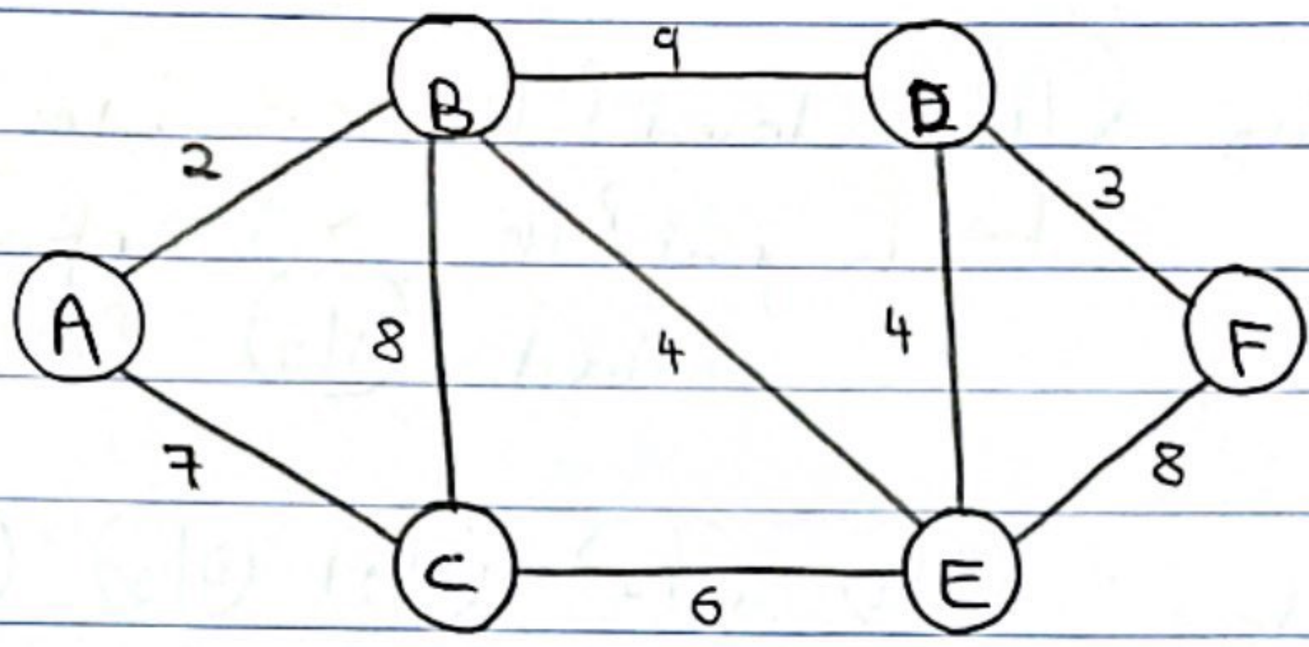
Prem A. Jethwa

1001861810

Task 1 -



* Breadth First Search : (BFS)

Initial Fringe : (A|0)

Closed Set : { }

[Node represented as (node|depth)]

1. Pop (A|0) → closed ? No → ∴ add to closed.
   → is goal ? No → ∴ expand.
   Add (B|1) and (C|1) to fringe.

Fringe : (B|1) (C|1) closed : { (A|0) }

2. Pop (B|1) → closed ? No → ∴ add to closed
   → is goal ? No → ∴ expand. Add (A|2) (C|2) (E|2) (D|2)

Fringe : (C|1) (A|2) (C|2) (E|2) (D|2)

closed : { (A|0), (B|1) }

3. Pop (C|1) → closed? No → ∴ add to closed.
→ is goal? No → ∴ expand.
∴ Add (A|2) (B|2) (E|2)

Fringe: (A|2) (C|2) (E|2) (D|2) (A|2) (B|2) (E|2)

closed: { (A|0), (B|1), (C|1) }

4. Pop (A|2) → closed? Yes → ∴ ignore & go to next state

5. Pop (C|2) → closed? Yes → ∴ ignore & go to next state

Fringe: (E|2) (D|2) (A|2) (B|2) (E|2)

closed: { (A|0), (B|1), (C|1) }

6. Pop (E|2) → closed? No → ∴ Add to closed
→ is goal? No → ∴ expand
Add (F|3) (D|3) (B|3) (C|3)

Fringe: (D|2) (A|2) (B|2) (E|2) (F|3) (D|3) (B|3) (C|3)

closed: { (A|0), (B|1), (C|1), (E|2) }

7. Pop (D|2) → closed? No → ∴ Add to closed
→ is goal? No → ∴ expand.
Add (B|3) (E|3) (F|3)

Fringe: (A|2) (B|2) (E|2) (F|3) (D|3) (B|3) (C|3) (B|3) (E|3) (F|3)

closed: { (A|0), (B|1), (C|1), (E|2), (D|2) }

8. Pop (A|2)

9. Pop (B|2) → closed? Yes → ∴ ignore & go to next

10. Pop (E|2) Fringe: (F|3) (D|3) (B|3) (C|3) (B|3) (E|3) (F|3)

11. Pop (F|3) → closed? No → ∴ Add to closed
→ is goal? Yes → ∴ stop & return solution

Path : A → B
B → E
E → F

* Depth First Search: (DFS)

