Design and Analysis of Algorithms

L43: Backtracking Algorithms

Hamiltonian Cycles & m-Coloring of a Graph

Dr. Ram P Rustagi
Sem IV (2019-H1)
Dept of CSE, KSIT/KSSEM
rprustagi@ksit.edu.in

Resources

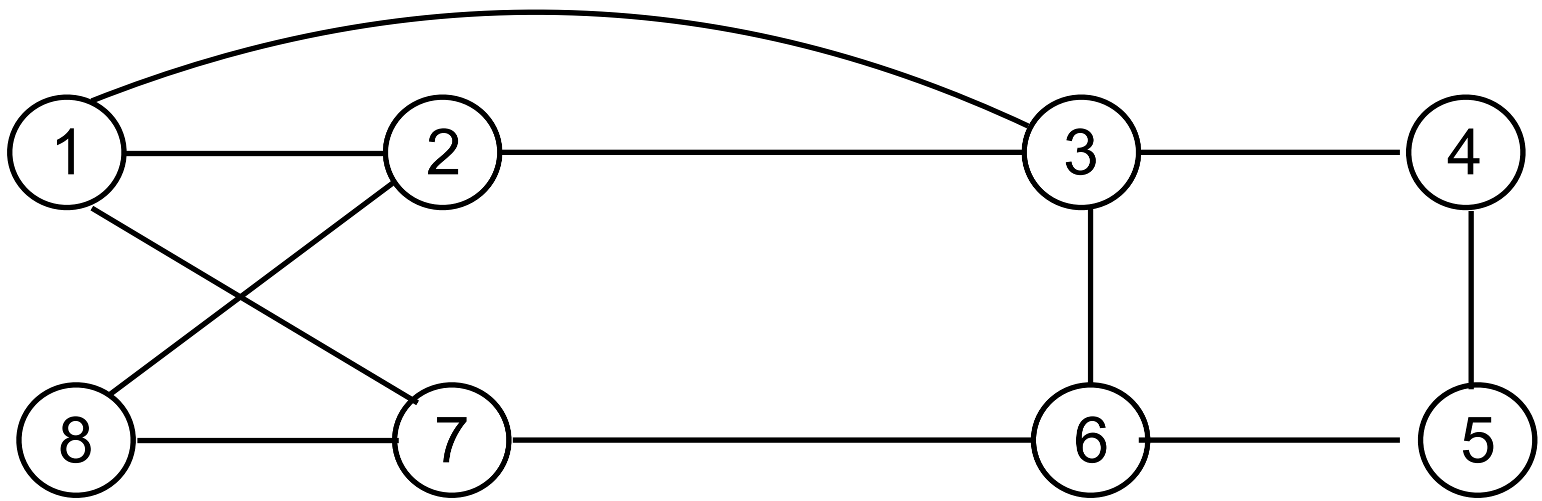
- Text book 2: Horowitz
  - Sec 7.1, 7.2, 7.3, 7.4, ***7.5***, 8.2, 11.1
- Text book 1: Levitin
  - Sec 12.1, 12.2
- R1: Introduction to Algorithms
  - Cormen et al.
- Youtube link of video lecture recording
  - https://www.youtube.com/watch?v=LgLrJJ3CaiQ

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?



