CZ3005 LAB 1 Report

Name: Kannan Shivani

Matric number: U1822998H

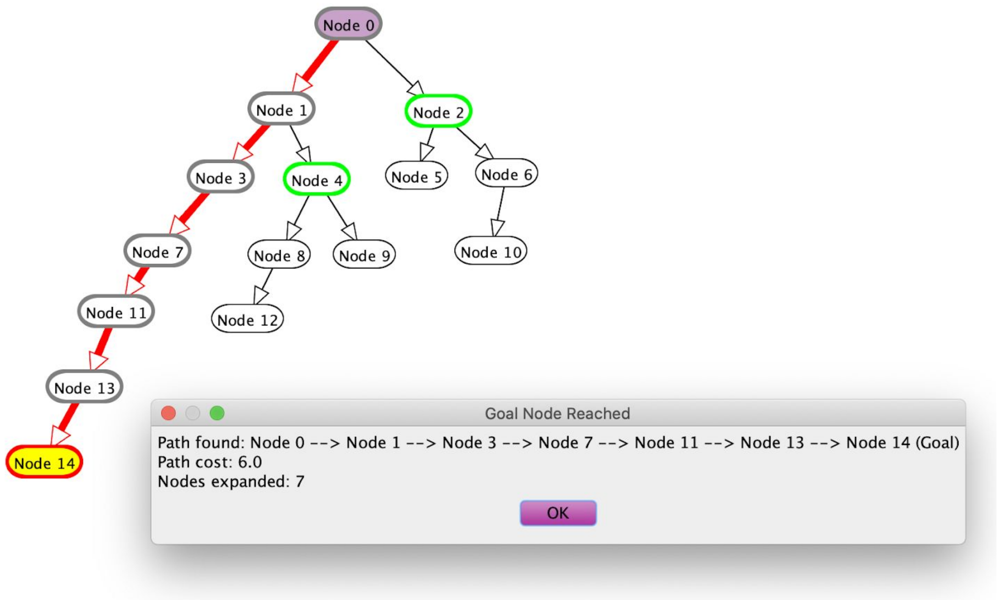
Lab Group: TSP6

Question One [55 marks]

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

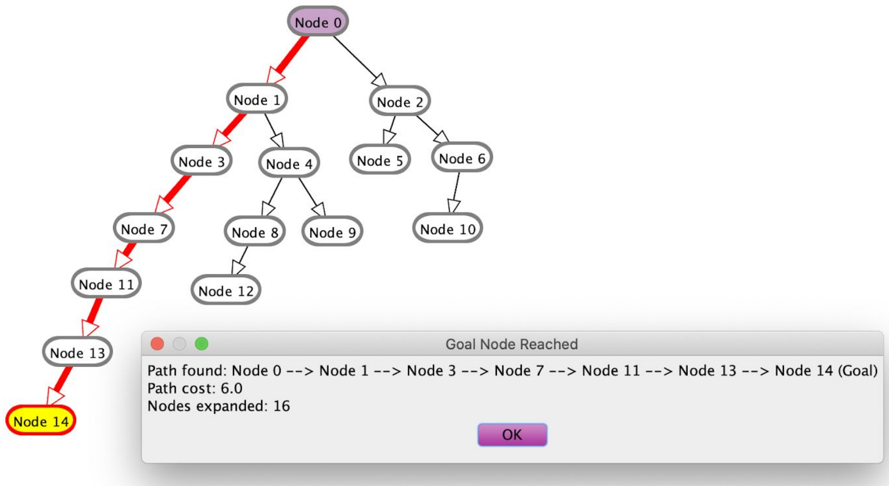
Note : All arc costs are 1.0, and arc lengths are not to scale.

Result of DFS:



```
Path found: Node 0 --> Node 1 --> Node 3 --> Node 7 --> Node 11 --> Node 13 --> Node 14 (Goal)
Path cost: 6.0
Nodes expanded: 7
```

Result of BFS:



