Nanyang Technological University (NTU)

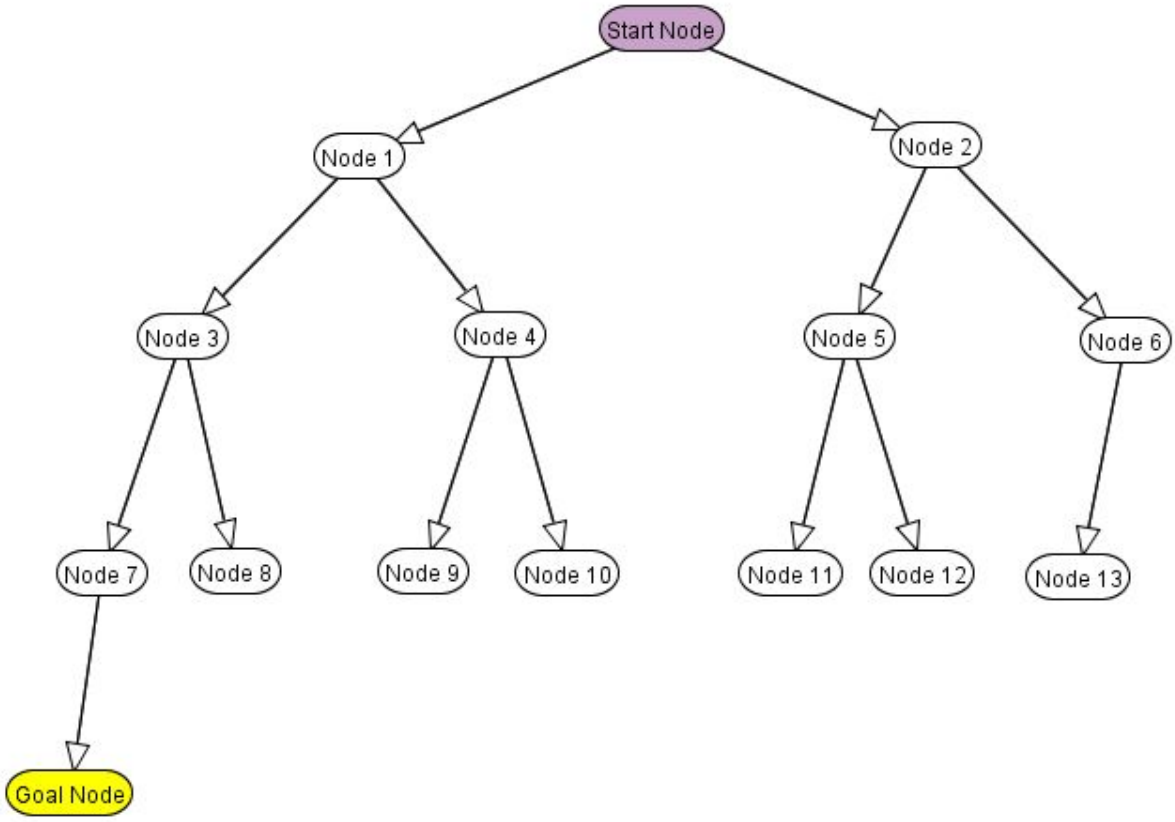
CZ3005 Artificial Intelligence

Lab Assignment 1

Adrian Goh Jun Wei

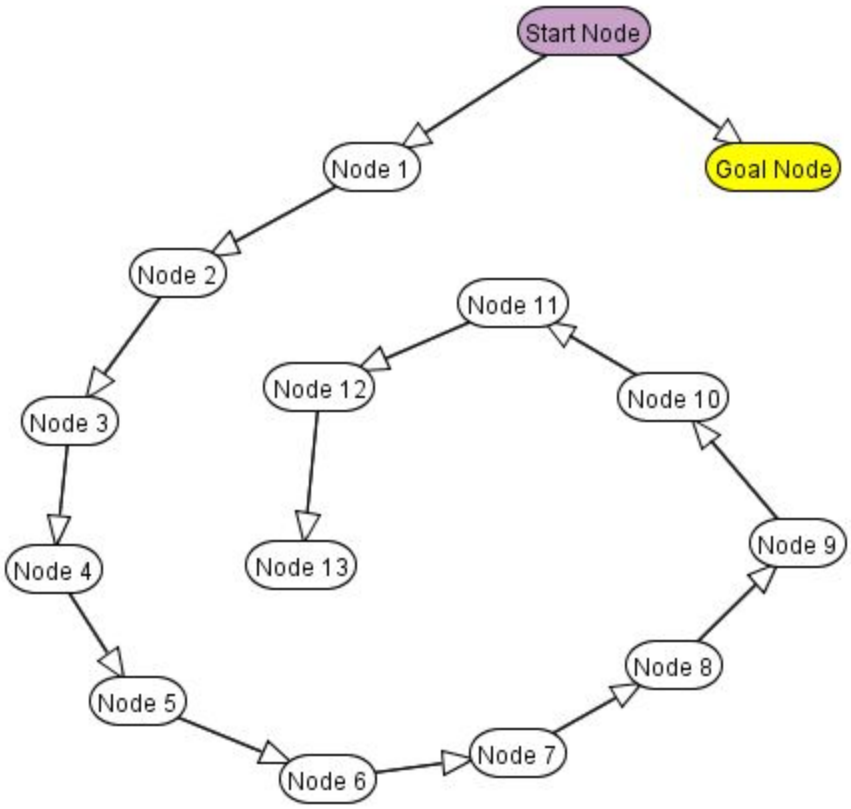
U1721134D

Question 1a



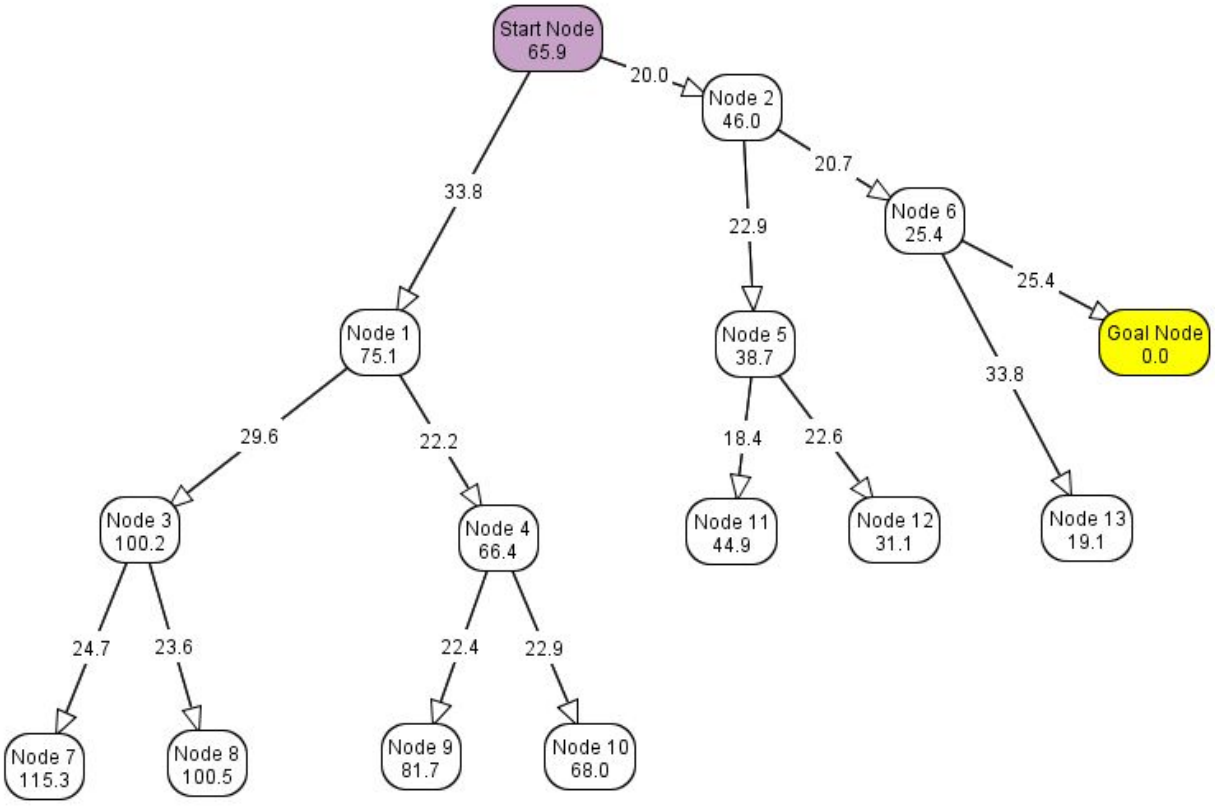
Search technique	**# of nodes**	**Search order**				
Depth-first search	5	**(1)** Start	**(2)** Node 1	**(3)** Node 3	**(4)** Node 7	**(5)** Goal
Breadth-first search	15	**(1)** Start	**(2)** Node 1	**(3)** Node 2	**(4)** Node 3	**(5)** Node 4
		(6) Node 5	**(7)** Node 6	**(8)** Node 7	**(9)** Node 8	**(10)** Node 9
		(11) Node 10	**(12)** Node 11	**(13)** Node 12	**(14)** Node 13	**(15)** Goal

Question 1b



Search technique	# of nodes	Search order				
Depth-first search	15	**(1)** Start	**(2)** Node 1	**(3)** Node 2	**(4)** Node 3	**(5)** Node 4
		(6) Node 5	**(7)** Node 6	**(8)** Node 7	**(9)** Node 8	**(10)** Node 9
		(11) Node 10	**(12)** Node 11	**(13)** Node 12	**(14)** Node 13	**(15)** Goal
Breadth-first search	3	**(1)** Start	**(2)** Node 1	**(3)** Goal		

