Design and Analysis of Algorithms

Backtracking

RAJESH KUMAR TRIPATHI

ASSISTANT PROFESSOR, DEPT. CEA

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.

Graph Coloring

Let G be a graph and m is an integer number. Nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used.

Graph Coloring

```
procedure MCOLORING(k)
//The graph is represented by its boolean adjacency matrix GRAPH(l:n, 1 :n ).
/ /k is the index of the next vertex to color
global integer m, n, X(l :n) boolean GRAPH(l :n, 1 :n)
integer k
loop //generate all legal assignments for X(k)
        call NEXTVALUE(k) //assign to X(k) a legal color//
        if X(k) = 0 then exit endif //no new color possible//
        if k = n
                then print(X) // at most m colors are assigned to n vertices//
                else call MCOLORING(k + 1)
        endif
repeat
end MCOLORING
```

Graph Coloring

```
procedure NEXTVALUE(k)
global integer m, n, X(l:n) boolean GRAPH(l:n, l:n)
integerj, k
loop
        X(k) = (X(k) + 1) mod (m + 1)
        if X(k) = 0 then return endif // all colors have been exhausted
        for j= 1 to n do // check if this color is distinct from adjacent colors
          if GRAPH(k,j) and  //if (k,j) is an edge//
              X(k) = X(j) //and if adjacent vertices have identical colors//
                then exit endif
        repeat
        if j = n + 1 then return endif    //new color found//
repeat //otherwise try to find another color/ I
end NEXTVALUE
```

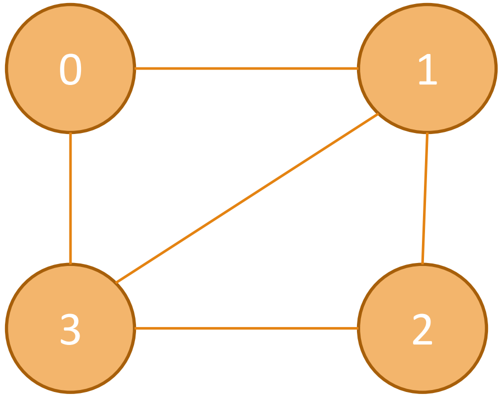
Graph Coloring

```
graphcolor(int k)
{
for(c=1;c<=m;c++)
     if(issafe(k,c))
       { x[k]=c
          if(k+1<n)
          graphcolor(k+1)
          else
          print x[];
return;
}
```

```
issafe(int k, int c)
{
for(i=0;i<n;i++)
 {   if(G[k][i]==1&&c==x[i]
       return false;
 }
return true;
}
```

Graph Coloring

```
graphcolor(int k)
{
for(c=1;c<=m;c++)
    if(issafe(k,c))
      { x[k]=c
        if(k+1<n)
        graphcolor(k+1)
        else
        print x[];
return;
}
```



n=4
m=3

Adjacency Graph G

n	0	1	2	3
0	1	1	0	1
1	1	1	1	1
2	0	1	1	1
3	1	1	1	1

X[k]=

0	0	0	0

```
issafe(int k, int c)
{
for(i=0;i<n;i++)
 {   if(G[k][i]==1&&c==x[i])
        return false;
 }
return true;
}
```

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.

