Question 1) *Consider the search tree shown in Figure 1. The number next to each edge is the cost of the performing the action corresponding to that edge. You start from the node A. The goal is the reach node G. List the order in which nodes will be visited using:*

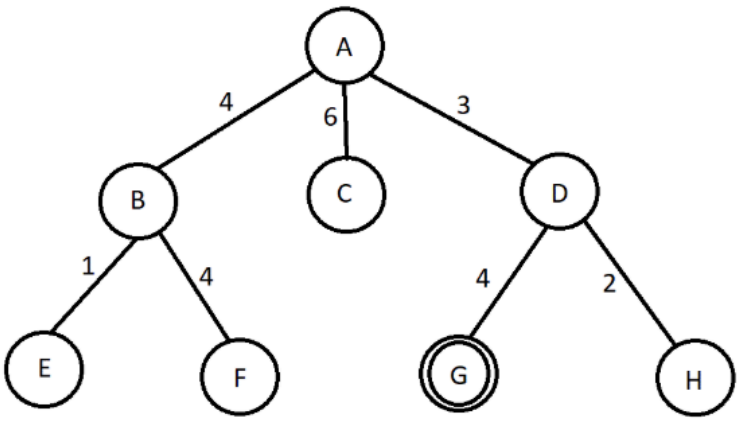


Figure 1: Search Tree for Problem 1

1. Breadth-First Search:
   Fringe -

A	B	C	D	E	F	**G**

2. Depth-First Search:
   Fringe -

A	B	E	F	C	D	**G**

3. Iterative Deeping Search (show all the iterations):
   Fringe at Iteration one -

A

   Fringe at Iteration two -

A	B	C	D

   Fringe at Iteration three -

A	B	E	F	C	D	**G**

4. Uniform Cost Search:
   Fringe -

A/0	D/3	B/4	E/5	H/5	C/6	**G/7**

Question 2) *A social network graph (SNG) is a graph where each vertex is a person and each edge represents an acquaintance. In other words, an SNG is a graph showing who knows who. For example, in the graph shown on Figure 2, George knows Mary and John, Mary knows Christine, Peter and George, John knows Christine, Helen and George, Christine knows Mary and John, Helen knows John, Peter knows Mary.*

The degrees of separation measure how closely connected two people are in the graph. For example, John has 0 degrees of separation from himself, 1 degree of separation from Christine, 2 degrees of separation from Mary, and 3 degrees of separation from Peter.

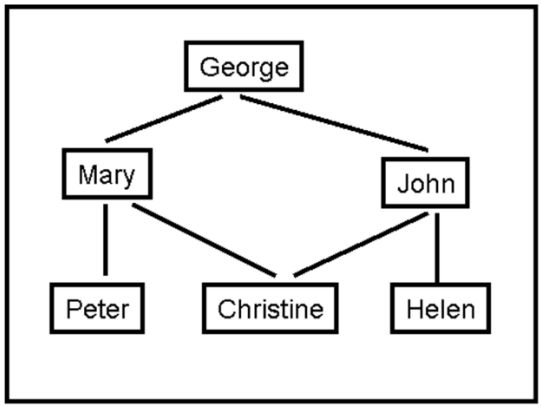
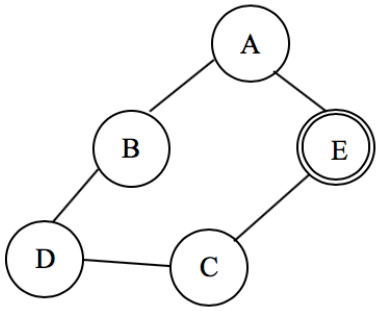


Figure 2: A Social Network Graph

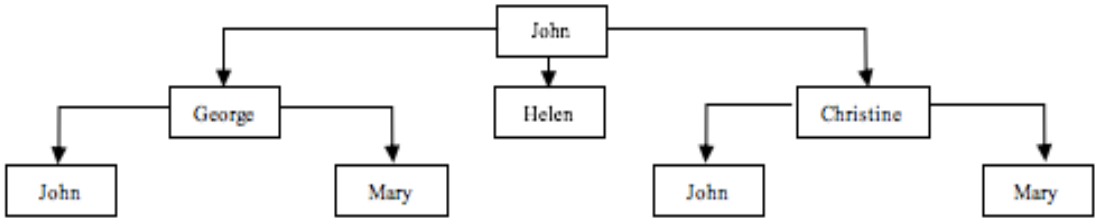
i. *From among general tree search using breadth-first search, depth-first search, iterative deepening search, and uniform cost search, which one(s) guarantee finding the correct number of degrees of separation between any two people in the graph (assume we are using the strategies without any modifications)?*

- Using a breadth-first search: Yes, this will find the expected output only when the degree of branching is less.
- Using a depth-first search: No, this may not lead to the required solution. There are two reasons for the same:
    - If we have no track of visited nodes, then the fringe would end up in an infinite iteration just between parent node and its first child.
    - In case, we track the visited nodes, then it would result in wrong output. For example, if fringe takes in node A and starts depth search, then the degree of separation of A from E would be 4. Indeed, the answer to the same is 1.



