Backtracking | Set 5 (m Coloring Problem)

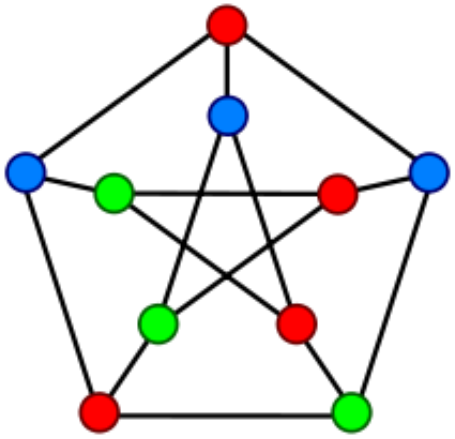
Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with same color. Here coloring of a graph means assignment of colors to all vertices.

Input:
1) A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.
2) An integer m which is maximum number of colors that can be used.

Output:
An array color[V] that should have numbers from 1 to m. color[i] should represent the color assigned to

the ith vertex. The code should also return false if the graph cannot be colored with m colors.

Following is an example graph (from Wiki page) that can be colored with 3 colors.



Naive Algorithm
Generate all possible configurations of colors and print a configuration that satisfies the given constraints.

```
while there are untried conflagrations
{
   generate the next configuration
   if no adjacent vertices are colored with same color
   {
      print this configuration;
   }
}
```

There will be V^m configurations of colors.

Backtracking Algorithm
The idea is to assign colors one by one to different vertices, starting from the vertex 0. 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 solution. If we do not a find color due to clashes then we backtrack and return false.

Implementation of Backtracking solution

```
#include<stdio.h>

// Number of vertices in the graph
#define V 4

void printSolution(int color[]);

/* A utility function to check if the current color assignment
   is safe for vertex v */
bool isSafe (int v, bool graph[V][V], int color[], int c)
{
    for (int i = 0; i < V; i++)
        if (graph[v][i] && c == color[i])
            return false;
    return true;
}
```

```
/* A recursive utility function to solve m coloring problem */
bool graphColoringUtil(bool graph[V][V], int m, int color[], int v)
{
    /* base case: If all vertices are assigned a color then
       return true */
    if (v == V)
        return true;

    /* Consider this vertex v and try different colors */
    for (int c = 1; c <= m; c++)
    {
        /* Check if assignment of color c to v is fine*/
        if (isSafe(v, graph, color, c))
        {
           color[v] = c;

           /* recur to assign colors to rest of the vertices */
           if (graphColoringUtil (graph, m, color, v+1) == true)
             return true;

            /* If assigning color c doesn't lead to a solution
               then remove it */
           color[v] = 0;
        }
    }

    /* If no color can be assigned to this vertex then return false */
    return false;
}

/* This function solves the m Coloring problem using Backtracking.
  It mainly uses graphColoringUtil() to solve the problem. It returns
  false if the m colors cannot be assigned, otherwise return true and
  prints assignments of colors to all vertices. Please note that there
  may be more than one solutions, this function prints one of the
  feasible solutions.*/
bool graphColoring(bool graph[V][V], int m)
{
    // Initialize all color values as 0. This initialization is needed
    // correct functioning of isSafe()
    int *color = new int[V];
    for (int i = 0; i < V; i++)
       color[i] = 0;

    // Call graphColoringUtil() for vertex 0
    if (graphColoringUtil(graph, m, color, 0) == false)
    {
      printf("Solution does not exist");
      return false;
    }

    // Print the solution
    printSolution(color);
    return true;
}

/* A utility function to print solution */
void printSolution(int color[])
{
    printf("Solution Exists:"
```

```
            " Following are the assigned colors \n");
    for (int i = 0; i < V; i++)
      printf(" %d ", color[i]);
    printf("\n");
}

// driver program to test above function
int main()
{
    /* Create following graph and test whether it is 3 colorable
      (3)---(2)
       |   / |
       |  /  |
       | /   |
      (0)---(1)
    */
    bool graph[V][V] = {{0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0},
    };
    int m = 3; // Number of colors
    graphColoring (graph, m);
    return 0;
}
```

Output:

```
Solution Exists: Following are the assigned colors
 1  2  3  2
```

References:
http://en.wikipedia.org/wiki/Graph_coloring

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Backtracking

Writing code in comment? Please use **ideone.com** and share the link here.