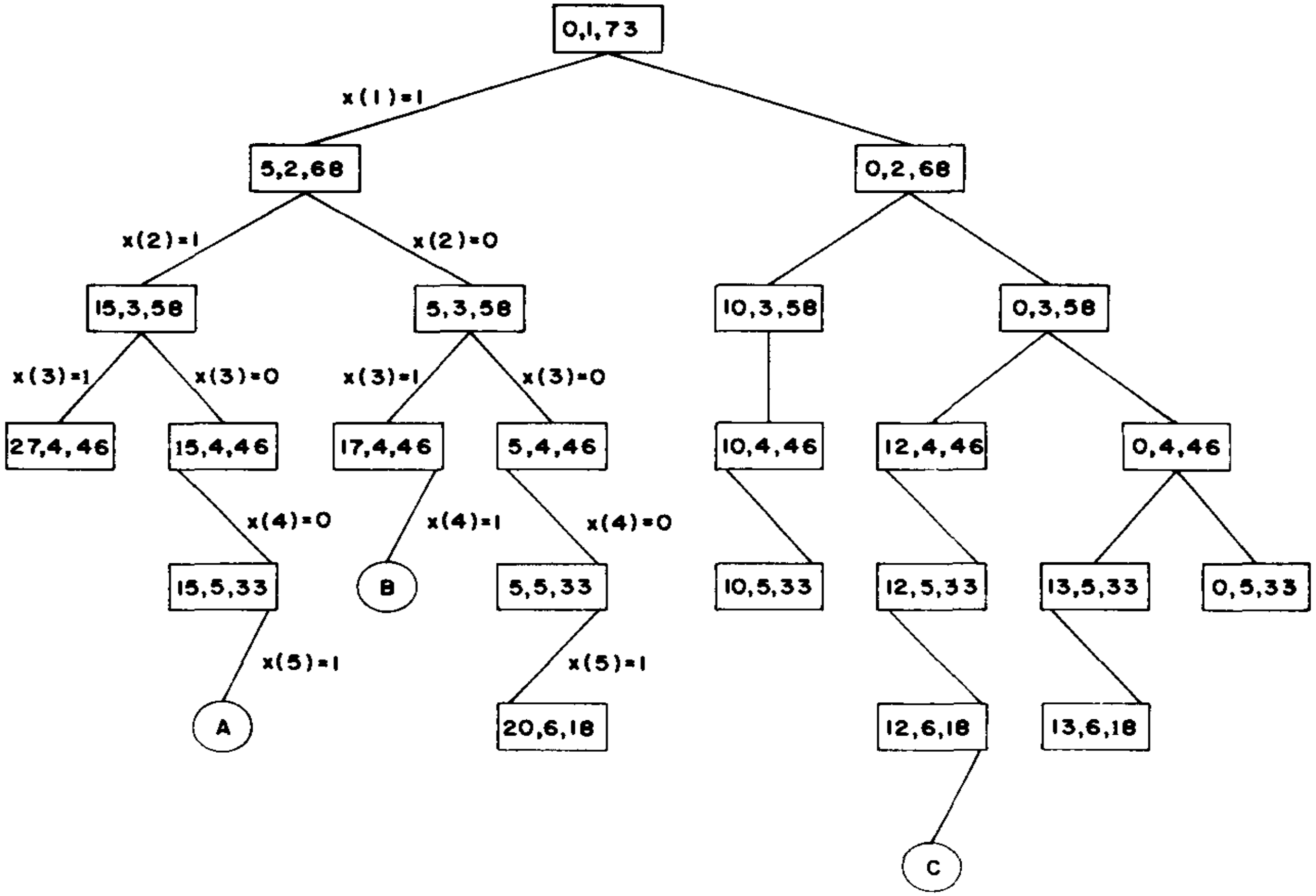
For a node at level i the left child corresponds to X(i) = 1 and the right to X(i) = 0.

Sum of Subset 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).