Initial Fringe : (A|0) closed : { }

1. Pop (A|0) → closed? No → ∴ Add to closed
   → is goal? No → ∴ Expand (B|1) & (C|1) to fringe.

Fringe : (B|1) (C|1) closed set : { (A|0) }

2. Choosing deepest node i.e. node with highest depth. Here depth of B = depth of C
   ∴ Choosing randomly.

Pop (B|1) → closed? No → ∴ Add to closed
→ Goal? No → ∴ Expand
Add (D|2) (E|2) (C|2) (A|2)

Fringe : (C|1) (D|2) (E|2) (C|2) (A|2) closed : {(A|0), (B|1)}

3. Highest depth in fringe = 2
   ∴ Choosing any node with depth = 2.

Pop (D|2) → closed? No → ∴ Add to closed
→ Goal? No → ∴ Expand
Add (F|3) (E|3) (B|3)

Fringe: (C|1) (E|2) (C|2) (A|2) (F|3) (E|3) (B|3)

Closed set : { (A|0) , (B|1) , (D|2) }

4. Highest depth = 3 ∴ Choosing (F|3) to expand.

Pop (F|3) → closed ? No → ∴ add to closed
→ Goal ? (Yes) → ∴ Stop & return solution.

Path :	A → B
	B → D
	D → F

* Uniform Cost Search - (UCS) -

Initial Fringe : (A|0) closed set : { }

[Node represented as (node|cost)]

1. Pop (A|0) → closed ? No → ∴ add to closed
→ Goal ? No → ∴ expand.

Add (B|2) (C|7) closed { (A|0) }

Fringe (B|2) (C|7)

2. Pop (B|2) → closed? No → ∴ add to closed
→ Goal? No → ∴ expand
Add (E|6) (C|10) (D|11)

Fringe: (C|7) (E|6) (C|10) (D|11) closed: {(A|0), (B|2)}

3. Pop (E|6) → closed? No → ∴ add to closed
→ Goal? No → ∴ expand
Add (B|10) (D|10) (C|12) (F|14)

Fringe: (C|7) (C|10) (C|12) (B|10) (D|10) (D|11) (F|14)

closed set: {(A|0), (B|2), (E|6)}

4. Pop (C|7) → closed? No → ∴ add to closed
→ Goal? No → ∴ expand
Add (A|14) (B|15) (E|13)

Fringe: (C|10) (C|12) (B|10) (B|15) (A|14) (E|13) (D|10) (D|11) (F|14)

closed set: {(A|0), (B|2), (E|6), (C|7)}

5.

5. Pop C|10
6. Pop C|12
7. Pop B|10
8. Pop B|15
9. Pop A|14
10. Pop E|13

} closed? Yes → ∴ ignore & go to next

Fringe: D|10 D|11 F|14

11. Pop D|10 → closed? No → ∴ add to closed
→ goal? No → expand
Add B|19 E|14 F|13

Fringe: D|11 B|19 E|14 F|13 F|14

closed state: {A|0, B|2, E|6, C|7, D|10}

12. Pop D|11
13. Pop B|19
14. Pop E|14

} closed? Yes → ∴ ignore & go to next

Fringe: F|13 F|14

15. Pop F|13 → closed? No → ∴ add to closed
→ Goal? Yes → Stop & return solution.

Path: A → B → E → D → F

* Iterative Deepening Search - (IDS)-

Iteration 1: Depth limit = 0

Initial Fringe: (A|0) closed: { }

1. Pop (A|0) → closed ? No → ∴ add to closed
   → Goal ? No → ∴ expand
   (cannot expand as depth limit is reached)

∴ Start new iteration

Iteration 2: Depth limit = 1

Initial Fringe: (A|0) closed: { }

1. Pop (A|0) → closed ? No → ∴ add to closed
   → Goal ? No → ∴ expand
   Add (B|1) (C|1)

Fringe: (B|1) (C|1) closed: { (A|0) }

2. Choose Deepest node.
∴ Pop (B|1) → closed ? No → ∴ add to closed
   → Goal ? No → ∴ expand
(cannot expand as depth limit is reached)

Fringe : (C|1) ____ closed : { (A|0) , (B|1) }

3. Choose next deepest node.

Pop (C|1) → closed ? No → ∴ add to closed.
↳ Goal ? No → ∴ expand
(not possible as depth limit reached)

Fringe : ____ closed : { (A|0) , (B|1) , (C|1) }

∴ Start new iteration.