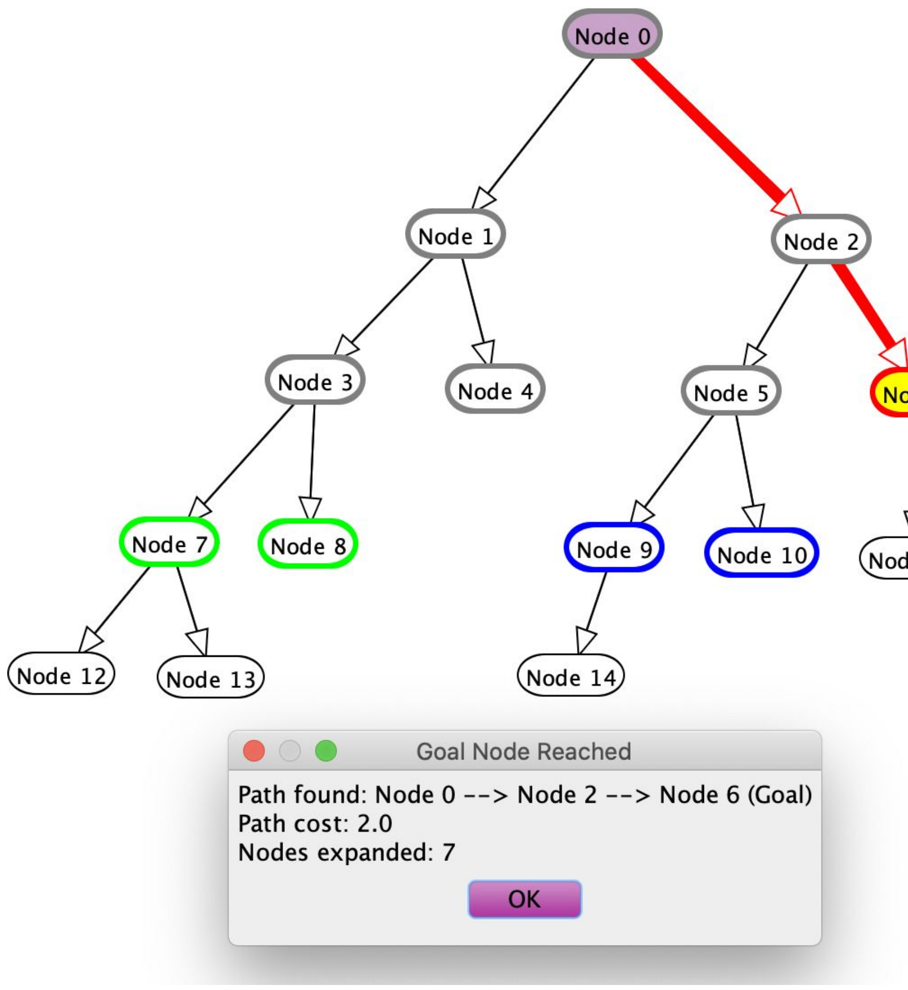
```
Path found: Node 0 --> Node 1 --> Node 3 --> Node 7 --> Node 11 --> Node 13 --> Node 14 (Goal)
Path cost: 6.0
Nodes expanded: 16
```

This shows that DFS is much more efficient in this case, while expanding only 7 nodes whereas BFS expands 16 nodes

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

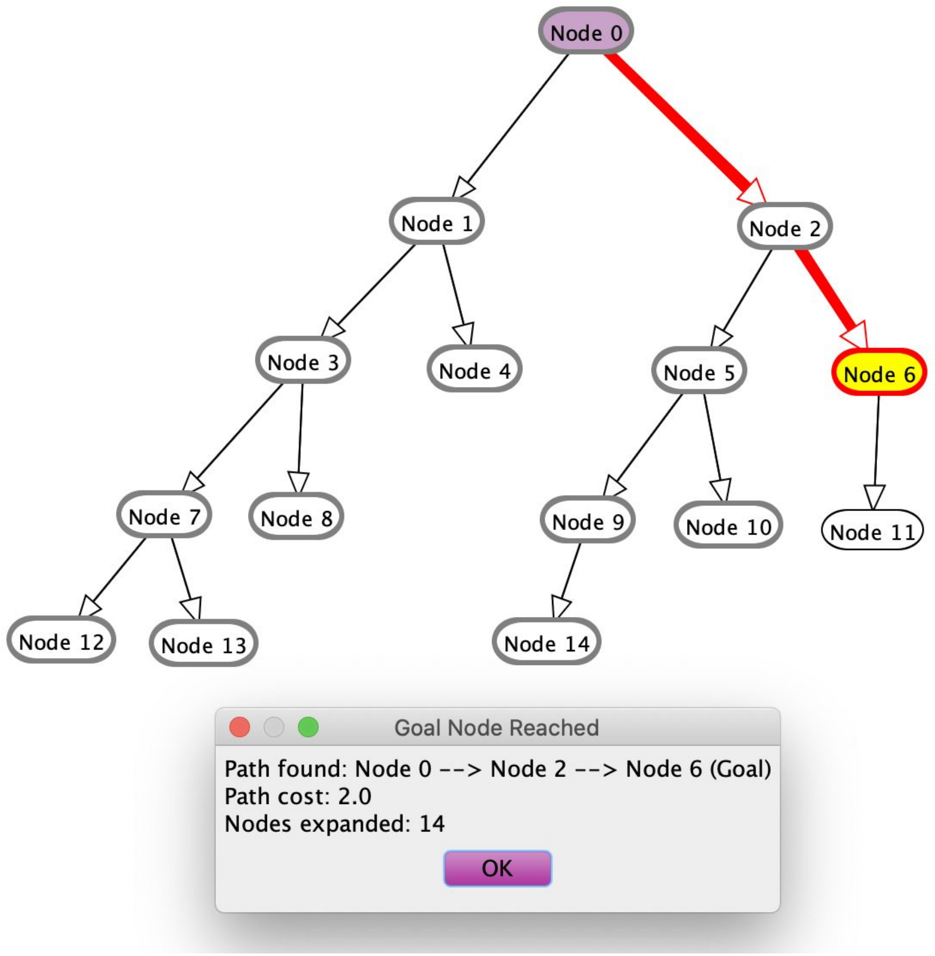
Note: All arc costs are 1.0, and arc length is not to scale.

