**Shubhan Singh**

**SE-Comps B/Batch C**

**2022300118**

DAA Experiment 7: Backtracking (Graph Coloring)

**Aim** – To implement an algorithm to generate graph colourings using backtracking

**Problem statement:**

Find all possible colourings of the given graph using the minimum number of colours i.e. chromatic number number of colours

**Pseudocode :**

Algorithm Graph-Coloring

Input: A graph represented by an adjacency matrix graph[1..v][1..v]

Output: A coloring of the vertices of the graph

1. chromatic = Find-Ch-Number(graph, v)

2. Initialize an array colors[1..v] with all elements as 0

3. Calculate(graph, v, colors, 0, chromatic)

Function Calculate(graph, v, colors, start, chromatic)

Input: The adjacency matrix graph, the number of vertices v, the colors array, the start index, and the chromatic number

Output: None

1. if start == v

2. Print-Coloring(colors, v)

3. for color = 1 to chromatic

4. if Is-Safe(graph, v, colors, start, color)

5. colors[start] = color

6. Calculate(graph, v, colors, start+1, chromatic)

7. colors[start] = 0

Function Is-Safe(graph, v, colors, start, color)

Input: The adjacency matrix graph, the number of vertices v, the colors array, the start index, and the color

Output: True if it is safe to assign the color to the start vertex, False otherwise

1. for i = 1 to v

2. if graph[start][i] and color == colors[i]

3. return False

4. return True

The algorithm assigns colors to vertices of a graph such that no two adjacent vertices share the same color. The time complexity of this algorithm is O(v^chromatic), where v is the number of vertices in the graph and chromatic is the chromatic number of the graph.

**Source code(C language):**

#include<stdio.h>

#include<stdlib.h>

int count=0;

void printcoloring(int\* colors,int n){

    printf("Coloring %d :\n",count);

    for(int i=0;i<n;i++){

        printf("%c -> %d\t",('a'+i),colors[i]);

        if(i%2==1){

            printf("\n");

        }

    }

    printf("\n");

}

int find(int\* arr, int n, int k){

    for(int i=0;i<n;i++){

        if(arr[i]==k){

            return 1;

        }

    }

    return 0;

}

int mex1(int\* arr,int n){

    for(int i=1;i<=n;i++){

        if(find(arr,n,i)==0){

            return i;

        }

    }

}

int findchno(int\*\* graph, int v){

    int colors[v];

    for(int i=0;i<v;i++){

        colors[i]=0;

    }

    colors[0]=1;

    int rowcolor[v];

    for(int i=1;i<v;i++){

        for(int j=0;j<v;j++){

            if(graph[i][j]==1){

                rowcolor[j]=colors[j];

            }

            else{

                rowcolor[j]=0;

            }

        }

        colors[i]=mex1(rowcolor,v);

    }

    int max=-1;

    for(int i=0;i<v;i++){

        if(colors[i]>max){

            max=colors[i];

        }

    }

    return max;

}

int chk\_valid(int\* row, int n, int\* colors, int c){

    for(int i=0;i<n;i++){

        if(row[i]){

            if(colors[i]==c){

                return 0;

            }

        }

    }

    return 1;

}

void calculate(int\*\* graph, int n, int\* colors,int i, int cno){

    for(int j=0;j<cno;j++){

        if(chk\_valid(graph[i],n,colors,j+1)){

            colors[i]=j+1;

            if(i+1<n)

                calculate(graph,n,colors,i+1,cno);

            else{

                count++;

                printcoloring(colors,n);

            }

        }

    }

    colors[i]=0;

}

int main(){

    int V,E;

    char v1,v2;

    printf("Enter number of vertices: ");

    scanf("%d", &V);

    printf("Enter number of edges: ");

    scanf("%d", &E);

    int \*\*graph = (int \*\*)malloc(V \* sizeof(int \*));

    for(int i=0;i<V;i++){

        graph[i] = (int \*)calloc(V,sizeof(int));

    }

    printf("Enter the edges as vortex1 vortex2, where names of vertices start from 'a': \n");

    for(int i=0;i<E;i++){

        while((getchar()) != '\n');

        scanf("%c %c", &v1, &v2);

        int v1i=v1-'a';

        int v2i=v2-'a';

        graph[v1i][v2i] = 1;

        graph[v2i][v1i] = 1;

    }

    int chromatic=findchno(graph,V);

    printf("\nThe chromatic number of the graph is : %d\n",chromatic);

    printf("All the possible colorings of the graph using %d colors are :\n",chromatic);

    int col[V];

    for(int i=0;i<V;i++){

        col[i]=0;

    }

    calculate(graph,V,col,0,chromatic);

    for(int i=0;i<V;i++){

        free(graph[i]);

    }

    free(graph);

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**A screenshot of a computer

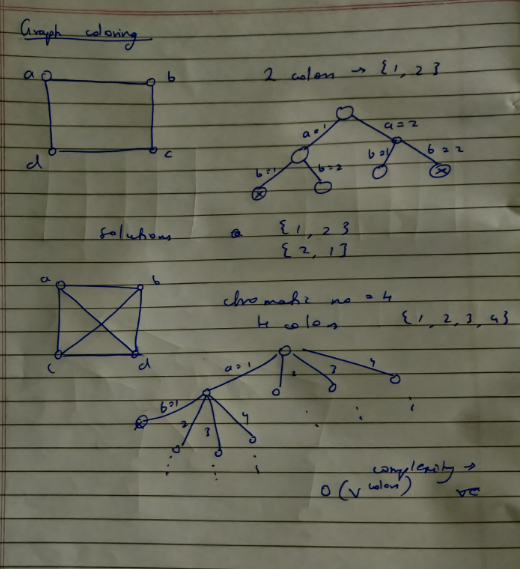
Description automatically generated

(All 24 colorings printed)

**Conclusion:**

* We wrote a program to find all the minimal colorings of a graph i.e. colorings using the minimum possible number of colors.
* The time complexity of this algorithm is O(v^no.of colors) where v is the number of vertices.
* We use backtracking to solve this problem, whenever we find that a certain color is not valid for the vertex in question, we backtrack and choose the next color.
* We find the chromatic number of the graph separately in O(v^2) time.

**Rough Working :**



**Theory:**

1. **Graph Coloring**: Graph coloring is a method of assigning colors to the vertices of a graph in such a way that no two adjacent vertices share the same color. This is a classic computer science problem and has many practical applications, such as in scheduling algorithms, register allocation, and solving puzzles like Sudoku.
2. **Chromatic Number**: The chromatic number of a graph is the smallest number of colors needed to color a graph. It is a fundamental concept in graph theory. For example, a graph is bipartite if and only if it is 2-colorable (i.e., its chromatic number is 2).
3. **Backtracking Algorithms for Graph Coloring**: Backtracking algorithms can be used to solve the graph coloring problem. The idea is to assign colors one by one to different vertices, starting from the first vertex. Before assigning a color, we check for safety by considering already assigned colors to the adjacent vertices. If we find a color assignment which is safe, we mark the color assignment as part of the solution. If we do not find a color due to clashes, we backtrack and return false.
4. **Efficiency**: Using backtracking, the graph coloring problem can be solved more efficiently than a naive approach that tries all possible combinations of colors. However, it is important to note that graph coloring remains a computationally challenging problem. In fact, determining if a graph can be colored with k colors for a given k is a well-known NP-complete problem.
5. **Applications**: Graph coloring algorithms have a wide range of applications in computer science and operations research, including register allocation in compilers, mobile radio frequency assignment, task scheduling, and many others.