Algorithm-Analysis-Module-4-Important-Topics

- Algorithm-Analysis-Module-4-Important-Topics
  - 1. Design method- control abstraction
  - 2. Optimality principle
  - 3. Dynamic programming
    - What is Dynamic Programming
    - Characteristics of Dynamic Programming
      - Overlapping Subproblems
      - Optimal Substructure
    - Steps in Dynamic Programming
  - 4. Matrix chain multiplication
    - Matrix Chain Multiplication Analysis
      - Scalar Multiplications
    - Matrix chain multiplication
      - Step 1: Structure of an optimal parenthesization
      - Step 2: Recursive solution
      - Step 3: Computing the optimal costs
      - Step:4: Constructing an optimal solution
    - Time complexity of matrix chain multiplication
  - 5. Floyd warshall Algorithm
    - All Pair Shortest path problem
    - Example
      - Step 1: Make Adjacency Matrix of D0
      - Step 2: Make Adjacency Matrix of D1
      - Step 3: Make Adjacency Matrix of D2

1. Design method- control abstraction

- By control abstraction we mean a procedure whose flow of control is clear but whose primary operations are specified by other procedures whose precise meaning is left undefined.

```
procedure DANDC(p, q)
  global n, A(1:n); integer m,p,q:        //1 ≤ p ≤ q ≤ n//
  if SMALL(p,q)
    then return (G(p, q))
    else m ← DIVIDE(p,q)      //p ≤ m < q//
      return(COMBINE(DANDC(p,m), DANDC(m + 1,q)))
  endif
end DANDC
```

- In the above example, DANDC Function has many functions inside it, So the flow of operations is clear, and its primary operations are specified by these smaller functions like SMALL, DIVIDE, COMBINE etc

2. Optimality principle

- The principle of optimality states that an optimal sequence of decisions has the property that whatever the initial state and decisions are, remaining decisions must constitute an optimal decision sequence with regard to the state resulting from first decision

Simpler explanation

Think of it like this: imagine you're playing a video game where you have to navigate through different levels to reach the final boss. The principle of optimality is like saying that to beat the game most efficiently, you should always make decisions that help you get closer to winning in the long run, not just what seems good at the moment. So, every move you make should be part of a bigger plan to reach your goal, rather than just solving each immediate problem as it comes up.

3. Dynamic programming

What is Dynamic Programming

- Its of the main algorithm design technique
- Its mainly an optimization over plain recurison
- Whenever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming.
- The idea is to simply store the results of subproblems, so that we do not have to recompute them when needed later
- Dynamic Programming follows the principles of optimality.

Characteristics of Dynamic Programming

Overlapping Subproblems

- In dynamic Programming the solution of subproblems are needed repeatedly. THe computed solutions are stored in a table, so that these dont have to be recomputed. Then combines the solutions of the sub problems

Optimal Substructure

- A given problem has Optimal substructure Property. If the optimal solution of the given problem can be obtained using optimal solutions of its subproblems
- Example: Shortest path problem
  - If a node x lies in the shortest path from a source node u to destination node v, then the shortest path from u to v is the combination of the shortest path from u to x, and the shortest path from x to v

Steps in Dynamic Programming

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution, typically in bottom up fashion
4. Construct an optimal solution from computed information

Simpler explanation

We can use a **Cooking Example**

1. **Characterize the structure of an optimal solution (What makes the perfect dish?):** This is like figuring out what makes the final dish cooked perfectly. In our example, it might be that all the ingredients are cooked just right and assembled in the correct order.
2. **Recursively define the value of an optimal solution (How do perfect ingredients lead to a perfect dish?):** Here, we define how the perfectly cooked dish relies on perfectly cooked smaller parts. A perfectly cooked dish requires perfectly chopped vegetables, preheated oven, and properly cooked meat (sub-problems).
3. **Compute the value of an optimal solution, typically in bottom-up fashion (Cooking step-by-step):** This translates to following the recipe step-by-step. We don't cook the entire dish at once. Instead, we start with the simplest tasks (chopping vegetables, preheating) and build up to cooking the whole dish. This ensures we only cook each ingredient once (avoiding waste).
4. **Construct an optimal solution from computed information (Following the recipe with prepped ingredients):** Once we have perfectly cooked ingredients (solutions to sub-problems), we follow the recipe (recorded information) to assemble the final dish (optimal solution). We use the prepped ingredients and follow the order to create the perfect dish.

4. Matrix chain multiplication

Matrix Chain Multiplication Analysis

- Suppose we wish to compute the product of 4 matrices A1 x A2 x A3 x A4
- We can multiple in many ways like

The different parenthesizations are

$(A_1(A_2(A_3 A_4)))$

$(A_1((A_2 A_3) A_4))$

$((A_1A_2)(A_3 A_4))$

$((A_1(A_2 A_3))A_4)$

$(((A_1A_2) A_3)A_4)$

Scalar Multiplications

$A_1 \rightarrow 10X100 \quad A_2 \rightarrow 100X5 \quad A_3 \rightarrow 5X50$

$((A_1A_2) A_3)$

- Number of scalar multiplications $=10x100x5$
- The resultant matrix dimensions will be $10x5$

- Number of scalar multiplications $=10x5x50$

- Total number of scalar multiplications

$= 10x100x5 + 10x5x50$

$= 7500$

$A_1 \rightarrow 10X100 \quad A_2 \rightarrow 100X5 \quad A_3 \rightarrow 5X50$

$(A_1 (A_2 A_3))$

- Number of scalar multiplications =**100x5x50**
- The resultant matrix dimensions will be 100x50
- Number of scalar multiplications =**10x100x50**
- Total number of scalar multiplications
  = 100x5x50 + 10x100x50
  = **75000**

- Here the first way is more optimal, because the number of scalar multiplications required is 7500

Matrix chain multiplication

- In matrix chain multiplication problem, we are not actually multiplying matrices
- Our goal is to determine an order for multiplying matrices that has the lowest cost
- Given a chain of A1,A2 ..... An matrices, for i = 1, 2.. n . Matrix Ai has dimension pi-1 X pi
- Fully parethesize the product A1,A2,A3... An in a way it minimizes scalar multiplications

Step 1: Structure of an optimal parenthesization

- Aij = AiAi+1 .... Aj where i<=j
- if i < j
  - Split the problem into 2 subproblems
  - (Ai + Ai+1 ... Ak and Ak+1 .... Aj) where i<=k < j
  - Steps
    - Compute Ai..k
    - Compute Ak+1..j
    - Ai..j = Ai..k x Ak+1 ..j

- Total cost = Cost of computing Ai..k + Cost of computing Ak+1..j + Cost of multiplying them together

Step 2: Recursive solution

- 2 Arrays
- m[i, j] -> Minimum number of scalar multiplications needed to compute the matrix Ai..j
- m[1,n] -> Lowest cost to compute A1..n
- Ai.i = Ai so m[i,i] = 0 for i=1,2 ... n

$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j. \end{cases}$$

$m[i,k] \rightarrow p_{i-1} x p_k \quad m[k+1,j] \rightarrow p_k x p_j$

$s[i,j] \rightarrow$ Value of k at which we split the product $A_i A_{i+1} \ldots A_j$

Step 3: Computing the optimal costs

```
Algorithm Matrix_Chain_Order(p)
n = p.length – 1
Let m[1..n, 1..n] and s[1..n-1 , 2..n] be new tables
for i=1 to n do
        m[i,i] = 0
for x=2 to n do
        for i=1 to n-x+1 do
                j=i+x-1
                m[i,j] = α
                for k=i to j-1 do
                        q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
                        if q<m[i,j] then
                                m[i,j] = q
                                s[i,j] = k
return m[][] and s[][]
```

Step:4: Constructing an optimal solution

```
Algorithm Print_Optimal_Parens(s,i,j)
{
    if i==j then
        print "A"i
    Else
        print "("
        print_Optimal_Parens(s,i,s[i,j])
        print_Optimal_Parens(s,s[i,j]+1,j)
        print ")"
}
```

Initial call is **PRINT-OPTIMAL-PARENS(s,1,n)**

Time complexity of matrix chain multiplication

We are generating **n(n-1)/2** number of elements in matrix m[].

To calculate each element it will take atmost **n** time.

So the time complexity = O(n.n(n-1)/2) = **$O(n^3)$**

5. Floyd warshall Algorithm

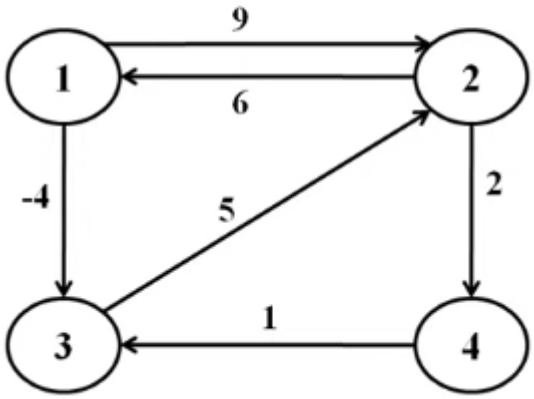
All Pair Shortest path problem

- The floyd warshall algorithm is for solving the all pairs shortest path problem
- As a result of this algorithm, it will generate a matrix, which will generate a matrix, which

will represent the minimum distance from any node to all other nodes in the graph

Example

Example: Using Floyd Warshall Algorithm, find the shortest path distance between every pair of vertices



Step 1: Make Adjacency Matrix of D0

- Below we have 1,2,3,4 vertices
- The distance is marked

	1	2	3	4
1	0	9	-4	∞
2	6	0	∞	2
3	∞	5	0	∞
4	∞	∞	1	0

$$D^0 = \begin{pmatrix} 0 & 9 & -4 & \alpha \\ 6 & 0 & \alpha & 2 \\ \alpha & 5 & 0 & \alpha \\ \alpha & \alpha & 1 & 0 \end{pmatrix}$$

Step 2: Make Adjacency Matrix of D1

- We need to find matrix D1
- Keep the first row, first column and diagonal elements of D0

	1	2	3	4
1	0	9	-4	∞
2	6	0	∞	2
3	∞	5	0	∞
4	∞	∞	1	0

- We need to find the remaining numbers (We need to replace the highlighted ones)
- Use this formula D1 (m,n) = minimum of (D0(m,n), D0(m,1) + D0(1,n))

Use this formula to find D1

D1 (m,n) = minimum of (D0(m,n), D0(m,1) + D0(1,n))

- For example
  - D1 (2,3)
  - Here m = 2
  - n = 3
  - = Minimum of (D0 (2,3), D0(2,1) + D0(1,3) = minimum (∞, 6+(-4)) = 2

  Similarly we can find all points like

Find the matrix D^1

- Keep the 1st row, 1st column and diagonal elements of D^0 as such
- $D^1(2,3) = \min\{ D^0(2,3), D^0(2,1) + D^0(1,3) \} = \min\{\alpha, 6+(-4)) = 2$
- $D^1(2,4) = \min\{ D^0(2,4), D^0(2,1) + D^0(1,4) \} = \min\{2, 6+\alpha) = 2$
- $D^1(3,2) = \min\{ D^0(3,2), D^0(3,1) + D^0(1,2) \} = \min\{5, \alpha +9) = 5$
- $D^1(3,4) = \min\{ D^0(3,4), D^0(3,1) + D^0(1,4) \} = \min\{\alpha, \alpha + \alpha) = \alpha$
- $D^1(4,2) = \min\{ D^0(4,2), D^0(4,1) + D^0(1,2) \} = \min\{\alpha, \alpha +9) = \alpha$
- $D^1(4,3) = \min\{ D^0(4,3), D^0(4,1) + D^0(1,3) \} = \min\{1, \alpha +(-4)) = 1$

	1	2	3	4
1	0	9	-4	∞
2	6	0	2	2

	1	2	3	4
3	∞	5	0	∞
4	∞	∞	1	0

Step 3: Make Adjacency Matrix of D2

- Keep the second row, second column and diagonal elements of D0

$$D^1 = \begin{pmatrix} 0 & 9 & -4 & \alpha \\ 6 & 0 & 2 & 2 \\ \alpha & 5 & 0 & \alpha \\ \alpha & \alpha & 1 & 0 \end{pmatrix}$$

