[illegible]

HW1

1. Code is written in Python language

   Please see attached python code

2. Expansion of Nodes:

   a. Breadth-First Search

      This algorithm will start from source node put that in a queue and will explore all the neighbours of that element and put the unexplored neighbours of queue and will do this until goal is visited.

      A -> (**B**, D, E) -> C -> (F, G)

      Red denotes which node is explored from the current neighbours

   b. Depth-First search

      This algorithm will start from a node explore it to the last step and then start back tracking if any neighbour node is unexplored and will do this until goal is found.

      A -> B -> C -> E -> D -> G

   c. Iterative Deepening search

Depth	Nodes explored
0	A
1	A B D E
2	A B C D G

   d. Uniform cost search

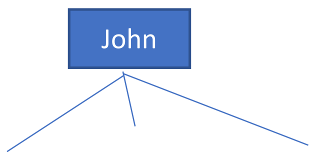
      Uniform-Cost Search is similar to Dijkstra's algorithm. In this algorithm from the starting state we will visit the adjacent states and will choose the least costly state then we will choose the next least costly state from the all un-visited and adjacent states of the visited states, in this way we will try to reach the goal state.

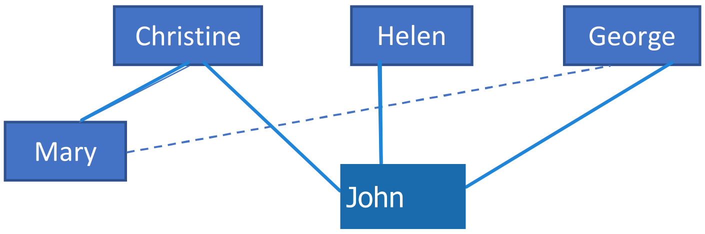
      A - > D -> E -> B -> C -> F -> G

3. Social Network Graph

   a. Algorithms able to find correct number of degrees of separation: Breadth First and Uniform cost search

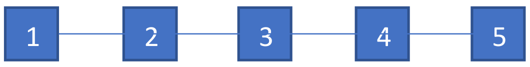
   b.



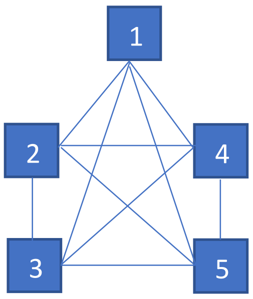


c. No there is no as such relation between nodes in search tree and vertices of graph as this might be the case that there are some vertices in graph which can not be in search tree (if degree of separation is infinite between two person, two disconnected graphs) but it is sure that if there is a node in search tree then it must be a vertex of the graph.

d.



e.



f. This can be achieved by making a flag variable corresponding to each vertex of graph which denotes whether that vertex is discovered or not and if it is then ignore that vertex thus in tree we will have nodes same as number of vertices.

4. A heuristic is admissible if the actual distance from that node is greater than or equal to the heuristic value.
   a. Heuristic 1
      i. $h_1(A) = 5$, actual distance = 60/65
      ii. $h_1(B) = 50$, actual distance = 45/50/80
      iii. $h_1(C) = 15$, actual distance = 15
      iv. $h_1(D) = 0$, actual distance = 0
      v. $h_1(E) = 10$, actual distance = 55
      vi. $h_1(F) = 0$, actual distance = 35/60/95

      Satisfy inequality for all possible paths of all nodes so admissible

   b. Heuristic 2
      $h_2(D)$ must be zero so not admissible

c. Heuristic 3

h_3(D) must be zero so not admissible

d. Heuristic 4

Satisfy inequality for all nodes so admissible

5. The best admissible heuristic is:

h(city) = 2 // as you can reach city from suburb, or a city

h(suburb) = 1 // as you can reach a suburb from suburb

h(village) = 4 // as you can reach a village from a city, village, suburb, or a farm

h(farm) = 4 // as you can reach a farm from a city, village, suburb, or a farm

h(mountain) = 5 // as you can reach a mountain from all 5 possible

6. A greedy method always chooses current best choice to find a solution. But it does not consider how optimal the solution is. At every step it uses a heuristic to find the current best path. In the given example of matrix, the heuristic function for each cell is the Manhattan distance of that node to goal position.

On the other hand A* algorithm check whether the current solution is optimal or not because it not only consider distance of the current node to the goal position (h(n)) but it also take care of the accumulated distance taken to reach that node (g(n)) and take a action which minimizes h(n) + g(n).

Thus A* will find optimal solution for a given graph. And in greedy algorithm it may also be the case for a path with obstacles it will go into a position from where it is difficult to reach goal. So A* is complete and optimal. But in some cases greedy and A* can perform same as well.

For example in fig 5 both A* and greedy will visit 16 nodes to reach goal. So performance is same.

So saying A* is always optimal is not completely true.

Therefore, sometime A* is better than greedy on the other scenario greedy is better than A* Or both are same performance

In figure 6 also we have a shortest path which can be covered just by visiting 16 nodes so performance would be same.