Iteration 3 : Depth limit = 2

Initial Fringe : (A|0) ____ closed : { }

1. Pop (A|0) → closed ? No → ∴ add to closed
↳ Goal ? No → ∴ expand.
Add (B|1) (C|1)

Fringe : (B|1) (C|1) ____ closed : { (A|0) }

2. Pop (B|1) → closed ? No → ∴ add to closed
↳ Goal ? No → ∴ expand
Add (D|2) (E|2) (C|2) (A|2)

Fringe : (C|1) (D|2) (E|2) (C|2) (A|2) closed : { (A|0), (B|1)

3. Pop (D|2) → closed? No → ∴ add to closed
→ Goal? No → ∴ expand
(Not possible as depth limit reached).

Fringe: (C|1) (E|2) (C|2) (A|2) closed: {(A|0), (B|1), (D|2)}

4. Pop (E|2) → closed? No → ∴ add to closed
→ Goal? No → ∴ expand
(Not possible as depth limit reached)

Fringe: (C|1) (C|2) (A|2) closed: {(A|0), (B|1), (D|2), (E|2)}

5. Pop (C|2) → closed? No → ∴ add to closed
→ Goal? No → ∴ expand
(Not possible as depth limit reached)

Fringe: (C|1) (A|2) closed: {(A|0), (B|1), (D|2), (E|2), (C|2)}

6. Pop (A|2) } closed? Yes → ∴ ignore & go to next
7. Pop (C|1) }

Fringe: _____ closed: {(A|0), (B|1), (D|2), (E|2), (C|2)}

Iteration 4: Depth limit = 3

Initial Fringe : (A|0) closed : { }

1. Pop (A|0) → closed? No → ∴ add to closed.
→ Goal? No → ∴ expand.
Add (B|1) (C|1)

Fringe : (B|1) (C|1) closed : { (A|0) }

2. Pop (B|1) → closed? No → ∴ add to closed
→ Goal? No → ∴ expand
Add (D|2) (E|2) (C|2) (A|2)

Fringe : (C|1) (D|2) (E|2) (C|2) (A|2) closed : { (A|0), (B|1) }

3. Pop (D|2) → closed? No → ∴ add to closed
→ Goal? No → ∴ expand
Add (F|3) (E|3) (B|3)

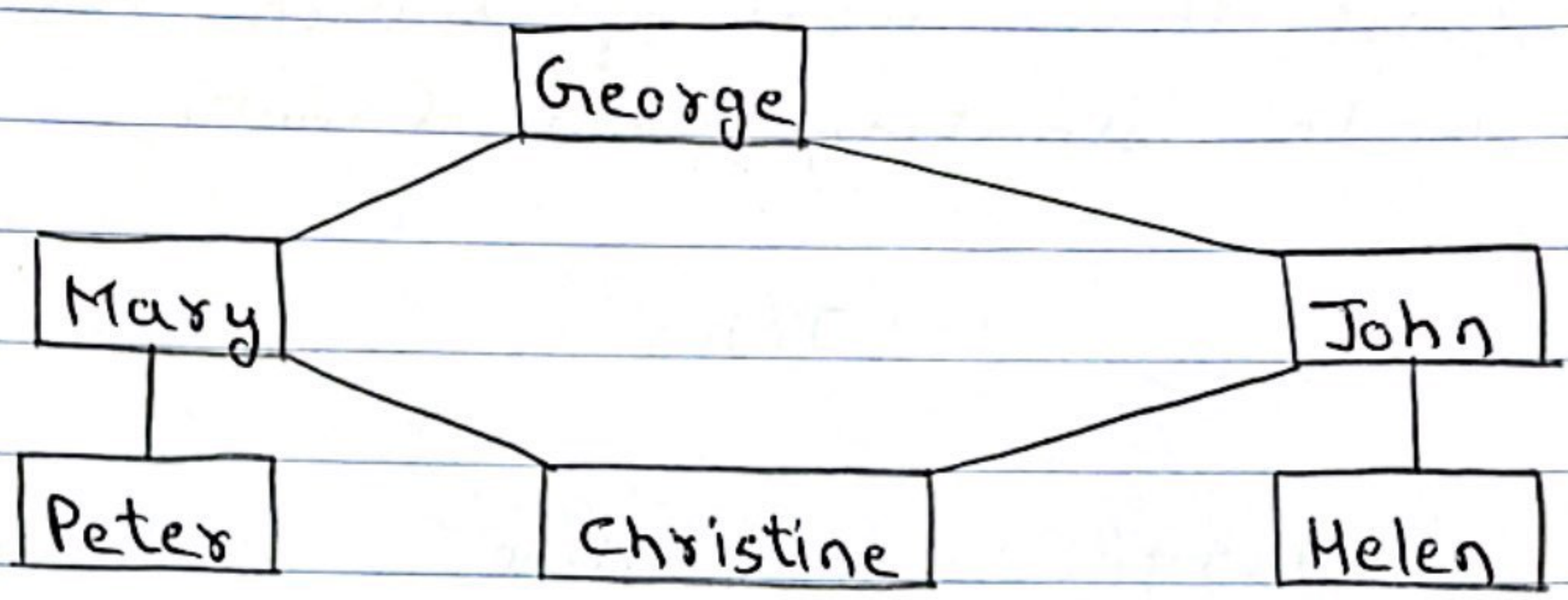
Fringe : (C|1) (E|2) (C|2) (A|2) (F|3) (E|3) (B|3)

