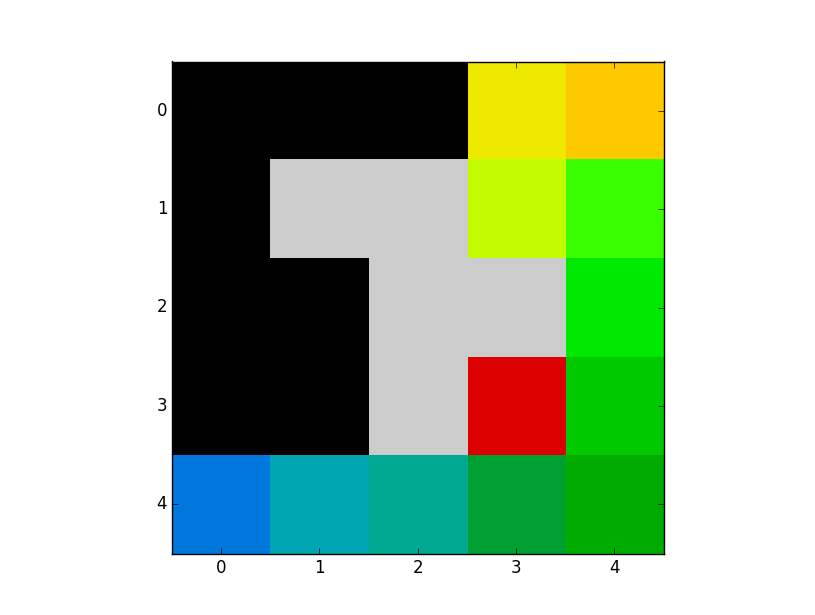
In order to run BFS the python files that are created are BFS.py and graph\_search.py

graph\_search.py file is imported into BFS.py

Note : Diagonal movement graph can be obtained by changing the argument in DFS.py from

‘\_ACTIONS ’ -🡪 ‘\_ACTIONS\_2

**Question 1.1** Performing BFS on map0



Path: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (3, 4), (2, 4), (1, 4), (1, 3), (0, 3), (0, 4)]

Set of states visited : [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (3, 4), (3, 3), (2, 4), (1, 4), (1, 3), (0, 3), (0, 4)]

It is not the shortest path. Because one can go directly to goal (0,4) from (1,4)

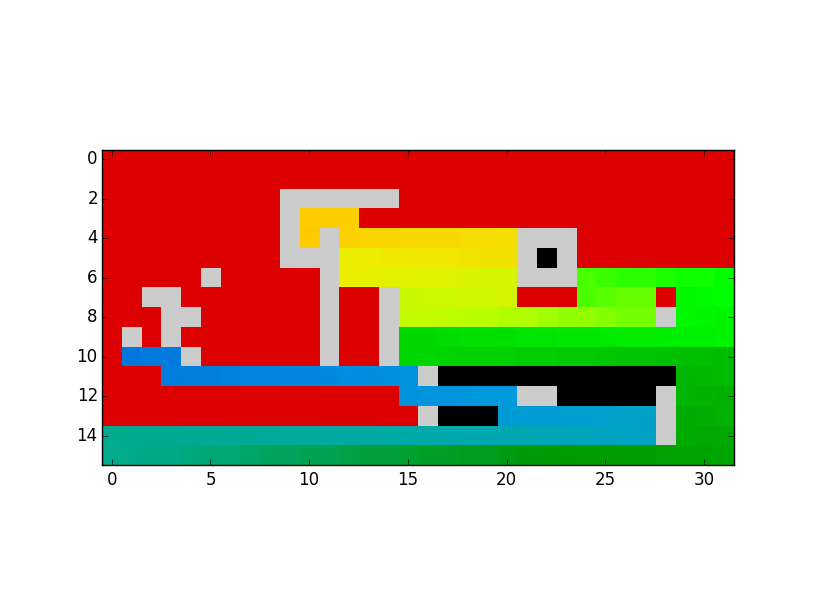
But it is going (1, 4), (1, 3), (0, 3), (0, 4)

Reason : In dfs right element in the frontier is always pop first . And priority of actions that we considered is right>left>down>up(( last in first out i.e right most element ). So when ever possibility is there it will explore right most node followed by left, down, up.

**Question 1.2** Performing BFS on map1

Path, Set of visited nodes too many states to write here.

They can be seen by running the program DFS.py



Initial state is Blue and final is yellow

Clearly we can see it is not the shortest path. As said earlier it is giving priority to right element ( last in first out i.e right most element ) . ). So when ever possibility is there it will explore right most node followed by left, down, up.

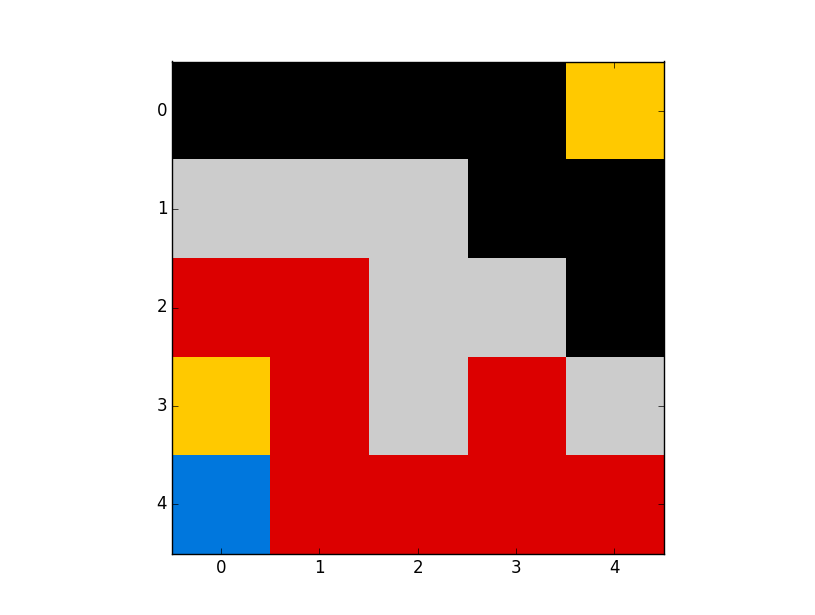
If it doesn’t find path to its right side it will consider left>down>up.

