Backtracking

N Queen, Subset sum, Hamiltonian cycle

Backtracking is an approach to construct the solution of one component of problem at a time and evaluate the partially constructed solution as per following.

- If partially constructed solution can be developed further without violating the problem's constraint, take/ select the first legitimate option for the next component.
- If there is no legitimate option for the next component, then no alternative of remaining component to be considered and the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

State space tree , promising node , non promising node.

A tree of choices to implement backtracking is called **state space tree**. The root of the represents initial state and other nodes represent the alternative choices.

A node is called **promising node,** if the partially constructed solution due to this node lead to a complete solution. Otherwise the node is called **non-promising node**.

If the current node is promising, its child is generated by adding the first remaining legitimate option for next component of solution and processing moves to this child.

If the current node is non-promising, the algorithm backtracks to the parent to consider next options for its last component.

The processing continues until it find the complete solution.

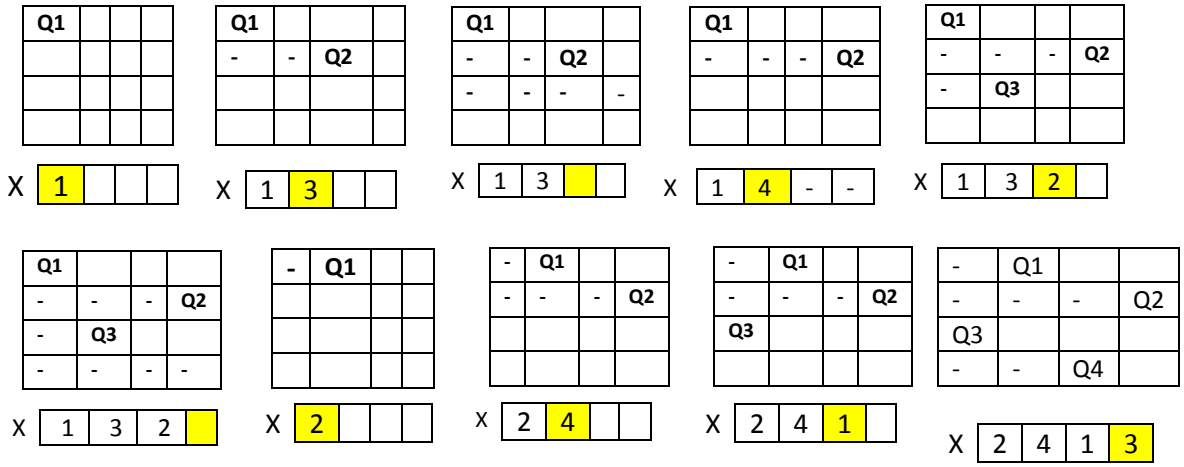
N-Queen Problem

Given n X n board, the N-Queens problem is to place n number of queens on the board, so that no two queens can attack each other. Two queens can attack each other if they are placed in same row or in same column or in same diagonal.

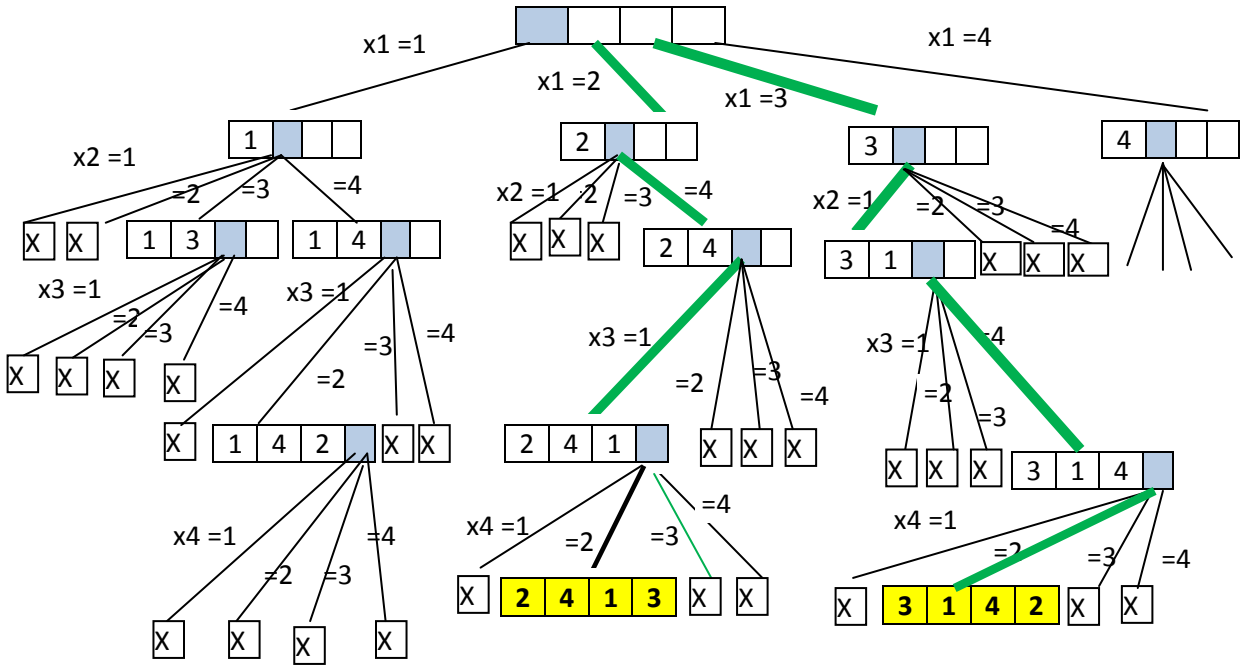
Let X[1. . . n] is the solution vector, where Xi is the column in which Queen i is placed. the constraints are

