Design and Analysis of Algorithms

L35: Backtracking Algorithms
Hamiltonian Cycles &
m-Coloring of a Graph

Dr. Ram P Rustagi
Sem IV (2020-Even)
Dept of CSE, KSIT
rprustagi@ksit.edu.in

Resources

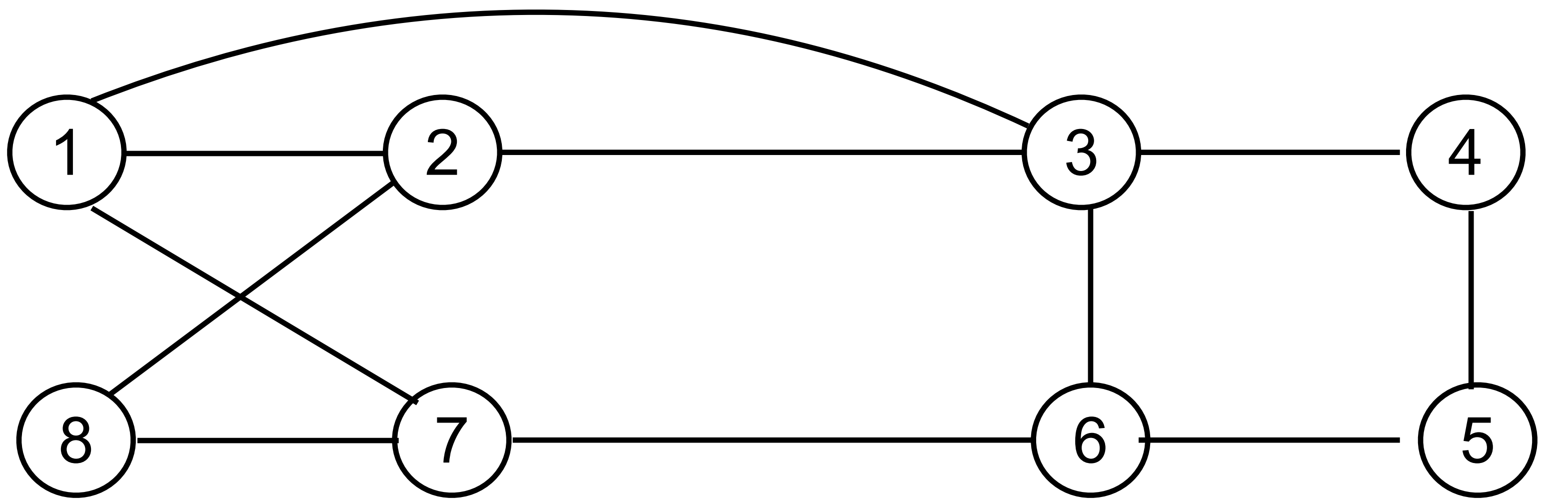
- Text book 2: Horowitz
  – Sec 7.5,
- Text book 1: Levitin
  – Sec 12.1,12.2
- R1: Introduction to Algorithms
    - Cormen et al.
- Youtube link of video lecture recording
  – https://www.youtube.com/watch?v=LgLrJJ3CaiQ

Hamilotonian Cycles

- Hamiltonian cycle:
- Given a graph $G=(V,E)$, a hamiltonian cycle is
  - A round trip path along n edges of G
  - That visits each vertex once, and
  - Returns to starting vertex.
  - Considering that $v_1 \in G$ is the start vertex, and
  - Vertex visited are in the order $v_1, v_2, \ldots, v_{n+1}$, then
    - Edge $(v_i, v_{i+1}) \in E$, $1 \leq i \leq n$, and all vertices v_i are distinct except that $v_1 = v_{n+1}$
- TSP:
  - TSP is a hamiltonian cycle with minimum cost.

Examples

- Does the following graphs have a hamiltonian cycle?