closed : { (A|0), (B|1), (D|2) }

4. Pop (F|3) → closed? No → ∴ add to closed
   → Goal? (Yes) → ∴ Stop & return to solution.

Path :	A → B
	B → D
	D → F

Task 2 -

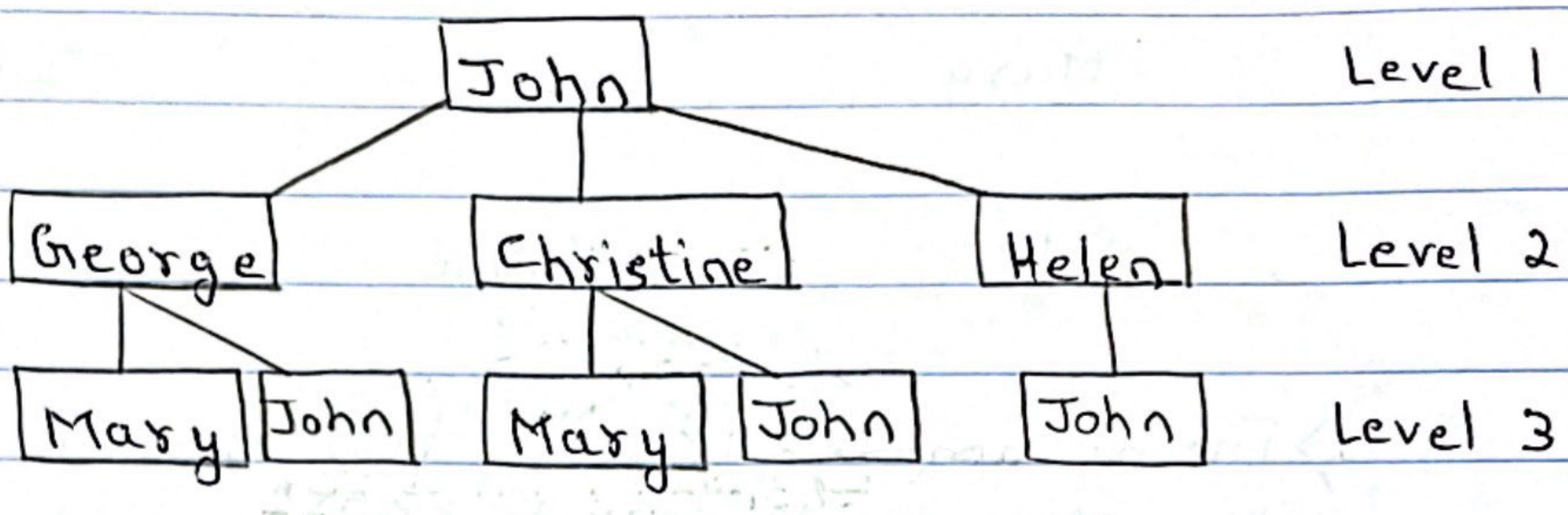


I) From among general tree search using breadth - first - search (BFS), depth - first - search (DFS), iterative deepining search (IDS), and uniform cost search (UCS).
Though DFS takes less memory to find a solution, there are possibilities where DFS not giving a solution.
On the other hand, BFS, IDS, UCS guarantee finding the correct degree of separation between the people in the graph considering all actions have the same cost.

BFS, IDS, UCS - Yes, find the correct degree of separation.
IDS - will also find shallowest (optimal) path.
DFS - No, cannot find the correct degree of separation.
DFS - May never terminate search.