Result of BFS:



```
Path found: Node 0 --> Node 2 --> Node 6 (Goal)
Path cost: 2.0
Nodes expanded: 7
```

<u>Result of DFS:</u>



```
Path found: Node 0 --> Node 2 --> Node 6 (Goal)
Path cost: 2.0
Nodes expanded: 14
```

This shows that BFS performs much better, expanding only 7 nodes compared to 14 nodes for DFS.

<u>Analysis of results: DFS vs BFS</u>

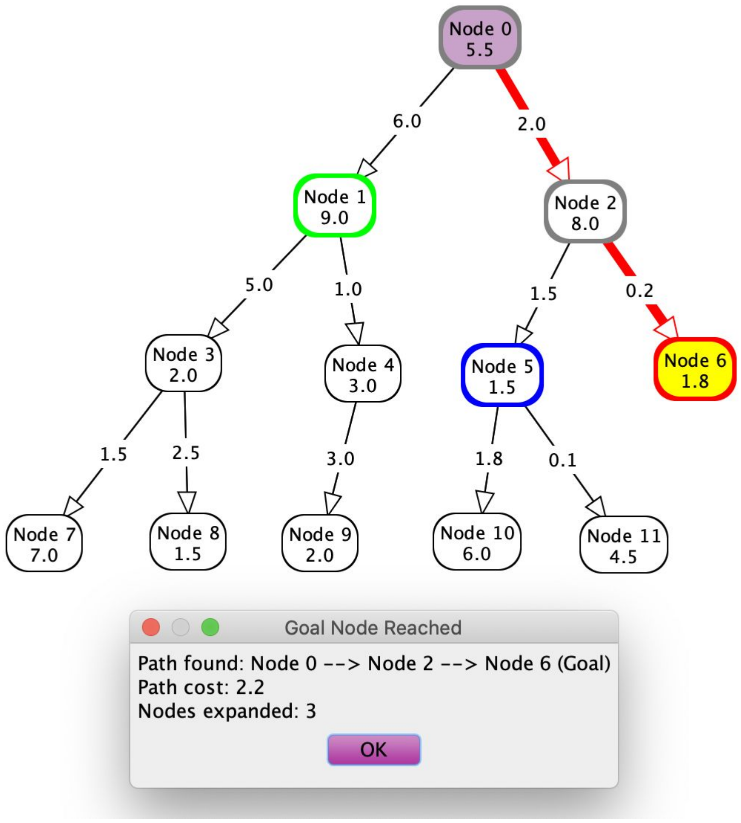
Performance of one over the other largely depends on the type of graph (structure of search tree) and the position and number of the goal nodes. Some <u>general</u> observations are that:

- In denser trees (more breadth), DFS will perform better
- BFS performs better for graphs with more frequent solutions
- BFS performs better if the goal states are to the start node, ie at lower depth
- Space complexity for BFS is larger (O(b^d) vs O(bm)),may not be practical to use BFS for larger search spaces

DFS also has some limitations to providing complete solutions. DFS will not work with infinite-depth spaces, or finite-depth spaces with loops.onsight

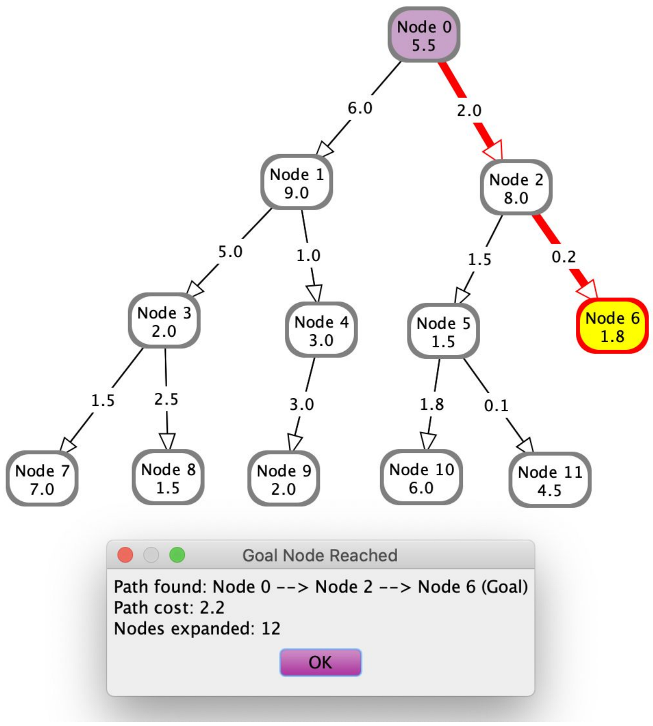
(c) Give a graph where A* search is more efficient than either DFS or BFS. [15 marks]

Result of A* search:



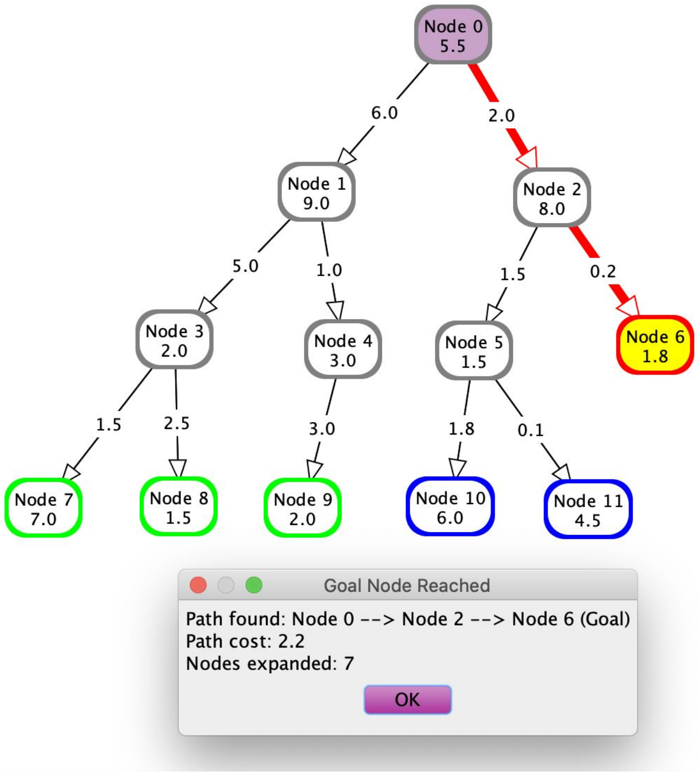
Path found: Node 0 --> Node 2 --> Node 6 (Goal)
Path cost: 2.2
Nodes expanded: 3

Result of DFS



```
Path found: Node 0 --> Node 2 --> Node 6 (Goal)
Path cost: 2.2
Nodes expanded: 12
```

Result of BFS

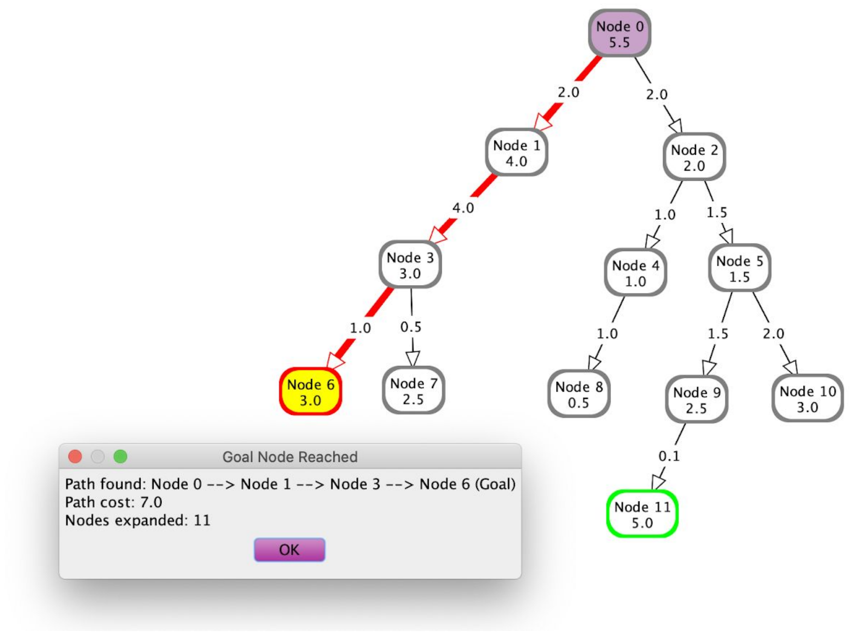


```
Path found: Node 0 --> Node 2 --> Node 6 (Goal)
Path cost: 2.2
Nodes expanded: 7
```

Search used	A* search	BFS	DFS
Order of expansion	0,2,6	0,1,2,3,4,5,6	0,1,3,7,8,4,9,2,5,10,11,6
Expanded nodes	3	7	12