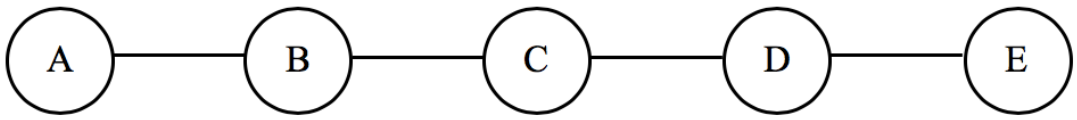
- Using iterative deepening search: Yes, this will find the expected output.
- Using uniform cost search: Yes, this will find the expected output only when the degree of branching is less.
- Using a depth-first search: No, this may not lead to the required solution. There are two reasons for the same:
    - If we have no track of visited nodes, then the fringe would end up in an infinite iteration just between parent node and its first child.
    - In case, we track the visited nodes, then it would result in wrong output. For example, if fringe takes in node A and starts depth search, then the degree of separation of A from E would be 4. Indeed, the answer to the same is 1.

ii. *For the SNG shown in Figure 2, draw the first three levels of the search tree, with John as the starting point (the first level of the tree is the root). Is there a one-to-one correspondence between nodes in the search tree and vertices in the SNG (i.e. does every node in the search tree correspond to a vertex in the SNG)? Why, or why not? In your answer here, you should assume that the search algorithm does not try to avoid revisiting the same state.*



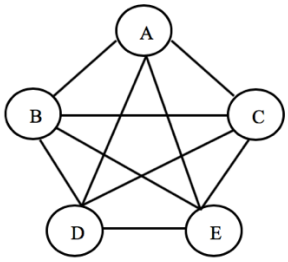
No, there is no one to one correspondence between nodes in the above search tree and the vertices in the SNG. This is because, one of the vertices in SNG graph i.e. 'Peter' is not a node in the above search tree.

iii. *Draw an SNG containing exactly 5 people, where at least two people have 4 degrees of separation between them.*



In the above SNG diagram, there are five people named 'A', 'B', 'C', 'D', and 'E'. Here, person 'A' has a degree of separation of 4 from person 'E' and person 'E' has a degree of separation of 4 from person 'A'.

iv. Draw an SNG containing exactly 5 people, where all people have 1 degree of separation between them.



In the above SNG diagram, there are five people named 'A', 'B', 'C', 'D', and 'E'. Here, every person has 1 degree of separation from each other except for themselves.

v. In an implementation of breadth-first tree search for finding degrees of separation, suppose that every node in the search tree takes 1KB of memory. Suppose that the SNG contains one million people. Outline (briefly but precisely) how to make sure that the memory required to store search tree nodes will not exceed 1GB (the correct answer can be described in one-two lines of text). In your answer here you are free to enhance/modify the breadth-first search implementation as you wish, as long as it remains breadth-first (a modification that, for example, converts breadth-first search into depth-first search or iterative deepening search is not allowed).

- We need to keep track of node address that have been visited and expanded. We could name the same array as 'visited_node'.
- Before we push any node to the fringe, we first check if the current node address is available in the 'visited_node' array. If the function, returns true, then we do not have to push the node to the fringe.
- This will help preserve space in the memory. And thus, even with the worst case Scenario, that is a skewed tree with the search node as the leaf, maximum nodes in the fringe would be a million, thus not exceeding memory more than 1GB..

  Max 1,000,000 nodes = 1KB * 1,000,000 = 1,000,000KB = 1GB

Question 3) *Consider the search space shown in Figure 3. D is the only goal state. Costs are undirected. For each of the following heuristics, determine if it is admissible or not. For non-admissible heuristics, modify their values as needed to make them admissible.*

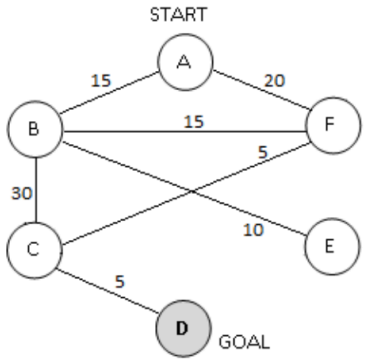


Figure 3. A search graph showing states and costs of moving from one state to another. Costs are undirected.

General:

- In order for a heuristic to be admissible, the estimated cost should always be lesser than the actual cost of reaching the goal.

- To check if the following heuristics are admissible, we need to compute the optimal path to reach the destination states ‘D’ from all other states. We can compute the same by computing the cost of ‘D’ to all other states until all nodes are not closed. Computation for the same is shown below using uniform cost search. Here we also have kept track of the visited states.

