MODULE - 3

7/12/21

Backtracking

Backtracking algorithm is applicable to the wide range of problems. The key point for the backtracking algorithm is a binary choice that means Yes or no. Whenever the backtracking have the choice No, that means the algorithm has encountered a deadend and it backtracks one step and tries a different path for choices. The backtracking resembles a depth first search tree in a directed graph. Were graph is either a tree or at least it does not have any cycles. The solution for the problem according to the backtracking can be represented as implicit graph on which backtracking performs an intelligent DFS so as to provide one or all possible solutions to the given problem. The whole task is accomplished by maintaining partial solution as the search proceeds. It can be seen that such partial solutions bind the function where a complete soln to the pablm can be obtained. Initially no solution to the pablm is known. When search proceeds a new element is added to the partial solution which in turn decreases the remaining possibilities for a complete soln.

The search is successful if a complete soln for the problem is defined, in this case Search either terminates or continue to search for all

other possible solns IP at any stage, the search is unsuccessful that means, the partial soln constructed so far are unable to define the complete soln, then the search backtracks one step. It should be noted that the element from the partial soln is also removed on backtracking.

In many applications of the backtracking method, the desire soln is expressible as an n tuple $\{x_1, x_2, \ldots, x_n\}$ where the x_i are chosen from finite set S_i of the pblm to be solved calls for finding on vector that maximises a criterian function $P(x_1, x_2, \ldots x_n)$.

Suppose m_i is the size of the set S_i, then there are $m = \{m_1, m_2 \ldots m_n\}$ that are possible candidates for satisfying the function P. The basic idea is to build up the soln vector, one component at a time and to use modifying criterian function $P(x_1, x_2, \ldots, x_i)$, whether the vector being formed as any chance of success.

•) Constraints can be divided into two categories:

1. Explicit Constraints: are rules that restrict each x_i to take m values only from a given set.

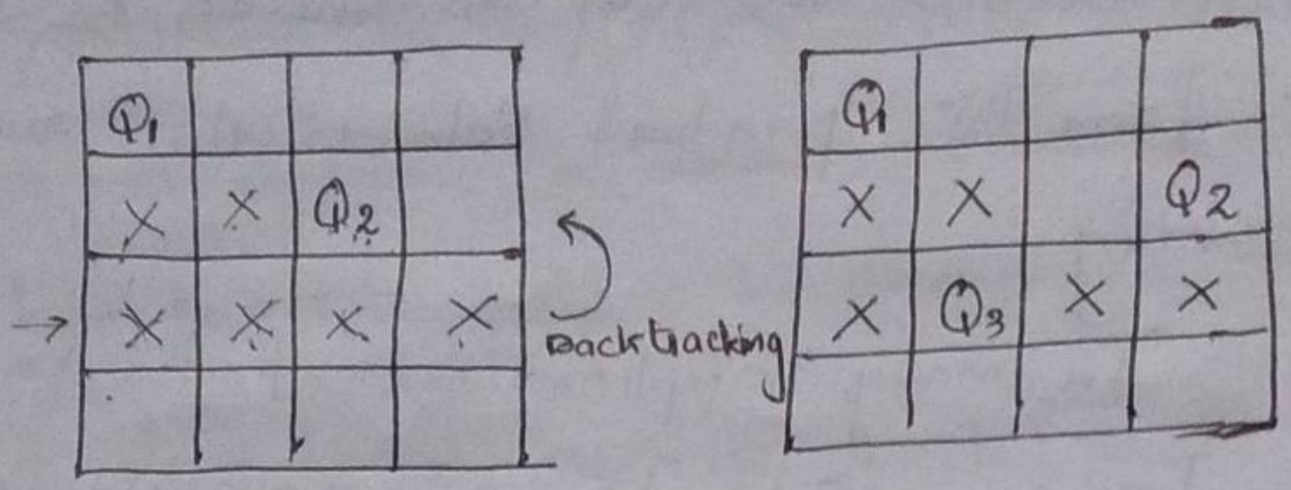
2. Implicit Constraints: are rules that determine which of the tuples in the soln makes satisfies the criterian function

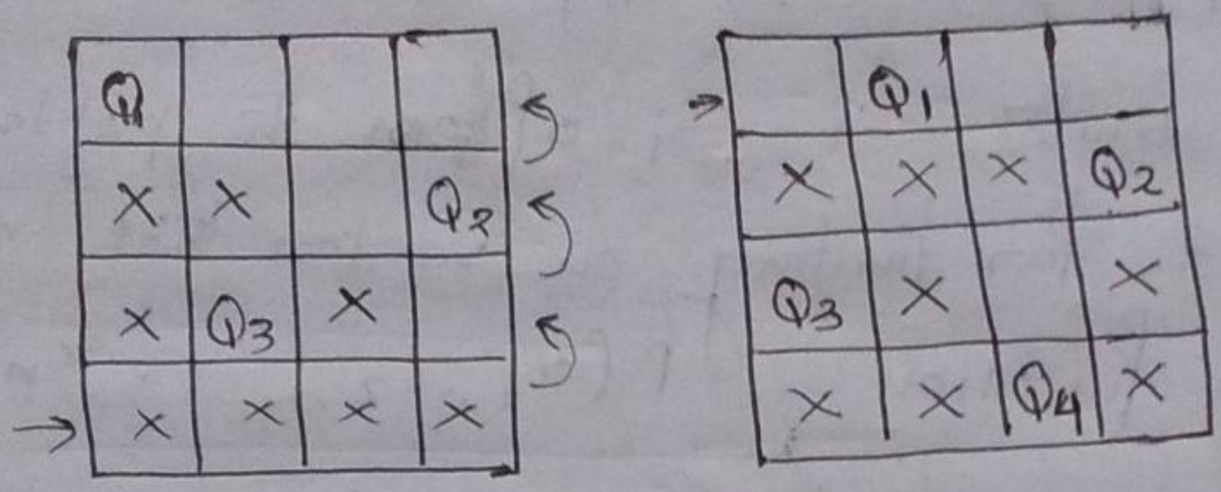
09/12/21
Thursday

N Queens Problem

4 Queens

(Q_1, Q_2, Q_3, Q_4)





