Design and Analysis of Algorithms

Backtracking

Rupali Khare

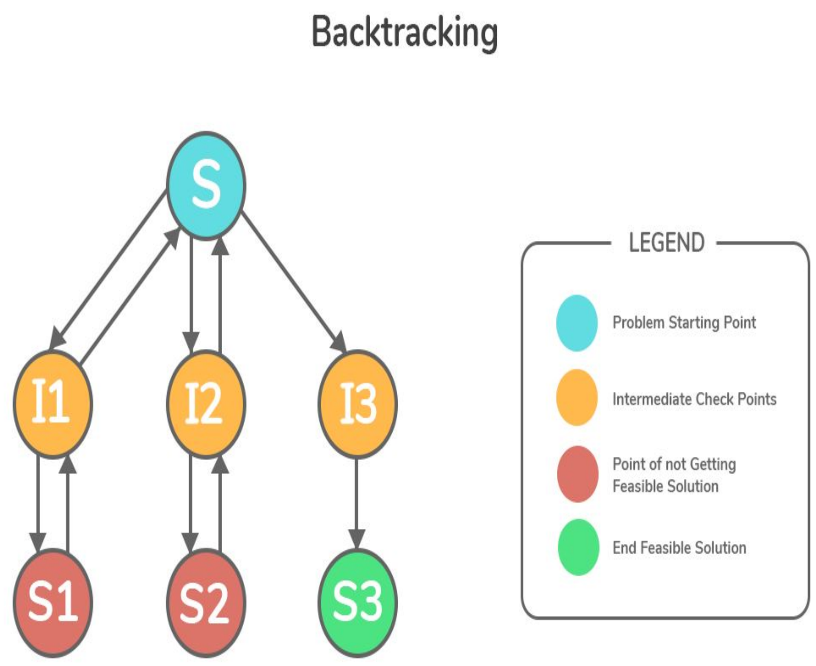
Assistant Professor, DCEA, GLAU

Backtracking

Backtracking algorithms determine problem solutions by systematically searching the solution space for the given problem instance.

This search is facilitated by using a tree organization for the solution space.

Depth first node generation with bounding functions is called backtracking.



State space tree

- All paths from the root to other nodes define the state space of the problem.
- Solution states are those problem states S for which the path from the root to S defines a tuple in the solution space.
- Answer states are those solution states S for which the path from the root to S defines a tuple which is a member of the set of solutions.
- The tree organization of the solution space will be referred to as the state space tree.

1. N-Queen Problem

- The n-queens problem is a generalization of the 8-queens problem of n queens are to be placed on a n x n chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.

```
Nqueens(k,n){
for i=1 to n{
     if(place(k,i)){
        x[k]=i
        if (k==n){
           for j=1 to n
              print x[i]}
        else
          Nqueens(k+1,n)}}}
```

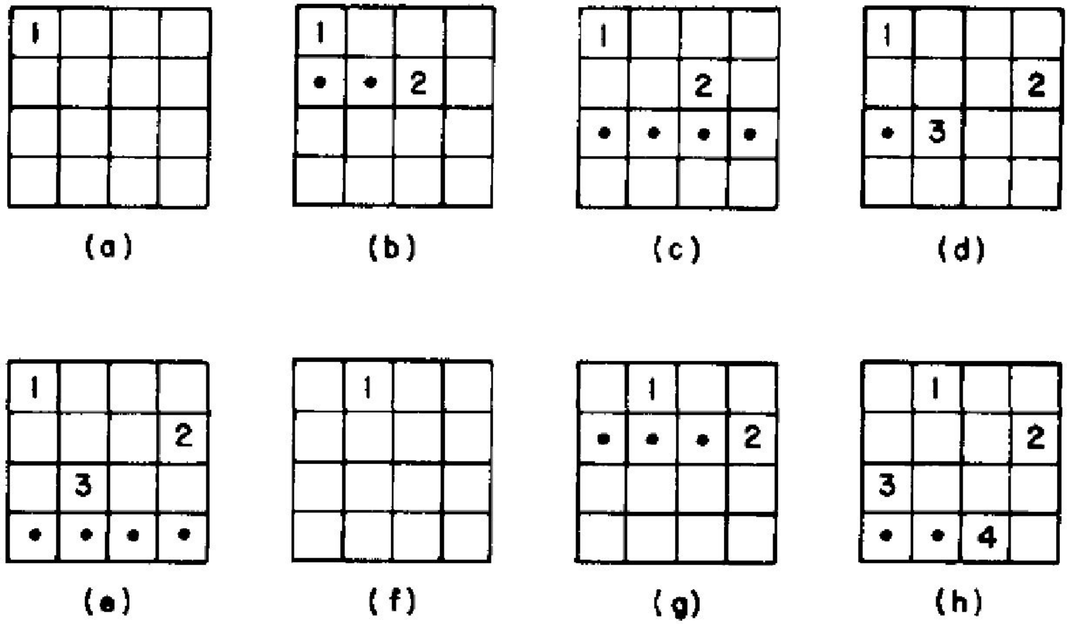
```
place(k,i){
for j=1 to k
     if(x[j]==i || abs(x[j]-i ==abs(j-k))
          return false
     return true
          }
```

