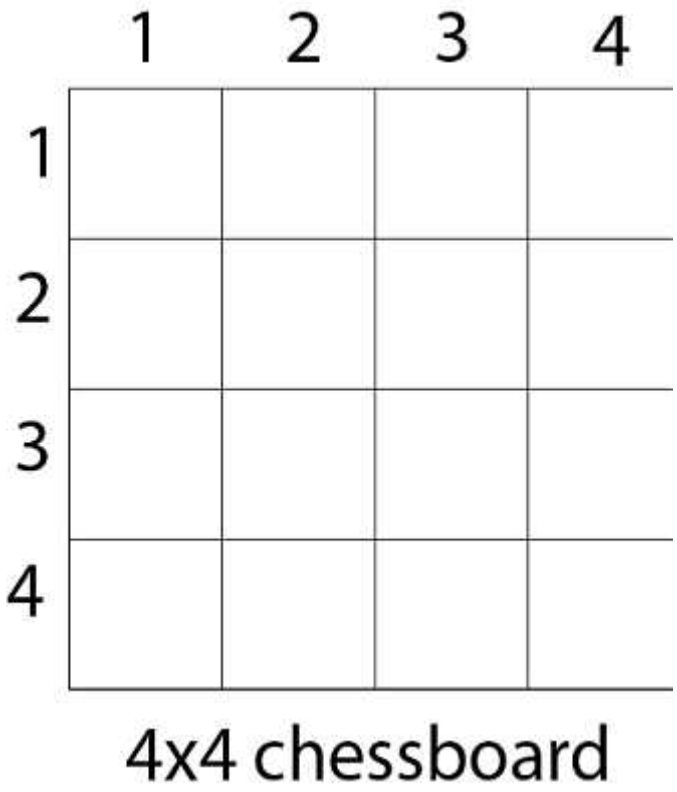
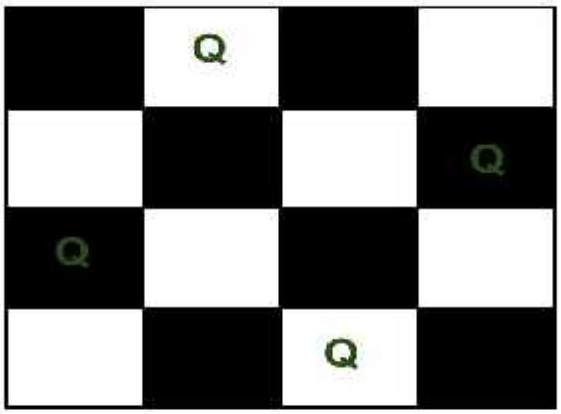
N-Queens Problem

N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.

It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - queens problem.

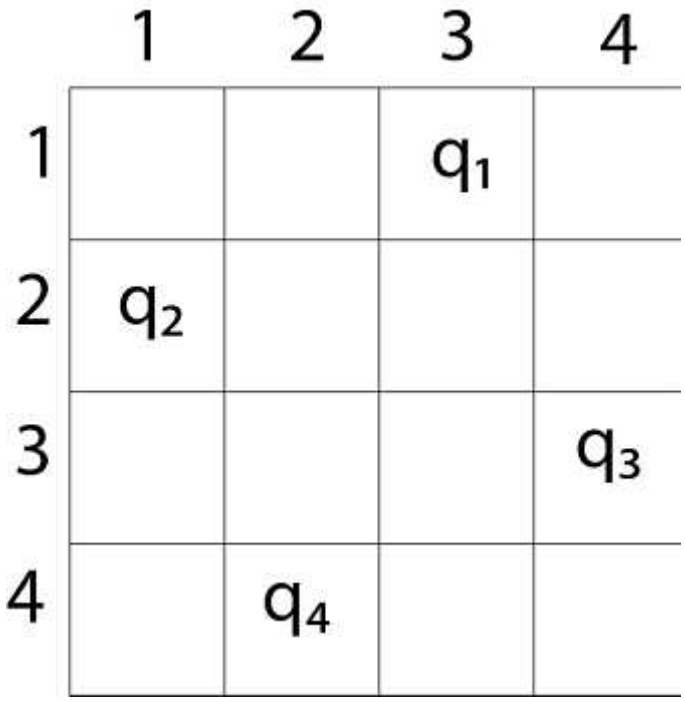
Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.





