Vidya Pratishthan's

Kamalnayan Bajaj Institute of Engineering and Technology

Vidyanagari, Baramati, Pune 413 133

INDEX

Department of :- Artificial Intelligence & Data Science

CLASS T.E

SR. NO.	NAME OF EXPERIMENT	Expt. Conducted on	Expt. Checked on	PAGE NO.	SIGN	REMARK
1.	Implement Solution for a Constraint Satisfaction problem (TSP & N-queen) using Branch & Bound & Backtrack	2/8/22	14/11/22	1		
2.	Implement BFS & DFS Algorithm.	23/8/22	14/11/22	7		
3.	Implement Greedy Search Algorithm For MST.	30/8/22	14/11/22	12		
4.	Implement A* algorithm for 8 puzzle problem.	18/9/22	14/11/22	18		
5.	Implement Alpha-Beta search algorithm to optimize min-max algorithm	18/10/22	14/11/2222			

CERTIFICATE

This is to certify that Mr./Miss Devrat Yashraj Deepak of Class T.E (A.I.D.S) Roll No. 228703 has Satisfactorily Completed the term work of the subject Software Laboratory I for V Semester of 2022-2023 (Artificial Intelligence)

Date : 12 / 11 / 22

Staff Member / In-charge

H.O.D.

PRINCIPAL

Vidya Pratishthan's

Kamalnayan Bajaj Institute of Engineering and Technology

Vidyanagari, Baramati, Pune 413 133

INDEX

Department of :- Artificial Intelligence & Data Science

CLASS T.E

SR. NO.	NAME OF EXPERIMENT	Expt. Conducted on	Expt. Checked on	PAGE NO.	SIGN	REMARK
6	Mini Project	12/11/22	14/11/22	-		

CERTIFICATE

This is to certify that Mr./Miss Pernat Yashraj Deepak of Class T.E(A.I.D.S) Roll No.22203) has Satisfactorily Completed the term work of the subject, Software laboratory I for V Semester of 2022-2023 (Artificial Intelligence)

Date : 12/11/22

Staff Member

In-charge

H.O.D.

PRINCIPAL

Assignment No. 1

Aim: Implement a solution for a Constraint satisfaction problem using Branch and Bound and Backtracking.

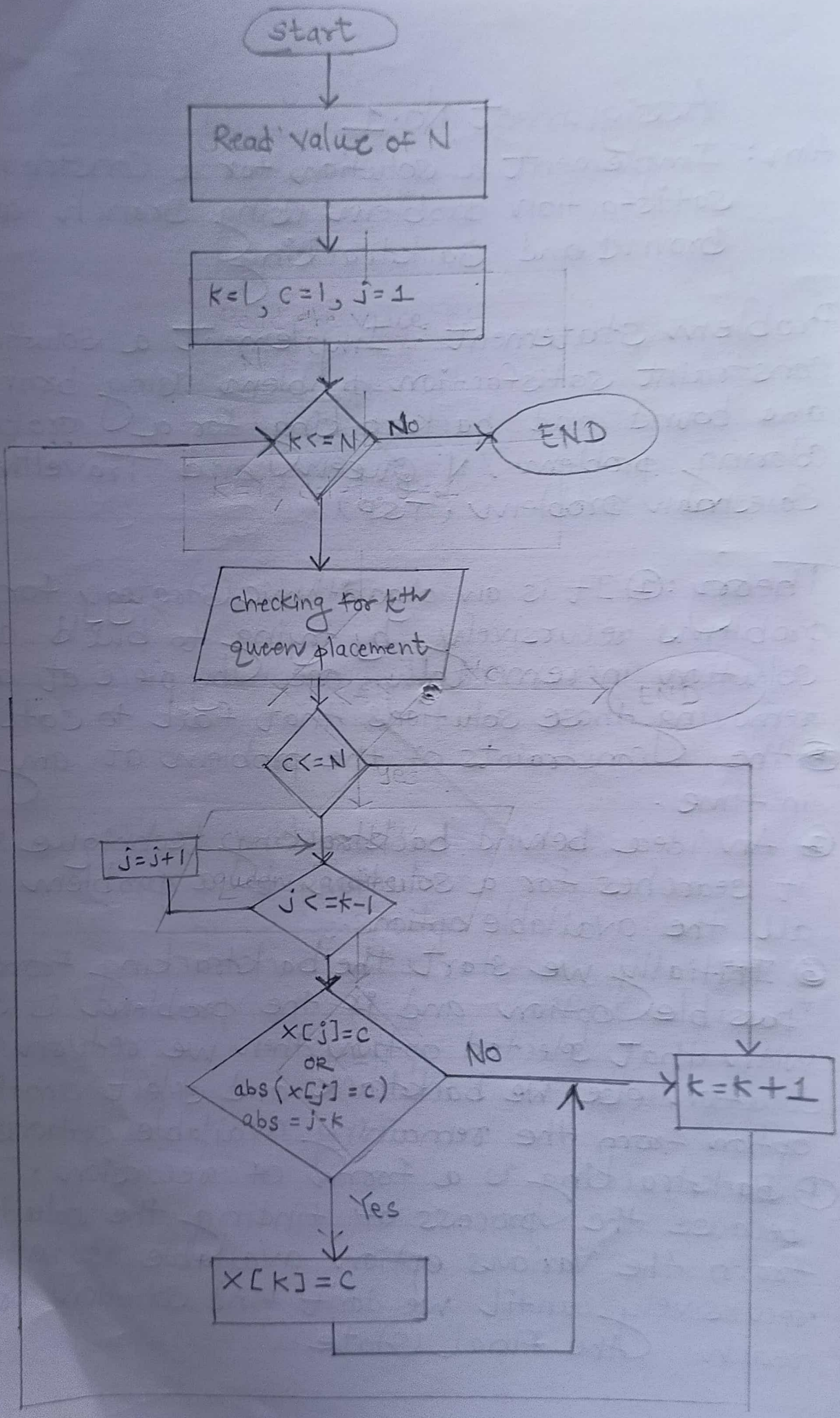
Problem statement: Implement a solution for Constraint satisfaction problem using branch and bound and backtracking for a Graph coloring problem / N queens and Travelling Salesman problem (TSP).

Theory: ① It is an algorithmic strategy for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time.

② An idea behind backtracking technique is that it searches for a solution to a problem among all the available options.

③ Initially we start the backtracking from one possible option and if the problem is solved with that selected option then we return the solution else we backtrack and select another option from the remaining available options.

④ Backtracking is a form of recursion. This is because the process of finding the solution from the various option available is repeated recursively until we don't find solution or we reach the final state.


Start
Read value of N
k=1, c=1, j=1
k<=N
No
END
checking for kth queen placement
c<=N
j=j+1
j<=k-1
x[j]=c
OR
abs(x[j]=c)
abs = j-k
No
k=k+1
Yes
x[k]=c

We can conclude that backtracking at every step eliminates those choices that cannot give us the solution and proceeds to those choices that have the potential of taking us to the solution.

Algorithm for Backtracking:

Step ①: Start from the leftmost column.