N- Queen Problem

Placing n queens on the n x n chessboard, such that none of them attack one another (no two queens are in the same row, column or diagonal)

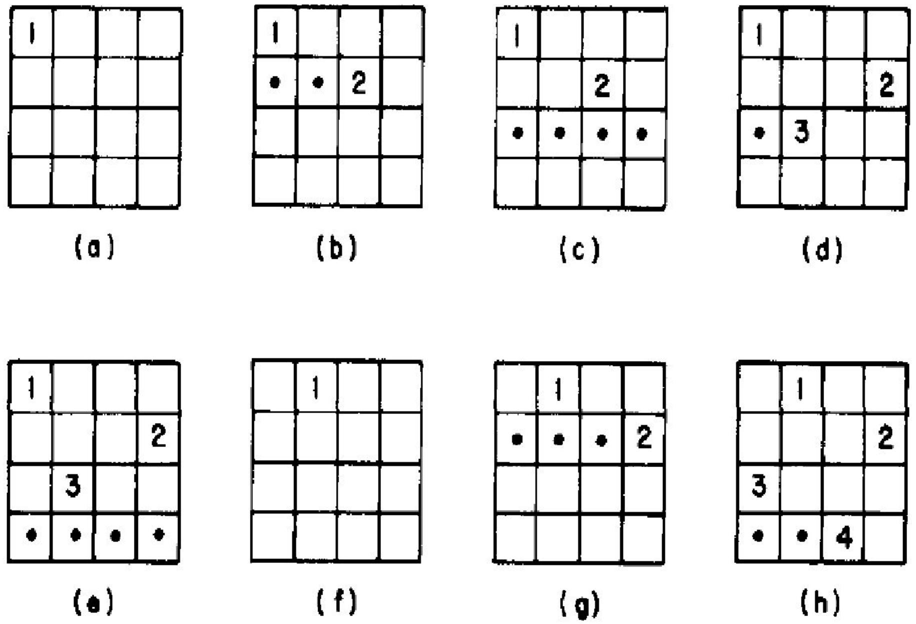
4-Queen Problem

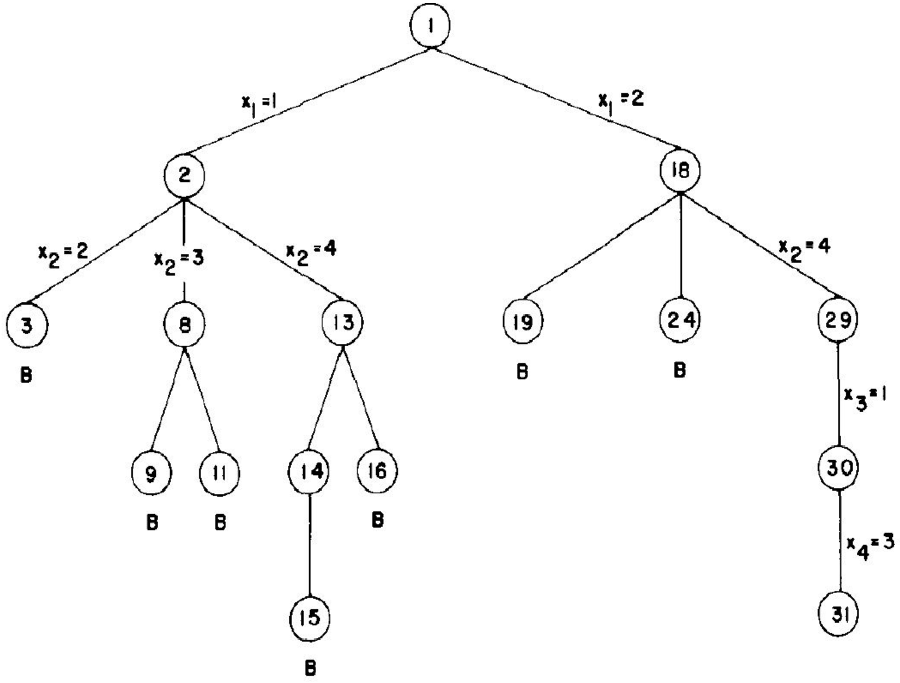
- The 4-queens problem is a generalization of 4 queens are to be placed on a 4 x 4 chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.