Question 1c

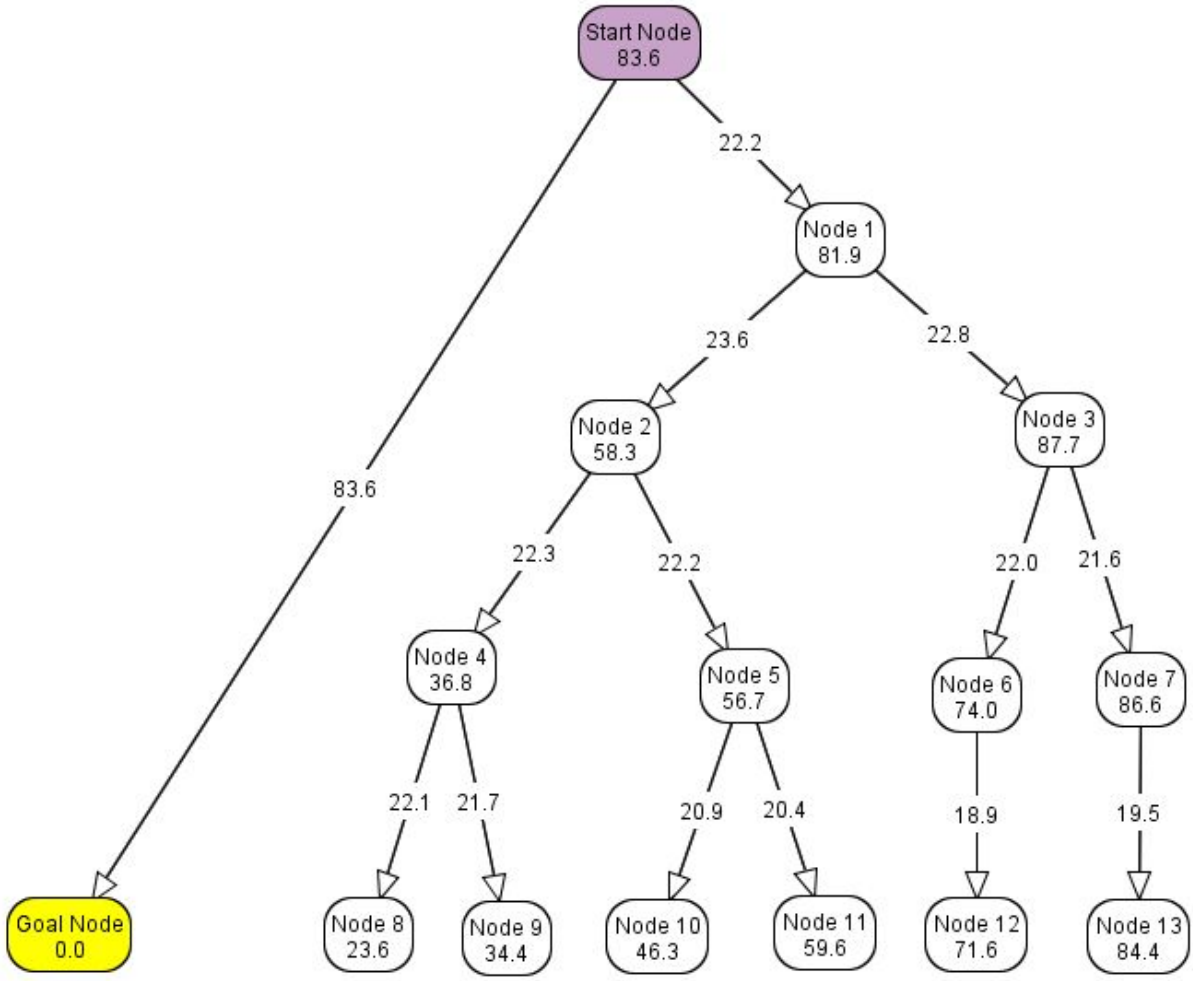


Arc costs are labeled on the arcs itself, while heuristic functions of nodes are within the node. Both the arc costs and heuristic functions are auto-generated using Alspace software, which is based on relative distance on the drawing.

Search technique	**# of nodes**	**Search order**				
Depth-first search	15	**(1)** Start	**(2)** Node 1	**(3)** Node 3	**(4)** Node 7	**(5)** Node 8
		(6) Node 4	**(7)** Node 9	**(8)** Node 10	**(9)** Node 2	**(10)** Node 5
		(11) Node 11	**(12)** Node 12	**(13)** Node 6	**(14)** Node 13	**(15)** Goal

Breadth-first search	15	**(1)** Start	**(2)** Node 1	**(3)** Node 2	**(4)** Node 3	**(5)** Node 4
		(6) Node 5	**(7)** Node 6	**(8)** Node 7	**(9)** Node 8	**(10)** Node 9
		(11) Node 10	**(12)** Node 11	**(13)** Node 12	**(14)** Node 13	**(15)** Goal
A* search	4	**(1)** Start	**(2)** Node 2	**(3)** Node 6	**(4)** Goal	

Question 1d



Arc costs are labeled on the arcs itself, while heuristic functions of nodes are within the node. Both the arc costs and heuristic functions are auto-generated using Alspace software, which is based on relative distance on the drawing.