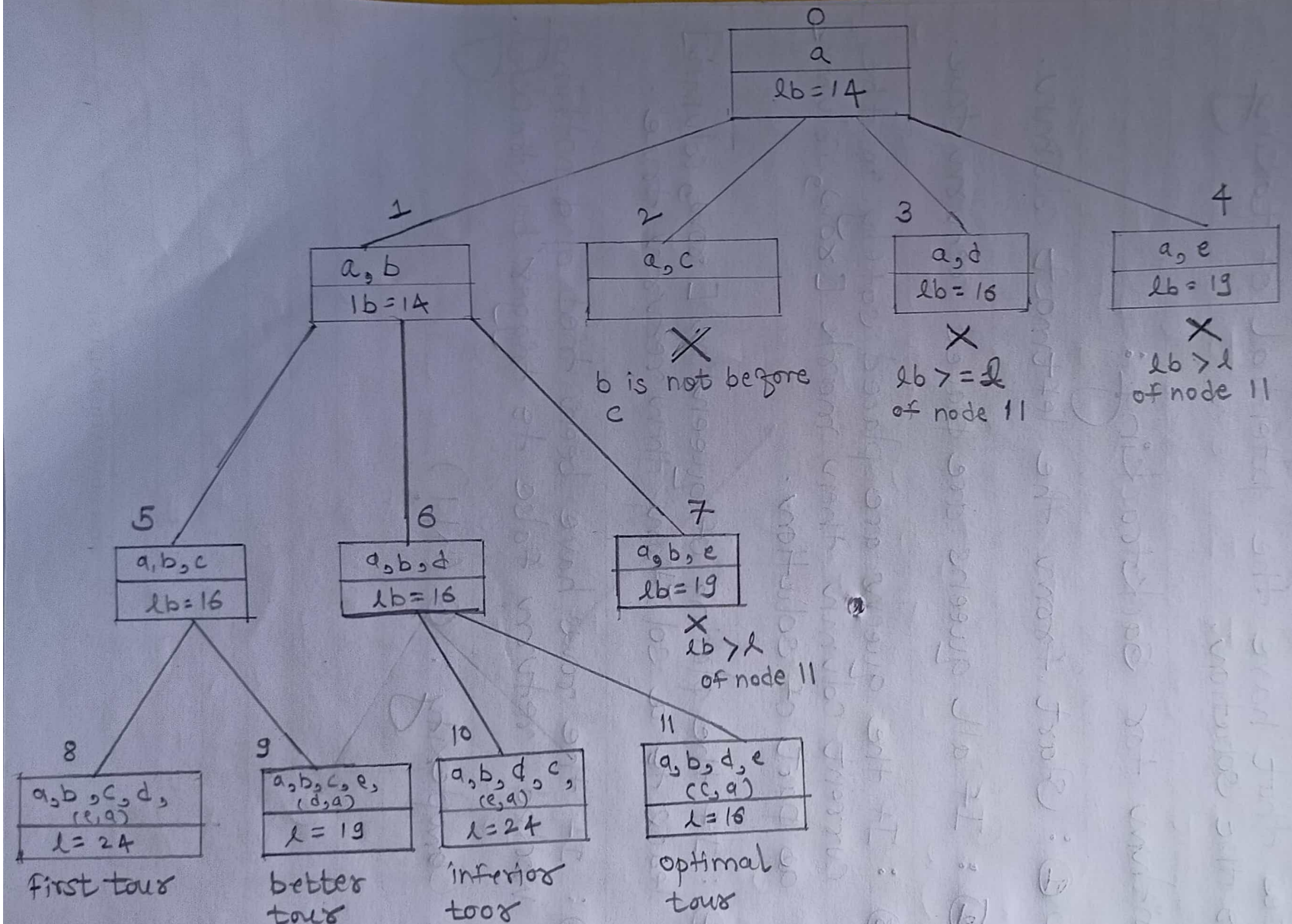
Step ②: If all queens are placed return true.

Step ③: If the queens are placed safely in the current column then mark [row, column] as part of solution.

Step ④: If the placing of queen in [row, column] leads to a solution then return true.

Step ⑤: If the rows have been tried and nothing worked, return false to trigger backtracking.

Time Complexity: O(N!)


0
a
lb = 14
1
a, b
lb = 14
2
a, c
b is not before c
3
a, d
lb = 16
lb >= l of node 11
4
a, e
lb = 19
lb > l of node 11
5
a, b, c
lb = 16
6
a, b, d
lb = 16
7
a, b, e
lb = 19
lb > l of node 11
8
a, b, c, d, (e, a)
l = 24
first tour
9
a, b, c, e, (d, a)
l = 19
better tour
10
a, b, d, c, (e, a)
l = 24
inferior toor
11
a, b, d, e (c, a)
l = 16
optimal tour

Branch and Bound:

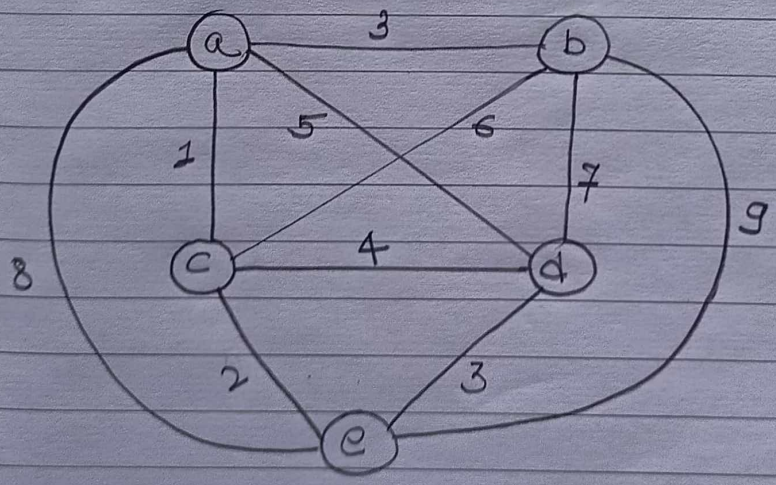
① Branch and Bound is an algorithm design paradigm which is generally used for solving Combinatorial optimization problems.

② Branch and Bound Algorithm technique solves these problems relatively quickly.

③ The algorithm explores branches of this tree, which represent subsets of the solution set.

④ The algorithm depends on efficients estimation of the lower and upper bounds of branches of search space. If no. of bounds are available, the algorithm generates to an exhaustive search.

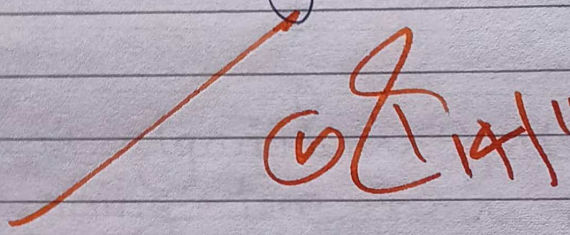
Solve the following Travelling Salesman Problem using branch & bound method.



Page No.
Date

Time Complexity for Travelling Salesman Problem : $O(n^2 2^n)$.

Conclusion: From this programme we about the implementation of backtracking and Branch and Bound problem solving methods.



Assignment No. 2

Aim: Implement Greedy Search algorithm for any of the minimum spanning tree (MST) Algorithm.

Problem Statement: Implement Greedy Search algorithm for Prim's Minimal Spanning tree Algorithm.