- Xi takes the values from {1,2, . . , n}
- No two Xi can be the same and no two queens can be on same diagonal.

Example: 4 Queen Problem



State-Space tree for 4 Queen problem



Checking two queens can attack

1. Checking queens in same column : If the elements in solution vector are not distinct, then two queens are in same column

2. Checking queens in same diagonals: Element in upper-left to lower-right diagonal has same (row - column) values. Element in upper-right to lower-left diagonal has same (row +column) values.

If the index of the queens are (i,j) and (k,l) then they are in same diagonal if

$(i - j) = (k - l)$

$\Rightarrow j - l = i - k \quad \ldots\ldots\ldots\ldots (1)$

$(i + j) = (k + l)$

$\Rightarrow j - l = k - i = -(i-k) \quad \ldots \quad \ldots (2)$

So two queens are in same diagonal if $\mathbf{|\, j - l \,| = |\, i - k \,|}$

Algorithm:

```
NQueen(k,n)
```
//This algorithm places n Queens in nXn board. Initial value of k = 1
```
1. for  l = 1 to n.
2.   If PLACE(k,l)
3.        X[k] = l
4.        if k = n
5.             Write X.
6.        else , NQueen(k + 1 , n)
```

```
PLACE(k,ℓ)
```
//Return true if position ℓ is feasible for k^{th} Queen. Return false position ℓ is not feasible
```
1. for i = 1 to k – 1
2.   if x[ i ] = ℓ or ABS( X[i] – ℓ ) =ABS( i – k)
3.        return False
4. return True
```

There are n places in solution vector. For 1^{st} position, there are n possible value(placement) of 1^{st} queen. For each value of 1^{st} position, there are n possible value for 2^{nd} queen and so on. So the upper bound of the running time of the N-Queen Algorithm is $\mathbf{O(n^n)}$ **.**

If we ignore checking the placement of queens in same row of already placed queen, then the complexity of algorithm will be **O(n!)**

Subset Sum Problem

Given a set W of n sorted distinct numbers and a target M, the objective of the subset sum problem is to generate all subsets of set W whose elements sums to M.

Example: Given n=4 , M = 31 , W={7,11,13,24}, solutions to the problem are:-

S1 = {7, 11, 13} and S2 = {7, 24}

The solution can be represents in two ways:--

I. Variable sized tuple

II. Fixed sized tuple

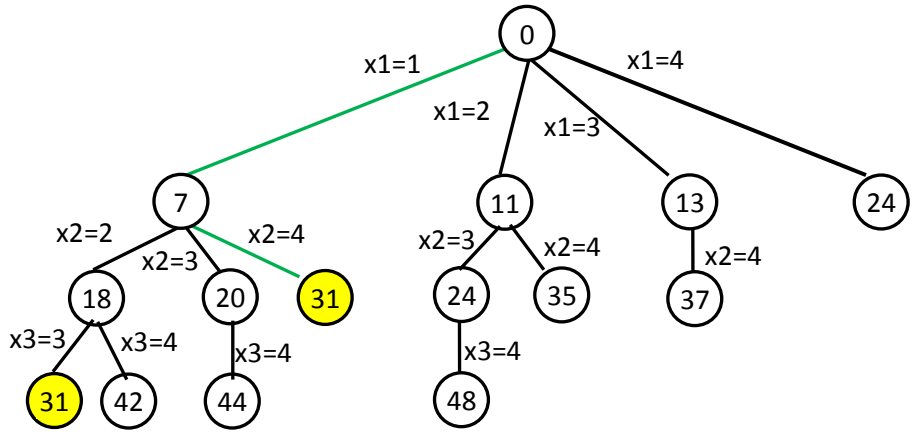
In **variable sized tuple,** the solution vector is represented by indices of W. i,e

S1 = { 1, 2, 3 } and S2 = { 1, 4}

In **fixed sized tuple**, the solution vector is represented a n-tuple {x1, x2, . . xn } such that $x_i = 0$ if w_i is not include and $x_i = 1$ if w_i is included in solution vector.