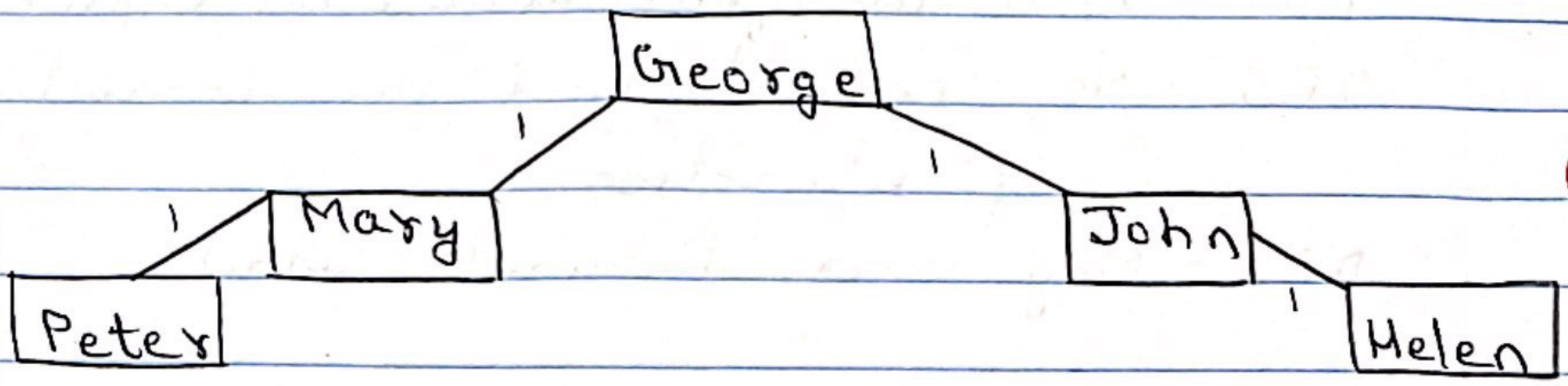
Task 2 -

II) First three level of search tree with John as the starting point. (root).



III) There is no one-to-one correspondance between the nodes of above search tree and vertices in SNG, because the vertex 'John' in SNG corresponds to multiple nodes in the tree. And vertex Peter which is present in SNG is not present in the above search tree. So, the algorithm could revisit same table/state more then once.

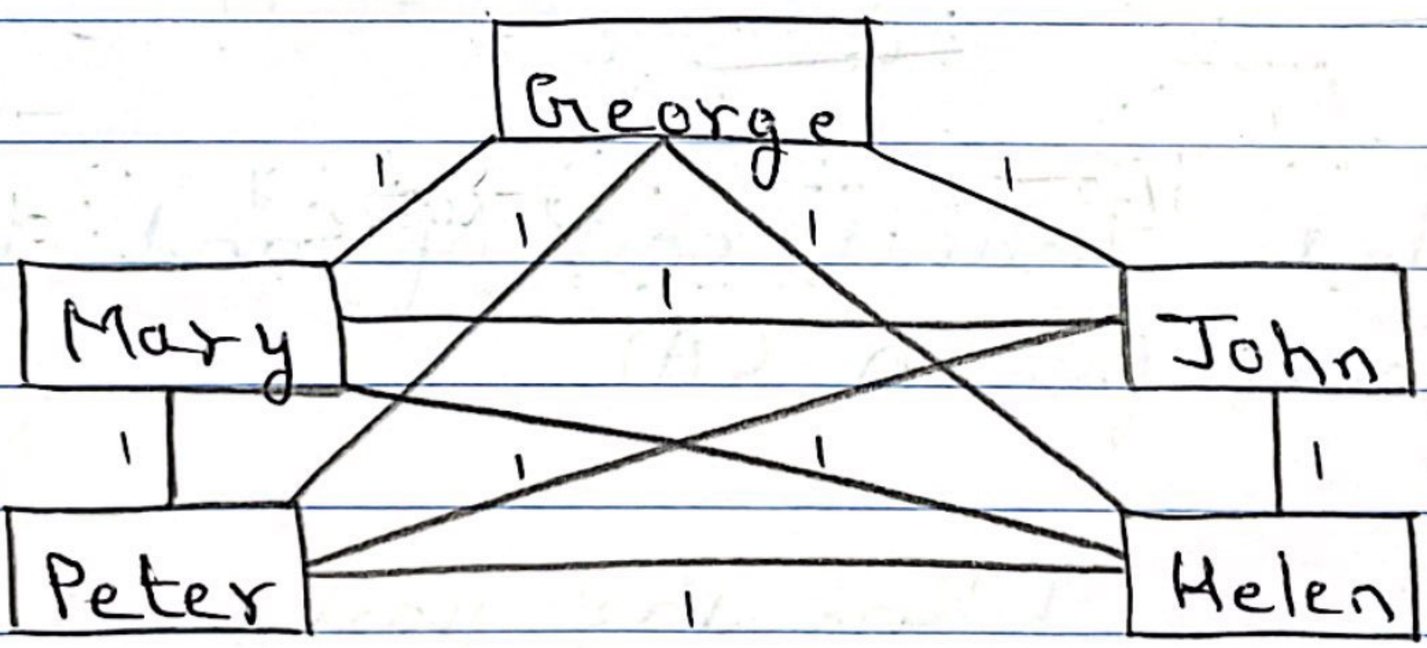
IV) 5 people, where atleast 2 have 4 degree of separation between them.