Search technique	**# of nodes**	**Search order**				
Depth-first search	2	**(1)** Start	**(2)** Goal			

Breadth-first search	2	**(1)** Start	**(2)** Goal			
A* search	15	**(1)** Start	**(2)** Node 1	**(3)** Node 2	**(4)** Node 4	**(5)** Node 8
		(6) Node 9	**(7)** Node 5	**(8)** Node 10	**(9)** Node 11	**(10)** Node 3
		(11) Node 6	**(12)** Node 12	**(13)** Node 7	**(14)** Node 13	**(15)** Goal

Question 2a

The evaluation function f(n) of A* search can be represented as:

$$f(n) = g(n) + h(n)$$

We know that A* search is:

- Optimal as it uses an admissible heuristic to find the most optimal path
- Complete, as long as there is not an infinite number of nodes to be explored

As such, as long as h(n) is lower (underestimation) than the actual cost from n to the goal, we know that A* search will guaranteed to always be optimal and complete. This is made possible because it maintains a priority queue of node options that it will consider, ranked by how ideal they appear to be. It will continue it's search until it finds a route to the goal node that is much more optimal than the other paths.

However, the larger the underestimation of the heuristic, the more A* search will behave like uniform-cost search, and might be very inefficient as it might search "aimlessly" further and further away from goal node. This is because the other options, which are non-optimal, will appear to be more ideal than they really are. As such, A* search will be "deceived" into checking these routes out, and thus, expanding more nodes than needed.

We will look at 3 examples to show the effect of heuristic underestimation.

- No heuristic underestimation
- Heuristic underestimation of 30%
- Heuristic underestimation of 100%

No heuristic underestimation

In Diagram 1.1, arc costs are labeled on the arcs itself, while heuristic functions of nodes are within the node. Both the arc costs and heuristic functions are auto-generated using AIspace software, which is based on relative distance on the drawing.

We can see that A* Search will reach the goal node by expanding only 2 nodes (Start Node → Goal Node). Also, the path taken from the start node to goal node is also optimal.

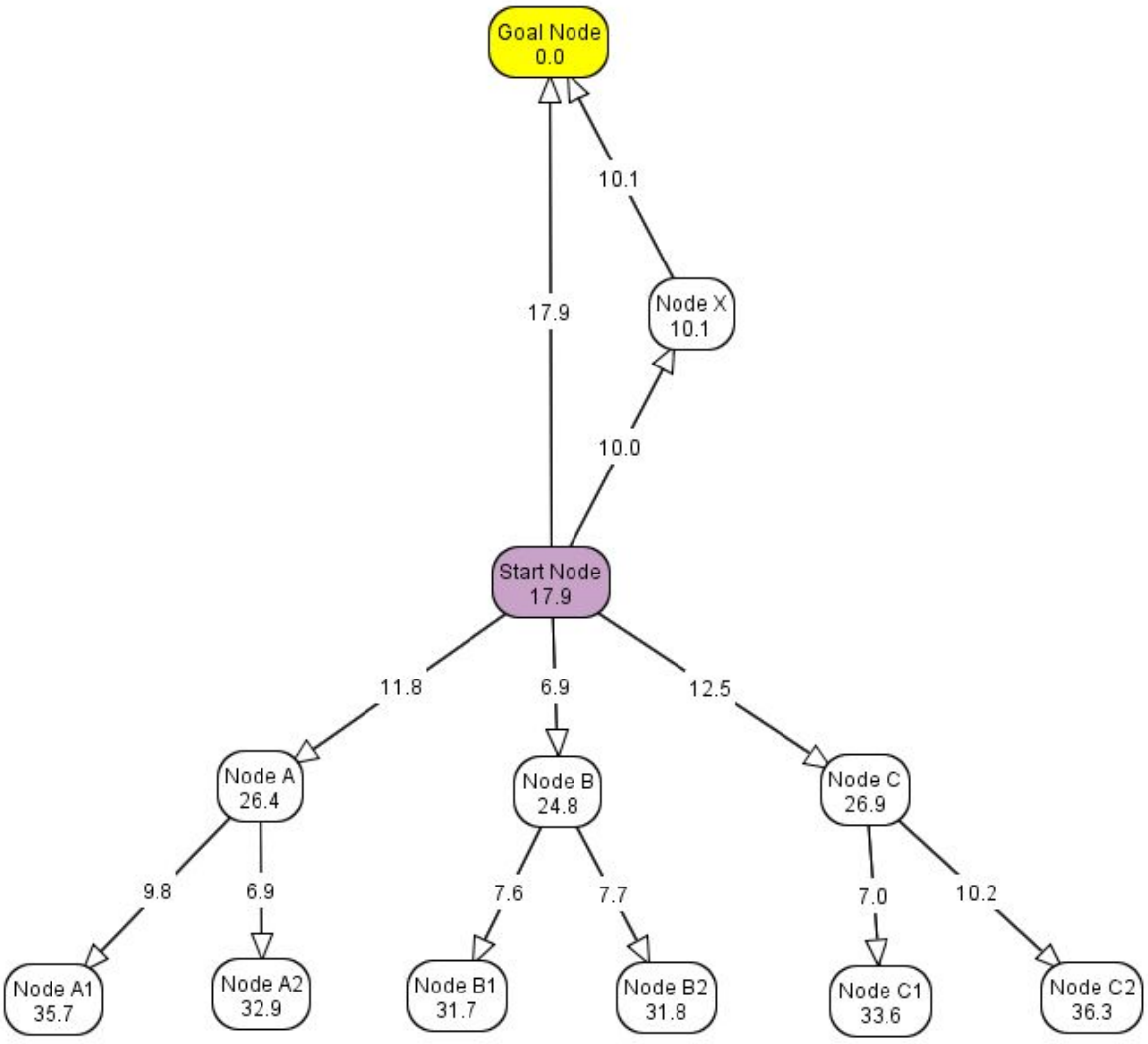


Diagram 1.1: A Search with exact h(n) that has been auto generated with AIsearch*

Heuristic underestimation of 30%

From the same graph in Diagram 1.1, we can derive the graph in Diagram 1.2 if we assume that there's an underestimation of h(n) by 30%.

