Unit – 5: Backtracking

M.B.Chandak

hodcs@rknec.edu

www.mbchandak.com

Unit wise course: What are you learning

UNIT-V: Backtracking: CO3

- *Basic Traversal and Search Techniques, breadth first search and depth first search, connected components.*
- ***Backtracking basic strategy, graph coloring, Hamiltonian cycles, 8-Queen's problem, sum of subset problem, Introduction to Approximation algorithm.***

- **Design Dynamic programming and Backtracking Paradigms to solve the real life problems.**

Basic of Backtracking

- Applications: Game Theory
- For every given problem: **Input** is represented in the form of Tuple [x1..xn].
- The solution is generated by selecting proper input from the tuple [x1..xn]. Each selected input must satisfy the **criterion function**.
- Since Backtracking is **"Selection" based solution,** at each stage the selection may be optimized by testing **validity** of criterion function [constraints]

Basics

- **Basic Principle:** Solution is constructed component-wise and at each point the component is tested against the criterion function. The construction of solution continues, if criterion function is satisfied.
- For example, if there are "n" elements then first component can be (x1..xi) is checked against (p1..pi) and if partial solution and partial criterion function are not matching then remaining part of solution is simply ignored.
- *(x1..xi) is partial solution and (p1..pi) is partial criterion function.*

Basics

- Problem solving using backtracking requires that all the solutions must satisfy the complex set of constraints. These constraints can be classified into two classes:
- ***Explicit Constraints***
- ***Implicit Constraints***
- **Explicit Constraints: [Direct]**
- They are set of rules which allows "xi" to take value only from the given set.
- For example: xi >= 0 or xi = 0 or 1

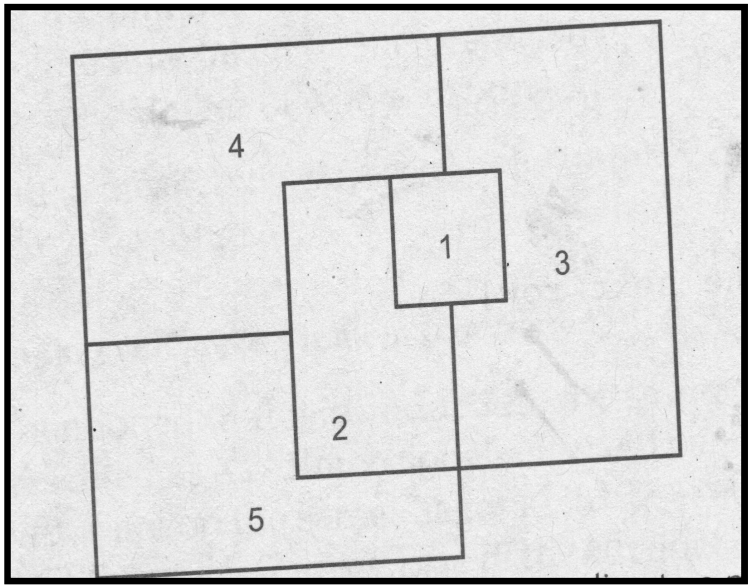
Basics

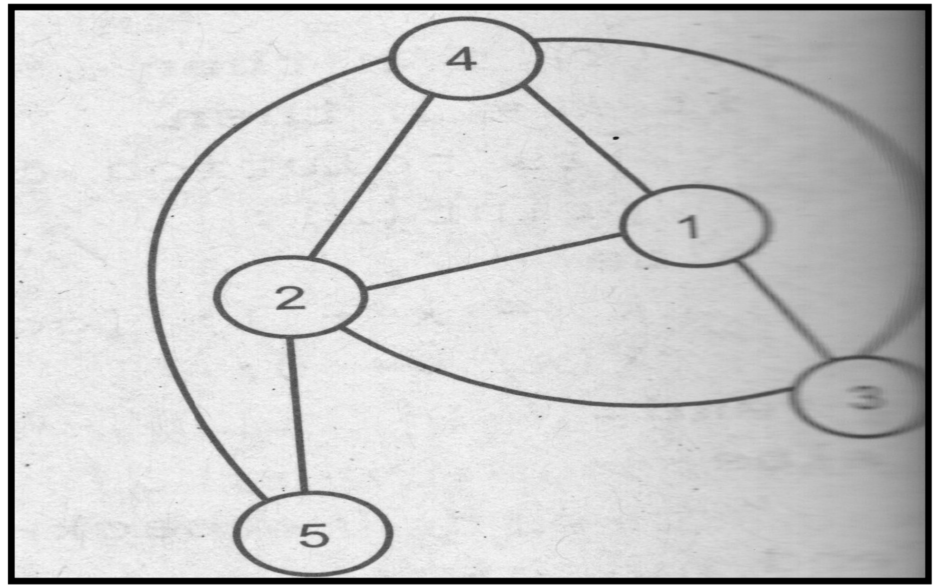
- **Implicit Constraints: [Indirect]**
- These constraints are the rules that decides which of the tuple in the solution space of "i" satisfies criterion function
- Example: Sum of subset problem: a=[12,5,22,9,2]
- Sum=24
- There are various combinations: 9,5,12 or 12,5,9
- Implicit constraints will allow combination 12,5,9 or 22,2 and will not allow 2,22 or 9,5,12 etc..
- For above problem Implicit constraint can be stated as: Use values in array only once and in the order of occurrence in given array.