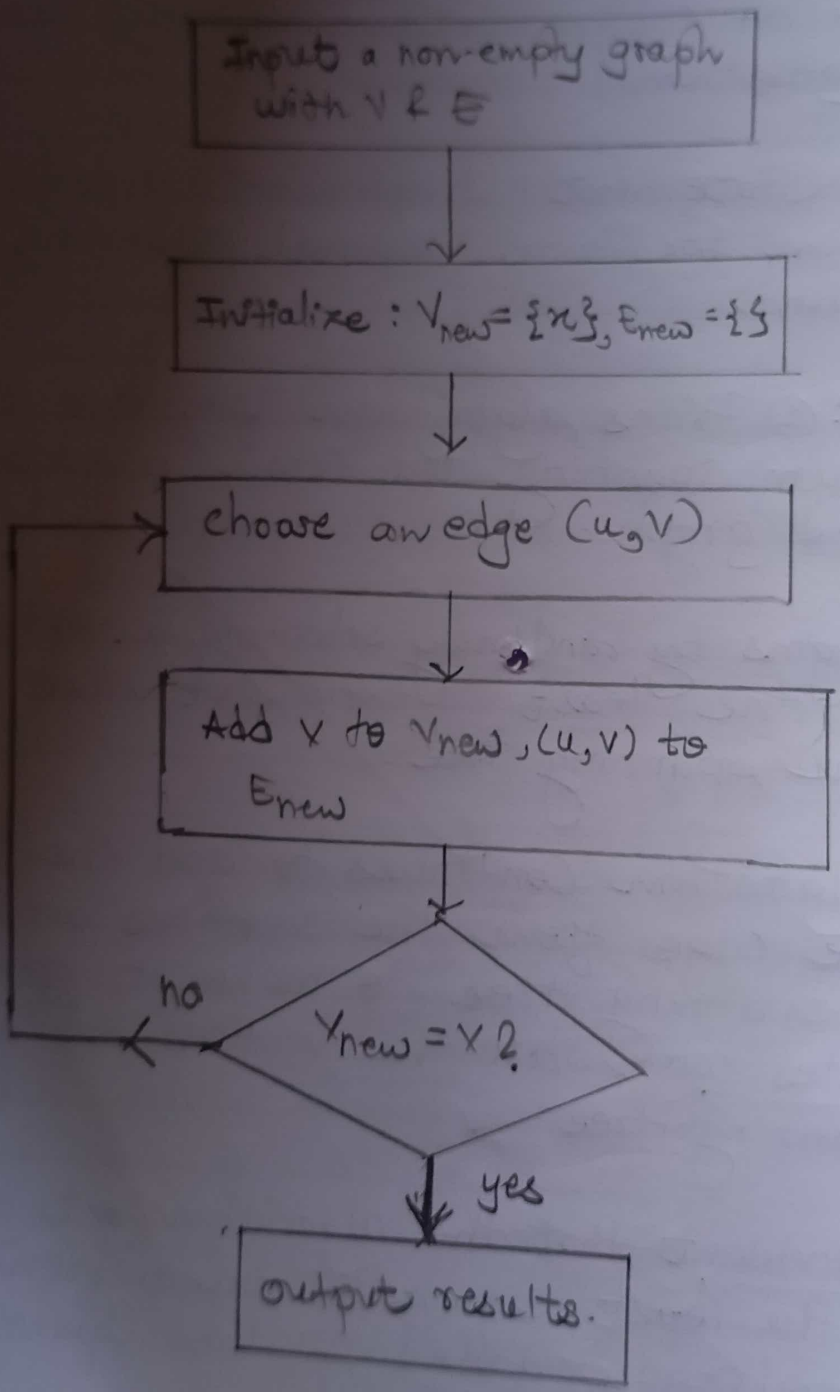
Theory: ① Prim's algorithm finds the cost of a minimum spanning tree from a weighted undirected graph.

② It begins by randomly selecting a vertex and adding the least expensive edge this vertex to the spanning tree.

③ The algorithm continues to add the least expensive edge from the vertices already added to the spanning tree to make it grow and terminates when all the vertices are added to the spanning tree.

④ It is evident that the algorithm get greedy by selecting the least expensive edge from the vertices already added to the spanning tree.

⑤ Greedy Algorithm is an approach to solve optimization problems (such as minimizing and maximizing a certain quantity) by making



Prim's Algorithm flowchart

locally optimal choices at each step which may then yield a globally optimal solution.

Algorithm:

Prims minimum spanning tree (Graph G, source node S):

Step (1): Create a dictionary (to be used as a priority queue). Pq to hold pairs of (node, cost).

Step (2): Push [S, 0] (node, cost) in the dictionary Pq i.e Cost of reaching vertex is from source node S is zero.

Step (3): While Pq contains (V, C) pairs:

Step (4): Get the adjacent node V (key) from the dictionary Pq.

Step (5): Cost C = Pq [V].

Step (6): Delete the key-value pair (V, C) from the dictionary Pq.

Step (7): If the adjacent node V is not added to the spanning tree.

Step (8): Add node V to the spanning tree.

Step (9): Cost of the spanning tree += Cost C.

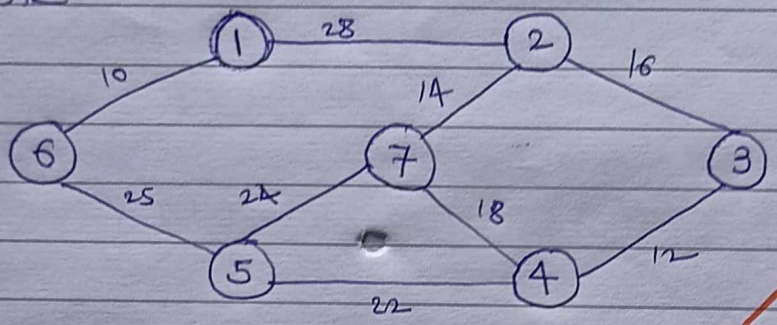
Step (10): For all vertices adjacent to vertex V not added to the spanning tree.

Step (11): Push pair of (adjacent node, cost) into the dictionary Pq.

Prims Algorithm Time Complexity:
Worst case time complexity of prim's algorithm is:

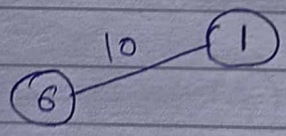
- O (E log V) using binary heap.
- O (E + V log V) using Fibonacci heap.

Example:

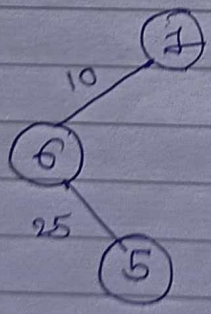


The above discussed steps are followed to find the minimum cost spanning tree using prims algorithm.