Task 2-

IV) In the constructed SNG, Peter and Helen have 4 degree of separation.

V) 5 people, where all have 1 degree of separation between them.



In the above SNG, all people have 1 degree of seperation between them.

Task 2 -

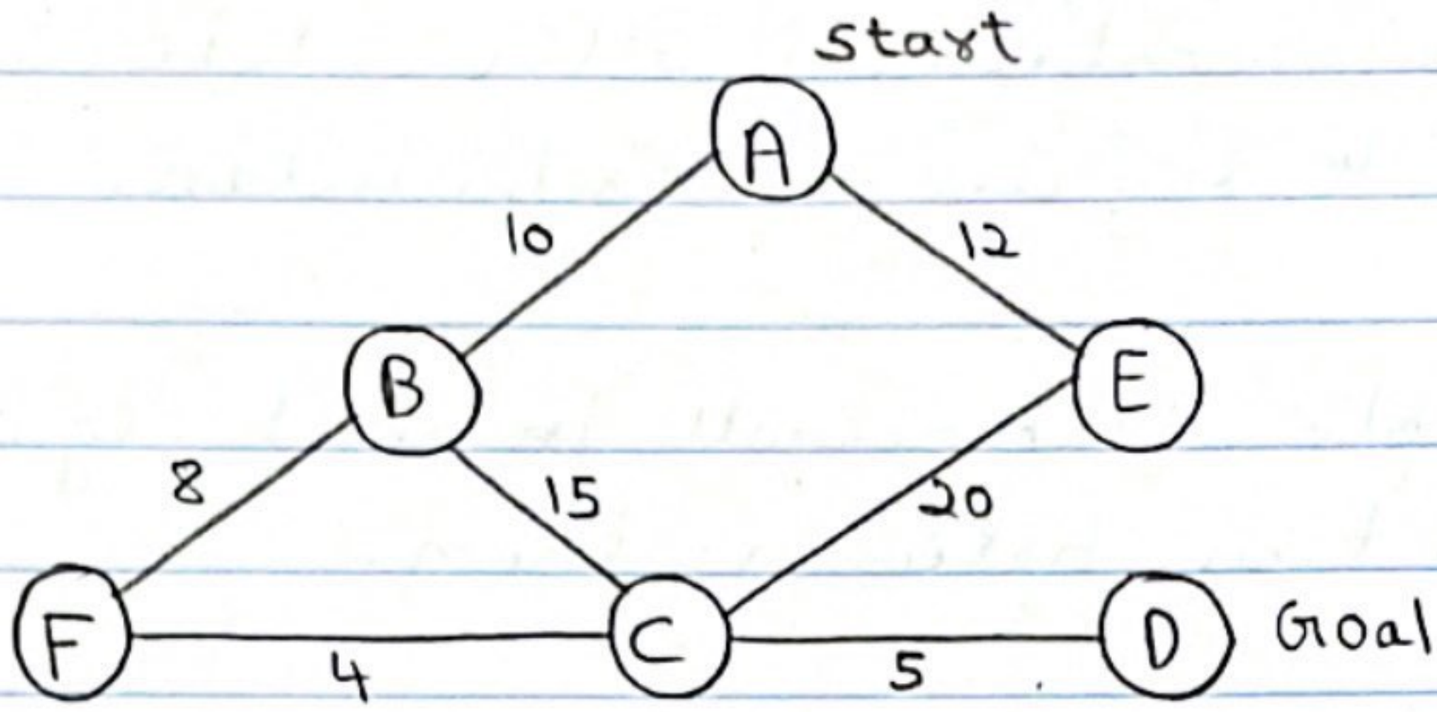
VI) In such a case, we can implement the Breadth First using Graph Search.

