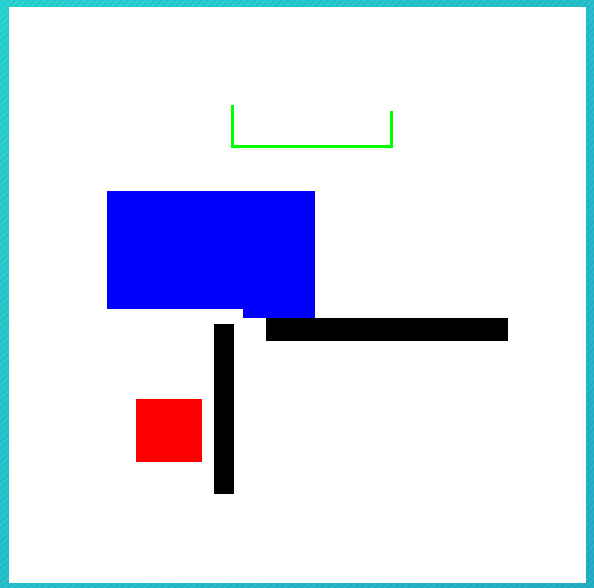
Assignment 02: Pathfinding

Brock Davis

Goal

- Given an array with defined costs between each cardinally adjacent element, return and highlight the lowest cost path between any two points

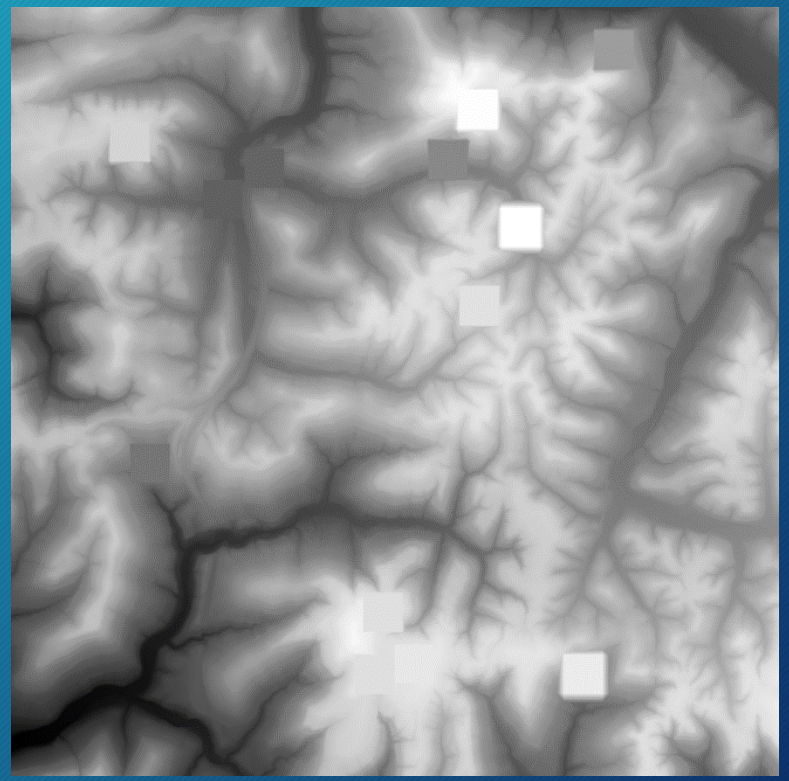
Input



Input 1:
H: 200 px
W: 200 px
Cost to move:

- 1 on white
- 2 on blue
- infinite on black

Input 2:
H: 10024 px
W: 10024 px
Cost to move:
1 + (diff in value)^2



Approach

- A more optimized version of Dijkstra's Algorithm was implemented
- A set of points w/ adj. points to be evaluated were kept in fringe
- The points in the fringe with the lowest values had adj. points evaluated and added to fringe
- The need to linearly search through the fringe for every point was eliminated with a counter with the lowest cost

```
c=0
while fringe not empty:
        for point in fringe with cost c:
                evaluate cost of adj points
                add adj to fringe
                remove point
```