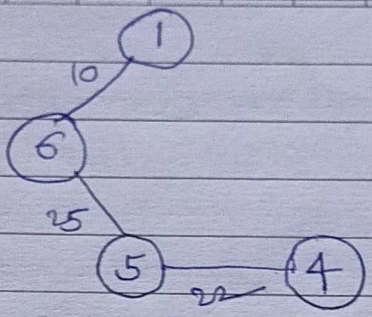
Step (1)



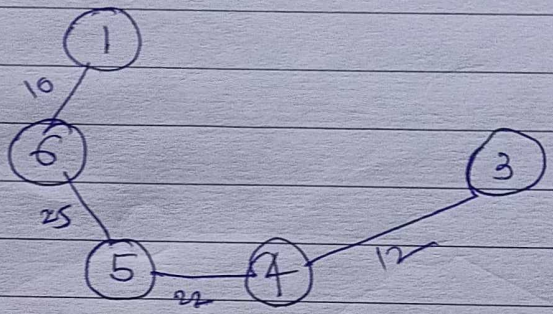
Step (2)



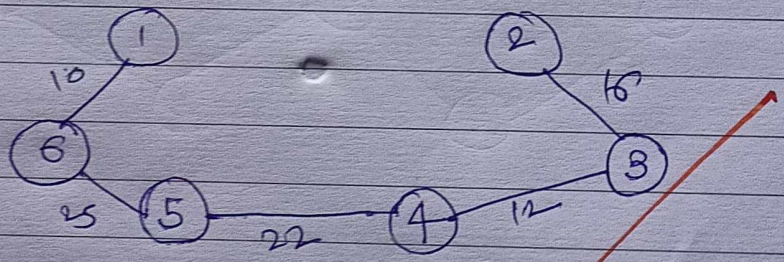
Step ③ :



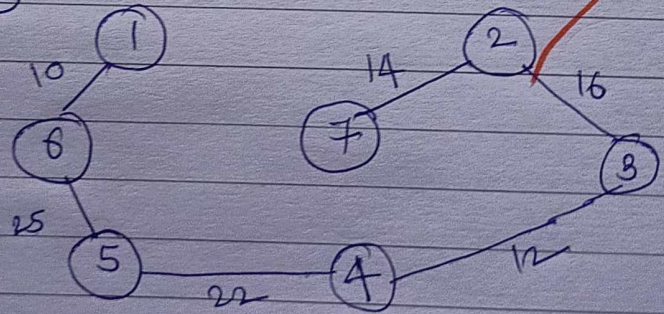
Step ④ :



Step ⑤ :



Step ⑥ :



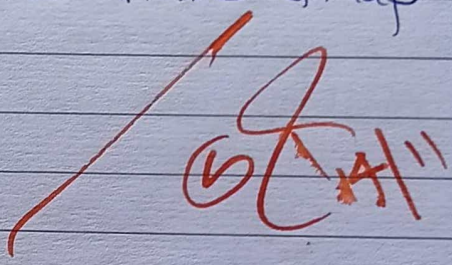
Cost of minimum spanning tree : Sum of edge weights

$= 10 + 25 + 22 + 12 + 16 + 14$

$= 99$ units.

Conclusion :

Hence We learn Greedy Search Algorithm for Prim's minimal spanning Tree algorithm. We can use same technique for real World problems for example we have GMap.



Assignment No. 3

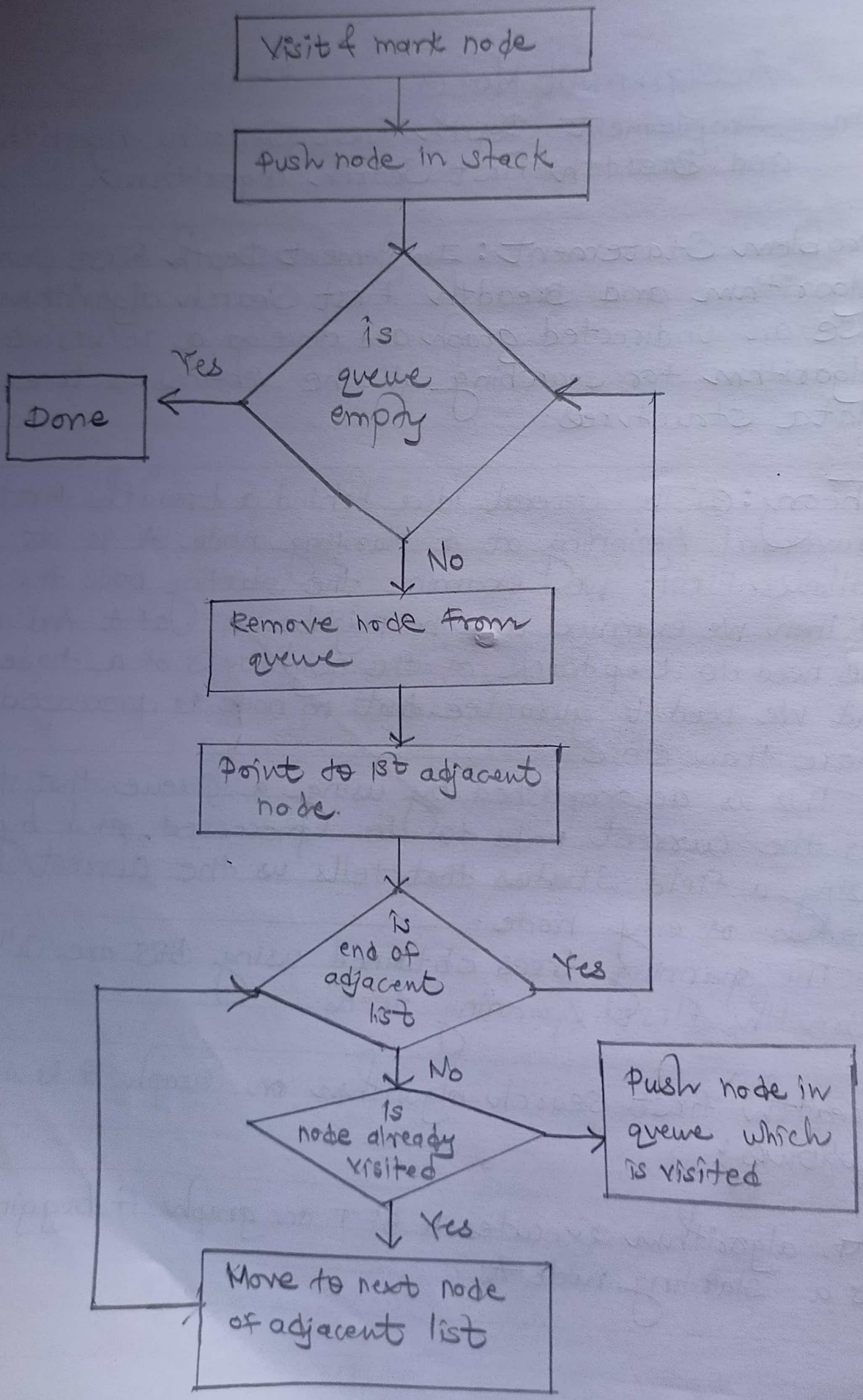
Aim: Implement Depth first Search algorithm and Breadth First Search algorithm.

Problem Statement: Implement Depth First search algorithm and Breadth First Search algorithm. Use an undirected graph and develop a recursive algorithm for searching all the vertices or tree data structures.

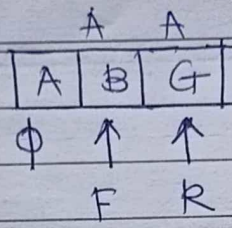
Theory: ① The General idea behind a breadth first traversal beginning at a starting node A is as follows. First we examine the starting node A.
② Then we examine all the neighbours of A. And so on. We need to keep track of the neighbours of a node, and we need to gurantee that no node is processed more than once.
③ This is accomplished by using a queue that tells us the current nodes to be processed, and by using a field status that tells us the current status of any node.
④ The spanning trees obtained using BFS are called Breadth First Spanning trees.

Breadth first search algorithm on Graph G is as follows:

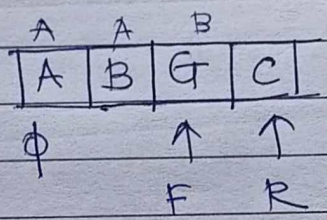
An algorithm executes a BFT on graph G begginning is a starting node A.