)²

Use this formula to find D2

D2 (m,n) = minimum of (D1(m,n), D1(m,2) + D1(2,n))

$$D^2(1,3) = \min\{ D^1(1,3), D^1(1,2) + D^1(2,3) \} = \min\{-4, 9+2) = -4$$
$$D^2(1,4) = \min\{ D^1(1,4), D^1(1,2) + D^1(2,4) \} = \min\{\alpha, 9+2) = 11$$
$$D^2(3,1) = \min\{ D^1(3,1), D^1(3,2) + D^1(2,1) \} = \min\{\alpha, 5+6) = 11$$
$$D^2(3,4) = \min\{ D^1(3,4), D^1(3,2) + D^1(2,4) \} = \min\{\alpha, 5+2) = 7$$
$$D^2(4,1) = \min\{ D^1(4,1), D^1(4,2) + D^1(2,1) \} = \min\{\alpha, \alpha+6) = \alpha$$
$$D^2(4,3) = \min\{ D^1(4,3), D^1(4,2) + D^1(2,3) \} = \min\{1, \alpha+2) = 1$$

Step 4: Make Adjacency Matrix of D3

- Keep the 3rd row, 3rd column and diagonal elements of D2

$$D^2 = \begin{pmatrix} 0 & 9 & -4 & 11 \\ 6 & 0 & 2 & 2 \\ 11 & 5 & 0 & 7 \\ \alpha & \alpha & 1 & 0 \end{pmatrix}$$

Find the matrix D^3

- Keep the 3rd row, 3rd column and diagonal elements of D^2 as such
- $D^3(1,2) = \min\{ D^2(1,2), D^2(1,3) + D^2(3,2) \} = \min\{9, -4+5) = 1$
- $D^3(1,4) = \min\{ D^2(1,4), D^2(1,3) + D^2(3,4) \} = \min\{11, -4+7) = 3$
- $D^3(2,1) = \min\{ D^2(2,1), D^2(2,3) + D^2(3,1) \} = \min\{6, 2+11) = 6$
- $D^3(2,4) = \min\{ D^2(2,4), D^2(2,3) + D^2(3,4) \} = \min\{2, 2+7) = 2$
- $D^3(4,1) = \min\{ D^2(4,1), D^2(4,3) + D^2(3,1) \} = \min\{\alpha, 1+11) = 12$
- $D^3(4,2) = \min\{ D^2(4,2), D^2(4,3) + D^2(3,2) \} = \min\{\alpha, 1+5) = 6$

Use this formula to find Dp

$$D_p(m,n) = \min\left(D_{p-1}(m,n), D_{p-1}(m,p) + D_{p-1}(p,n)\right)$$

- Here p is 3
- D3 (1,2) = min(D2(1,2), D2(1,3) + D2(3,2)) min (9, -4+5) = 1

Step 5: Make Adjacency Matrix of D4

$$D^3 = \begin{pmatrix} 0 & 1 & -4 & 3 \\ 6 & 0 & 2 & 2 \\ 11 & 5 & 0 & 7 \\ 12 & 6 & 1 & 0 \end{pmatrix}$$

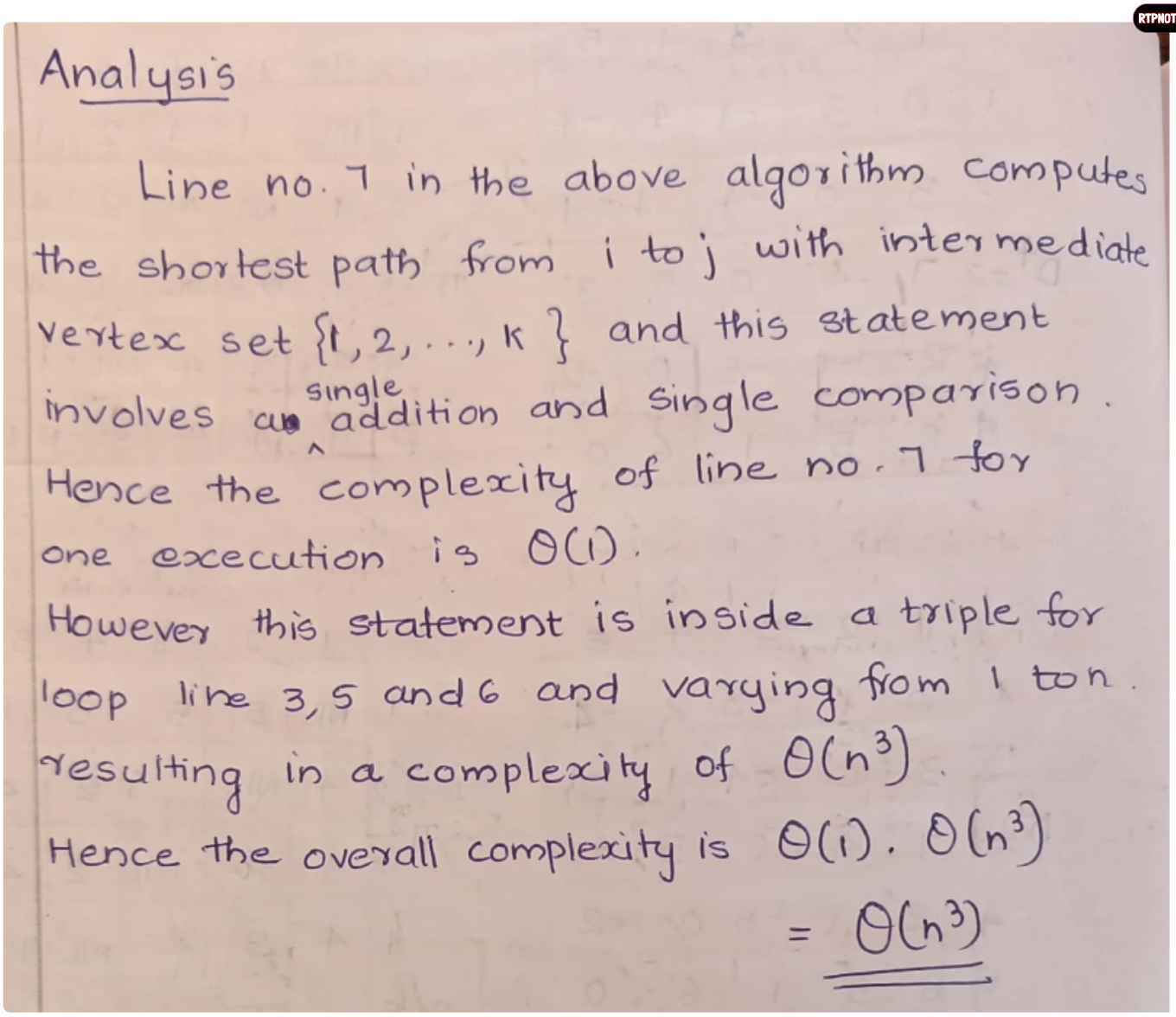
Find the matrix D^4

- Keep the 4th row, 4th column and diagonal elements of D^3 as such
- $D^4(1,2) = \min\{ D^3(1,2), D^3(1,4) + D^3(4,2) \} = \min\{1, 3+6) = 1$
- $D^4(1,3) = \min\{ D^3(1,3), D^3(1,4) + D^3(4,3) \} = \min\{-4, 3+1) = -4$
- $D^4(2,1) = \min\{ D^3(2,1), D^3(2,4) + D^3(4,1) \} = \min\{6, 2+12) = 6$
- $D^4(2,3) = \min\{ D^3(2,3), D^3(2,4) + D^3(4,3) \} = \min\{2, 2+1) = 2$
- $D^4(3,1) = \min\{ D^3(3,1), D^3(3,4) + D^3(4,1) \} = \min\{11, 7+12) = 11$
- $D^4(3,2) = \min\{ D^3(3,2), D^3(3,4) + D^3(4,2) \} = \min\{5, 7+6) = 5$

Floyd Warshall Algorithm

Algorithm Floyd-Warshall (W)

1. $n \leftarrow$ no. of vertices, W. rows
2. $D^0 \leftarrow W$ # direct edge weight
3. for $k = 1$ to n do
4. Let $D^{(k)} = (d_{ij}^{(k)})$ new matrix created.
5. for $i = 1$ to n do
6. for $j = 1$ to n do
7. $d_{ij}^{(k)} = \min(d_{ij}^{k-1}, \ldots, d_{ik}^{k-1} + d_{kj}^{k-1})$
8. Return D^k



6. Backtracking

Its one of the main algorithm design techniques

- The solution or answer is n tuple

$$\text{Solution is n-tuple} \rightarrow (x_1, x_2, \ldots\ldots x_n)$$

-

Example 1: Array sorting

Example-1: Sort the following array

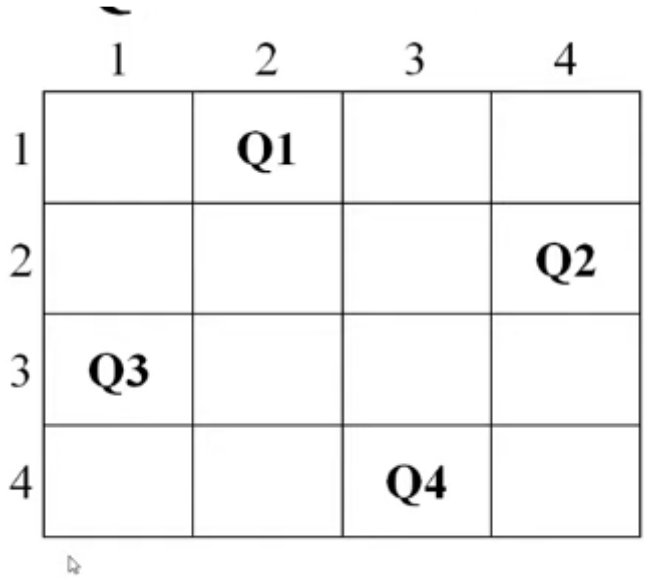
70	50	40	80	90
1	2	3	4	5

-
- Smallest entry is 40, its index is 3, inserting
- Next smallest is 50, index is 2 , inserting, and so on..
- We get the following solutions

$$\text{Solution} = (3, 2, 1, 4, 5)$$

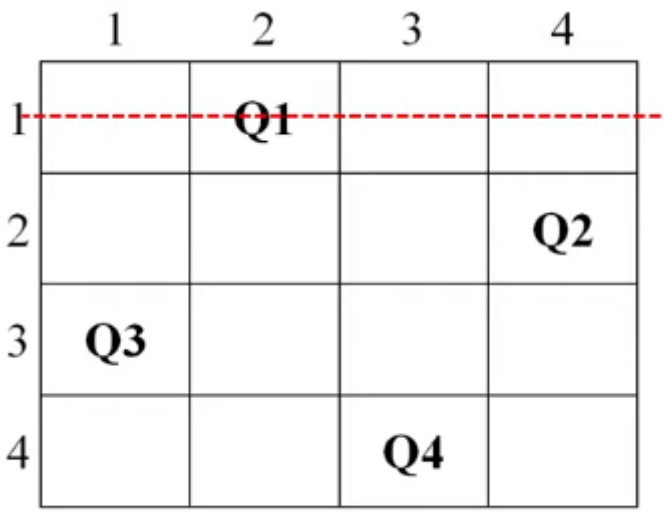
Example 2: 4 Queens problem

- Given 4x4 chessboard and 4 queens
- Arrange the queens in such a way that no two queens are in the same row column and diagonal