4-Queen Problem

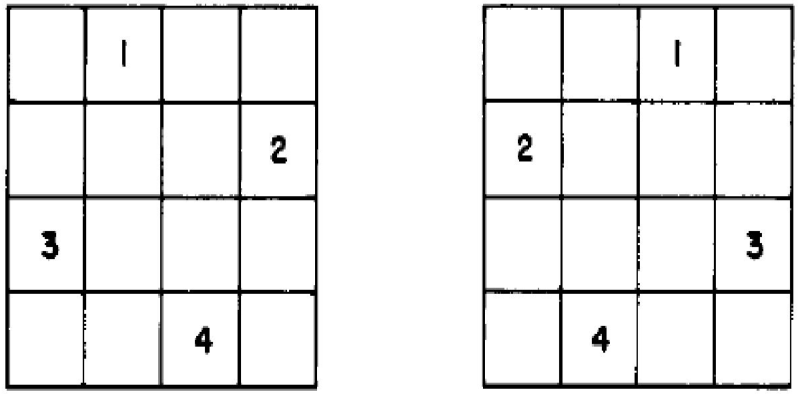
- The 4-queens problem is a generalization of 4 queens are to be placed on a 4 x 4 chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.





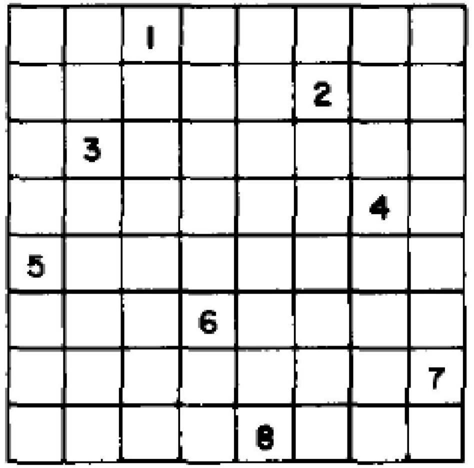
4-Queen Problem

- The 4-queens problem is a generalization of 4 queens are to be placed on a 4 x 4 chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.



8-Queen Problem

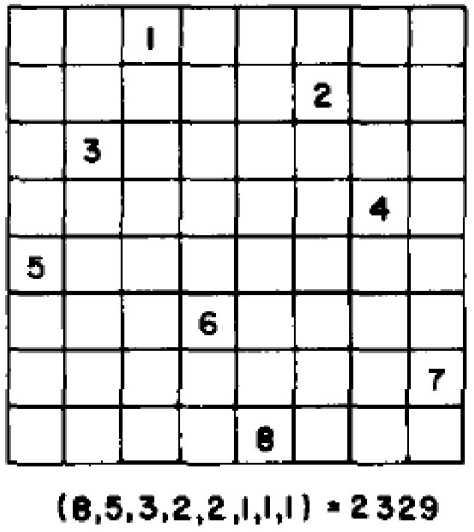
- The 8-queens problem is a generalization of 8 queens are to be placed on a 8 x 8 chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.



(8,5,3,2,2,1,1,1) = 2329

8-Queen Problem

- The 8-queens problem is a generalization of 8 queens are to be placed on a 8 x 8 chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.