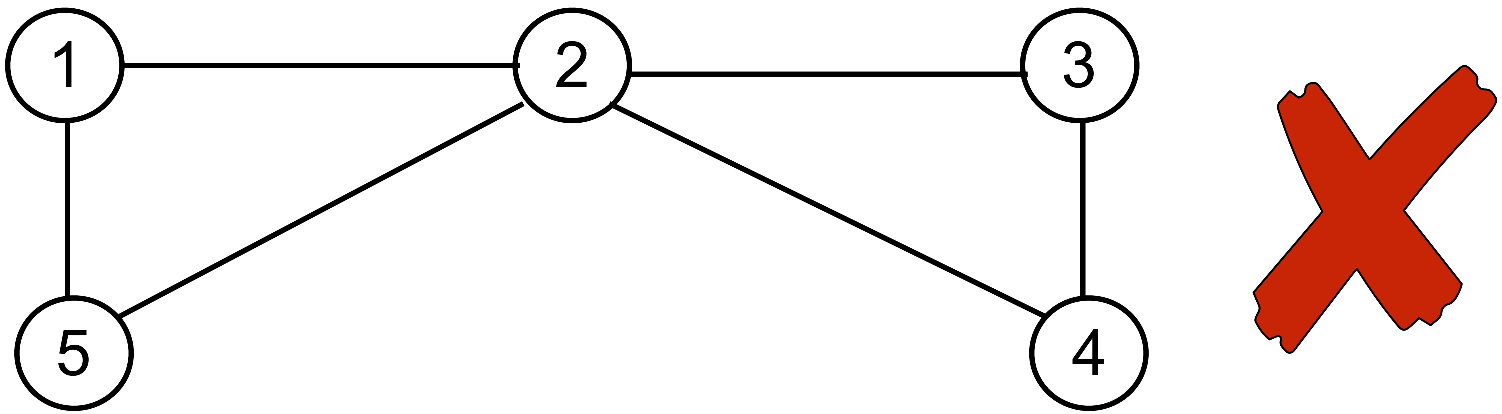
- HC1:

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

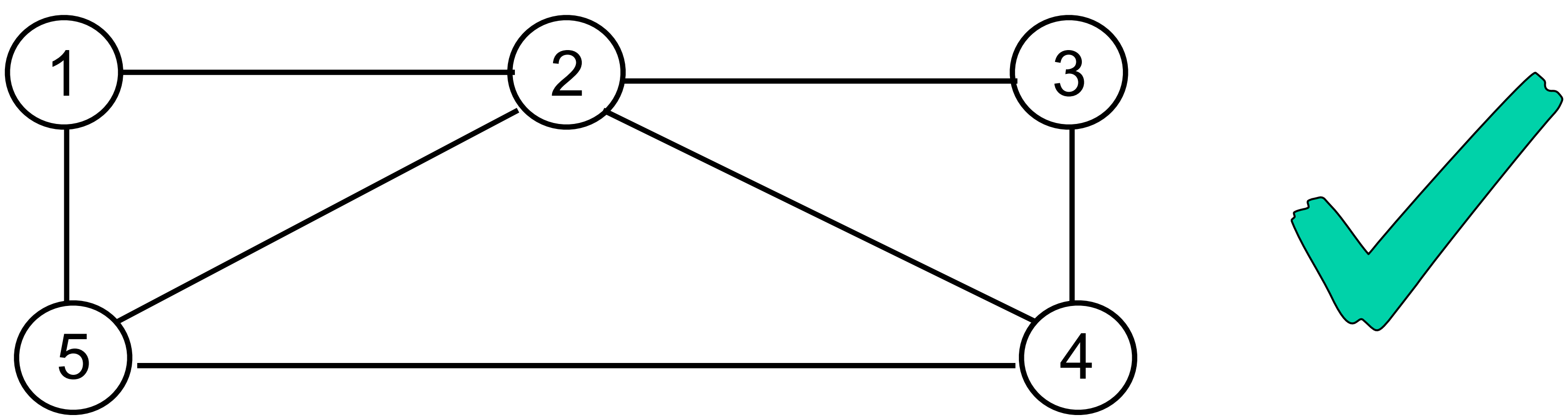
  `1,3,4,5,6,7,8,2,1`

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 $x_1, x_2, \ldots, x_{k-1}$ is chosen
- If $k=1$, then x_1 can be any vertex $v \in V$
- For simplicity, assume $x_1=1$.
- when $1<k<n$, then x_k can be any vertex v that is distinct from $x_1, x_2, \ldots, x_{k-1}$, 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

