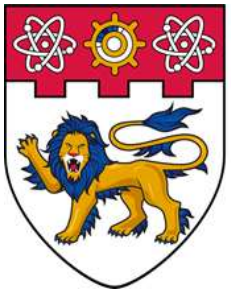


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Lab Exercise 1: Problem Solving

Name: Neo Rui Xuan Berlynn

Lab Group: TS3

Question 1

(a) Give a graph where depth-first search (DFS) is much more efficient (expands fewer nodes) than breadth-first search (BFS). [10 marks]

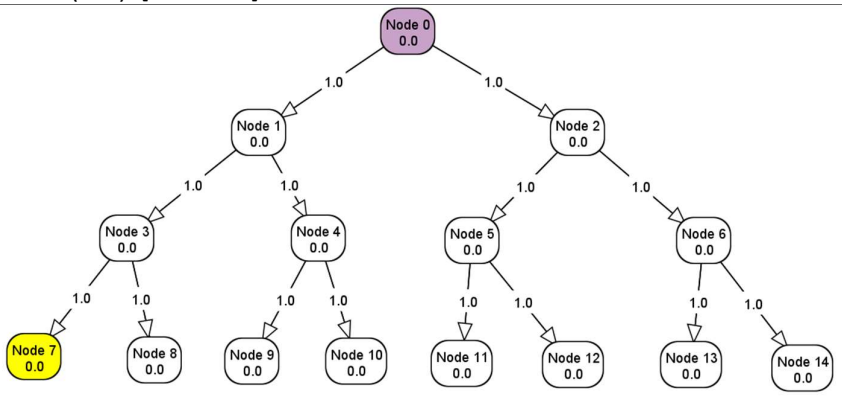
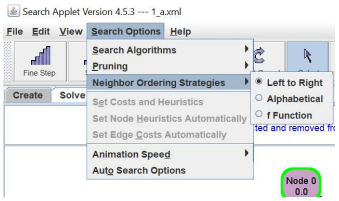


Figure 1: Graph 1

Start node: Node 0
Goal node: Node 7
Order of the neighbours:

- 

The edge costs are all 1.0 and all nodes have no heuristic functions.

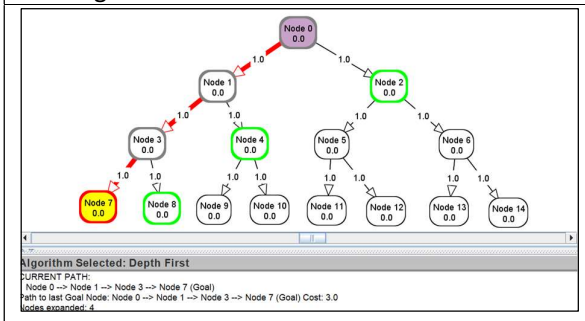


Figure 2: DFS on Graph 1

A DFS on this graph results in 4 expanded nodes to reach the goal node.

- 10 fine steps, 4 steps
- Node 0 → Node 1 → Node 3 → Node 7

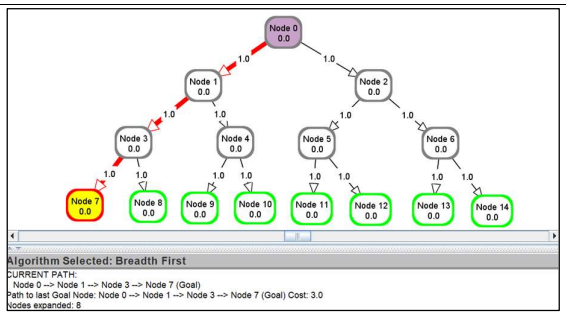


Figure 3: BFS on Graph 1

A BFS on this graph results in 8 expanded nodes to reach the goal node.

- 22 fine steps, 8 steps
- Node 0 → Node 1 → Node 2 → Node 3 → Node 4 → Node 5 → Node 6 → Node 7

Conclusion:
DFS is more efficient than BFS as fewer nodes (by 4 nodes) are expanded during DFS. Additionally, DFS had 120% lesser fine steps than BFS.
The goal node is on the left most and on the lowest level of the graph. The BFS searches level by level starting from the highest level before reaching the goal node at the bottom level, whereas the DFS searches through the left most path first, where the goal node happens to be.
Therefore, a DFS on the graph is more efficient than BFS when the goal node is at the lowest level and is at the left most of the graph.