We can see that A* Search is slightly more inefficient as it will reach the goal node by expanding 3 nodes, which is 1 more (Node X) than what happened for Diagram 1.1. Even though Node X is being explored, it is then discovered that the optimal path is still directly from the start node to the goal node, and thus, A* search is still accurate and optimal, but slightly less efficient.

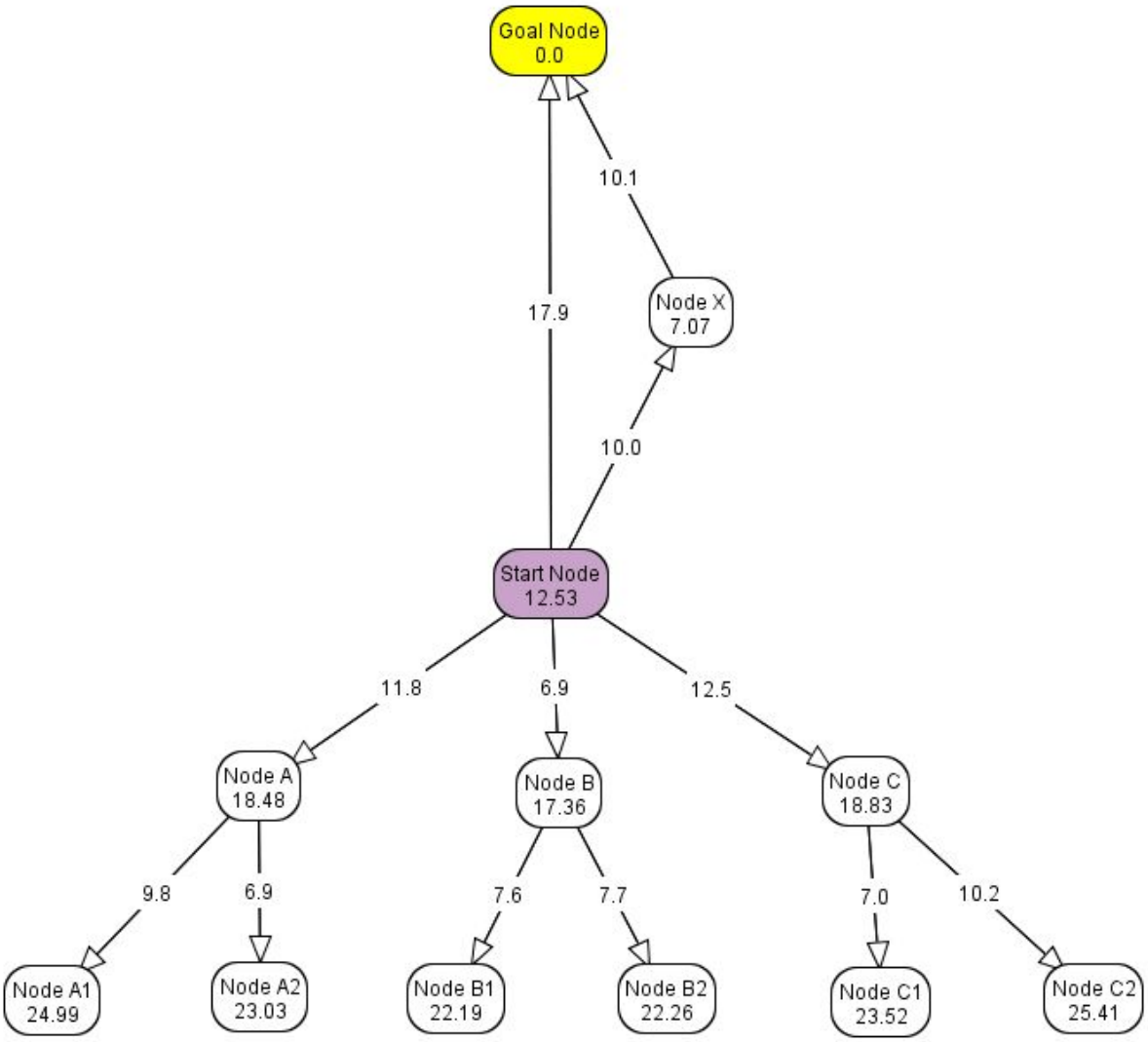


Diagram 1.2: A Search with underestimation of h(n) by 30%*

Heuristic underestimation of 100%

In the extreme case where the underestimation is so large, h(n) will have a value so small that it is so insignificant. In this scenario, we observe that $f(n) = g(n)$ when $h(n) \rightarrow 0$ and that A* search will behave like Dijkstra's algorithm, which is still guaranteed to find the shortest path (optimal), but becomes very inefficient as it expands a lot of nodes that don't lead to the optimal path.