Path from state ‘D’ to all other states [until all states are not visited] -

Current Node	Fringe Additions	Sorted Fringe
NULL	D/0	D/0
D/0	C/5	C/5
C/5	B/35 F/10	F/10 B/35
F/10	B/35 A/30 B/25	B/25 A/30 B/35
B/25	A/30 B/35 A/40 E/35	A/30 B/35 E/35 A/40
A/30	B/35 E/35 A/40	B/35 E/35 A/40
B/35 – Visited	E/35 A/40	E/35 A/40
E/35	A/40	A/40
A/40 – Visited	NULL	NULL

Smallest cost from state ‘D’ to all other nodes are:

A	B	C	E	F
30	25	5	35	10

i. *Heuristic 1:*

h(A, B, C, D, E, F) = (50 , 35 , 5 , 0 , 45 , 10)

The above heuristics are not admissible. They need to less or equal to the smallest cost. Therefore, we can rewrite the same as:

h(A, B, C, D, E, F) = (30 , 35 , 5 , 0 , 35 , 10)

ii. *Heuristic 2:*

h(A, B, C, D, E, F) = (70 , 70 , 70 , 70 , 70 , 70)

The above heuristics are not admissible. The heuristic for destination state ‘D’ has to be 0 and other all heuristics also need to be less or equal to the smallest cost to destination state. Therefore, we can rewrite the same as:

h(A, B, C, D, E, F) = (30 , 35 , 5 , 0 , 35 , 10)

iii. *Heuristic 3:*

h(A, B, C, D, E, F) = (0 , 0 , 0, 0 , 0, 0)

The above heuristics follow all the rules and are admissible.

Question 4) *Consider a search space, where each state can be a city, suburb, farmland, or mountain. The goal is to reach any state that is a mountain. Here are some rules on the successors of different states:*

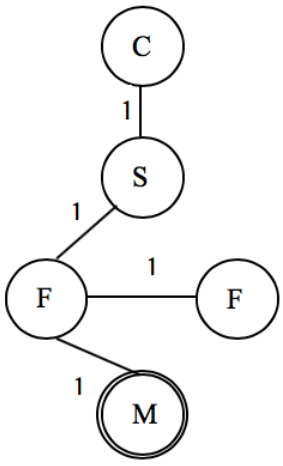
- *Successors of a city are always suburbs.*
  - *Each city has at least one suburb as a successor.*
- *Successors of a suburb can only be cities, or suburbs, or farms.*
  - *Each suburb has at least one city as a successor.*
- *Successors of a farm can only be farms, or suburbs, or mountains.*
  - *Each farm has at least one other farm as a successor.*
- *Successors of a mountain can only be farms.*

Define the best admissible heuristic h you can define using only the above information (you should not assume knowledge of any additional information about the state space). By "best admissible" we mean that h(n) is always the highest possible value we can give, while ensuring that heuristic h is still admissible.

You should assume that every move from one state to another has cost 1.

The figure below, which is constructed based on the rules given. Based on the following we can have the following heuristics for the given graph:

h(city) = 3
h(suburbs) = 2
h(farm) = 1
h(mountain) = 0



City	C
Suburbs	S
Farm	F
Mountain	M

Question 5) *The 24-puzzle is an extension of the 8-puzzle, where there are 24 pieces, labeled with the numbers from 1 to 24, placed on a 5x5 grid. At each move, a tile can move up, down, left, or right, but only if the destination location is currently empty. For example, in the start state shown above, there are three legal moves: the 12 can move down, the 22 can move left, or the 19 can move left. The goal is to achieve the goal state shown above. The cost of a solution is the number of moves it takes to achieve that solution.*

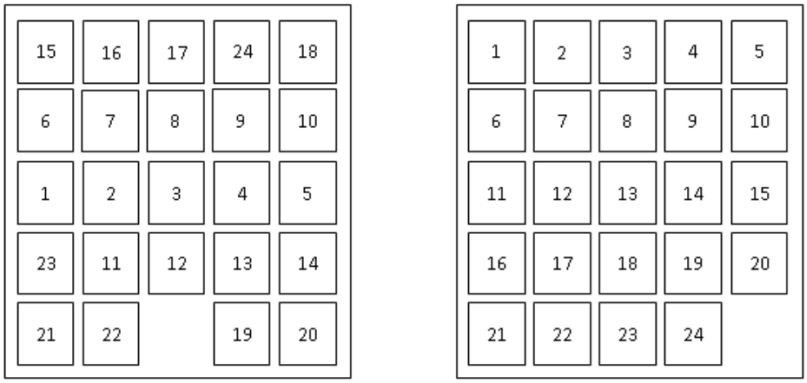


Figure 4. An example of a start state (left) and the goal state (right) for the 24-puzzle.

For some initial states, the shortest solution is longer than 100 moves. For all initial states, the shortest solution is at most 208 moves.

An additional constraint is that, in any implementation, storing a search node takes 1000 bytes, i.e., 1KB of memory. Consider general tree search using the strategies of breadth-first search, depth-first search, iterative deepening search and uniform cost search.

Let ‘b’ => the maximum branching factor for the given tree,
‘d’ => depth of the least-cost solution.
‘m’ => maximum depth of the state space
‘C*’ => cost of optimal solution

Then, the space property for each of the algorithm can be defined as:

Algorithm	Space Property
Breadth First Search	$O(b^{(d+1)})$
Depth First Search	O(bm)
Iterative Deepening Search	O(bd)
Uniform Cost Search	$O(b^{(C^*/\varepsilon)})$

As given:

⇒ For some initial states, the shortest solution is longer than 100 moves. For all initial states, the shortest solution is at most 208 moves. Let us consider the worst case scenario, then ‘d’ = 208.