(b) Give a graph where BFS is much better than DFS. [15 marks]

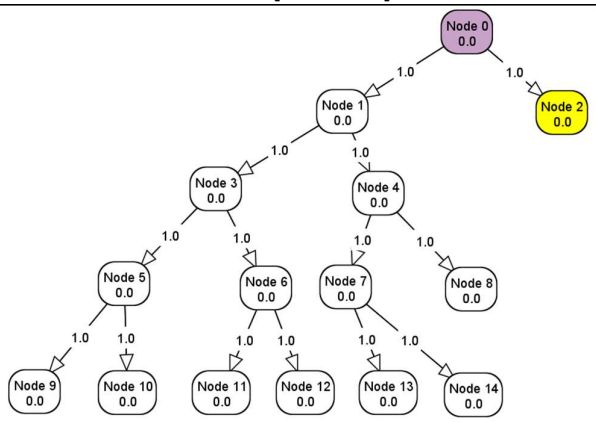
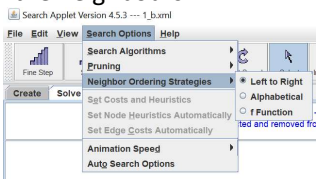


Figure 4: Graph 2

Start node: Node 0
Goal node: Node 2
Order of the neighbours:

- 

The edge costs are all 1.0 and all nodes have no heuristic functions.

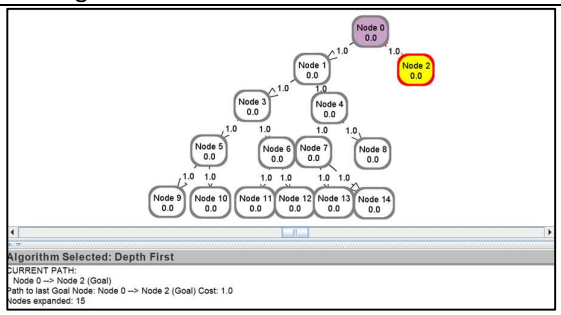


Figure 5: DFS on Graph 2

A DFS on this graph results in 15 expanded nodes to reach the goal node.

- 43 fine steps, 15 steps
- Node 0 → Node 1 → Node 3 → Node 5 → Node 9 → Node 10 → Node 6 → Node 11 → Node 12 → Node 4 → Node 7 → Node 13 → Node 14 → Node 8 → Node 2

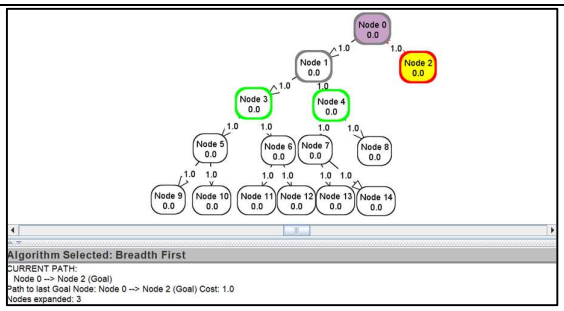


Figure 6: BFS on Graph 2

A BFS on this graph results in 3 expanded nodes to reach the goal node.

- 7 fine steps, 3 steps
- Node 0 → Node 1 → Node 2

Conclusion:
BFS is much better than DFS as fewer nodes (by 12 nodes) are expanded during BFS. Additionally, BFS had 514% lesser fine steps than DFS.
The goal node is at the right most of the graph and nearest to the root node. The DFS would search through all children nodes of Node 1 to the maximum depth and from left to right before arriving at the goal node. On the other hand, the BFS would search through level by level, starting from the top where the goal node happens to be.
Therefore, a BFS on the graph is much better than DFS when the goal node is located towards the right most side of the graph.

(c) Give a graph where A* search is more efficient than either DFS or BFS. [15 marks]

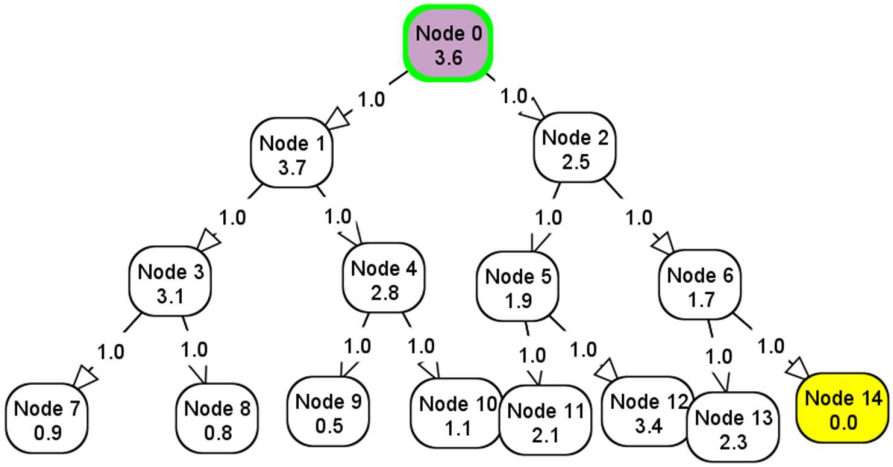
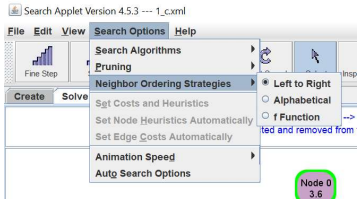


Figure 7: Graph 3

Start node: Node 0

Goal node: Node 14

Order of the neighbours:

- 

The edge costs are all 1.0 and all nodes have heuristic functions as shown in Figure 7.