Since, we have to place 4 queens such as q_1 q_2 q_3 and q_4 on the chessboard, such that no two queens attack each other. In such a conditional each queen must be placed on a different row, i.e., we put queen "i" on row "i."

Now, we place queen q_1 in the very first acceptable position (1, 1). Next, we put queen q_2 so that both these queens do not attack each other. We find that if we place q_2 in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for q_2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q_3' safely. So we backtrack one step and place the queen 'q_2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q_3' which is (3, 2). But later this position also leads to a dead end, and no place is found where 'q_4' can be placed safely. Then we have to backtrack till 'q_1' and place it to (1, 2) and then all other queens are placed safely by moving q_2 to (2, 4), q_3 to (3, 1) and q_4 to (4, 3). That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem. For another possible solution, the whole method is repeated for all partial solutions. The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.


1
2
3
4
1
2
3
4
q_1
q_2
q_3
q_4

The implicit tree for 4 - queen problem for a solution (2, 4, 1, 3) is as follows:

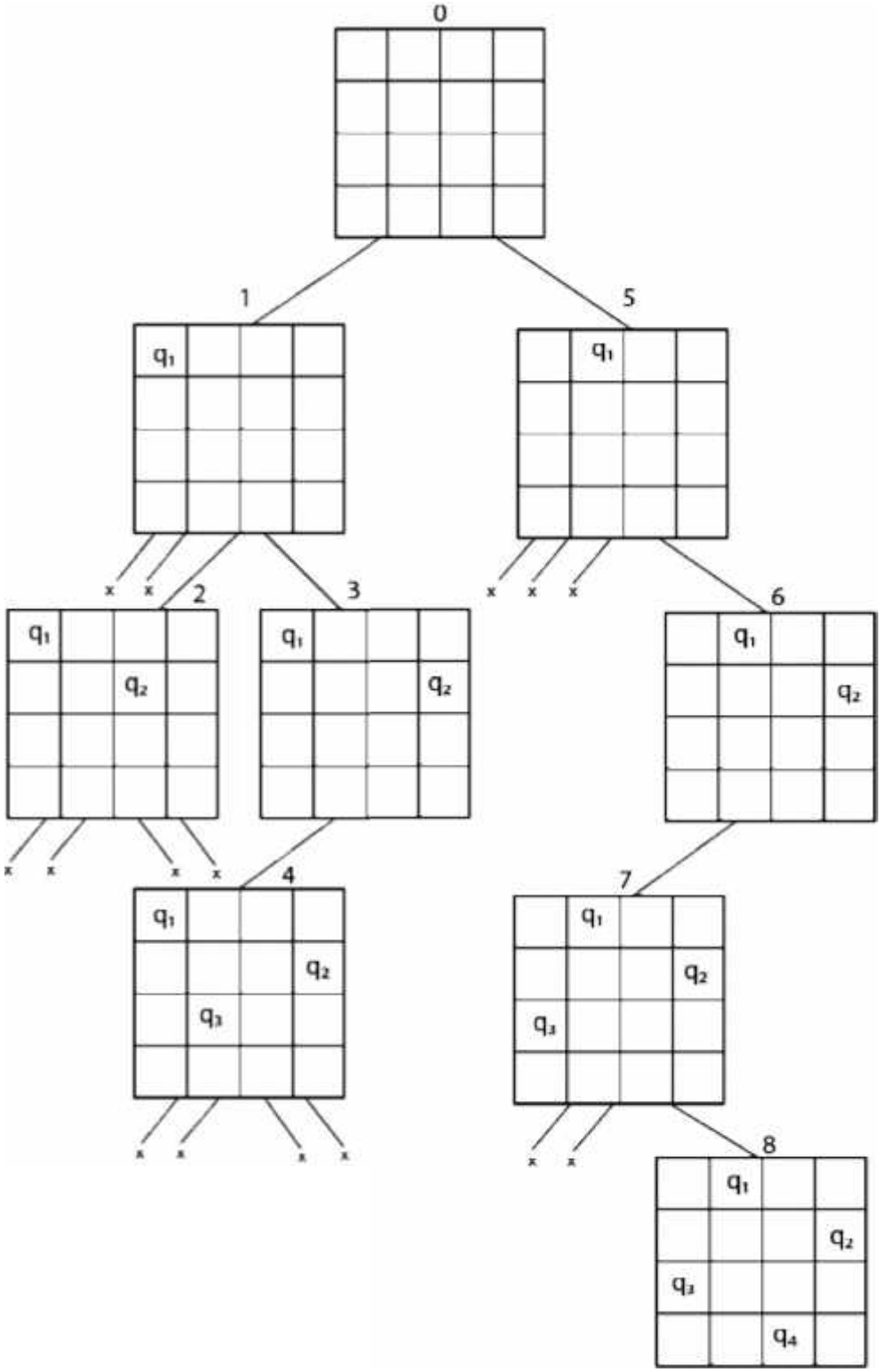
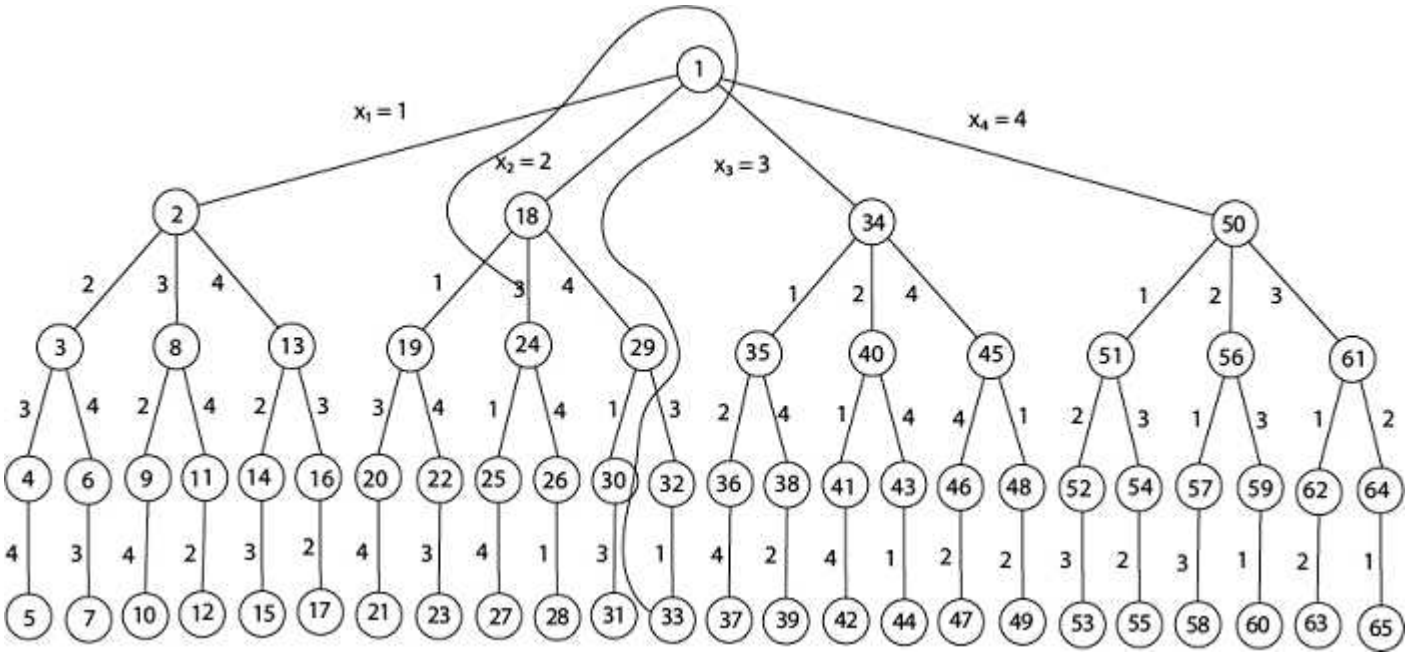


