Isomorphism & Hamiltonian Graphs

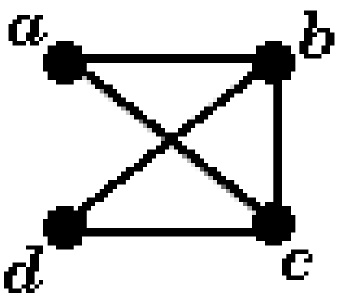
Isomorphism

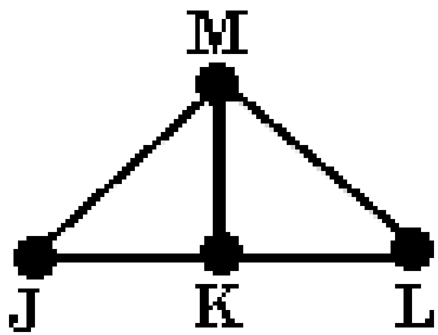
- Two graphs are *isomorphic* when the vertices of one can be re-labeled to match the vertices of the other in a way that preserves adjacency.
- Formally,
  Graphs G_1 and G_2 are *isomorphic* if there exists a one-to-one function, called an *isomorphism*,

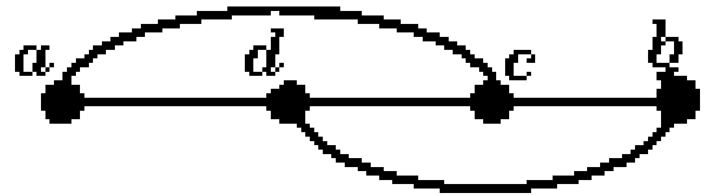
$$f\colon V(G_1) \to V(G_2)$$

  such that uv is an element of $E(G_1)$ if and only if $f(u)f(v)$ is an element of $E(G_2)$.

Example







G_1 G_2 G_3

- G_1 and G_2

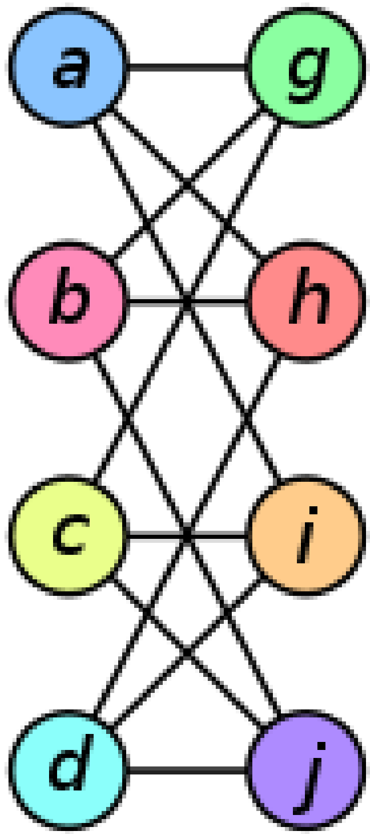
  $f(a) = J$, $f(b) = K$, $f(c) = M$, and $f(d) = L$.