(2, 4, 1, 3)

The famous combinatorial problem, N Queens problem is to place N queens on an NxN chess board that no two queens attack each other by being in the same row, column or diagonal. It can be seen that the for N = 1, the problem has a trivial solution and no soln exist for N = 2 and N = 3.

4 Queens Problem

Given a 4x4 chess board, let us number the rows and columns of chess board 1, to 4 since we have to place 4 queens on a chess board such that no two queens attack each other. We number this as Q_1, Q_2, Q_3, Q_4. Each queen must be placed

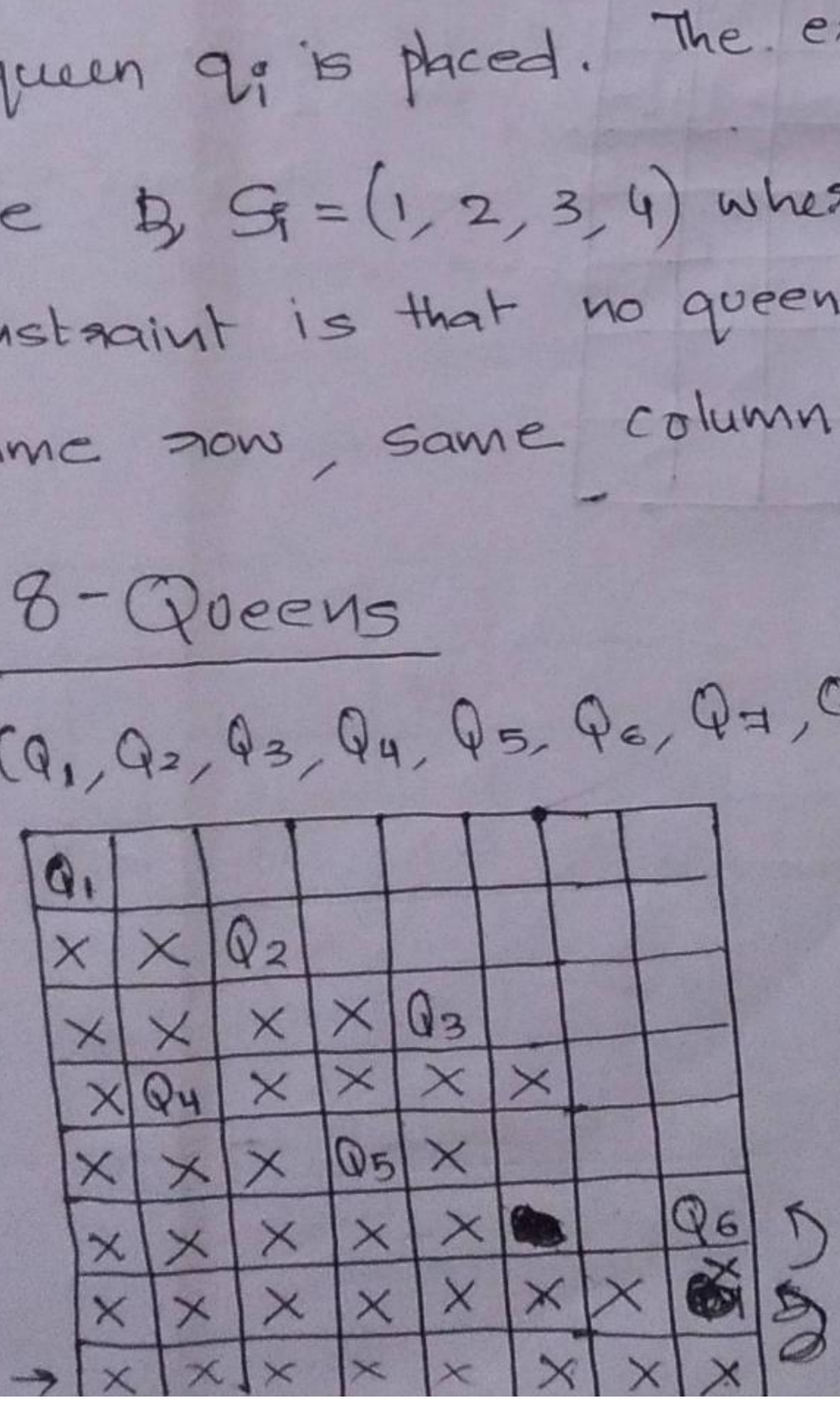
on a diff row, so we place queen i on row i.