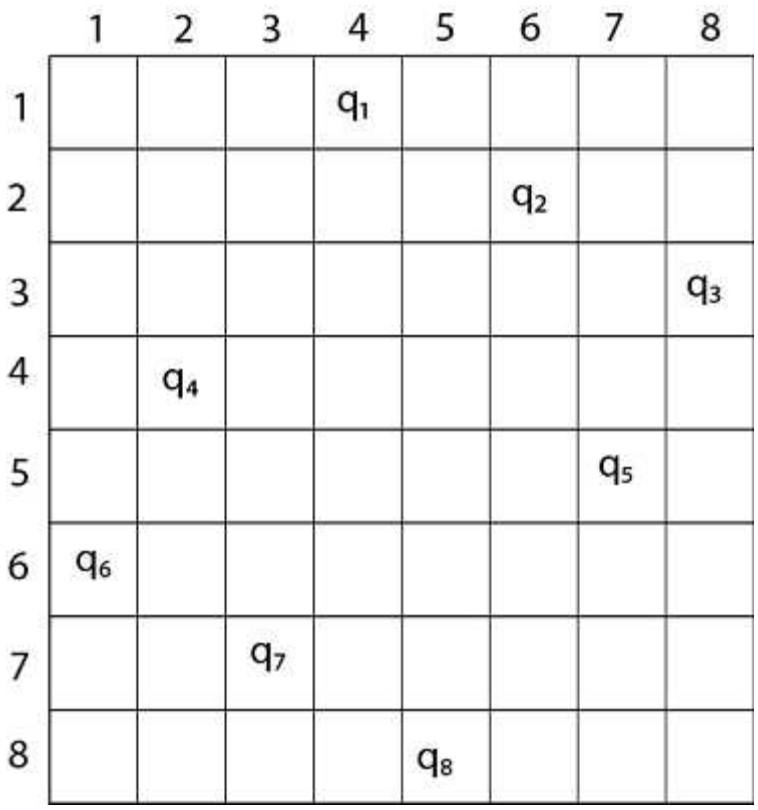
Fig shows the complete state space for 4 - queens problem. But we can use backtracking method to generate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.



4 - Queens solution space with nodes numbered in DFS

It can be seen that all the solutions to the 4 queens problem can be represented as 4 - tuples (x_1, x_2, x_3, x_4) where x_i represents the column on which queen "q_i" is placed.

One possible solution for 8 queens problem is shown in fig:



1. Thus, the solution **for** 8 -queen problem **for** (4, 6, 8, 2, 7, 1, 3, 5).
2. If two queens are placed at position (i, j) and (k, l).
3. Then they are on same diagonal only **if** (i - j) = k - l or i + j = k + l.
4. The first equation implies that j - l = i - k.
5. The second equation implies that j - l = k - i.
6. Therefore, two queens lie on the duplicate diagonal **if** and only **if** |j-l|=|i-k|

Place (k, i) returns a Boolean value that is true if the kth queen can be placed in column i. It tests both whether i is distinct from all previous costs x_1, x_2,....x_{k-1} and whether there is no other queen on the same diagonal.

Using place, we give a precise solution to then n- queens problem.

```
1.  Place (k, i)
2.    {
3.      For j  ←  1 to k - 1
4.       do if (x [j] = i)
5.        or (Abs x [j]) - i) = (Abs (j - k))
6.     then return false;
7.      return true;
8.  }
```

Place (k, i) return true if a queen can be placed in the kth row and ith column otherwise return is false.

x [] is a global array whose final k - 1 values have been set. Abs (r) returns the absolute value of r.

```
1.  N - Queens (k, n)
2.  {
3.    For i  ←  1 to n
4.         do if Place (k, i) then
5.    {
6.       x [k]  ←  i;
7.       if (k ==n) then
8.         write (x [1....n));
9.       else
10.      N - Queens (k + 1, n);
11.   }
12. }
```

Hamiltonian Circuit Problem

Hamiltonian path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path. Determine whether a given graph contains Hamiltonian Cycle or not. If it contains, then prints the path. Following are the input and output of the required function.