S1 = { 1,1,1,0 } and S2 = { 1,0,0,1,}

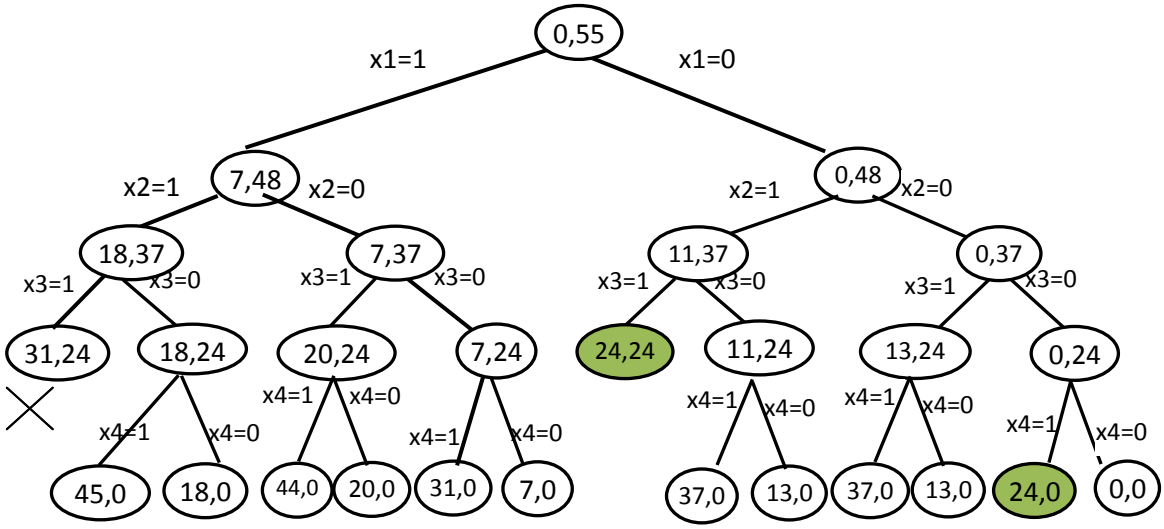
State-space tree for the problem : W={7,11,13,24} and M=31 using fixed sized tuple .



Question-1: Draw state space tree for W = {3,5,6,7},M=15.

Question-2: Draw state space tree for W={5,10,15,20,28} and M=35.

State-space tree for the problem: M =24, W = {7, 11, 13, 24} using fixed sized tuple.



In order to find a non-promising node, where we should stop going further and backtrack, following bounding function is used.

1. $S + W_i > M \quad , \quad if \quad sum \quad is \quad too\ large$

2. $S + \sum_{j=i+1}^{n} a_j < M \quad if \quad sum\ is \quad too \quad small$

M =31, W = {7, 11, 13, 24}

```
SubsetSum(S,k,r,X,M,W)
```

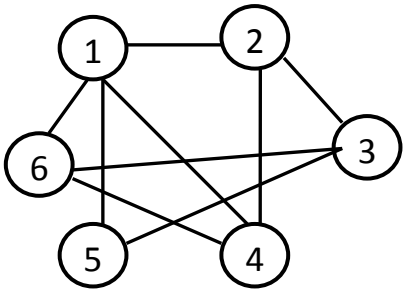
// Initial value of X={0,0,0,0},k=1, S=0,r(=55) is the sum all the numbers in W.

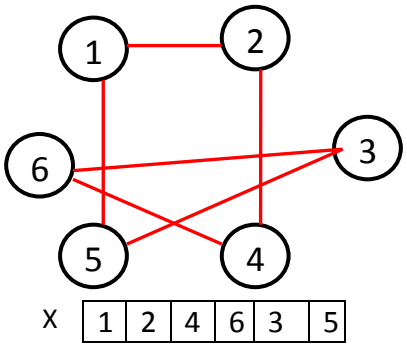
```
1. X[k] = 1
2. if S + W[k] = M
3.      Write X
4. Else, if S + W[k] + W[k+1] ≤ M
5.      SubsetSum(S+w[k], k + 1, r-W[k], X, M, W)
6. If (S + (r - W[k]) ≥ M)  AND (S + W[k+1] ≤ M)
7.      X[k] = 0
8.      SubsetSum(S,k + 1, r - W[k], X, M, W)
```

Hamiltonian Circuit problem / Hamiltonian cycle

---Suggested by Sir William Hamilton

Given an undirected graph G=(V,E) , Hamiltonian circuit is a simple cycle that contains each vertex in V visited only once (except the start vertex) and return to start vertex.