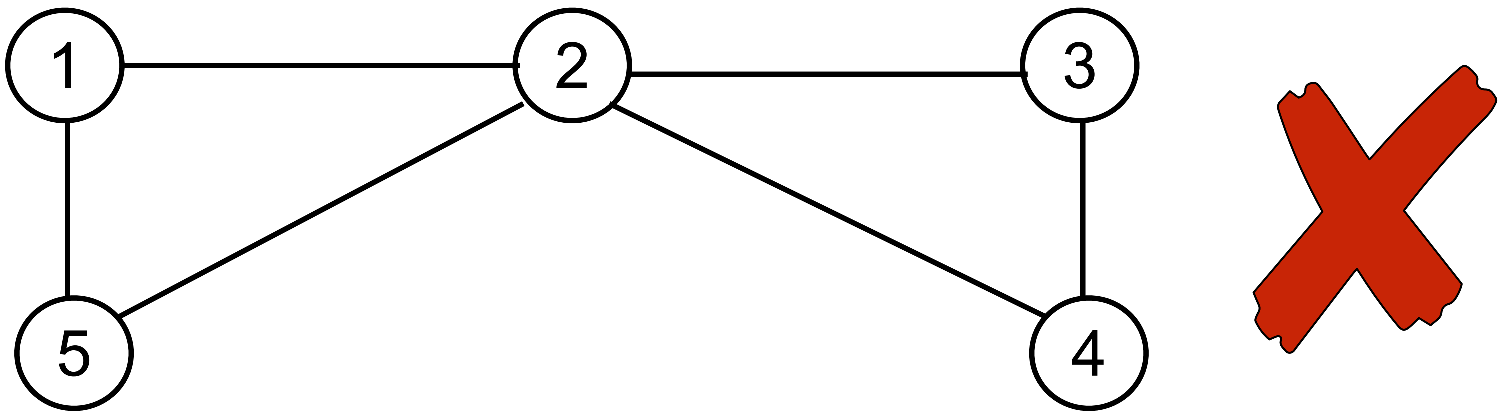


- HC1:

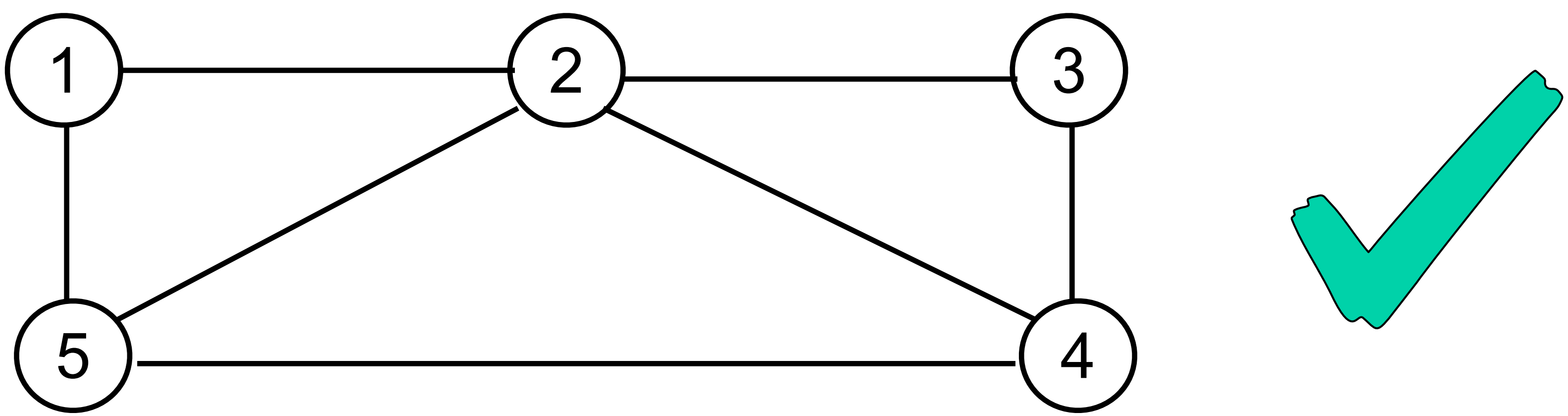
  `1,2,8,7,6,5,4,3,1` or
  `1,3,4,5,6,7,8,2,1`

Examples

- Does the following graphs have a hamiltonian cycle?