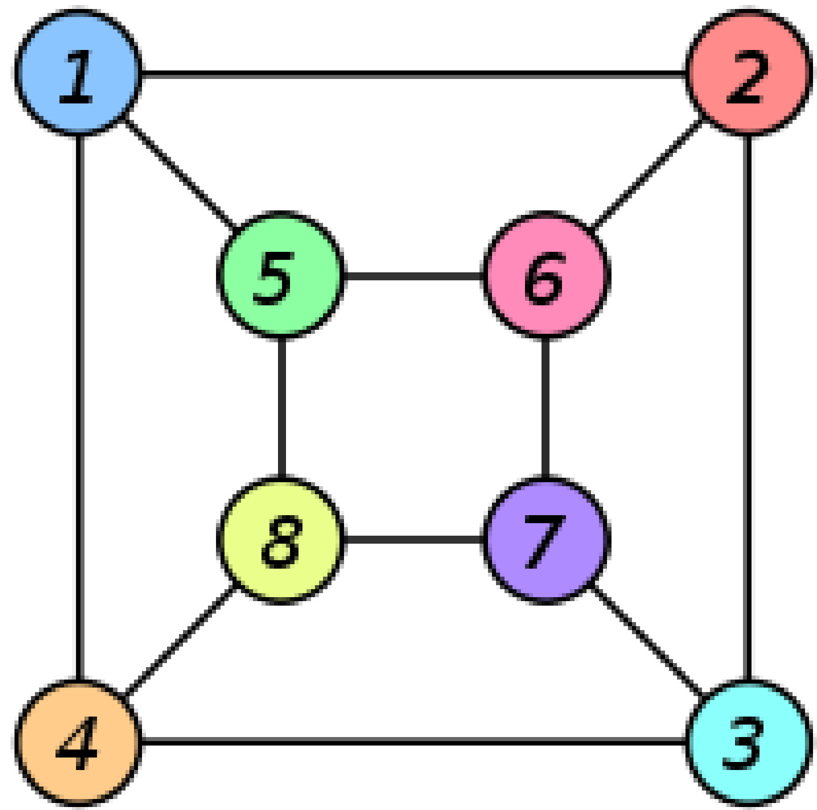
- G_1 and G_3

  $f(a) = a$, $f(b) = d$, $f(c) = c$, and $f(d) = b$.

Example

- $f(a) = 1$
- $f(b) = 6$
- $f(c) = 8$
- $f(d) = 3$
- $f(g) = 5$
- $f(h) = 2$
- $f(i) = 4$
- $f(j) = 7$





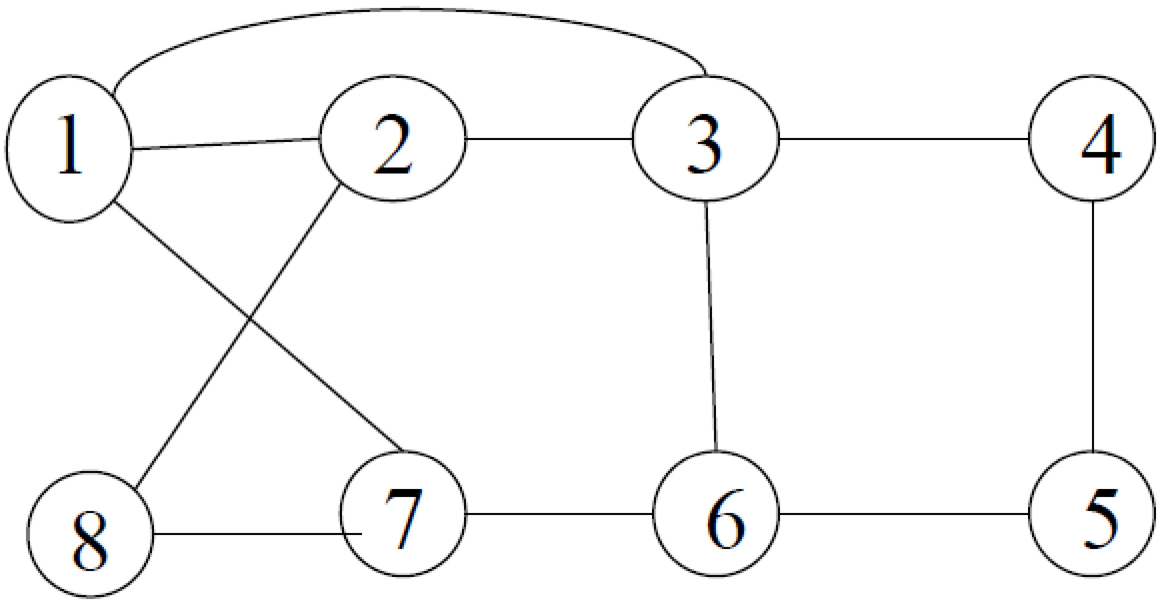
Graph Isomorphism Problem

- Determining whether two finite graphs are isomorphic or not?
- Applications:
  - Cheminformatics,
  - Mathematical chemistry (identification of chemical compounds), and
  - Electronic design automation (verification of equivalence of various representations of the design of an electronic circuit).
  - And Similar other problems

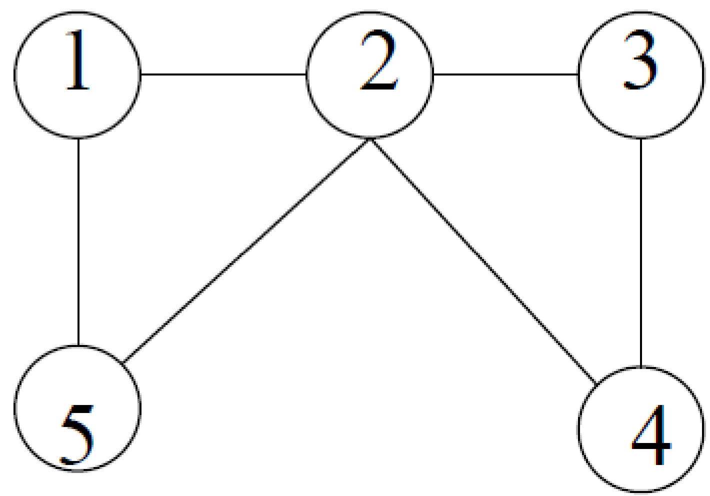
Hamiltonian Graphs

- A path passing through all the vertices of a graph is called a **Hamiltonian path**.
  - A graph containing a Hamiltonian path is said to be traceable.
- A cycle passing through all the vertices of a graph is called a **Hamiltonian cycle** (or **Hamiltonian circuit**).
- A graph containing a Hamiltonian cycle is called a **Hamiltonian graph**.
- **Hamiltonian path problem**: Determine the existence of Hamiltonian paths and cycles in graphs (**NP-complete**).