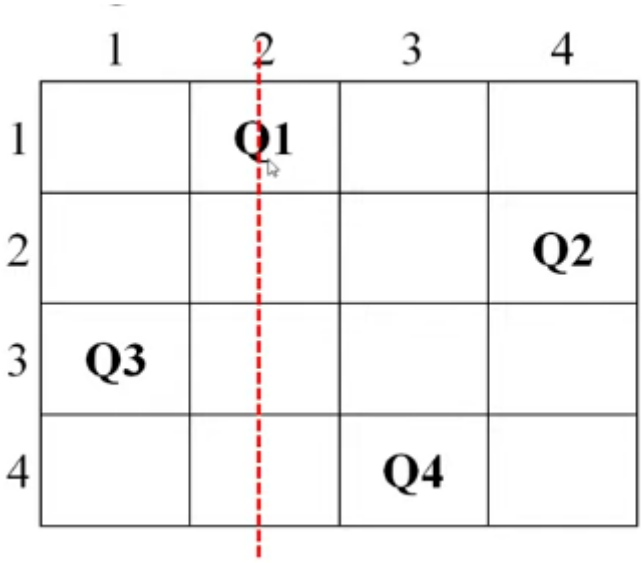


- The above is a 4 x 4 chessboard
- 4 rows, 4 columns
  - The queens are Q1, Q2,Q3 and Q4

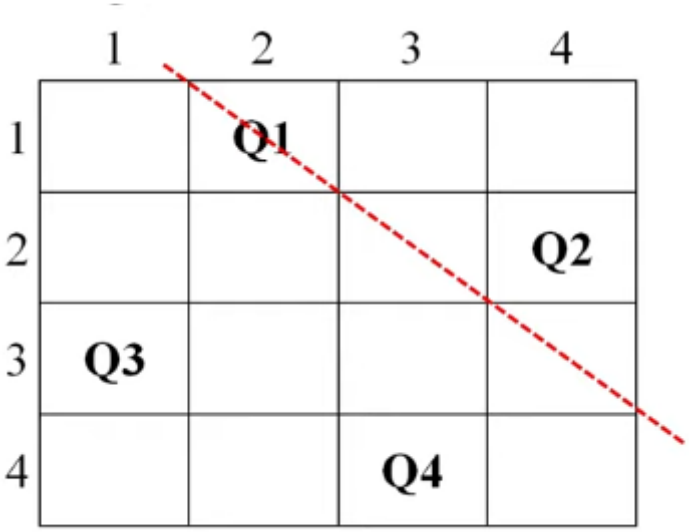
- Consider Q1
  - No other queen in that row



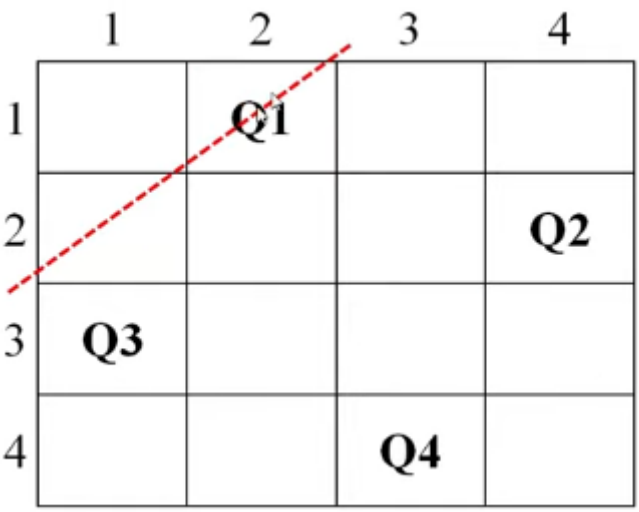
  - No other queen in that column



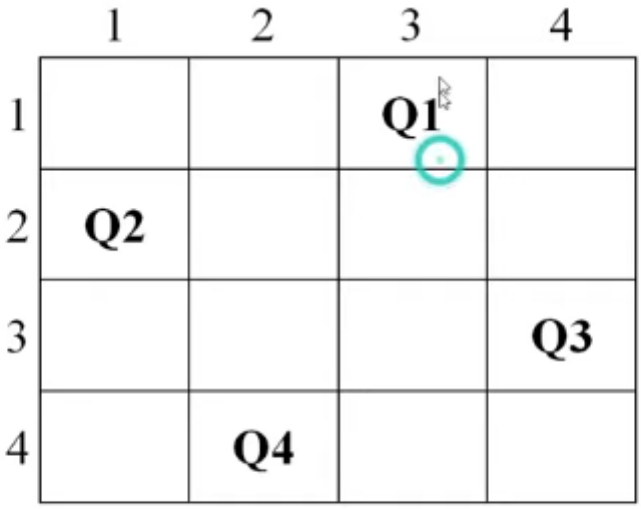
  - No queen in the positive diagonal



- No queen in negative diagonal



- One solution of 4 queen problem
  - Q1 is in column 2
  - Q2 is in column 4
  - Q3 is in column 1
  - Q4 is in column 3
  - This gives us the following solution (2,4,1,3)
- Another solution



Second solution of 4-queen problem is **(3,1,4,2)**

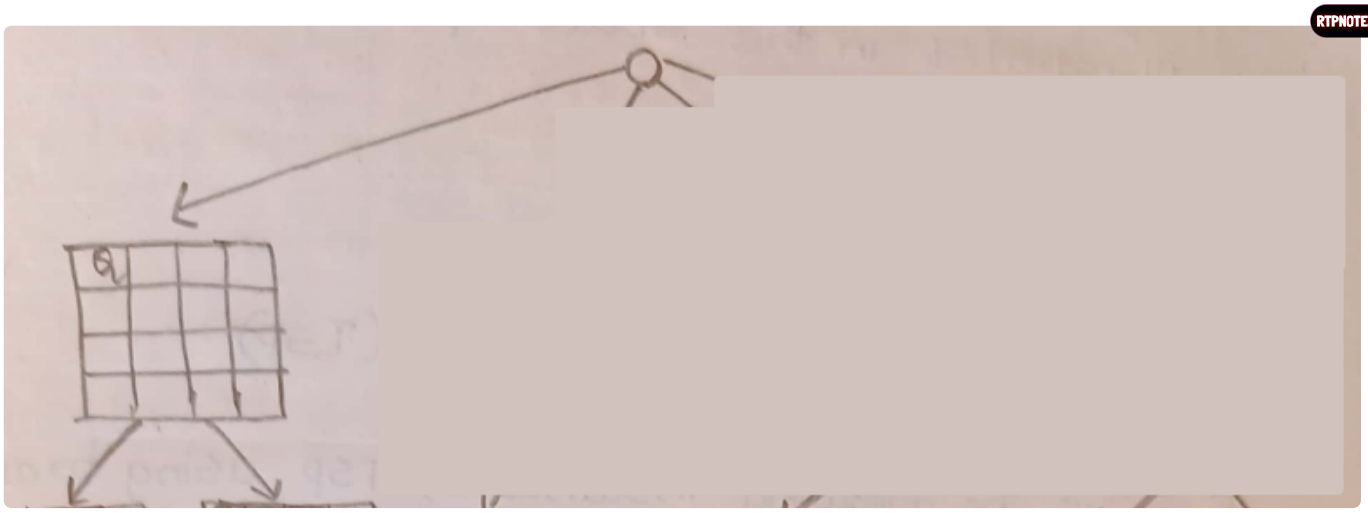
- In Backtracking a systematic approach is followed to find out the feasible solutions
- It employs a DFS strategy to explore the search space
- Usually the search is depicted in the form of a tree called state space tree

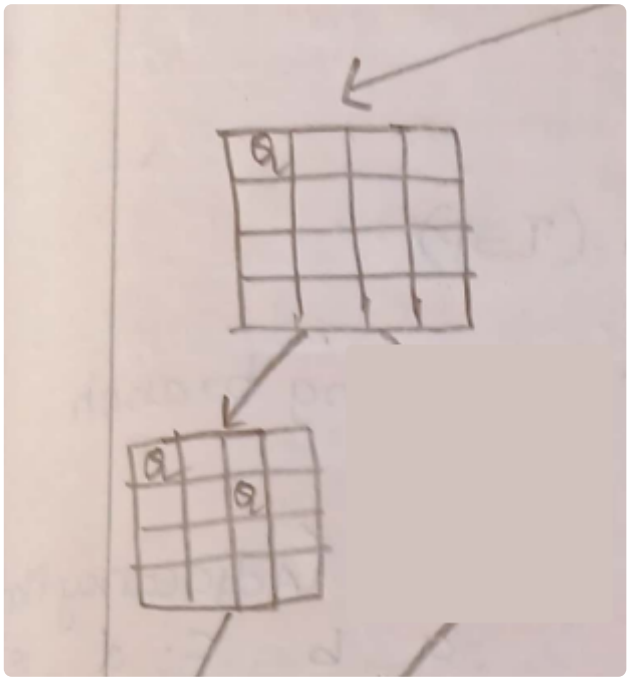
7. State space tree for 4 queens problem

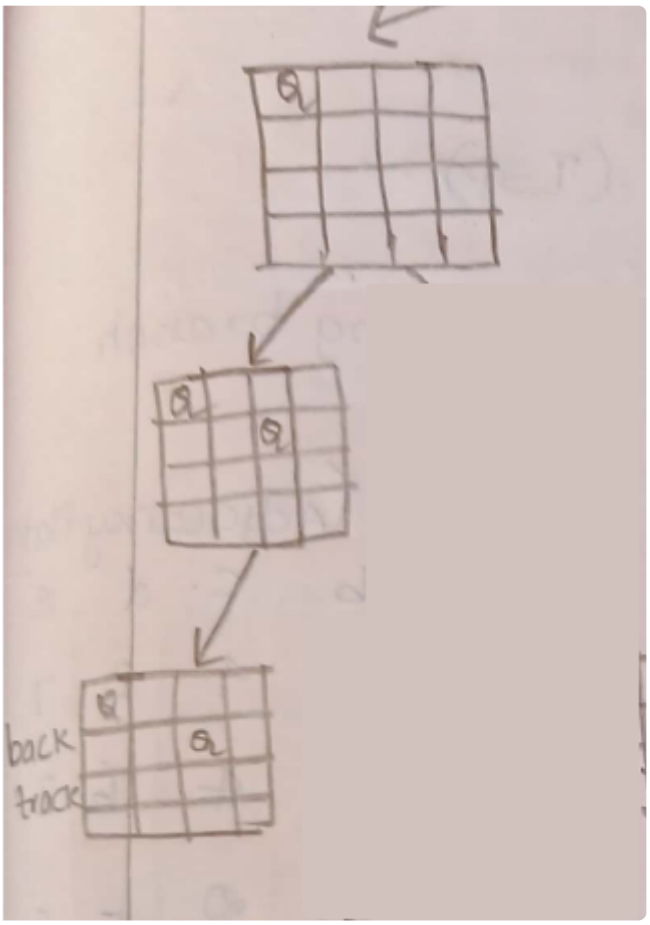
- Our goal here is to place 4 queens in a chessboard such that they dont attack each other, Lets try each solution one by one

Solution 1

- We place the first queen in the first column

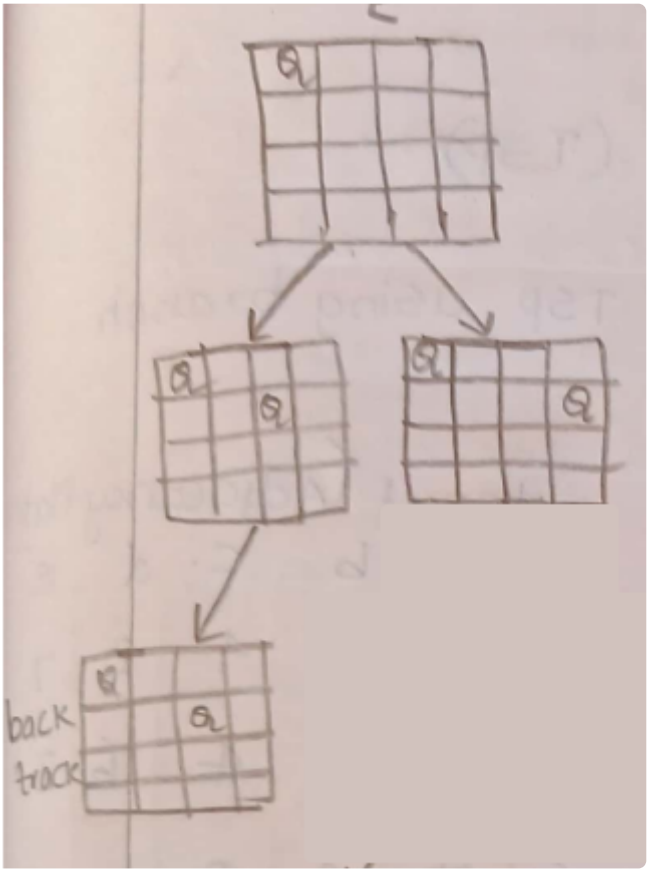


- Now we need to place the next queen
- 
- We cant place the second queen in 2nd column, because the first queen can attack diagonally
- So placing it in 3rd column
- Trying to place the 3rd queen
  - Cant place in any of the columns because attack will happen
  - So backtracking