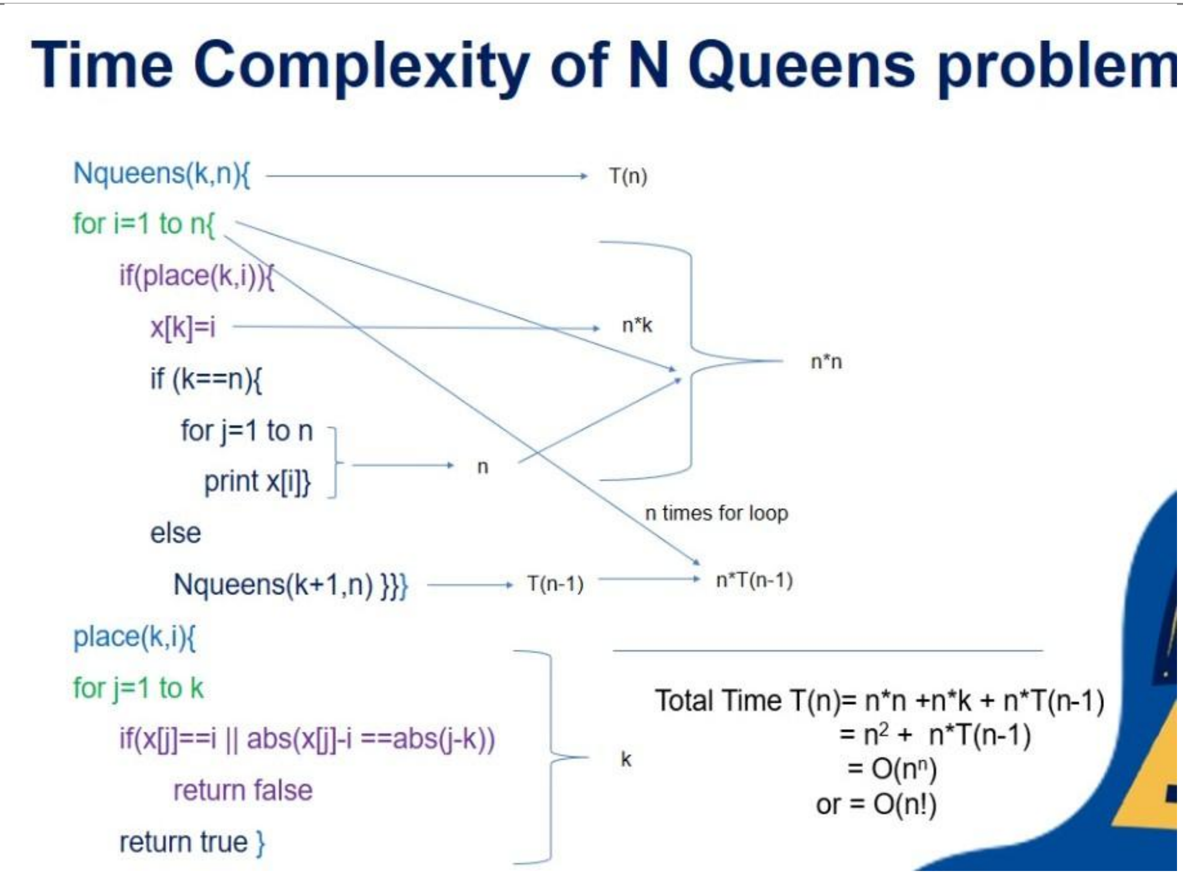
Number of Solution exists

- The number of different solutions, not necessarily distinct, is known for $n \leq 26$.
- It is clear to see that the number of solutions grows exponentially.
- The number of solutions that exist when $n \geq 27$ is still an open problem.

n	Number of different solutions
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200
13	73712
14	365596
15	2279184
16	14772512
17	95815104
18	666090624
19	4968057848
20	39029188884
21	314666222712
22	2691008701644
23	24233937684440
24	227514171973736
25	2207893435808352
26	22317699616364044

n	Number of distinct solutions
1	1
2	0
3	0
4	1
5	2
6	1
7	6
8	12
9	46
10	92
11	341
12	1787
13	9233
14	45752
15	285053
16	1846955
17	11977939
18	83263591
19	621012754
20	4878666808
21	39333324973
22	336376244042
23	3029242658210
24	28439272956934
25	275986683743434
26	2789712466510289