Example



- Hamiltonian graph
  - 128765431



Not a Hamiltonian graph.

Solution 1 – Brute Force Search Algorithm

- A Hamiltonian Path in a graph having N vertices is nothing but a permutation of the vertices of the graph

$$[v_1, v_2, v_3, \ldots\ldots v_{N-1}, v_N]$$

  such that there is an edge between v_i and v_{i+1} where $1 \leq i \leq N-1$.

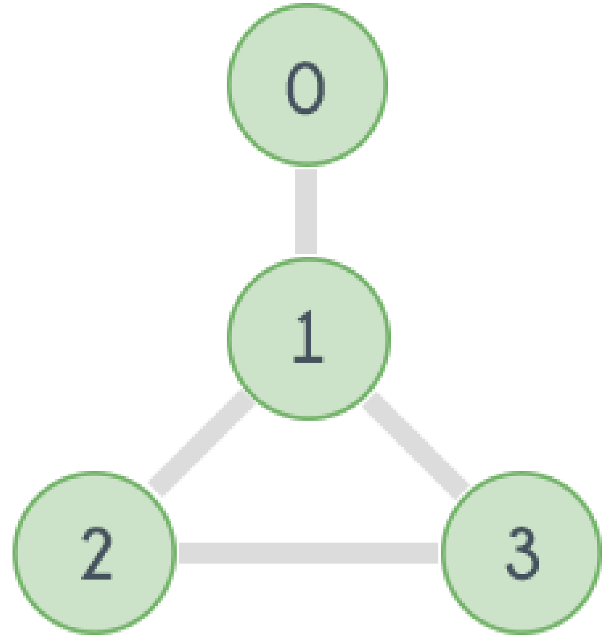
- So it can be checked for all permutations of the vertices whether any of them represents a Hamiltonian Path or not.

Contd…

```
function check_all_permutations(adj[][], n)
    for i = 0 to n
        p[i]=i
    while next permutation is possible
        valid = true
        for i = 0 to n-1
            if adj[p[i]][p[i+1]] == false
                valid = false
                break
        if valid == true
            print_permutation(p)
            return true
        p = get_next_permutation(p)
    return false
```

Example

Not a Hamiltonian graph as path exists and not a circuit.



1. **0-1-2-3**
2. **0-1-3-2**
3. 0-2-1-3
4. 0-2-3-1
5. 0-3-1-2
6. 0-3-2-1
7. 1-0-2-3
8. 1-0-3-2
9. 1-2-0-3
10. 1-2-3-0
11. 1-3-0-2
12. 1-3-2-0
13. 2-0-1-3
14. 2-0-3-1
15. 2-1-0-3
16. 2-1-3-0
17. **2-3-1-0**
18. 2-3-0-1
19. 3-0-1-2
20. 3-0-2-1
21. 3-1-0-2
22. 3-1-2-0
23. 3-2-0-1
24. **3-2-1-0**

Solution 2 – Backtracking

1. Create an empty path array and add vertex 0 to it.
2. Add other vertices, starting from the vertex 1.
3. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added.
4. If such a vertex is found, add that vertex as part of the solution.
5. Otherwise, return false.

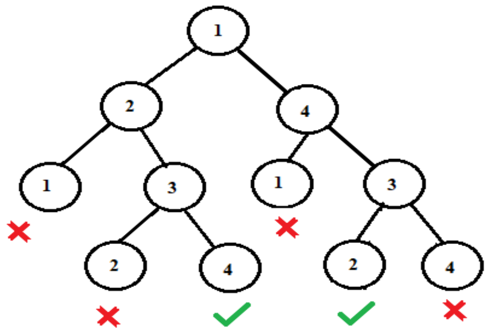
Contd…

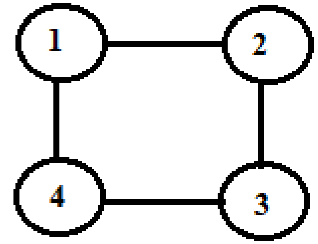
```
Algorithm Hamiltonian(k)
// This algorithm uses the recursive formulation of
// backtracking to find all the Hamiltonian cycles
// of a graph. The graph is stored as an adjacency
// matrix G[1 : n, 1 : n]. All cycles begin at node 1.
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k].
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

- Let,
  - Array x is a solution vector. $x[i]$ represents the i^{th} visited vertex of the proposed cycle.
  - $G[1{:}n,1{:}n]$ is the adjacency matrix of the given graph.
- Initialize $x[1]$ to 1 (as vertex 1 is the starting vertex) and $x[2{:}n]$ to zero, then call Hamiltonian(2).

