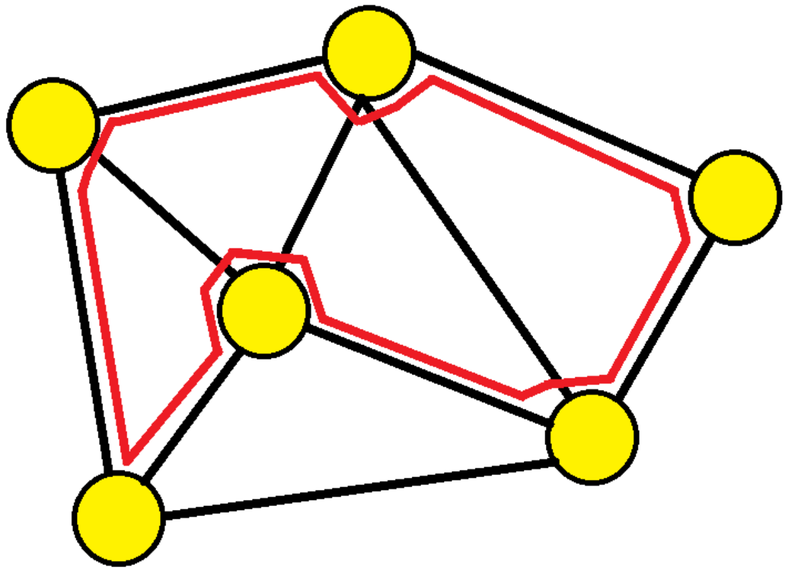


DAA PRESENTATION

Hamiltonian Cycle

Simran Kumari(A018)
Yash Mange(A019)
Jitesh Mishra(A020)
Amaan Mukadam(A021)
Kartik Padave(A022)

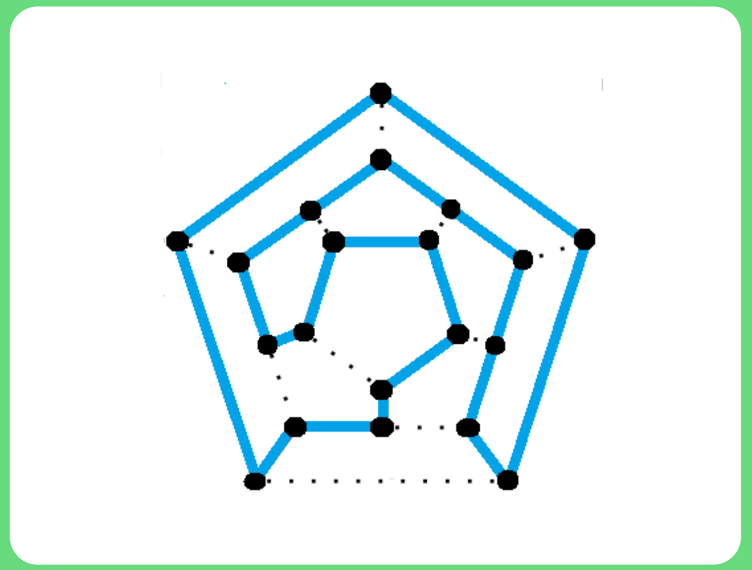
Contents

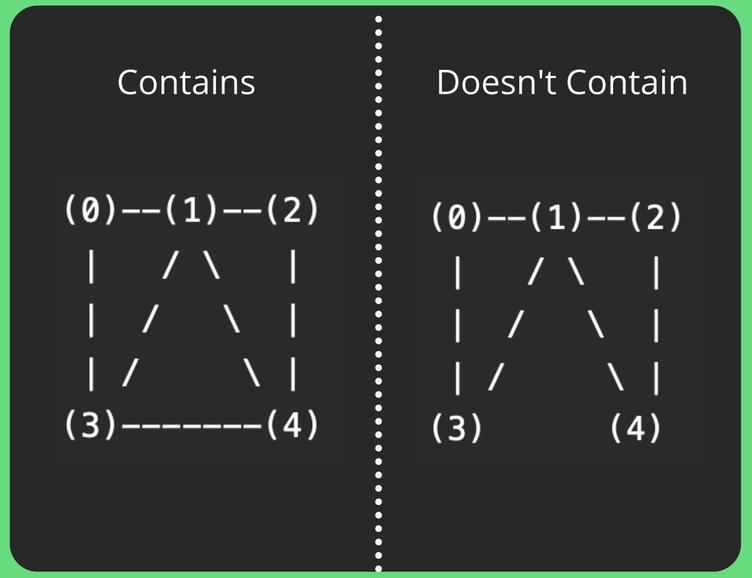
POINTS FOR DISCUSSION:

- Introduction to Hamiltonian Cycle
- Applications of Hamiltonian Cycle
- Backtracking
- Backtracking vs Heuristic Approach
- Backtracking Example
- Detailed Explaination with Example
- Solve Along Example

Hamiltonian Cycle

- The Hamiltonian cycle is named after the famous Irish mathematician Sir William Rowan Hamilton.
- A Hamiltonian cycle is a closed-loop on a graph where every node (vertex) is visited exactly once.
- Knight's Tour Problem and the Traveling Salesman Problem, involves finding a Hamiltonian Cycle.
- Finding a Hamiltonian cycle in any given graph is an NP-Complete problem.
- A Hamiltonian cycle of a graph can be computed efficiently in the Wolfram Language using FindHamiltonianCycle[g].

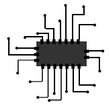




Applications of Hamiltonian Cycle

Computer Graphics

Hamiltonian Cycle application involves stripification of triangle meshes in computer graphics.



Electronic circuit

Power gating design, which can effectively reduce the leakage power by turning off inactive circuit domains.

Navigation

Hamiltonians cycle is used to plan the best route. People considered nodes, the paths between them edges, and the vehicle to travel a route that will pass through each location once.

Mapping Genomes

Mapping genomes scientists must combine many tiny fragments of genetic code where each of the reads are considered nodes in a graph and each overlap and is considered to be an edge.

Lets learn about

"Backtracking"

What is BackTracking?

Backtracking is a technique based on an algorithm to solve a problem. It uses "Recursive Calling" to find the solution by building a solution which, step by step, increasing values with time. It removes the options that doesn't give rise to the valid solution of the problem based on the constraints given to solve the respective problem.

Backtracking a Hamiltonian cycle

01 It searches for all possible solutions in the graph looking for a Hamiltonian cycle.

02 These algorithms find the solution by building up the path vertex by vertex, by calling the procedure Recursive_Backtrack with the path built in the procedure so far as input. The number of active Recursive_Backtrack function calls is equal to the length of the current path -1.

03 These algorithms check every option in the search space to find a Hamiltonian cycle, and so they will not return until they find a Hamiltonian cycle or until it is clear that no Hamiltonian cycle exists.

04 If this procedure is not able to extend the path, or the path obtained does not allow the Hamiltonian cycle to exist, then the procedure backtracks by removing the vertex addedand repeating the above process, by selecting a different unvisited neighbor of the end point.

- Recurse(Path p, endpoint e)
- While (e has unvisited neighbors)
- GetNewNode x (add x node to P)
- PruneGraph. (Prune graph. If result graph
- does not permit a HC forming, remove x from P and continue)
- FormCycle (If P includes all nodes then
- try to form cycle. Fail, remove x and continue
- Success, return success)
- BackTrack Recurse(P,x)
- if not,
- Return fail.

Backtracking Algorithm

(A BASIC EXPLANATION)

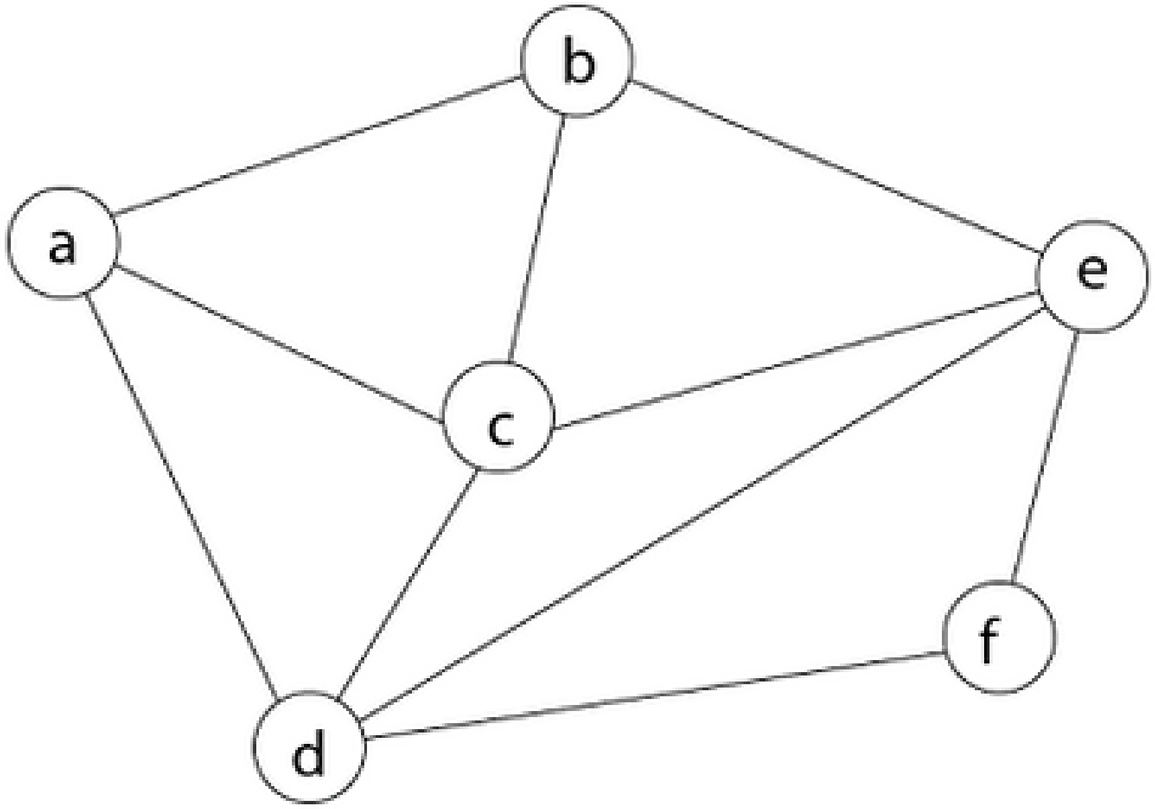
Backtracking Algorithm

- These algorithms will find the Hamiltonian cycles if the input graph has one, or will determine that no Hamiltonian cycle exists in the graph. It searches for all possible solutions in the graph looking for a Hamiltonian cycle.
- This takes comparatively more time but gives out a guaranteed output
- These algorithms find the solution by building up the path vertex by vertex, by calling the procedure Recursive_Backtrack with the path built in the procedure so far as input
- If this procedure is not able to extend the path, or the path obtained does not allow the Hamiltonian cycle to exist, then the procedure backtracks by removing the vertex added and repeating the above process, by selecting a different unvisited neighbor of the end point.

Heuristic Approach

- These heuristics help the algorithm to execute quickly, but the algorithm cannot guarantee that it will find a solution even if one exists.
- This is really Quick but less trustworthy as it's not sure to show a cycle even if one exists.
- The Hamiltonian cycle algorithms using a heuristic approach always try to extend the partial solution path, i.e., the length of the partial solution is always non-decreasing. No backtracking is ever done in these algorithms.
- If the path formed contains all the vertices of the graph, then the algorithm tries to form a cycle. If the cycle can be formed successfully, then the algorithm returns success. If the cycle cannot be formed, then the algorithm returns failure.

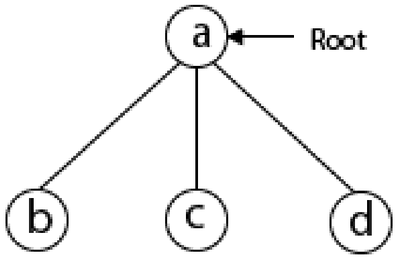
Example: Consider a graph G = (V, E) shown in fig. we have to find a Hamiltonian circuit



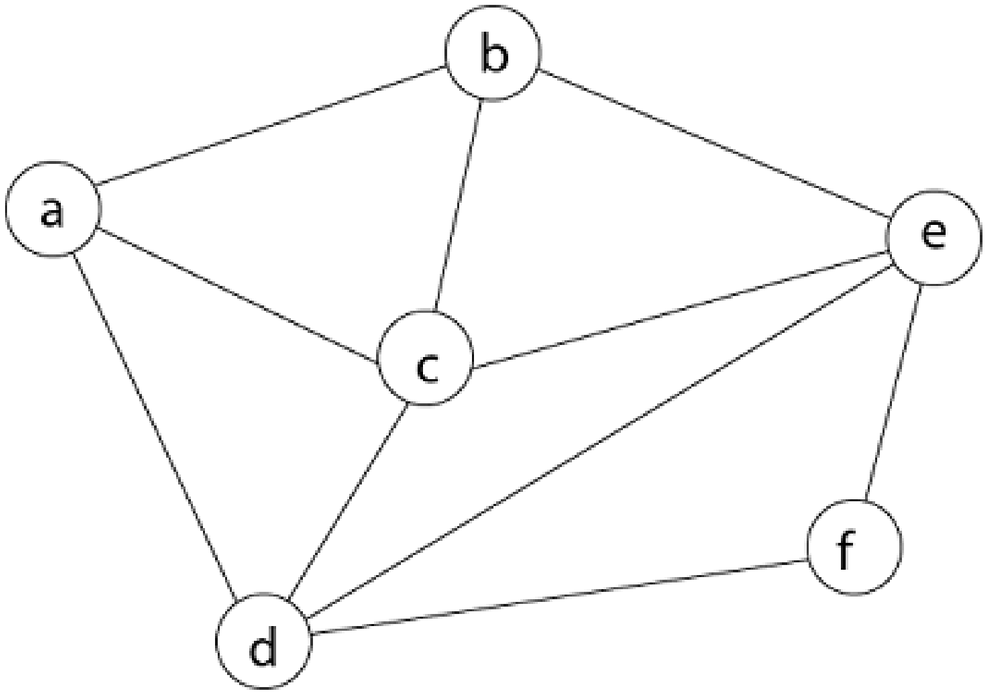
Let us understand backtracking with an Example

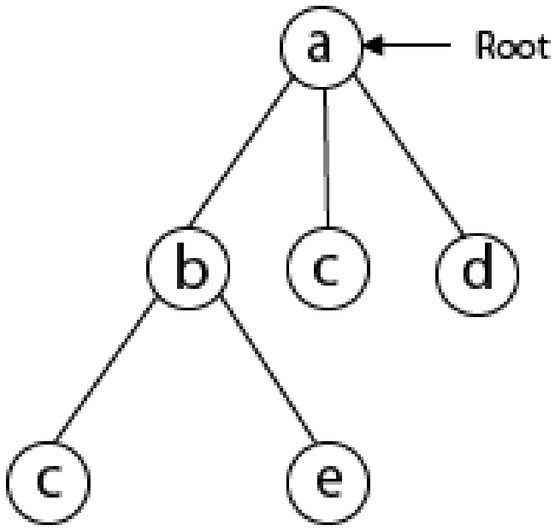
Solution: Firstly, we start our search with vertex 'a.' this vertex 'a' becomes the root of our implicit tree.

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

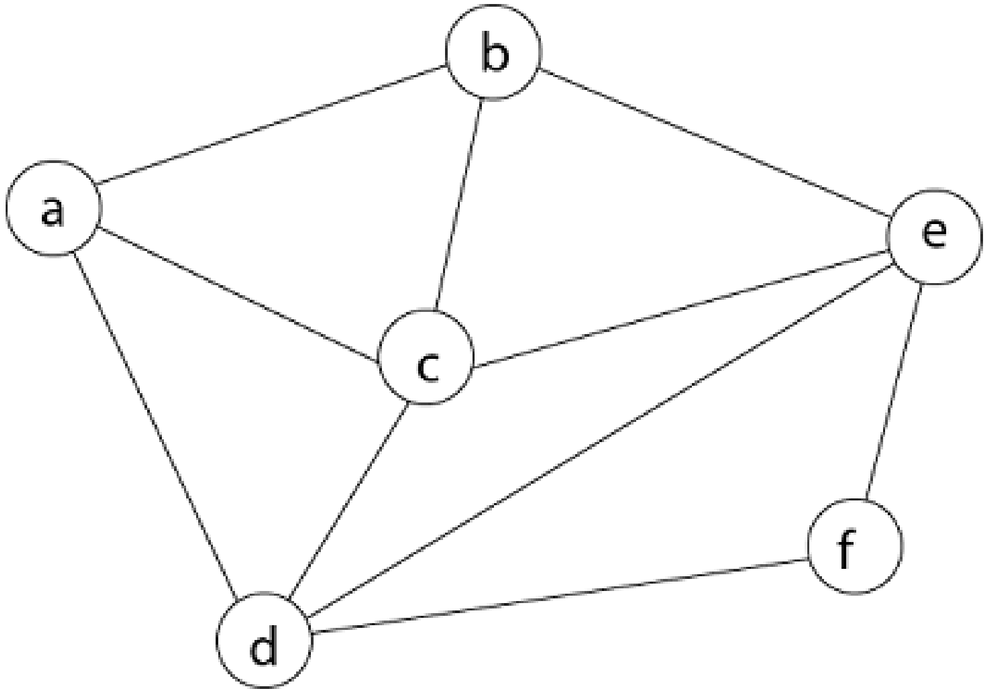


Next, we select 'c' adjacent to 'b.'