Note : Diagonal movement graph can be obtained by changing the argument in DFS.py from

‘\_ACTIONS ’ -🡪 ‘\_ACTIONS\_2

**Question1.3** Performing BFS on map2

pathp



BFS\_map3\_visited\_nodes: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (3, 3), (3, 1), (3, 0), (2, 0), (2, 1)]

Since the actions we have selected are up, down, left, right .There is no path to the goal from initial to final goal.

**Question 1.4**

By adding all search nodes for all actions of a given node at once will creates duplication and creates a problem while performing iterative deepening DFS.

For example if we are exploring (4,1) during its left action its updates frontier with (4,0). But (4,0) is visited/explored already. Although ‘ n\_i not in visited’ removes it . Its unnecessarily increasing computations. Secondly I am interested in finding cost it updates wrongly.

(4,0) 🡪(4,1)🡪 (4,0) will update cost of 2. If one consider cost of each step is 1.

During iterative deepening of DFS since popping from right side a node may be visited path before the shorted path and get updated in the visited list since node is visited already short path will not get executed. And also one has to compare cost because the longest path will give more cost than shortest if it is not visited. One has to write a check points.

Note: I have implement code for iterative deepening of DFS with all constraints.

Problems like given might have been in the frontier but with more cost.

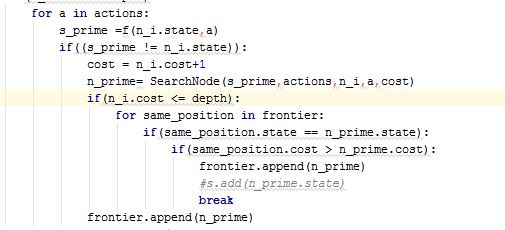
And problems like given node already visited with more cost so one can’t explore with shorted paths.

These can be alleviated by writing proper check point like

**if**(s\_prime != n\_i.state)

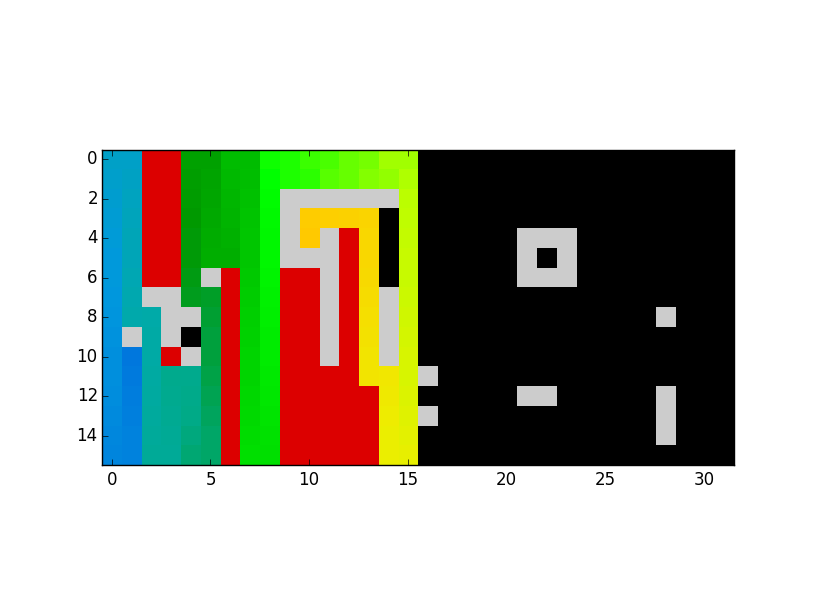
frontier.append(n\_prime)

for comparing the cost and if the node already present in frontier.

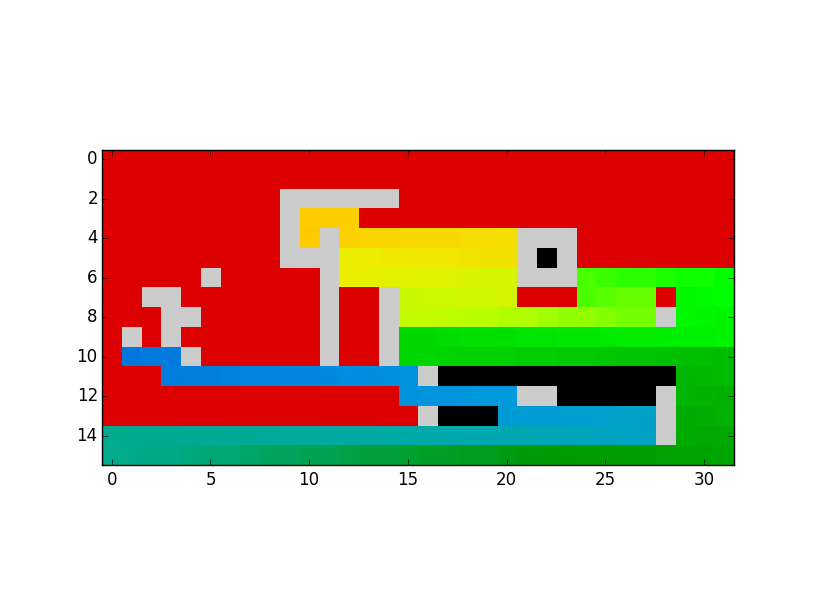


**Question: 1.5**

BFS on map1 with reverse actions



BFS on map1

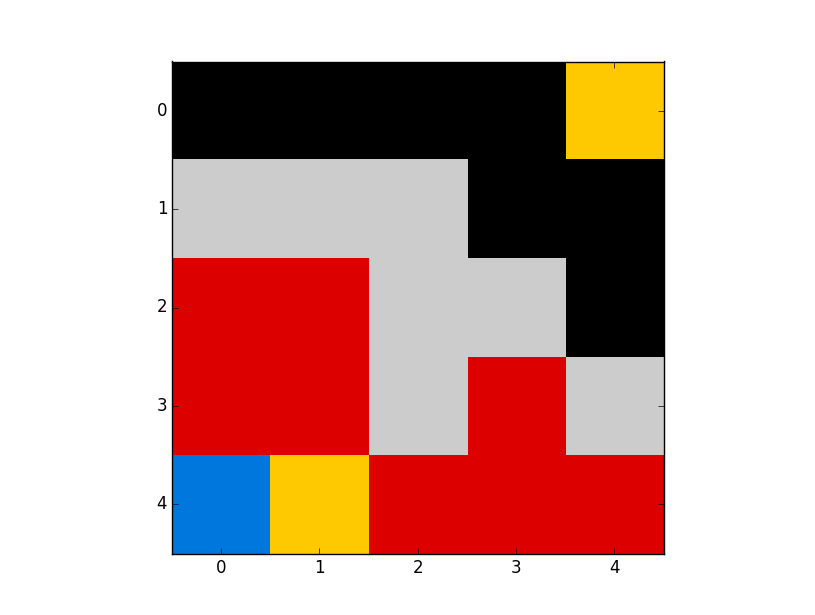


We can clearly see reverse action the visited nodes are very less. More black colour area which implies unvisited nodes unlike in normal case.