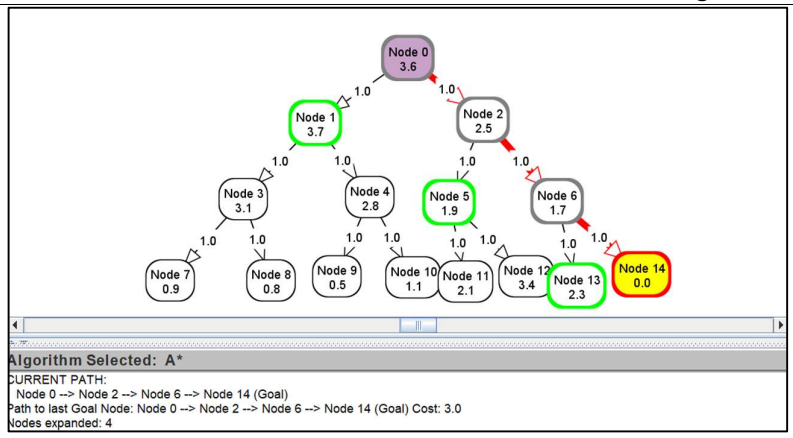
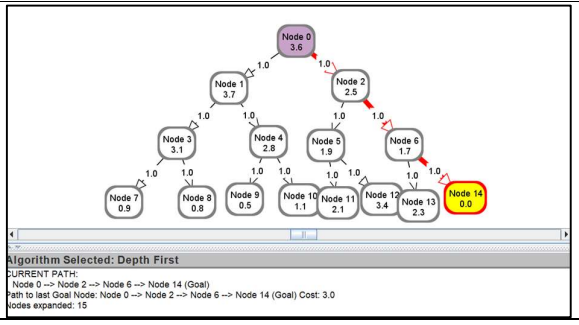


Figure 8: A* Search on Graph 3

An A* Search on this graph results in 4 expanded nodes to reach the goal node.

- 10 fine steps, 4 steps



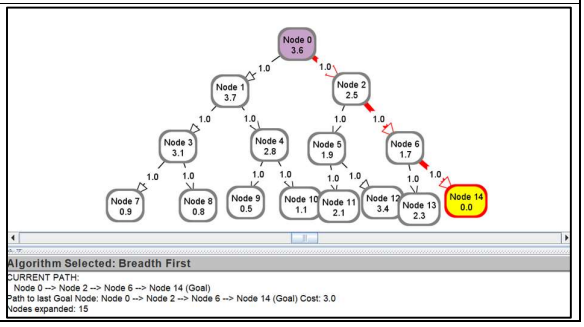


	Figure 9: DFS on Graph 3 A DFS on this graph results in 15 expanded nodes to reach the goal node. - 43 fine steps, 15 steps - Node 0 → Node 1 → Node 3 → Node 7 → Node 8 → Node 4 → Node 9 → Node 10 → Node 2 → Node 5 → Node 11 → Node 12 → Node 6 → Node 13 → Node 14	Figure 10: BFS on Graph 3 A BFS on this graph results in 15 expanded nodes to reach the goal node. - 43 fine steps, 15 steps - Node 0 → Node 1 → Node 2 → Node 3 → Node 4 → Node 5 → Node 6 → Node 7 → Node 8 → Node 9 → Node 10 → Node 11 → Node 12 → Node 13 → Node 14
	Conclusion: Evidently, the A* Search on this graph is more efficient DFS and BFS as the least number of nodes were expanded in the A* Search. Additionally, the A* Search had 330% lesser fine steps than DFS and BFS. The goal node is at the lowest level and the rightmost of the graph. As DFS searches from the leftmost path to the rightmost path and BFS searches level by level starting from the highest level, they would need to expand all nodes. Both DFS and BFS do not consider the heuristic cost, and in this case, the path cost does not affect either search as well. On the other hand, A* Search expands nodes by using the evaluation function of f(n)=g(n)+h(n), where g(n) is the cost from initial start current state (node n) and h(n) is the heuristic function, i.e. the estimated cost of the cheapest path from n to a goal state h(n). The node with the lowest f(n) is expanded. h(n) helps for the expansion of nodes that are heuristically closer to the goal node while ignoring other nodes. Therefore, an A* Search on the graph is more efficient than DFS and BFS.	