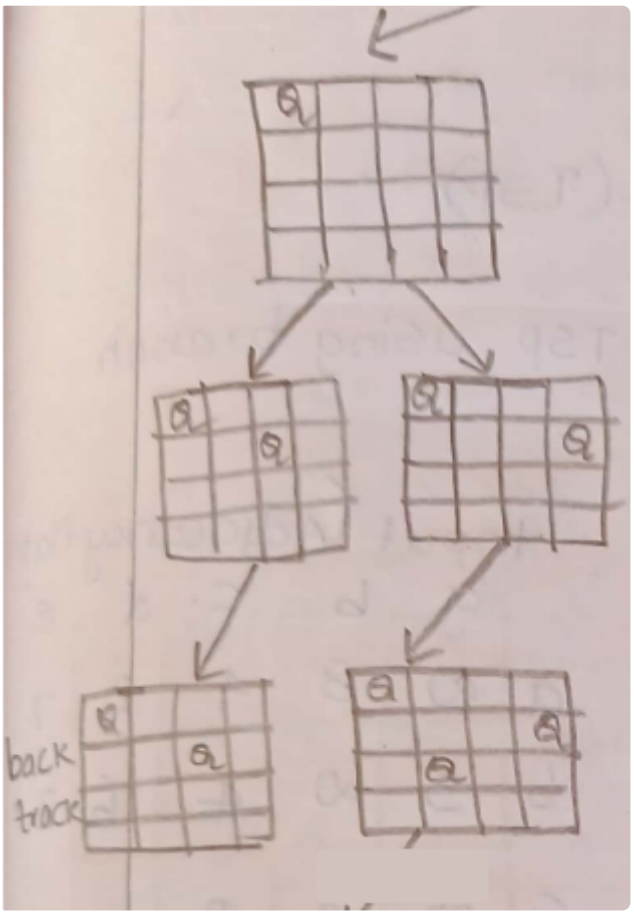
- 

Backtrack 1

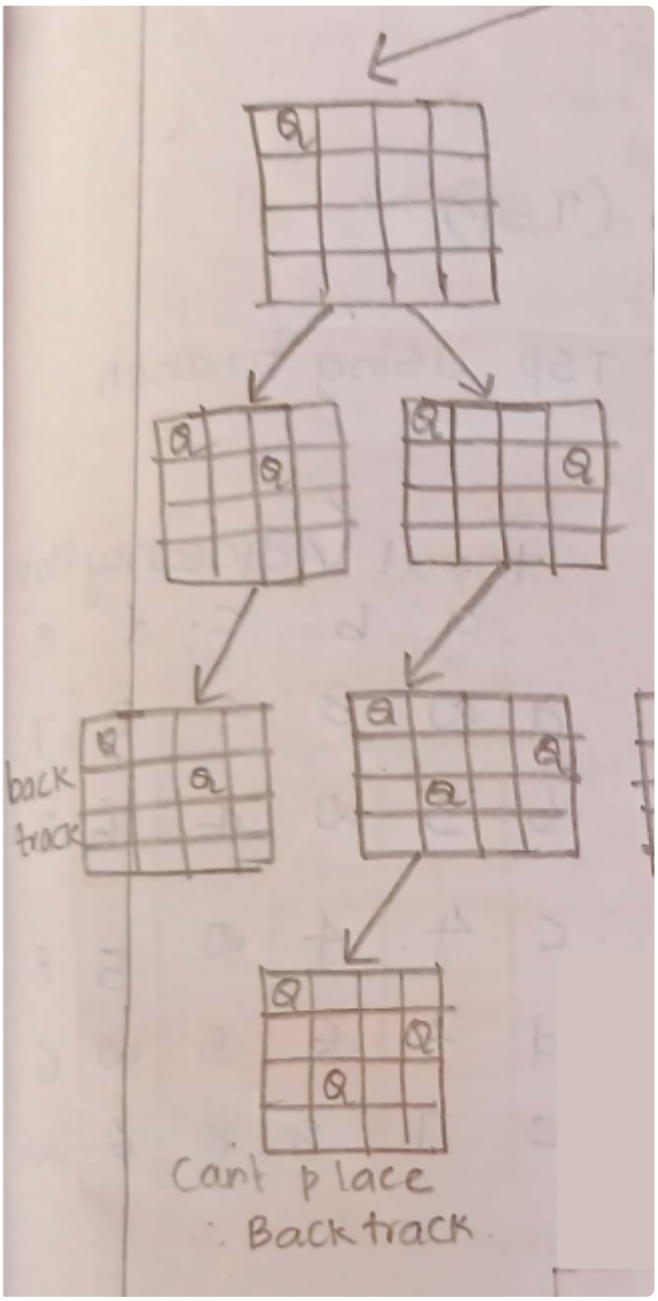
After backtracking, we got to the previous step to find another place for the 2nd queen
Taking 4th column for the 2nd queen



- Trying to place 3rd queen
  - 3rd queen cant be placed in 1st
  - Placing queen in 2nd column, no attacks there

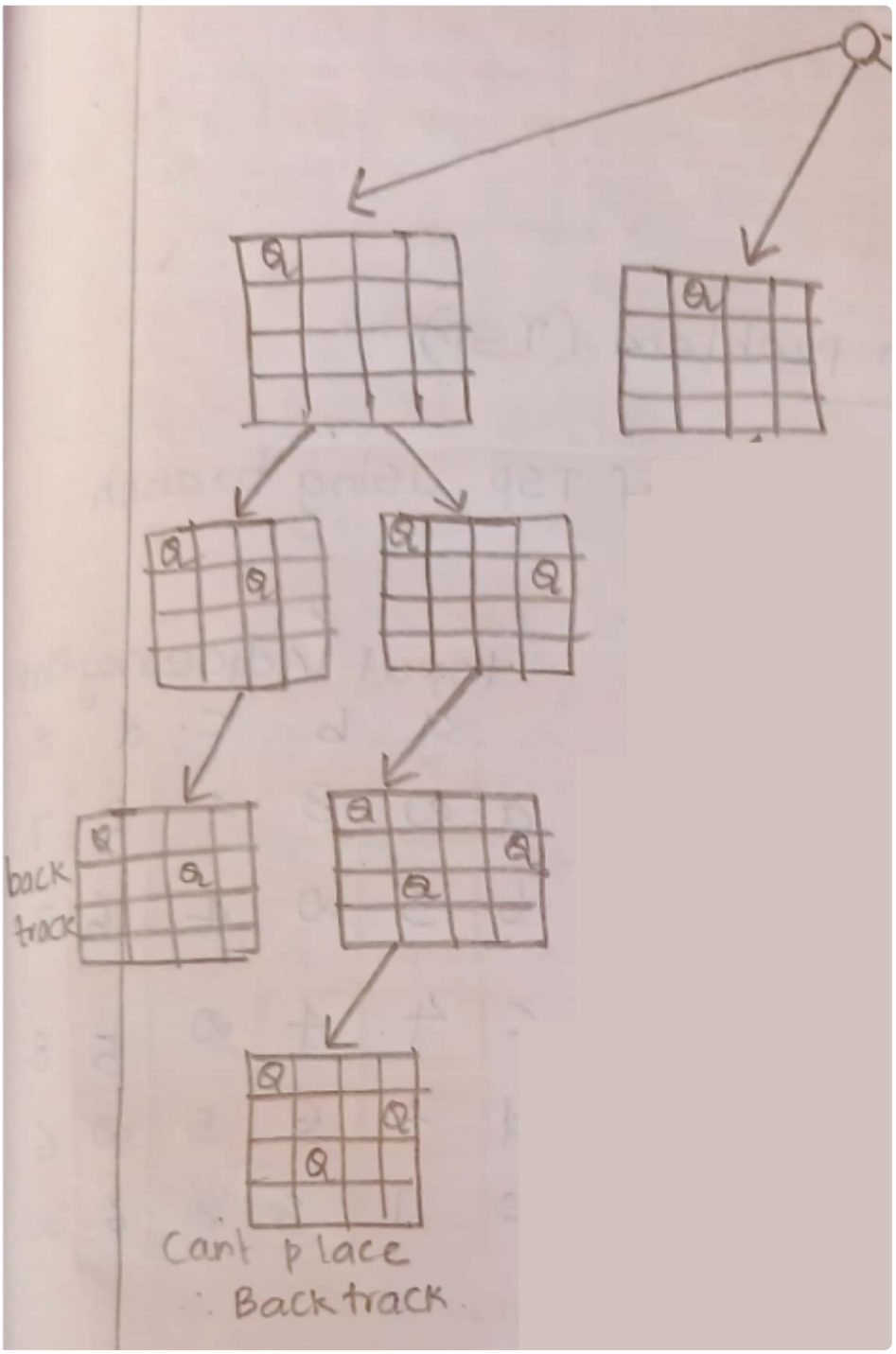
- 

- Trying to place 4th queen
  - Cant place the 4th queen anywhere, because all columns are attackable by the other queens