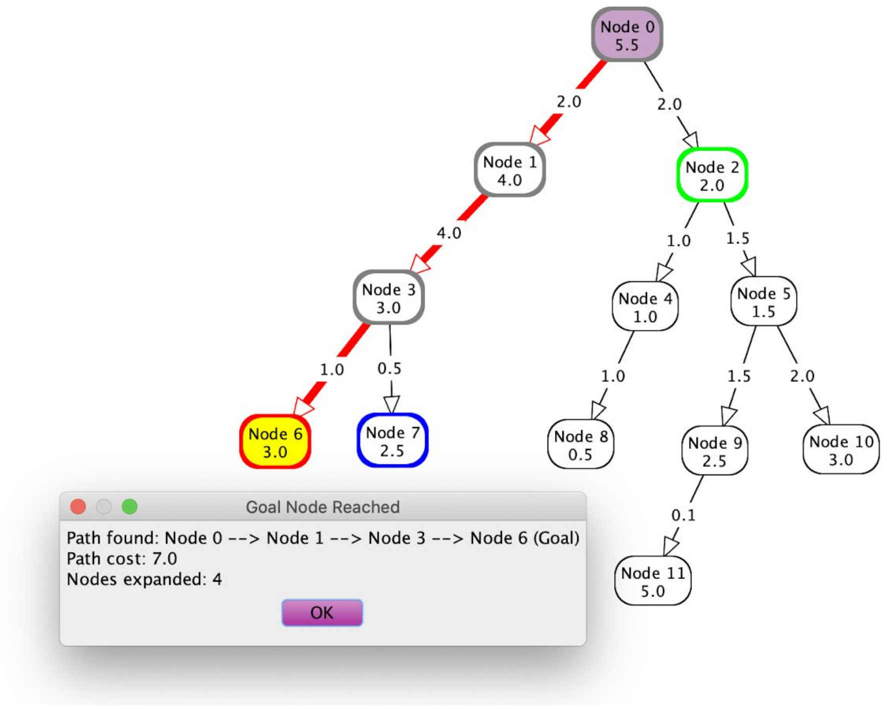
(d) Give a graph where DFS and BFS are both more efficient than A* search. [15 marks]

Result of A* search:



Path found: Node 0 --> Node 1 --> Node 3 --> Node 6 (Goal)
Path cost: 7.0
Nodes expanded: 11

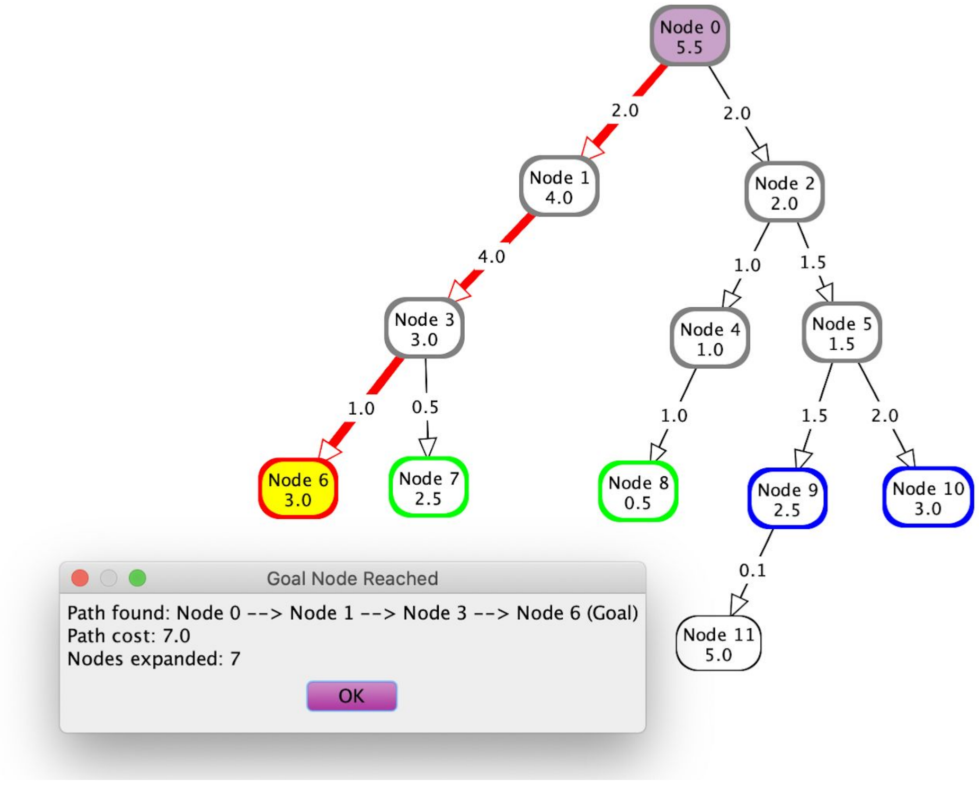
DFS



Goal Node Reached

Path found: Node 0 --> Node 1 --> Node 3 --> Node 6 (Goal)
Path cost: 7.0
Nodes expanded: 4

BFS



Goal Node Reached

Path found: Node 0 --> Node 1 --> Node 3 --> Node 6 (Goal)
Path cost: 7.0
Nodes expanded: 7

Search used	A* search	BFS	DFS
Order of expansion	0,2,4,8,5,1,9,10, 3,7,6	0,1,2,3,4,5,6	0,1,3,6
Expanded nodes	11	7	4

A* tries to find the shortest path, and avoids the goal node. BFS and DFS ignore the edge costs and thus will expand the goal node before A*.

Question Two [45 marks]

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

In any graph, f(n) represents the actual cost of an optimal path from s to n.

$f(n) = g(n) + h(n)$, where g(n) represents the actual optimal path cost, and h(n) is a heuristic estimation.

Hypothesis: Reducing h(n) will **always** lead to equal or more nodes expanded. The more h(n) is reduced, the more nodes expanded. However, A* will still always find an optimal path.