(d)	Give a graph where DFS and BFS are both more efficient than A* search. [15 marks]

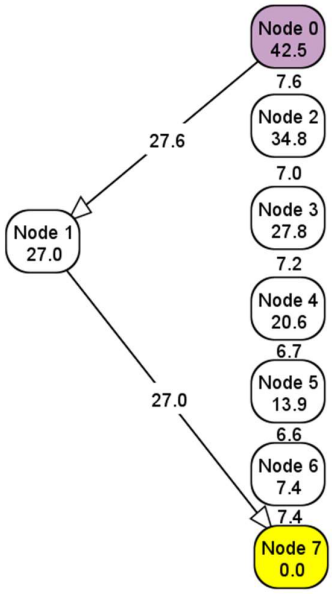
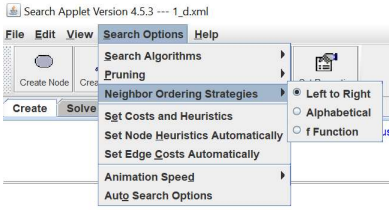


Figure 11: Graph 4

Start node: Node 0
Goal node: Node 7
Order of the neighbours:



-

The edge costs and heuristic functions are all shown in Figure 11.
There are two paths for the start node to reach the goal node:

1. Node 0 → Node 1 → Node 7
   Cost = 54.6
2. Node 0 → Node 2 → Node 3 → Node 4 → Node 5 → Node 6 → Node 7 (Optimal)
   Cost = 42.5

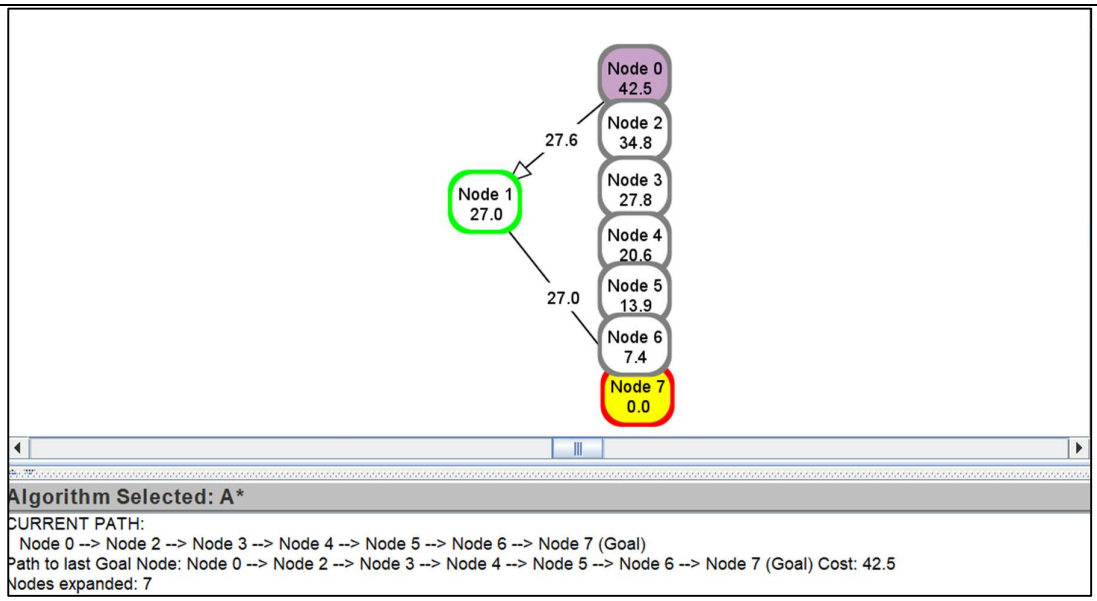


Figure 12: A* Search on Graph 4

An A* Search on this graph results in 7 expanded nodes to reach the goal node.
The path expanded is the optimal path.

- 19 fine steps, 7 steps

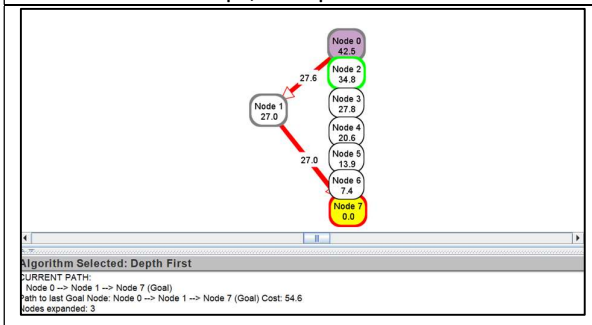


Figure 13: DFS on Graph 4

A DFS on this graph results in 3 expanded nodes to reach the goal node.

