SNS College of Engineering
Coimbatore - 641107

Backtracking

INTRODUCTION

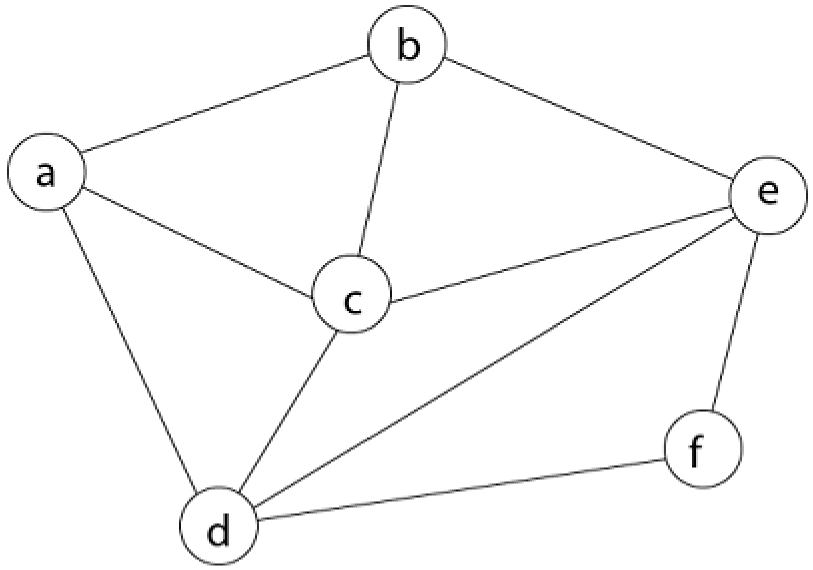
- One of the most general technique.
- Search for set of optimal solutions
- Satisfies the constraints.
- Variation of exhaustive search
- Search is refined by eliminating certain possibilities.
- Faster than exhaustive search.
- Applications:

✓ N- queens problem
✓ Hamiltonian problem
✓ Sum of subset problems
✓ Knapsack problems
✓ Graph coloring

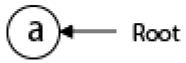
Hamiltonian Circuit Problem

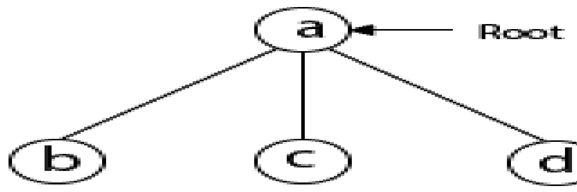
- Given a graph G = (V, E) we have to find the Hamiltonian Circuit using Backtracking approach. We start our search from any arbitrary vertex say 'a.' This vertex 'a' becomes the root of our implicit tree.
- The first element of our partial solution is the first intermediate vertex of the Hamiltonian Cycle that is to be constructed. The next adjacent vertex is selected by alphabetical order. If at any stage any arbitrary vertex makes a cycle with any vertex other than vertex 'a' then we say that **dead end** is reached.
- In this case, we backtrack one step, and again the search begins by selecting another vertex and backtrack the element from the partial; solution must be removed.
- The search using backtracking is successful if a Hamiltonian Cycle is obtained.

Example: Consider a graph G = (V, E) shown in fig. we have to find a Hamiltonian circuit using Backtracking method.

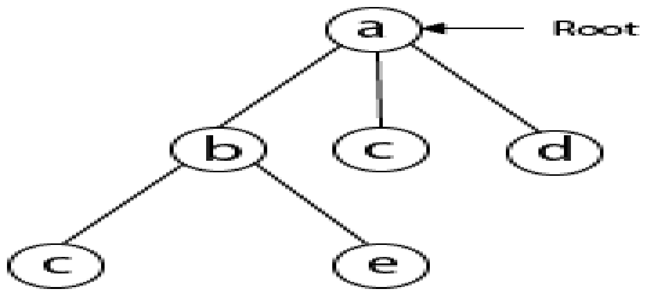


Solution: Firstly, we start our search with vertex 'a.' this vertex 'a' becomes the root of our implicit tree.

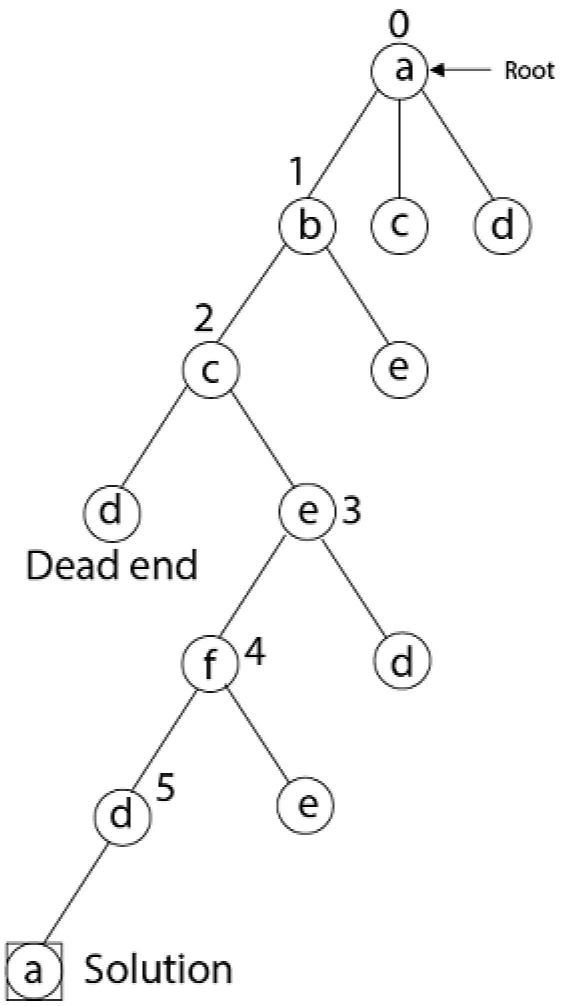




Next, we select 'c' adjacent to 'b.'



Next, we select 'd' adjacent to 'c.'



Here we have generated one Hamiltonian circuit, but another Hamiltonian circuit can also be obtained by considering another vertex.

N-Queen's Problem

- Consider n*n chessboard on which we have to place n queens so that no two queens attack each other by being in the same row or column or diagonal.

Algorithm:

```
N queen (k, n)
{
For i=1 to n do
{
If(place(row, column))then
{
Board[row] column
If(row = n) then
Print board(n)
else
Queen (row+1, n)
}
}
```

```
Place(row, column)
{
For i← 1 to row-1 do
{
If(board[i]=column) then
Return 0
Elseif((board[i] – column)-(i-row))then
Return0
}
return1
```