Visit & mark node
Push node in stack
Is queue empty
Yes
Done
No
Remove node From queue
Point to 1st adjacent node.
Is end of adjacent list
Yes
No
Is node already visited
Push node in queue which is visited
Yes
Move to next node of adjacent list

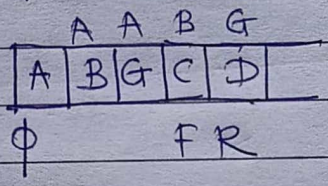
Step ② :



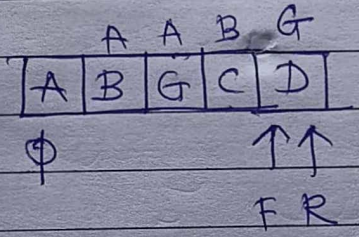
Step ③ :



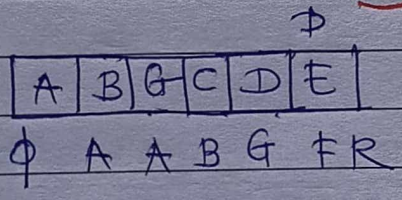
Step ④ :



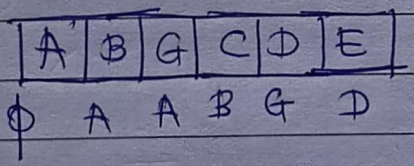
Step ⑤ :



Step ⑥ :



Step ⑦ :



Adjacency list :	
A	B, G
B	C
C	D
D	B, E
E	—
F	D, E, G
G	D

A → G → D → E

BFS path

1. Initialize all nodes to the ready state (status = 1).

2. Put the starting node A in queue and change its status to the waiting state (status = 2).

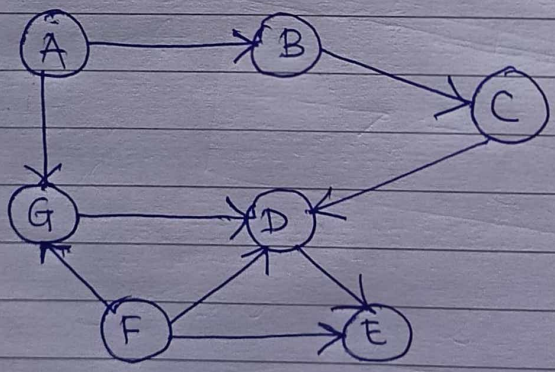
3. Repeat the following steps until queue is empty:

   a. Remove the front node N of queue. Process N and change the status of N to the processed state (status = 3).

   b. Add to the rear of queue all the neighbours of N that are in the ready state (status = 1) and change their status to the waiting state (status = 2).

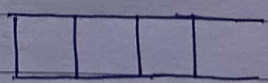
4. Exit.

Breadth First Search Example:



Min (A) → (E)

Step (1):



ϕ

Depth First Search (Stack):

Algorithm:

Step 1: Push starting vertex into stack.

Step 2: While (stack doesn't get empty)

{

i. P = Top() pop the top node

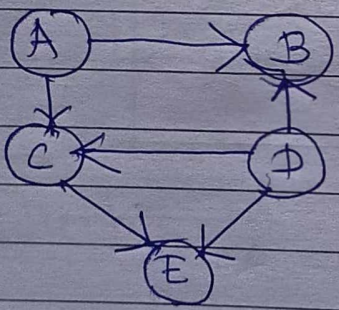
ii. Then put the popped element to the adjacent stack.

iii. If there is no adjacent vertex then pop the node.

}

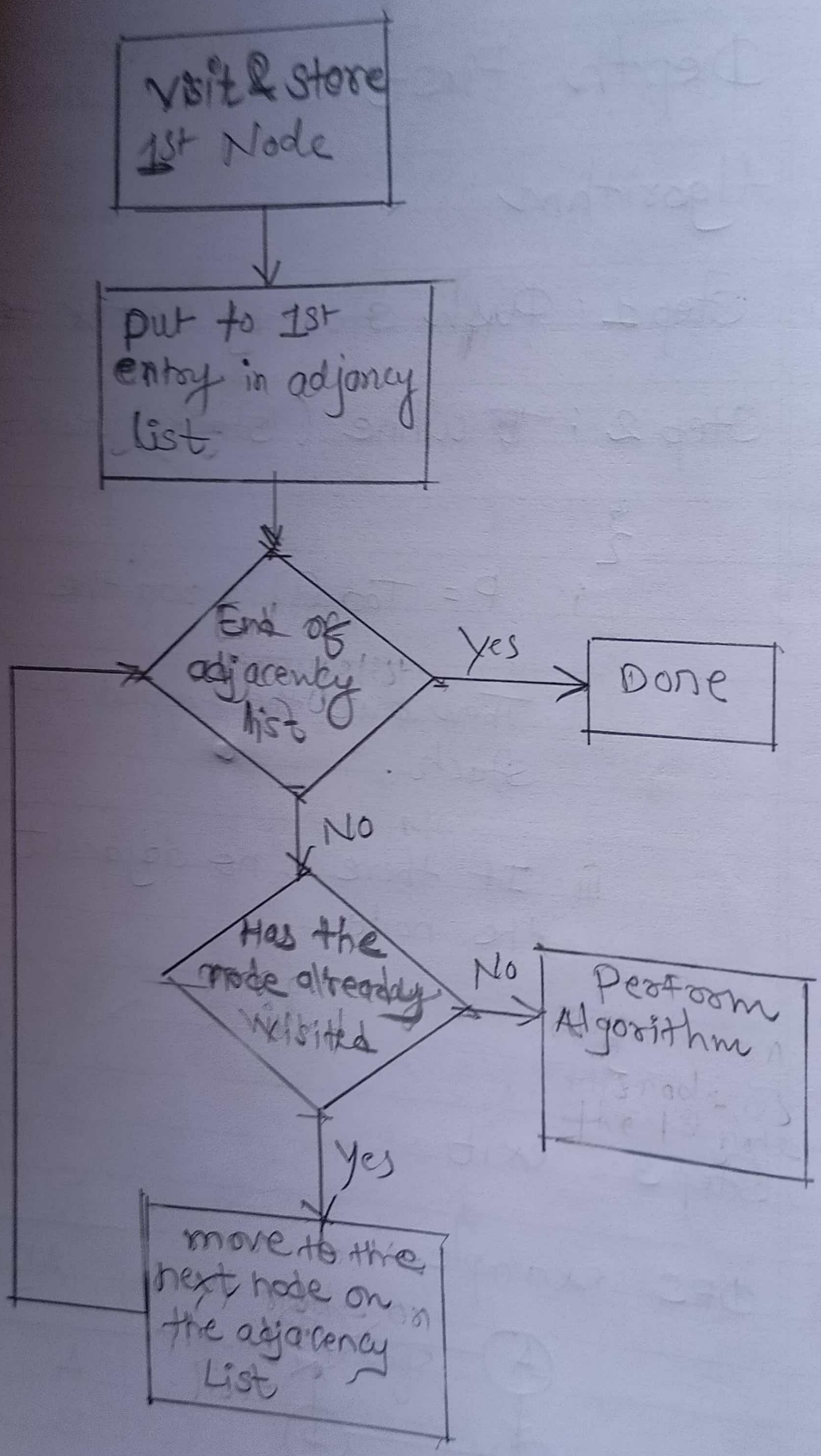
Step 3: exit.