Time complexity



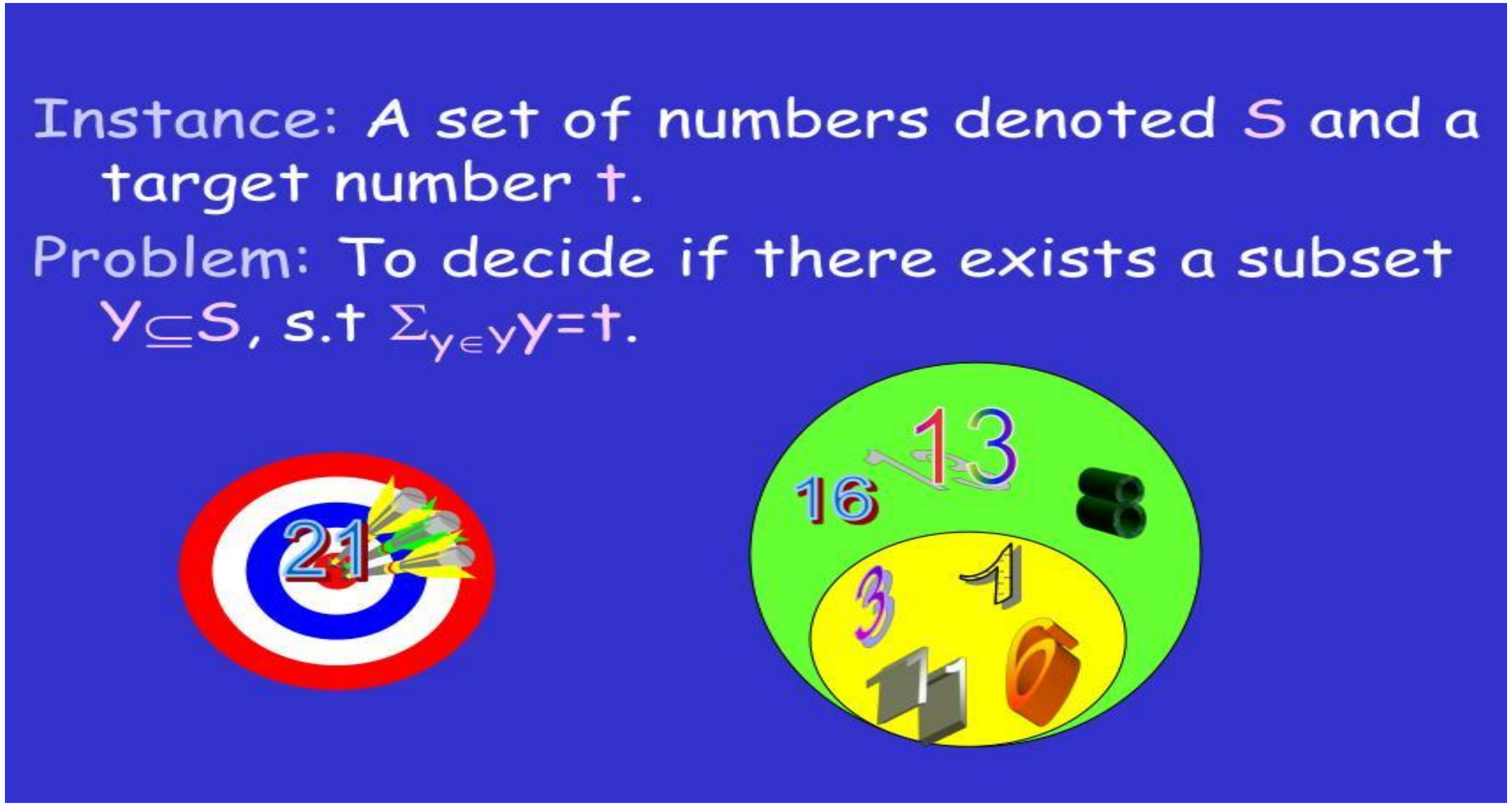
2. Sum of Subset Problem

- Suppose we are given *n* distinct positive numbers (usually called weights) and we desire to find all combinations of these numbers whose sum is *M*. This is called the *sum of subsets* problem.