Basically,

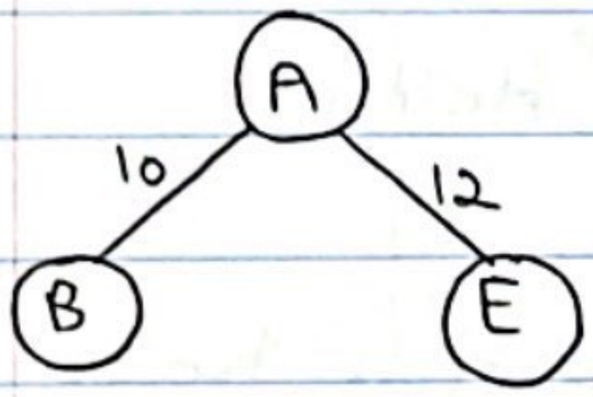
a. We need to keep track of the nodes that are already visited by listing them as closed (Since further expansion of these nodes will only cause generation of multiple nodes for the same person).

b. And before expansion of any node, we should check if that node belongs to a closed list or not. If it belongs then ignore it and go to the next node or else expand and add the new nodes to the fringe.

Task 3 -



Calculate actual cost of node from start to goal. for $h^*(A)$



Taking the least cost we traverse from A $\xrightarrow{10}$ B and will continue further on same approach.

$\therefore$ $h^*(A) = 27$ $(A \xrightarrow{10} B \xrightarrow{8} F \xrightarrow{4} C \xrightarrow{5} D)$

$h^*(B) = 17$ $(B \xrightarrow{8} F \xrightarrow{4} C \xrightarrow{5} D)$

$h^*(C) = 5$ $(C \xrightarrow{5} D)$

$h^*(D) = 0$ —— (Goal state)

$h^*(E) = 25$ —— $(E \xrightarrow{20} C \xrightarrow{5} D)$

$h^*(F) = 9$ $(F \xrightarrow{4} C \xrightarrow{5} D)$

Task 3 -

A heuristic to be admissible should be

$[h(n) \leq h^*(n)]$

Heuristic 1 :

$h(A) = 5$	$h^*(A) = 27 \rightarrow$ admissible	if
$h(B) = 20$	$h^*(B) = 17 \rightarrow$ not admissible	$h(n) \leq h^*(n)$
$h(C) = 15$	$h^*(C) = 5 \rightarrow$ not admissible	admissible
$h(D) = 0$	$h^*(D) = 0 \rightarrow$ admissible	else
$h(E) = 10$	$h^*(E) = 25 \rightarrow$ admissible	not admissible.
$h(F) = 0$	$h^*(F) = 9 \rightarrow$ admissible	

Now, new admissible heuristics 1 value;

Heuristic 1 :

$h(A) = 5$

$h(B) = 17$

$h(C) = 5$

$h(D) = 0$

$h(E) = 10$

$h(F) = 0$

Task 3-

Heuristic 2 :

$h(A) = 45$	$h^*(A) = 27$	Not admissible
$h(B) = 45$	$h^*(B) = 17$	
$h(C) = 45$	$h^*(C) = 5$	
$h(D) = 45$	$h^*(D) = 0$	
$h(E) = 45$	$h^*(E) = 25$	
$h(F) = 45$	$h^*(F) = 9$	

Now, new admissible heuristic 2 values:

$h(A) = 27$

$h(B) = 17$

$h(C) = 5$

$h(D) = 0$

$h(G) = 25$

$h(F) = 9$

Heuristic 3 :

$h(A) = 10$	$h^*(A) = 27$	admissible
$h(B) = 15$	$h^*(B) = 17$	
$h(C) = 0$	$h^*(C) = 5$	
$h(D) = 0$	$h^*(D) = 0$	
$h(E) = 25$	$h^*(E) = 25$	
$h(F) = 5$	$h^*(F) = 9$	