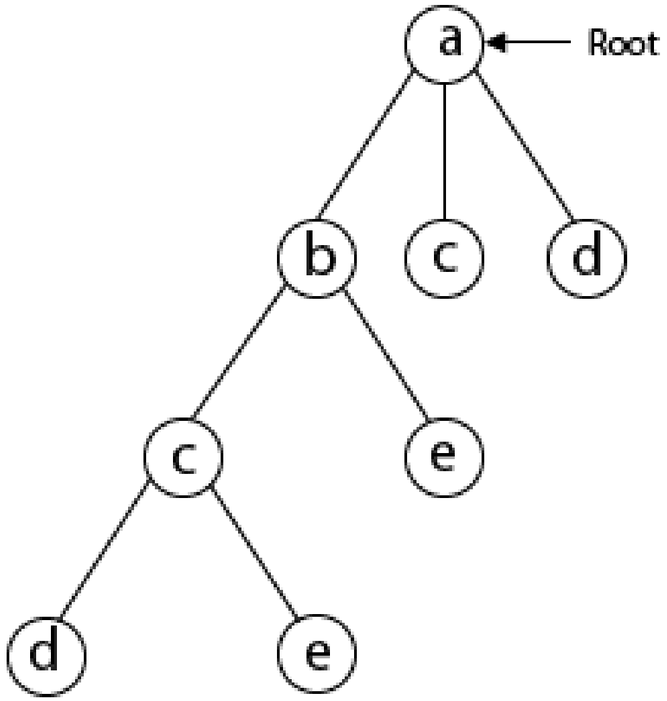


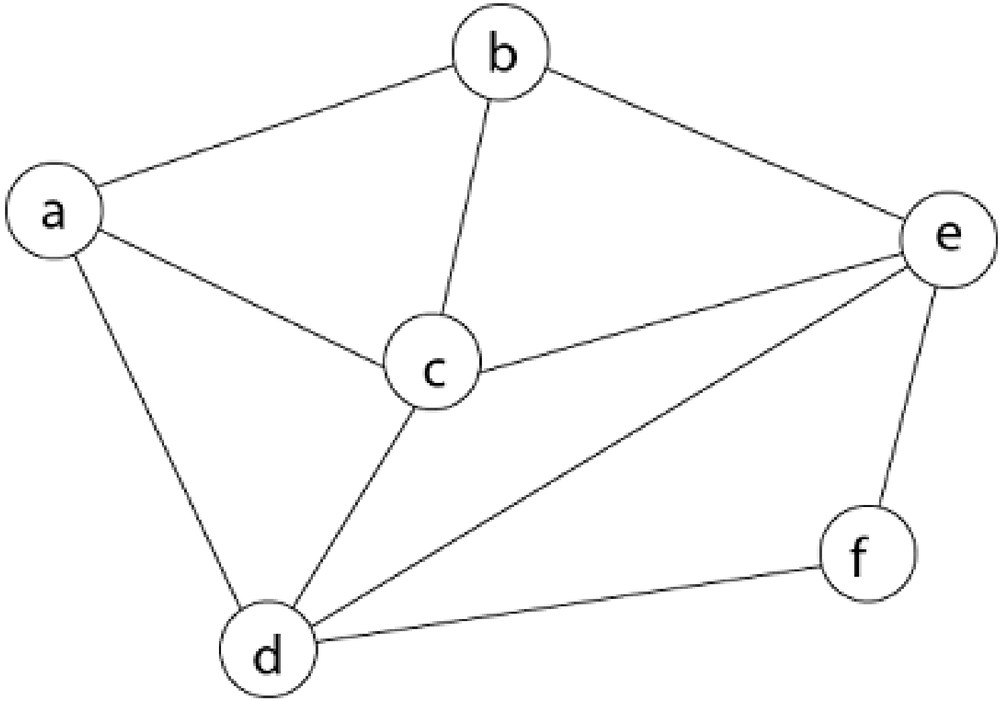


Next, we select 'd' adjacent to 'c.'

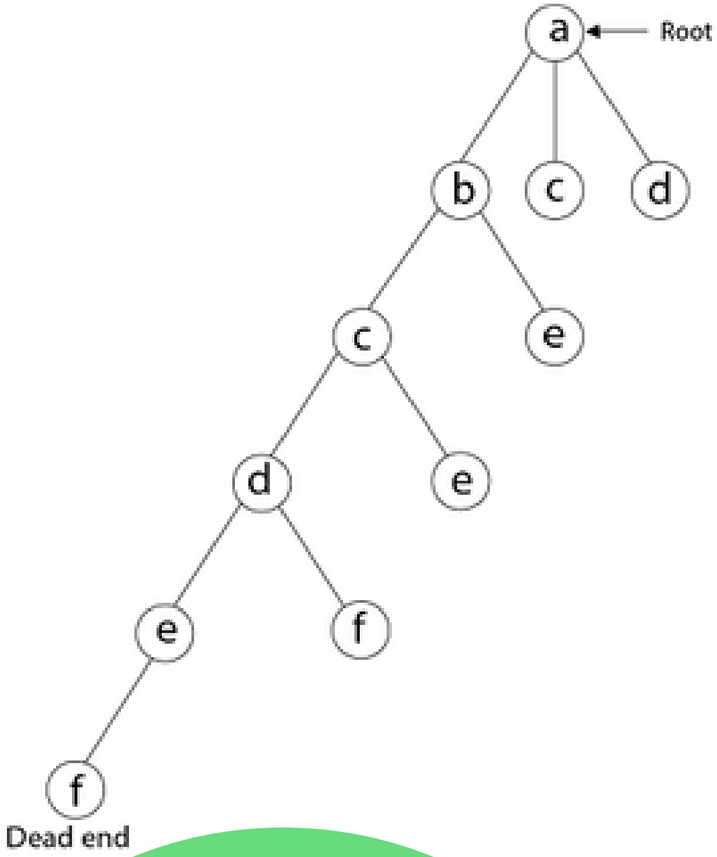


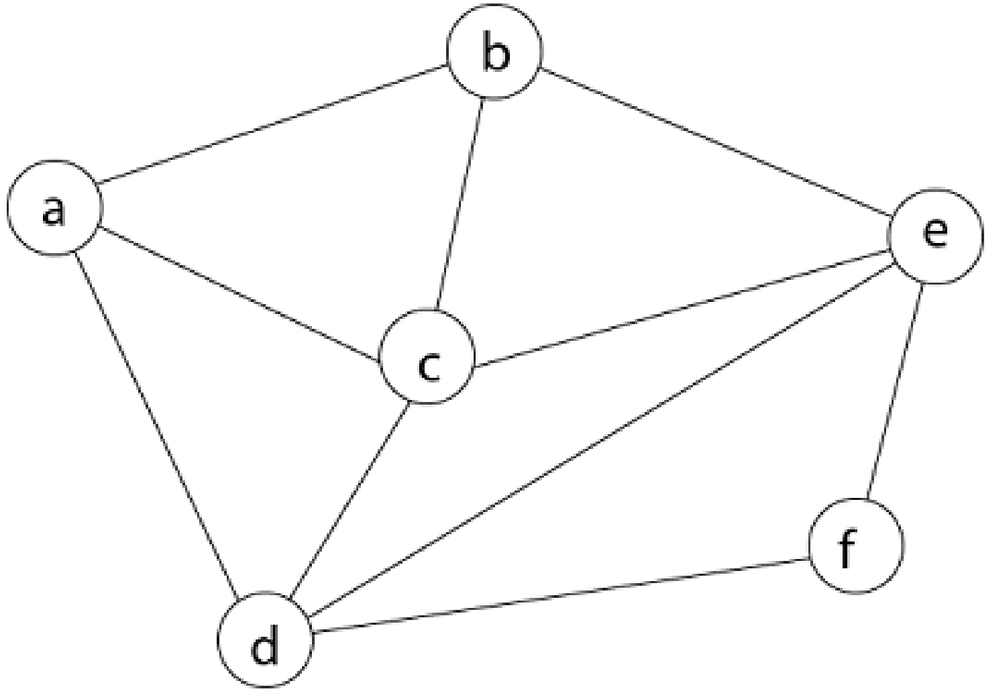
Next, we select 'd' adjacent to 'c.'

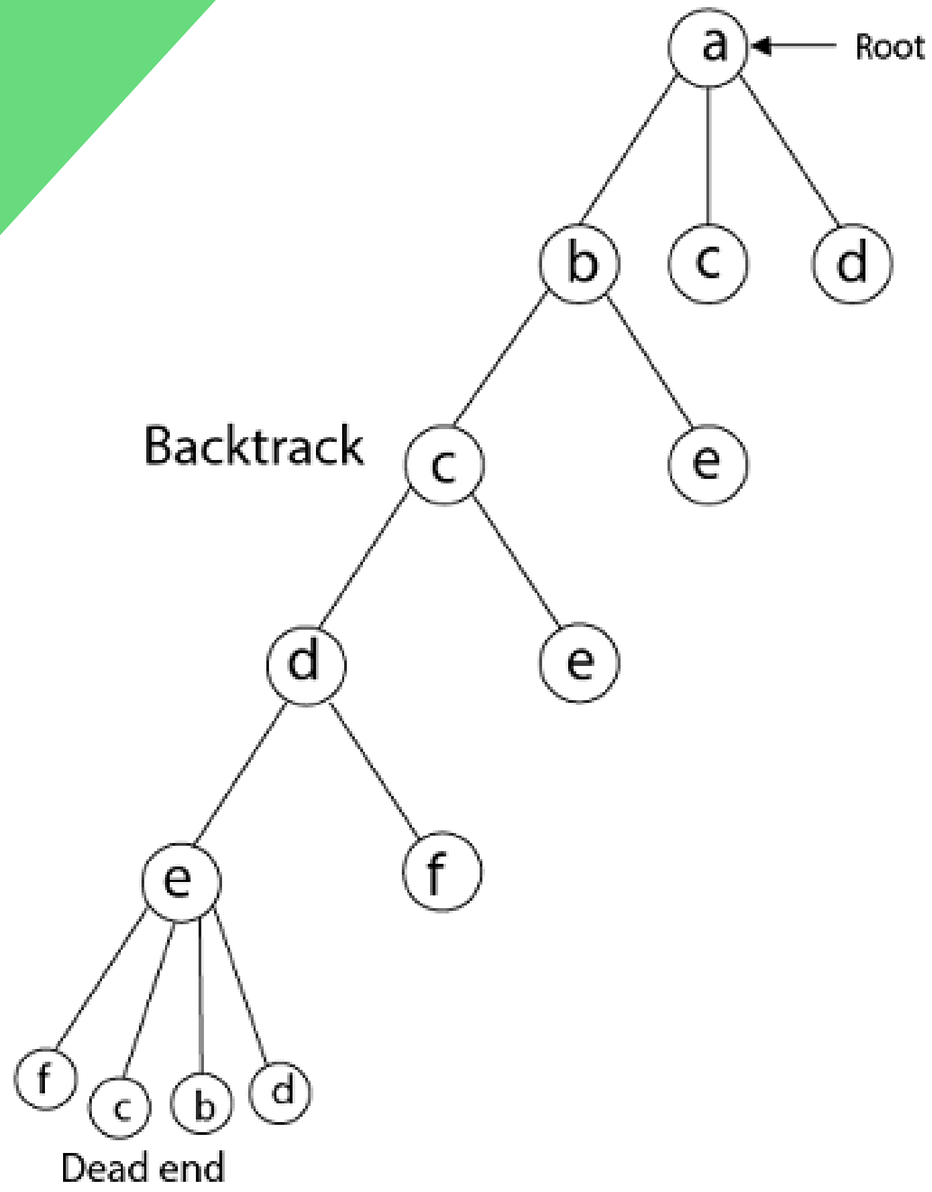


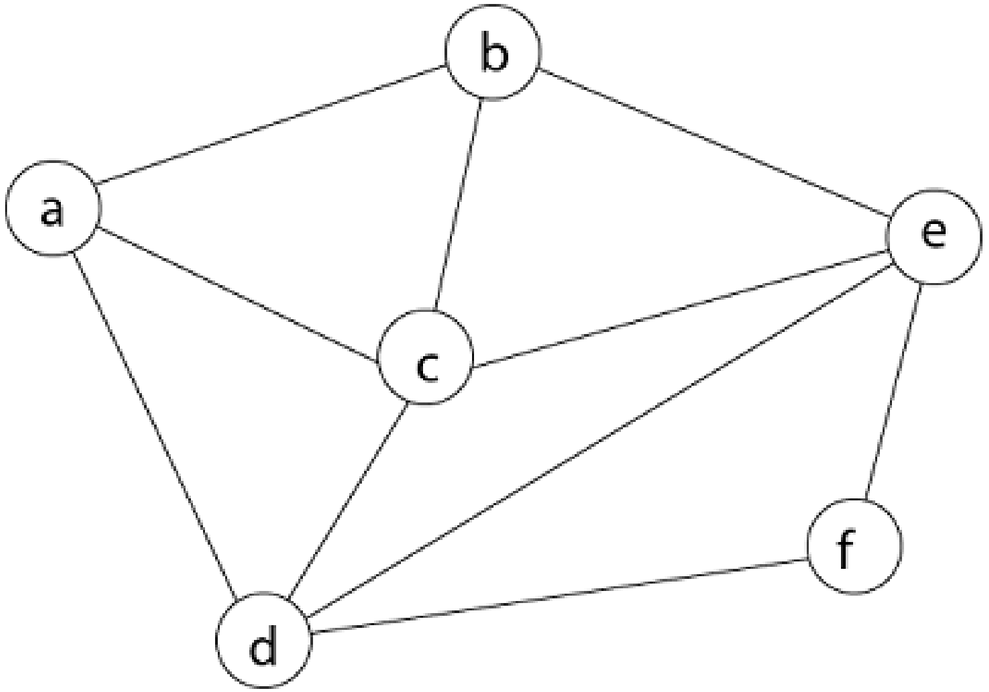


Next, we select vertex 'f' adjacent to 'e.' The vertex adjacent to 'f' is d and e, but they have already visited. Thus, we get the dead end, and we backtrack one step and remove the vertex 'f' from partial solution.