- Does the following graphs have a hamiltonian cycle?



`1,2,3,4,5,1` or
`1,5,4,3,2,1`

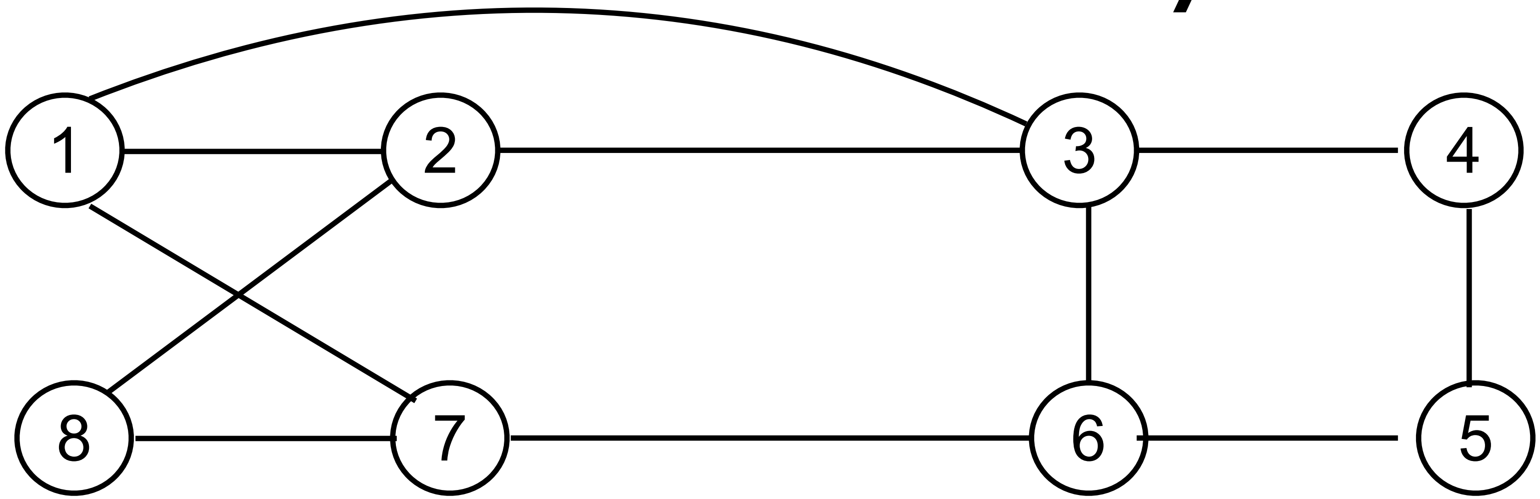
Hamiltonian Cycle

- It is an NP complete problem i.e.
  - There is no easy way (polynomial time computation) to know if the graph contains a hamiltonian cycle.
- Backtracking is an approach to find all hamiltonian cycles
  - Graph can be directed or undirected.
- Backtracking approach
  - Consider solution vector: $(x_1, x_2, \ldots, x_n)$
    - x_i represents i^{th} visited vertex of proposed cycle
  - How to compute possible vertices for x_k when vertices $x_1, x_2, \ldots, x_{k-1}$ has already been chosen

HC: Backtracking Approach

- Graph `G` is maintained as adjacency matrix
- Choosing `x[k]` when `x[1], x[2],..., x[k-1]` is chosen
  - i.e. after first `k-1` vertices are chosen, chose next k^{th}
- If `k=1`, then `x[1]` can be any vertex `v`∈`V`
  - Note `x[k]` doesn't imply vertex `k`
  - it implies k^{th} vertex on the Hamiltonian cycle path
- For simplicity, assume `x[1]=1` (we can start from any `v`)
- When `1<k<n`, then `x[k]` can be any vertex `v` that is distinct from `x[1], x[2],..., x[k-1]`, and
  - `v` is connected to `x[k-1]` by an edge.
- Vertex `x[n]` must be connected to both `x[1]` and `x[n -1]`
- The algo has two parts,
  - `nextValue(k)` to determine next vertex
  - the main algo loop `Hamiltonian(k)`