Recommend Share Sort by Newest

Join the discussion…

sans • 16 days ago
WTF is BOOL?
• Reply • Share ›

helper • 6 months ago
1. isSafe: indicates whether color c is safe for vertex v according to graph and color info.
2. graphColoringUtil: it asks its children whether a solution exists and uses another delegate function printSolution to print, if it gets a solution. but it can delegate that printing responsiblity too to its child.
• Reply • Share ›

shishir dwivedi • 7 months ago
mujhe iss question se pyaar ho gya hai...
• Reply • Share ›

shishir dwivedi • 7 months ago
mujhe ye ques bohot achcha lagta hai :D
• Reply • Share ›

rakesh shishir dwivedi • 4 months ago
Bhosadi wale .. teri amma ko chodu .. madarchod .. ja is question se apni gaand mara le. ha ha ha
• Reply • Share ›

Shaili • 8 months ago
@GeeksforGeeks

I know the time complexity of solving it with naive approach is V^m as stated above. But I need to know the time complexity of solving this problem by Backtracking approach? Please guide.
• Reply • Share ›

Guest • 8 months ago
@GeeksforGeeks
I know the time complexity of solving it with naive approach is V^m as stated above. But I need to know the time complexity of solving this problem by Backtracking approach? Please guide.
• Reply • Share ›

Uzma Khan • 8 months ago

Uzma Khan • 8 months ago

In graphColoringUtil() shouldn't there be a c--; after the color[v]=0; ? that is if the current color doesn't lead to a solution down the road, we back track and try the previous color?

 • Reply • Share ›

dheeraj • 8 months ago

http://ideone.com/R4qXVd

 • Reply • Share ›

dheeraj • 8 months ago

```
int check_for_safe_color(int color[], int ver_index, int adj[], int total_vert)

{

int x, flag = 0;

for(x = 0; x < total_vert; x++)

{

if(adj[x] != 0)

{

if(color[ver_index] != color[x])

flag = 1;

else

{
```

see more

 • Reply • Share ›

sr7 • a year ago

you can further bring down the complexity by avoiding the check for each color . Since the no of colors can go upto N and no of vertices is N .

Checking for each color will take O(n^2) time . Instead

I assume the base color (uncolored node) is -1 . Colors start from 0.

I will check all the adjacent nodes of the given vertex in O(n) time .

If i th node is adjacent to vertex v and color[i]>=0 (colored node) , then temp[color[i]] = 1 (temp[i] if set , indicates that the color is in use by one of its adjacent nodes) , finally I will search the 'temp' array starting from i =0,

if temp[i]!=1, then that is the least color not used by any of the adjacent nodes, then color[v] is made i ; It takes O(n) on the whole, instead of O(n^2).

int temp[V];

//Initialize all temp values to zero

for (i=0;i<v;i++) temp[i]="0;" for="" (i="0;i<V;i++)" if="" (graph[v][i]="" &&="" color[i]="">=0) //Node i is adjacent and colored

temp[color[i]]=1;

for (i=0; i<m ;="" i++)="" {="" only="" 0="" to="" m-1="" colors="" are="" allowed="" if="" (!temp[i])="" {="" color[v]="i;" if="" (graphcoloringutil(graph,m,color,v+1))="" return="" true;="" color[v]="-1;" backtrack="" }="" return="" true;="">

• Reply • Share ›

pavi.8081 • 2 years ago

Can we modify this algorithm to find the min. number of colors to color the graph nodes. I think we can but haven't tried though.

If we can do so then the modified algorithm can be used to solve problems like: find the minimum number of party halls required to organize parties with given starting and finishing times.

Please comment...

• Reply • Share ›

soupboy • 3 years ago

Hi, shouldn't the brute force algorithm's complexity be m^V and not V^m as written above ?