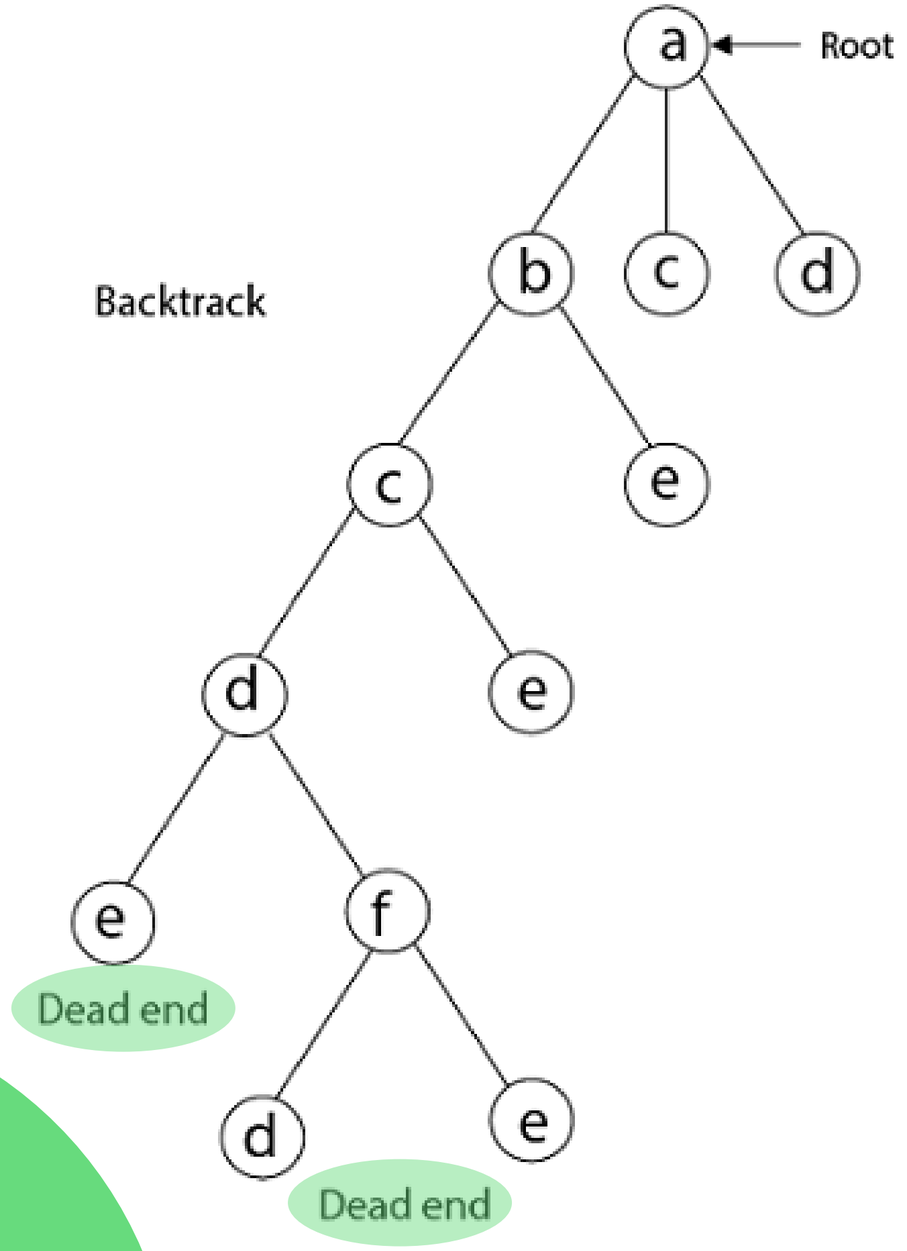


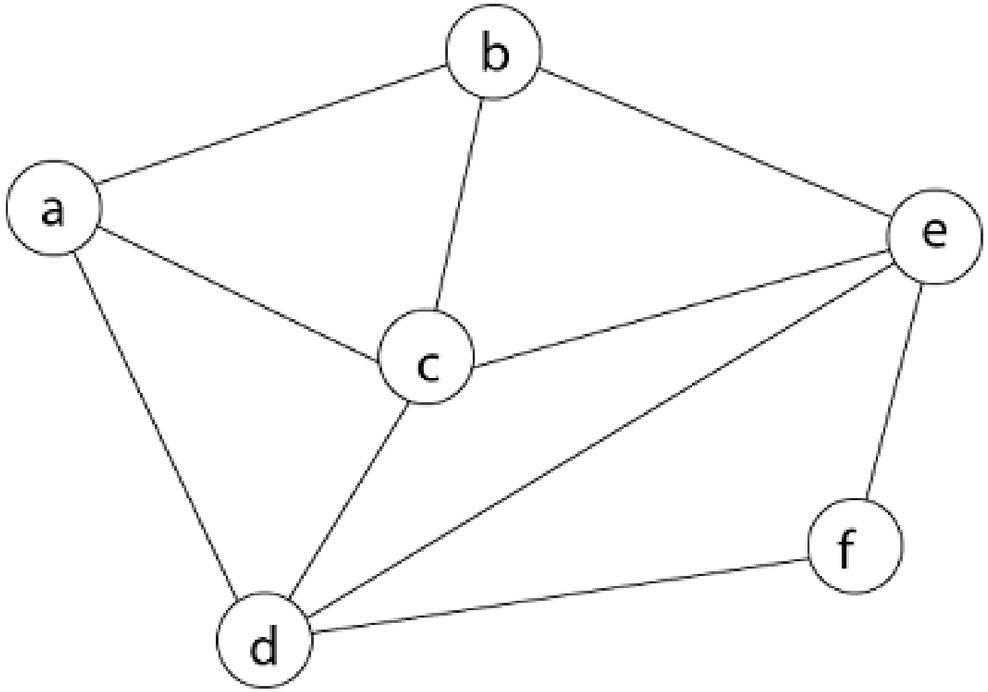



a
Root
b
c
d
Backtrack
c
e
d
e
e
f
f
c
b
d
Dead end

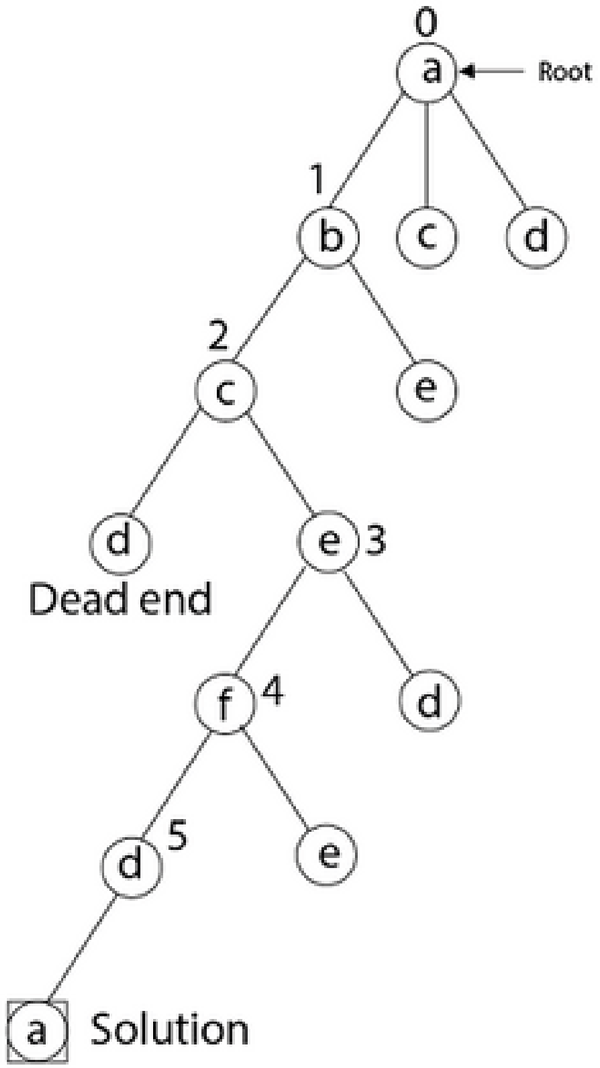

b
a
e
c
f
d

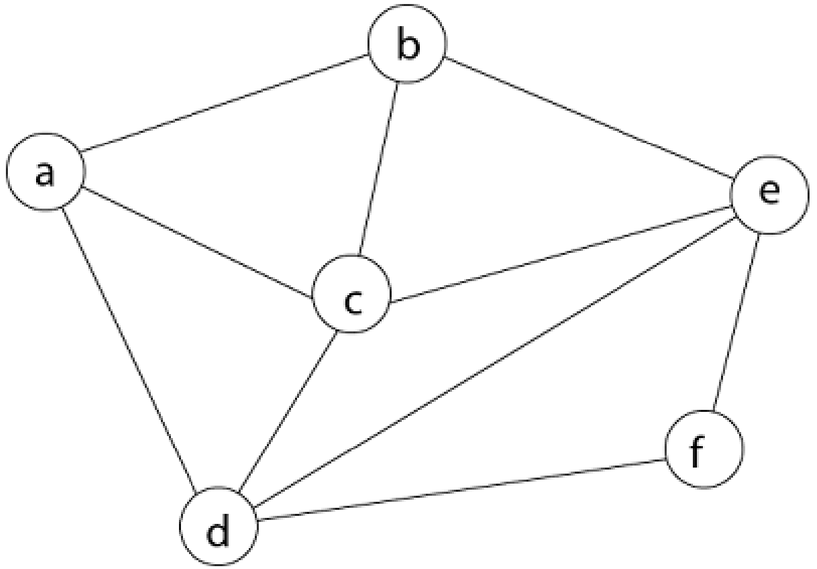
Backtrack





Again Backtrack



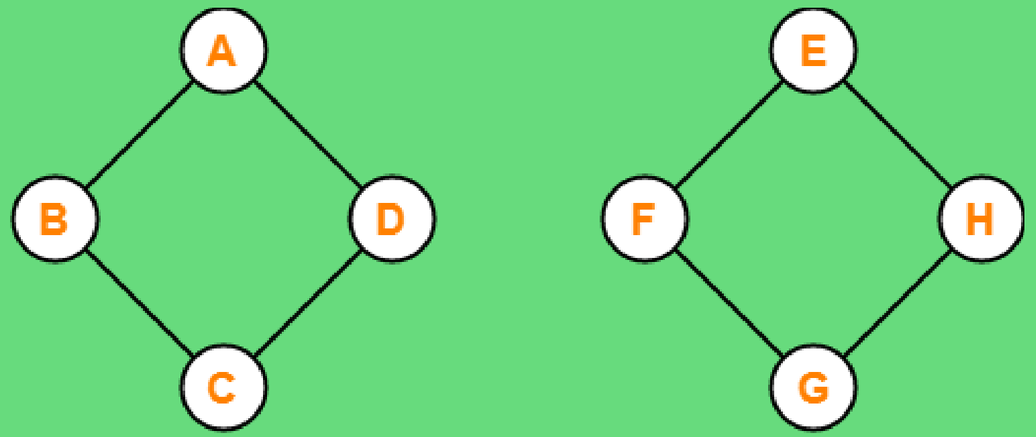


Here we have generated one Hamiltonian circuit, but another Hamiltonian circuit can also be obtained by considering another vertex.

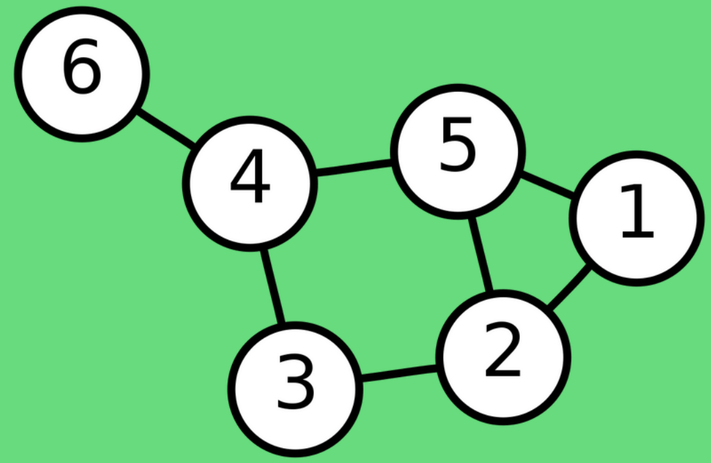
That was how backtracking can be done using trees.

Global Checking:

- The graph is checked for connectivity. If the graph is not connected then no Hamiltonian cycle can exist.

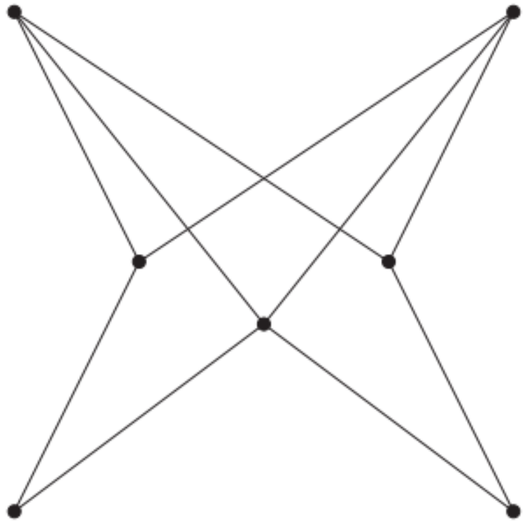


- The graph is checked for a vertex with degree one (Pendant Vertex). No path can exist through a vertex with degree one and so no Hamiltonian cycle can exist.

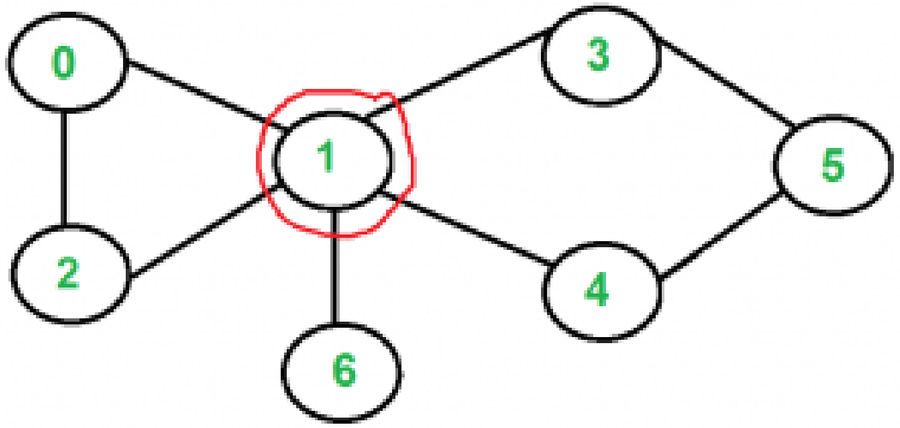


Global Checking:

- The graph is checked whether it is a bipartite graph with unequal partition sets. If this is the case for the given graph then no Hamiltonian cycle can exist.

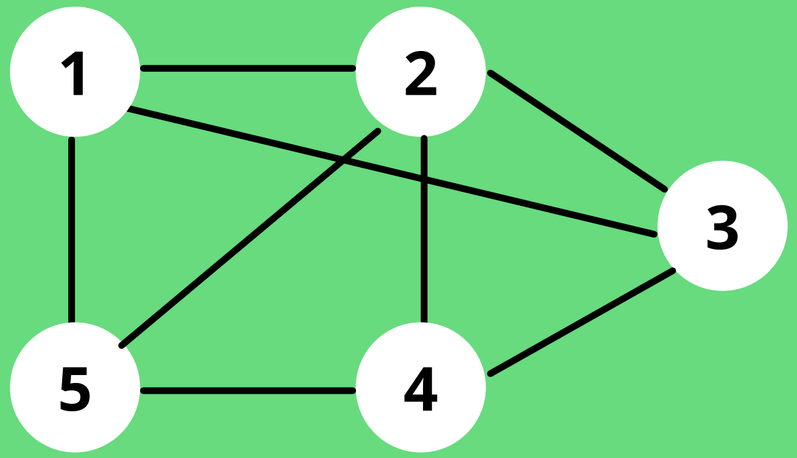


- The graph is checked for a cut vertex (Articulation Point). If a graph contains a cut point no Hamiltonian cycle can exist.



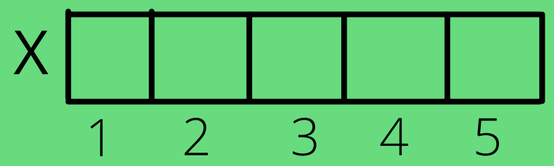
Articulation Point is 1

Example:

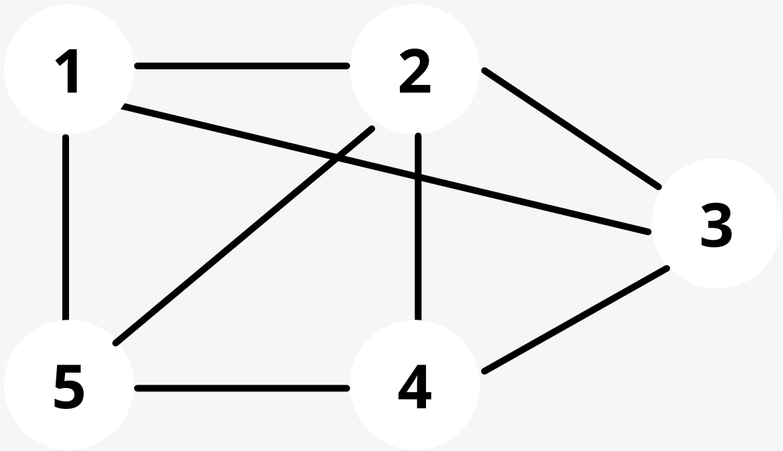


G=

	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0




```
-> Hamiltonian (k)                          //For backtracking
      Repeat until false:
        Next value (k)
        if (x[k] == 0) return
        if ( x[k] == n) write x[1:n]
        else Hamiltonian (k+1)

-> Next value (k)                           //For finding next node
      Repeat until false:
        X [k] = (x[k] + 1 mod (n+1))
        if (x[k] == 0) return
        if (G(x[k-1]), x[k] =! 0) then
          for j=1 to k-1
            if ( x[ j ] == x[ k ]) break
            if ( j == k) true
            if ( k<n) or ( k = n ) and G(x[ n ], x[1] != 0) return
```