```
Algorithm NextValue(k)
// x[1 : k − 1] is a path of k − 1 distinct vertices. If x[k] = 0, then
// no vertex has as yet been assigned to x[k]. After execution,
// x[k] is assigned to the next highest numbered vertex which
// does not already appear in x[1 : k − 1] and is connected by
// an edge to x[k − 1]. Otherwise x[k] = 0. If k = n, then
// in addition x[k] is connected to x[1].
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k − 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k − 1 do if (x[j] = x[k]) then break;
                                  // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

Example - 1





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

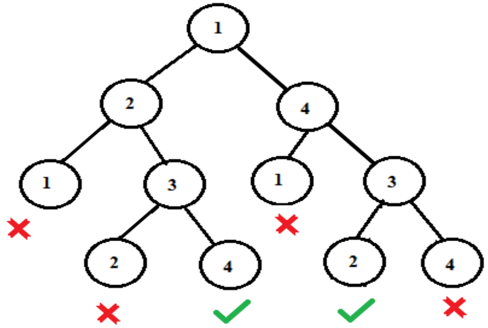
```
Algorithm NextValue(k)
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k - 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k - 1 do if (x[j] = x[k]) then break;
                        // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

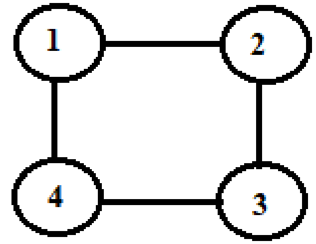
```
Algorithm Hamiltonian(k)
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
1 0 0 0 1 1 0 0	2	X[2] = X[2]+1= (0+1)% 5 => **X[2] = 1**	G [X[1], X[2]] => G[1,1] ! = 0 **[False]**	**SKIP**	**SKIP**
1 2 0 0		X[2] = X[2]+1= (1+1)%5 => **X[2] = 2**	G [X[1], X[2]] => G[1,2] ! = 0 **[True]**	j = 1 to 1; j++	X[1] = X[2] => 1 == 2 **[False]**

Now, j == k i.e., 2 ==2 (distinct) and 2<4 , so NextValue(2) returns. In Hamiltonian(2) => X[2] !=0 and 2 != 4, so Hamiltonian(k+1) i.e., Hamiltonain(3) is recursively called.

Contd..





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

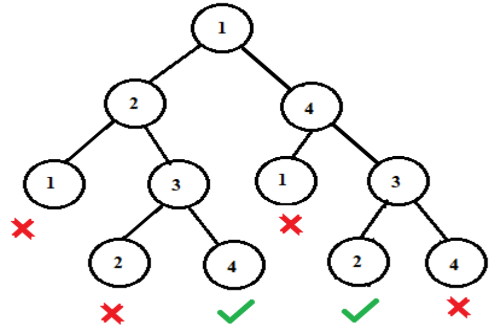
```
Algorithm NextValue(k)
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k − 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k − 1 do if (x[j] = x[k]) then break;
                        // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

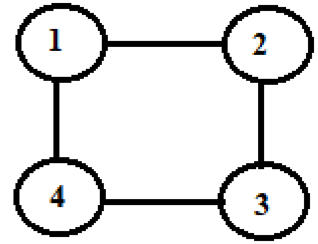
```
Algorithm Hamiltonian(k)
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
1 2 0 0 1 2 1 0	3	X[3] = X[3]+1= (0+1)% 5 => **X[3] = 1**	G [X[2], X[3]] => G[2,1] ! = 0 **[True]**	j = 1	X[1] = X[3] **[T]** **Break**
1 2 2 0		X[3] = X[3]+1= (1+1)%5 => **X[3] = 2**	G [X[2], X[3]] => G[2,2] ! = 0 **[False]**	**SKIP**	**SKIP**
1 2 3 0		X[3] = X[3]+1= (2+1)%5 => **X[3] = 3**	G [X[2], X[3]] => G[2,3] ! = 0 **[True]**	j = 1 j = 2	X[1] = X[3] **[F]** X[2] = X[3] **[F]**

Now, j == k i.e., 3 ==3 (distinct) and 3<4 , so NextValue(3) returns. In Hamiltonian(3) => X[3] !=0 and 3 != 4, so Hamiltonian(k+1) i.e., Hamiltonain(4) is recursively called.