⇒ According to the puzzle problem, each state can have a maximum of four moves. That is to move up, down, left or right. This implies, that ‘b’ = 4.

Points:

⇒ We know that, the maximum depth of the state space problem can be taken as infinity. This gives ‘m’ = ∞.

⇒ As we are taking the step cost to be one from one state to another, then both Breadth-First Search and Uniform Cost Search would have same space complexity that is $O(b^{(d+1)})$.

Let us compute the space property for each of the algorithms, in the worst case scenario for 5x5 puzzle:

Algorithm	Space Property	Number of Nodes
Breadth First Search	$O(4^{(208+1)})$	$4^{(209)}$
Depth First Search	$O(4 \times \infty)$	∞
Iterative Deepening Search	O(4 x 208)	832
Uniform Cost Search	$O(4^{(208+1)})$	$4^{(209)}$

(a): Which (if any), among those methods, can guarantee that you will never need more than 50KB of memory to store search nodes? Briefly justify your answer.

Since every node search node holds a size of 1KB of memory, then there should not be more than 50 nodes in the memory. From the calculations above, the number of nodes visited for all the algorithms is more than 50. In that case, none of the search techniques can solve the problem with just 50KB of memory to store search nodes.

Algorithm	Number of Nodes in some initial states	Number of Nodes in all initial states
Breadth First Search	$4^{(101)} > 50$	$4^{(209)} > 50$
Depth First Search	$\infty > 50$	$\infty > 50$
Iterative Deepening Search	$400 > 50$	$832 > 50$
Uniform Cost Search	$4^{(101)} > 50$	$4^{(209)} > 50$

(b): Which (if any), among those methods, can guarantee that you will never need more than 1200KB of memory to store search nodes? Briefly justify your answer.

Since every node search node holds a size of 1KB of memory, then there should not be more than 1200 nodes in the memory. From the calculations above, only iterative deepening search technique can solve the problem as it generates only 400 and 832 search nodes in the memory, which is equal to 400KB and 832KB in memory respectively for some and all initial states. All the other algorithms generate more than 1200 nodes. Thus, with the 1200 KB of memory for search nodes, only iterative deepening search technique could be used.

Algorithm	Number of Nodes in some initial states	Number of Nodes in all initial states
Breadth First Search	$4^{(101)} > 1200$	$4^{(209)} > 1200$
Depth First Search	$\infty > 1200$	$\infty > 1200$
Iterative Deepening Search	**400 < 1200**	**832 < 1200**
Uniform Cost Search	$4^{(101)} > 1200$	$4^{(209)} > 1200$

Question 6) *Figures 5 and 6 show maps where all the towns are on a grid. Each town T has coordinates (Ti, Tj), where Ti Tj are non-negative integers. We use the term Euclidean distance for the straight-line distance between two towns, and the term driving distance for the length of the shortest driving route connecting two towns. The only roads that exist connect towns that have Euclidean (straight-line) distance 1 from each other (however, there may be towns with Euclidean distance 1 from each other that are NOT directly connected by a road, for example in Figure 4.*

Consider greedy search, where the node to be expanded is always the one with the shortest Euclidean distance to the destination [h(n) where h(n) is the Euclidean distance from n to the destination]. Also consider A search where the next node to expand is picked f(n) = g(n) + h(n) [when g(n) is cumulative actual road distance between start and n]. For each of the maps showing on Figures 5 and 6, which of the following statements is true?*

- *Greedy search always performs better than or the same as A*, depending on the start and end states.*
- *Greedy search always performs worse than or the same as A*, depending on the start and end states.*
- *Greedy search performs sometimes better, sometimes worse, and sometimes the same as A*, depending on the start and end states.*
- *Greedy search always performs the same as A*, irrespective of the start and end states.*

Justify your answer. For the purposes of this question, the performance of a search algorithm is simply measured by the number of nodes visited by that algorithm. Note that you have to provide separate answers for Figure 5 and for Figure 6.

Notes:

- Step cost be 1
- Euclid Distance Formula: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, where 'd' is distance between 2 points (x_1, y_1) and (x_2, y_2).
- Using the Euclid distance formula, let us compute the heuristics for all the nodes in the above graphs.
- Greedy Algorithm finds the best route with the help of smallest heuristic value h(x).
- A* Algorithm finds the best route with f(x) = g(x) + h(x), where g(x) is the cumulative cost and h(x) is the heuristic cost.

Figure 5:

- Let start node be (0, 0)
- Let destination node be (2, 1)

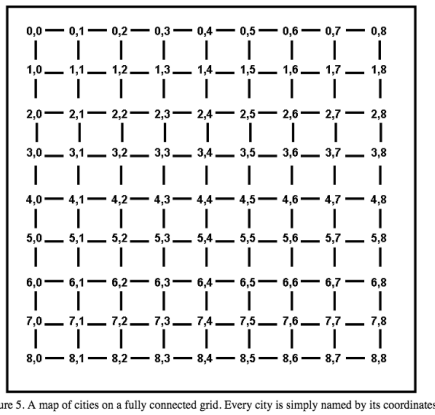


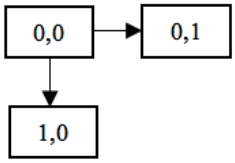
Figure 5. A map of cities on a fully connected grid. Every city is simply named by its coordinates.