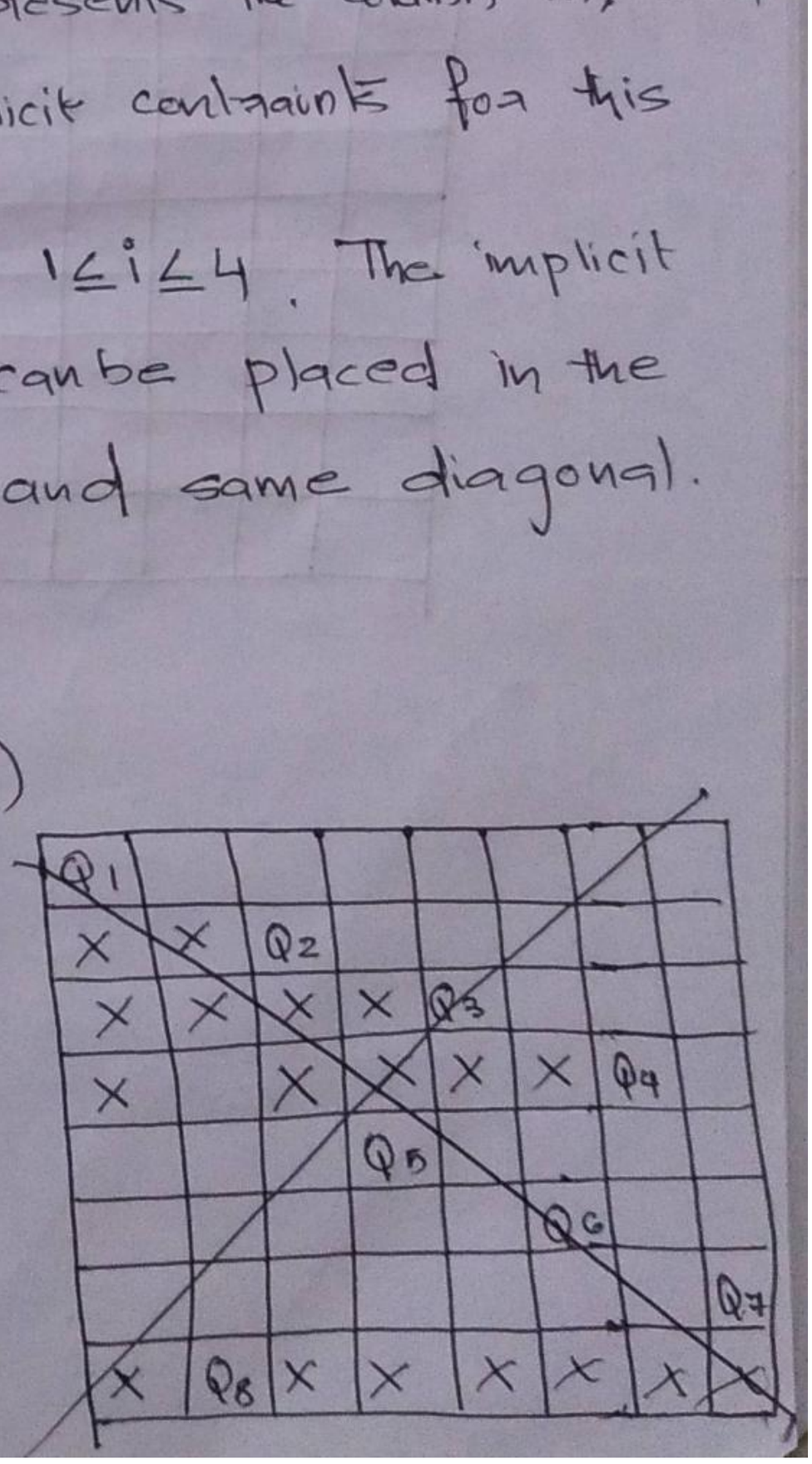
First we place queen Q_1 in the very first acceptable position which is (1,1). The first acceptable position for queen 2, Q_2 is (2,3). But later this position proves to be deadend as no position is left for placing Q_3 safely. So we backtrack one step and place the queen Q_2 in (2,4), the next possible location. Each node describes its partial solution, one possible solution is shown above. For other solutions, the whole method is repeated for the whole partial soln.

It can be seen that all the solns to the 4 queens problem can be represented as 4 tuples t_1, t_2, t_3, t_4 where t_i represents the column no. which queen q_i is placed. The explicit constraints for this are $S_i = (1, 2, 3, 4)$ where $1 \le i \le 4$. The implicit constraint is that no queen can be placed in the same row, same column and same diagonal.