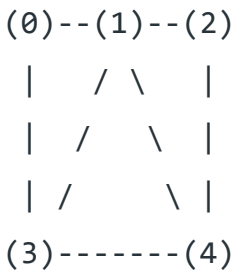
Input:

A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

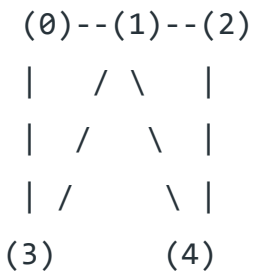
Output:

An array path[V] that should contain the Hamiltonian Path. path[i] should represent the ith vertex in the Hamiltonian Path. The code should also return false if there is no Hamiltonian Cycle in the graph.

For example, a Hamiltonian Cycle in the following graph is {0, 1, 2, 4, 3, 0}.



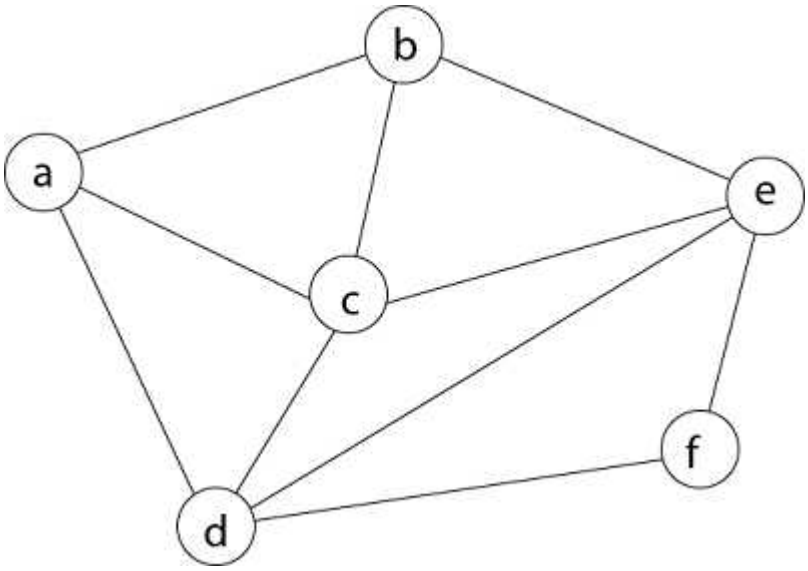
And the following graph doesn't contain any Hamiltonian Cycle.



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

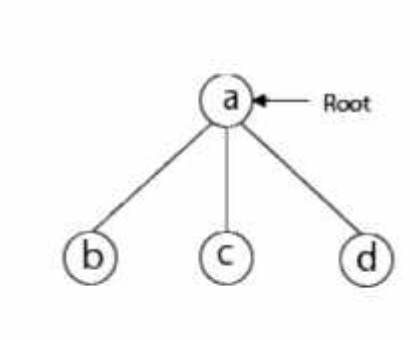
with any vertex other than vertex 'a' then we say that **dead end** is reached. In this case, we backtrack one step, and again the search begins by selecting another vertex and backtrack the element from the partial; solution must be removed. The search using backtracking is successful if a Hamiltonian Cycle is obtained.

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

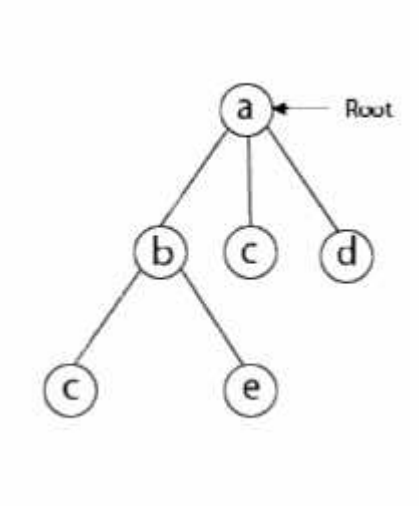


Solution: Firstly, we start our search with vertex 'a.' this vertex 'a' becomes the root of our implicit tree.