• Reply • Share ›

Bhavik ➚ soupboy • 6 months ago

Yes, I think you are right as we can convert the logic into following recurrence
T(i)=m*T(i-1)+m
and if you put i =1 then T(1)=m
T(2)=m^2
T(3)=m^3
and this can be up to V so T(V)=m^V

• Reply • Share ›

coolguy ➚ soupboy • 3 years ago

You can understand graph coloring using backtracking by watching this video
youtu.be/CI3A_9hokjU

1 • Reply • Share ›

Rohit Raj • 3 years ago

bool isSafe (int v, bool graph[V][V], int color[], int c)
{
for (int i = 0; i < V; i++)
if (graph[v][i] && c == color[i])
return false;
return true;
}

as the problem statement says that if (i == j) then also the graph[i][j] = 1; so shouldn&#039t we check for this condition in this function?
i.e if(graph[v][i] && c == color[i] && v != i).

1 ▲ | ▼ • Reply • Share ›

kartik ➚ Rohit Raj • 3 years ago

@Rohit Raj

We don't need to check for this as we assign a color c only after isSafe() returns true. Before calling isSafe(), we assign the color value as 0 which is not a valid value. When we call isSafe(), the value of c will be greater than 0 and value of color[v] will be 0. So color[v] will never be equal to c. I hope this clarifies your doubt.

▲ | ▼ • Reply • Share ›

gautam • 3 years ago

Can be use some dynamic programming to reduce complexity

```
/* Paste your code here (You may delete these lines if not writing code) */
```

▲ | ▼ • Reply • Share ›

Venki • 3 years ago

There is small bug in the code. The colour vector should be of size V in lieu of m. In the current code, the program tries to access elements past the buffer end (while checking validity of colour assignment).

What is the worst case complexity? O(V * m V) as we are exploring all configurations and doing O(V) work at each call.

▲ | ▼ • Reply • Share ›

GeeksforGeeks ➚ Venki • 3 years ago

@Venki: Thanks for pointing this out. This has been fixed. Yes, the worst case complexity will be O(V * (m^V)) as we may have to try all configurations in worst case.

▲ | ▼ • Reply • Share ›

Guddu sharma • 3 years ago

What is the need of following if condition:
if (graphColoringUtil (graph, m, color, v+1) == true)
return true;

I think when all the vertices have been assigned color,the first condition if(v==V) return true,which is sufficient.

Please comment if i have understood something wrong..

• Reply • Share ›

Venki ➚ Guddu sharma • 3 years ago

@Guddu sharma, We need to find *one* feasible solution. The condition if(v==V) is recursion base case to end infinite recursion. When we reach the end branch in the state space tree and still not found solution, we need to backtrack to one level upper and restore the state.

Similarly, if the current assignment is safe, (means the branch generated in the tree can be part of solution state), we recur to next level for next feasible color.

The condition

```
 /* recur to assign colors to rest of the vertices */
if (graphColoringUtil (graph, m, color, v+1) == true)
    return true;
```

checks next color. If this next color reaches last color, we are done which will be returned at the if condition

```
if (v == V)
        return true;
```

Try to draw the state space tree for small values of V and m, you can easily understand the backtracking procedure.

• Reply • Share ›

kartik ➚ Guddu sharma • 3 years ago

@Guddu sharma: The statement serves two purposes:
1) It calls the function recursively for the other vertices.
2) It returns true as soon as one of the color assignments lead to a solution. So that other colors are not checked as soon as a color assignment works.

Also, just putting following line is not sufficient. We need to forward the result of this function call to parent calls.

```
    if (v == V)
       return true;
```


- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string

- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

- Follow @GeeksforGeeks

Recent Comments

- Ashish Aggarwal

  Try Data Structures and Algorithms Made Easy -...

  Algorithms · 17 minutes ago

- Vlad

  Thanks. Very interesting lectures.

  Expected Number of Trials until Success · 1 hour ago

- cfh

  My implementation which prints the index of the...

  Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

- Gaurav pruthi

  forgot to see that part ;)

  Bloomberg Interview | Set 1 (Phone Interview) · 1 hour ago

- saeid aslami

  thanks

  Greedy Algorithms | Set 7 (Dijkstra’s shortest path algorithm) · 1 hour ago

- Cracker

  Implementation:...

  Implement Stack using Queues · 2 hours ago