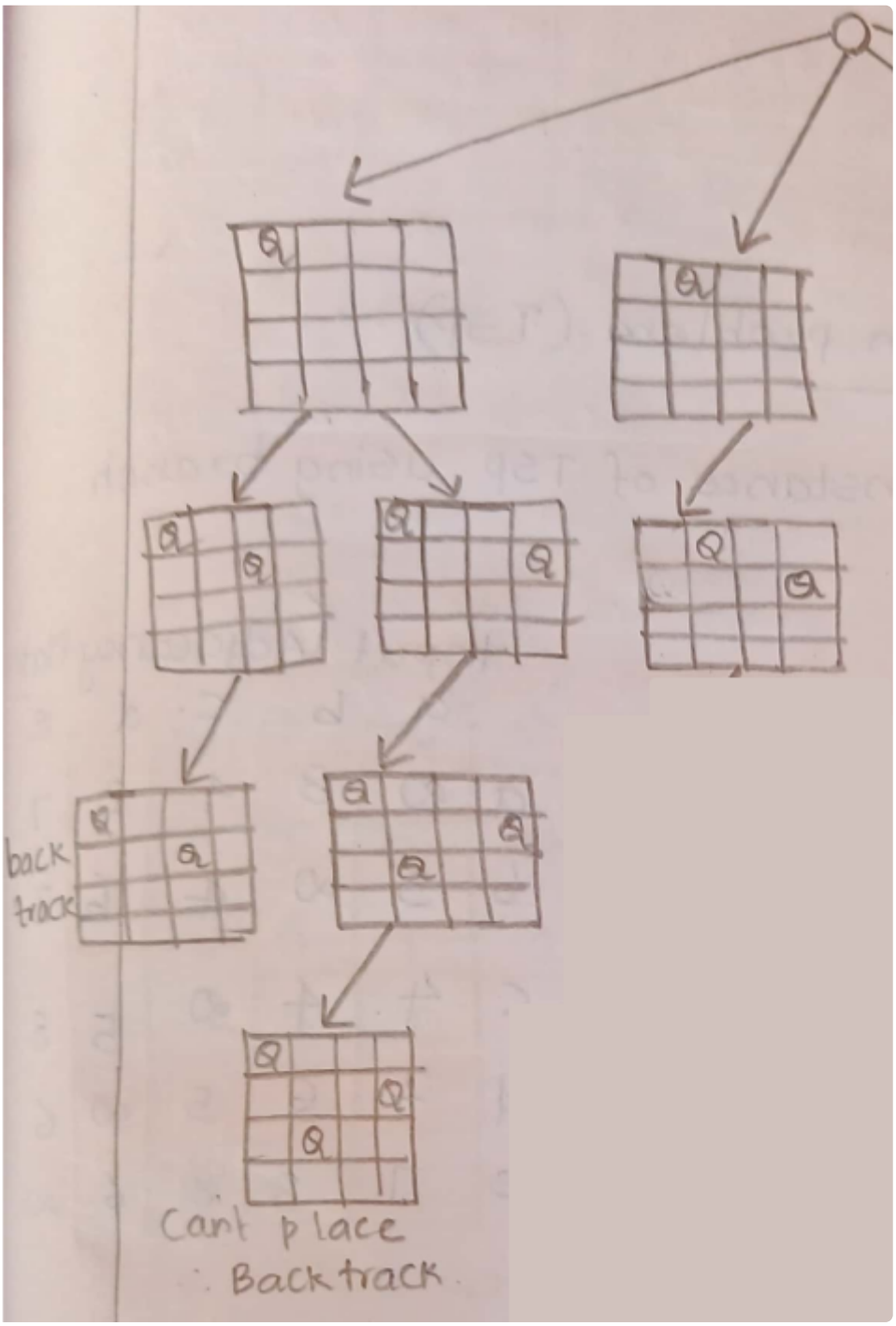
- 

Backtrack 2

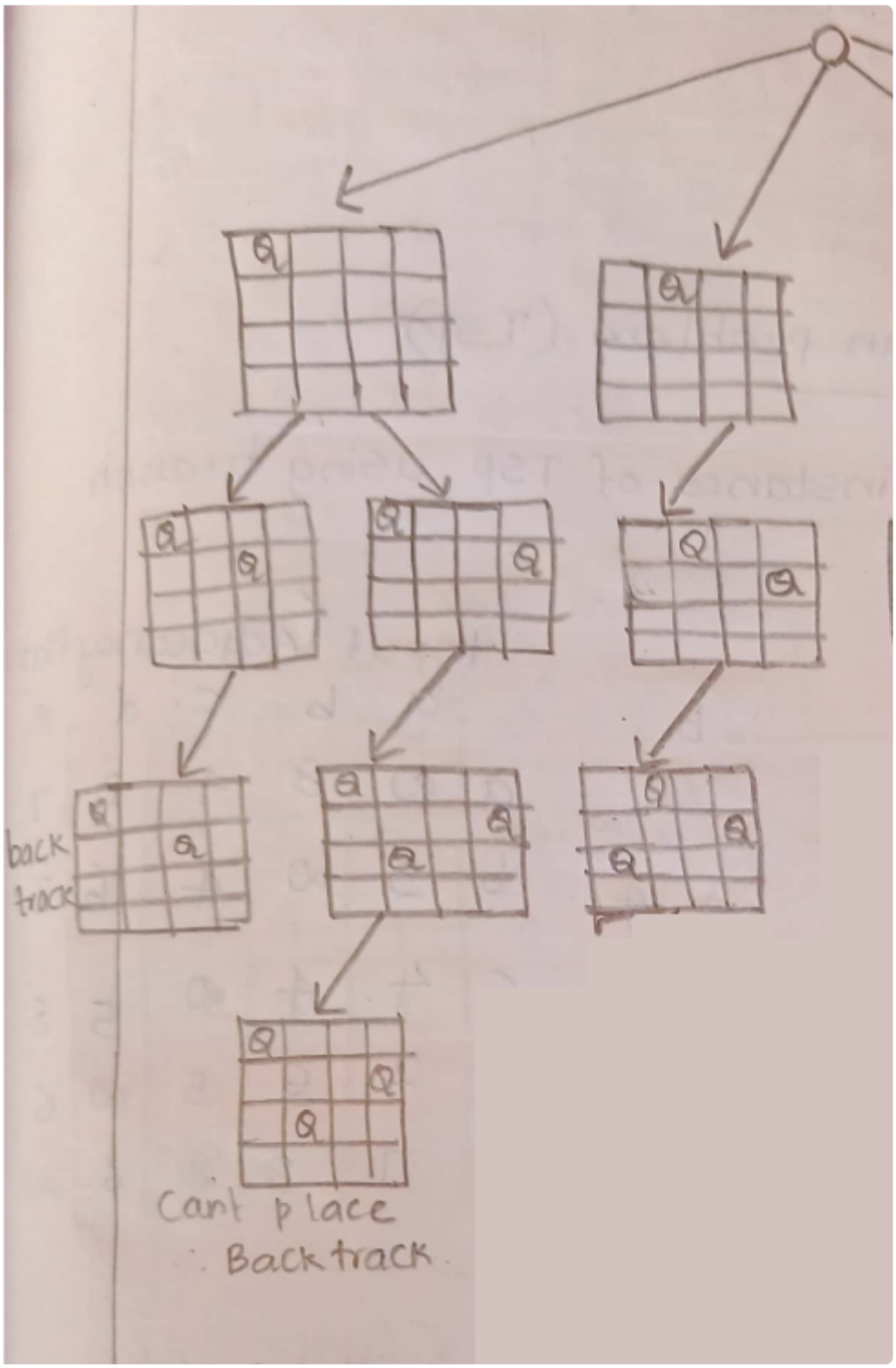
- Going all the way back to the first queen
- Changing position of first queen to 2nd column

- 

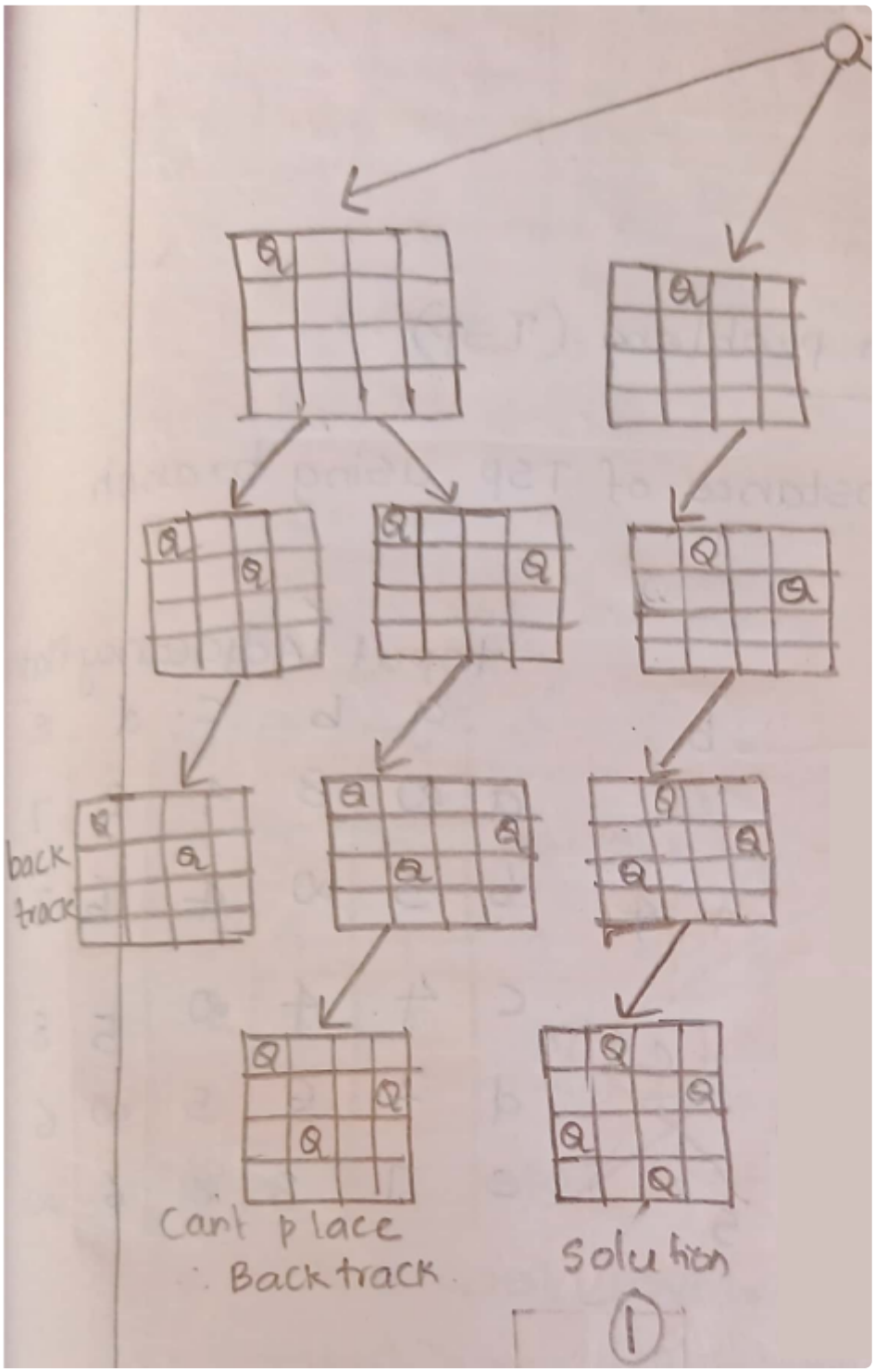
- Choosing 2nd queen position
  - 4th position doesnt have any attacks

- 

- Choosing 3rd queens position
  - First column can be chosen, it cant be attacked by queen 1 or 2



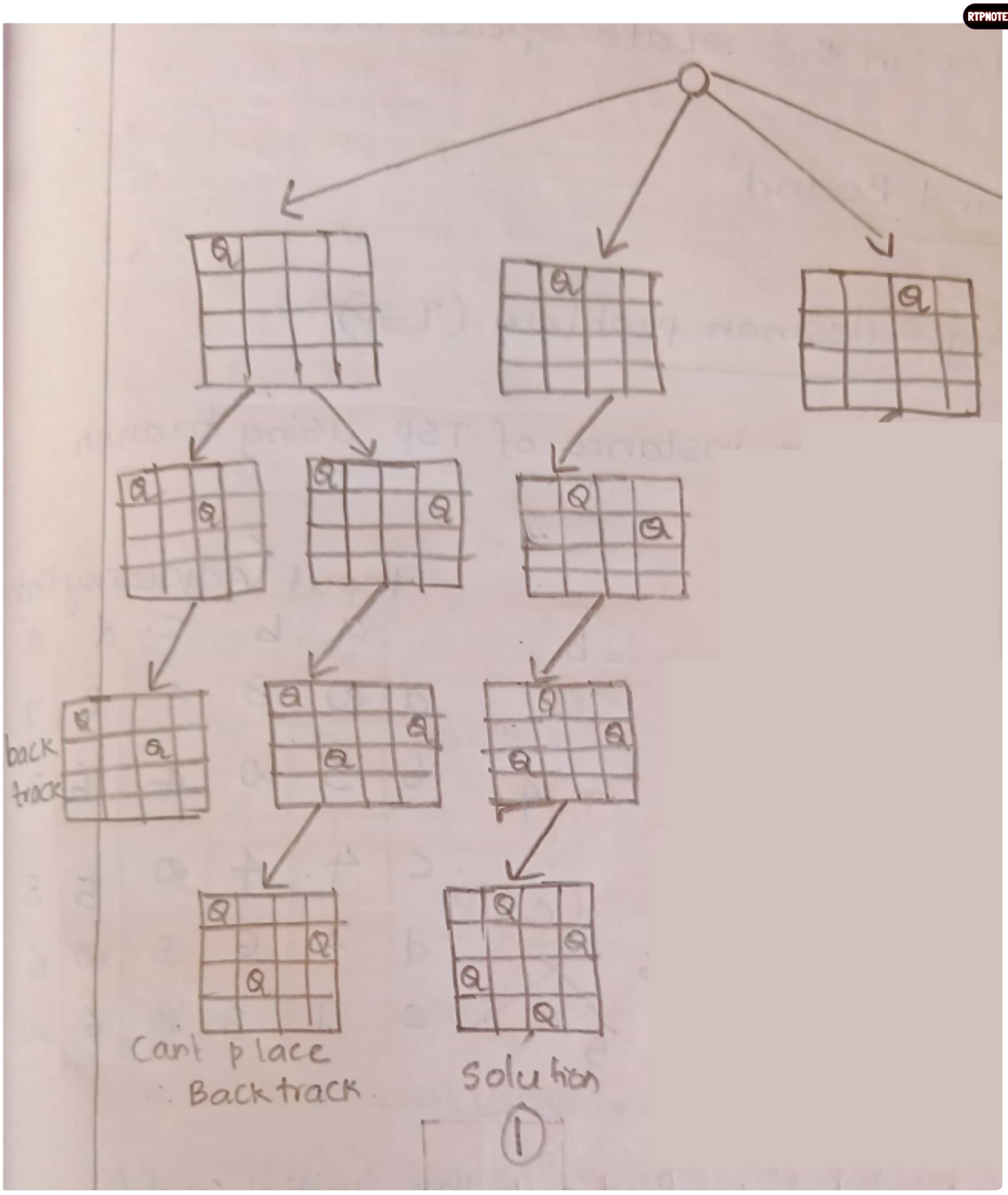
- Choosing 4th queens position
  - 3rd column can be chosen, it cant be attacked by any of the queens