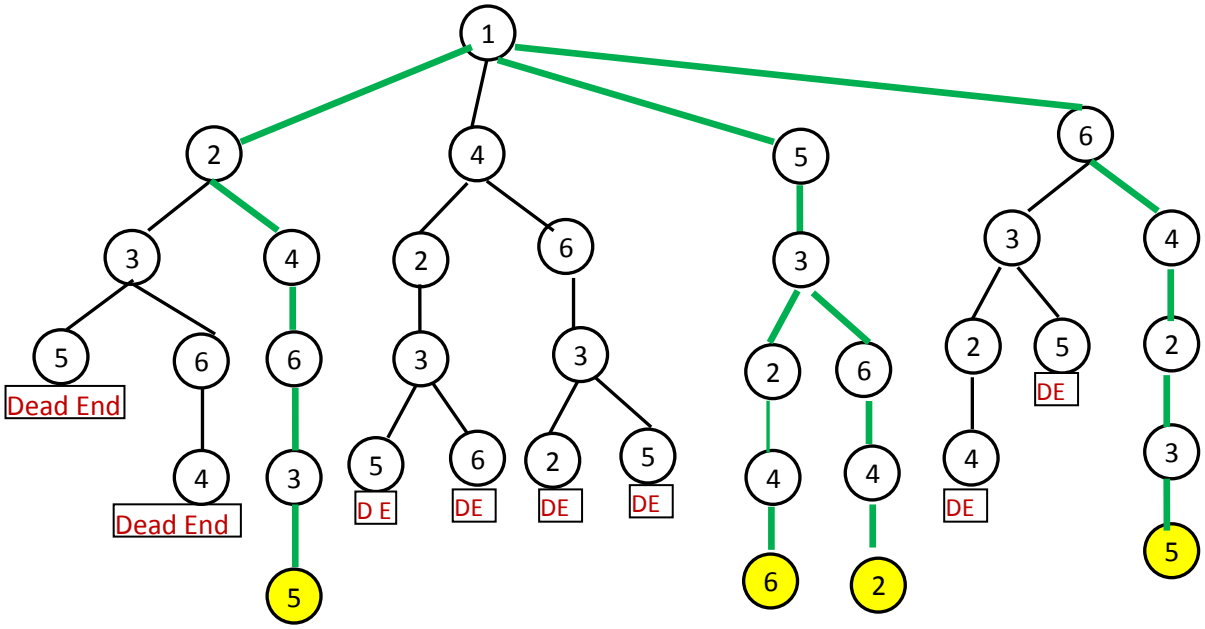




Backtracking solution:

Suppose X is the solution vector X=<x_1,. . . x_n>and suppose x_1,x_2,. . . x_{k-1} have already chosen. If k = 1, then x_1 be any vertex from V. For any other k <n, x_k can be any vertex ***v*** that is distinct from x_1,x_2,. . . x_{k-1} and ***v*** is connected to xk-1 . x_n will be one remaining vertex and is connected to both x_{n-1} and x_1.

State space Tree for constructing Hamiltonian cycle



Hamiltonian (A,n,X,k)

//The graph with n vertices is represented by Adjacency matrix A.X is solution vector, whose //X[k] value will be fixed. Initial value of k=2, X [1] =1 and X [2. . . n] = 0.

```
1. While True
2.      NextValue(A,n,X,k)
3.      if X[k] = 0
4.            return      //if no vertex is suitable, backtrack to previous solution (X_{k-1})
5.      if k = n          // If all values of X are fixed?
6.            write X[1. . .n]
7.      else, Hamiltonian (A,n,X,k+1)      // Call to fix next value of X
```

NextValue(k) is used to determine k^{th} vertex of Hamiltonian cycle, assuming k-1 vertices are already determined. A vertex v is set in k^{th} index if it is not already added and is connected to the vertex in index k-1.

```
NextValue(A,n,X,k)
```

//This algorithm set a vertex in k^{th} index. If it finds no suitable vertex, it assigns X[k] = 0

```
1. While True
2.    X[k] = (X[k] + 1) MOD (n+1)
3.    if X[k] = 0                  // No vertex is suitable?
4.         return
5.    if A[ X[k-1],X[k] ] ≠ 0        //vertex X_k is adjacent to X_{k-1}?
6.         for j = 1 to k-1          // Checking  X_k is already added or not
7.              if X[j] = X[k]
8.                   break
9.         if k = j              //  X_k is distinct?
10.             if (k < n) or (k=n and A[X[n], X[1]]) ≠0)
11.                  return
```