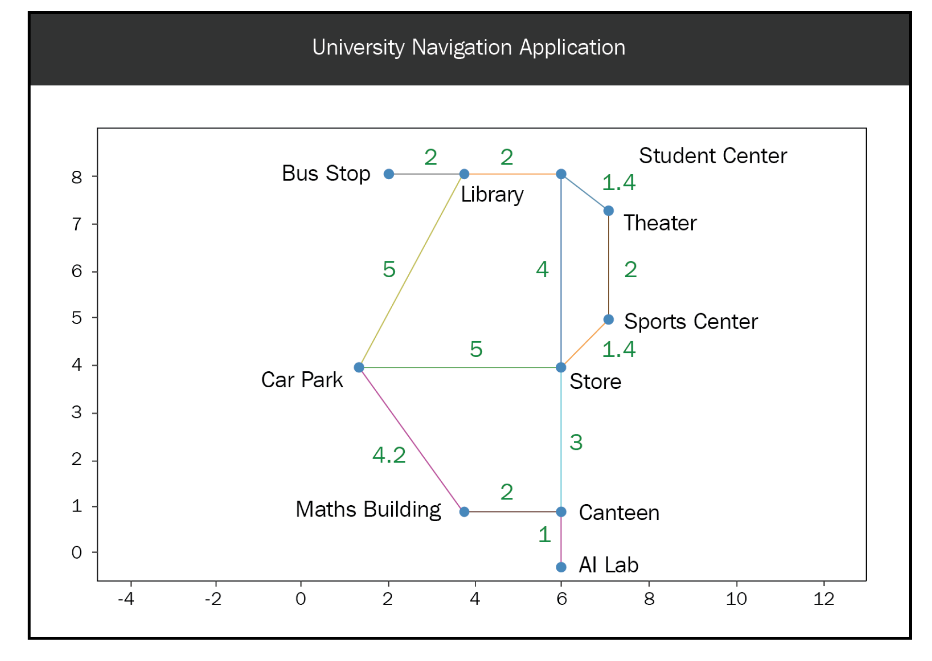
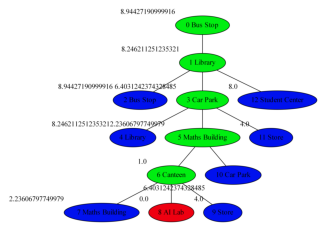
**Name: Nestor Romero Leon**

**StudentID: 301133331**

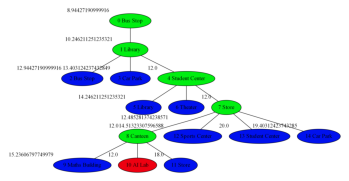
**Exercise # 2: Map search**

**University Map Layout**

**GREEDY TEST RESULT**

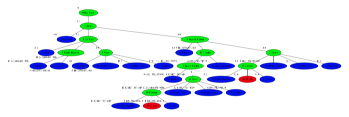
**Path**

-> Bus Stop -> Library -> Car Park -> Maths Building -> Canteen -> AI Lab

**A\* TEST RESULT**

**Path**

-> Bus Stop -> Library -> Student Center -> Store -> Canteen -> AI Lab

**UNIFORM COST SEARCH RESULT**

Path

-> Bus Stop -> Library -> Student Center -> Store -> Canteen -> AI Lab

**SUMMARY ANALYSIS**

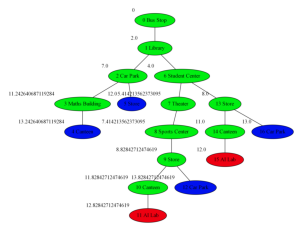
For the **Uniform Cost Search**, I removed the heuristic from the code and only used the **costFromRoot** value as the priority value for the queue.

As expected from the design of the algorithm, the **Greedy Search** Tree is narrower focusing rapidly on the smallest distance to the goal by taking the “Car Park” route

Although the **A\* algorithm** seems to have the same depth as the Greedy, it creates a wider search tree when considering the both the cost and the distance to goal in the priority assignment. This tree however is the more balanced tree of all

Finally, the Uniform Cost Search, while using only the **costFromRoot** value for priority and therefore navigation of the tree, starts small, but then towards the middle of the tree starts expanding considerably more than its counterparts and also going a couple more layers deep as the algorithm has to start traversing the tree searching for the lowest cost to continue looking for the goal node. The Uniform Cost Search end up finding the same path as A\* but by looking at the tree, the resource consumption was considerably higher

**ADDITIONAL TEST**

If loops are removed from the Navigation Data >> Connections Dictionary, the Uniform Cost Search algorithm does a better job

**Exercise # 3: A\***

a)

g(n) = 0

h(n) = 4

f(n) = 0+4 = 4

2

8

3

1

6

4

7

5

g(n) = 1

h(n) = 5

**f(n) = 1+5 = 6**

2

8

3

1

6

4

7

5

2

8

3

1

6

4

7

5

g(n) = 1

h(n) = 5

**f(n) = 1+5 = 6**

2

8

3

1

4

7

6

5

g(n) = 1

h(n) = 3

**f(n) = 1+3 = 4**

g(n) = 0

h(n) = 4

f(n) = 0+4 = 4

g(n) = 4

h(n) = 0

**f(n) = 4+0 = 4**

1

2

3

8

4

7

6

5

g(n) = 3

h(n) = 1

**f(n) = 3+1 = 4**

2

3

1

8

4

7

6

5

g(n) = 2

h(n) = 2

**f(n) = 2+2 = 4**

2

3

1

8

4

7

6

5

b)

I think it is not a good heuristic because at each turn there is only one move possible, and that move will increase the depth of the tree by one. However, the heuristic will keep decreasing so the net effect will not be seen. In the previous example the last 3 steps have the same f(n), even if the agent was getting close to the solution, this heuristic could cause additional branching by not focusing the search.

c)

If I define h’(n) as the number of well-placed tiles, and instead of minimizing the number of misplaced tiles, the agent tries to maximize the correct placed ones, at each turn and each level down the tree, the heuristic will increase in addition to the depth and point the agent towards the desired final state. In point a, if h’ was used, the last 3 steps could have been f(n)=7, fn=8 and f(n)=9