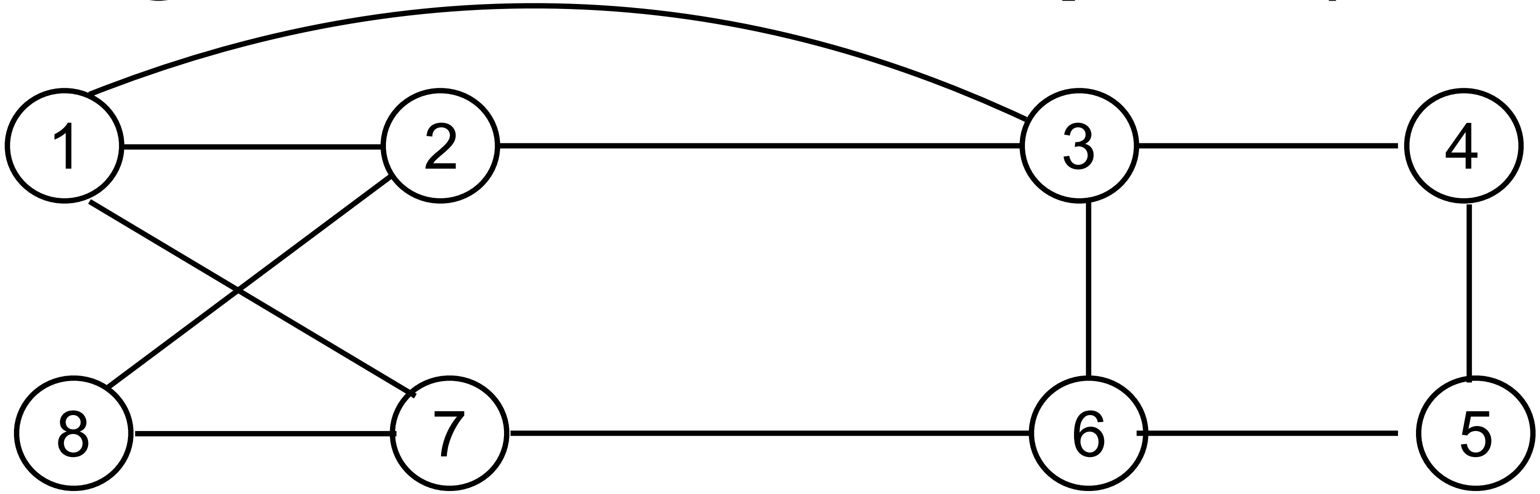
Algo: Hamiltonian Cycle…

```
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]
//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
    if (x[k]==0) then return
    if (G[x[k-1]][x[k]]==1) // is there edge x_{k-1}—x_k
      for j=1 to k-1 do
        if (x[j]==x[k]) // vertex already in the path
          break
      if (j==k) //if last vertex, check for edge with x[1]
        if ((k<n)||(k==n)&&(G[x[n][x1]]==1))
          return
while True
```

Algo: Hamiltonian Cycle (Main)

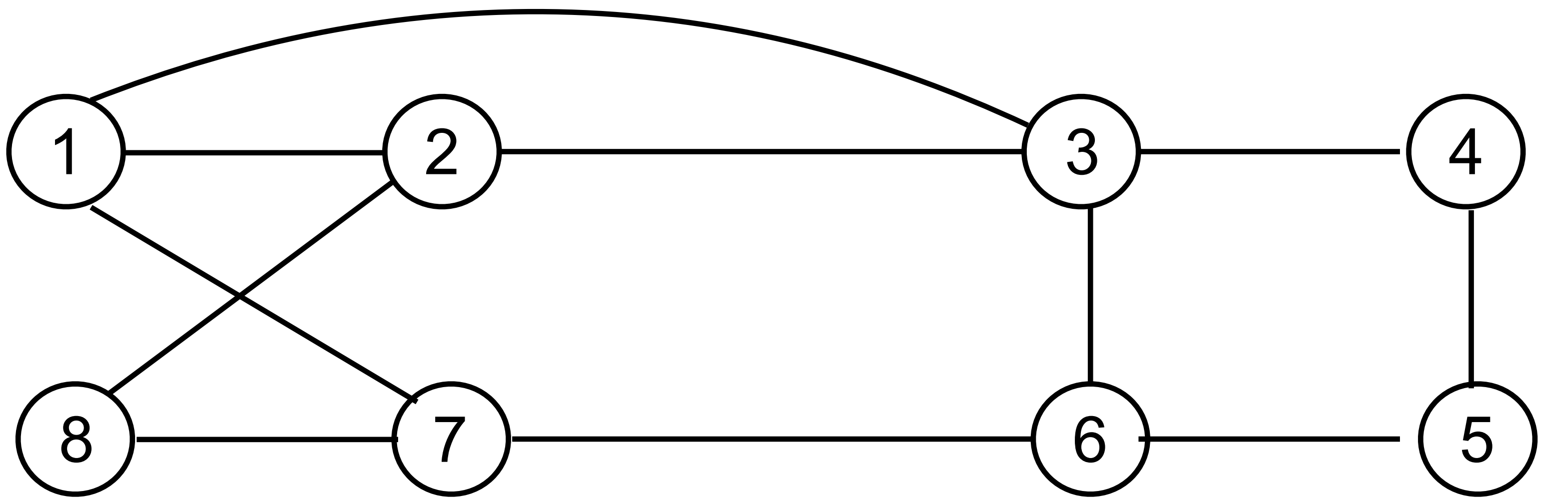
```
Algo Hamiltonian(k)
// uses recursive formulation of backtracking to find all HCs of G
// Graph is stored as adjacency matrix G[1:n][1:n]
//All cycles start at node 1. Initially, all x[i]=0
do // generate values for k^th node i.e. x[k]   ............
    NextValue(k) // assign a legal value to  x[k]   ............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
```

