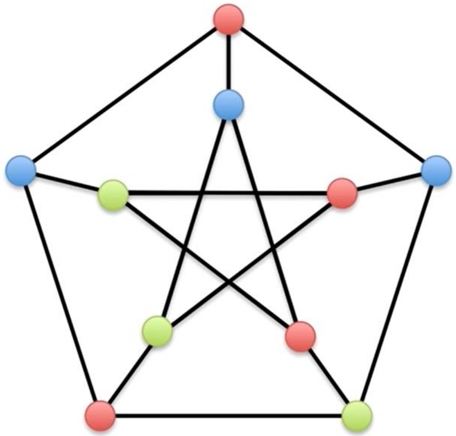
Graph Coloring Problem

Presented by Nabanita Das

Graph Coloring Problem

- Given an undirected graph, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with the same color.
- Goal: Given a graph G and an integer m, find if we can satisfy the problem description using at most m colors. n = 10, m=3



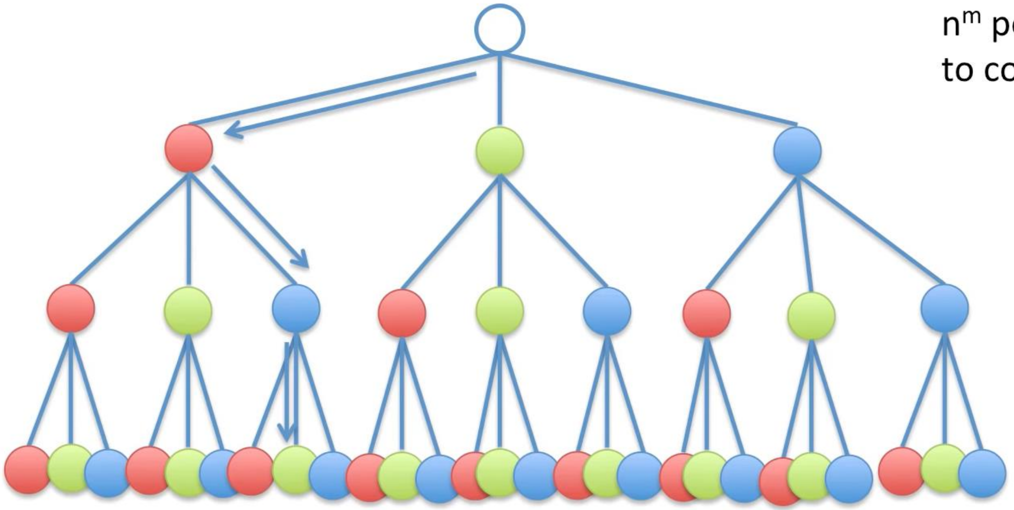
Graph Coloring Problem Finding

- **m-Coloring Decision Problem**-If a graph is given and also some colors are given. This graph can be colored or not by those colors.
- **m-Coloring Optimization Problem**- If a graph is given if we want to know minimum how many colors are required to color the graph.
- **Chromatic Number**- The chromatic number of a graph is the smallest number of colors needed to color the vertices so that no two adjacent vertices share the same color.

- Let's look at a smaller problem:

  n=3, m=3

- No Restrictions

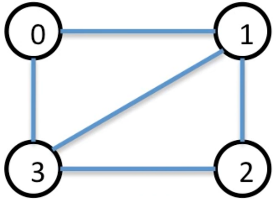


n^m possible ways to colour the 3 nodes

$3^3=27$

Graph Colouring Backtracking

Example Problem: n=4, m=3



Adjacency Matrix (G)

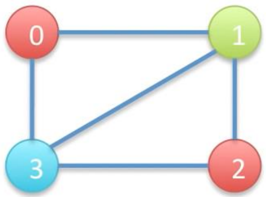
n	0	1	2	3
0	1	1	0	1
1	1	1	1	1
2	0	1	1	1
3	1	1	1	1

```
graphColour(int k){
   for(int c = 1; c<=m; c++){
      if(isSafe(k,c)){
        x[k] = c;
        if((k+1)<n)
           graphColour(k+1);
        else
            print x[]; return;
      }
   }
}
```

k = the node that we're going to colour in this level of the recursion

x[k] = Is an array that holds the current colour at each node.