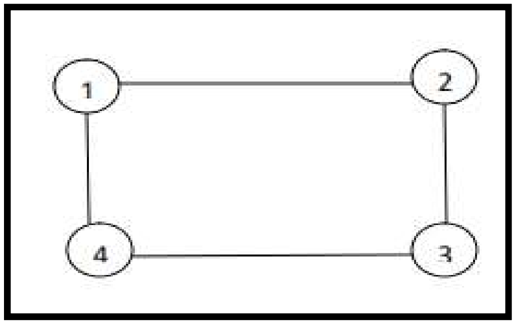
Graph Coloring: Algorithm

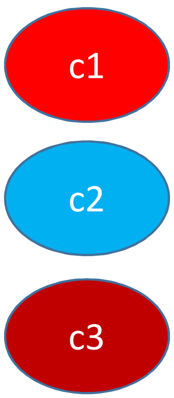
- Algorithm solution for problem solved using **BACKTRACKING are RECURSIVE**
- The input to algorithm is vertex number present in the graph
- The algorithm generates the color number assigned to vertex and stores it an array.
- If the constraint are not matched at any point, then remaining part of algorithm is not executed and new cycle is initiated.
- The algorithm terminates with a solution satisfying the constraints.
- ***Application: Area demarcation/Map Coloring/Remote Sensing Applications***