DFS example:

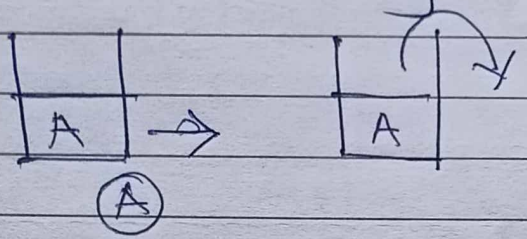


Adjacency List	
A	B, C
B	–
C	E
D	E, C
E	–

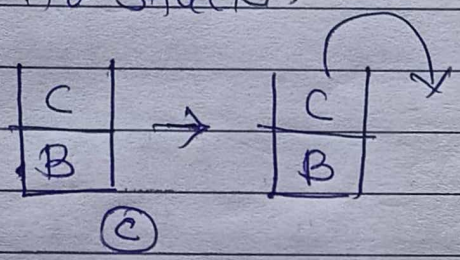
DFS flowchart



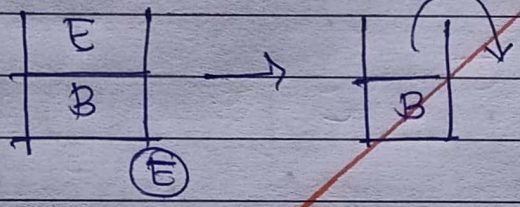
Step (1) : Add the Starting vertex:



Step (2) : Pop A and push the adjacent vertex of A to stack.

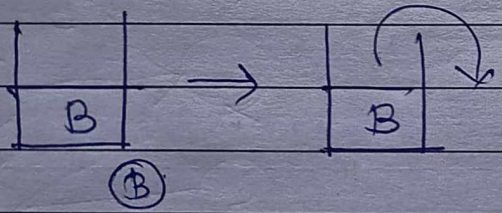


Step (3) : Pop C and then put adjacent of C to the stack.



Step (4) :

Pop E and then put adjacent of E to the stack



DFS Path ⟶ A, C, E, B

Conclusion : Hence we studied and implemented DFS & BFS Search algorithms.

Assignment No. 4

Aim: Implement A* algorithm for finding shortest path.

Problem Statement: Implement A* informed search algorithm for finding shortest path. Write Algorithm & pseudocode for the same. And solve 8 puzzle problem.

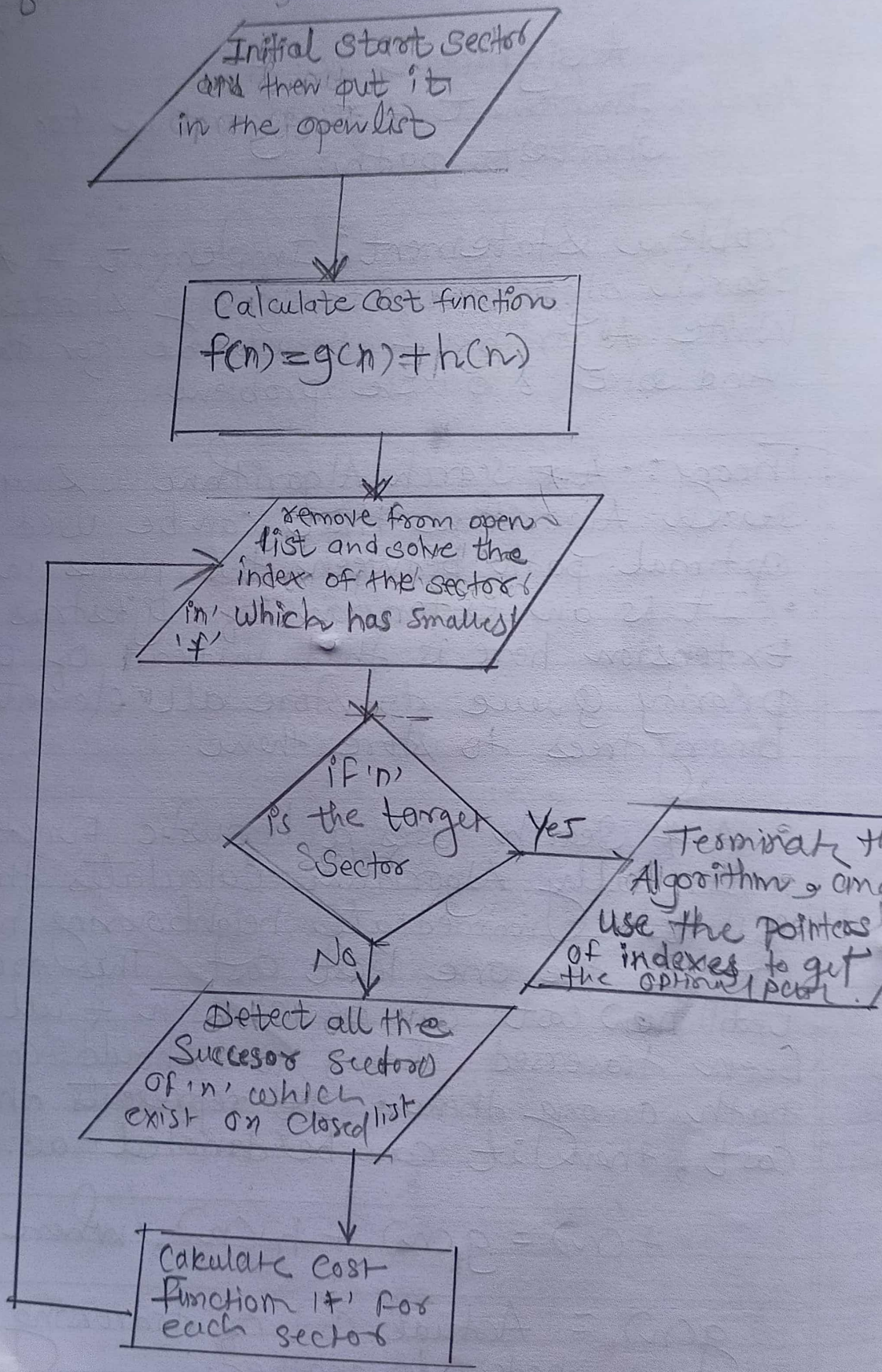
Theory:

- A* search Algorithm is simple & efficient search Algorithm that can be used to find optimal path between two nodes in a graph.
- It is an extension of Dijkstra's Algorithm. Extension here is that instead of using a priority queue to store all elements, this use binary trees to store them.
- A* Search uses a heuristic Function.
- Initially Algorithm calculates the cost and all its immediate neighbouring nodes n and chooses the one least cost. This process repeats until new cost can be choosen & all paths have been traversed. Then, you should consider the path among them. It represents the final cost, then it can be denoted as:

$$f(n) = g(n) + h(n), \text{ where}$$

$g(n)$ = Actual cost of traversing from one node to another.

A* flowchart


Initial start sector and then put it in the openlist
Calculate cost function
f(n)=g(n)+h(n)
remove from open list and solve the index of the sector 'n' which has smallest 'f'
IF 'n' is the target Sector
Yes
Terminate the Algorithm, and use the pointers of indexes to get the optimal path
No
Detect all the Successor sectors of 'n' which exist on Closed list
Cakulate cost function 'f' for each sector

$h(n)$: Heuristic Approximation of the nodes value (heuristic value).