Algo: Hamiltonian Cycle (Main)




```
Algo Hamiltonian(k)
do // generate values for kth 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
```

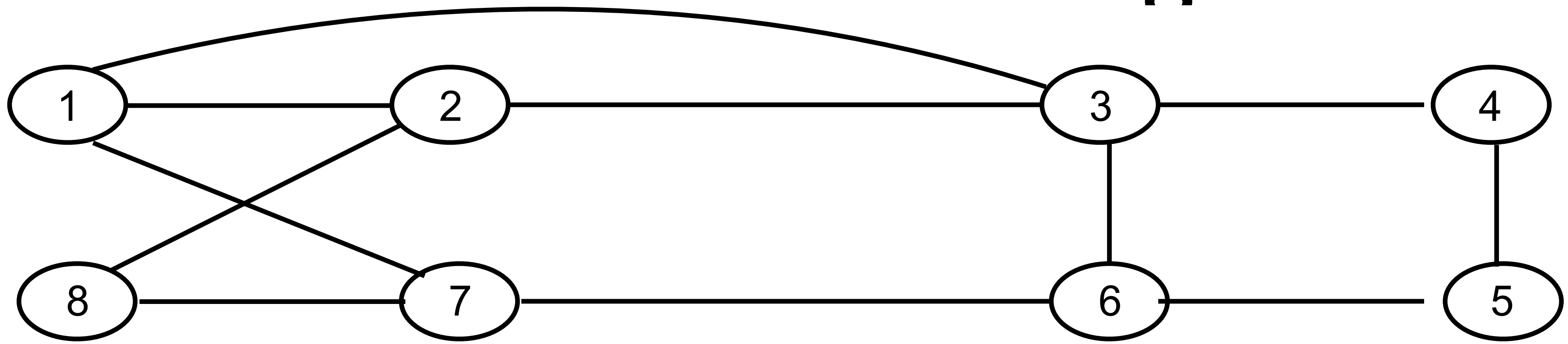
Execution of HC Algo



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, edge exists)
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)

Execution of HC Algo



`N6: break` (Continue from do-while loop)

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

`N2: x[k]==0` False

`N3: G[x[2]][x[3]]→G[2][2]==1` (False no self edge)

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`)