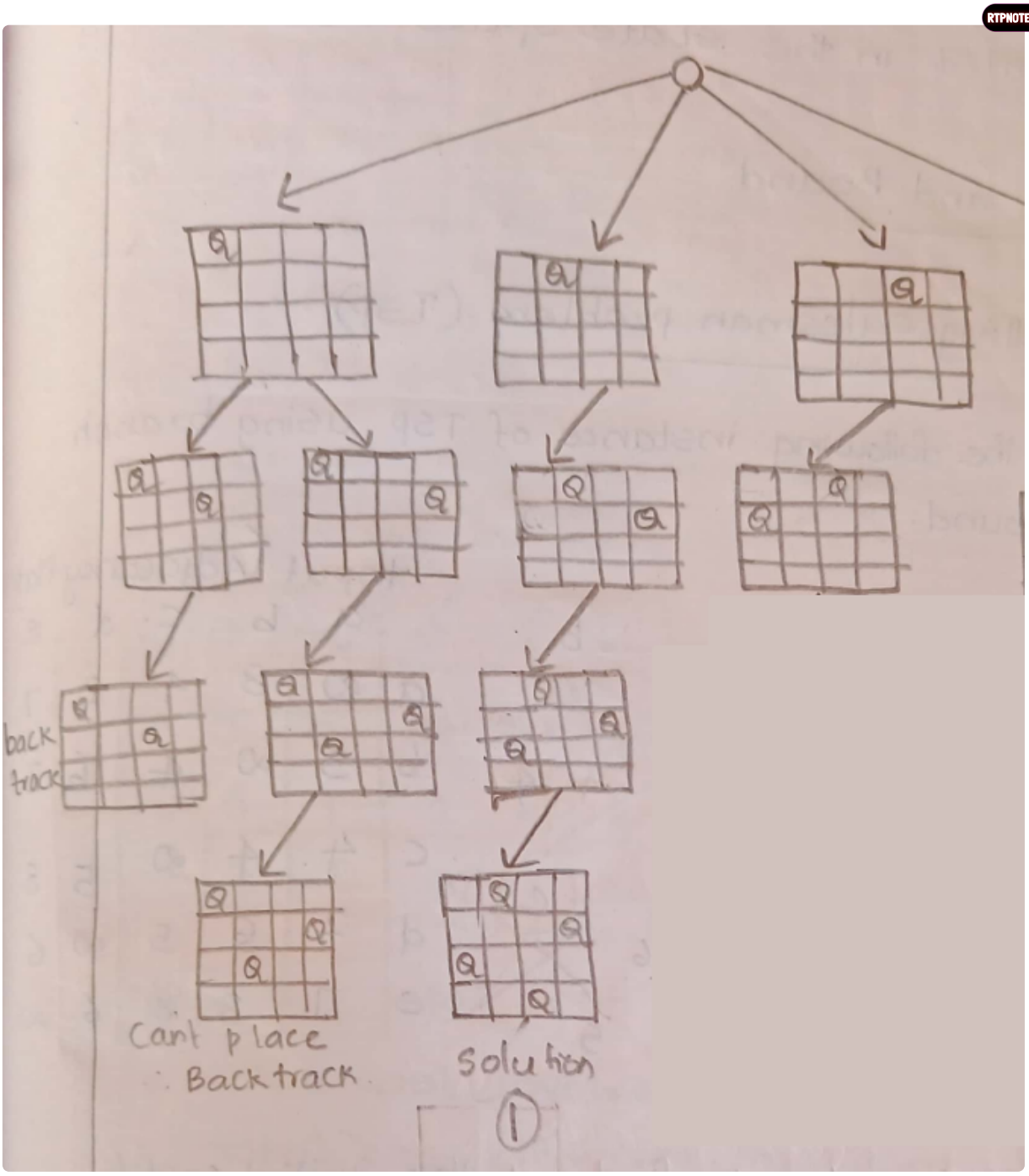
- We got our first solution

Solution 2

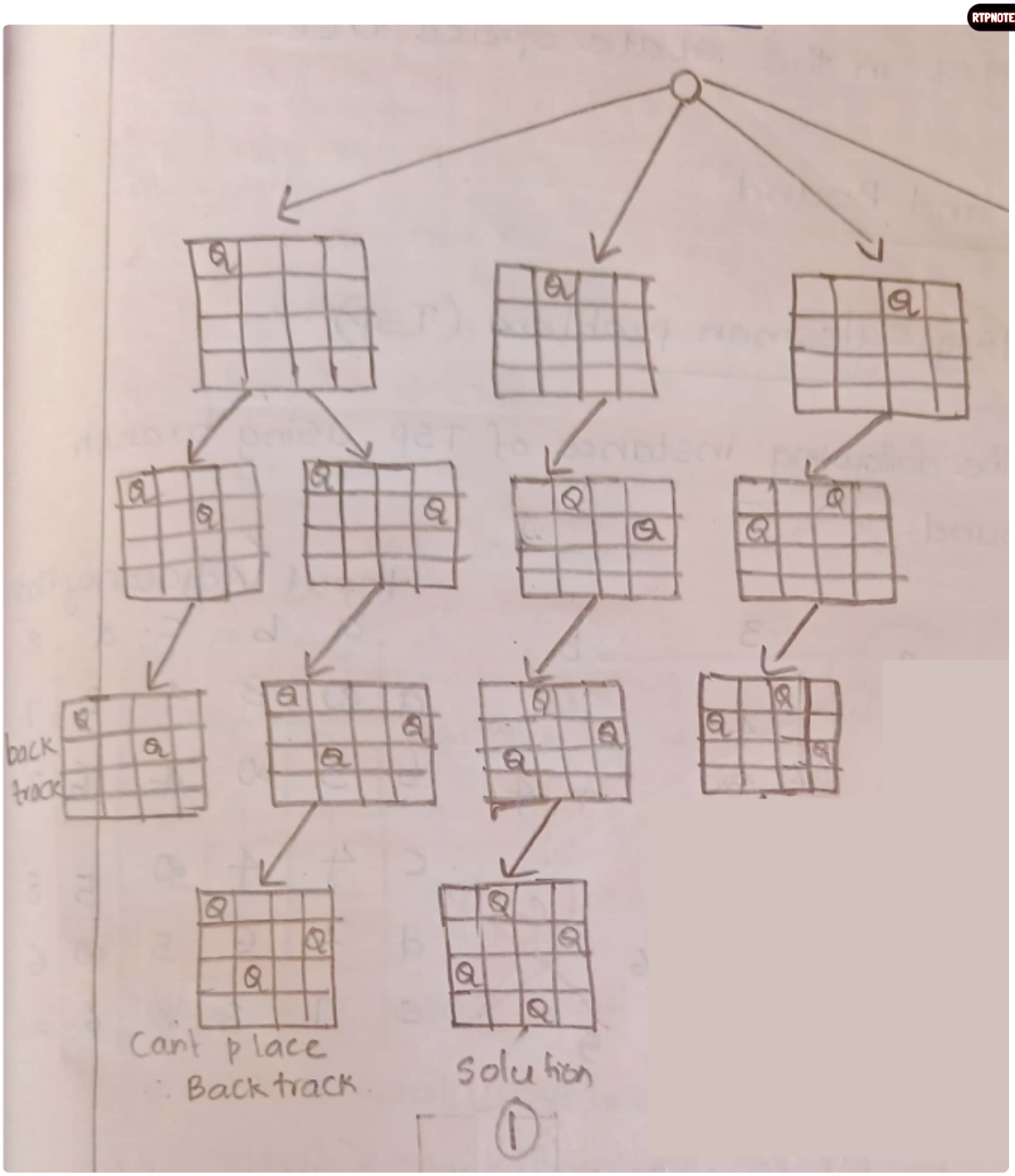
- Lets go back to the first queen, and try the 3rd column

- 

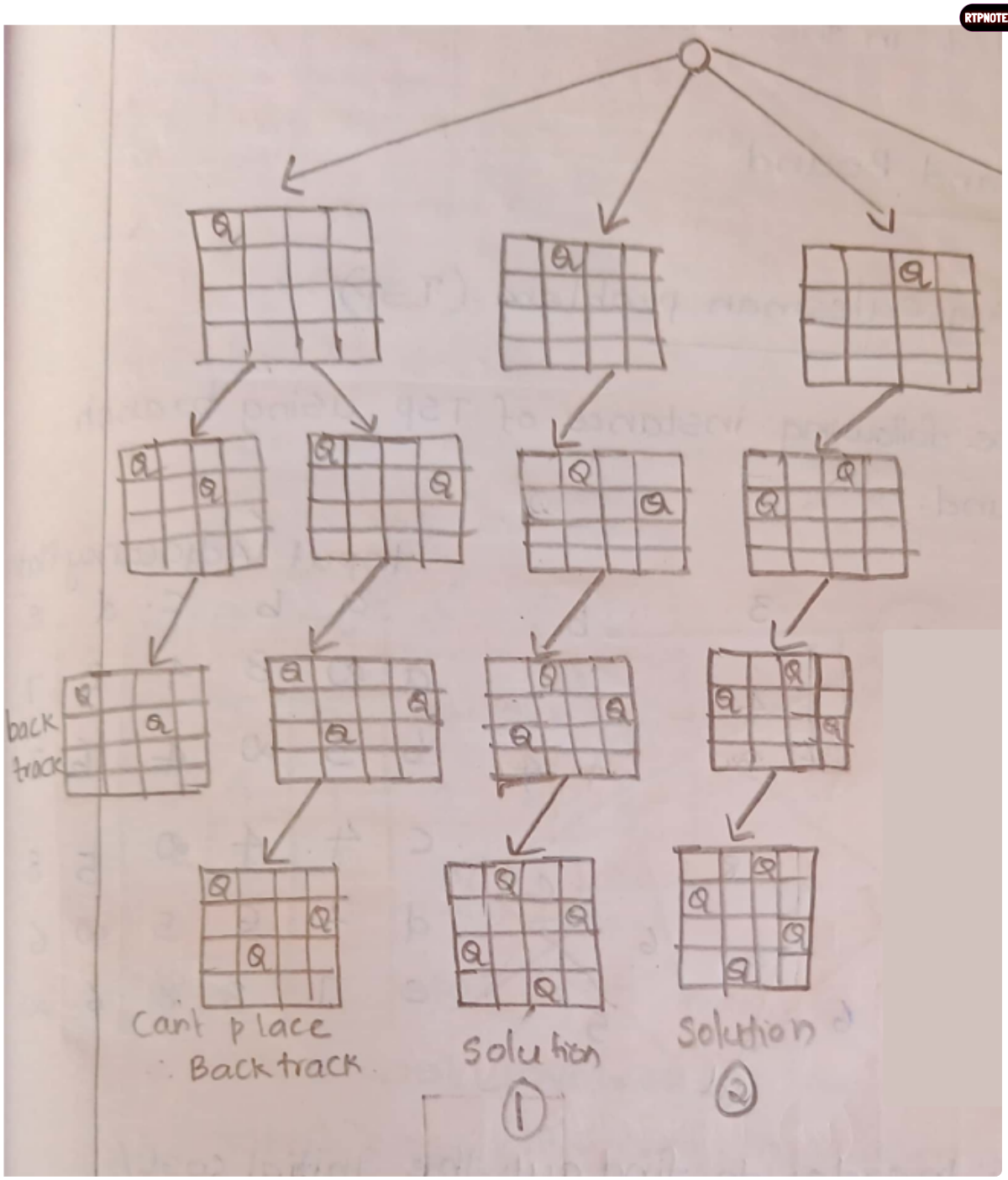
- Finding 2nd queen position
  - First column can be chosen