Contd..





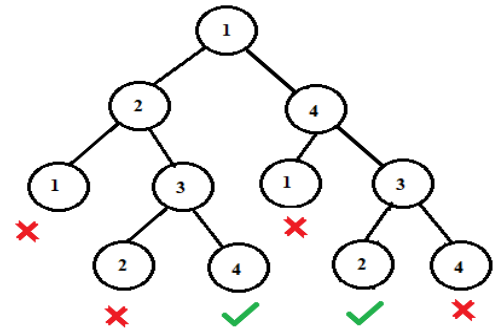
	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

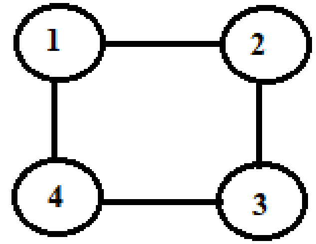
```
Algorithm NextValue(k)
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k − 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k − 1 do if (x[j] = x[k]) then break;
                          // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then return;
        }
    } until (false);
}
```

```
Algorithm Hamiltonian(k)
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
1 2 3 0 1 2 3 1	4	X[4] = X[4]+1= (0+1)% 5 => **X[4] = 1**	G [X[3], X[4]] => G[3,1] ! = 0 **[False]**	**SKIP**	**SKIP**
1 2 3 2		X[4] = X[4]+1= (1+1)%5 => **X[4] = 2**	G [X[3], X[4]] => G[3,2] ! = 0 **[True]**	j = 1 j = 2	X[1] = X[4] **[F]** X[2] = X[4] **[T]** **BREAK**
1 2 3 3		X[4] = X[4]+1= (2+1)%5 => **X[4] = 3**	G [X[3], X[4]] => G[3,3] ! = 0 **[False]**	**SKIP**	**SKIP**

Contd..





$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 0 & 1 \\ 4 & 1 & 0 & 1 & 0 \end{array}$$

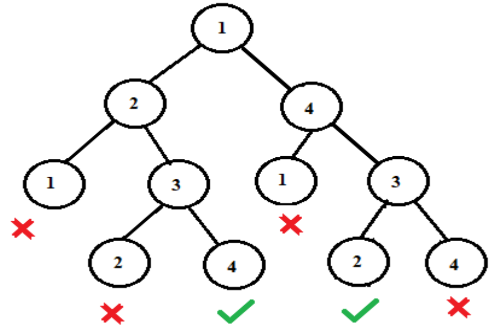
```
Algorithm NextValue(k)
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k - 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k - 1 do if (x[j] = x[k]) then break;
                    // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

