HW1 IAI

Question 1

2)

3)

4)

5)

Yes, there are cycles in the search space. In any two tiles which are not ‘H’ tiles you could move back and forth from one to another.

6)

4, since from any state there can be at most 4 possible moves.

7)

Since there are cycles in the search space, a general agent might never reach the goal state.

8)

16, the agent could go directly down to the left dragon ball, then proceed to go right up right down to the second one, then right and down to goal. Then when they have all the dragon balls they could wish this hw is over.

9)

This is not true.

|  |  |  |
| --- | --- | --- |
| H | H | S |
| H | H | D |
| H | H | D |
| A | A | F |
| A | H | F |
| A | H | F |
| A | H | G |
| G | H | H |

The red route is shortest by Manhattan distance but costs more.

Question 2

2)

The condition on the graph is that the graph should be a tree. If the graph is a tree then for each node there will be only one route to him.

3)

Todo: GO BACK

4)

The function will multiply each node its cost. For example: (Upper graph will become lower graph)

A diagram of a flowchart

Description automatically generated

5)

The number of nodes that will be developed: , all the nodes beside the G node.

The number of nodes that will be created: , all the nodes.

Question 3

1)

DFS-G is complete, unlike normal DFS, DFS-G prevent cycles by not visiting nodes that were developed more than once.

DGS-G is not admissible, it does not take into account the weight of the tile only the length of the path to G node.

2)

For this question we are assuming that the order of the actions is [D, R, U, L].

In this scenario, the algorithm will always try to go down, as long as the move is legal (there isn’t a hole in the way and we aren’t at the end of the board), if it can’t, it will go right, if it can’t then up and then finally try to go left.

In case the agent reaches the bottom right corner of the board or is stuck where it can only do one up or left move, it will cause a cycle in which it will go back and forth (for example. right, left, right ,left....)

In most boards this will not produce a solution to the dragon ball problem, since in most cases this will not create a path to collect both balls and reach the goal afterwards.

3)

The number of nodes that will be developed: , all the first column nodes and all the last row nodes besides the G node.

The number of nodes that will be created: , all the first two columns and the last 2 rows.

4)

The number of nodes that will be developed: , all the first column nodes and all the last row nodes besides the G node.

The number of nodes that will be created: , all the first column nodes and all the last row nodes including the G node.

Question 3

1) (a)+(b)+(c)(There isn’t any problem with (b))A white background with black text

Description automatically generated

2) + 3) (a)

Let’s assume we have path finding problem without holes, costs and etc,

If the depth of the goal node is then it is preferred to use ReversedDFS.

On the other hand, if the depth of the goal node is 1 we’ll prefer ID-DFS.

(b)

Our proposition to make the algorithm better is to update variable in each iteration the same as binary search.

Question 6

1)

UCS and BFS algorithms will operate in the same way if all the edges have the same cost.

In this case cost to get from the source node to any other node is the same as the distance.

2)

In the lecture we learned that if the best solution has a finite cost and minimum arc cost is positive, then UCS is complete and admissible. In our problem this condition applied and so UCS is complete and admissible for our DragonBall problem.

3)

A diagram of a triangle

Description automatically generated

This is an example where Edi’s implementation won’t work. His algorithm will return the path

A->B->G cost=105 instead of A->C->G cost=7

A diagram of a diagram

Description automatically generated

This is an example where Edi’s algorithm will the same as UCS. A->B->G cost = 6

Question 8

1)

Because our “World” is connected and has finite number of states then the algorithm is complete.

In case of admissibility, it’s dependent on the heuristic, for the perfect heuristic the algorithm is admissible but there our some heuristic that GreedBestSearch return an non-optimal solution.

2)

GreedyBestSearch will more likely to return a solution that is closer to the optimal solution, on the other hand BeamSearch uses less memory and is faster.

**Question 9**

2) (a) (b)

Rather if h is admissible or not, this condition does not necessarily apply. For example, for heuristic:

Which means W-A\* will operate in the same way for w1 and for w2 therefore .

**Question 10**

**1)**

In cases where memory usage is not a factor, we will prefer to use A\* and not IDA\* because A\* will find a solution faster even though both algorithm works with the same complexity.

Because IDA\* has less memory usage, we will prefer to use it in cases where our branching factor is large or in cases where our “world” contains a lot of nodes.

2)

GOBACK HERE

**Question 11**

2)

An advantage of A\*-Epsilon is that it could potentially solve the problem faster in some cases, on the other hand A\* will always returns the optimal solution (if h is admissible) unlike A\*-Epsilon.

3)

Another method to choose a node from the local beside by g value is by random selection,

By using g value, the behavior is similar weighted A\* with a small weight, which reminds a little bit of UCS algorithm, which means that more likely to get an optimal solution.

By using random selection, we have a chance to get a faster solution with less expended nodes and it is more likely to not be an optimal solution.

4)

When epsilon is and the selection from focal is by minimum g value then the behavior we get is exactly like UCS, from all developed nodes we choose the node with the minimum g value.

**Question 12**

2) Cost:

We expected BFS to give the worst solutions, since it does not take into account the cost of edges. This was true. What was surprising to us was that WA\* with different weights gave the same cost solution, we expected the solution quality to deteriorate as the weight we gave the heuristic increased, since the heuristic is very different from h\*.

Expanded:

The results were that in our benchmarks WA\*(0.7) expanded the least amount of nodes, second best was WA\*(0.5) and then (WA\*(0.9) or BFS).

We expected that the higher the weight in WA\* then more nodes would be expanded since the heuristic is not very informative for example once we collected a dragon ball it still incentivized the agent to go in the direction of the collected dragon ball.

We expected BFS to develop more nodes than WA\*(0.5) because it operates in an arbitrary order while not taking into account any heuristic that “points” it to the right direction, this was true, also we didn’t know what to expect when increase the weight of the heuristic that’s because of the paragraph above.

When given a more informed heuristic then WA\*(0.9) and WA\*(0.7) will perform even better since in this case they both return the optimal solution it would only affect the number of expanded nodes.

We expect WA\*(0.7) and WA\*(0.9) to be best in this case, then WA\*(0.5) and then BFS, that is of curse depended on the heuristic.

**Question 13**

**1)**

**2)**

**The maximum steps count is 3, A->B->F->G (in case beta > 4)**

**3)**

**No, it would go to H state and will be stuck there. From C it would not move to B since it wouldn’t improve the utility. Which means it will return maximum of 3 even though there are higher utilities in the graph.**

**4)**

**The probability that the algo will not go to B is , either it would go to D and be stuck there or it would go to C and will return H. If the algo will go to B then:**

**If then , ins this case**

**If then , If we went to B then for sure we will get to global maximum.**