8-Queens

$(Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8)$

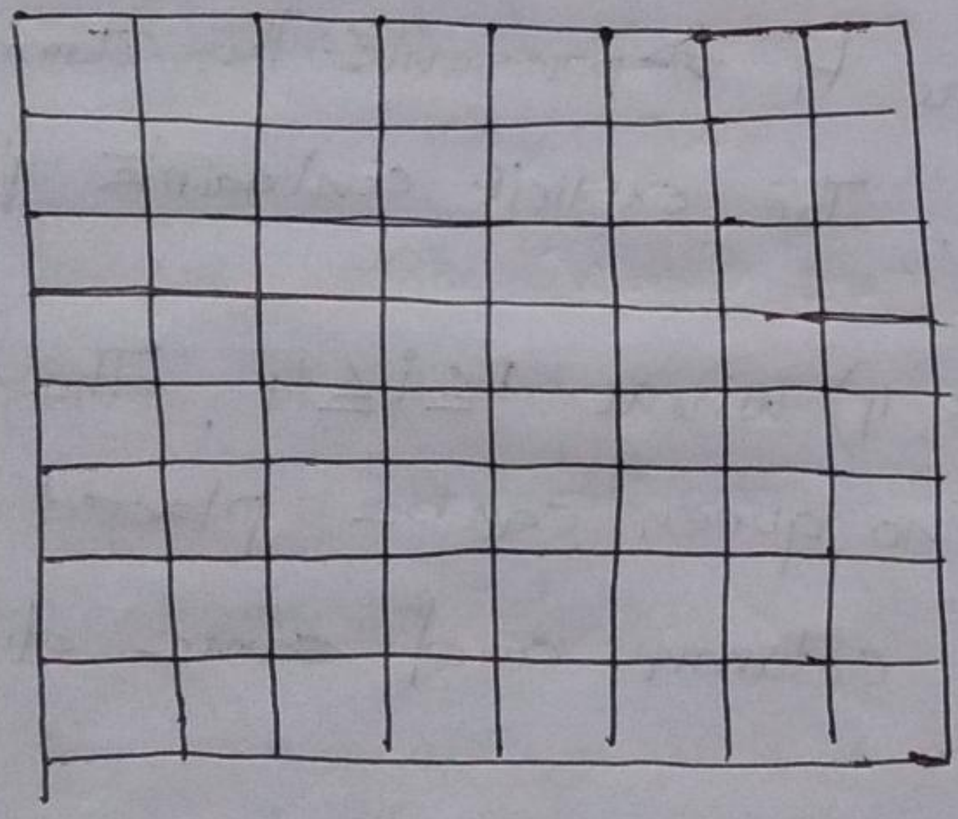




We can problem which need 8 tuples for the representation t_1 to t_8. where t_i represents the column on which queen q_i is placed. The solution phase consist of all 8 factorial combo permutations.

Suppose 2 queens are placed at positions (i,j) and (k,l) then they are in the same diagonal if $i-j = k-l$ or $i+j = k+l$. ie, two queen lie on the same diagonal iff absolute value of $(j-l)$ = absolute value of $(i-k)$, $|j-l| = |i-k|$.

Continuation...



Algorithm

A simple algorithm yielding a solution to the N queens solution to the N queens puzzle for N = 1 or for any N ≥ 4.

Step 1: Divide N by 12. Remember the remainder.

Step 2: Write a list of even numbers from 2 to N in order.

Step 3: If the remainder is 3 or 9, move 2 to the end of the list.

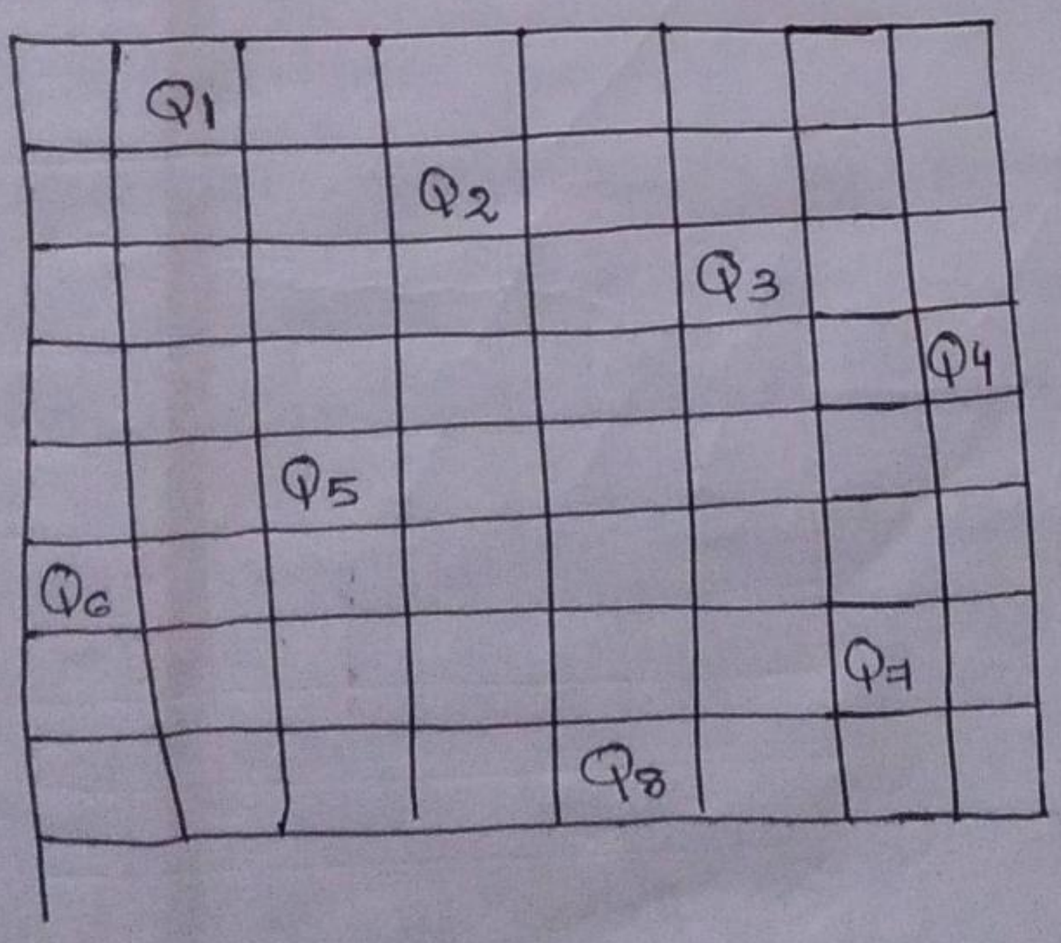
Step 4: Write the odd numbers from 1 to N in order.

If the remainder is 8, switch pairs.

Step 5: If the remainder is 2, switch the places 1 and 3, then move 5 to the N^{th} list.

Step 6: If the remainder is 3 or 9, move 1 and 3 to the end of the list.

Step 7: Place the first column queen in the row with the first row in the list and place the second column queen with the second row in the list and place the 2nd number.