Similarly, from the same graph in Diagram 1.1, we can derive the graph in Diagram 1.3 if we assume that there's an underestimation of h(n) by 100%. We can see that A* Search is very inefficient as it will reach the goal node by expanding 8 nodes, which is a lot more than what happened for Diagram 1.1. However, the optimal path is still directly from the start node to the goal node, and thus, A* search is still accurate and optimal.

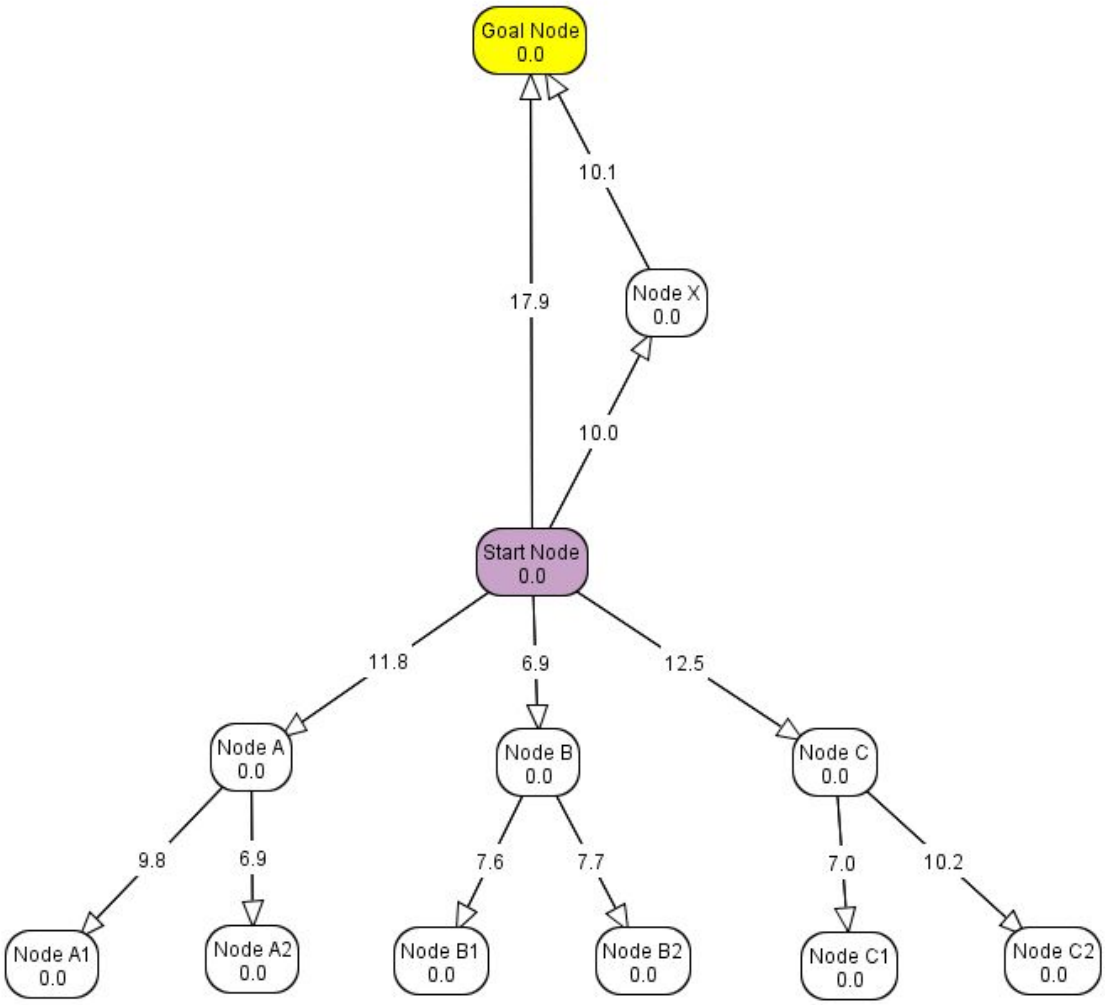


Diagram 1.3: A Search with extreme underestimation of h(n) by 100%*

Conclusion

As such, as long as h(n) is an underestimation of the actual cost from n to the goal, we can conclude that A* search will

- Always return a complete and optimal solution
- Be increasingly more inefficient as the degree of underestimation increases

Question 2b

If h(n) is equal to the exact distance from n to a goal, then A* will only follow the most optimal path and never expand anything else, making it very fast and efficient and perfect.

As mentioned in Question 2a, we know that A* search will guaranteed to always be optimal and complete because of its priority queue and search for the best nodes. We will look at this through 4 potential routes from the start node to the end note in Diagram 2.1.

In Diagram 2.1, arc costs are labeled on the arcs itself, while heuristic functions of nodes are within the node. Both the arc costs and heuristic functions are auto-generated using AIspace software, which is based on relative distance on the drawing.