Proof for optimality of A* when h(n) is an underestimate:

Proof by contradiction for optimality, assume that there exists a solution where A* search is not optimal for when h(n) is an underestimate, ie a ‘true’ optimal path with cost less than the path cost returned by A* search.

Let f’(n) = g’(n) + h’(n) be the evaluation function of A*, where g’(n) is the cost from s to n with minimum cost so far found by A*.

Suppose A* terminates at a goal node *t,* with f’(n) = g’(t) > f(s).

There always exists an open node n’ on P with g’(n’)=g(n) ----- (1).

Let $P = (s = n_0, n_1, n_2, ...n_k = n)$. nWhen n’ = s, g’(s)=g(s)=0, and thus g’(n’) = g(n)

For any optimal path P from s to any goal node of s, there exists an open node n’ on P with f’(n’)<f(s).

By definition and from (1),

$$f'(n') = g'(n') + h(n') = g(n') + h'(n') \leq g(n') + h(n') = f(n')$$

Since P is an optimal path, for all n'∈ P.
From this, we can determine that there exists an open node n' on an optimal path with $f'(n') \leq f(s) < f'(t)$

At this stage, n' would be expanded by A* instead of t. This contradicts the assumption that A* terminates, and proves that A* is optimal.

<u>Proof that reducing h(n) will not have less nodes expanded:</u>

Let **PC** be the path cost of the optimal path to a goal.

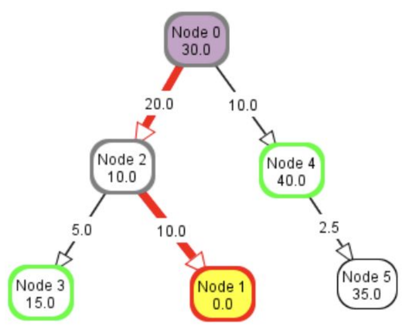
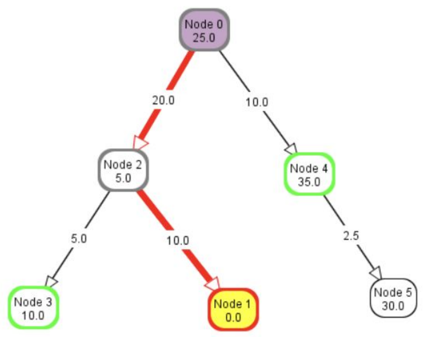
At each step i, let K_i be the set of nodes not on the optimal path and O_i be the set of nodes on the optimal path.

Since h(n) is an underestimate,
At step i, for every node in sets K_i and O_i,
$$f(k) = g(k) + h(k) < PC$$

$$f(o) = g(o) + h(o) < PC$$

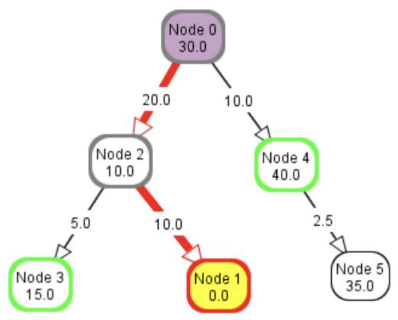
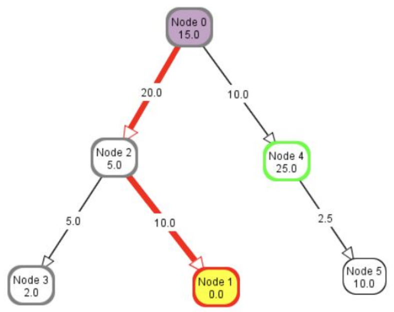
Depending on how underestimated h(n) is, 2 situations will occur. Case 1 is when the underestimation is not as large, and $f(o) < f(k)$ for all o and k, leading to A* expanding node o. Case 2 occurs when $h(n)$ is so underestimated that $f(k) < f(o)$. Case 1 leads to the same number of nodes expanded as A* originally, while case 2 would lead to an increase in nodes expanded.

<u>Example of case 1, same number of nodes generated:</u>

Original A* result	Underestimation of h(n),
 Node 0 30.0 20.0 10.0 Node 2 10.0 Node 4 40.0 5.0 10.0 2.5 Node 3 15.0 Node 1 0.0 Node 5 35.0	 Node 0 25.0 20.0 10.0 Node 2 5.0 Node 4 35.0 5.0 10.0 2.5 Node 3 10.0 Node 1 0.0 Node 5 30.0

Path to last Goal Node: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 30.0 Nodes expanded: 3	Path to last Goal Node: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 30.0 **Nodes expanded: 3**

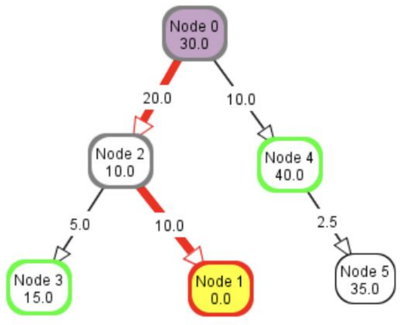
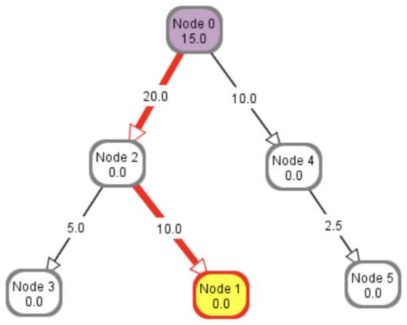
<u>Example of case 2, more nodes generated:</u>