8

2 4 6 8

1 3 5 7

3 1 7 5

Result : 2 4 6 8 3 1 7 5

```
Algorithm NQueens(k,n)
{
   for i:=1 to n do
   {   if Place(k,i) then
       {   x[k]:=i;
           if(k=n) then write (x[1:n]);
           else NQueens(k+1,n);
       }
   }
```

```
Algorithm Place(k,i)
{
   For j:=1 to k-1 do
          if ((x[j]=i)  // Two in the same column
          or (Abs(x[j]-i) = Abs(j-k)))
                // or in the same diagonal.
          then return false;

   return true;
}
```

Sum of Subsets

Suppose we are given n distint positive numbers and we have to find all combinations of these numbers whose sums are m, are called sum of subsets problem. In this problem we have to find a subset s of given set $s = \{s_1, s_2, s_3 \ldots s_n\}$ where the element of set S are n positive integers in such a manner that $s' \in S$ and sum of the elements of the subsets is equal to a positive integer m. If a given set $n = \{1, 2, 3, 4\}$ and $m = 5$, then $s' = \{1, 4\}$ or $s' = \{2, 3\}$

The sum of subsets problem can be solved using the backtracking approach. In this, implicit tree is created which is a binary tree, the route of the tree is selected in such a way that no decision is get taken on any input. We assume that the elements of a given set are arranged in an increasing order.

The left child of the root node indicates that we have to include s_1 from set S and the right child of the root node indicates that we have to exclude s_1 proceeding to next level. Starting from the root's left child indicates inclusion of s_2 and right child indicates exclusion of s_2. Each node stores the sum of the partial solution elements. Ever if at any stage, the no. equals m, then the search is successful and teaminates.

The deadend in the tree occurs only when either of the following two condition.

1. Sum of s is too large.
2. Sum of s is too small.

We consider a backtracking using fixed tuple size strategy. In this case the element x_i of solution vector is either 1 or 0 depending on whether the weight w_i is included or not. A Bounding function is $B_k(x_1, x_2, \ldots x_n)$ = true iff

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \leq m.$$

Clearly $x_1, x_2, \ldots x_k$ cannot lead to an answer node if this condition is not satisfied.

$$x_i \in \{j / j \quad i \leq j \leq n\} = m$$

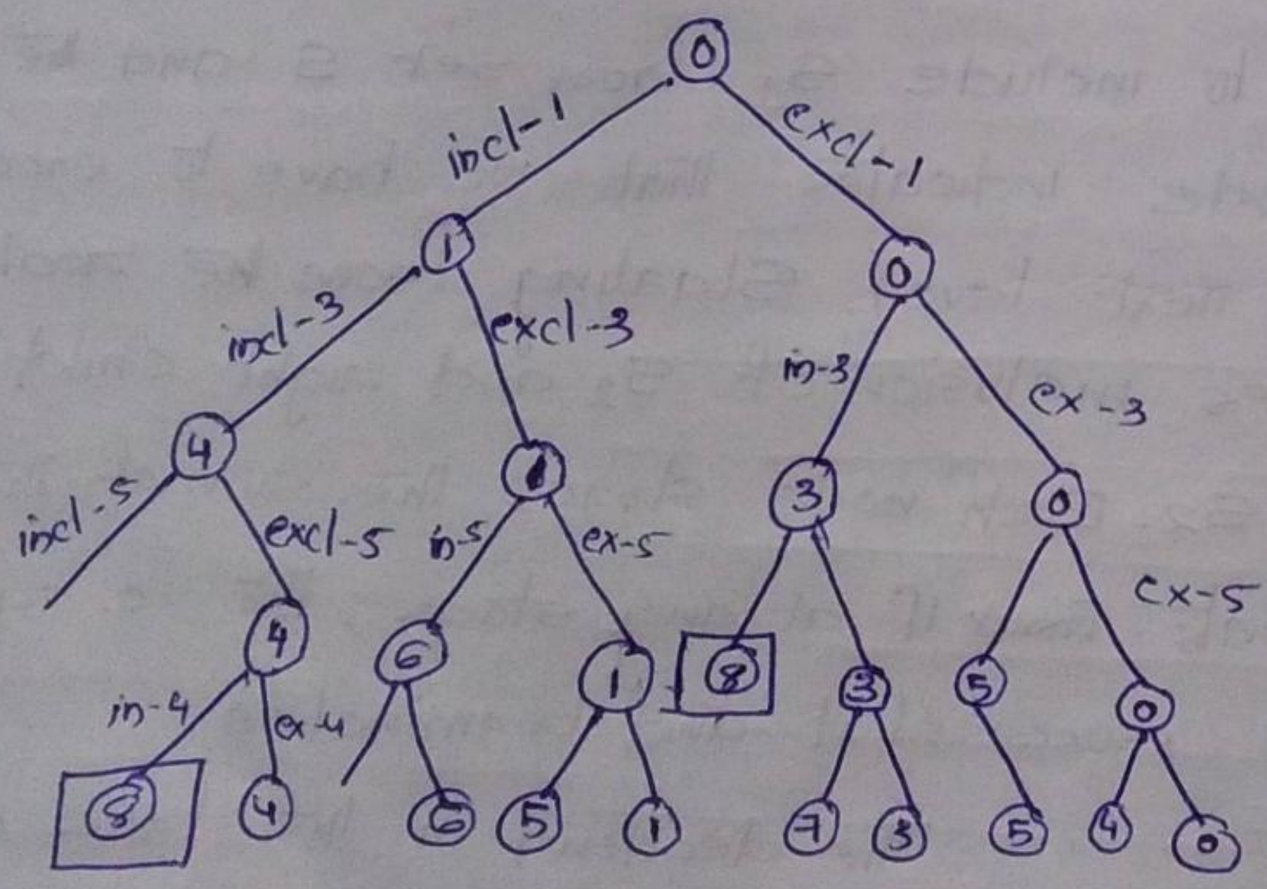
Eg:

$w/S = \{1, 3, 5, 4\}, \quad m = 8.$

$S_1/w_1 = \{3, 5\}$

$S_2 = \{1, 3, 4\}$ — $(1, 2, 4)$

$(1, 1, 0, 1)$

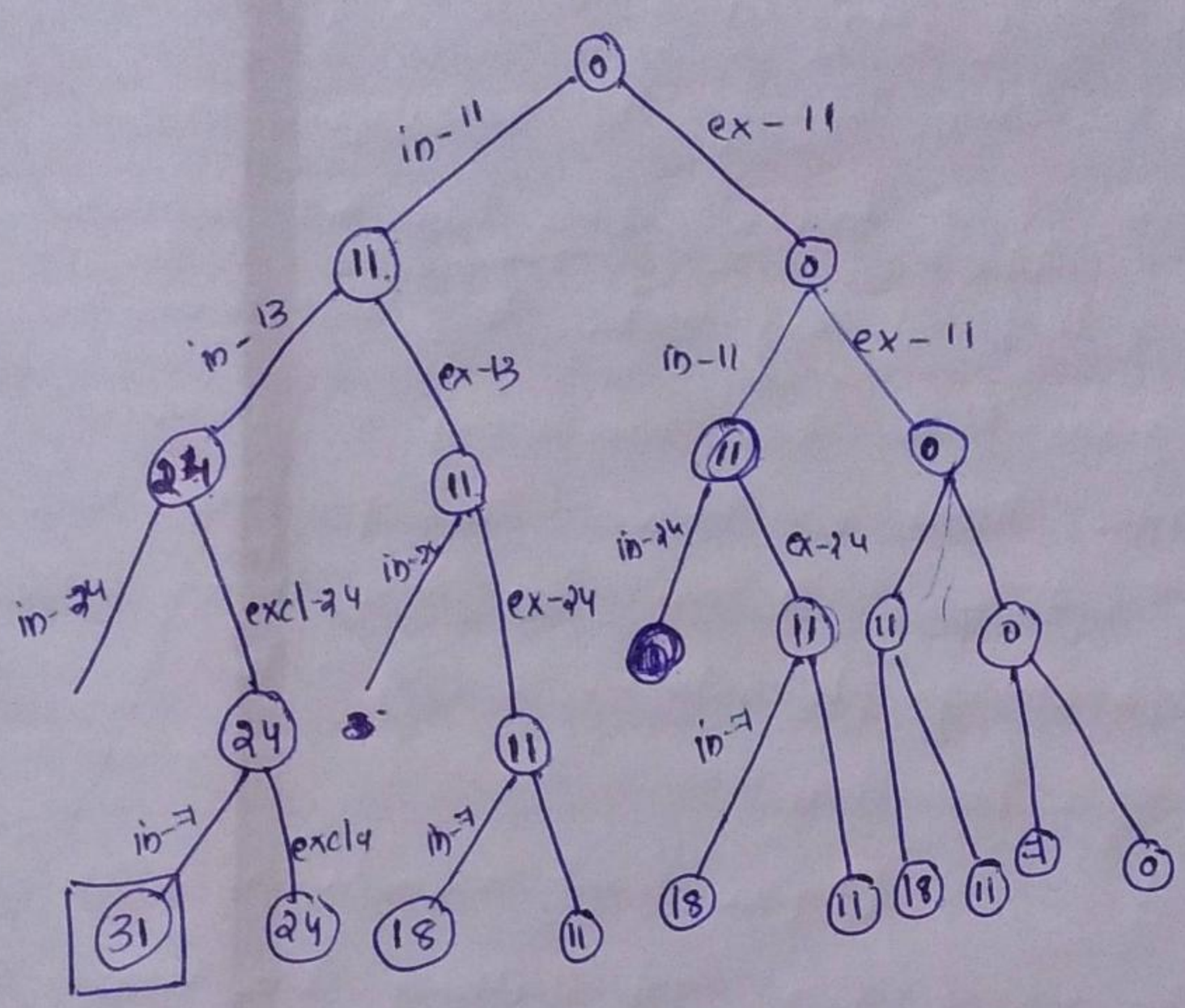


Eg: 2) $S = \{11, 13, 24, 7\}$ $m = 31$

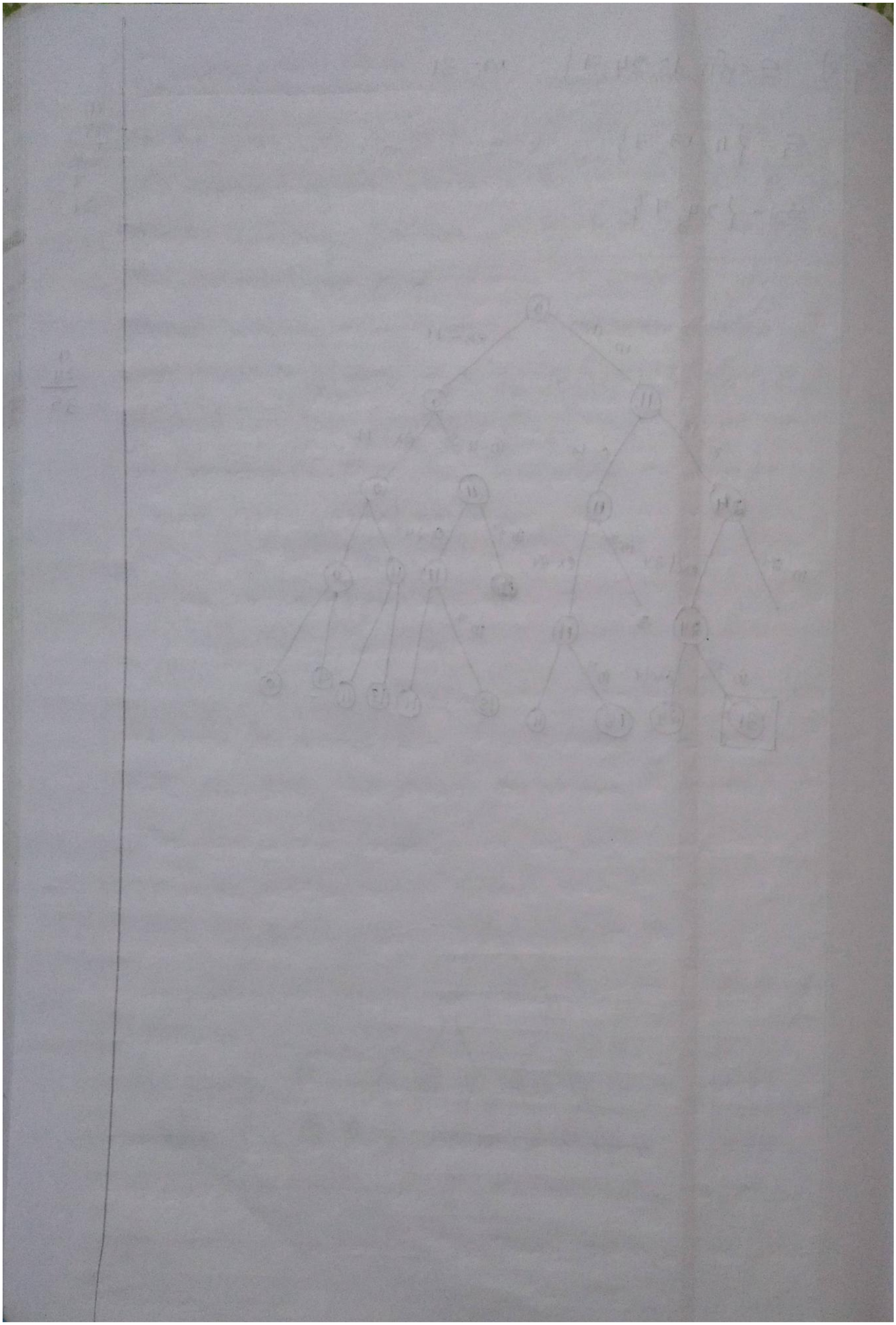
$S_1 = \{11, 13, 7\}$

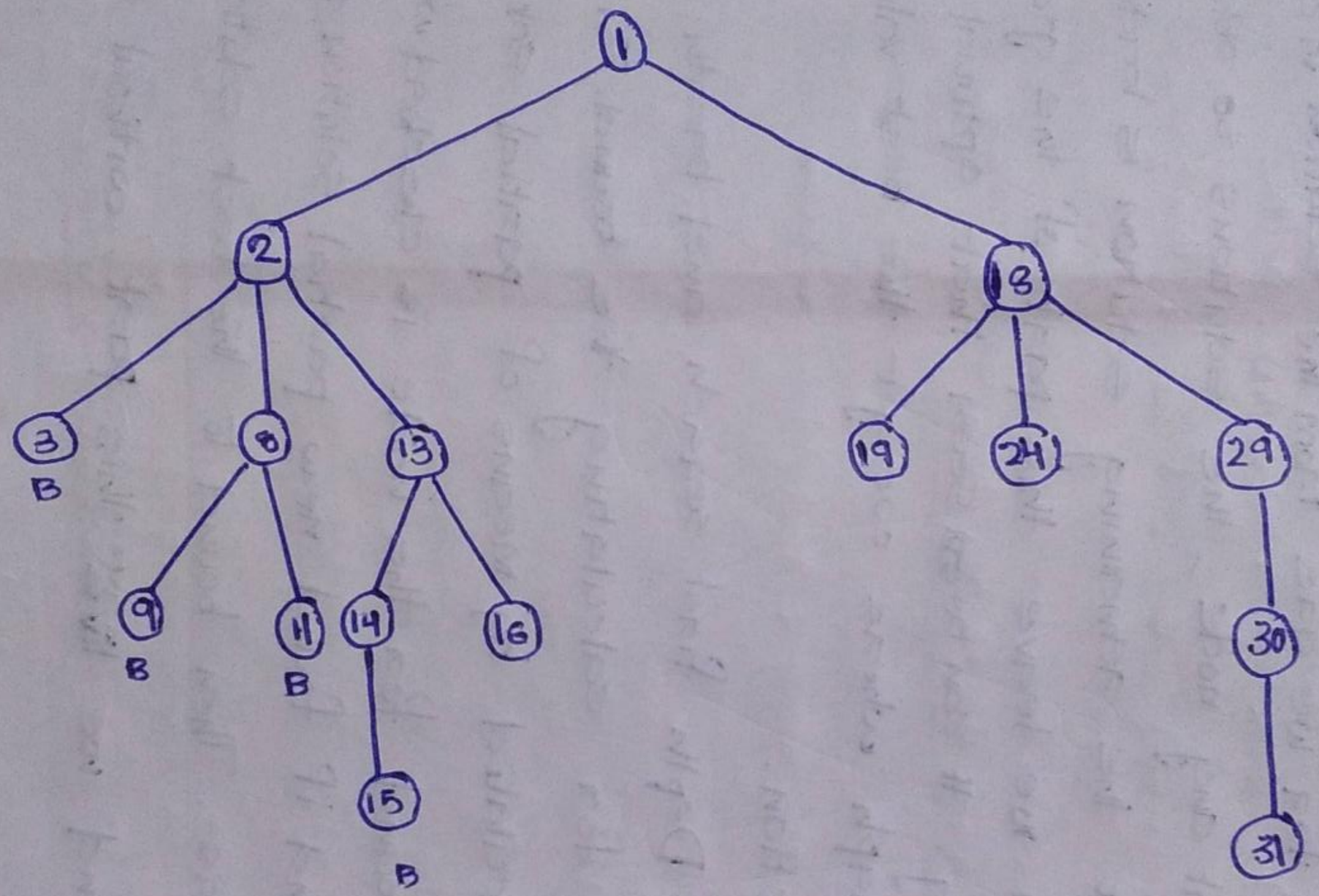
$S_2 = \{24, 7\}$

11
13
24
7
31



11
24
35





17/12/21
Friday

Branch and Bound

The branch and bound technique like backtracking explores the implicip graph and deals with the optimal solution to a given problem. In this technique, at each stage, we calculate the bound for a particular node and check whether the bound will be able to give the solution or not. That means we calculate how far we are from the solution in a graph. If we find that at any node, the solutions so obtained is appropriate. But the remaining solution is leading to a worst case, then we leave this part of the graph without exploring. It can be seen that optimal solution in an implicit graph where we get the best value for the objective function.

Depth first seqach and breath first seqach is used for calculating the bound. For each node, bound is calculated by means of partial solution. The calculated bound for the node is checked with previous node and if found new partial solution results leads to worst case. Then bound to the best solution so far selected and we leave this part without exploring further.

Branch and bound is a general algorithmic method for finding optimal solution of various optimization problem. It is basically an enumeration approach in a ~~fa~~ fashion that proves the non-promising space tree.

The first one is a smart way of covering feasible region by several smaller feasible subregion. Since the procedure may be repeated recursively to each of the subregions and all produced subregions naturally form a tree structure.