- 7 fine steps, 3 steps
- Node 0 → Node 1 → Node 7

The path expanded is not the optimal path.

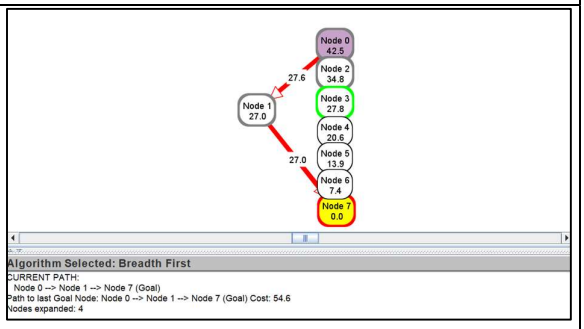


Figure 14: BFS on Graph 4

A BFS on this graph results in 4 expanded nodes to reach the goal node.

- 10 fine steps, 4 steps
- Node 0 → Node 1 → Node 2 → Node 7

The path expanded is not the optimal path.

Conclusion:

With this graph, DFS and BFS are both more efficient than A* Search. This is because A* Search considers the heuristics h(n) when taking the evaluation value of f(n)=g(n)+h(n) (as explained in 1b) and expands on the optimal path, which happens to have 7 nodes. On the other hand, BFS and DFS do not consider the heuristics. BFS searches through level by level starting from the top and DFS searches through from left to right.

Additionally, DFS and BFS had 171% and 90% lesser fine steps than A* Search.

Thus, as DFS and BFS expands lesser nodes as compared to A* Search, they are both more efficient than A* Search.

Question 2

(a) What is the effect of reducing h(n) when h(n) is already an underestimate? [15 marks]

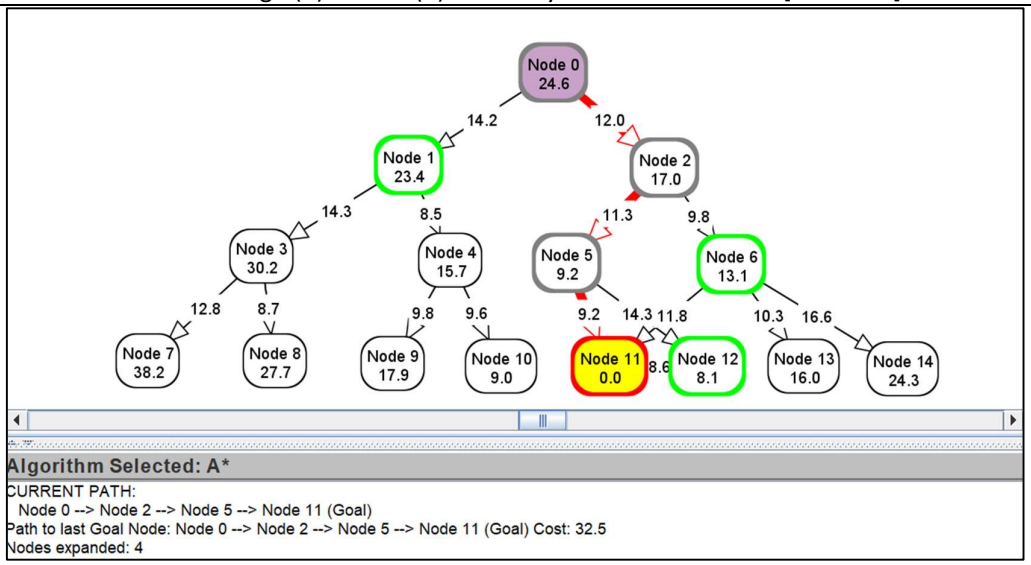


Figure 15: Graph 5 (original)

Start node: Node 0
Goal node: Node 11
The edge costs and heuristic functions are all shown in Figure 15.
4 nodes are expanded in the original graph.

- 10 fine steps, 4 steps

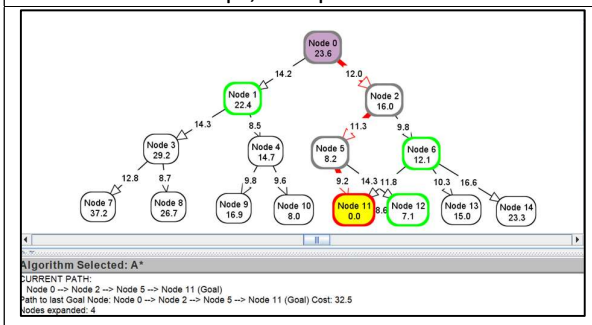


Figure 16: Graph 5 (each node has underestimated h(n) by 1)

Here, each node's heuristic function is underestimated by 1 and this results in 4 nodes expanded.