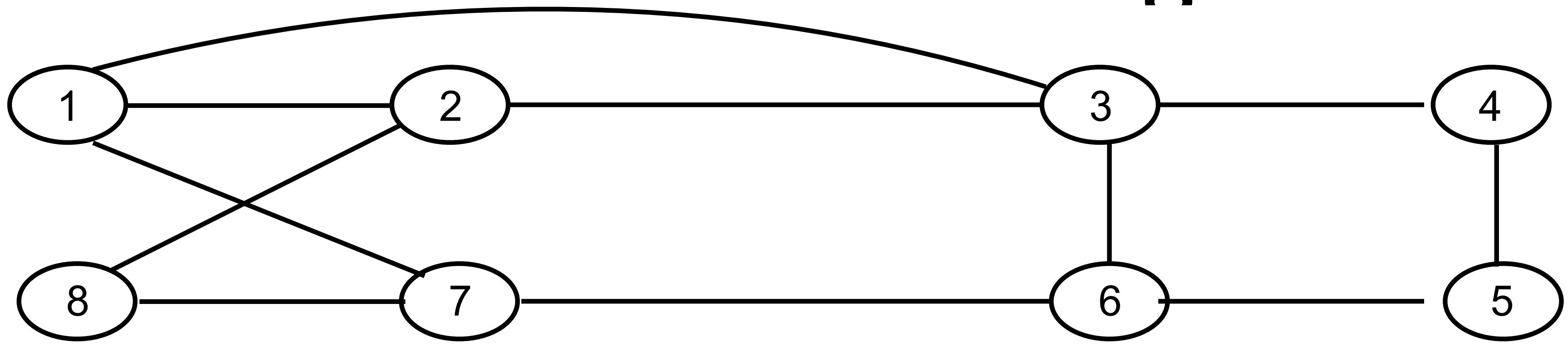
`N5: x[j]==x[k]` (False `j=1,k=3` and `x[1]=1,x[3]=3`)

`N4: j=2`

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

`N4: j=3` (loop condition breaks)

Execution of HC Algo

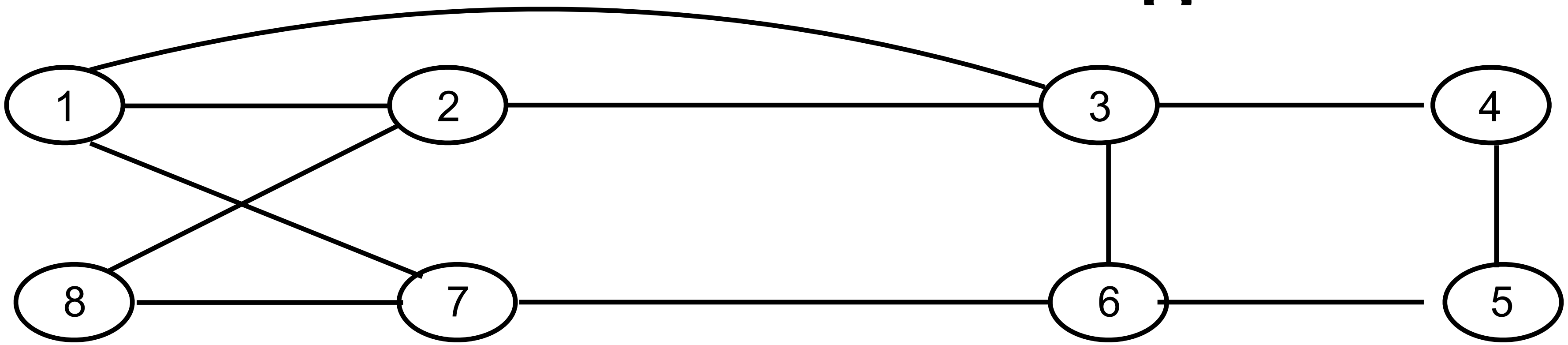



```
  N4: j=3  (loop condition breaks)
  N7: j==k  (True)
    N8: k<n  (True k=3, n=8)
    N9: return to A1 with k=3,x[3]=3
A1: k=3, x[3]=3
A2: x[k]==0  (False)
A3: k==n  (False)
A6: Hamiltonian(k+1=4) //next invocation.
```

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)
```

Execution of HC Algo



Invocation of `Hamiltonian(8)`

`A1`: invoke `NextValue(8)`

It will fail at condition (`G[x[n][x1]]==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=2, x[2]=(2+1)%9=3` at `N1`

It will proceed in this further and will find a cycle `1,3,4,5,6,7,8,2.`

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
  - and 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
  - $1, 2, 3, \ldots, m$.
- 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`
  - to determine children corresponding legal coloring
- Thus, total time complexity is given by

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

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

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

  $= n[(m^{n+1}-1)/(m-1)-1]$

  $= O(nm^n)$

Summary

- Hamilotonian Cycles
- m-Coloring of a graph