The term Branch and Bound refers to all state space search methods in which all children of the E node are generated before any other live node can become the E node. In Branch and Bound terminology a BFS, like state space search will be called before search as the list of live nodes is a first-in-first-out list. A D-search like state space search will be called LIFO search as the list of live node is a last in-first-out list. As in the case of backtracking bounding functions are used to help to avoid the generation of subtree that do not contain an answer node.

Least Cost Search

Algorithm

```
1. Algorithm LCSearch (t)
2. // Search t for an answer node.
3. {
4.    if *t is an answer node then output *t and
      return;
5.    E := t; // E-node
6. Initialize the list of live nodes to be empty;
```

```
7.    repeat.
8.    {
9.       for each child x of E do
10.      {
11.         if x is an answer node then output the path
12.                from x to t and return;
13.         Add(x); //x is a new live node.
14.         (x → parent) := E; // Pointer for path to root.
15.      }
16.   if there are no more live nodes then
17.   {
18.      write ("No answer node"); return;
19.   }
20.      E := Least();
21.   } until (false);
22. }
```

The search for an answer node can often be speeded by using an intelligent ranking function $\hat{c}(\cdot)$.
The next E-node is selected on the basis of this ranking function. The ideal way is to assign ranks would be on the basis of the additional computational effort/cost needed to reach an answer node from the live node. For any node x, the cost would be:

1. The no. of nodes in subtree x, that need to be

generated before an answer node is generated.

2. The no. of levels between the nearest answer node is from x.

Let $\hat{g}(x)$ be an estimate of the additional effort needed to reach an answer node from x. Node x is assigned a rank using a function $\hat{c}$ such that $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ where $h(x)$ is the cost of reaching x from the root and $f()$ is any non-decreasing function.