- 10 fine steps, 4 steps

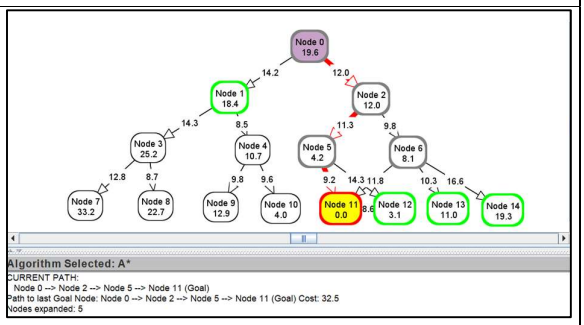


Figure 17: Graph 5 (each node has underestimated h(n) by 5)

Here, each node's heuristic function is underestimated by 5 and this results in 5 nodes expanded.

- 13 fine steps, 5 steps

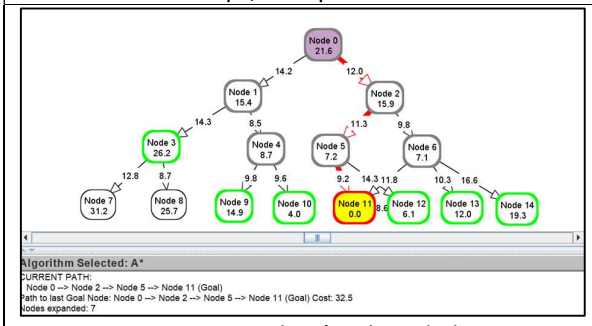


Figure 18: Graph 5 (each node has underestimated h(n) by different amounts)

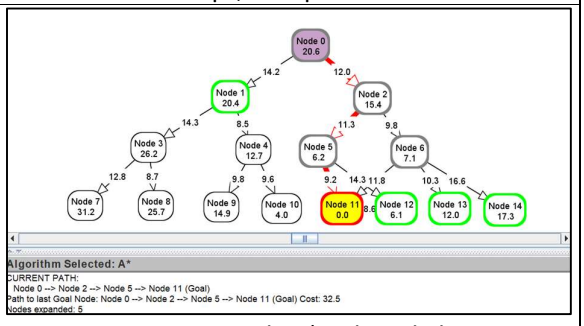


Figure 19: Graph 5 (each node has underestimated h(n) by different amounts)

	Here, each node's heuristic function is underestimated by different amounts and this results in 7 nodes expanded. - 19 fine steps, 7 steps	Here, each node's heuristic function is underestimated by different amounts, that are different from those in Figure 18, and this results in 5 nodes expanded. - 13 fine steps, 5 steps

Conclusion:
There are three paths for the start node to reach the goal node:

1. Node 0 → Node 2 → Node 5 → Node 11 (Optimal)
   Cost = 32.5
2. Node 0 → Node 2 → Node 5 → Node 12 → Node 11
   Cost = 46.2
3. Node 0 → Node 2 → Node 6 → Node 11
   Cost = 33.6

In all cases, the optimal path Node 0 → Node 2 → Node 5 → Node 11 is obtained.

When h(n) is underestimated by the same amount for each node, for example a small number like 1 as seen in Figure 16, the same number nodes as the original graph are expanded. When h(n) is further underestimated by 4 as seen in Figure 17, even more nodes are expanded. As such, an underestimation of h(n) when h(n) is already an underestimate results in the A* Search becoming less efficient only when the underestimation is a large enough constant, as more nodes are expanded.

When h(n) is further underestimated by a different amount for each node as seen in Figure 18, a greater number of nodes are expanded (7 nodes) as compared to the above scenario in Figure 17. This result depends on which node's h(n) is decreased and by what amount, as seen in Figure 19, where only 5 nodes are expanded.

The further underestimation of h(n) also led to the increase of fine steps as seen in the elaboration after each figure.

This shows that when h(n) is further underestimated by a different amount for each node, A* Search sometimes becomes less efficient. However, the least-cost path will always be found as long as h(n) is not an overestimate of the cost from n to the goal.