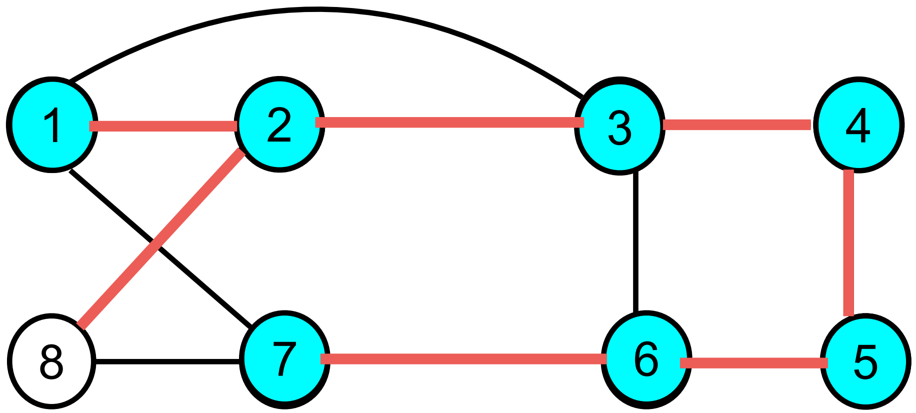
Hamiltonian Cycle



- HC1: 1,2,8,7,6,5,4,3,1 or
- HC2: 1,3,4,5,6,7,8,2,1

	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]	X[8]	X[1]
HC1	1	2	8	7	6	5	4	3	1
HC2	1	3	4	5	6	7	8	2	1

Hamiltonian Cycle



X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]	X[8]	X[1]
1	2	3	4	5	6	7	0	0

Invocation: x[1]=1, k=1

Choose next vertex i.e. k=2

x[2] can be 2, 3, or 7; Let x[2]=2

next k=3; x[3] can be 1,3 or 8; Let x[3]=3

next k=4; x[4] can be 1, 2,4 or 6; 1, 2 already visited; Let x[4]=4

next k=5; x[5] can only be 5; Thus x[5]=5

next k=6; x[6] can only be 6; Thus x[6]=6

next k=7; x[7] can only be 3 or 7
3 is already visited; Thus x[7]=7

next k=8;
Last vertex
x[8] can only be 1 or 8
1 is already visited; From 8 can't reach 1
Thus need to backtrack

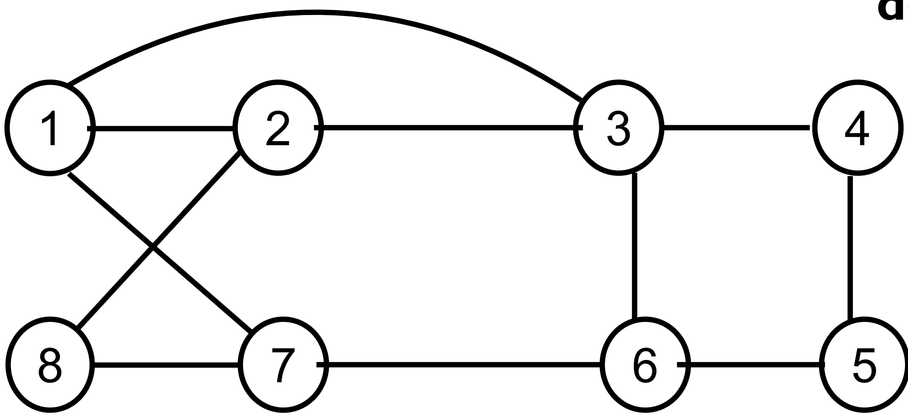
Algo: Hamiltonian Cycle…

```
proc NextValue(k)
// x[1],…,x[k-1] is a path of k-1 distinct vertices
// x[k]=0 implies no vertex is assigned to x[k]
//  Initially, all x[i]=0
//x[k] is a vertex not in x[1],…,x[k-1], and connected to x[k-1]
do
    x[k]=(x[k]+1)%(n+1) // next vertex from 1 to n .....N1
    if (x[k]==0) then return // all n vertices explored .....N2
    if (G[x[k-1]][x[k]]==1) // edge x[k-1]–x[k] ............N3
      for j=1 to k-1 do ............N4
        if (x[j]==x[k]) // vertex already in the path ............N5
          break ............N6
      if (j==k) //if last vertex, check for edge with x[1] ............N7
        if ((k<n)||(k==n)&&(G[x[n][x1]]==1)) .........N8
          return ............N9
while True
```