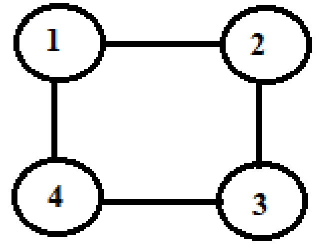
```
Algorithm Hamiltonian(k)
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
1 2 3 4	4	X[4] = X[4]+1= (3+1)%5 => **X[4] = 4**	G [X[3], X[4]] => G[3,4] ! = 0 **[True]**	j = 1 j = 2 j = 3	X[1] = X[4] **[F]** X[2] = X[4] **[F]** X[3] = X[4] **[F]**

Now, j == k i.e., 4 == 4 (distinct) and k==n (4==4) and G[X[n],X[1]]
i.e., G[4,1] != 0 (there is an edge between last and first vertex),
So, NextValue(4) returns. In Hamiltonian(4) => as X[4] !=0 and k == 4,
So, First solution is printed as: **1, 2, 3, 4**
And then cycle continues for finding out other solutions in the state space search tree.

Contd..





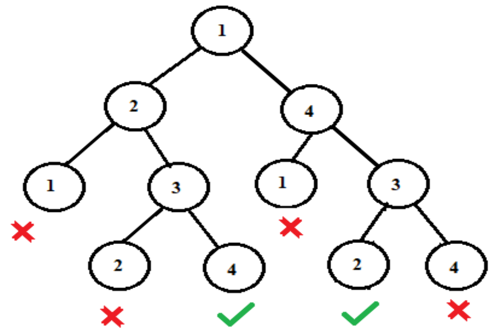
	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

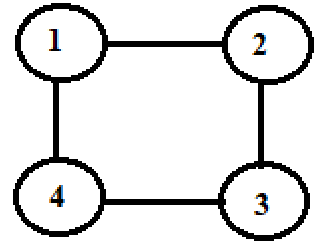
```
Algorithm NextValue(k)
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k − 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k − 1 do if (x[j] = x[k]) then break;
                        // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

```
Algorithm Hamiltonian(k)
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
1 2 3 0	4	X[4] = 5%5 = **0**	Control returns to Hamiltonian(4) and then to Hamiltonian(3)		
1 2 4 0	3	X[3]=4%5 = **4**	G[X[2],X[3]] i.e., G[2,4] != 0 **[False]**	**SKIP**	**SKIP**
1 2 0 0		X[3]=5%5 = **0**	Control returns to Hamiltonian(3) and then to Hamiltonian(2)		
1 3 0 0	2	X[2] = 3%5 = **3**	G[X[1],X[2]] i.e., G[1,3] != 0 **[False]**	**SKIP**	**SKIP**

Contd..





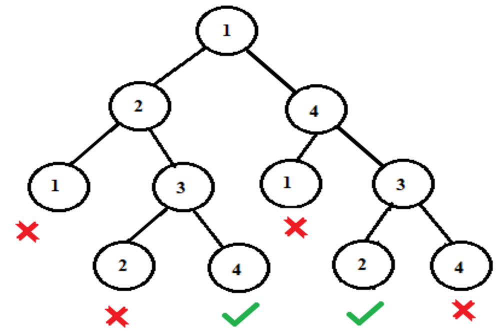
	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

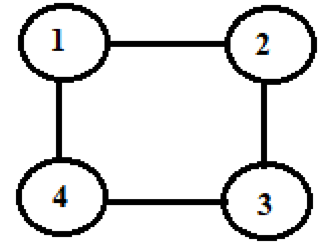
```
Algorithm NextValue(k)
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k - 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k - 1 do if (x[j] = x[k]) then break;
                        // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

```
Algorithm Hamiltonian(k)
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
1 4 0 0	2	X[2] = 4%5 = **4**	G[X[1],X[2]] i.e., G[1,4] != 0 **[True]**	j = 1	X[1] = X[2] **[F]**
Now, j == k i.e., 2 ==2 (distinct) and 2<4 , so NextValue(2) returns. In Hamiltonian(2) => X[2] !=0 and 2 != 4, so Hamiltonian(k+1) i.e., Hamiltonain(3) is recursively called.					
1 4 1 0 1 4 2 0 1 4 3 0	3	X[3] = 1%5 = **1** X[3] = 2%5 = **2** X[3] = 3%5 = **3**	G[4,1] != 0 **[True]** G[4,2] != 0 **[False]** G[4,3] != 0 **[True]**	j = 1 to 2 **SKIP** j = 1 j = 2	X[1] = X[3] **[T]** **SKIP** X[1] = X[3] **[F]** X[2] = X[3] **[F]**

Contd..





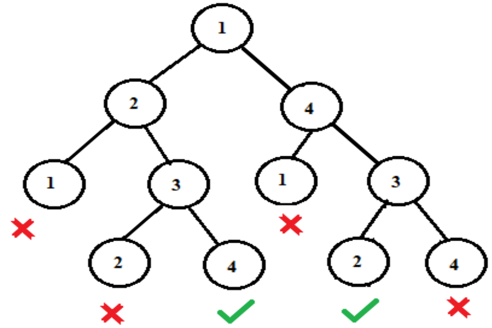
	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

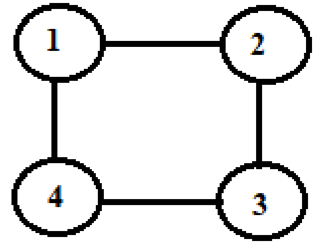
```
Algorithm NextValue(k)
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k − 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k − 1 do if (x[j] = x[k]) then break;
                            // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

```
Algorithm Hamiltonian(k)
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
Now, j == k i.e., 3 == 3 (distinct) and 3<4 , so NextValue(3) returns. In Hamiltonian(3) => X[3] !=0 and 3 != 4, so Hamiltonian(k+1) i.e., Hamiltonain(4) is recursively called.					
1 4 3 1	4	X[4] = 1%5 = **1**	G[3,1] != 0 **[False]**	**SKIP**	**SKIP**
1 4 3 2		X[4] = 2%5 = **2**	G[3,2] != 0 **[True]**	j = 1 j = 2 j = 3	X[1] = X[4] **[F]** X[2] = X[4] **[F]** X[3] = X[4] **[F]**
Now, j == k i.e., 4 == 4 (distinct) and k==n (4==4) and G[X[n],X[1]] i.e., G[2,1] != 0, So, NextValue(4) returns. In Hamiltonian(4) => as X[4] !=0 and k == 4, So, Second solution is printed as: **1, 4, 3, 2**					

Contd..





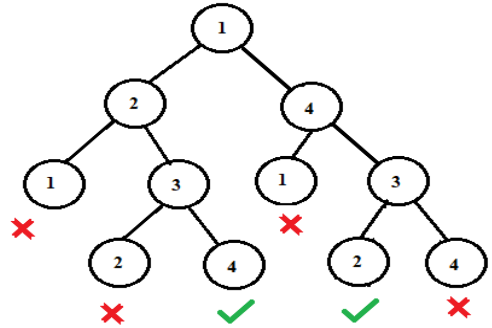
	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

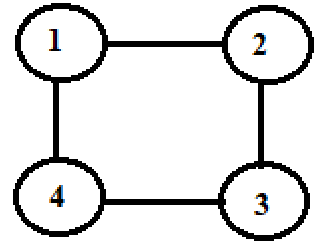
```
Algorithm NextValue(k)
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k − 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k − 1 do if (x[j] = x[k]) then break;
                        // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then return;
        }
    } until (false);
}
```