Y —

Heuristics	0	1	2	3	4	5	6	7	8
0	2.236	2.000	2.236	2.828	3.605	4.472	5.385	6.324	7.280
1	1.414	1.000	1.414	2.236	3.162	4.123	5.099	6.082	7.071
2	1.000	0.000	1.000	2.000	3.000	4.000	5.000	6.000	7.000
3	1.414	1.000	1.414	2.236	3.162	4.123	5.099	6.082	7.071
4	2.236	2.000	2.236	2.828	3.605	4.472	5.385	6.324	7.280
5	3.162	3.000	3.162	3.605	4.242	5.000	5.830	6.708	7.615
6	4.123	4.000	4.123	4.472	5.000	5.656	6.403	7.211	8.062
7	5.099	5.000	5.099	5.385	5.830	6.403	7.071	7.810	8.602
8	6.082	6.000	6.082	6.324	6.708	7.211	7.810	8.485	9.219

Let us find the best route from greedy search algorithm.

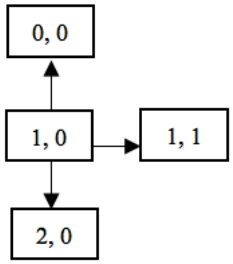
Iteration 1)



Fringe

1,0 **0,1**
1.414 **2.000**

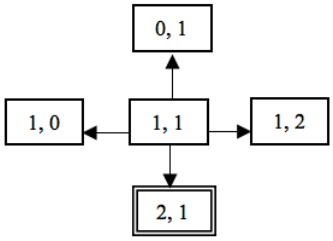
Iteration 2)



Fringe

1,1 **2,0** 0,1 **0,0**
1.000 **1.000** 2.000 **3.605**

Iteration 3)



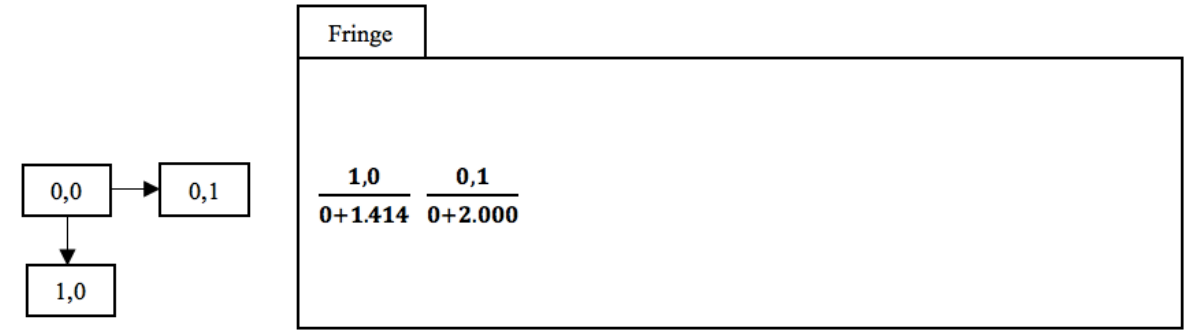
Fringe

2,1 2,0 **1,2** **1,0** 0,1 **0,1** 0,0
0.000 1.000 **1.414** **1.414** 2.000 **2.000** 3.605

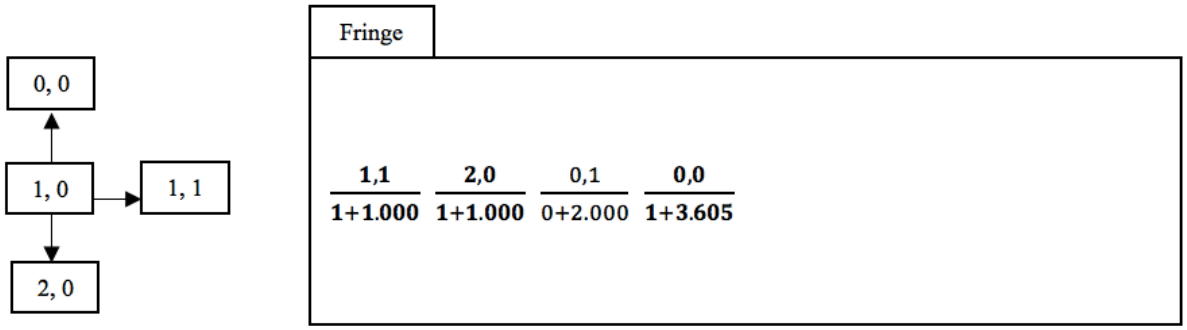
Iteration 4) Destination point (2,1) reached and the path to the same is
(0, 0) -> (1, 0) -> (1, 1) -> (2, 1)

Let us find the best route from A* search algorithm:

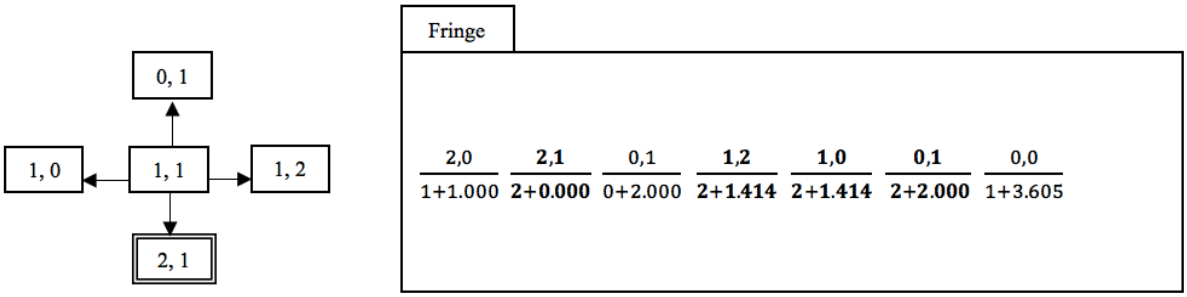
Iteration 1)



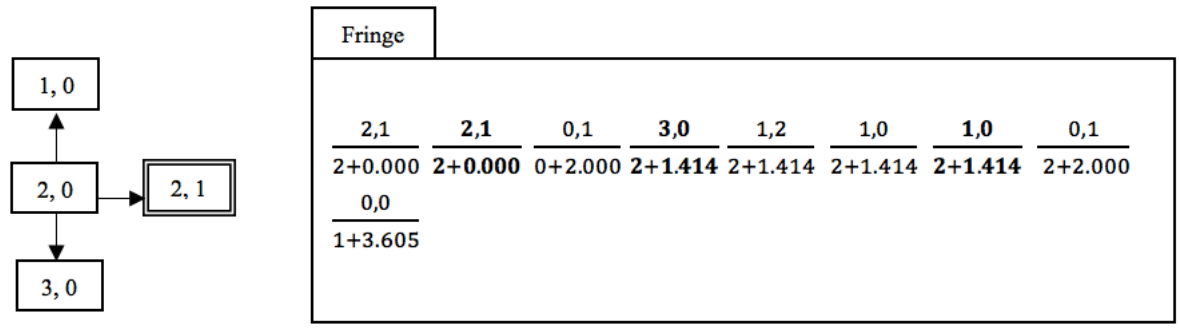
Iteration 2)



Iteration 3)



Iteration 4)



Iteration 5) Destination point (2, 1) reached in 5th Iteration. And the path for the same is
(0, 0) -> (1, 0) -> (2, 0) -> (2, 1)

Analysis:

As shown above, greedy search takes only four iterations to perform a search, while A* search takes five iterations. Moreover, the nodes in the fringe too, is comparative less in greedy search when compared to A* search fringe. A* would have got to the destination node with iteration 4 itself, but this is purely dependent on the sorting algorithm used. **This shows that greedy search analysis performs better than or the same as A*, depending on the start and end states.**

Figure 6:

- Let start node be (2,)
- Let destination node be (2, 1)

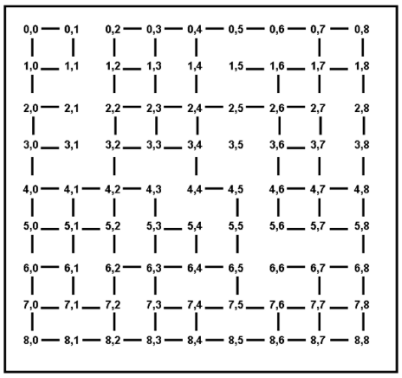


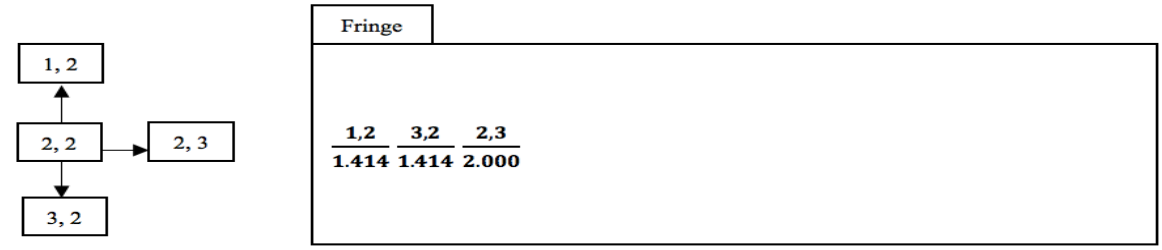
Figure 6. A map of cities on a partially connected grid. Every city is simply named by its coordinates.