Task 3 -

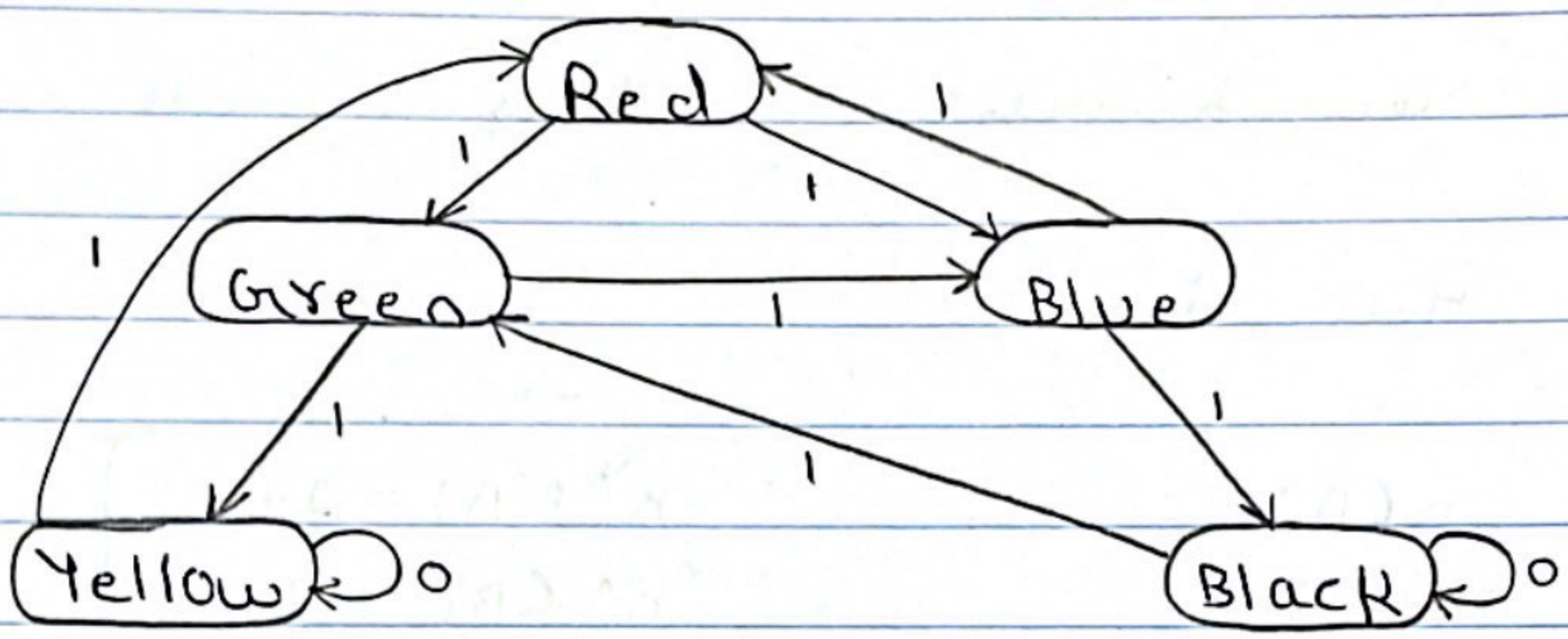
New heuristic 3 values are all same.

Heuristic 4 :

$h(A) = 0$	$h^*(A) = 27$	admissible
$h(B) = 0$	$h^*(B) = 17$	
$h(C) = 0$	$h^*(C) = 5$	
$h(D) = 0$	$h^*(D) = 0$	
$h(E) = 0$	$h^*(E) = 25$	
$h(F) = 0$	$h^*(F) = 9$	

New heuristic 4 values are all same.

Task 4 -



Given - Goal → Black

Cost of one state to another is 1

Cost of one state to itself is 0.

Goal - To reach 'Black' state.

h^* (Red) = Red → Blue → Black = 1 + 1 = 2

h^* (Green) = Green → Blue → Black = 1 + 1 = 2

h^* (Blue) = Blue → Black = 1

h^* (Yellow) = Yellow → Red → Blue → Black = 1 + 1 + 1 = 3

h^* (Black) = 0

∴ Maximally admissible Heuristic ⟹

h^* (Red) = 2

h^* (Green) = 2

h^* (Blue) = 1

h^* (Yellow) = 3

h^* (Black) = 0

Task 5 -

Here the greedy search algorithm is represented as h(n) which is Euclidean Distance

A* algorithm is represented as f(n)

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

where,

h(n) → Euclidean distance

g(n) → minimum distance between nodes.

* Figure 4 -

We assume that it starts from (5,0) and ends on (5,3).

∴ Heuristic values of following items:

h(5,0) = 6.9	h(5,1) = 6.6	h(5,2) = 6.3
h(4,0) = 6.8	h(4,1) = 6.5	h(4,2) = 6.2
h(6,0) = 6.7	h(6,1) = 6.4	h(6,2) = 6.1
		h(5,3) = 0

greedy algorithm - start node (5,0) it has 3 ways from which (5,1) is the shortest value, where (5,1) = 6.6.

Now, it has 4 options & it will select (5,2) or its shortest, h(5,2) = 6.3

Task 5 -

Similarly, now it will select the shortest path having the shortest value, where $h(5,3) = 0$. which is goal node.

A* algorithm - It works similarly to greedy algorithm and also it will follow the same path & result will also be same as greedy algorithm.

Here, we can say that greedy search algorithm performs better or same as A* algorithm, depending on start state and end state locations.

Task 5 -

* Figure 6 -

Consider the start node (2,6) and
end node (5,8).
Here, greedy is better than A* algorithm.

Consider the start node (2,0) and
end node (2,2).
Here, greedy is worse than A* algorithm.

Consider the start node (2,2) and
end node (2,7).
Here, both greedy search and A* algorithm
performance the same.

Conclusion,
So, greedy will perform better than, the
same as or worse than A* depending
on start state and end state location.

∴ greedy search performs sometimes better,
sometimes worse and sometimes the same
as A*, depending on the start & end
state ; is true.