Algo: Hamiltonian Cycle (Main)

```
Algo Hamiltonian(k)
// uses recursive formulation of backtracking to find all HCs of G
// Graph is stored as adjacency matrix G[1:n][1:n]
//All cycles start at node 1. Initially, all x[i]=0
do // generate values for k^th node i.e. x[k]   ...........
    NextValue(k) // assign a legal value to  x[k]    ...........A1
    if (x[k] == 0) // no legal value can be found   ...........A2
        return
    if (k==n) // if last node reached, print path    ...........A3
        for i=1 to n do                              ...........A4
            print x[i]                               ...........A5
    else  // discover next node in the path
        Hamiltonian(k+1)                             ...........A6
while True
```

Execution of HC Algo



X[1]	X[2]
1	2

Algo NextValue(k)

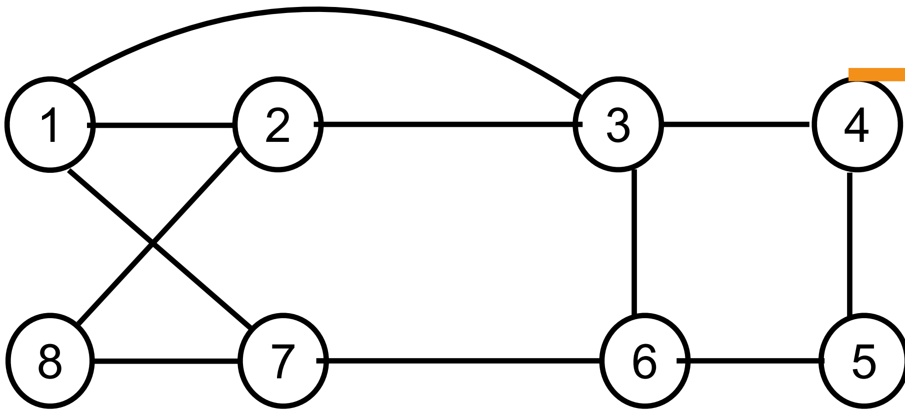
```
do
  x[k]=(x[k]+1)%(n+1) // .....N1
  if (x[k]==0) then return // .....N2
  if (G[x[k-1]][x[k]]==1) //.......N3
    for j=1 to k-1 do        // ............N4
      if (x[j]==x[k]) //     ............N5
        break                ............N6
    if (j==k) //check for edge with x[1] ......N7
      if ((k<n)||(k==n)&&(G[x[n][x1]]==1))...N8
        return
while
```

Invocation: x[1]=1,
Hamiltonian(2) i.e. k=2
A1: invoke NextValue(2)

- N1: k=2→x[2]=(0+1)%9=1=1, Thus x[2]=1
- N2: x[k]==0 (False)
- N3: G[x[1]][x[2]]→G[1][1]==1 (False, no self edge)
- N1: x[2]=(1+1)%9=1=2, Thus x[2]=2 (do while loop)
- N2: x[2]==0 (False)
- N3: G[x[1]][x[2]]→G[1][2]==1 (True, edge 1-2)
- N4: j=1 (iterates over 1)
- N5: x[1]==x[2] (False)

N7: j==k //2=2 (True)
N8: k<n //2<8 (True)
return (i.e. x[2]=2)

Execution of HC Algo



X[1]	X[2]	X[3]
1	2	1

```
Algo Hamiltonian(k)
do // for kth node i.e. x[k]   ............
    NextValue(k)// next  x[k]  ............A1
    if (x[k]  ==  0)// no legal value............A2
        return
    if (k==n) // last node, print path............A3
        for i=1 to n do                   ............A4
            print x[i]                    ............A5
    else  // discover next node in the path
        Hamiltonian(k+1)                  ............A6
while True
```

