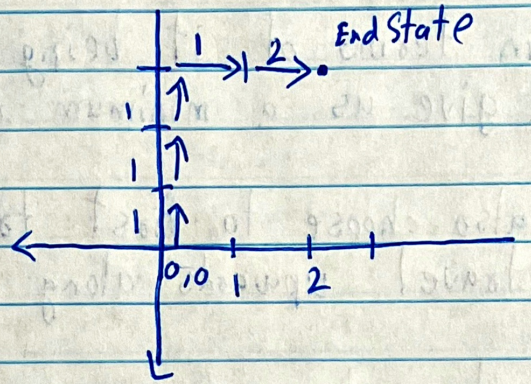
1a. In our given example, our starting point is (0, 0).
It is also clearly defined that our end state i.e. (m, n) has $m, n \geq 0$

The cost function states that $Cost((x,y), a) = 1 + max(x, 0)$
This can be interpreted as, the cost increasing and it being more expensive as we move towards the positive x axis

Assume our end state was (2, 3)



Moving along y axis would have constant cost of 1 for each action.
Moving along x axis would have a cost that would increase by the x value of the current state.

Since cost increases as we go to the right along x axis, it would be better to first get to co-ordinate 'n' along y axis. Then go right until we get to 'm'.

Cost of Moving along y axis to n = $1 \times n$
Cost of Moving along x axis to m = $(1+0) + (1+1) + (1+2) \ldots (1+m)$
$= \sum_{x=1}^{m} (1+x)$

Minimum Cost Expression = $1 \times n + \sum_{x=1}^{m} (1+x)$

As we deduced previously, the best possible path to achieve minimum cost would be travelling along y axis first as each step would have a constant cost of 1 until we reach 'n'.
Once we reach n, we can then travel along x axis until we reach 'm'.

This path would be unique if we were to consistently first travel upwards and then to the right.

However, in terms of it being the only path that would give us a minimum cost, this would not be true.

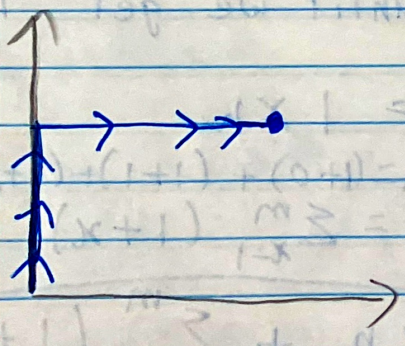
We could also choose to first travel along x axis and then travel upwards along y axis.

We could also choose a zig zag path where we would alternate between moving right and up.

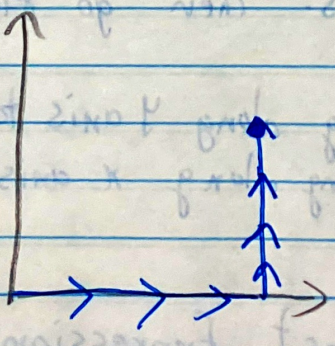
As long as we don't take any unnecessary turns, all paths would lead to the end state with minimum cost.
Yes, there are multiple paths that would achieve minimum cost.

Possible Minimum Cost Paths.

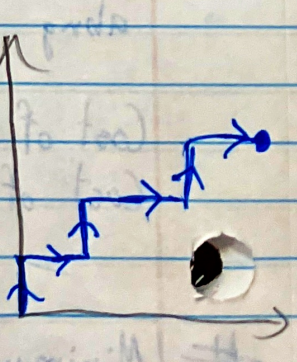
1.



2.



3.



1. Grid City

b)

i. UCS will never terminate because the number of states is infinite.
Ans: False. Since there is a positive constant cost associated with every state, and there is a finite reachable goal, UCS will terminate.

ii. UCS will return the minimum cost path and explore only locations between (0, 0) and (m, n); that is, (x, y) such that $0 \leq x \leq m$ and $0 \leq y \leq n$.
Ans: False. It will explore locations and states possible based on actions.

iii. UCS will return the minimum cost path and explore only locations whose past costs are less than the minimum cost from (0,0) to (m,n).
Ans: True.

c)

i. If you add a connection between two locations, the minimum distance cannot go up.