G=

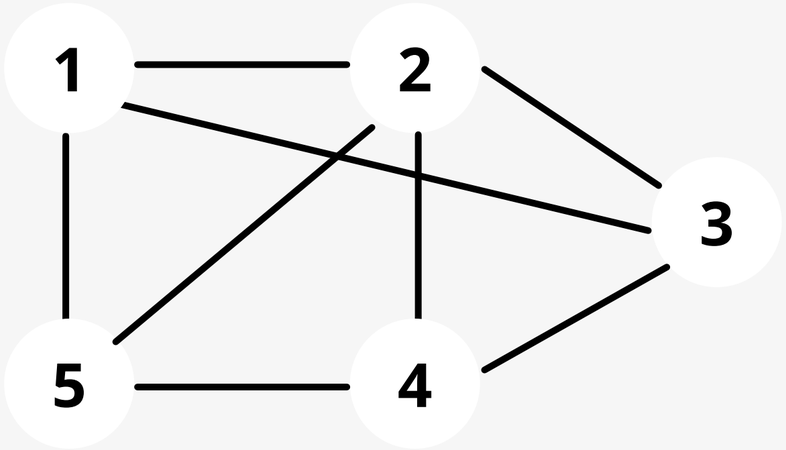
	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

X

1	2	3	4	5
0	**0**	**0**	**0**	**0**

Duplicate Cycles: 1-2-5-4-3-1

2-5-4-3-1-2



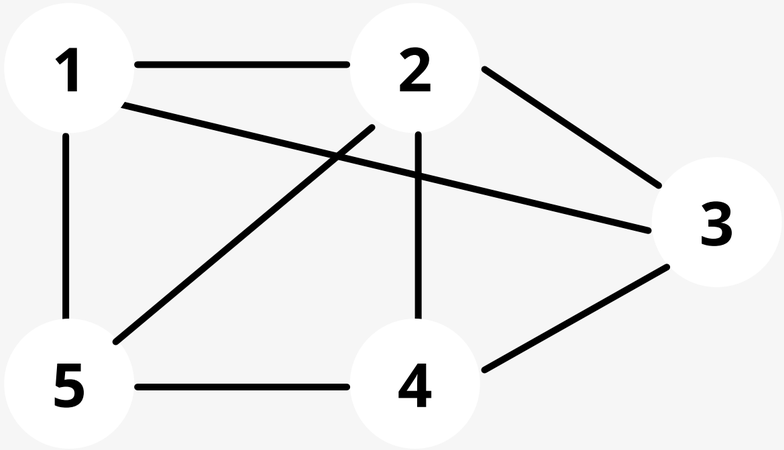
1

G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

X

1	2	3	4	5
1	**0**	**0**	**0**	**0**



X

1	1	0	0	0
1	2	3	4	5

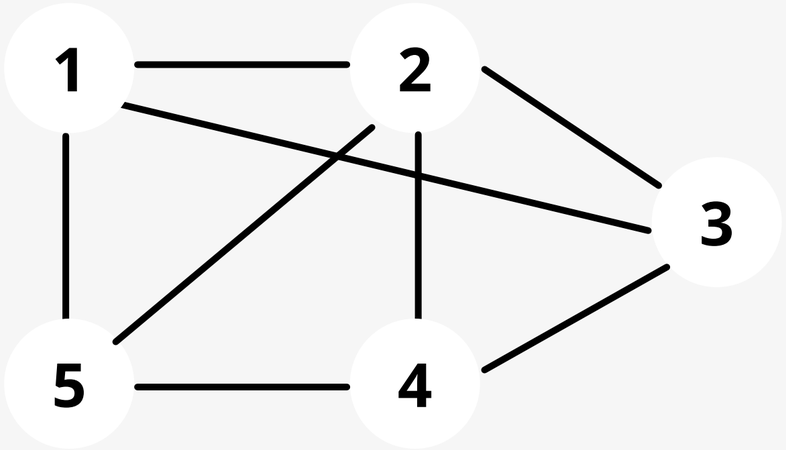
1

X

1	2	0	0	0
1	2	3	4	5

G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

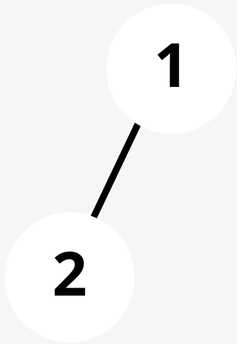


X

1	2	0	0	0
1	2	3	4	5

X

1	2	1	0	0
1	2	3	4	5



G=

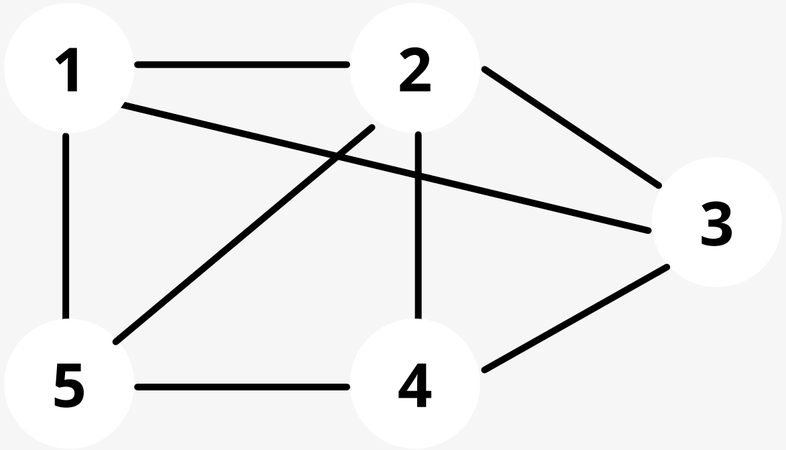
	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

X

1	2	2	0	0
1	2	3	4	5

X

1	2	3	0	0
1	2	3	4	5



G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

X

1	**2**	**0**	**0**	**0**
1	2	3	4	5

X

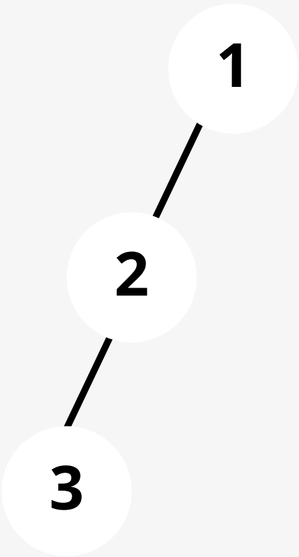
1	**2**	**1**	**0**	**0**
1	2	3	4	5

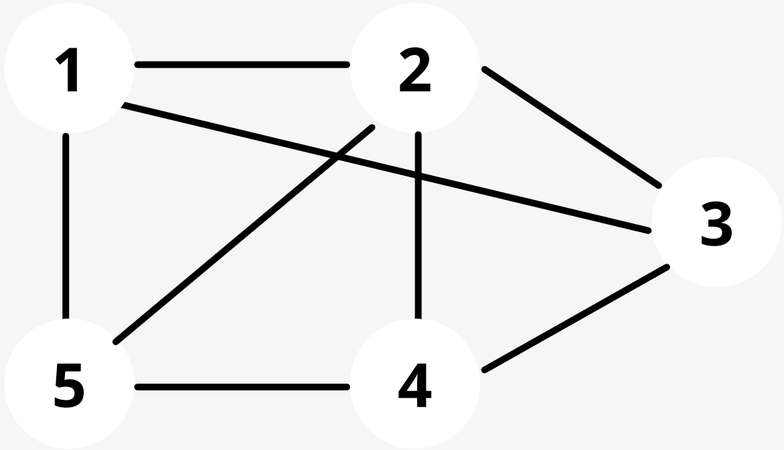
X

1	**2**	**2**	**0**	**0**
1	2	3	4	5

X

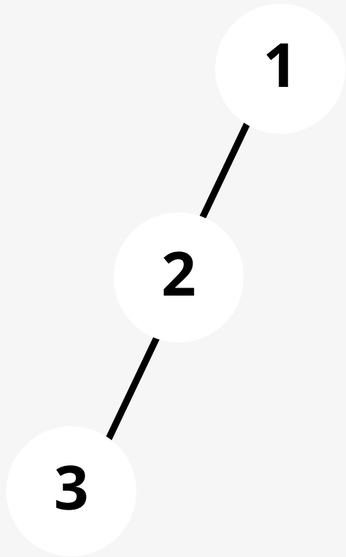
1	**2**	**3**	**0**	**0**
1	2	3	4	5





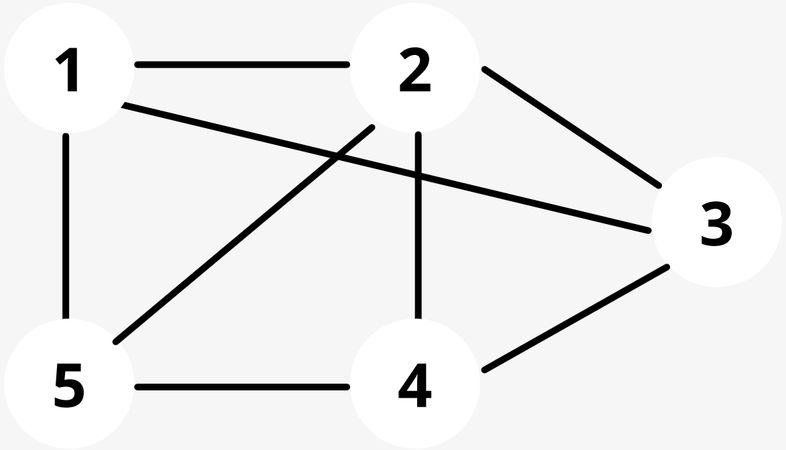
X	1	2	3	0	0
	1	2	3	4	5

X	1	2	3	4	0
	1	2	3	4	5



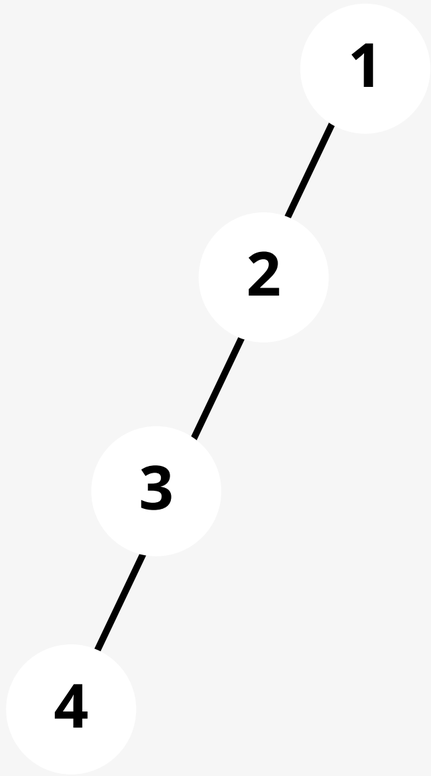
G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**



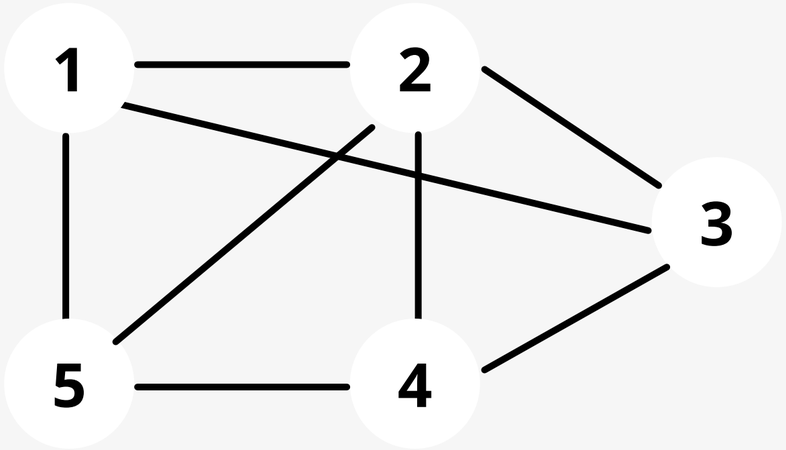
X	1	2	3	0	0
	1	2	3	4	5

X	1	2	3	4	0
	1	2	3	4	5



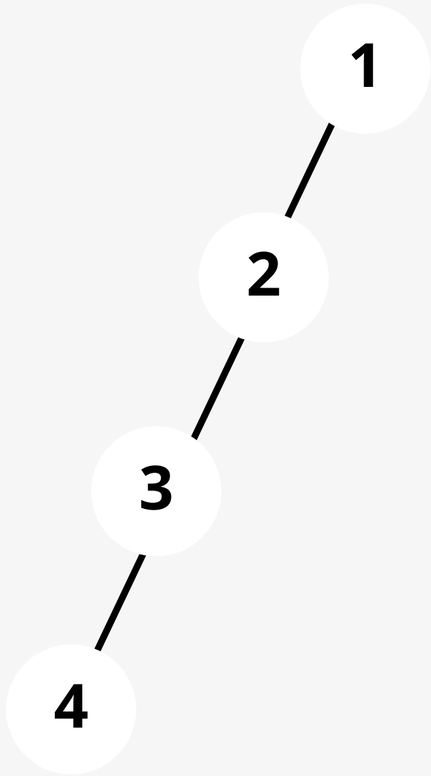
G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**