Original A* result	Underestimation of h(n),
 Node 0 30.0 20.0 10.0 Node 2 10.0 Node 4 40.0 5.0 10.0 2.5 Node 3 15.0 Node 1 0.0 Node 5 35.0	 Node 0 15.0 20.0 10.0 Node 2 5.0 Node 4 25.0 5.0 10.0 2.5 Node 3 2.0 Node 1 0.0 Node 5 10.0
Path to last Goal Node: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 30.0 Nodes expanded: 3	Path to last Goal Node: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 30.0 **Nodes expanded: 4**

An extreme example of case 2 is when h(n) is reduced to 0, and uniform cost search is executed instead

<u>Further reduction of h(n) to 0:</u>

Original A* result	h(n) = 0

 Node 0 30.0 20.0 10.0 Node 2 10.0 Node 4 40.0 5.0 10.0 2.5 Node 3 15.0 Node 1 0.0 Node 5 35.0	 Node 0 15.0 20.0 10.0 Node 2 0.0 Node 4 0.0 5.0 10.0 2.5 Node 3 0.0 Node 1 0.0 Node 5 0.0
Path to last Goal Node: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 30.0 Nodes expanded: 3	Path to last Goal Node: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 30.0 **Nodes expanded: 6**

Observations:

The number of nodes expanded increases as h(n) is reduced. Accuracy of A* is still achieved, with all the searches achieving the optimal solution. When h(n) is reduced to 0, the search acts as uniform cost search, and h(n) has no effect on the choice of which node to expand.

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

Hypothesis: A* will always find the best path(optimal solution). When h(n) is the exact distance, A* will expand no more than the number of nodes it needs to reach the goal state. The only exception to this is when there are multiple optimal paths to the goal state.

Proof of optimality:

A similar explanation from 2a can be used to prove this

Let $f'(n) = g'(n) + h'(n)$ be the evaluation function of A*, where g'(n) is the cost from s to n with minimum cost so far found by A*.

There always exists an open node n' on P with g'(n')=g(n)

For any optimal path P from s to any goal node of s, there exists an open node n' on P with f'(n')<f(s).

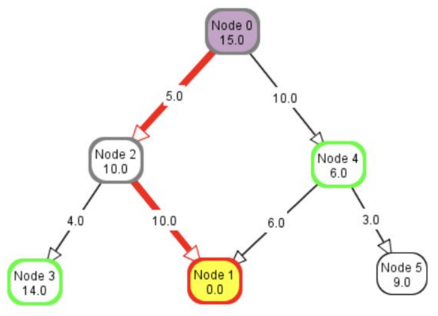
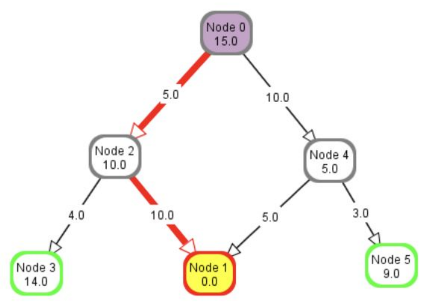
By definition and from (1),

$$f'(n') = g'(n') + h'(n') = g(n') + h'(n') \leq g(n') + h(n') = f(n')$$

Since P is an optimal path, for all n'∈ P.
From this, we can determine that there exists an open node n' on an optimal path with $f'(n') \leq f(s) < f'(t)$

This will occur at every step of A*, guaranteeing that it will expand an optimal node at every path, unless there is an additional optimal path, in which case extra nodes along that path will be expanded.

Examples of the 2 cases:

Only 1 optimal path	Multiple optimal paths
 Node 0 15.0 5.0 10.0 Node 2 10.0 Node 4 6.0 4.0 10.0 6.0 3.0 Node 3 14.0 Node 1 0.0 Node 5 9.0	 Node 0 15.0 5.0 10.0 Node 2 10.0 Node 4 5.0 4.0 10.0 5.0 3.0 Node 3 14.0 Node 1 0.0 Node 5 9.0
Path to last Goal Node: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 15.0 **Nodes expanded: 3**	Path to last Goal Node: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 15.0 **Nodes expanded: 4**

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

Hypothesis: A* will not be guaranteed to find the shortest path to the goal(not optimal) in some cases. If an optimal solution is found, no additional nodes outside of the optimal path will be expanded. If an optimal solution is not found, the number of nodes expanded is indeterminable.

<u>Proof that more nodes are expanded if optimal solution is found:</u>

Let **PC** be the path cost of the optimal path to a goal.