Example Problem: $n=4$, $m=3$



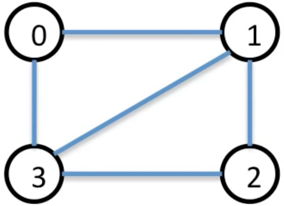
Red = 1
Green = 2
Blue = 3

x[k]

0	1	2	3
1	2	1	3

$x[0] = 1, x[1] = 2, x[2] = 1, x[3] = 3$

Example Problem: n=4, m=3



Adjacency Matrix (G)

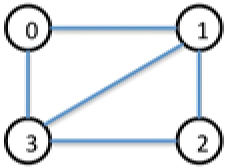
n	0	1	2	3
0	1	1	0	1
1	1	1	1	1
2	0	1	1	1
3	1	1	1	1

```
graphColour(int k){
    for(int c = 1; c<=m; c++){
        if(isSafe(k,c)){
           x[k] = c;
           if((k+1)<n)
               graphColour(k+1);
           else
                print x[]; return;
        }
    }
}
```

```
isSafe(int k, int c){
    for(int i = 0; i<n; i++){
        if(G[k][i] == 1 && c == x[i]{
            return false;
        }
    }
    return true;
}
```

Checks to see if the current colour c is safe to place

Example Problem: n=4, m=3



Adjacency Matrix (G)

n	0	1	2	3
0	1	1	0	1
1	1	1	1	1
	0	1	1	1
	1	1	1	1

```
graphColour(int k){
   for(int c = 1; c<=m; c++){
        if(isSafe(k,c)){
          x[k] = c;
          if(k+1 <n))
             graphColour(k+1);
          else
               print x[]; return;
        }
   }
}
```

graphColour(0);

k = 0

c = 1 (red)

x[k] = 1 (red)

```
isSafe(int k, int c){
   for(int i = 0; i<n; i++){
       if(G[k][i] == 1 && c == x[i]{
          return false;
       }
   }
   return true;
}
```

i = 0

G[0][0] == 1 **Node is adjacent to itself**

c != x[i], 1 !=0

i = 1

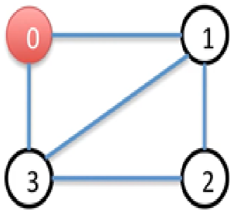
G[0][1] == 1

c != x[i], 1 !=0

Loop continues for all n

Returns true

Example Problem: n=4, m=3



Adjacency Matrix (G)

n	0	1	2	3
0	1	1	0	1
1	1	1	1	1
2	0	1	1	1
3	1	1	1	1

```
graphColour(int k){
   for(int c = 1; c<=m; c++){
       if(isSafe(k,c)){
         x[k] = c;
         if(k+1 <n))
            graphColour(k+1);
         else
              print x[]; return;
       }
   }
}
```

```
isSafe(int k, int c){
   for(int i = 0; i<n; i++){
      if(G[k][i] == 1 && c == x[i]{
         return false;
      }
   }
   return true;
}
```

graphColour(1);

K = 1

c = 1 (red)

c = 2 (green)

x[1] = 2 (green)

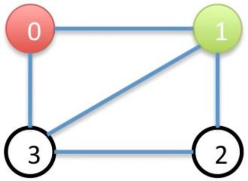
i = 0

G[1][0] == 1

c == x[i], 1 == 1

Returns false

Example Problem: n=4, m=3



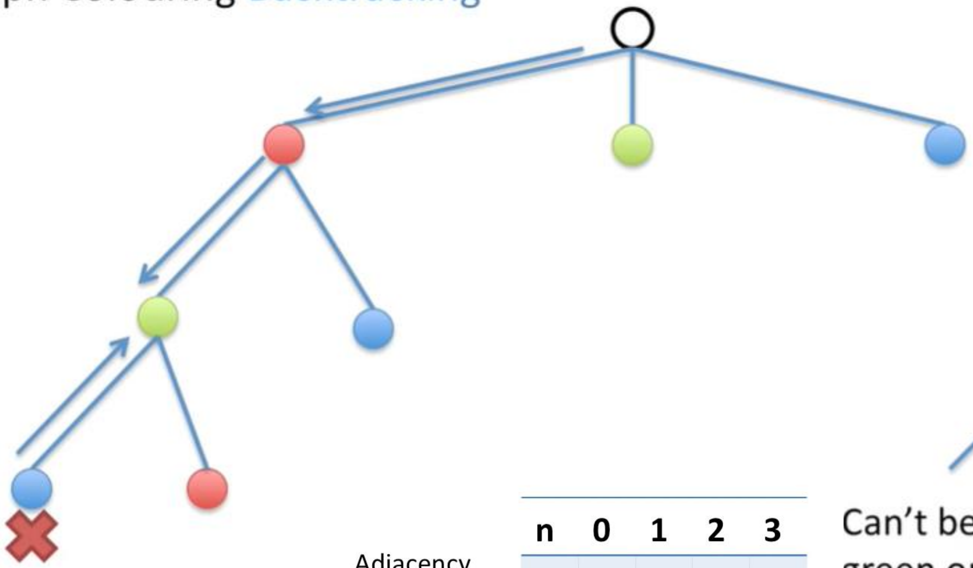
Adjacency Matrix (G)