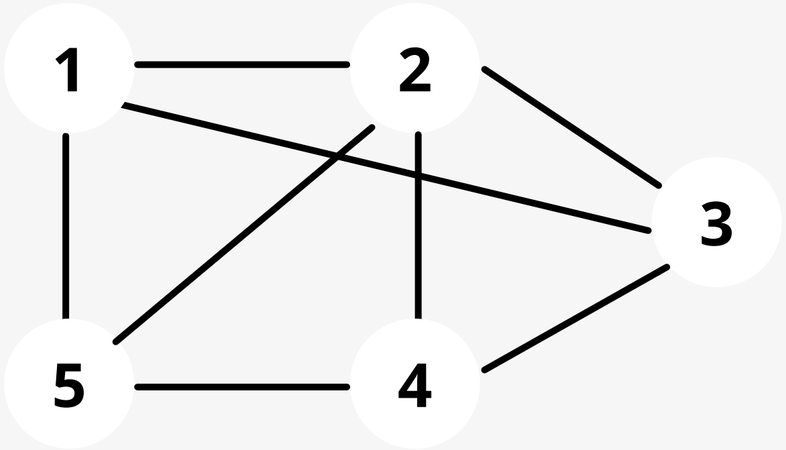
X	1	2	3	4	0
	1	2	3	4	5

X	1	2	3	4	5
	1	2	3	4	5



G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

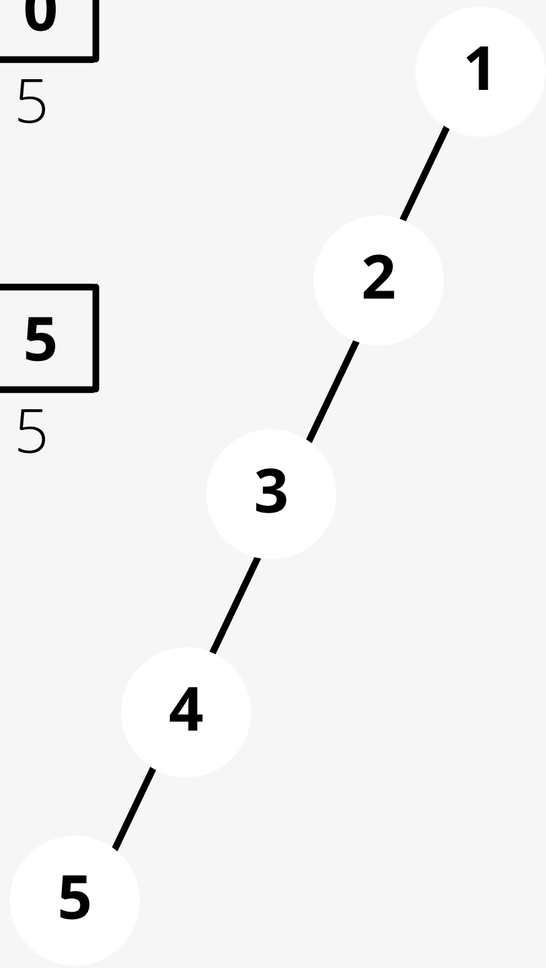


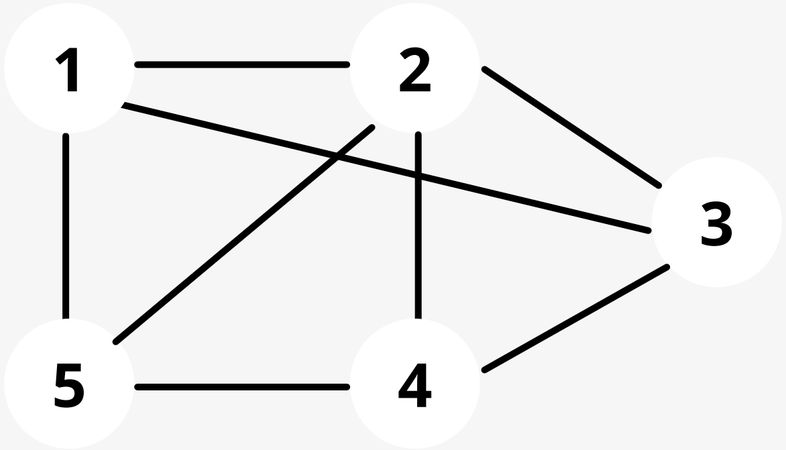
X	1	2	3	4	0
	1	2	3	4	5

X	1	2	3	4	5
	1	2	3	4	5

G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**



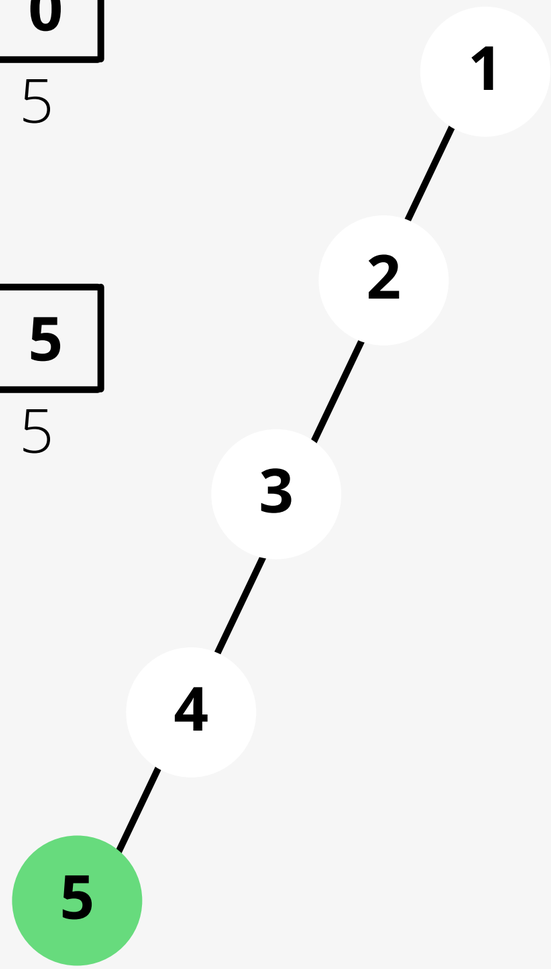


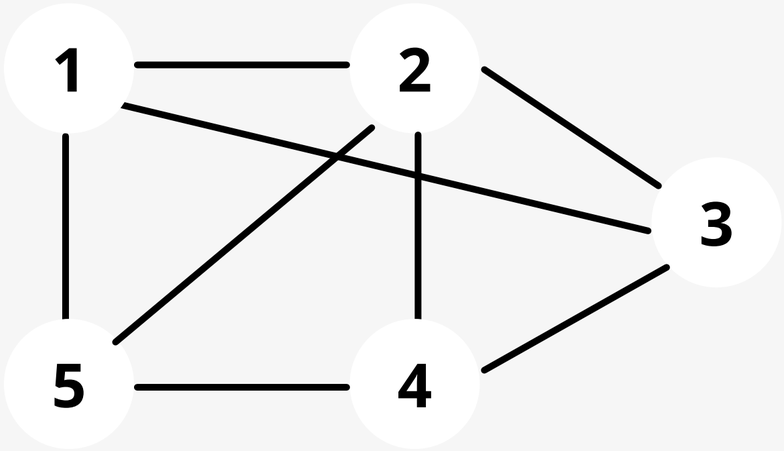
X	1	2	3	4	0
	1	2	3	4	5

X	1	2	3	4	5
	1	2	3	4	5

G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**



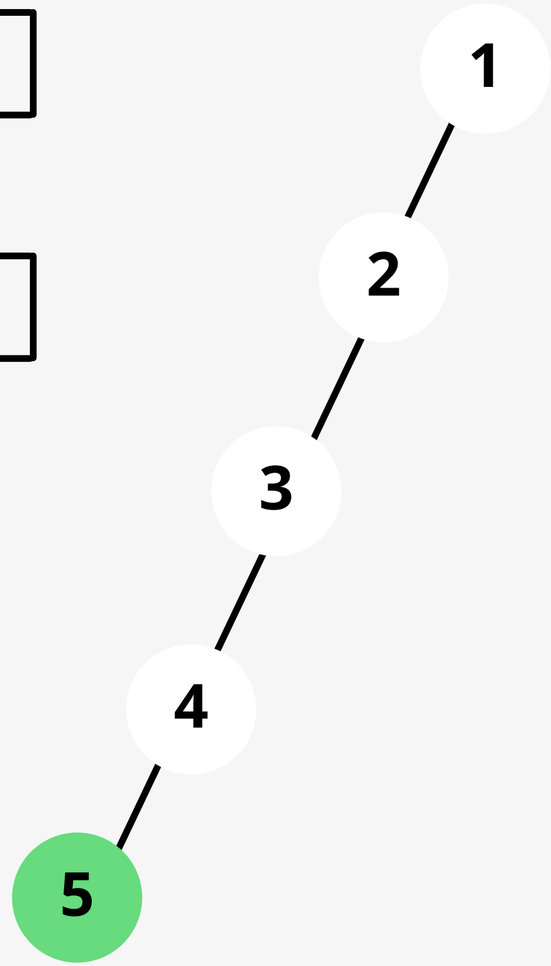


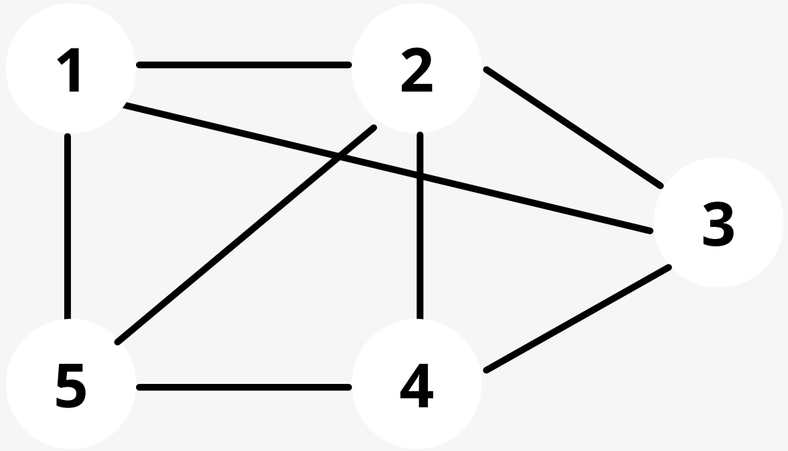
X	1	2	3	4	5
	1	2	3	4	5

X	1	2	3	4	0
	1	2	3	4	5

G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**





G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

X

1	**2**	**3**	**4**	**5**
1	2	3	4	5

X

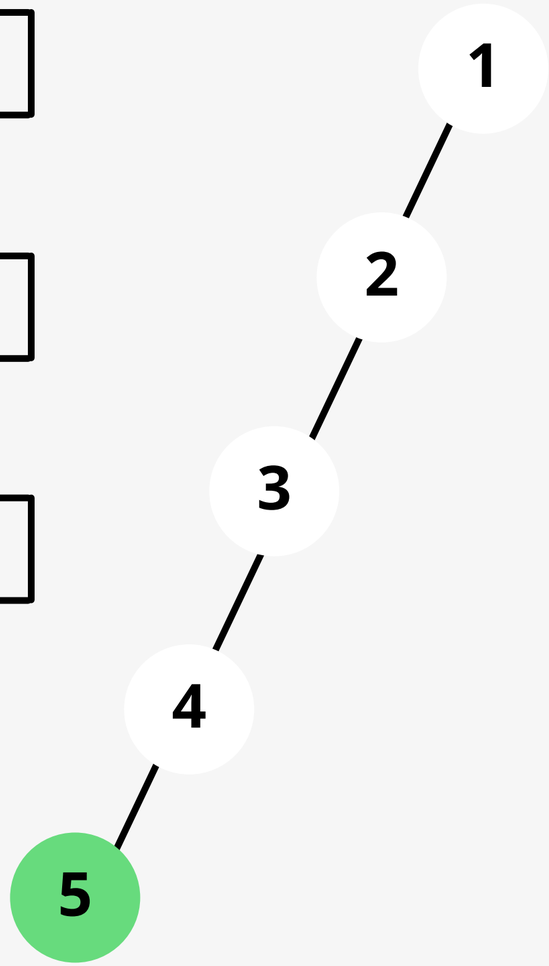
1	**2**	**3**	**4**	**0**
1	2	3	4	5

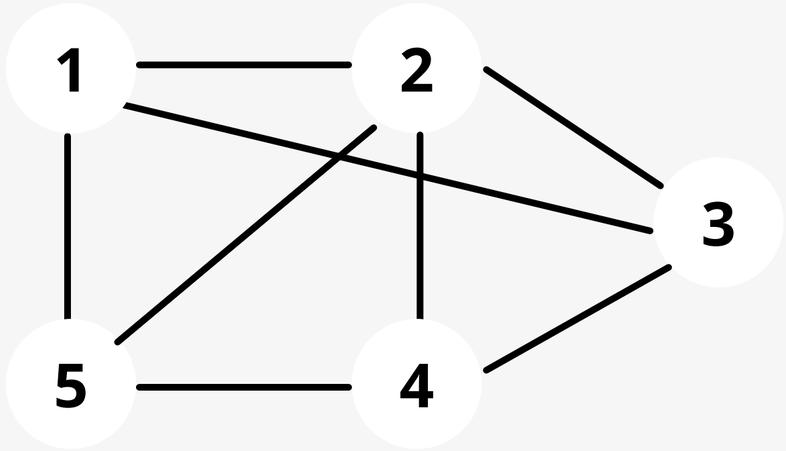
X

1	**2**	**3**	**5**	**0**
1	2	3	4	5

X

1	**2**	**3**	**0**	**0**
1	2	3	4	5



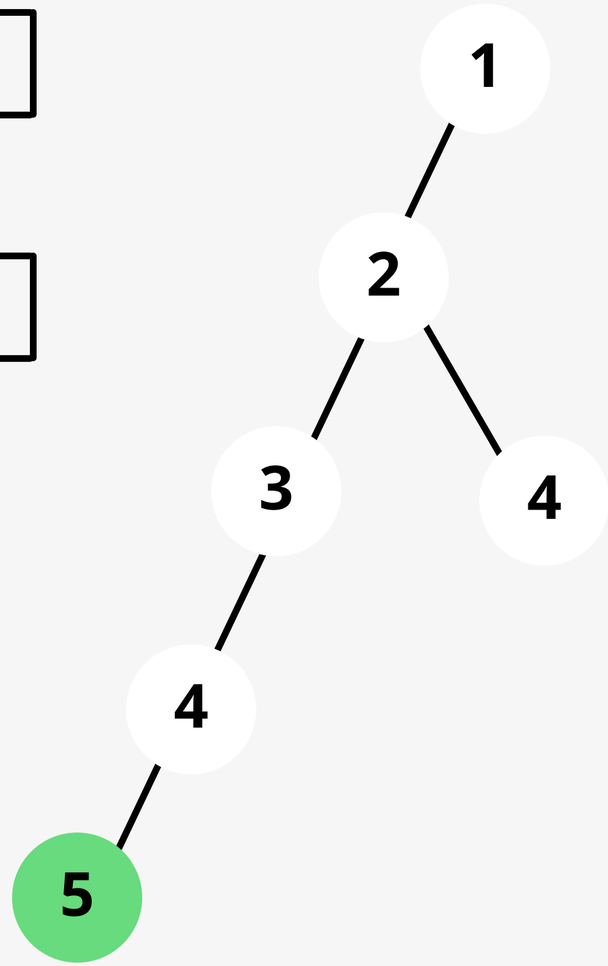


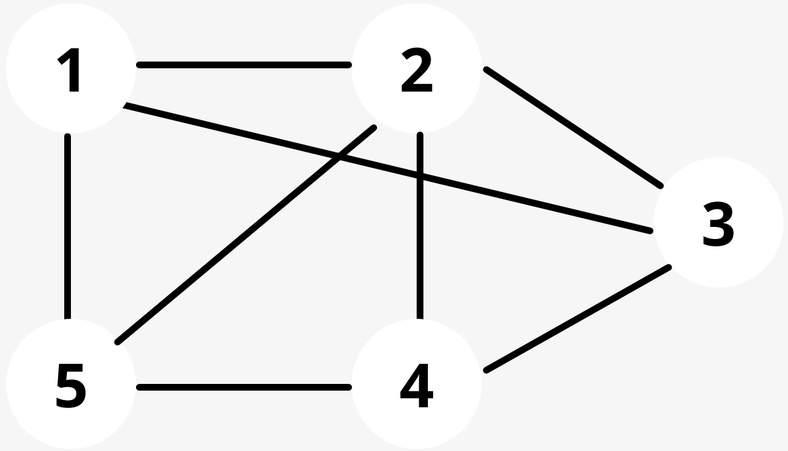
X	1	2	3	0	0
	1	2	3	4	5

X	1	2	4	0	0
	1	2	3	4	5

G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**





G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

X

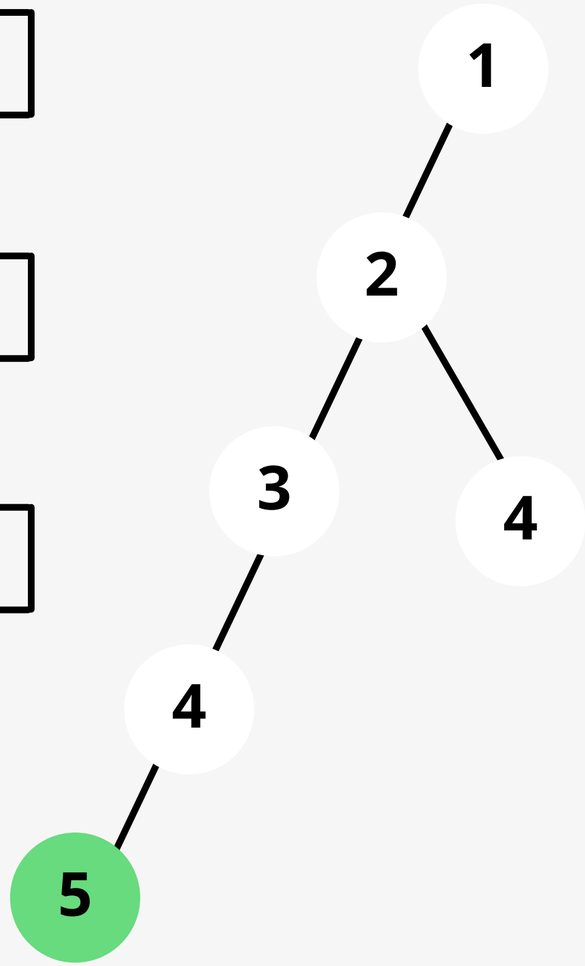
1	**2**	**3**	**0**	**0**
1	2	3	4	5

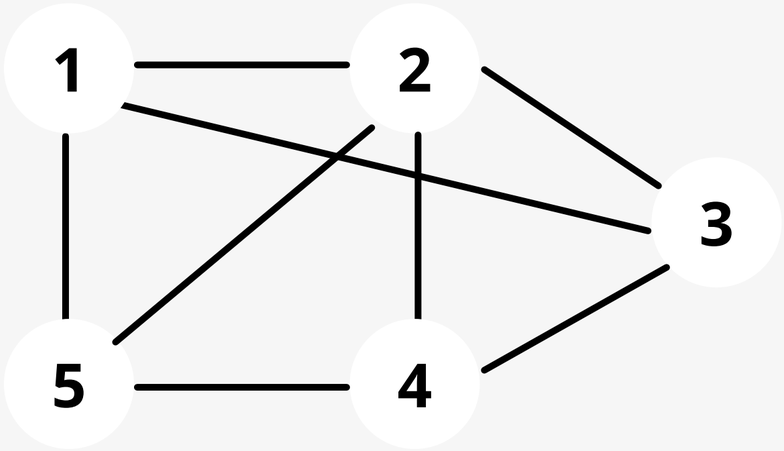
X

1	**2**	**4**	**0**	**0**
1	2	3	4	5

X

1	**2**	**4**	**3**	**0**
1	2	3	4	5





G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

X

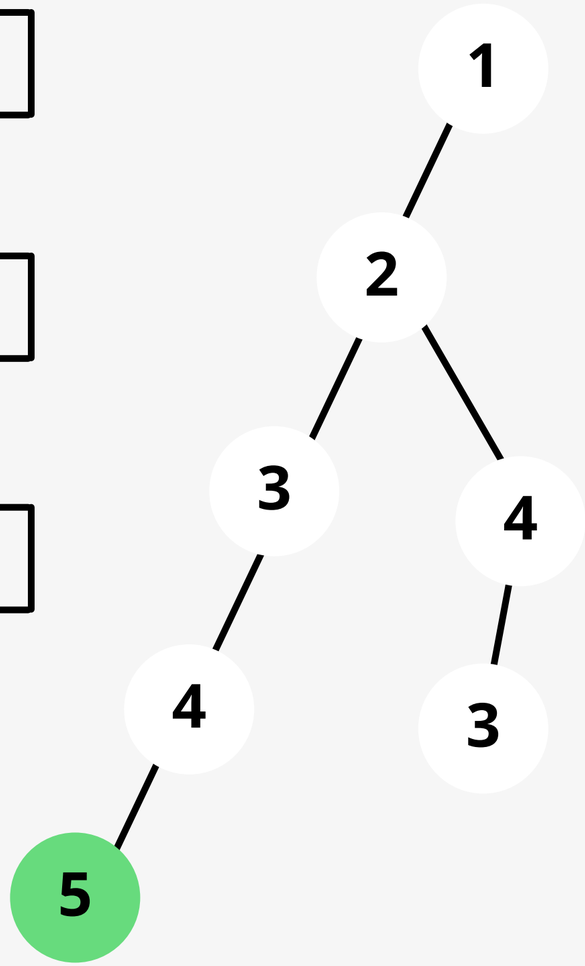
1	**2**	**3**	**0**	**0**
1	2	3	4	5

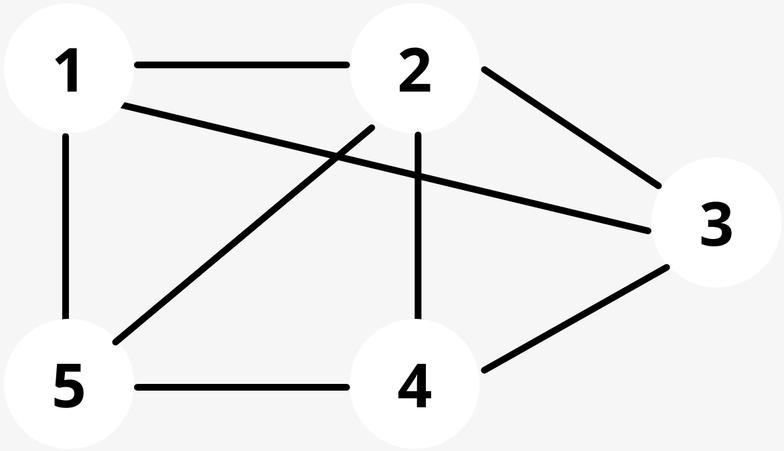
X

1	**2**	**4**	**0**	**0**
1	2	3	4	5

X

1	**2**	**4**	**3**	**0**
1	2	3	4	5



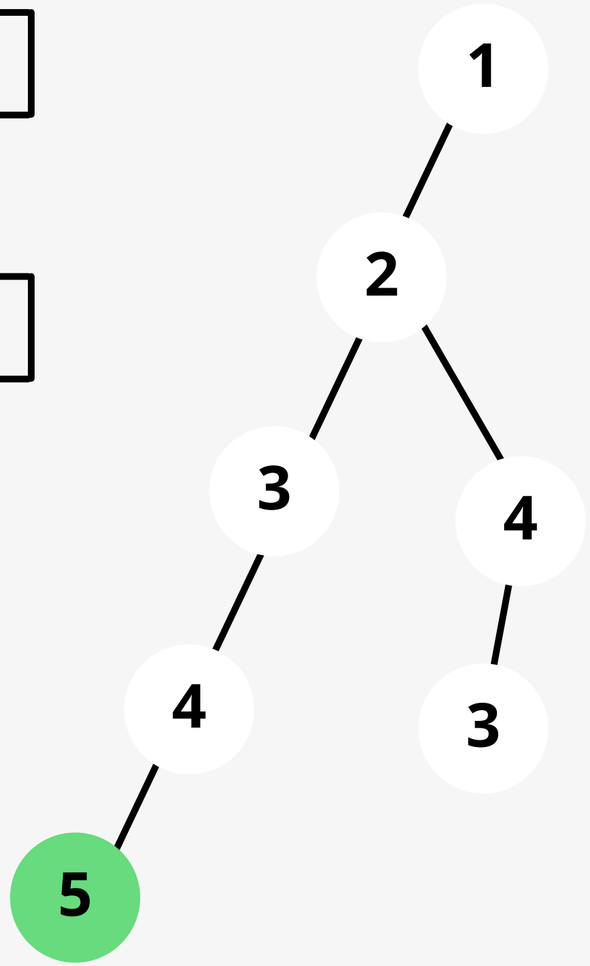


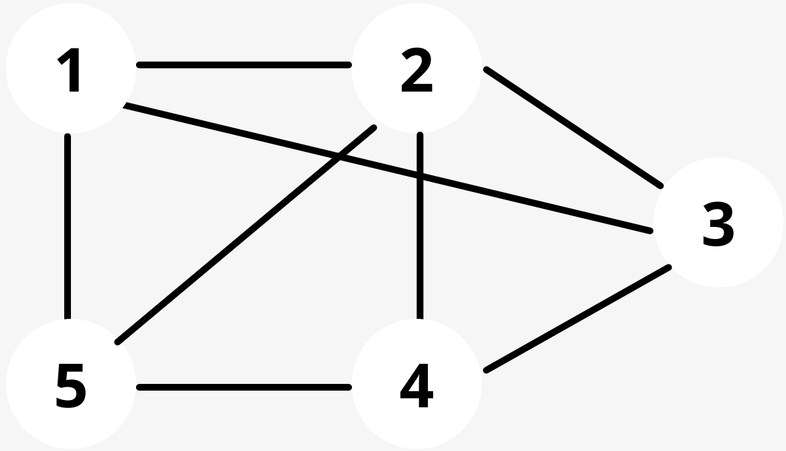
X	1	2	4	3	5
	1	2	3	4	5

X	1	2	4	5	0
	1	2	3	4	5