- Approximation Heuristic :

To determine h, there are 3 approx heuristics

1) Manhattan Distance : It is the total of the absolute values of the descrepencies between the x and y co-ordinates of the current & goal cells.

Formula :

$h = abs(curr\text{-}cell.x - goal.x) + abs(curr\text{-}cell.y - goal.y)$

2) Diagonal Distance :

Formula : $dx = abs(curr\text{-}cell.x - goal.x)$
$dy = abs(curr\text{-}cell.y - goal.y)$

$h = D * (dx + dy) + (D_2 - 2 * D) * \min(dx, dy)$

3) Euclidean Distance : Distance between goal cell & current cell using distance formula :

$h = sqrt(curr\text{-}cell.x - goal.x)^{1/2} + (curr\text{-}cell.y - curr.cell.y)^{1/2}$

Algorithm:

Initial Condition: Created two lists - open & closed.

- Open list must be initialized.
- put starting node with open list is non-empty.

Step 1: Find node with least f on the open list & name it "q".

Step 2: Remove q from open list.

Step 3: Produce q's eight descendents ← set q as their parent.

Step 4: For every descendent:

1) If finding a successor is the goal, cease looking.

2) Else, Calculate g and h for the successor.

3) Skip this successor if there is node in the ~~open~~ open list with the same location as it but lower if but lower f values then there successor is present.

4) Skip the successor if there is node is CLOSED list with same position as successor but a

lower of f value,

otherwise, add the node to open, list end (for loop).

- push g into the closed list f end while loop.

example: 8 puzzle problem.

2	5	4
6	8	7
1		3

Initial State

1	2	3
8		4
7	6	5

Goal State.

Consider g(n) depth of node

h(n) = no. of misplaced tiles.

Time Complexity of A* : $O(b^d)$

Space Complexity of A* : $O(b^d)$

Conclusion: We have learned to implement A* Algorithm & solve 8 puzzle problem using A* Algorithm.

Assignment No. 5.

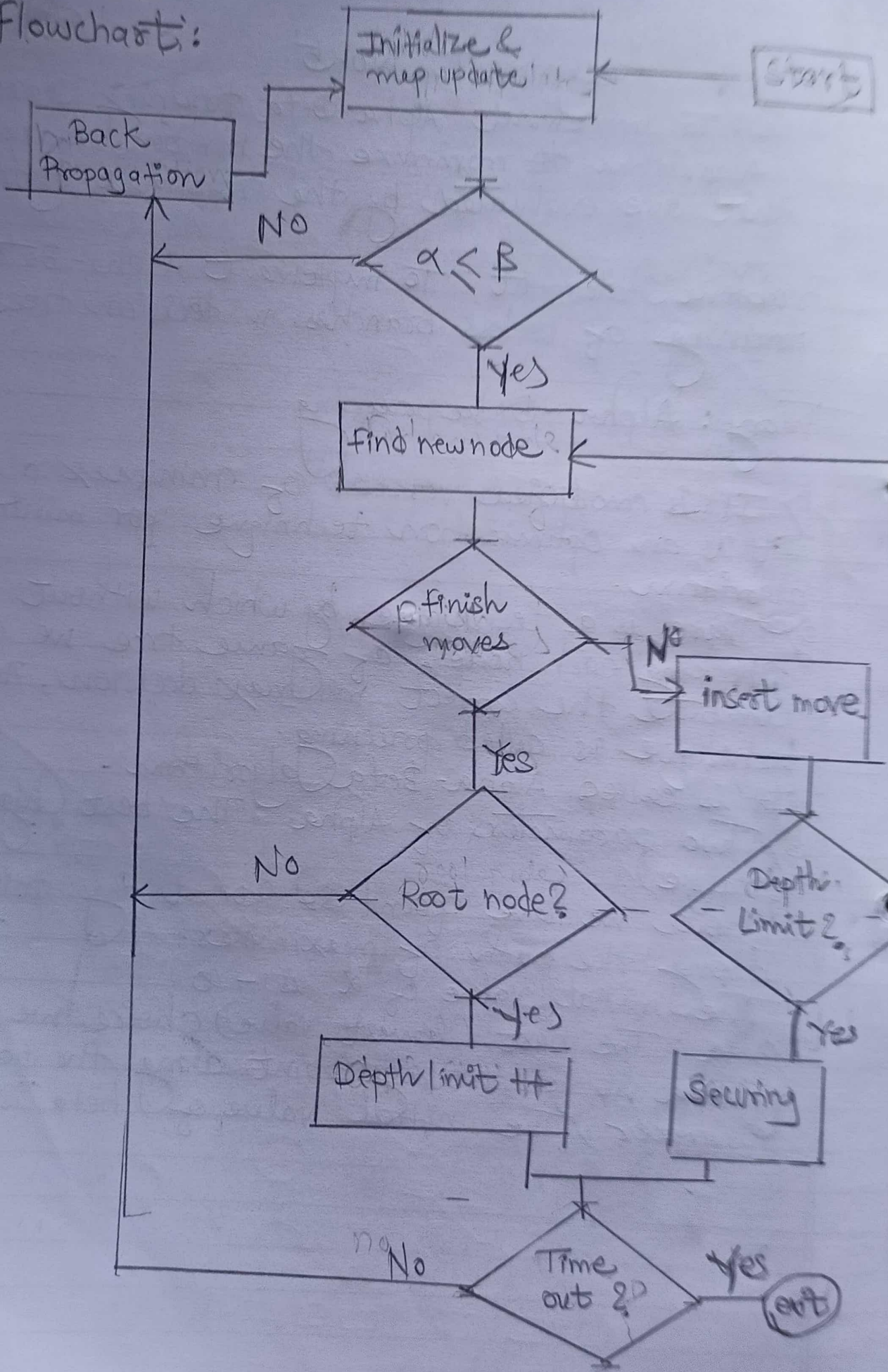
Aim: To implement Alpha-Beta pruning search Algorithm to minimize the number of nodes that are evaluated by the minimax Algorithm.

Problem Statement: To implement Alpha-Beta pruning of useless branches in decision trees.

Theory: Alpha-Beta pruning.

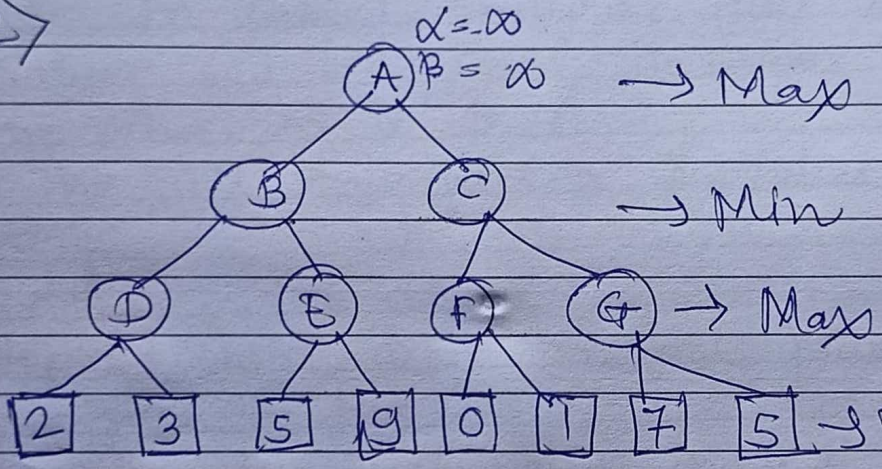
1) It is modified version of minimax algorithm. It is an optimization technique for minimax algorithm.
2) There is a technique by which without checking each node of game tree we can compute the correct minimax decision, And this technique is called pruning.
So It is called Alpha-Beta algorithm.
3) Two parameters of Alpha: The best (highest value) choice we have found so far at any point along the path of maximizer.
The initial value of α is $-\infty$.
Beta: The best (lowest value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$.