(b) How does A* perform when h(n) is the exact distance from n to a goal? [15 marks]

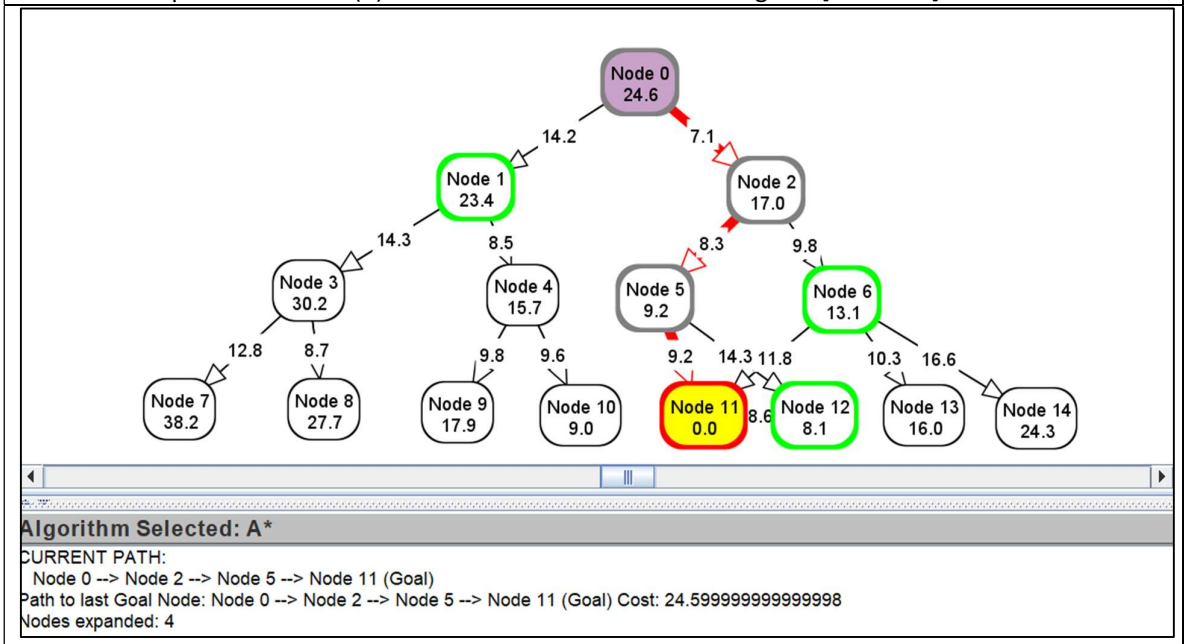


Figure 20: Graph 6

Start node: Node 0
Goal node: Node 11
The edge costs and heuristic functions are all shown in Figure 20.
4 nodes are expanded here.

- 10 fine steps, 4 steps

Conclusion:
There are three paths for the start node to reach the goal node:

1. Node 0 → Node 2 → Node 5 → Node 11 (Optimal)
   Cost = 24.6
2. Node 0 → Node 2 → Node 5 → Node 12 → Node 11
   Cost = 38.3
3. Node 0 → Node 2 → Node 6 → Node 11
   Cost = 28.7

Here, the optimal path 1 is returned and 4 nodes are expanded.
When h(n) is the exact distance from n to a goal, i.e. h(n) is the actual cost from n to the goal, the evaluation function f(n) = g(n) + h(n) = 2*g(n). In A* Search, the node with the lowest f(n) is expanded. As such, the A* Search always performs the best as it expands only the nodes in the optimal path, without expanding any other nodes outside of the optimal path.

(c) What happens if h(n) is not an underestimate? You can give an example to justify your answer. [15 marks]

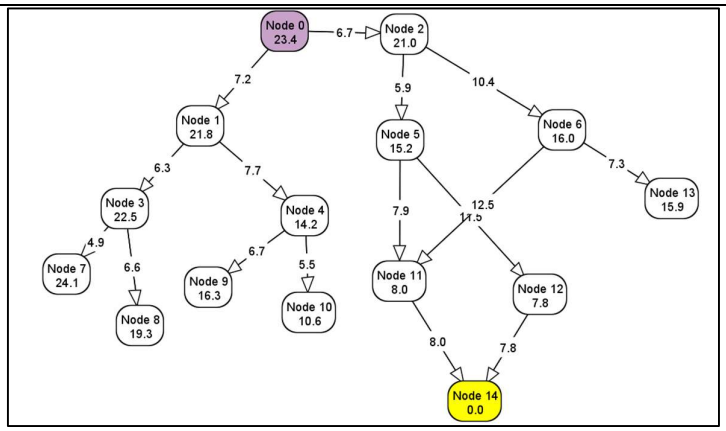


Figure 21: Graph 7 (original)

Start node: Node 0
Goal node: Node 14
The edge costs and heuristic functions are all shown in Figure 21.
There are three paths for the start node to reach the goal node:

1. Node 0 → Node 2 → Node 5 → → Node 11 → Node 14 (Optimal)
   Cost = 28.5
2. Node 0 → Node 2 → Node 5 → Node 12 → Node 14
   Cost = 31.9
3. Node 0 → Node 2 → Node 6 → Node 11 → Node 14
   Cost = 37.6