Ans: True. A new connection could open up paths that were not previously available. However, this new connection would show up in the minimum path only if the path's total cost is smaller than the previous minimum path. Thus, the minimum distance could decrease but never go up.

ii. If you make the cost of an action from some state small enough (possibly negative), that action will show up in the minimum cost path.
Ans: False. Even though the cost of an action is very small, it might lead to a state that ends up increasing the overall cost. Thus, it cannot be guaranteed that this action would show up in the minimum path.

iii. If you increase the cost of each action by 1, the minimum cost path does not change (even though its cost does).
Ans: True. Since the cost of every action increases by 1, the minimum path would not change because the costs of all paths would increase by the same ratio.

2. Finding Shortest Paths

b)

i.

Ans: From this map data visualization, I see that the Rains Apartments landmark is at the bottom right corner of the map and is one of the outermost landmarks. Also from the data, there are **22 different locations** with the **parking_entrance** tags. As expected, some of the farthest parking entrances from the rains landmark are at the opposite corner of the landmark in the top left of the map. I was able to find the distance of rains landmark from a few different parking entrances and these are some of the top ones I found. I have also attached the screenshot of the farthest parking entrance from Rains apartment.

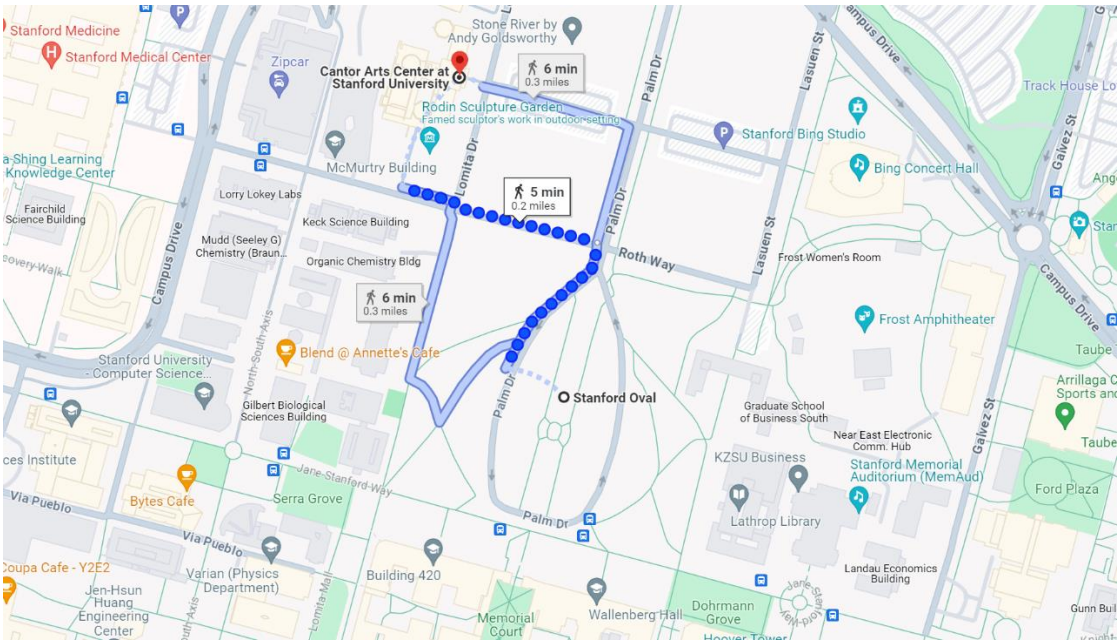
- Cost between parking_entrance at location **6443368816** and landmark **rains** is **2987.240569586965**
- Cost between parking_entrance at location 540510207 and landmark rains is 2927.8526535393294
- Cost between parking_entrance at location 6318792955 and landmark rains is 2889.3365657245013
- Cost between parking_entrance at location 1330035391 and landmark rains is 2836.5764316936834
- Cost between parking_entrance at location 2572379215 and landmark rains is 2122.2906297448294