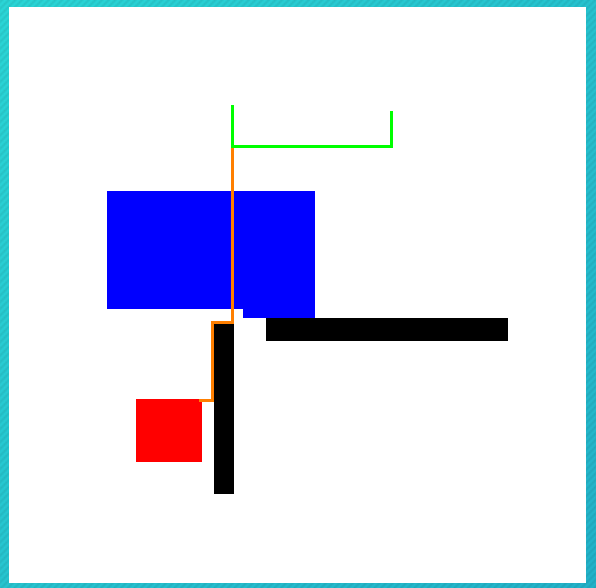
Pseudocode

```
c=0
while fringe not empty:
        for point in fringe with cost c:
                evaluate(adj(point))
                fringe.add(adj(point))
                fringe.remove(point)
        c++
```

Also Implemented

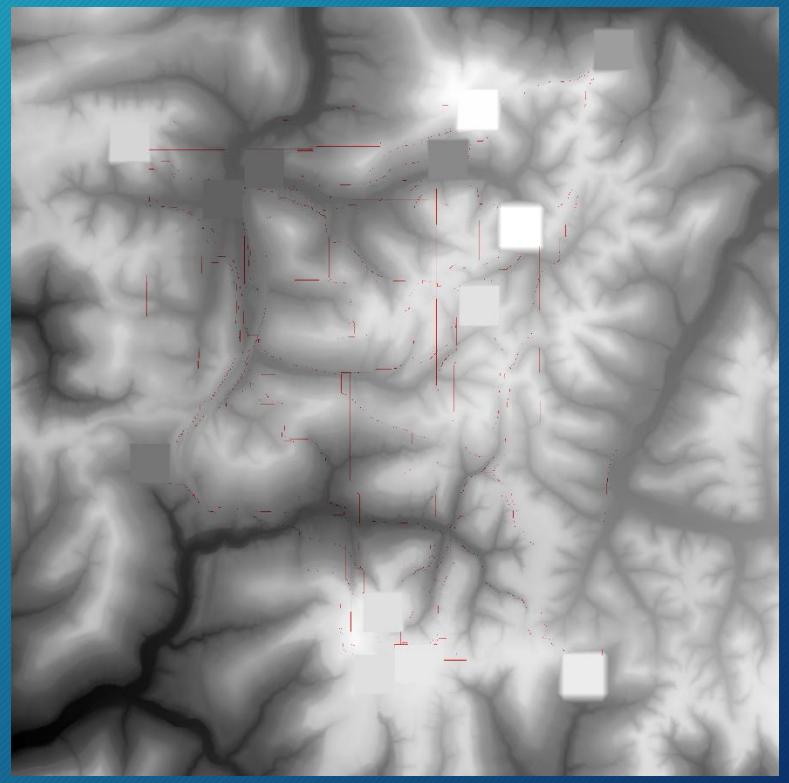
- When the adjacent points were evaluated, they should not be added to the list of points right away because the array is being iterated through while the adjacent points are being evaluated
- After all points have been evaluated, the goal is traced back to 0, starting with the lowest cost in the goal and moving to the lowest cost adjacent cell until 0 is reached