A2: x[k]==0 (False since k=2, x[2]=2)

A3: k==n (False since k=2, n=8)

A6: Hamiltonian(3) (since k=2)

A1: invoke NextValue(3)

N1: k=3→x[3]=(0+1)%9=1

N2: x[k]==0 (False)

N3: G[x[2]][x[3]]→G[2][1]==1 (True)

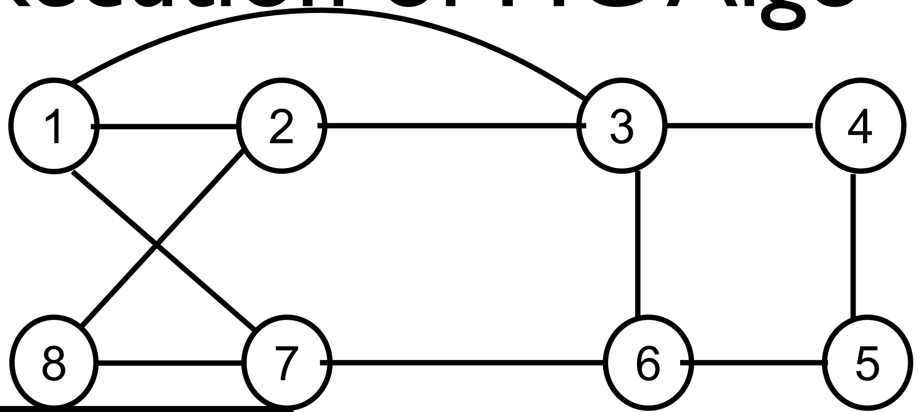
N4: j=1 (iterates over 1, 2)

N5: x[1]==x[3] (True, 1=1, node 1 already in path)

N6: break (continue from do-while loop)

```
Algo NextValue
do
    x[k]=(x[k]+1)%(n+1) // .....N1
    if (x[k]==0) then return // .....N2
    if (G[x[k-1]][x[k]]==1) //......N3
        for j=1 to k-1 do       // ...........N4
            if (x[j]==x[k]) //     ...........N5
                break              ...........N6
        if (j==k) //check for edge with x[1].
            if ((k<n)||(k==n)&&(G[x[n]][
                return
while
```

Execution of HC Algo



X[1]	X[2]	X[3]
1	2	3

```
do                        Algo NextValue
  x[k]=(x[k]+1)%(n+1) // .....N1
  if (x[k]==0) then return // .....N2
  if (G[x[k-1]][x[k]]==1) //......N3
    for j=1 to k-1 do        // ...........N4
      if (x[j]==x[k]) //     ...........N5
          break               ...........N6
    if (j==k) //check for edge with x[1]...N7
        if ((k<n)||(k==n) &&
            (G[x[n][x1]]==1))...N8
            return
while
```

N6: break (Continue from do-while loop)

N1: k=3, x[k]=(1+1)%9=2

N2: x[k]==0 //False x[3]=2

N3: G[x[2]][x[3]]→G[2][2]==1 (False)

Go to next iteration of do-while

N1:k=3, x[3]=(2+1)%9=3

N2: x[3]==0 (False)

N3: G[G[x[2]][x[3]]→G[2][3]==1 (True)

N4: j=1 (iterate over 1, 2)