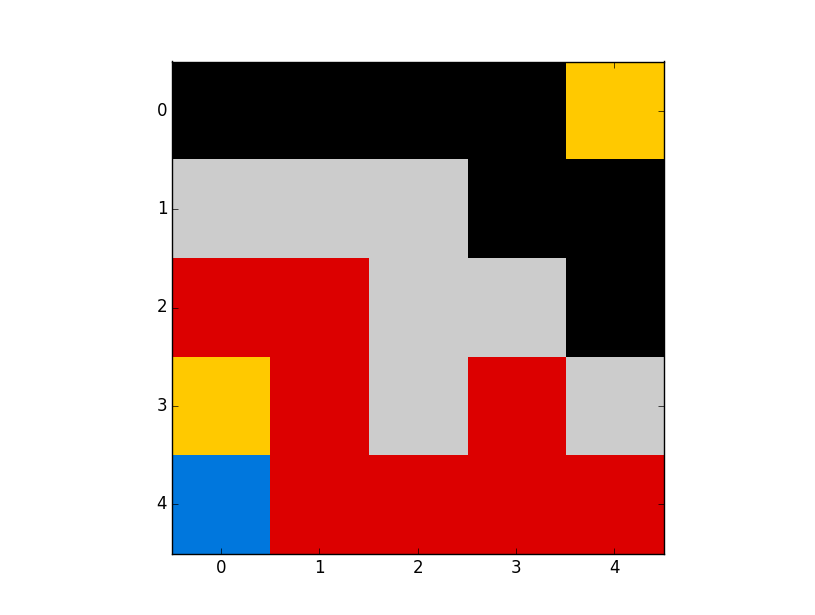
And also path is aligned vertically since up movement is given priority. Where in normal operation i.e when right movement is given more priority path is aligned horizontally.

Its actions priority changes the course of path and visited nodes before reaching the goal.

BFS on map2 with reverse actions



BFS on map2



In map after reversing the actions we don’t find much difference except initial and final position(not goal)since goal and initial are separated by boundaries and we are not considering cross path actions. It explore all possible ways/nodes .

**Question: 1.6**

DFS on iterative deepening :

Run iterative\_DFS.py

Along with iterative\_graph\_search.py (with iterative\_graph\_search.py is imported in iterative\_DFS)

Iterative deepening of DFS on map1:

The graph becoming more like a breadth for search and it is widening its area. Since it is exploring all the levels one by one its gives shortest path.

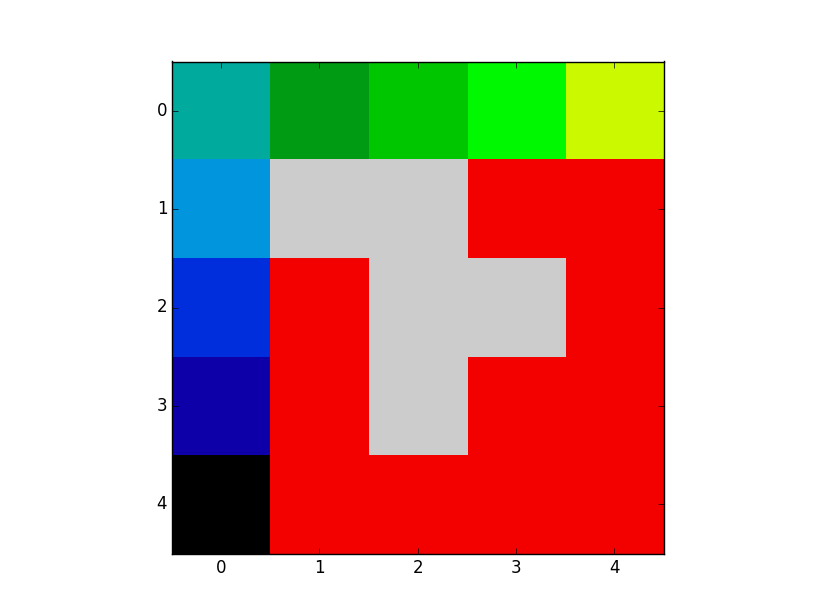
Iterative deepening of DFS on map2:

If path exist between initial and goal it will give shortest path. But here they are separated by boundaries.

Question: 2

**Question: 2.1**

BFS on map0



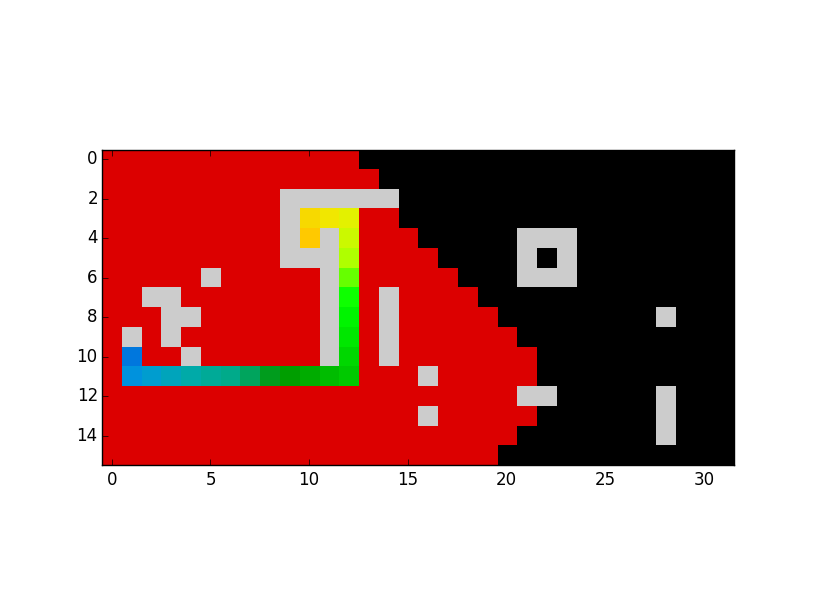
BFS on map0 Path : [(4, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0, 4)]

BFS on map0 visited nodes: [(4, 0), (3, 0), (4, 1), (2, 0), (3, 1), (4, 2), (1, 0), (2, 1), (4, 3), (0, 0), (3, 3), (4, 4), (0, 1), (3, 4), (0, 2), (2, 4), (0, 3), (1, 4), (1, 3), (0, 4)]

The path found is shortest. Since BFS moves level by level. Its visits all possible node before reaching the goal.