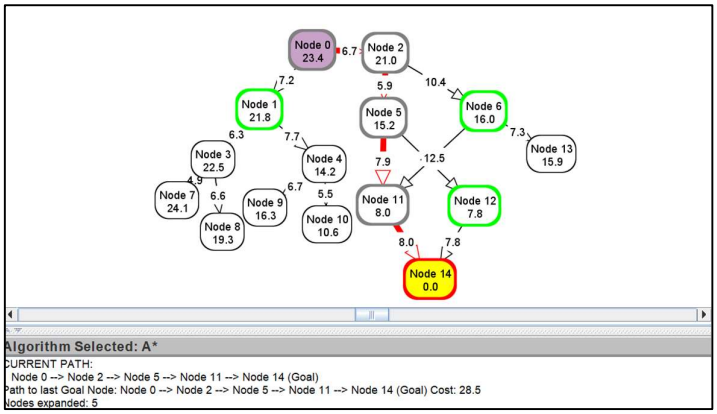


Figure 22: Graph 7 (original)

5 nodes are expanded in the original graph.

- 13 fine steps, 5 steps

Node n	Path cost g(n)	h(n)	f(n) = g(n) + h(n)
5	12.6	15.2	27.8
6	17.1	16.0	33.1
11	20.5	8.0	128.5
12	24.1	7.8	131.9

The optimal path 1 is taken as f(5)<f(6) and f(11)<f(12).

Case 1: h(n) is the exact distance from n to a goal.

- Results are as seen in 2(b).

Case 2: h(n) is an overestimate.

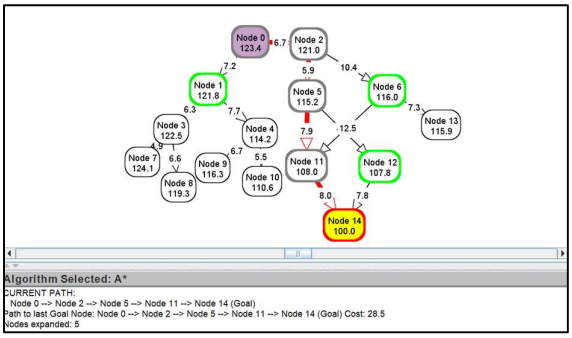


Figure 23: Graph 7 (overestimate by 100)

Here, each node's heuristic function is overestimated by 100 and this results in 5 nodes expanded.

- 13 fine steps, 5 steps

Node n	Path cost g(n)	h(n)	f(n) = g(n)+h(n)
5	12.6	115.2	127.8
6	17.1	116.0	133.1
11	20.5	108.0	128.5
12	24.1	107.8	131.9

The optimal path 1 is taken. This is because f(5)<f(6) and f(11)<f(12).

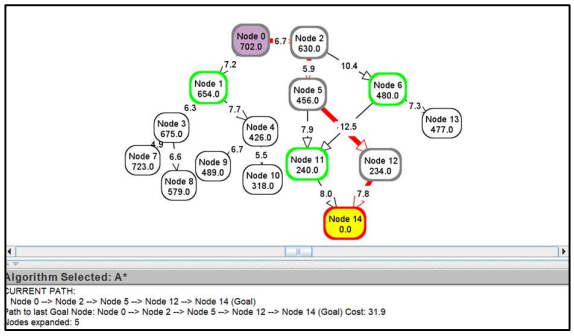


Figure 24: Graph 7 (overestimate by *30)

Here, each node's heuristic function is multiplied by 30 and this results in 5 nodes expanded. A non-optimal path, path 2, is taken.

- 13 fine steps, 5 steps

Node n	Path cost g(n)	h(n)	f(n) = g(n)+h(n)
5	12.6	456.0	468.6
6	17.1	480.0	497.1
11	20.5	240.0	260.5
12	24.1	234.0	258.1

The non-optimal path 2 is taken. This is because although f(5)<f(6), it is now f(11)>f(12) due to the overestimation of h(n).

Conclusion:

When h(n) is not an underestimate and is the exact distance of a goal, the conclusion is seen in 2b where it is observed that A* Search always performs most efficiently.

When h(n) is an overestimate by a small constant, there is no change to the A* Search as the same number of nodes are expanded and the optimal path is taken.

However, when h(n) is an overestimate by a large enough constant, the A* Search becomes less efficient although the same number of nodes are expanded, as a less optimal path with a higher cost is taken.

It is known that A* Search considers h(n). In this question, the value of h(n) and thus f(n) are changed, such that when the A* Search is deciding between some nodes for the search path, the f(n) values of some nodes which were originally less than the rest are now greater, and vice versa. Thus, the non-optimal path would sometimes be chosen.

Therefore, when h(n) is overestimated by a large enough number, the A* Search is not guaranteed to find the optimal path.