G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**





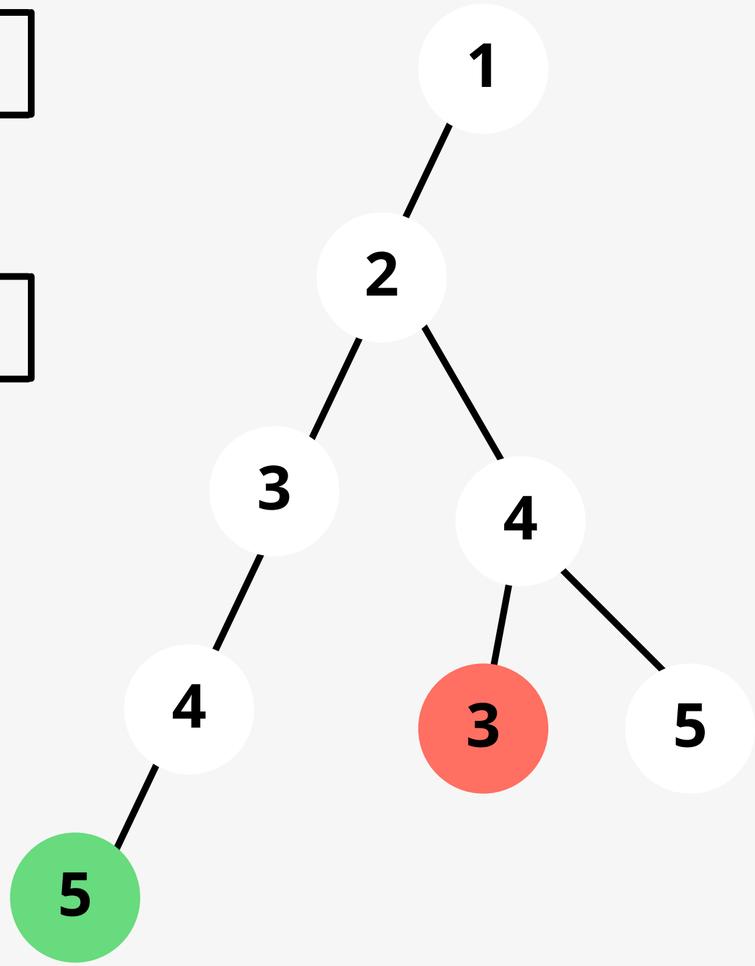
X	1	2	4	3	5
	1	2	3	4	5

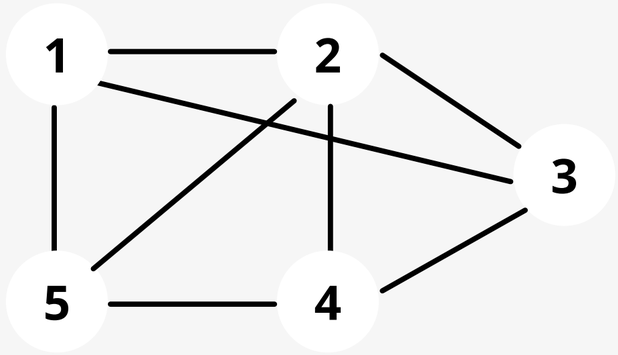
X	1	2	4	3	0
	1	2	3	4	5

X	1	2	4	5	0
	1	2	3	4	5

G=

	1	2	3	4	5
1	**0**	**1**	**1**	**0**	**1**
2	**1**	**0**	**1**	**1**	**1**
3	**1**	**1**	**0**	**1**	**0**
4	**0**	**1**	**1**	**0**	**1**
5	**1**	**1**	**0**	**1**	**0**

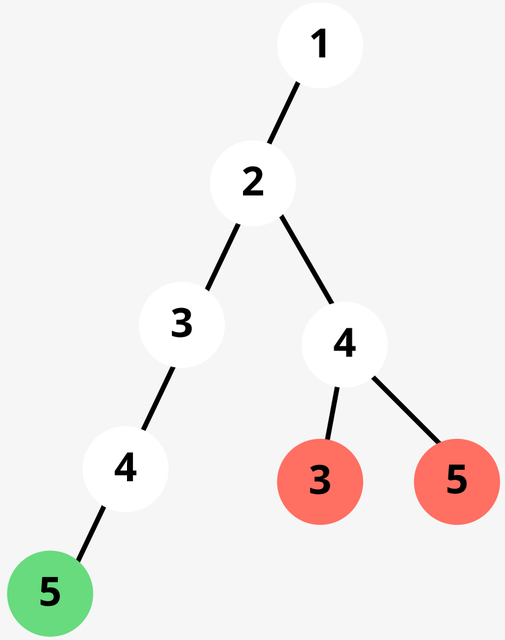




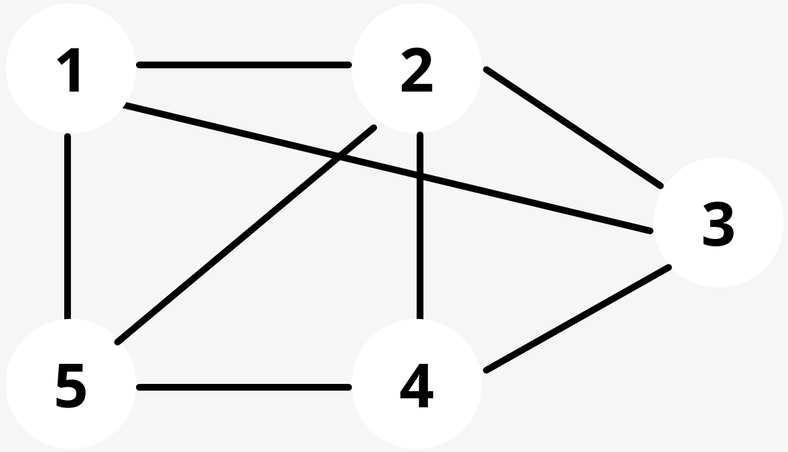
X	1	2	4	5	3
	1	2	3	4	5

X	1	2	4	0	0
	1	2	3	4	5

X	1	2	5	0	0
	1	2	3	4	5




```
-> Hamiltonian (k)                    //For backtracking
      Repeat until false:
        Next value (k)
        if x[k] == 0: return
        if x[k] == n: print x[1:n]
        else: Hamiltonian (k+1)

-> Next value (k)                     //For finding next node
      Repeat until false:
        x[k] = (x[k] + 1mod (n+1))
        if x[k] == 0: return
        if G(x[k-1]), x[k] =! 0: then
          for j=1 to k-1:
            if x[j] == x[k]: break
            if j == k: true
            if (k<n or k = n) and G(x[ n ], x[1] != 0):
              return
```

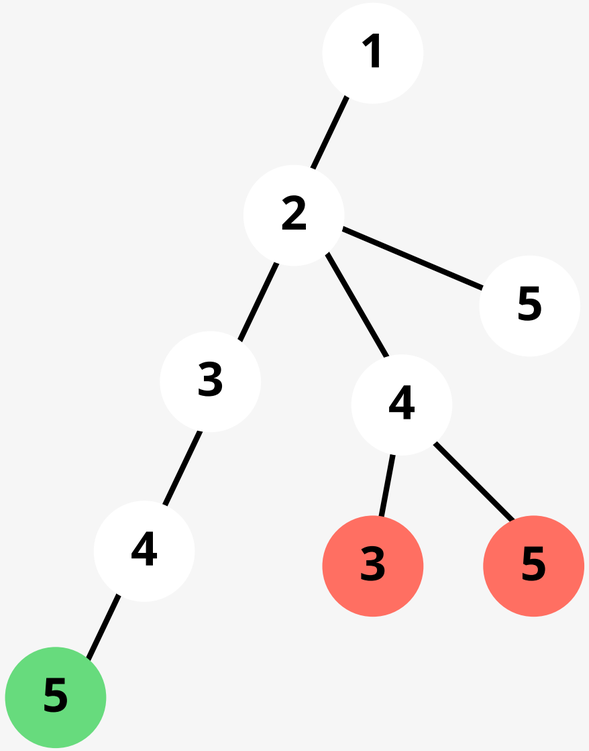


X	1	2	4	5	0
	1	2	3	4	5

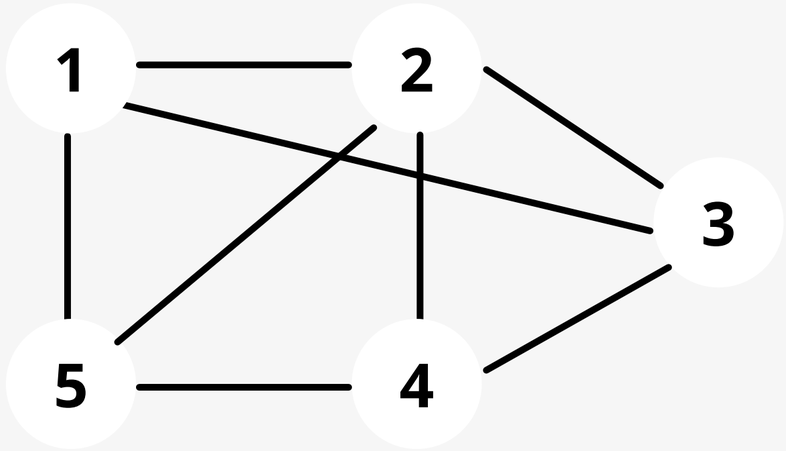
X	1	2	4	5	3
	1	2	3	4	5

X	1	2	4	0	0
	1	2	3	4	5

X	1	2	5	0	0
	1	2	3	4	5



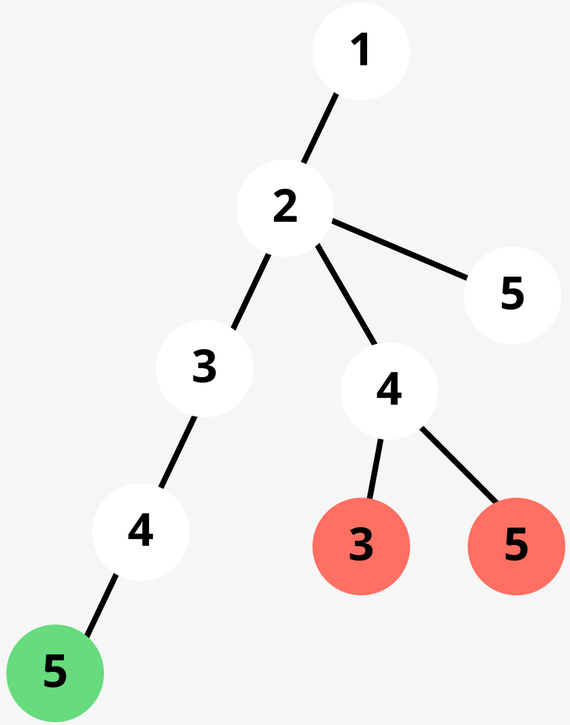

```
-> Hamiltonian (k)                    //For backtracking
     Repeat until false:
       Next value (k)
       if x[k] == 0: return
       if x[k] == n:print x[1:n]
       else: Hamiltonian (k+1)

-> Next value (k)                     //For finding next node
     Repeat until false:
       x[k] = (x[k] + 1mod (n+1))
       if x[k] == 0: return
       if G(x[k-1]), x[k] =! 0: then
          for j=1 to k-1:
             if x[j] == x[k]: break
             if j == k: true
             if (k<n or k = n) and G(x[ n ], x[1] != 0):
                return
```