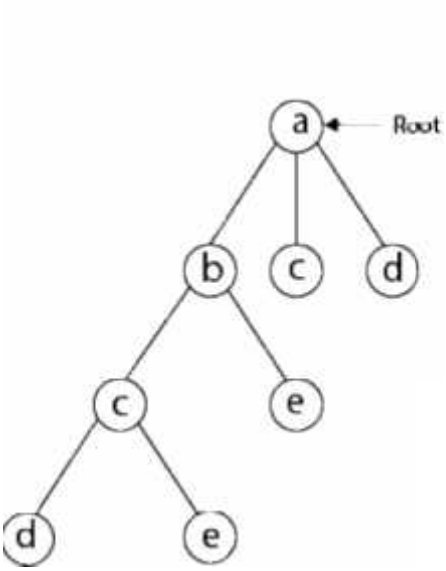
Next, we choose vertex 'b' adjacent to 'a' as it comes first in lexicographical order (b, c, d).



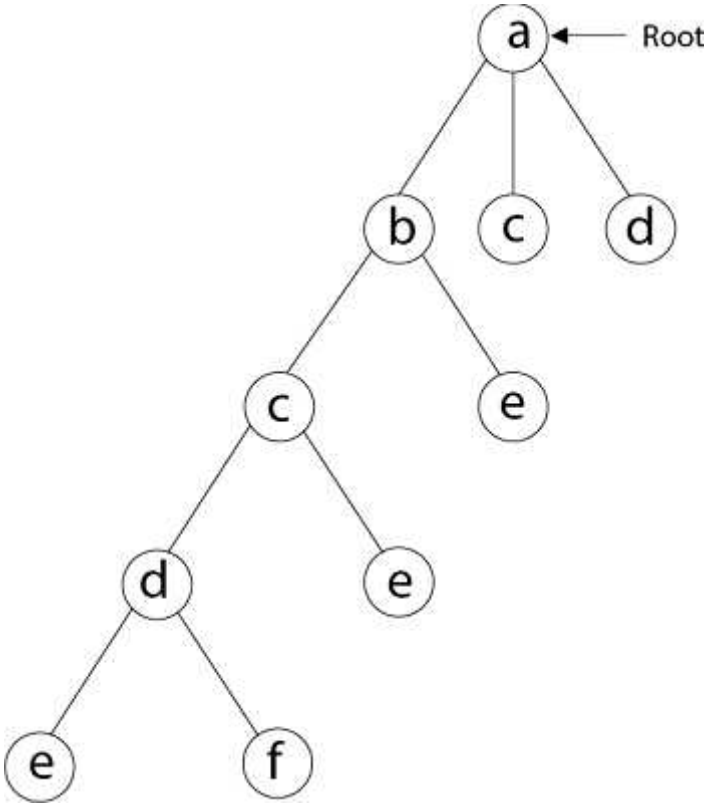
Next, we select 'c' adjacent to 'b.'