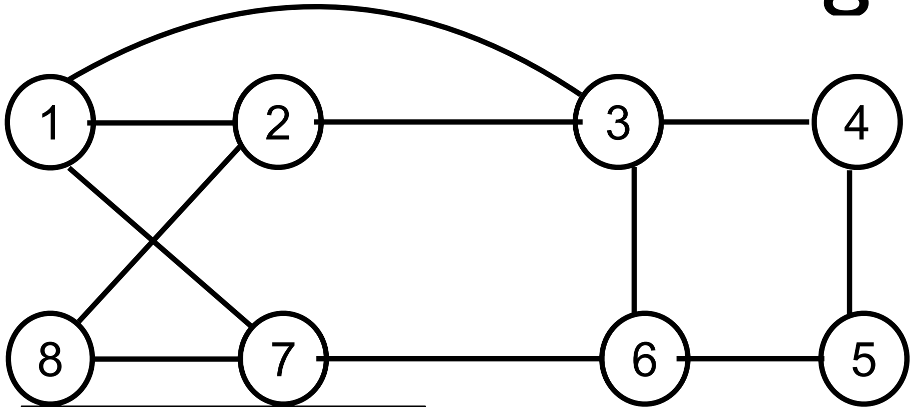
N5: x[j]==x[k] (False x[1]=1,x[3]=3)

N4: j=2

N5: x[j]==x[k] (False x[2]=2,x[3]=3)

N4: j=3 (loop condition breaks)

Execution of HC Algo



X[1]	X[2]	X[3]	X[4]
1	2	3	0

```
do                              Algo NextValue
   x[k]=(x[k]+1)%(n+1) // .....N1
   if (x[k]==0) then return // .....N2
   if (G[x[k-1]][x[k]]==1) //......N3
      for j=1 to k-1 do        // ............N4
         if (x[j]==x[k]) //      ............N5
            break               ............N6
      if (j==k) //check for edge with x[1]...N7
         if ((k<n)||(k==n) &&
            (G[x[n][x1]]==1))...N8
               return
while
```

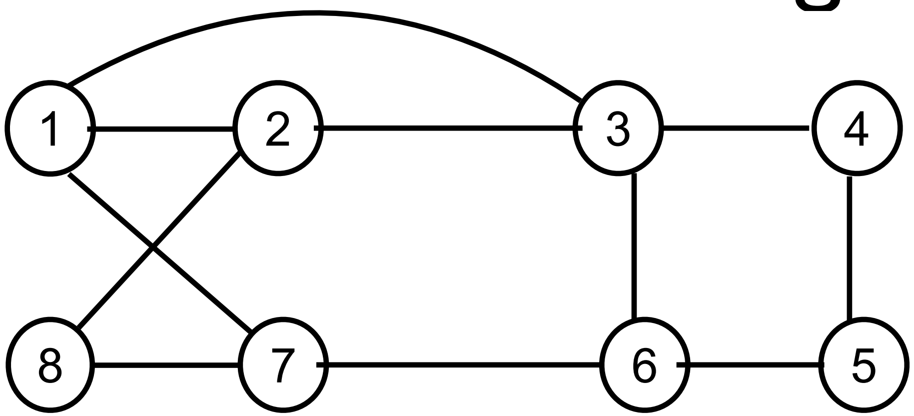
```
N4: j=3 (loop condition breaks)
N7: j==k (True)
  N8: k<n (True k=3, n=8)
  N9: return to A1 with k=3,x[3]=3
A1: k=3, x[3]=3
A2: x[k]==0 (False)
A3: k==n (False)
A6: Hamiltonian(k+1=4) //next.
```

Proceeding in this way will lead to

```
x[4]=4, Hamiltonian(5)
x[5]=5, Hamiltonian(6)
x[6]=6, Hamiltonian(7)
x[7]=7, Hamiltonian(8)
```

```
Algo Hamiltonian(k)
do // for kth node i.e. x[k]   ............
   NextValue(k)// next x[k]   ............A1
   if (x[k] == 0)// no legal value............A2
      return
   if (k==n) // last node, print path............A3
      for i=1 to n do                   ............A4
         print x[i]                     ............A5
   else // discover next node in the path
      Hamiltonian(k+1)                  ............A6
while True
```

Execution of HC Algo



do Algo NextValue

```
x[k]=(x[k]+1)%(n+1) // .....N1
if (x[k]==0) then return // .....N2
if (G[x[k-1]][x[k]]==1) //......N3
  for j=1 to k-1 do      // ............N4
    if (x[j]==x[k]) //     ............N5
      break              ............N6
  if (j==k) //? edge with x[1]  ...N7
    if ((k<n)||(k==n) &&
       (G[[x[n][x[1]]]==1))  ...N8
       return
```

while

Invocation of `Hamiltonian(8)`

`A1`: invoke `NextValue(8)`

It will fail at condition (`G[[x[n][x[1]]]==1`) ...`N8`, and then

at `N1`, `x[8]=(8+1)%=0`

and thus condition at `N2`, `x[k]==0` becomes **True**

return.