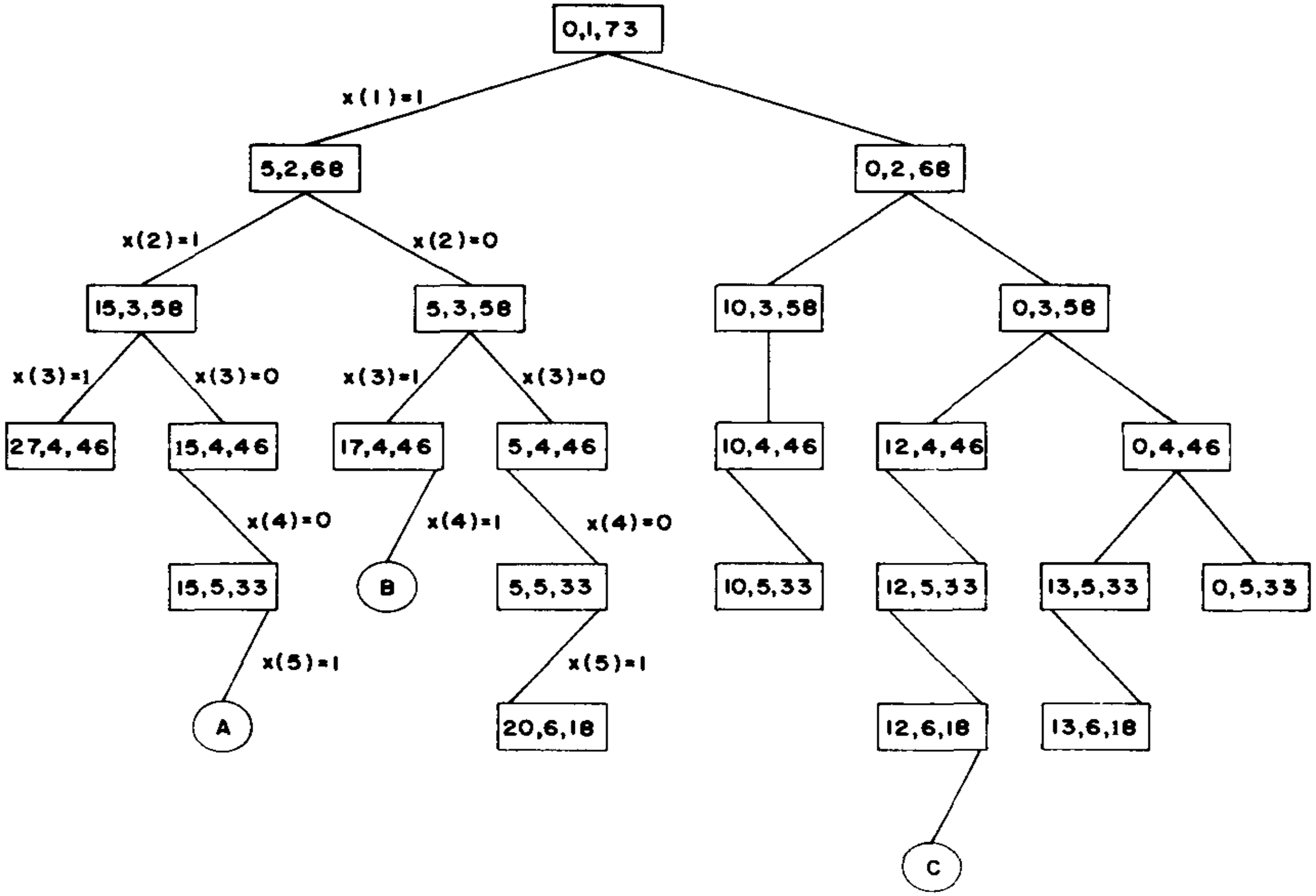
Sum of Subset Problem

The state space tree generated by procedure SUMOFSUB while working on the instance
n = 6, M = 30 and
W(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(l :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 ≤ j < k have already been determined. s = Σ_{j=1}^{k-1} W(j)X(j)//
  //and r = Σ_{j=k}^{n} W(j) The W(j)s are in nondecreasing order.//