**Question: 2.2**

BFS on map1



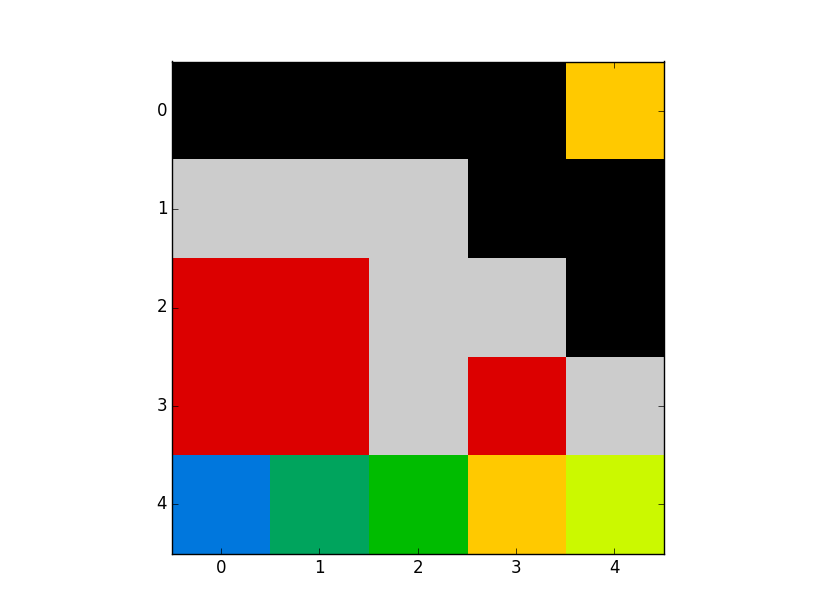
BFS on map1 Path : [(10, 1), (11, 1), (11, 2), (11, 3), (11, 4), (11, 5), (11, 6), (11, 7), (11, 8), (11, 9), (11, 10), (11, 11), (11, 12), (10, 12), (9, 12), (8, 12), (7, 12), (6, 12), (5, 12), (4, 12), (3, 12), (3, 11), (3, 10), (4, 10)]

BFS on map1 visited: It visited so many nodes to append them here.

The path found is shortest. Since BFS moves level by level. Its visits all possible node before reaching the goal.

**Question: 2.3**

BFS on map2



Since path is not available BFS explores all possible nodes. If path does exists it will give shortest path.

Question: 2.4

In DFS (regular) the visited/explores nodes are very less in map0. when compared to iterative and BFS. But it will not give shortest path. map0 is very small and we are lucky that goal is aligned with priority of actions considered hence it took very less visited nodes when compared to other. But if goal is not aligned with actions priority we are considering we might end up long path and more visited nodes when compared to others.

In iterative deepening DFS it goes first in the direction of more priority until it reaches maximum deepening specified. And according to the priory it searches all the levels.

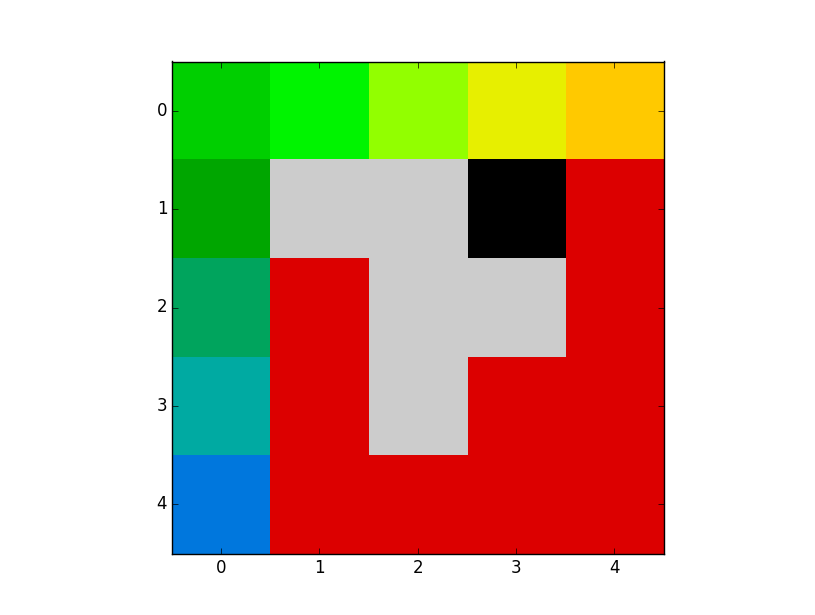
Visited nodes are same in both idDFS and BFS. Only thing is BFS moves more uniformly. Although idDFS covers all nodes but it moves biased.

Although BFS takes more time, BFS is easy to implement. Because as said in question 1.4 one need to update cost/path length and keep a check for visited node in idDFS.

QUESTION -3

**Question -3.1**

Uniform Cost Search:



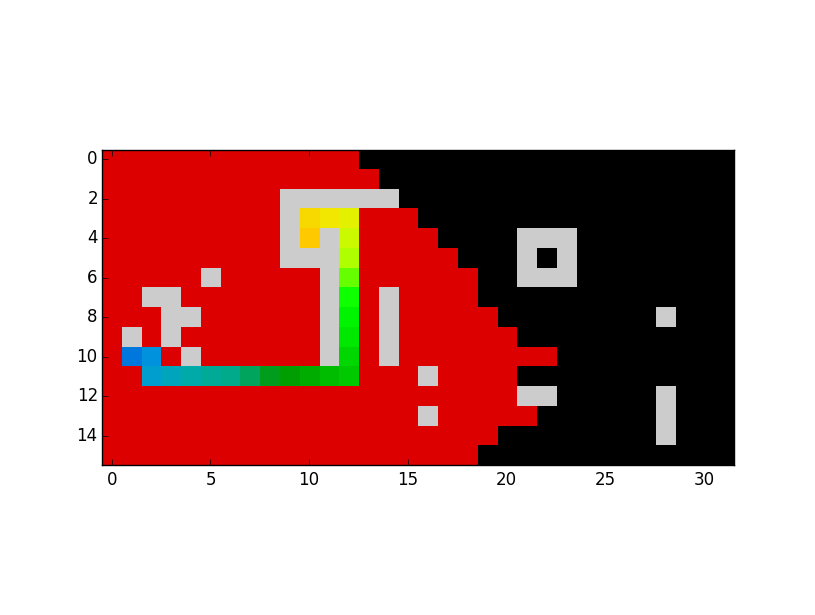
Uniform cost search onmap0:

Path: [(4, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0, 4)]

Visited: [(4, 0), (4, 1), (3, 0), (2, 0), (4, 2), (3, 1), (4, 3), (2, 1), (1, 0), (0, 0), (3, 3), (4, 4), (3, 4), (0, 1), (0, 2), (2, 4), (0, 3), (1, 4), (0, 4)]

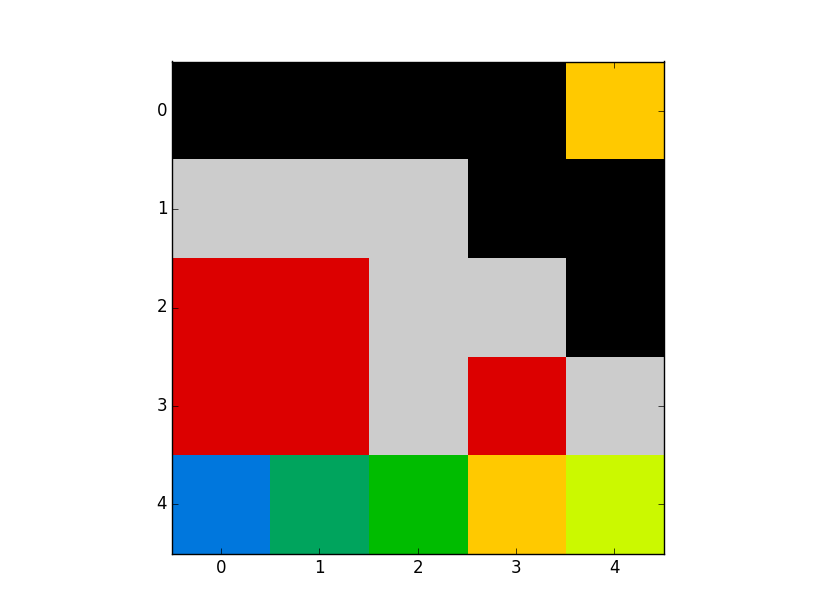
Yes the path found is the lowest cost and it is the shortest. Uniform cost search visits all lowest cost nodes around it. In this case it gave shorted path since costs are same to each node of one step.

**Question -3.2**



Yes the path found is the lowest cost and it is the shortest. One can see that so many nodes visited before reaching the goal. They are so many nodes of equal coat in graph hence it visited so many nodes.

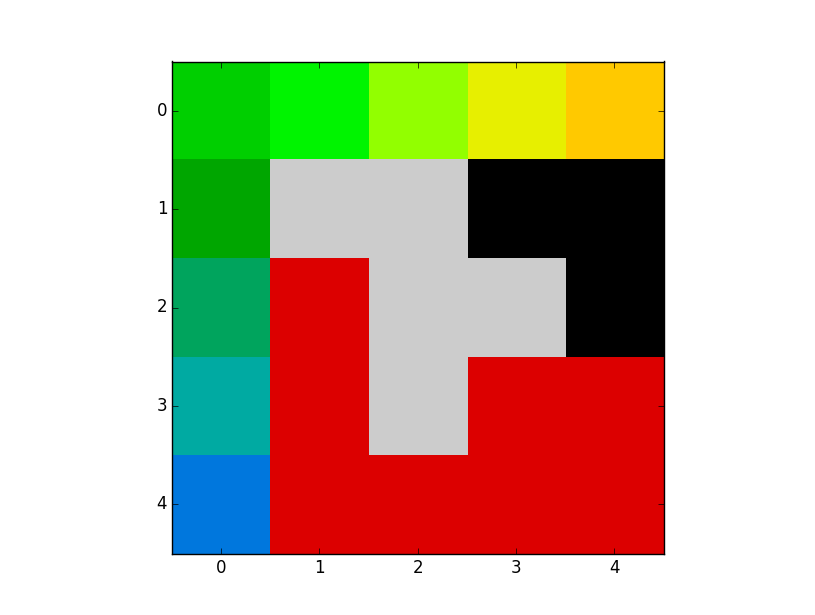
**Question -3.3**

****

Uniform cost has explored all possible nodes in frontier. Since path is not there path hasn’t found. It moved uniformly in all direction until to reached obstacles.