Alpha Beta flowchart:



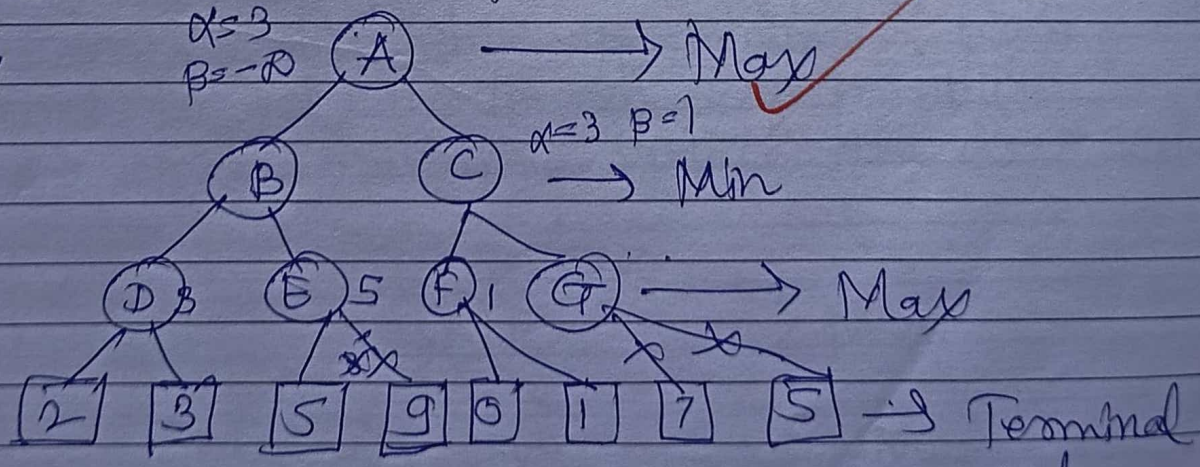
Max player begins moving from A, the value of D will be determined as Max's turn at D & then compared, value at node will be Max. (2,8) = 8 node value also be 8. The algorithm now returns to node B, where the value of will change as this turn Min.

2)



In next step algorithm traverse the next successor of nodes which is Node E and value of α=β & β=3 will also be passed

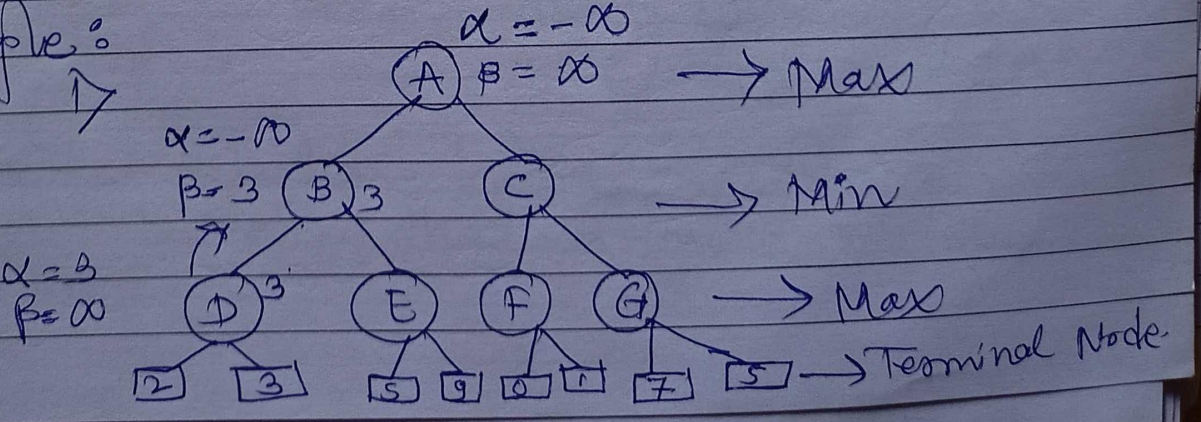
3)



6) Conditions for Alpha-Beta pruning:
1) $\alpha >= \beta$
2) The max player will only update the value of alpha.
3) The min player will only update the value of beta.
4) While backtracking a tree, the node values will be passed to upper nodes instead of values will be passed to upper nodes instead of values of α, β.
5) We will only pass α, β values to child node.

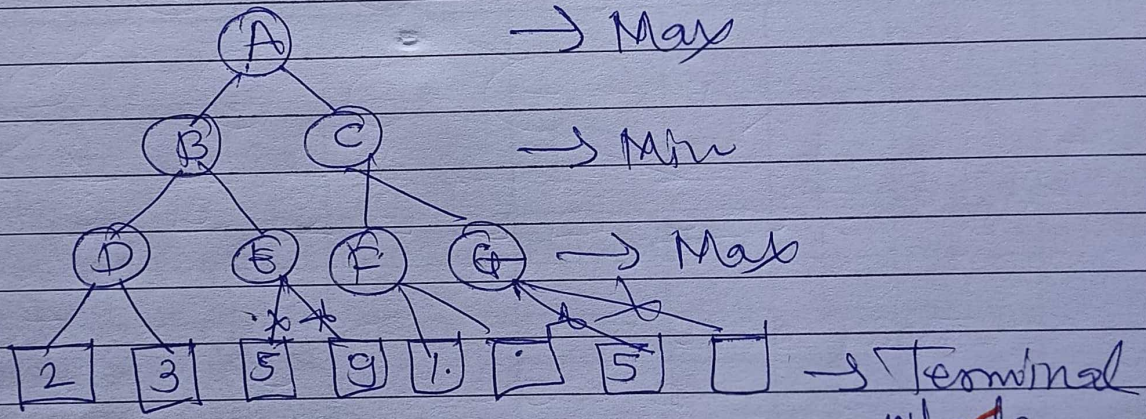
Algorithm:
Step 1) Start DFS traversal from root of game tree.
Step 2) Set initial values of α, β as follow.
α = Int-min (-infinity)
β = Int-max (+infinity)
Step 3) Traverse tree in DFS manner where maximizer player tries to get the highest score possible while the minimizer player tries to get the lowest score possible.
Step 4) While traversing update the $\alpha - \beta$ value

Example:
1)



The node f, the values of will be compared with the left child which is 0. & max(3,0)=3 & then with the right child, which is max(3,1)=3 will remain.

5. Node of f returns value 1 to node C at C=3 & =1, Beta modified compared to 1 resulting in min(1,=). Now the ab C=3 & =1 again. It meets the condition >= the algorithm will prune the next child of C, which is G & never compute subtree G.



Time Complexity : Worst ordering $O(b^d)$

Best ordering $O(b^{d/2})$

Conclusion: We have learnt concepts of α-β search method that optimize minimax algorithm.