Results

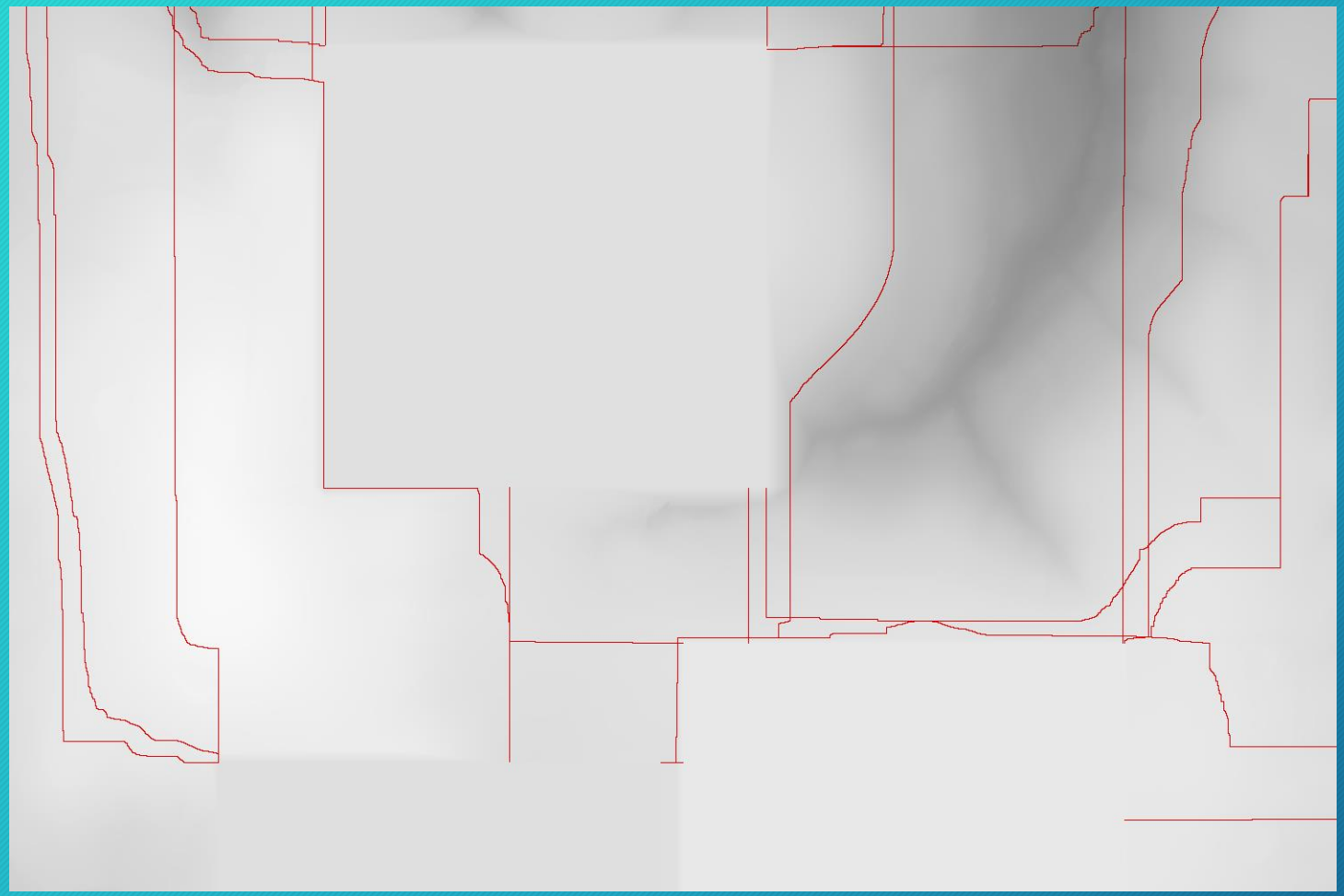


Output 1:
Lowest cost: 140
Path highlighted in orange
Approx time: 1 sec on lab computer

Output 2:
Costs vary between cities
Path highlighted in redscale
Approx time: 40 min on lab computer



Results (cont.)



Output 2 (Zoomed)

Conclusion

- The path costs were verified to be correct, so the program was successful
- However, the program took ~40 min to complete on a desktop computer, so optimization would be needed to scale up to more connections between nodes or a larger map
  - This could be done by calculating all costs between nodes and then using a preexisting pathfinding algorithm
  - Also adding a heuristic could greatly improve time of pathfinding to a certain point