**Question -4**

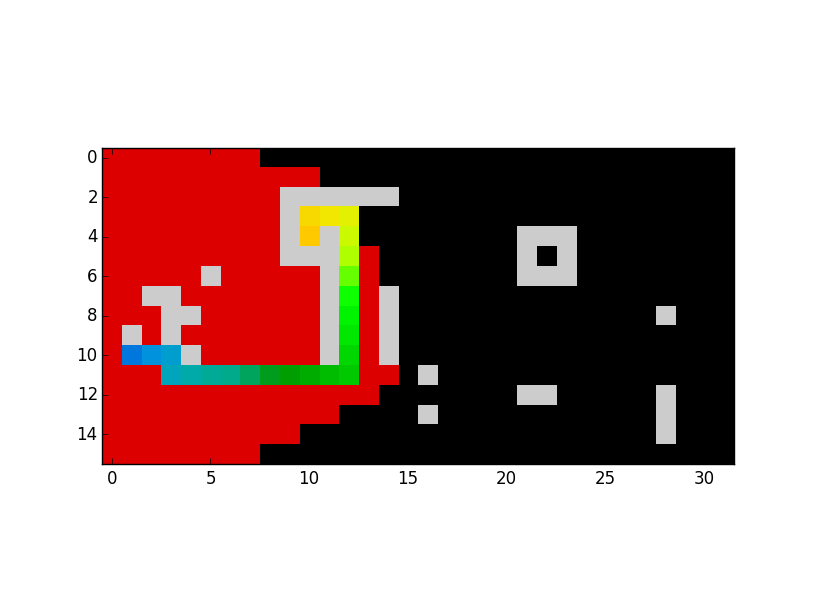
Question -4.1

map0

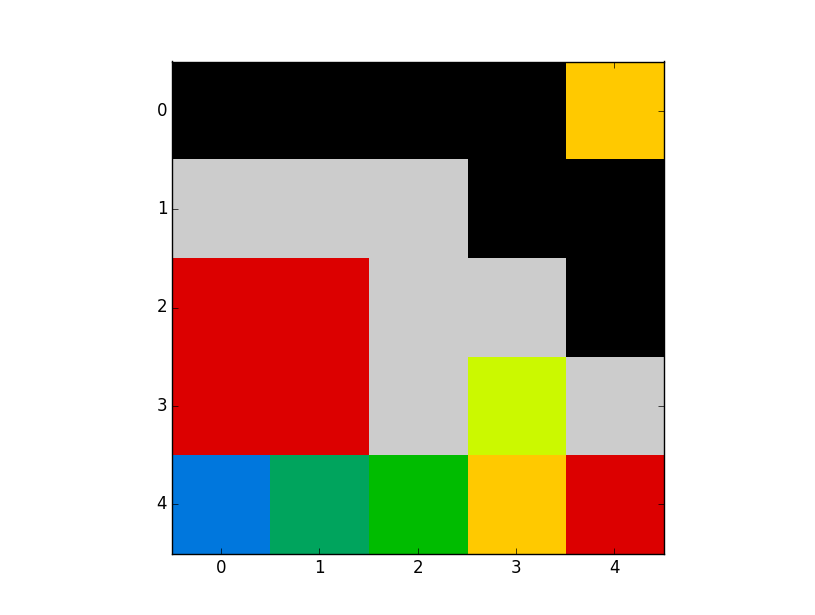
A\* with Euclidean distance heurist

Path on Map0: [(4, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0, 4)]

Visited Nodes on Map0: [(4, 0), (4, 1), (3, 0), (3, 1), (2, 0), (4, 2), (2, 1), (4, 3), (1, 0), (3, 3), (0, 0), (0, 1), (3, 4), (4, 4), (0, 2), (0, 3), (0, 4)]

map1

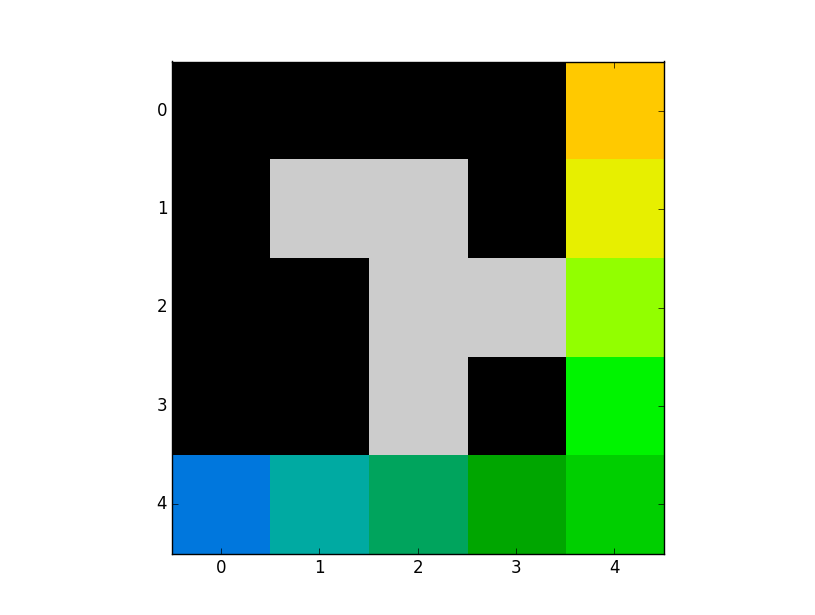
Path on Map1 : [(10, 1), (10, 2), (10, 3), (11, 3), (11, 4), (11, 5), (11, 6), (11, 7), (11, 8), (11, 9), (11, 10), (11, 11), (11, 12), (10, 12), (9, 12), (8, 12), (7, 12), (6, 12), (5, 12), (4, 12), (3, 12), (3, 11), (3, 10), (4, 10)]

map2

The path remained same in all cases

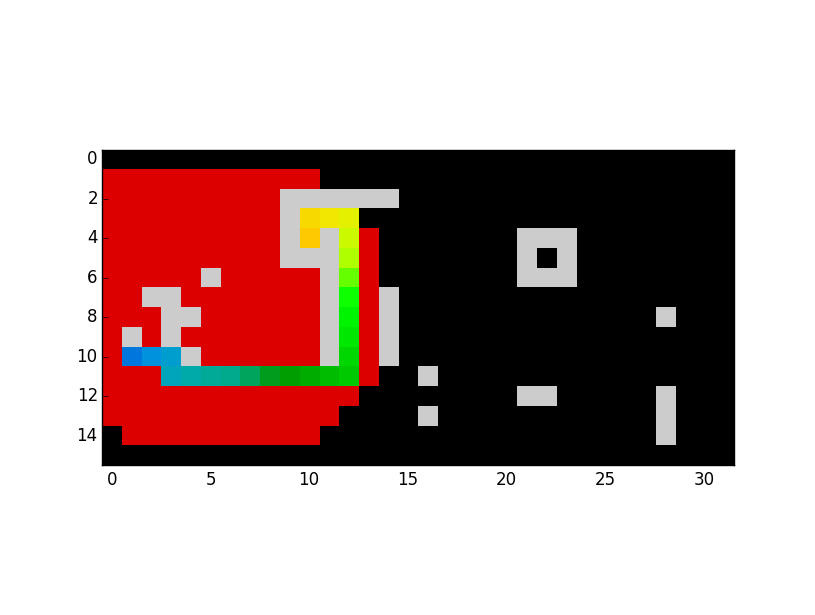
Although the path remains the same. The number of visited nodes is less when compared to uniform cost search. The states explored here a more based heuristic value depending on the goal(h\_value), hence it not much after the goal position but uniform cost search just compared the next step cost(only g\_value).