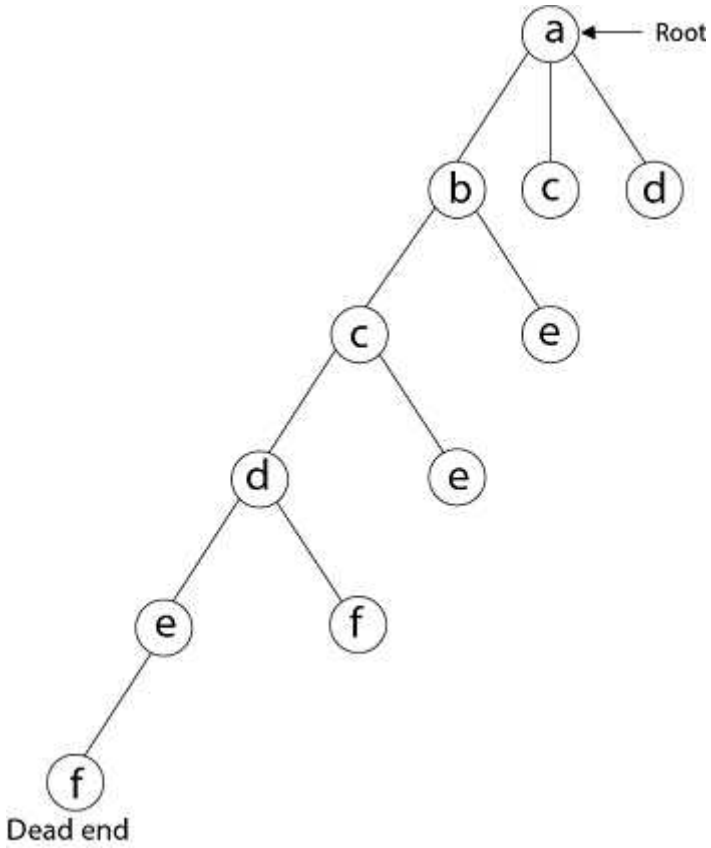
Next, we select 'd' adjacent to 'c.'