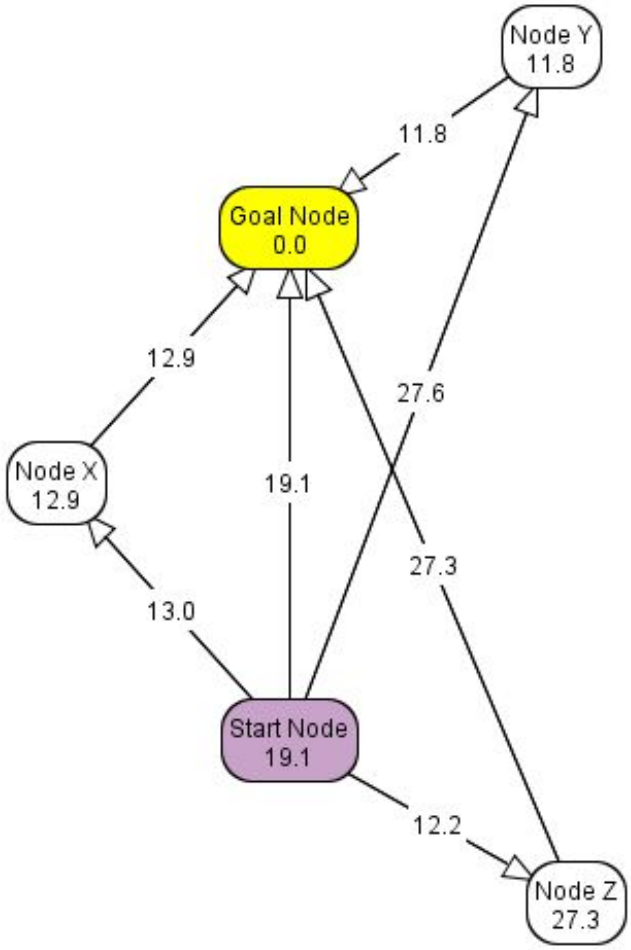


Diagram 2.1: A Search with exact h(n) that has been auto generated with AIsearch*

The optimal path for Diagram 2.1 is the direct path from the start node to the end node. Given that the A* search evaluation function is $f(n) = g(n) + h(n)$, the evaluation function for the optimal path is as such $f(n) = 19.1 + 0 = 19.1$. Let's explore the other 3 potential routes to show how exact heuristics will always give the optimal and accurate path.

Start Node → Node X → Goal Node

The total cost of travelling this path is equals to $13.0 + 12.9 = 25.9$, which is larger than 19.1 and thus not optimal. Let us proceed to prove why A* search will not choose this path.

We start off by computing the evaluation function of the path from the start node to Node X.

$$f(n) \; of \; Start \; Node \; to \; Node \; X = 13.0 + 12.9 = 25.9 \; .$$

As the evaluation function f(n) of the start node to Node X is larger than that of the direct path, the direct path is chosen instead, and there is actually no need to further calculate the f(n) from Node X to the goal node.

Start Node → Node Y → Goal Node

The total cost of travelling this path is equals to $27.6 + 11.8 = 39.4$, which is larger than 19.1 and thus not optimal. Let us proceed to prove why A* search will not choose this path.

We start off by computing the evaluation function of the path from the start node to Node Y.

$$f(n) \; of \; Start \; Node \; to \; Node \; Y = 27.6 + 11.8 = 39.4 \; .$$

As the evaluation function f(n) of the start node to Node Y is larger than that of the direct path, the direct path is chosen instead, and there is actually no need to further calculate the f(n) from Node Y to the goal node.

Start Node → Node Z → Goal Node

The total cost of travelling this path is equals to $12.2 + 27.3 = 39.5$, which is larger than 19.1 and thus not optimal. Let us proceed to prove why A* search will not choose this path.

We start off by computing the evaluation function of the path from the start node to Node Z.

$$f(n) \; of \; Start \; Node \; to \; Node \; Z = 12.2 + 27.3 = 39.5 \; .$$

As the evaluation function f(n) of the start node to Node Z is larger than that of the direct path, the direct path is chosen instead, and there is actually no need to further calculate the f(n) from Node Z to the goal node.

Conclusion

Assuming other scenarios where there are many more nodes between the start node to the goal node along a path that is similar to that of Node X, Y or Z, where the nodes are all vertically closer to the goal node on the y-axis. It is impossible to find a node such that the evaluation function f(n) of the node is smaller than that of a direct path from the start node to the end node. Therefore, it is always true that that with exact heuristics, A* search will have high

As such, as long as we have perfect information and are able to achieve exact heuristics at all times, such that h(n) will always be equal to the actual cost from n to the goal, we can conclude that A* search will:

- Always return a complete and optimal solution
- More efficient than if there's heuristics underestimation

Question 2c

As mentioned in question 2a, the evaluation function f(n) of A* search can be represented as:

$$f(n) = g(n) + h(n)$$

As such, the larger the overestimation of h(n), the more A* search will behave like greedy search. This means that it might find the shortest path sometimes but not all the time, thus we can deem it as non-optimal. However, it can be very fast and search space considerably.

We will look at 2 examples to show the effect of heuristic overestimation.

- No heuristic underestimation
- Heuristic underestimation of 20%

No heuristic underestimation

In Diagram 3.1, arc costs are labeled on the arcs itself, while heuristic functions of nodes are within the node. Both the arc costs and heuristic functions are auto-generated using AIspace software, which is based on relative distance on the drawing.

The optimal path for Diagram 3.1 by adding up arc costs is Path B (through B1, B2, B3, B4).