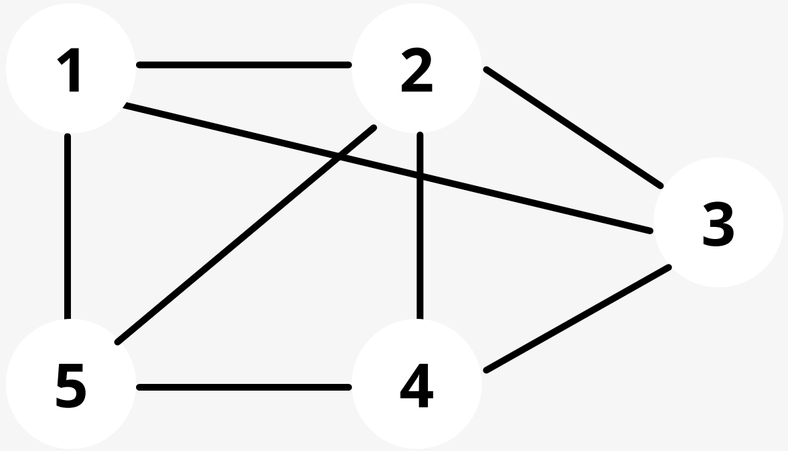
X	1	2	5	0	0
	1	2	3	4	5

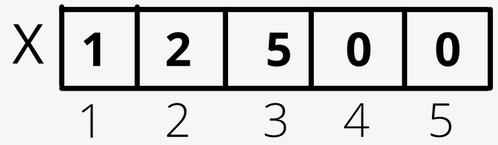
X	1	2	5	3	0
	1	2	3	4	5

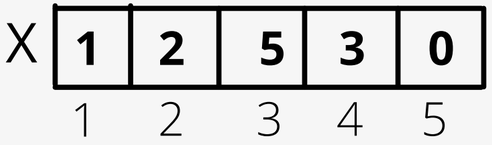
X	1	2	5	4	0
	1	2	3	4	5

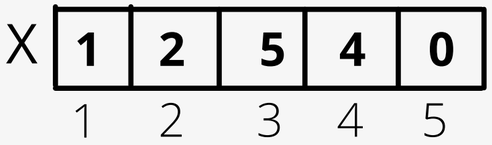


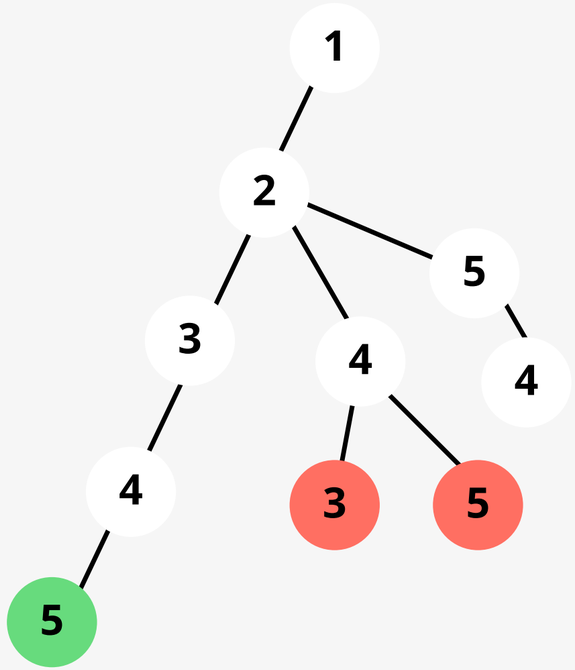

```
-> Hamiltonian (k)                    //For backtracking
      Repeat until false:
        Next value (k)
        if x[k] == 0: return
        if x[k] == n:print x[1:n]
        else: Hamiltonian (k+1)

-> Next value (k)                     //For finding next node
      Repeat until false:
        x[k] = (x[k] + 1mod (n+1))
        if x[k] == 0: return
        if G(x[k-1]), x[k] =! 0: then
          for j=1 to k-1:
            if x[j] == x[k]: break
            if j == k: true
            if (k<n or k = n) and G(x[ n ], x[1] != 0):
              return
```



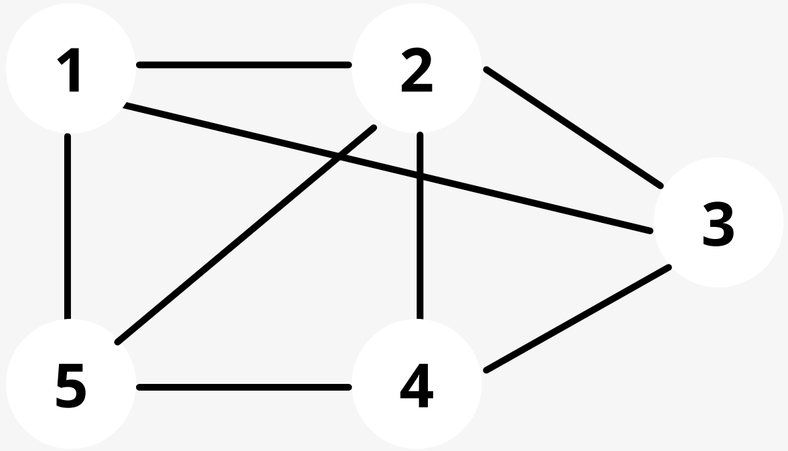




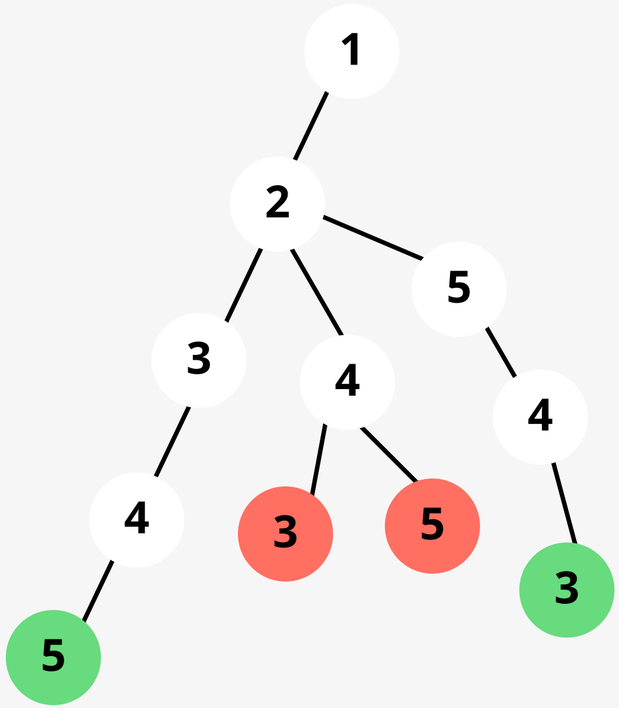





```
-> Hamiltonian (k)                    //For backtracking
      Repeat until false:
        Next value (k)
        if x[k] == 0: return
        if x[k] == n:print x[1:n]
        else: Hamiltonian (k+1)

-> Next value (k)                     //For finding next node
      Repeat until false:
        x[k] = (x[k] + 1mod (n+1))
        if x[k] == 0: return
        if G(x[k-1]), x[k] =! 0: then
          for j=1 to k-1:
            if x[j] == x[k]: break
            if j == k: true
            if (k<n or k = n) and G(x[ n ], x[1] != 0):
              return
```



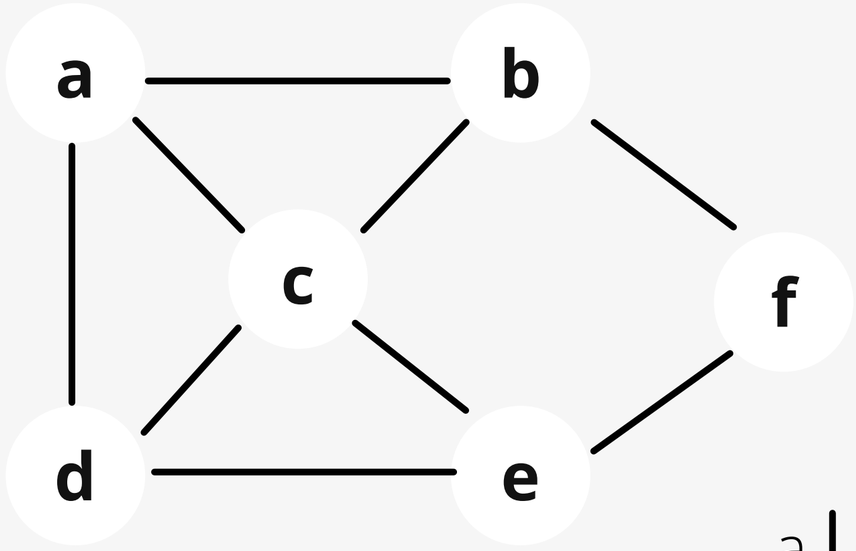
X	1	2	5	4	3
	1	2	3	4	5




```
-> Hamiltonian (k)                    //For backtracking
      Repeat until false:
        Next value (k)
        if x[k] == 0: return
        if x[k] == n:print x[1:n]
        else: Hamiltonian (k+1)

-> Next value (k)                     //For finding next node
      Repeat until false:
        x[k] = (x[k] + 1mod (n+1))
        if x[k] == 0: return
        if G(x[k-1]), x[k] =! 0:
          for j=1 to k-1:
            if x[j] == x[k]: break
            if j == k: true
            if (k<n or k = n) and G(x[ n ], x[1] != 0):
              return
```

Solve Along:
Hamiltonian Cycle

A Solve Along

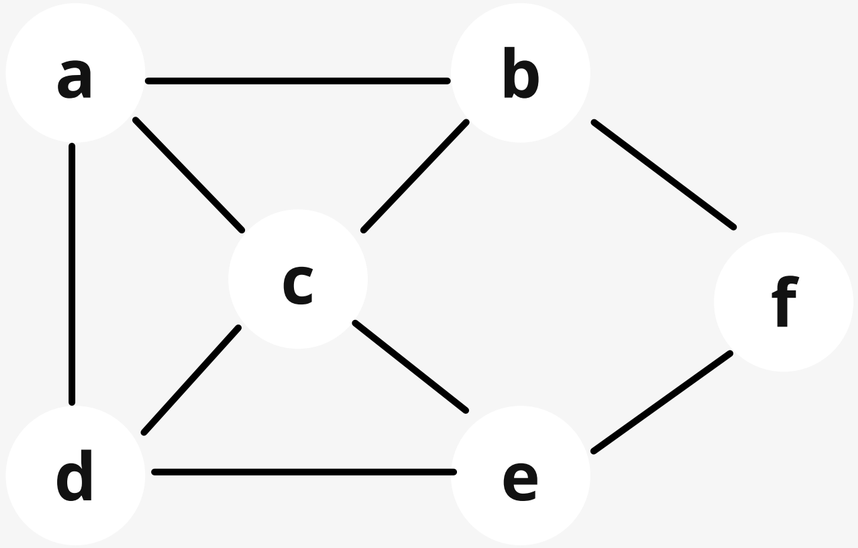


- Find Hamiltonian Cycle using Backtracking
- Start from node a

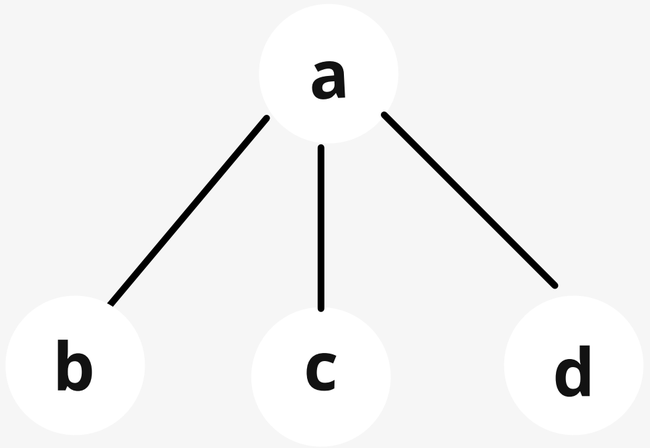
G=

	a	b	c	d	e	f
a	**0**	**1**	**1**	**1**	**0**	**0**
b	**1**	**0**	**1**	**0**	**0**	**1**
c	**1**	**1**	**0**	**1**	**1**	**0**
d	**1**	**0**	**1**	**0**	**1**	**0**
e	**0**	**0**	**1**	**1**	**0**	**1**
f	**0**	**1**	**0**	**0**	**1**	**0**

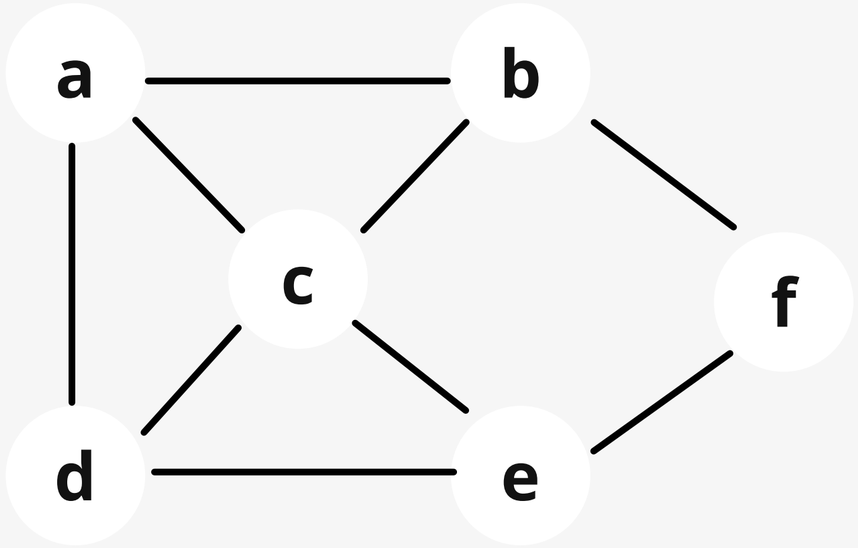
A Solve Along



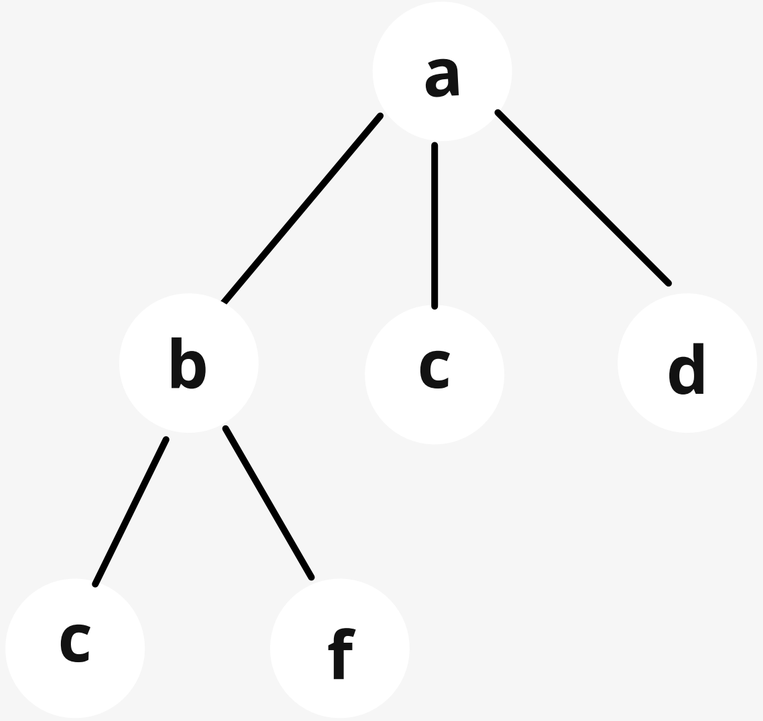
- Find Hamiltonian Cycle using Backtracking
- Start from node a