  //It is assumed that W(1) ≤ M and Σ_{i=1}^{n} W(i) ≥ 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
```

Thank You

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.

N-Queen Problem

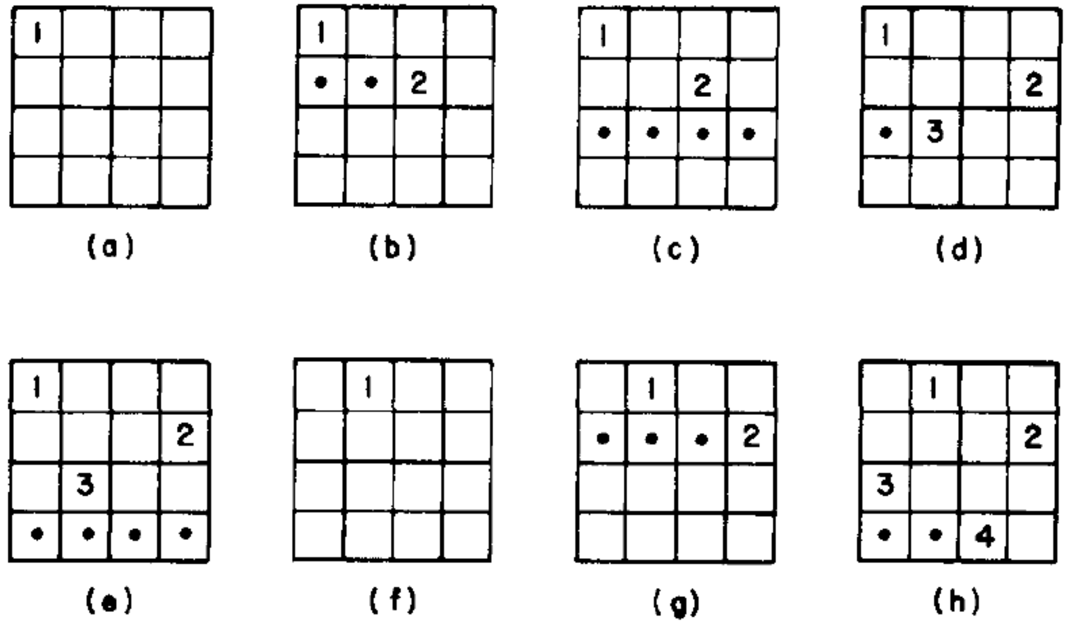
- The n-queens problem is a generalization of the 8-queens problem of n queens are to be placed on a n x n chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.