Route from Rains Apartment to parking_entrance at location 6443368816

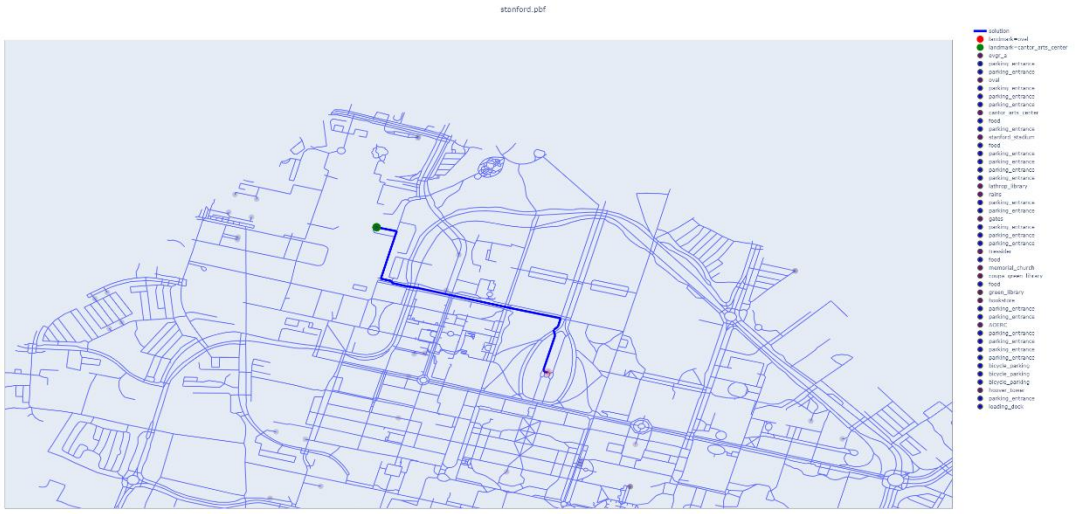
ii.

Ans: From the data used to generate routes around the Stanford map, I found that the **landmark - cantor arts center** is not mapped correctly. Its location is considerably farther from the Stanford oval than expected.

a. This is the expected route from Stanford Oval to Cantor Arts Center from Google Maps.



b. This is the route and location obtained from map data and model



3. Finding Shortest Paths with Unordered Waypoints

b)

Ans: If there are n locations and k waypoint tags, every time the algorithm explores a given location, it would check if the k waypoint tags have been satisfied. Every waypoint could either already been visited or not visited. Thus, there would be 2^k possibilities of waypoint states in every new location.

The maximum number of states UCS could visit = $n * 2^k = 2^k n$

c)

Ans: For my starting location I picked the outermost parking entrance with location tag **6443368816**. My end location was the landmark **Rains** apartments. Initially I wanted to add the entire list of landmarks that were available into my waypoint tags. This would be 14 waypoint tags. Since there are 8193 unique locations, my algorithm would have to test **8193 * 2^{14}** ($n * 2^k$) states. As expected, it crashed my computer and I had to restart. I reduced the number of waypoint tags to 7. Here is the list of waypoint tags I used –

[gates, bookstore, oval, memorial church, coupa green library, hoover tower, tressider]

The total cost of visiting all these waypoint tags is 4691.520269602774 which is almost 1.5 times the cost of going from start to end without visiting the waypoint tags. Also, the route always seems to visit the next closest waypoint from its current location. This seems to be the most optimal route to visit all waypoint tags on the way from start to finish.

4. Speeding up Search with A*

d)

Ans: An example of how using the A* with NoWaypointsHeuristic would result in the same running time complexity as solving the relaxed shortest path problem would be when the way points are part of the shortest path or equidistant from points on the shortest path while going from start to end.

Example of such waypointTags would be:

i. **[“1,1”, “2,2”, “3,3”, “4,4”, “5,5”, “6,6”, “7,7”, “8,8”]**
ii. **[“0,1”, “0,3”, “0,5”, “0,7”, “0,9”, “1,9”, “3,9”, “5,9”, “7,9”, “8,9”]**