It keeps returning from recursive invocation, and then at the first invocation of `Hamiltonian(2)`,

for `k=3`, `x[3]=8` at `N1`

It will proceed in this further and will find a cycle

`1,2,8,7,6,5,4,3,1`

`mColoring` of Graph

- Problem:
  - Given a graph `G=(V,E)`, and a number `m`
  - Color the nodes of the graph in such a way that
  - No two adjacent nodes have same color
  - At most `m` colors are used.
- Note: if `d` is degree of graph, then graph can be colored with `d+1` colors.
- `m-colorability` optimization problem
  - Find smallest integer `m` for which `G` can be colored.
  - `m` is called `chromatic number` of G.

Planar Graph

- Problem:
  - A graph G=(V,E) which can be drawn in a plane in such a way that no two edges cross each other.
- A planar graph can always be colored with 4 colors.
  - For a long time, value 5 was considered sufficient.
- Planar graph has a useful application in map coloring.
  - A map (in a plane) can always be represented as a graph.
  - Each region in the map is a node
  - For two neighbour regions in the map, graph has an edge between those two respective nodes
- Consider graph is represented by adjacency matrix.
  - G[i][j]=1 if there is a edge (i,j) else G[i][j]=0

`m-coloring` of Graph

- For simplicity, consider that colors are represented as
  - `1,2,3,…,m`.
- Solution of `m`-color problem is given by a tuple
  - $x_1, x_2, \ldots, x_n$, where x_i is the color of i^{th} node
- Approach: Recursive backtracking formulation
  - Consider state space tree of degree `m`
    - Each edge represents color assignment to a node
    - Each intermediate node at level `i` has `m` children.
      - Corresponding to `m` possible values for x_i.
    - Tree height is `n+1`
      - Nodes at level `n+1` are leaf nodes.

Algo mColoring...

```
Algo mColoring(k)
// color for a node i is given by x[i], initialized to 0
// Graph is adjacency matrix, value 1 when edge exists else 0
do  // generate all legal assignments for x[k]
    NextColor(k) //assign to x[k] a legal value
    if (x[k]==0)
        break //no new color possible.
    if (k==n) // all nodes have been colored, at most m colors
        //out put the color of each node
        for i=1 to n do
            print(x[i])
    else
        mColoring(k+1)
while True
```

Algo mColoring...

```
proc NextColor(k)
// i/p: nodes x[1],…,x[k-1] are assigned colors, range [1..m]
// o/p: value of x[k] is assigned in range [0..m], 0 means no color
do
    x[k]=(x[k]+1)%(m+1) // next highest color
    if (x[k]==0) // no color can be assigned.
        return
    for j=1 to n do //is color of x[k] is distinct from neighbours
        if (G[k][j]==1)&&(x[k]==x[j]) // adjacent same color
            break
    if (j==n+1) // for loop index completed
        return // new color found
while True //try to find next color
```

Complexity Analysis: `mColoring`

- Number of internal nodes in state space tree

$\Sigma_{0\le i\le n-1}\ m^i$

- At each node, $O(mn)$ time is spent by `NextColor`
  - Determines corresponding legal coloring
- Thus, total time complexity is given by

$$\Sigma_{0\le i\le n-1}\ m^i * mn$$
$$=\Sigma_{0\le i\le n-1}\ m^{i+1} * n$$
$$=n((\Sigma_{1\le i\le n}\ m^i))$$
$$=n((\Sigma_{0\le i\le n}\ m^i)-\ 1)$$
$$=n[(m^{n+1}-1)/(m-1)-1]$$
$$=O(nm^n)$$

Summary

- Hamilotonian Cycles
- m-Coloring of a graph