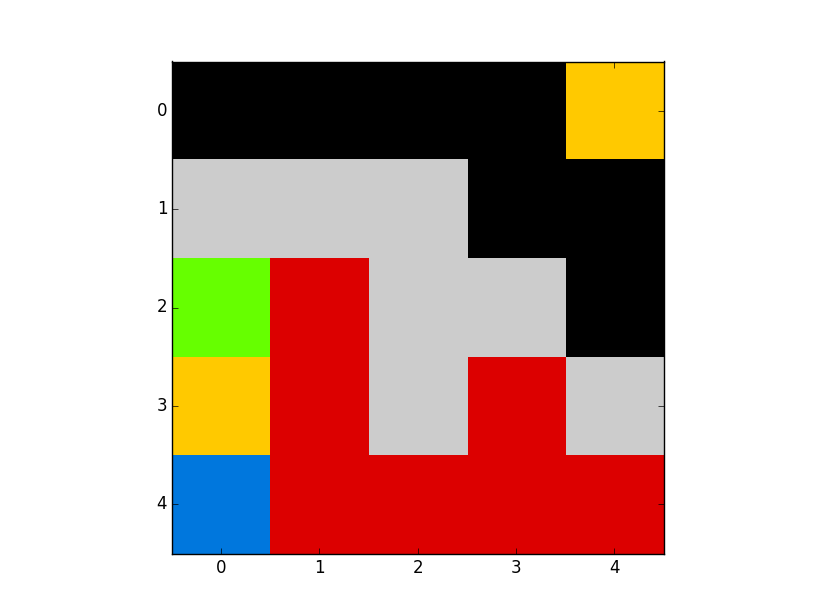
**Question – 4.2: A\* with Manhattan heuristics**

map0

Path: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (3, 4), (2, 4), (1, 4), (0, 4)]

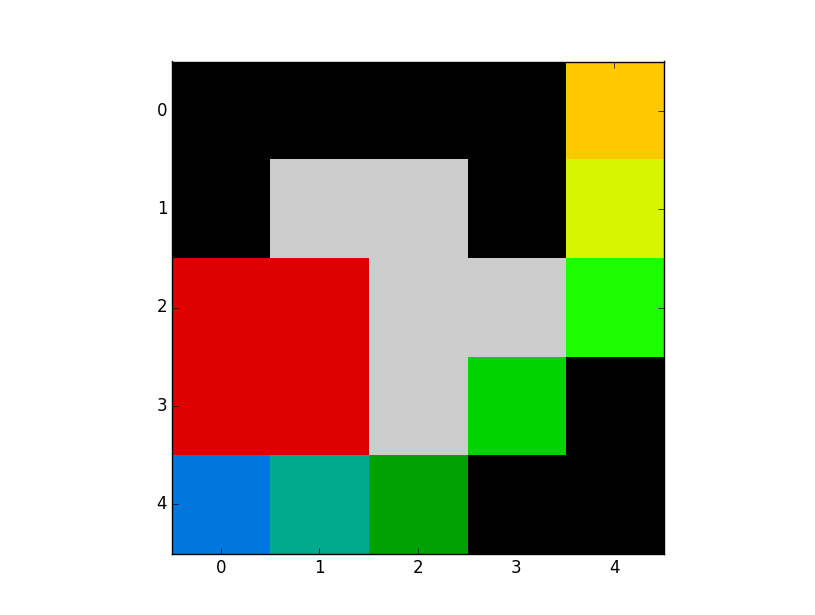
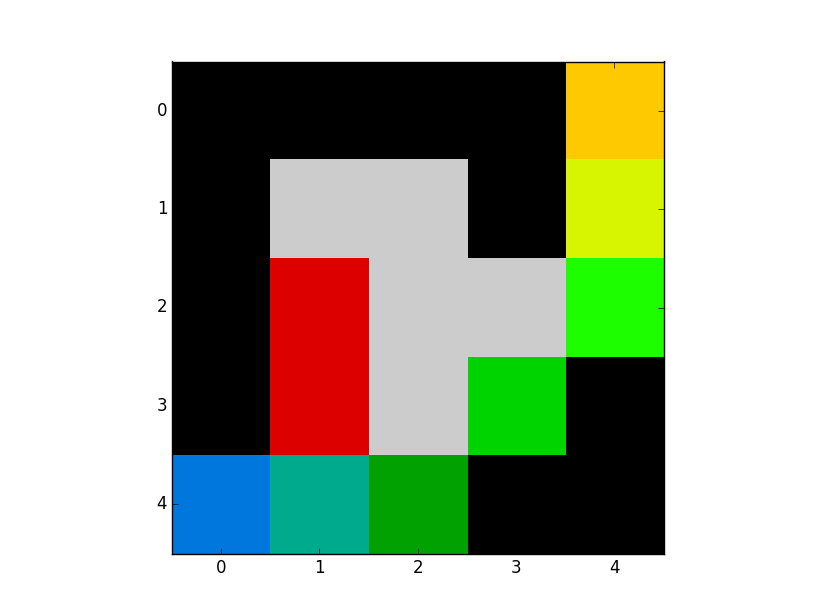
Visited nodes: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (3, 4), (2, 4), (1, 4), (0, 4)]

map1

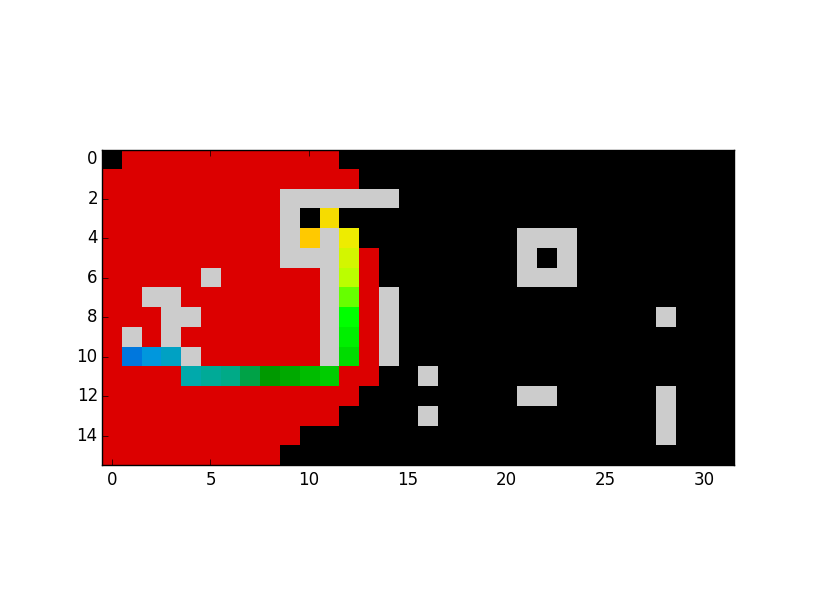
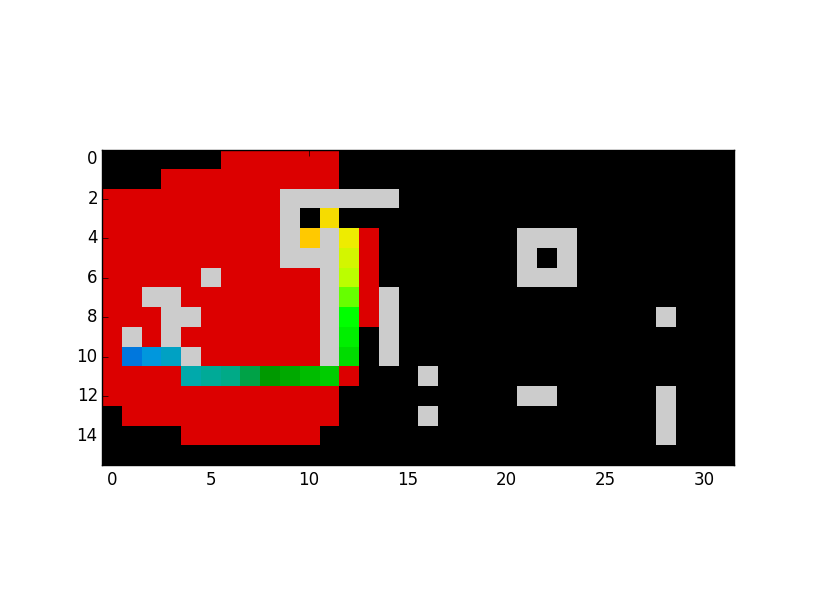
map2