- Finding 3rd queen position
  - 4th position can be chosen

- 

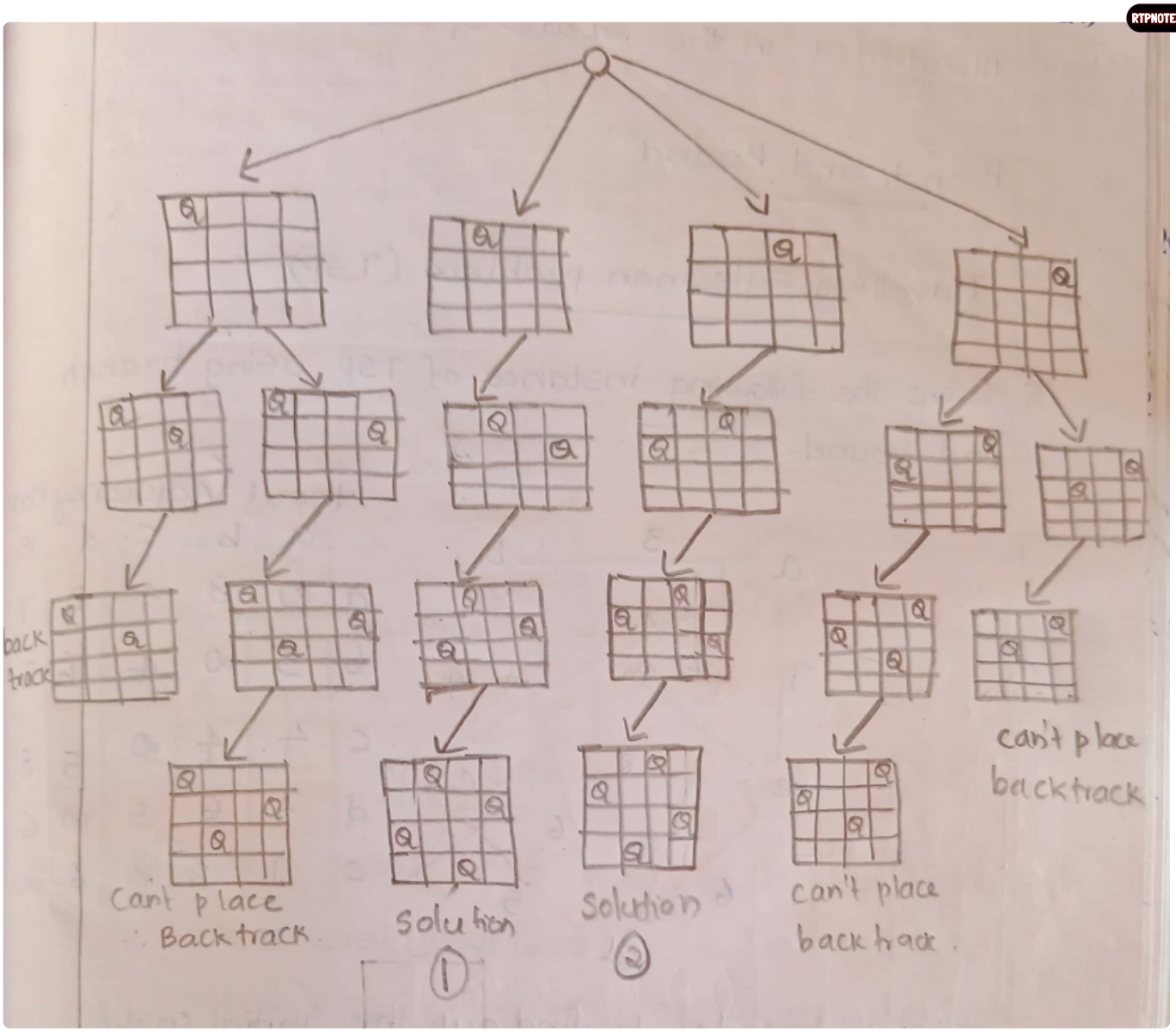
- Finding 4th queen position



- We got the 2nd solution

Solution 3

- We can try to find the 3rd solution
- Placing the first queen in 4th position
- Doing similar steps, we got 2 backtracks
- Theres no other solution



8. Travelling salesman problem

Branch and Bound

Branch and Bound is a problem-solving technique used in computer science to find optimal solutions by systematically exploring all possible solutions, but efficiently.

E-node Concept:

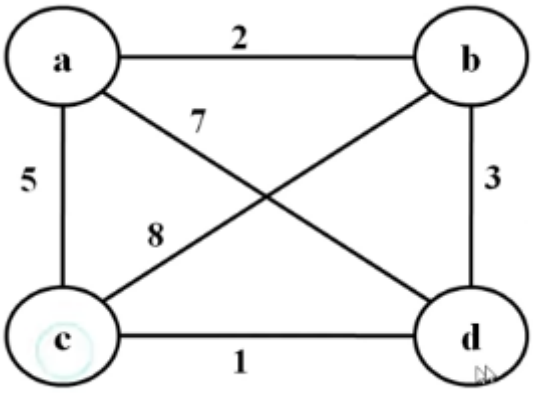
- Imagine you are exploring a tree. The nodes you are currently looking at are called **E-nodes**. They remain E-nodes until you finish exploring them (i.e., until they are "dead").

Strategies for Exploring the Tree:

- **Breadth-First Search (BFS)**:
  - Think of BFS like a queue at a store where the first person in line gets served first (FIFO: First In, First Out). In BFS, you explore all nodes at one level before moving to the next level.
- **Depth-First Search (D-Search)**:
  - D-Search is like a stack of plates where you always take the top plate first (LIFO: Last In, First Out). In D-Search, you explore as far down one path as possible before backtracking.
  Improved Strategy: Least Cost Search (LC Search):
- To speed up finding the best solution, we use a **ranking function** to prioritize which node to explore next.
- Each node gets a score from this function, and the node with the lowest score (i.e., the "least cost") is explored next.
- Both BFS and D-Search can be seen as special cases of LC-Search:
  - In BFS, all nodes at the same level have the same priority.
  - In D-Search, the most recently encountered node has the highest priority.
- When LC-Search is combined with a **bounding function** (which helps cut off unpromising paths early), it becomes **LC Branch and Bound Search**.

Travelling Salesman Problem using Branch and Bound

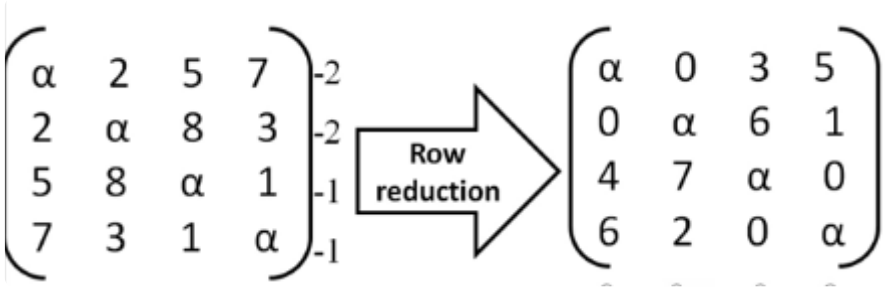
- Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point
- **Consider this graph**



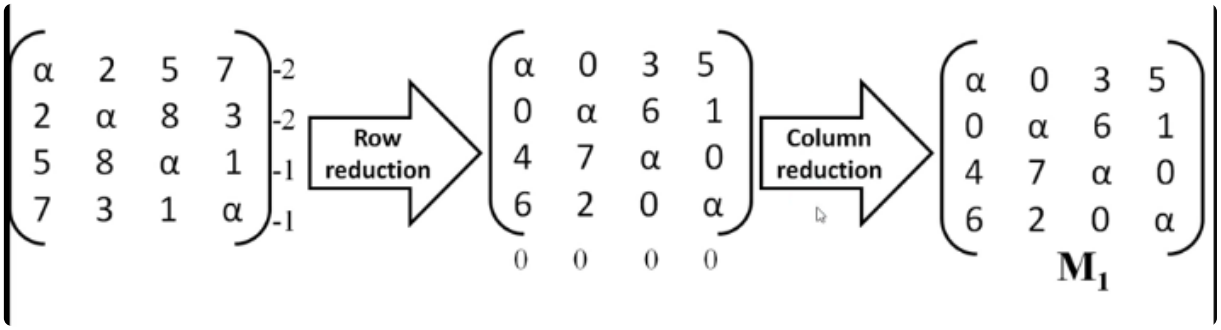
- **This is the adjacency matrix**

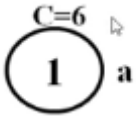
	a	b	c	d
a	α	2	5	7
b	2	α	8	3
c	5	8	α	1
d	7	3	1	α

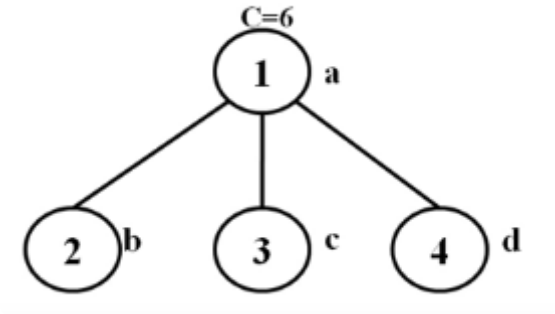
-
- Setting infinity where there is no edge like infinity-infinity

- **Perform row reduction**
  - Consider the first row
    - infinite, 2,5, and 7
    - Here 2 is the minimum number
    - Subtract 2 from all the numbers
  - 2nd row
    - 2, infinite, 8, 3
    - Minimum is 2
    - Subtracting 2
  - 3rd row
    - 5,8,infinite,1
    - Minimum is 1
    - subbing 1
  - 4th row
    - 7,3,1,infinity
    - Subtracting 1
  - After reduction
    - 

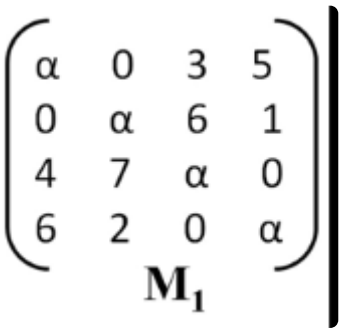
- **Perform Column reduction**
  - All 4 columns have minimum value 0

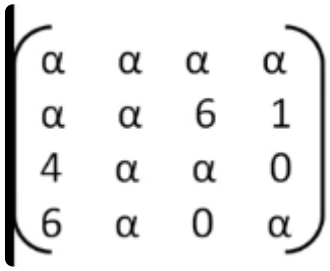
- So 0 is subtracted
- 

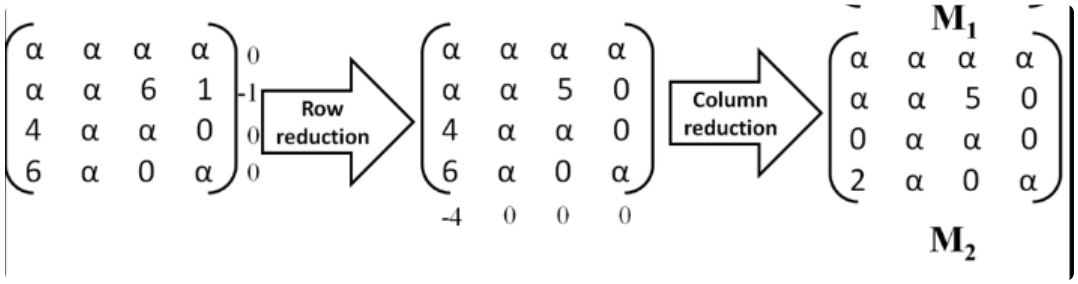
- **Now we need to count the total reduction**
  - First count the rows,
    - We have subtracted 2, 2, 1 and 1
  - Count the columns
    - All are 0s
  - Adding them up
    - 2 + 2 + 1 + 1 = 6
- **Total Reduction = 6**
- **Next, Root node of state space tree is generated**
  - Cost is set as the total reduction = 6
  - 