Y —

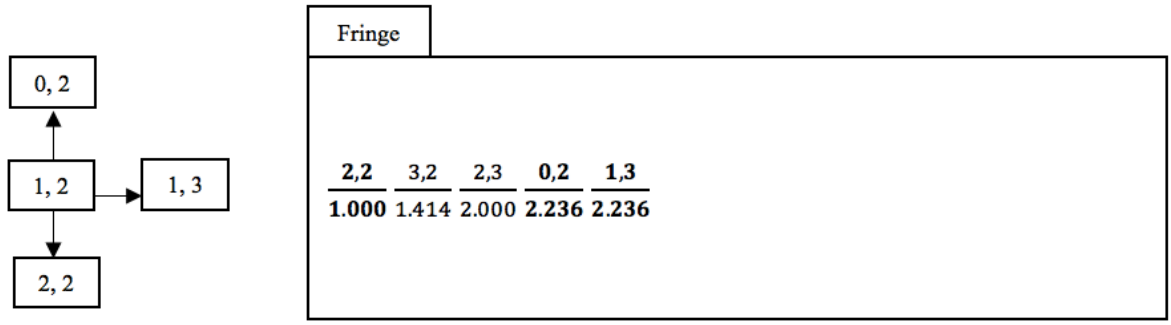
X | Heuristics	0	1	2	3	4	5	6	7	8
0	2.236	2.000	2.236	2.828	3.605	4.472	5.385	6.324	7.280
1	1.414	1.000	1.414	2.236	3.162	4.123	5.099	6.082	7.071
2	1.000	0.000	1.000	2.000	3.000	4.000	5.000	6.000	7.000
3	1.414	1.000	1.414	2.236	3.162	4.123	5.099	6.082	7.071
4	2.236	2.000	2.236	2.828	3.605	4.472	5.385	6.324	7.280
5	3.162	3.000	3.162	3.605	4.242	5.000	5.830	6.708	7.615
6	4.123	4.000	4.123	4.472	5.000	5.656	6.403	7.211	8.062
7	5.099	5.000	5.099	5.385	5.830	6.403	7.071	7.810	8.602
8	6.082	6.000	6.082	6.324	6.708	7.211	7.810	8.485	9.219

Let us find the best route from greedy search algorithm.

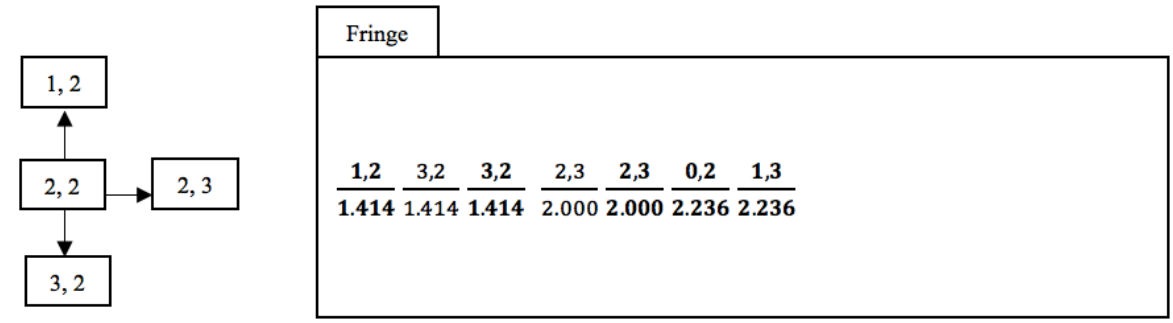
Iteration 1)



Iteration 2)



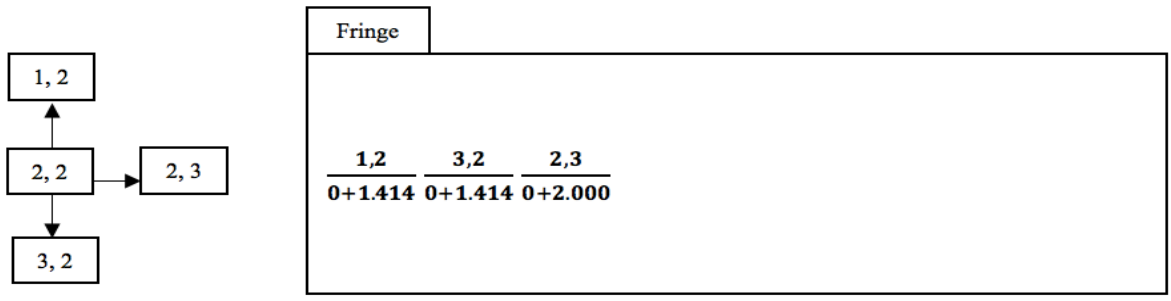
Iteration 3)



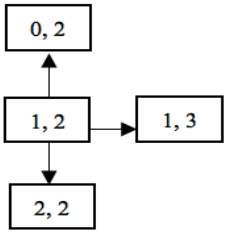
Iteration 4 and onwards) The destination is never reached as the search node can keep on hopping between (1, 2) and (2, 2)

Let us find the best route from greedy search algorithm.

Iteration 1)



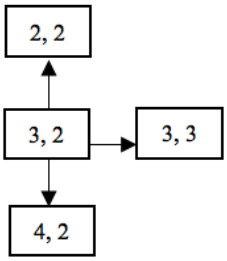
Iteration 2)



Fringe

3,2	**2,2**	2,3	**0,2**	**1,3**
0+1.414	**1+1.414**	0+2.000	**1+2.236**	**1+2.236**

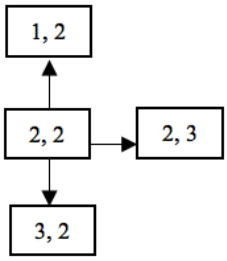
Iteration 3)



Fringe

2,2	**2,2**	2,3	**4,2**	**3,3**	0,2	1,3
1+1.414	**1+1.414**	0+2.000	**1+2.000**	**1+2.236**	1+2.236	1+2.236