```
Algorithm Hamiltonian(k)
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
1 4 3 3	4	X[4] = 3%5 = **3**	G[3,3] != 0 **[False]**	**SKIP**	**SKIP**
1 4 3 4		X[4] = 4%5 = **4**	G[3,4] != 0 **[True]**	j = 1	X[1] = X[4] **[F]**
				j = 2	X[2] = X[4] **[T]**
					BREAK
1 4 3 0		X[4] = 5%5 = **0**			
			RETURN		
1 4 4 0	3	X[4] = 4%5 = **4**	G[4,4] != 0 **[False]**	**SKIP**	**SKIP**
1 4 0 0		X[4] = 5%5 = **0**	**RETURN**		

Contd..





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

Algorithm NextValue(k)

```
{
    repeat
    {
        x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
        if (x[k] = 0) then return;
        if (G[x[k - 1], x[k]] ≠ 0) then
        { // Is there an edge?
            for j := 1 to k - 1 do if (x[j] = x[k]) then break;
                         // Check for distinctness.
            if (j = k) then // If true, then the vertex is distinct
                if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
                    then  return;
        }
    } until (false);
}
```

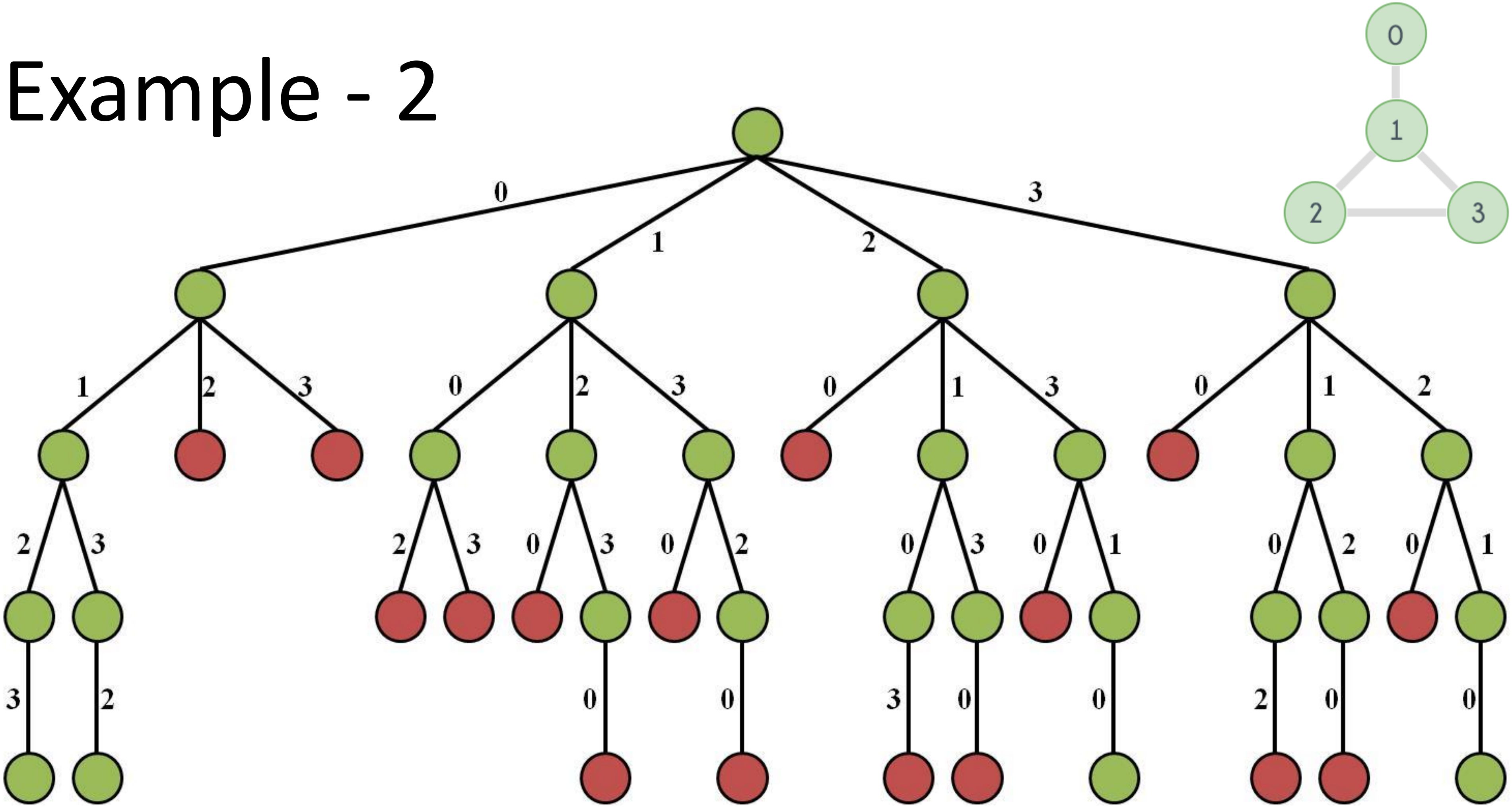
Algorithm Hamiltonian(k)

```
{
    repeat
    { // Generate values for x[k].
        NextValue(k); // Assign a legal next value to x[k]
        if (x[k] = 0) then return;
        if (k = n) then write (x[1 : n]);
        else Hamiltonian(k + 1);
    } until (false);
}
```