A search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next E-node would always choose of its next E-node, a live node with least $\hat{c}$. Hence such a strategy is called an LC Search.

1	2	3
4	5	6
7	8	X

$N^2 - 1$ Puzzle

15 puzzle consist of 15 number tiles on a square frame with a capacity of 16 tiles. Our objective is to transform the initial arrangement into the goal arrangement through a series of legal vaster moves. The 15 puzzle has different sized variables. The smallest size involves about 2×3 and is called the 5-puzzle. The 8-puzzle involves about 3×3 and 35 puzzle involves about 6/6 ~~2/3 and its called as 5 puzzle. The 8 puzzle involves of board~~

The family of these puzzles ~~are~~ is called as the n-puzzle where the n stands for the no. of tiles. In all of the n-puzzle, we use the tiles in the goal stage where ordered from left to right and top - to down with an empty space located in the bottom-right corner. It is known as the family of n-puzzles belong to the class of NP Complete problems. Which means that the no. of path grows exponentially with no. of tiles and finding the shortest path from the start to the goal where require performing an exhoslive search. ~~The goal may require~~

Thus from the initial arrangement 4 moves are possible. The only legal moves are one in which a tile adjusant to the empty tile spot is move to ES. Each move creates a new arrangement and is called as the stage to the puzzle. The initial and

goal arrangments are called the initial and goal state

The state space of an initial stage consist of all stages that can be reached from the initial state. The most straight forward way to solve the puzzle ~~could be to store the~~ to search state space for the goal space ——— and use the path from the initial state to the goal state as the answer