- In this case the element X(i) of the solution vector is either one or zero depending upon whether the weight W(i) is included or not.

- For a node at level i the left child corresponds to X(i) = 1 and the right to X(i) = 0.

Sum of Subset

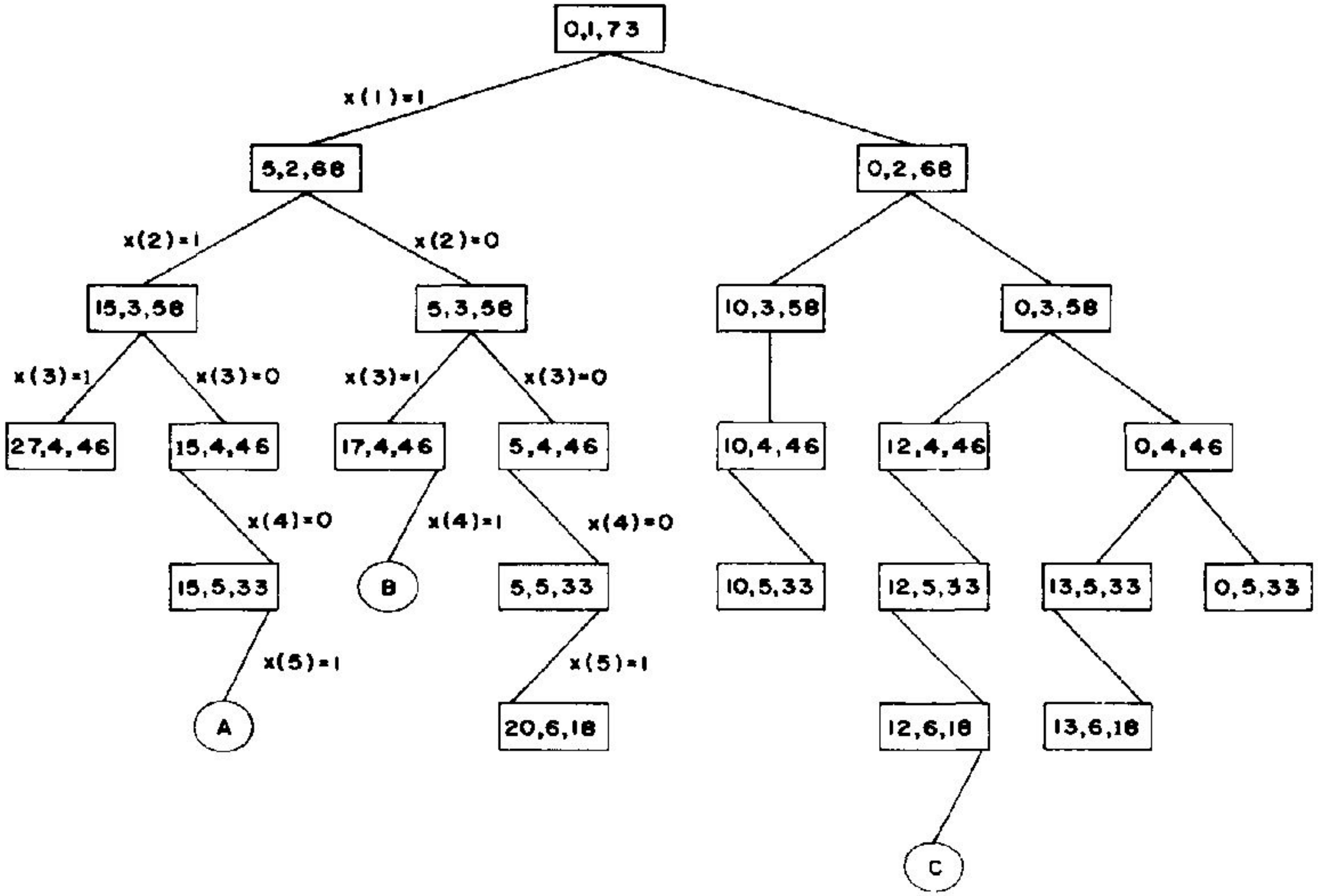


Sum of Subset Problem

The state space tree generated by procedure SUMOFSUB while working on the instance n = 6, M = 30 and W(l :6) = (5, 10, 12, 13, 15, 18).