When we run the A* search, we discovered the following:

- Path to be taken is Path B, which is optimal
- 7 nodes expanded, as follow (in order of expansion)
    - Start → A1 → B1 → B2 → B3 → B4 → End

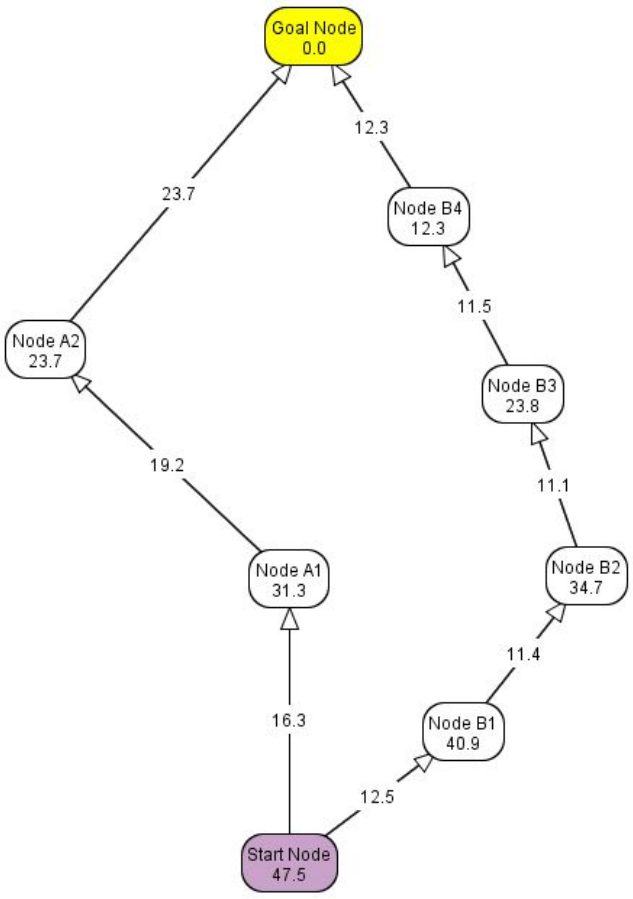


Diagram 3.1: A Search with exact h(n) that has been auto generated with AIsearch*

Heuristic overestimation of 20%

From the same graph in Diagram 3.1, we can derive the graph in Diagram 3.2 if we assume that there's an overestimation of h(n) by 20%.

When we run the A* search, we discovered the following:

- Path to be taken is not the optimal Path B, but Path A (through A1, A2)
- 5 nodes expanded, as follow (in order of expansion)
    - Start → A1 → B1 → A2 → End

As we can see, when compared to exact heuristics, an overestimation can sometimes result in a non-optimal path as the solution, but at the same time, being faster and more efficient as it expands less nodes.

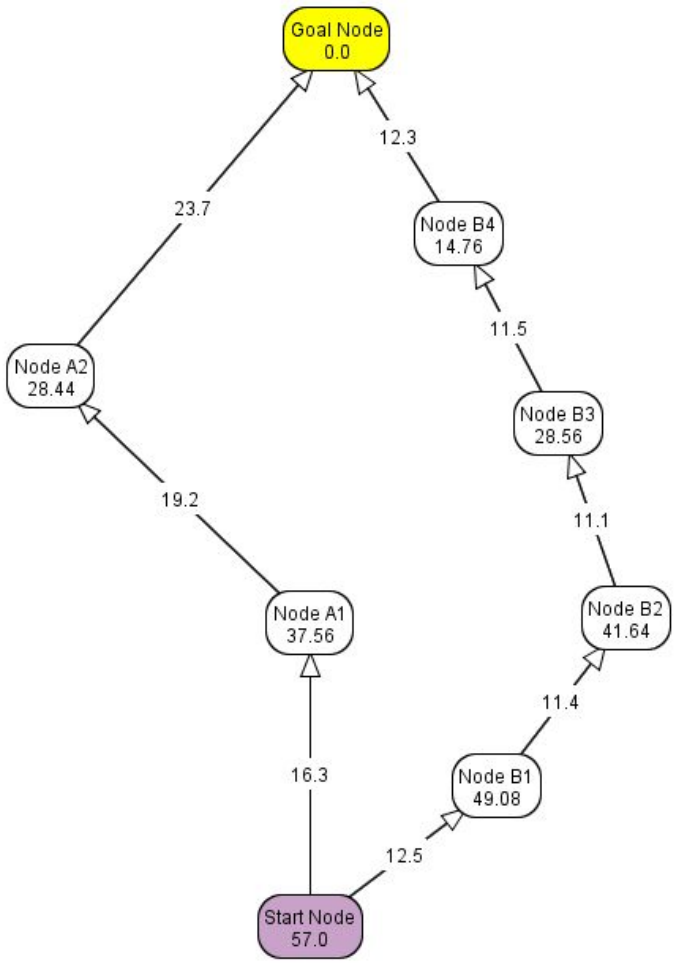


Diagram 3.2: A Search with overestimation of h(n) by 20%*

Conclusion

As such, as long as h(n) is an overestimation of the actual cost from n to the goal, we can conclude that A* search will

- Sometimes produce an non-optimal solution
- Sometimes be more efficient