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

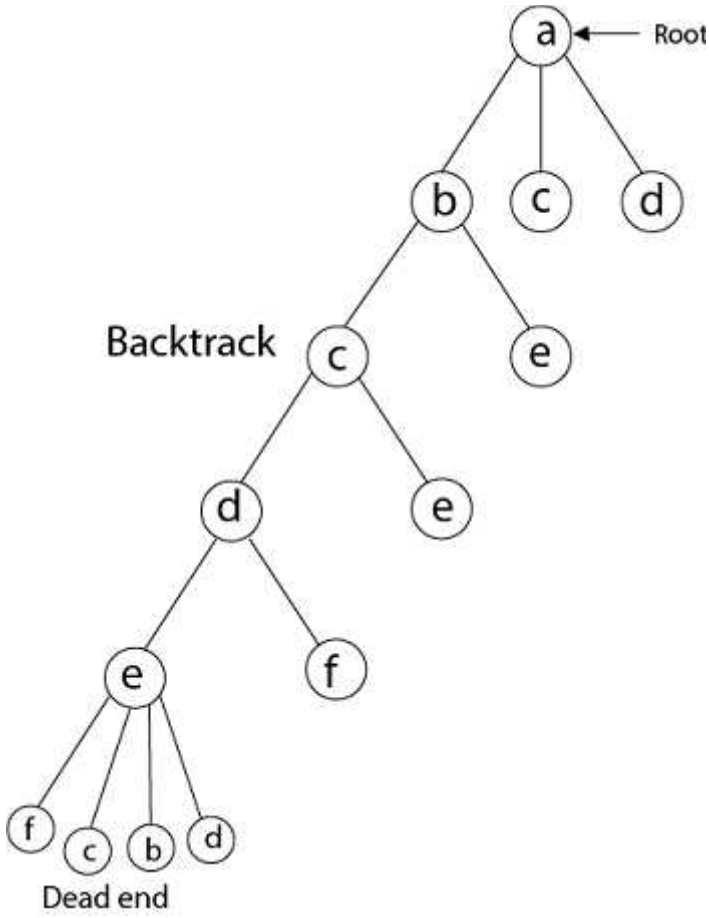


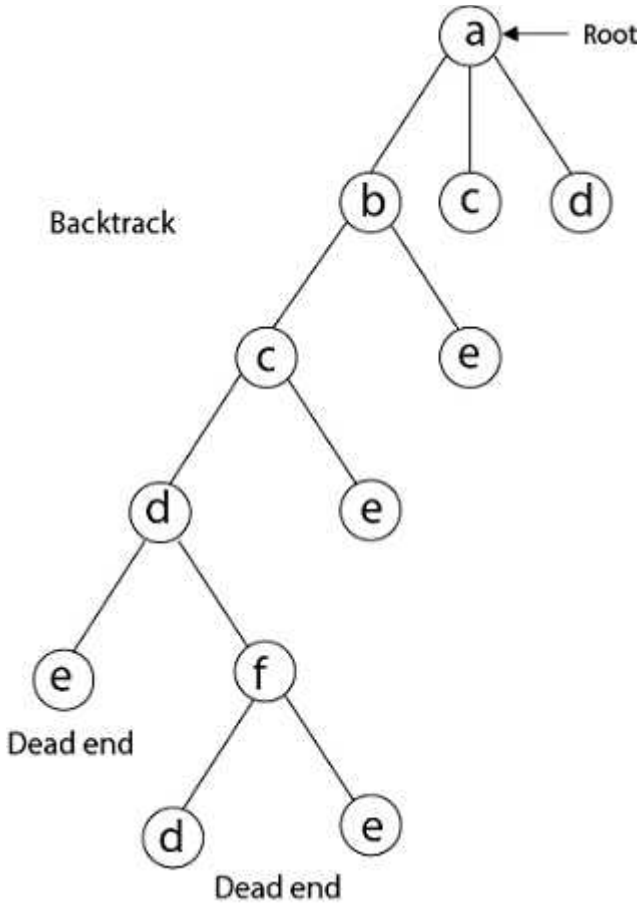
Next, we select vertex 'f' adjacent to 'e.' The vertex adjacent to 'f' is d and e, but they have already visited. Thus, we get the dead end, and we backtrack one step and remove the vertex 'f' from partial solution.



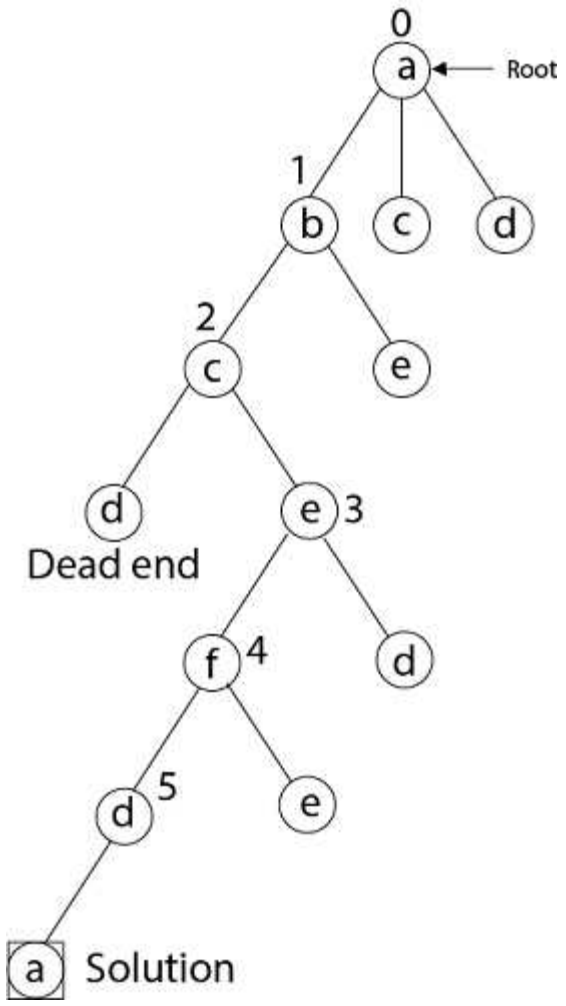
From backtracking, the vertex adjacent to 'e' is b, c, d, and f from which vertex 'f' has already been checked, and b, c, d have already visited. So, again we backtrack one step. Now, the vertex adjacent to d are e, f from which e has already been checked, and adjacent of 'f' are d and e. If 'e' vertex, revisited them we get a dead state. So again we backtrack one step.

Now, adjacent to c is 'e' and adjacent to 'e' is 'f' and adjacent to 'f' is 'd' and adjacent to 'd' is 'a.' Here, we get the Hamiltonian Cycle as all the vertex other than the start vertex 'a' is visited only once. (a - b - c - e - f -d - a).


a
Root
b
c
d
Backtrack
c
e
d
e
e
f
f
c
b
d
Dead end



Again Backtrack



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