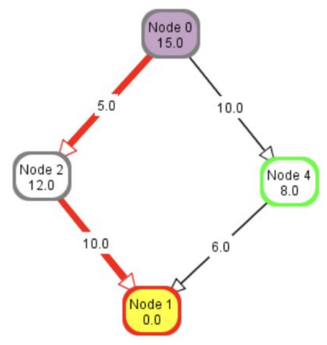
At each step i, let K_i be the set of nodes not on the optimal path and O_i be the set of nodes on the optimal path.

Since h(n) is an overestimate, $f(n) = g(n) + h(n) > PC$
At step i, for every node in sets K_i and O_i,
$f(k) = g(k) + h(k) > PC$
$f(o) = g(o) + h(o) > PC$

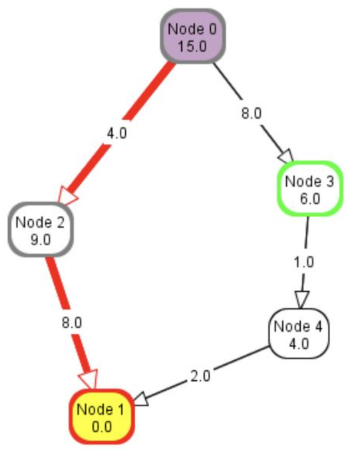
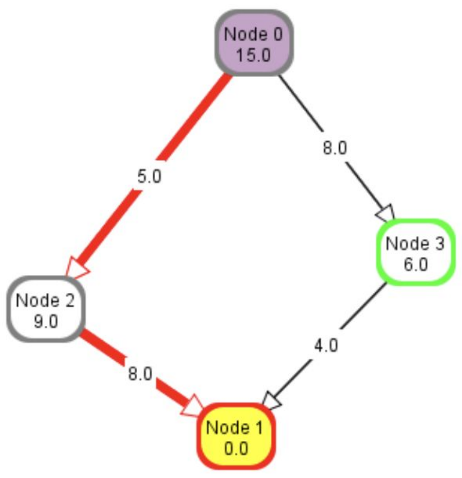
Depending on how overestimated h(n) is, 2 situations will occur. Case 1 is when the overestimation is not as large, and $f(k) > f(o)$ for all o and k, leading to A* expanding node o. Case 1 leads to the same number of nodes expanded as A* originally.

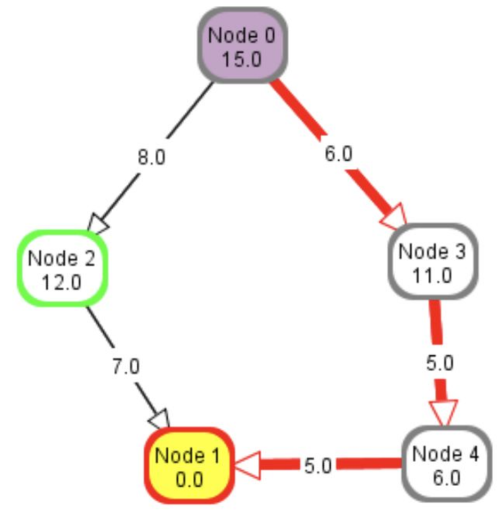
In case 2, $f(k) < f(o)$ for at least 1 node in K_i, and A* search would expand a node in a non optimal path. For a non optimal solution to be generated, $f(k) < f(o)$ for every step of A* search for all generated o. When this occurs, depending on the graph, more, less or an equal number of nodes will be expanded.

<u>Case 1</u>

h(n) is not a large overestimate	
 Node 0 15.0 5.0 10.0 Node 2 12.0 Node 4 8.0 10.0 6.0 Node 1 0.0	
Path found by A*: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 15.0 Nodes expanded: 3	Optimal path: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 15.0 Nodes expanded: 3

<u>Case 2, non-optimal solution found:</u>

Number of nodes less than optimal solution	
 Node 0 15.0 4.0 8.0 Node 3 6.0 Node 2 9.0 1.0 8.0 Node 4 4.0 2.0 Node 1 0.0	
Path found by A*: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 12.0 Nodes expanded: 3	Optimal path: Node 3 --> Node 4 --> Node 1 (Goal) Cost: 11.0 Nodes expanded: 4
Number of nodes equal to optimal solution	
 Node 0 15.0 8.0 5.0 Node 3 6.0 Node 2 9.0 4.0 8.0 Node 1 0.0	
Path found by A*: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 13.0 Nodes expanded: 3	Optimal path: Node 0 --> Node 3 --> Node 1 (Goal) Cost: 12.0 Nodes expanded: 3

Number of nodes more than the optimal solution	
 Node 0 15.0 8.0 6.0 Node 2 12.0 Node 3 11.0 7.0 5.0 Node 1 0.0 5.0 Node 4 6.0	
Path found by A*: Node 0 --> Node 3 --> Node 4 --> Node 1 (Goal) Cost: 16.0 Nodes expanded: 4	Optimal path: Node 0 --> Node 2 --> Node 1 (Goal) Cost: 15.0 Nodes expanded: 3

When overestimating h(n), A* acts more similarly to a greedy algorithm, and optimality is not guaranteed, but search time usually improves.