  - M1 is the matrix for node 1
- **Now Node 1 is the E node, Generate the child nodes of node 1**
  - From the graph, We can see that, a can access b, c and d
  - 

- **Now we need to calculate the cost and matrix of 2, 3 and 4**
  - **Calculating matrix and cost of node 2**
    - Our initial Matrix

- 

  - Here a = 1, b = 2
  - **Set row a and column b elements to infinity**
    - Row 1 and column 2 elements are set to infinity
  - Set M1[b,a] = infinity
    - M1[2,1] = infinity
  - We get the following matrix
    - 

  - Perform Row and column reduction
    - 

  - Cost reduced = 1 + 4 = 5
  - M2 is the matrix for node 2
  - Cost of node 2 = Cost of node 1 + M1[a,b] + Cost Reduced (Node 2) = 6 +0 +5 = 11
- Similarly finding node 3

- Find the matrix and cost of node 3
  - Set row a and column c elements are α
  - Set $M_1[c, a] = \alpha$

$$M_1 = \begin{pmatrix} \alpha & 0 & 3 & 5 \\ 0 & \alpha & 6 & 1 \\ 4 & 7 & \alpha & 0 \\ 6 & 2 & 0 & \alpha \end{pmatrix}$$

$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & 1 \\ \alpha & 7 & \alpha & 0 \\ 6 & 2 & \alpha & \alpha \end{pmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ -2 \end{matrix} \xrightarrow{\text{Row reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & 1 \\ \alpha & 7 & \alpha & 0 \\ 4 & 0 & \alpha & \alpha \end{pmatrix} \xrightarrow{\text{Column reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & 1 \\ \alpha & 7 & \alpha & 0 \\ 4 & 0 & \alpha & \alpha \end{pmatrix} = M_3$$

$$0 \quad 0 \quad 0 \quad 0$$

  - Cost Reduced = 2
  - M_3 is the matrix for node 3
  - Cost of node 3 = Cost of node 1 + $M_1[a, c]$ + r = 6 + 3 + 2 = **11**

- Similarly finding node 4
  - Find the matrix and cost of node 4
    - Set row a and column d elements are α
    - Set $M_1[d, a] = \alpha$

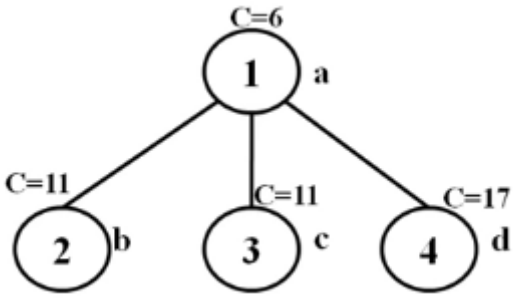
$$M_1 = \begin{pmatrix} \alpha & 0 & 3 & 5 \\ 0 & \alpha & 6 & 1 \\ 4 & 7 & \alpha & 0 \\ 6 & 2 & 0 & \alpha \end{pmatrix}$$

$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & 6 & \alpha \\ 4 & 7 & \alpha & \alpha \\ \alpha & 2 & 0 & \alpha \end{pmatrix} \begin{matrix} 0 \\ 0 \\ -4 \\ 0 \end{matrix} \xrightarrow{\text{Row reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & 6 & \alpha \\ 0 & 3 & \alpha & \alpha \\ \alpha & 2 & 0 & \alpha \end{pmatrix} \xrightarrow{\text{Column reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & 6 & \alpha \\ 0 & 1 & \alpha & \alpha \\ \alpha & 0 & 0 & \alpha \end{pmatrix} = M_4$$

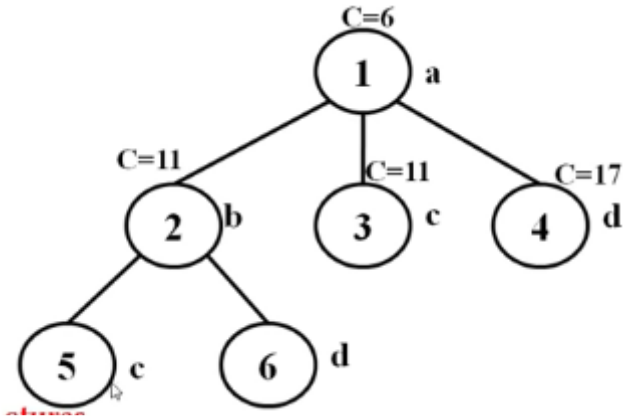
$$0 \quad -2 \quad 0 \quad 0$$

    - Cost Reduced = 6
    - M_4 is the matrix for node 4
    - Cost of node 4 = Cost of node 1 + $M_1[a, d]$ + r = 6 + 5 + 6 = **17**

- **Now the state space tree is**



- Now the Live nodes are 2,3 and 4. Minimum cost is node 2 and 3, Choosing one node (Node 2) as next E-Node
- Generate Child node for node 2



- Finding cost and matrix for 5

- Find the matrix and cost of node 5
  - Set row b and column c elements are α
  - Set $M_2[c, a] = \alpha$

$$M_2 = \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 5 & 0 \\ 0 & \alpha & \alpha & 0 \\ 2 & \alpha & 0 & \alpha \end{pmatrix}$$

$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 2 & \alpha & \alpha & \alpha \end{pmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ -2 \end{matrix} \xrightarrow{\text{Row reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & \alpha \end{pmatrix} \xrightarrow{\text{Column reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & 0 \\ 0 & \alpha & \alpha & \alpha \end{pmatrix}$$

$$0 \quad 0 \quad 0 \quad 0$$

$$M_5$$

  - Cost Reduced = 2
  - M_5 is the matrix for node 5
  - Cost of node 5 = Cost of node 2 + $M_2[b, c]$ + r = 11 + 5 + 2 = **18**

- Finding cost and matrix for 6

- Find the matrix and cost of node 6
  - Set row b and column d elements are α
  - Set $M_2[d, a] = \alpha$

$$M_2 = \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & 5 & 0 \\ 0 & \alpha & \alpha & 0 \\ 2 & \alpha & 0 & \alpha \end{pmatrix}$$

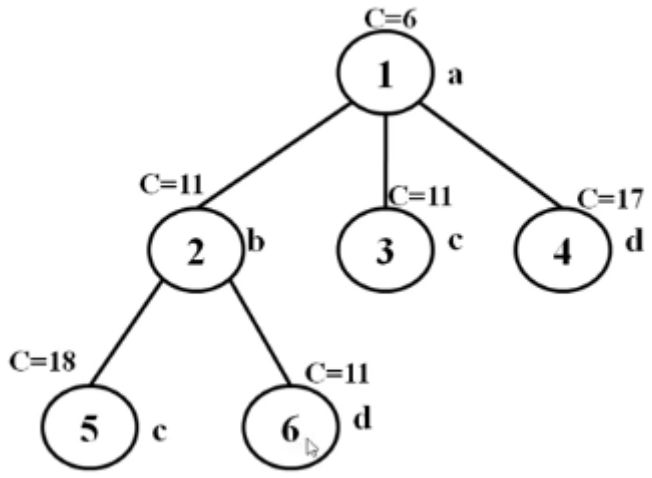
$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & \alpha \end{pmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} \xrightarrow{\text{Row reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & \alpha \end{pmatrix} \xrightarrow{\text{Column reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & \alpha \end{pmatrix}$$

$0 \quad 0 \quad 0 \quad 0$

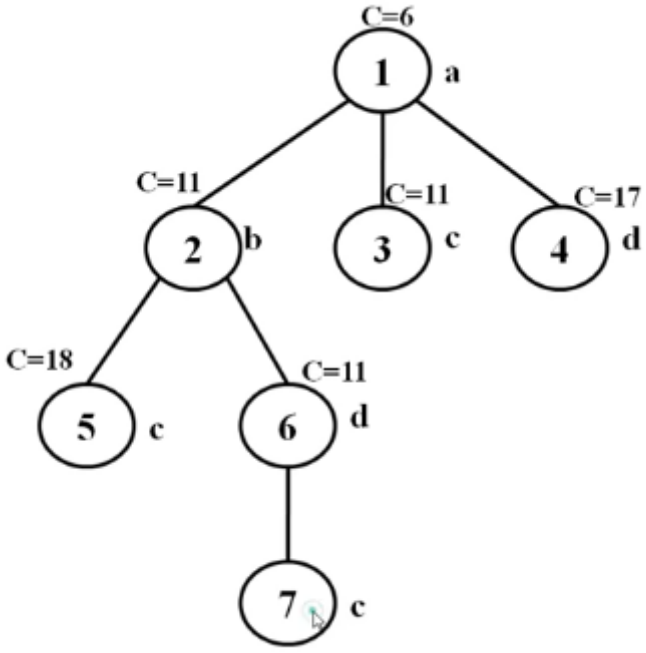
M_6

  - Cost Reduced = 0
  - M_6 is the matrix for node 6
  - Cost of node 6 = Cost of node 2 + $M_2[b, d]$ + r = 11 + 0 + 0 = **11**

- Now the state space tree is



- Now the live nodes are 3, 4,5 and 6
- Node 6 is the next E node
- Generating child node of node 6



- Calculating matrix and cost of node 7

- Find the matrix and cost of node 7
  - Set row d and column c elements are α
  - Set $M_6[c, a] = \alpha$

$$M_6 = \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ 0 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 0 & \alpha \end{pmatrix}$$

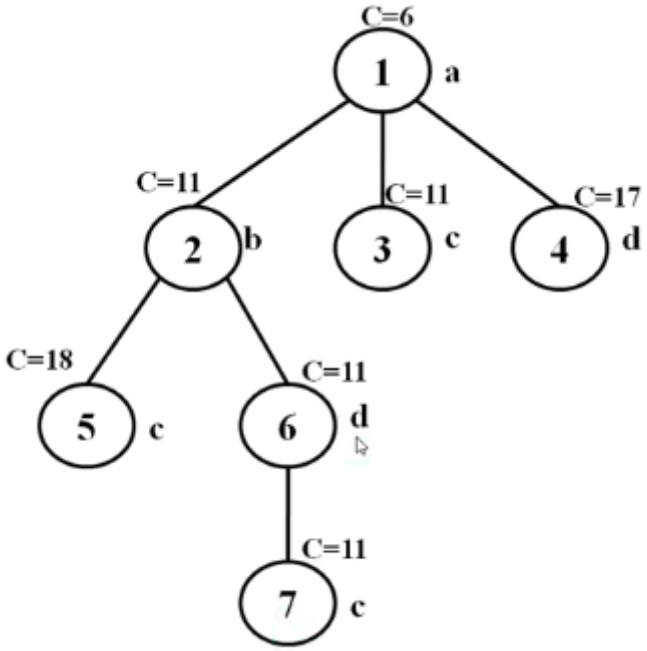
$$\begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \end{pmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} \xrightarrow{\text{Row reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \end{pmatrix} \xrightarrow{\text{Column reduction}} \begin{pmatrix} \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \\ \alpha & \alpha & \alpha & \alpha \end{pmatrix}$$

$$\begin{matrix} 0 & 0 & 0 & 0 \end{matrix}$$

$$M_7$$

  - Cost Reduced = 0
  - M_7 is the matrix for node 7
  - Cost of node 7 = Cost of node 6 + $M_6[d, c]$ + r = 11 + 0 + 0 = **11**

- Now the state space tree is



- **This is our final answer**