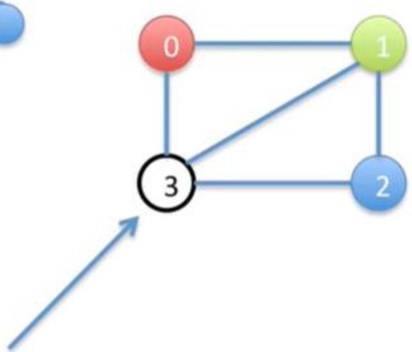
n	0	1	2	3
0	1	1	0	1
1	1	1	1	1
2	0	1	1	1
3	1	1	1	1

The recursion continues for all the nodes in the graph, trying the different colours.

If no colour is safe, and not all nodes are filled, it'll back track and try a different colour on the last node set.

Graph Colouring Backtracking



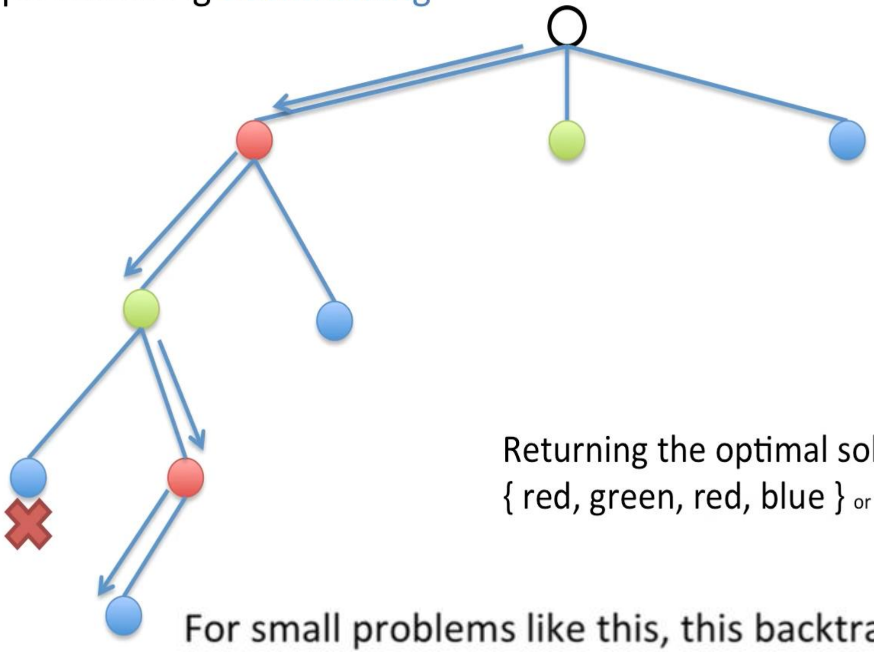


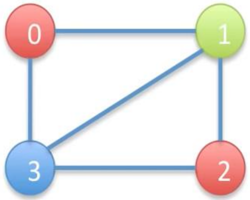
Adjacency Matrix (G)

n	**0**	**1**	**2**	**3**
0	1	1	0	1
1	1	1	1	1
2	0	1	1	1
3	1	1	1	1

Can't be red, green or blue!

Graph Colouring Backtracking





Returning the optimal solution of:
{ red, green, red, blue } or {1,2,1,3}

For small problems like this, this backtracking approach is ok, however the graph colouring problem is a known NP-hard problem

This algorithm is still $O(m^n)$ where m is the number of colours, and n is the number of vertices.

Thanks!

Any questions?

You can find me at @username & user@mail.me