X	k	X[k]=X[k]+1/mod(n+1)	G[X[k-1],X[k]]!=0	j = 1 to k-1	X[j] = X[k]
1 | 4 | 0 | 0	2	X[4] = 5%5 = **0**	**RETURN**		
1 | 0 | 0 | 0					

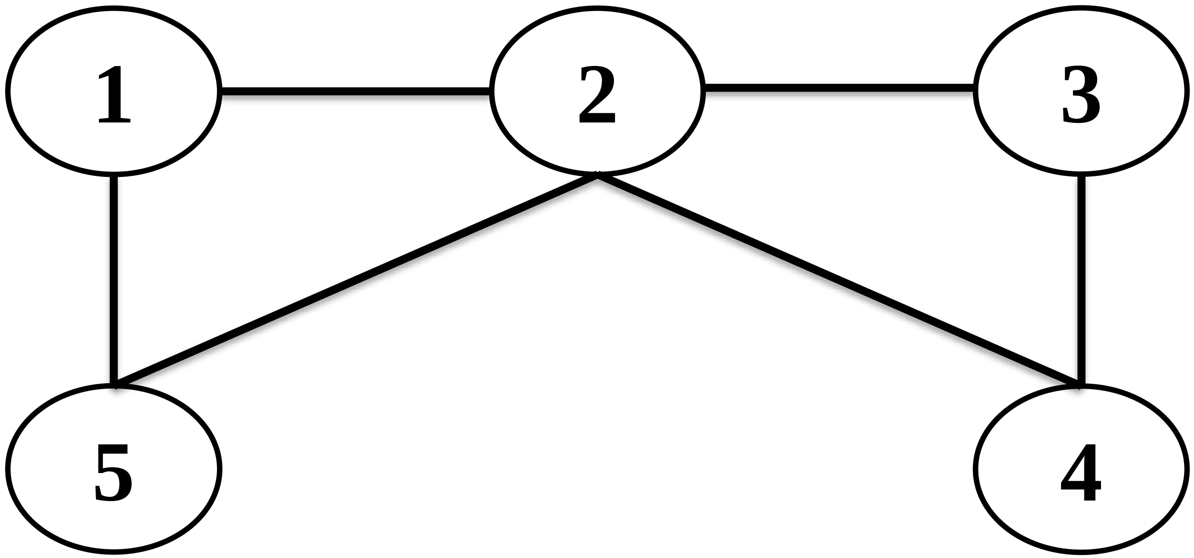
Control finally returns from Hamiltonian(2) [The original initial function call]
and
Algorithm Stops !!

Example - 2



Only four paths are there which are ending with green leaf nodes. None of them is a circuit as starting and ending vertices are not adjacent in any of the obtained path, thus this graph is non-hamiltonian.

Home Work



Applications

- Painting road lines,
- Plowing roads after a snowstorm,
- Checking meters along roads,
- Garbage pickup routes, etc.

Thank you