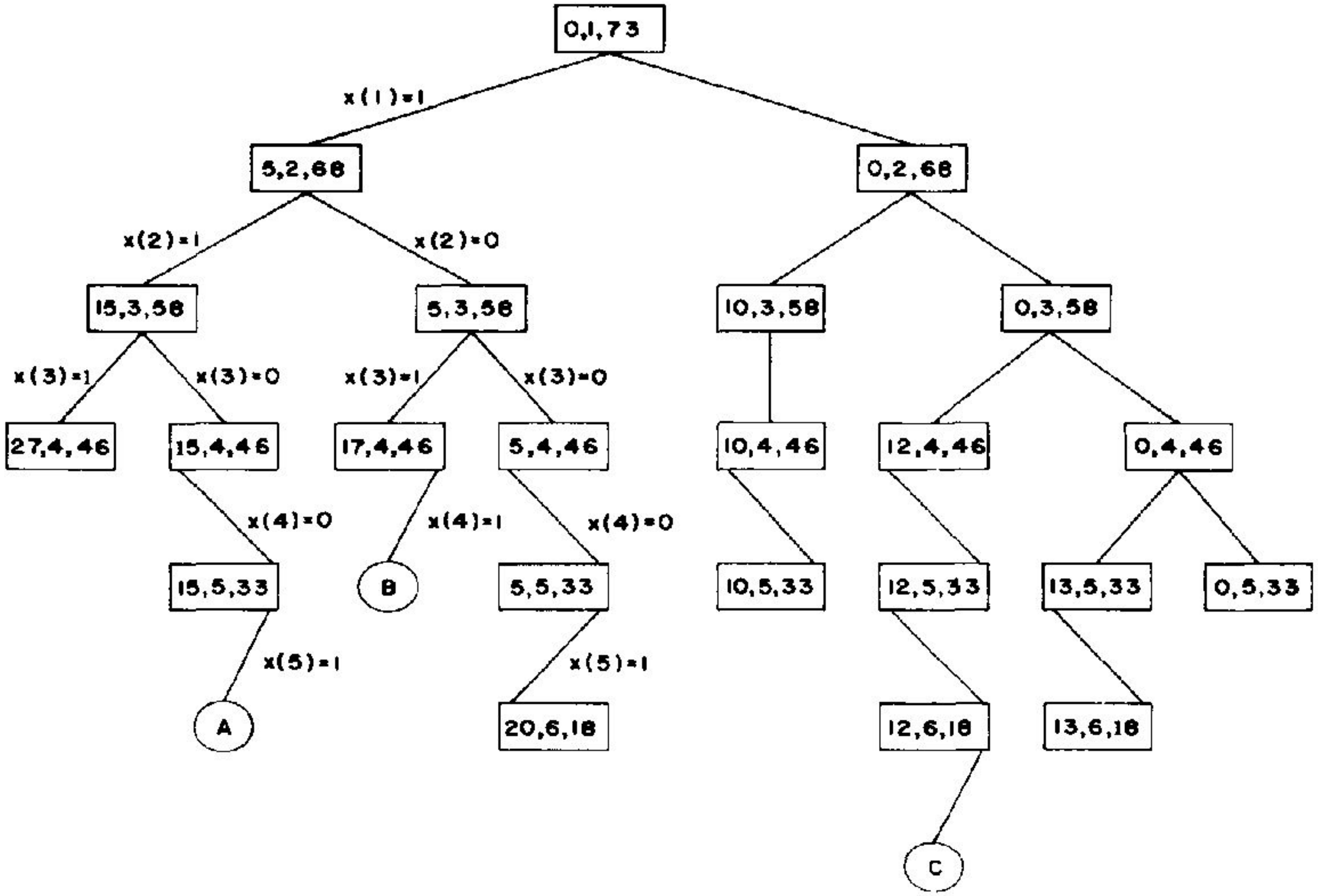
Sum of Subset Problem

The state space tree generated by procedure SUMOFSUB while working on the instance
n = 6, M = 30 and
W(1 :6) = (5, 10, 12, 13, 15, 18).