Although the path remains the same. The number of visited nodes is less when compared to uniform cost search. The states explored here a more based heuristic value depending on the goal(h\_value), hence it not much after the goal position but uniform cost search just compared the next step cost(only g\_value). We can also observe that in Manhattan visited nodes are less(more black area).

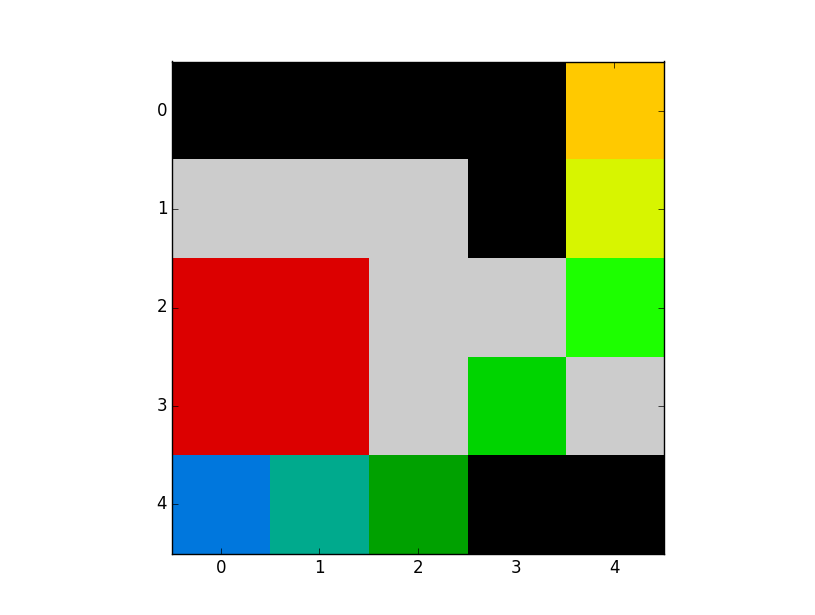
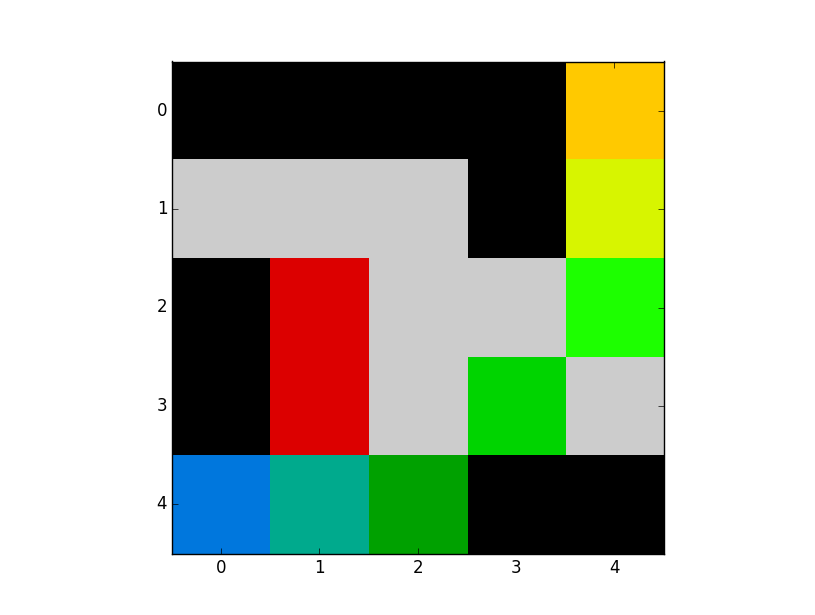
**Question – 4.2: A\* with crosspaths**

****

Map0: A\* Manhattan heuristic Map0: A\* Euclidian heuristics

****

Map1: A\* Manhattan heuristic Map1: A\* Euclidian heuristics

****

Map2: A\* Manhattan heuristics Map2: A\* Euclidian heuristics

Heuristics are admissible. Since value 1.5 is taken diagonally is much closer to the value of hypotenuse which is sqrt(2) = 1.414

The effect of adding new actions is whenever path is taking right angle turn now its taking a cross path since cost incurred in taking cross path is less than taking a right angle turn.

Yes heuristics effect the solution of the problem. In general heuristics are chosen such that h(B) – h(A) <= g(B)-g(A) A,B are in general 2positons on graph. The more close they the better the solution . visited nodes will be less. Here in this problem

Question 5:

Question-5.1

The hardest part of the question is back path and iteration deepening of DFS. Since it involves hidden understanding and clarity of code &things happening with the code.

Question-5.2

Once understand the structure rest of the things are easy.by end of the uniform search I am pretty comfortable with python. Hence I felt A\* a bit easier.

Question-5.3

Iterative deepeningof DFS and backpath function. It involves understanding of list and how objects are stored when objects are passed.

Queastoin-5.4

The BFS structure provided is very useful. Infact the structure is same all other graph searches .helped a lot.

Question-5.5

Although I know concept .Writing description is difficult.