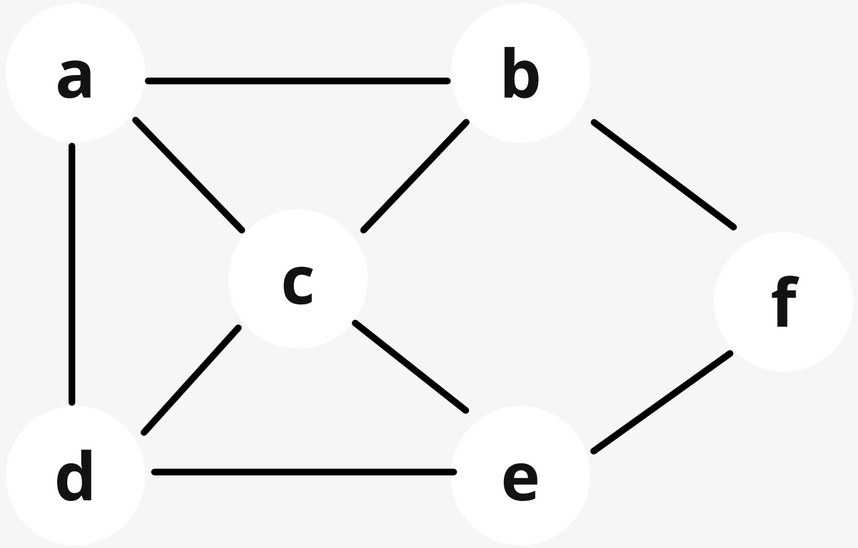
A Solve Along

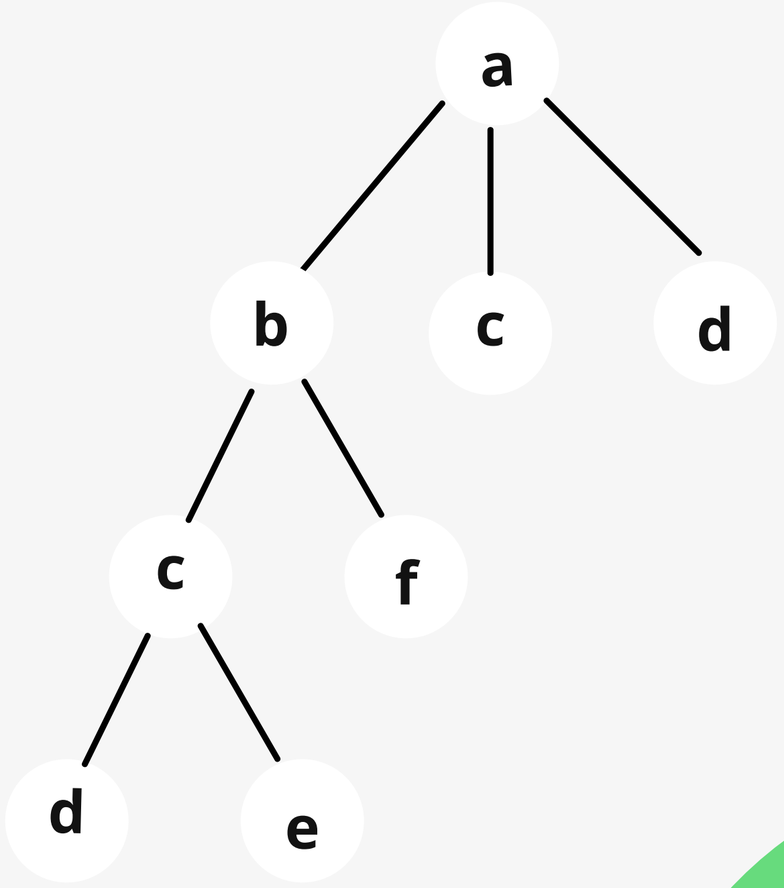


- Find Hamiltonian Cycle using Backtracking
- Node a->b->c



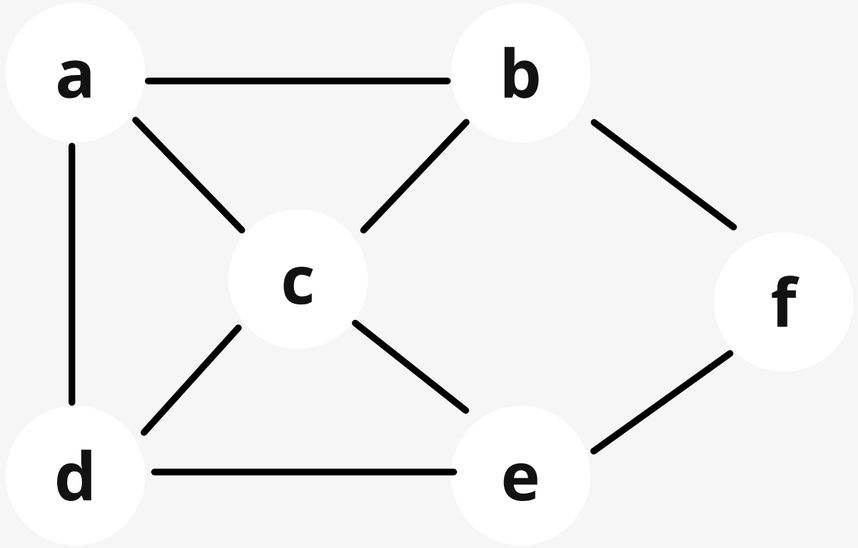
A Solve Along



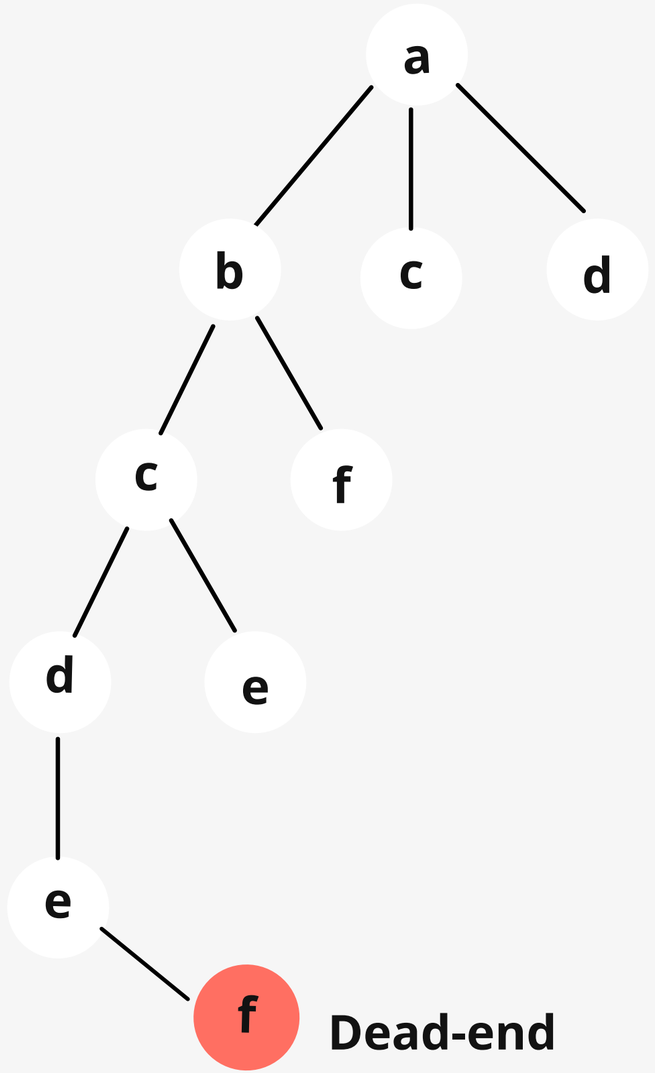


- Find Hamiltonian Cycle using Backtracking
- Node a->b->c->d

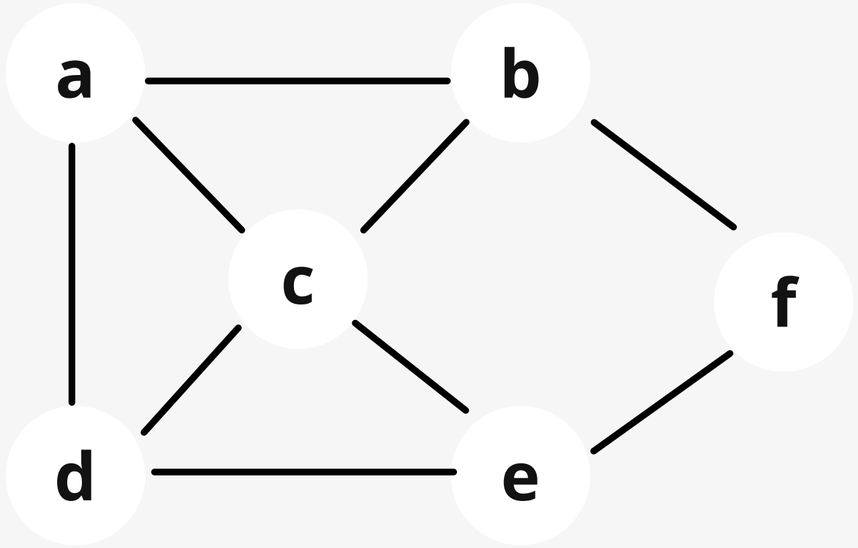
A Solve Along



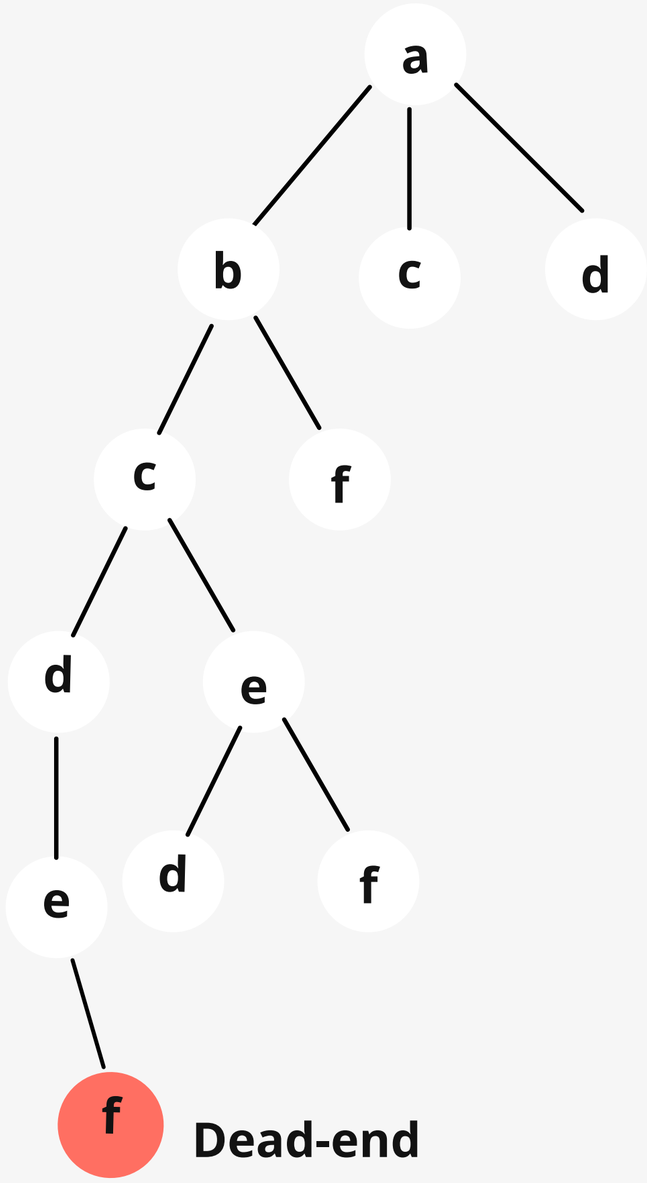
- Find Hamiltonian Cycle using Backtracking
- Node a->b->c->d->e->f



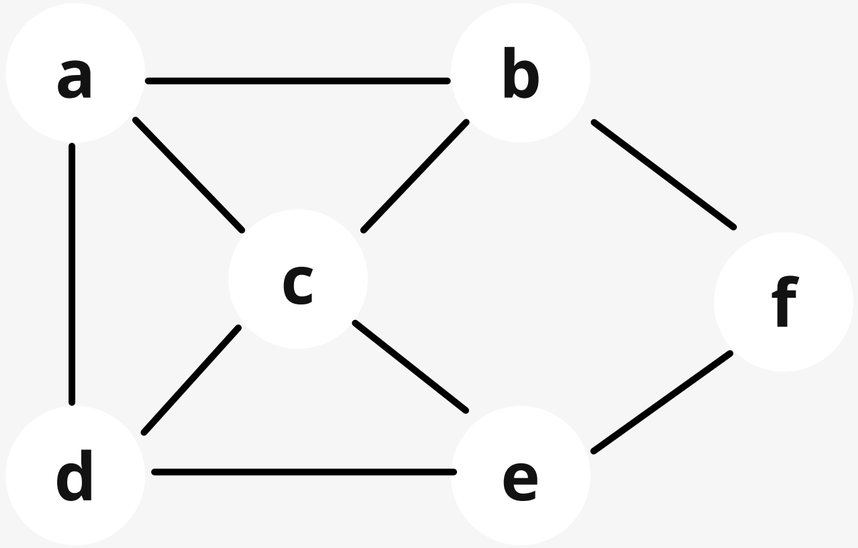
A Solve Along



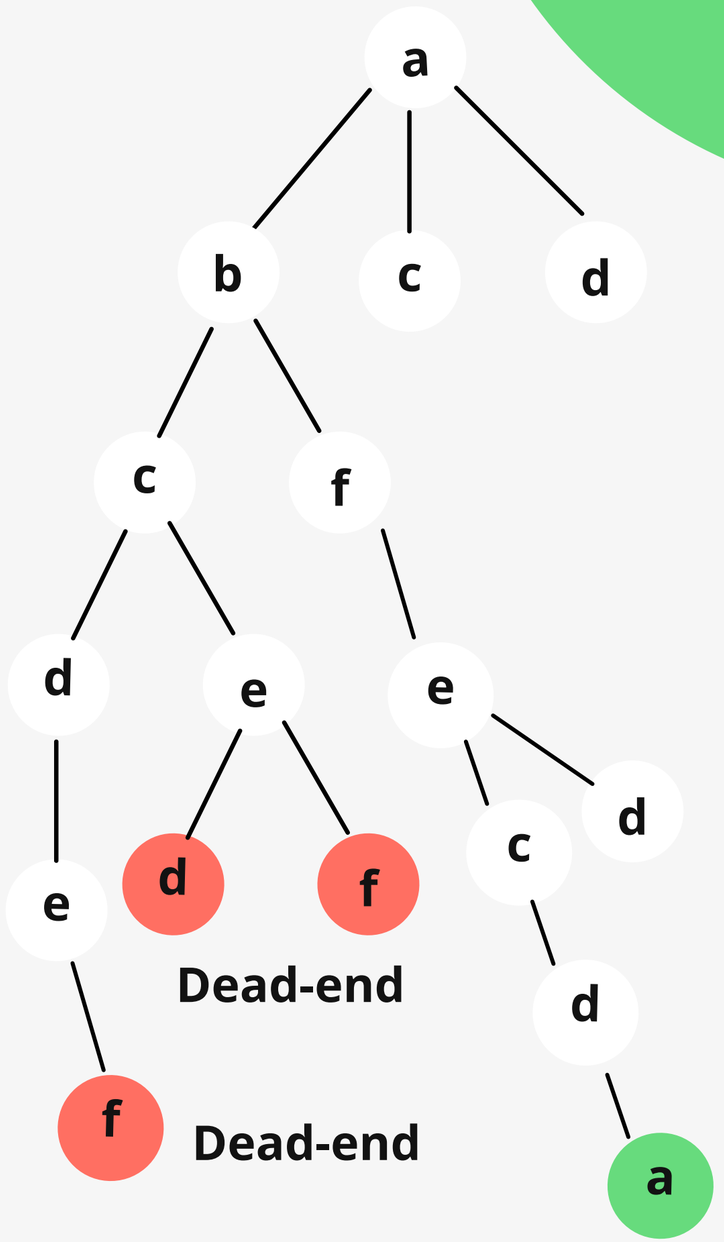
- Find Hamiltonian Cycle using Backtracking
- Node a->b->c-(backtrack)->e->d->deadend
- Node a->b->c-(backtrack)->e->f->deadend

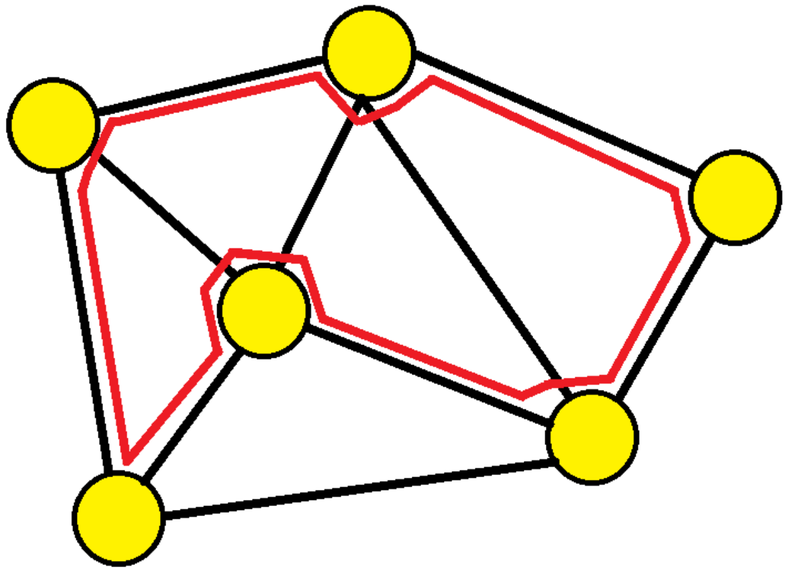


A Solve Along



- Find Hamiltonian Cycle using Backtracking
- Node a->b->f->e->c->d->a Solved





Thank You