Chromatic Number

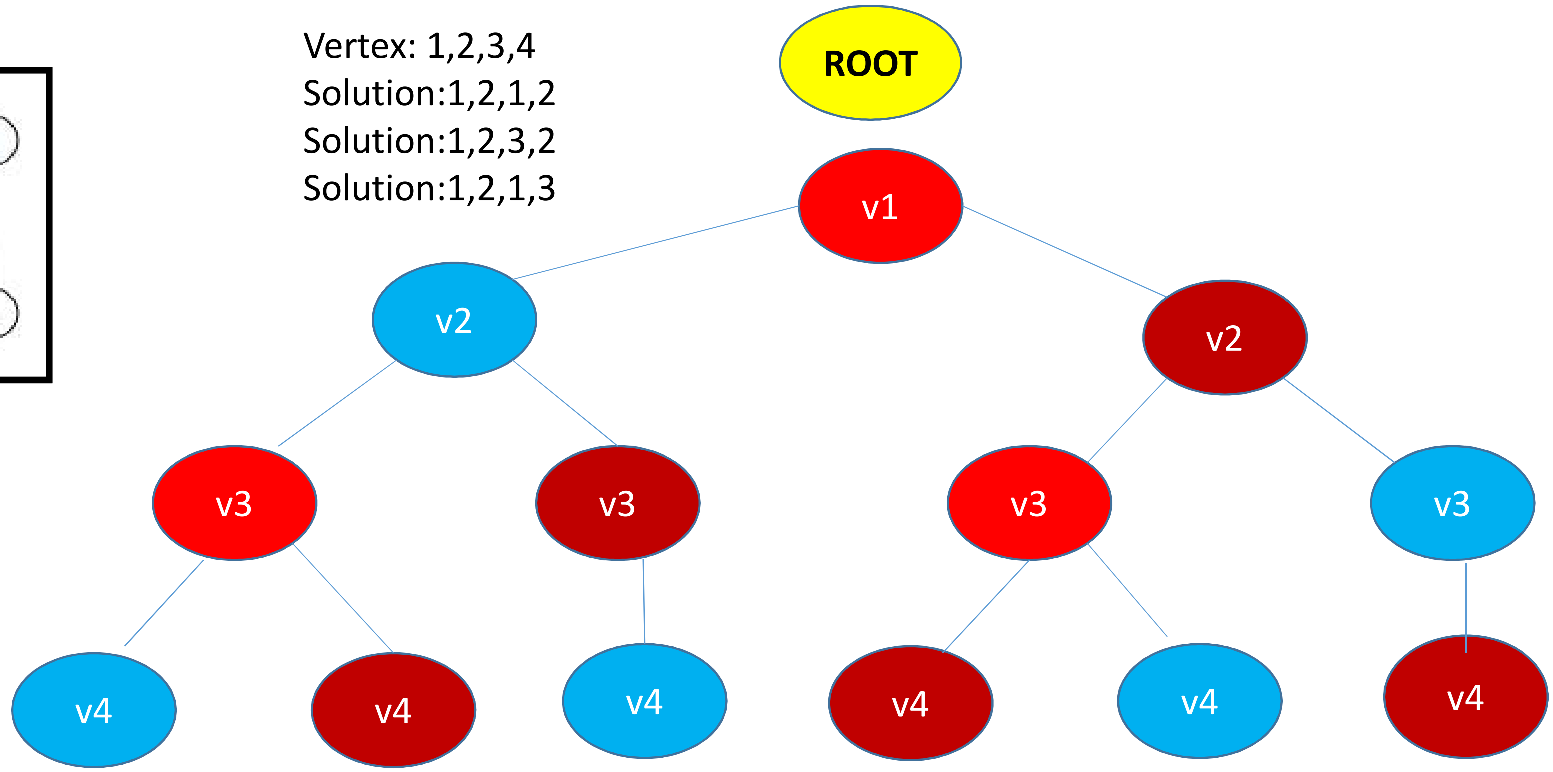








Vertex: 1,2,3,4
Solution:1,2,1,2
Solution:1,2,3,2
Solution:1,2,1,3



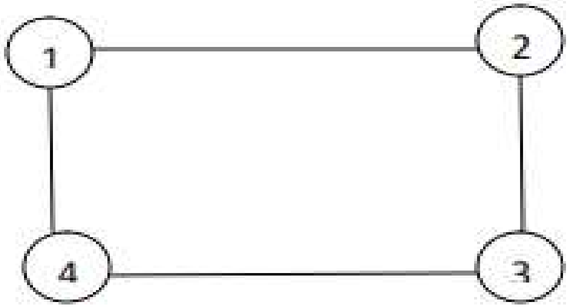
Degree of Vertex=2
Number of colours required will be d+1 = 3 [Generic solutions]. Optimization is possible in some cases based on density of graph

Part – I: mcolor

- This component passes the different values of vertex to “nextvalue” algorithm, and accept the value returned by “nextvalue” algorithm.
- The value is checked and if found suitable then stored in output array.
- The output array x[1..n] is solution for the given problem.

ALGORITHM FOR GRAPH COLORING

```
Algorithm mcolor(k)
{
    The graph is represented in the form of matrix nxn
    "k" is an index of vertex to be colored
    {
        repeat
        {
                Nextvalue(k)
                If (x[k] = 0) then return
                If (k=n) then
                        Write(x[1:n])
                Else
                        Mcolor(k+1)
        } until(false)
    }
}
```



Algorithm: nextvalue

- This algorithm will accept the vertex number from “mcolor” and generates color value for the vertex, with constraint satisfaction.
- To check constraints following conditions are tested:
- Let vertex “k” is in process, then all vertices in the graph are tested for adjacency test.
- If the vertices are adjacent, then, should be different colors
- The color information is stored in array “x”
- Hence G[k,j] != 0 and x[k] = x[j] then color is not suitable otherwise suitable.

Logic for m=3 [v1=c1, v2=c2, v3=c1, v4=c2]

Array “X”	**X[1]=V1**	X[2]	X[3]	X[4]
Initial stage	0	0	0	0
Mcolor(1)	Call nextvalue(1)			
	X[1] = 1			
Mcolor(2)		**X[2]=1 [Break]**		
		X[2]=2		
Mcolor(3)			X[3]=1	
Mcolor(4)				X[4]=2

```
Algorithm nextvalue(k)
Assume x[1..k-1] are assigned integer in the range of [1,m] such that no
two adjacent vertices are in the same color
x[k] is assigned the next value such that distinctness is maintained
If no such color exists then x[k]=0
{
    repeat
    {
         x[k] = (x[k] + 1) mod m+1
         if x[k] = 0 then return
         for j = 1 to n do
         {
              If (G[k,j]!=0) and (x[k]=x[j]) then
                   Break
         }
         If (j=n+1) then
              Return
    }until (false)
}
```