```
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
         }
```

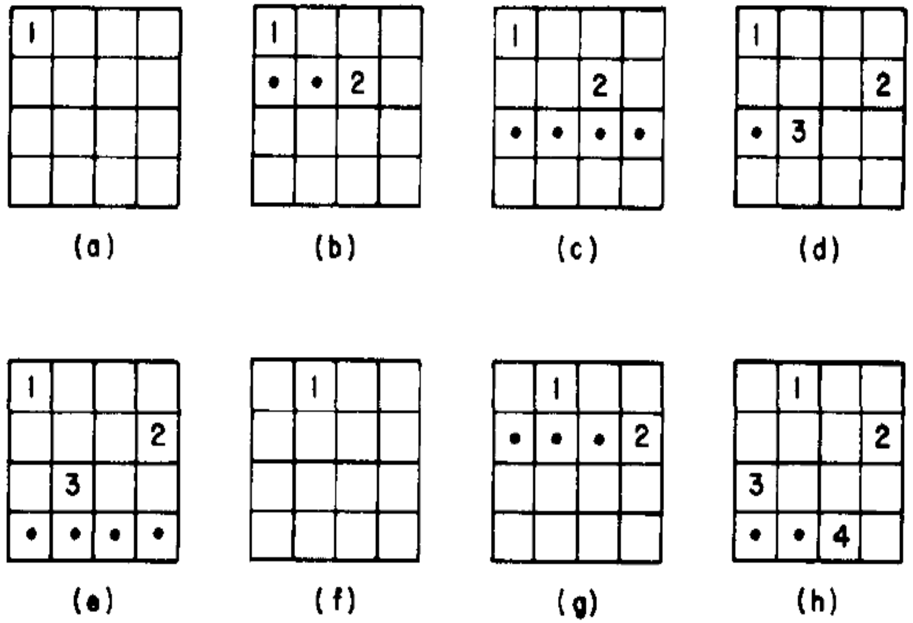
4-Queen Problem

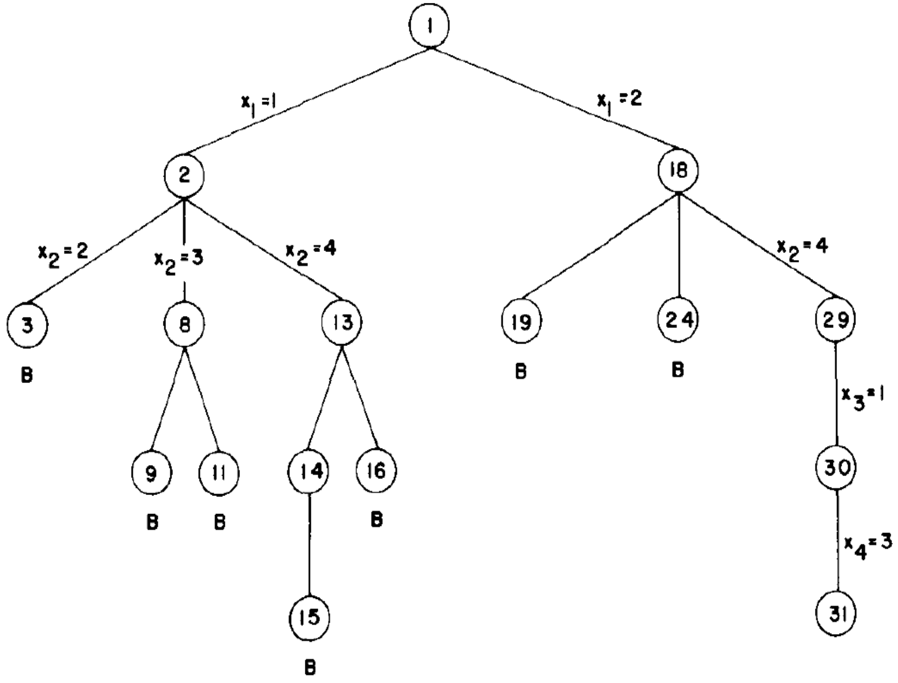
- The 4-queens problem is a generalization of 4 queens are to be placed on a 4 x 4 chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.



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

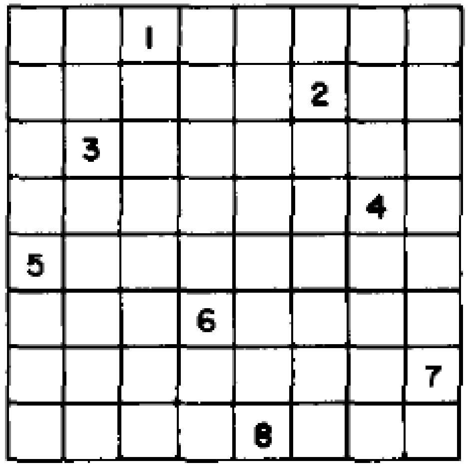
- The 4-queens problem is a generalization of 4 queens are to be placed on a 4 x 4 chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.

	1		
			2
3			
		4	

		1	
2			
			3
	4		

8-Queen Problem

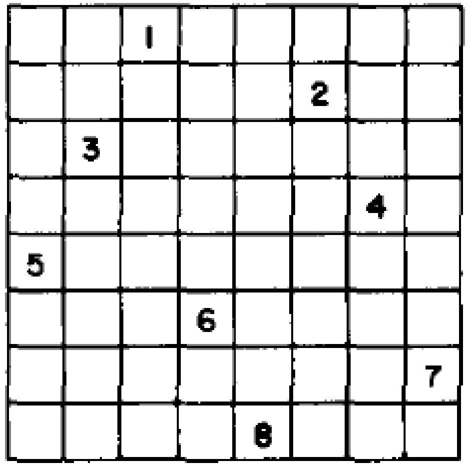
- The 8-queens problem is a generalization of 8 queens are to be placed on a 8 x 8 chessboard so that no two attack, i.e., no two queens are on the same row, column or diagonal.



(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.



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