Iteration 4)

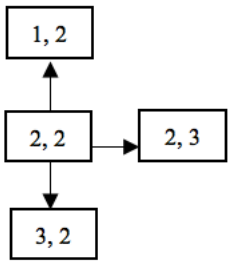


Fringe

2,2	2,3	4,2	3,3	0,2	1,3	**3,2**	**1,2**
1+1.414	0+2.000	1+2.000	1+2.236	1+2.236	1+2.236	**2+1.414**	**2+1.414**

2,3
2+2.000

Iteration 5)

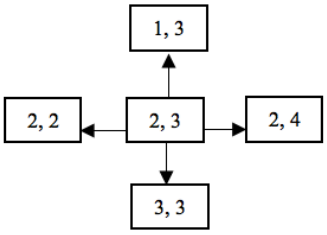


Fringe

2,3	4,2	3,3	0,2	1,3	3,2	1,2	**3,2**
0+2.000	1+2.000	1+2.236	1+2.236	1+2.236	2+1.414	2+1.414	**2+1.414**

1,2	2,3	**2,3**
2+1.414	2+2.000	**2+2.000**

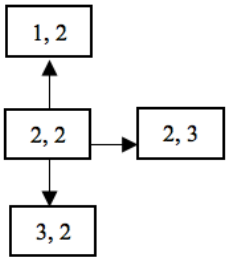
Iteration 6)



Fringe

2,2	4,2	3,3	0,2	1,3	**1,3**	**3,3**	3,2
1+1.414	1+2.000	1+2.236	1+2.236	1+2.236	**1+2.236**	**1+2.236**	2+1.414
1,2	3,2	1,2	2,3	**2,4**	2,3		
2+1.414	2+1.414	2+1.414	2+2.000	**1+3.000**	2+2.000		

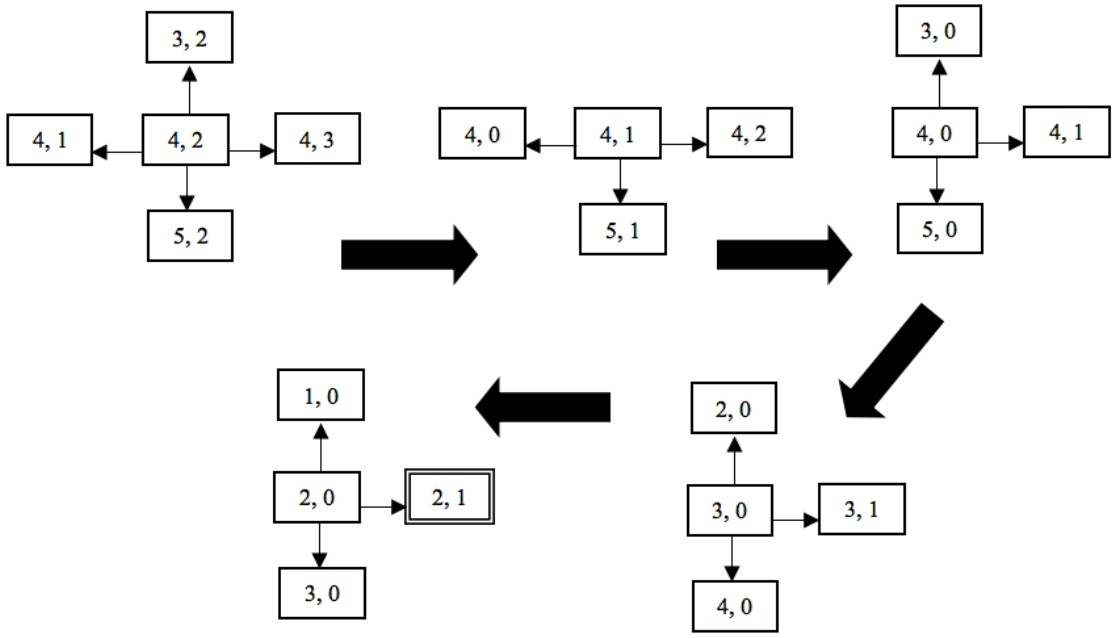
Iteration 7)



Fringe

4,2	3,3	0,2	1,3	1,3	3,3	3,2	**3,2**
1+2.000	1+2.236	1+2.236	1+2.236	1+2.236	1+2.236	2+1.414	**2+1.414**
1,2	3,2	1,2	2,3	**2,3**	2,4	2,3	**2,3**
2+1.414	2+1.414	2+1.414	2+2.000	**2+2.000**	1+3.000	2+2.000	**2+2.000**

Iteration 8 and onwards)



Analysis:

As shown above, greedy search fails to even reach the destination node. Let gets inside an infinite loop. While, A* search algorithm still finds the route to the destination node. Thus,

Greedy search always performs worse than or the same as A*, depending on the start and end states.

Overall:

- As shown both in figure 5 and 6, we can say that both greedy search and A* search algorithms perform based on start and end states. So we generally can say that, **Greedy search performs sometimes better, sometimes worse, and sometimes the same as A*, depending on the start and end states.**

References:

https://en.wikipedia.org/wiki/Admissible_heuristic