Sum of Subset Problem

The state space tree generated by procedure SUMOFSUB while working on the instance
n = 6, M = 30 and
W(1 :6) = (5, 10, 12, 13, 15, 18).
At nodes A, B and C the output is respectively (1, 1, 0, 0, 1), (1, 0, 1, 1) and (0, 0, 1, 0, 0, 1).



Sum of Subset Problem

procedure *SUMOFSUB*(s, k, r)

//find all subsets of $W(1{:}n)$ that sum to M. The values of//

//$X(j)$, $1 \le j < k$ have already been determined. $s = \sum_{j=1}^{k-1} W(j)X(j)$//

//and $r = \sum_{j=k}^{n} W(j)$ The $W(j)$s are in nondecreasing order.//

//It is assumed that $W(1) \le M$ and $\sum_{i=1}^{n} W(i) \ge M$.//

```
global integer M, n; global real W(1:n); global boolean X(1:n)
real r, s; integer k, j
//generate left child. Note that s + W(k) ≤ M because B_{k-1} = true//
X(k) ← 1
if s + W(k) = M   //subset found//
     then print (X(j), j ← 1 to k)
     //there is no recursive call here as W(j) > 0, 1 ≤ j ≤ n//
     else
        if s + W(k) + W(k + 1) ≤ M then   //B_k = true//
             call SUMOFSUB(s + W(k), k + 1, r − W(k))
        endif
endif
//generate right child and evaluate B_k//
if s + r − W(k) ≥ M and s + W(k + 1) ≤ M   //B_k = true//
     then X(k) ← 0
          call SUMOFSUB(s, k + 1, r − W(k))
endif
end SUMOFSUB
```

Recursive solution for Sum of Subset problem

```
// A recursive solution for subset sum problem
#include <stdio.h>
// Returns true if there is a subset of set[] with sun equal to given sum
bool SOS(int set[], int n, int sum)
{
// Base Cases
if (sum == 0)
    return true;
if (n == 0 && sum != 0)
    return false;
// If last element is greater than sum, then ignore it
if (set[n-1] > sum)
    return SOS(set, n-1, sum);

/* else, check if sum can be obtained by any of the following
    (a) including the last element
    (b) excluding the last element */
return SOS(set, n-1, sum) || SOS(set, n-1, sum-set[n-1]);
}
```

Cont..

```
// Driver program to test above function
int main()
{
int set[] = {3, 34, 4, 12, 5, 2};
int sum = 9;
int n = sizeof(set)/sizeof(set[0]);
if (SOS(set, n, sum) == true)
        printf("Found a subset with given sum");
Else
printf("No subset with given sum");
return 0;
}
```

Dynamic Programming solution for subset sum problem

```
// A Dynamic Programming solution for subset sum problem
#include <stdio.h>
// Returns true if there is a subset of set[] with sun equal to given sum
bool SOS (int set[], int n, int sum)
{  // The value of subset[i][j] will be true if there is a
        // subset of set[0..j-1] with sum equal to i
        bool subset[n+1][sum+1];
        // If sum is 0, then answer is true
        for (int i = 0; i <= n; i++)
        subset[i][0] = true;
        // If sum is not 0 and set is empty, then answer is false
        for (int i = 1; i <= sum; i++)
        subset[0][i] = false;
```

Cont.

```
// Fill the subset table in botton up manner
        for (int i = 1; i <= n; i++)
                                     {    for (int j = 1; j <= sum; j++)
        {                  if(j<set[i-1])
                  subset[i][j] = subset[i-1][j];
                  if (j >= set[i-1])
                   subset[i][j] = subset[i-1][j] ||  subset[i - 1][j-set[i-1]];
        }
}                  /* // uncomment this code to print table
        for (int i = 0; i <= n; i++)
        {              for (int j = 0; j <= sum; j++)
                  printf ("%4d", subset[i][j]);
                                     printf("\n");
        }*/
```

```
// Driver program to test above function
int main()
{
int set[] = {3, 34, 4, 12, 5, 2};
int sum = 9;
int n = sizeof(set)/sizeof(set[0]);
if (isSubsetSum(set, n, sum) == true)
        printf("Found a subset with given sum");
else
printf("No subset with given sum");
return 0;
}
```