T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE2046 - Analysis of Algorithms

Homework #2 Report

Group Members

150119639 - Erdem PEHLİVANLAR

150120056 - Haldun Halil OLCAY

170517033 - Yasin Alper BİNGÜL

GRAPH COLORING PROBLEM

In this experiment, we developed a unique algorithm by ourselves which solves the graph coloring problem. We tried different algorithms like the most common greedy ones; Welsh Powell, Brook’s, Backtracking, Recursive Largest First, Dsatur etc. We examined their algorithm logics, how they handle the inputs and how they give color to the vertices. Thus, we finally had an intuition about how to make a better algorithm on the graph coloring problem.

In our graph coloring approach, we realized that different algorithms give different results according to the construct of the graph. So we developed four different sub coloring algorithms under one main program. Our program takes the input, evaluates it with these four algorithms independently, takes the best result and writes it as an output file.

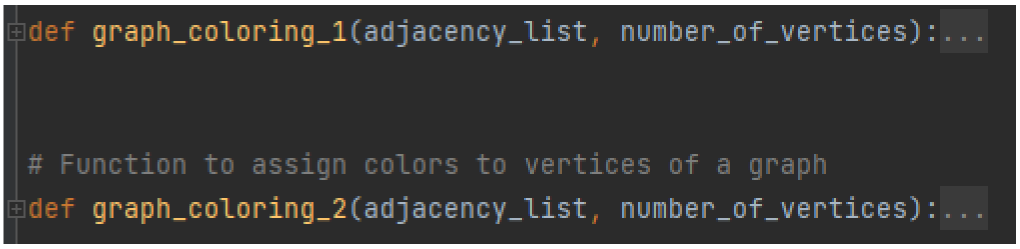
DESIGNING THE ALGORITHM:

-First Approach

At the beginning, the algorithm takes the input file and stores the vertex information into a list called ***input_list*** with ***addEdge()*** function. This list has the information of the adjacencies of vertices. Before applying the coloring algorithm, it sorts the input_list in decreasing order. After this process, we noticed that some index problems occurred in the new sorted input_list (containing adjacency information). Because while sorting the list, the index of every vertex and its adjacent vertices changes. It behaves in a different way when coloring vertices. Because of this issue, we added an algorithm to fix it. ***fixGraphList()*** function checks and fixes the adjacency list according to the original adjacency list.

After the pre-sorting methods are done, the adjacency list is ready for the coloring algorithm. In the ***graph_coloring_1()*** function, we have a dictionary that stores the assigned colors for each vertex called ***color_dict{}***. Also we have a list, ***assigned_colors[],*** for the colors of the adjacent vertices in order to check the colors which are already given .

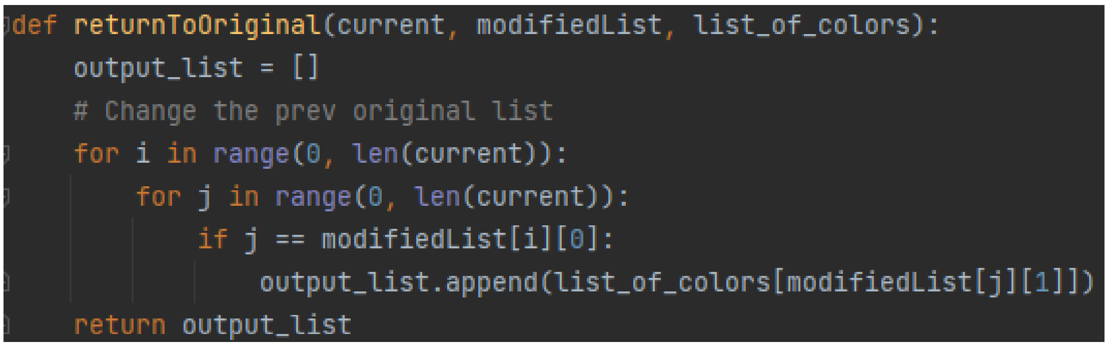
This list is updated on each iteration for coloring each vertex. Every iteration, the algorithm checks the colors of adjacent vertices and assigns the first available color for the current vertex. This process continues until there isn’t any uncolored vertex. After coloring, the function returns the dictionary ***color_dict{}*** that has the assigned colors for each vertex.



```
def graph_coloring_1(adjacency_list, number_of_vertices):...


# Function to assign colors to vertices of a graph
def graph_coloring_2(adjacency_list, number_of_vertices):...
```

After the coloring process, the list which has the assigned colors has different vertex indexes. This issue occurred due to the sorting process. For this reason, the algorithm converts that list index into the original index with help of ***returnToOriginal()*** function. This function reorders the final output list according to the input file and returns the updated list, ***output_list[]***.



```
def returnToOriginal(current, modifiedList, list_of_colors):
    output_list = []
    # Change the prev original list
    for i in range(0, len(current)):
        for j in range(0, len(current)):
            if j == modifiedList[i][0]:
                output_list.append(list_of_colors[modifiedList[j][1]])
    return output_list
```

-Second Approach:

After the pre-sorting methods are done which are the same as the first approach, the adjacency list is ready for the coloring algorithm. In the ***graph_coloring_2()*** function, we have a list of assigned colors of vertices, ***color_list[]***. At the beginning of the function, the algorithm assigns “-1” to the assigned color of the vertices list. Thus,the algorithm can detect whether the vertex is already colored or not. Also we have a list, ***available[]*** that has information about the availability of coloring.

Firstly, in every iteration the algorithm checks the assigned colors and determines the updated ***available[]*** list according to the assigned colors. If there is any “-1” in the ***color_list[]*** which means that the vertex is not colored, then the coloring availability for this iteration will be true. After completing the ***available[]*** list information, the algorithm uses that list while assigning colors to find the first available color. It considers all of the colors of adjacency vertices, finds the best color and stores them in the ***color_list[]***. This process continues until there isn’t any uncolored vertex.

-Third Approach:

First of all, the algorithm takes the input file and stores the vertex information into a dictionary like ***{vertex_number: vertex_object}***. As it seems, the algorithm uses object oriented programming and the program has two classes named as ***Vertex*** and ***Management***.

Algorithm reads the problem before starting assigning color and determines the problem char(‘p’ or ‘e’), number of vertices and number of edges in the ***read_problem()*** function. Then, it creates an object of vertices and stores it in a dictionary called ***self.id_vertex*** in descending order with respect to the degree of vertices called ***self.degree*** in ***create_vertices()*** function. Now, everything is ready to assign colors to the vertices. Basically, the algorithm iterates every vertex with descending order according to the degree of the vertices and while iterating, it gives the minimum color code it can give to the current vertex, then it iterates all neighbor vertices called ***self.adjacency*** of vertex object in ***Vertex*** class and still, it gives the minimum color code it can give to the current neighbor vertex in ***graph_coloring_3()*** function. This process continues up until the iterator comes to the end in ***self.id_vertex*** dictionary.

Finally, the algorithm creates the output file in the ***show_result()*** function.

-Fourth Approach:

This algorithm also has object oriented programming and there is only one difference between the third and fourth approach. That is ***graph_coloring_4()*** function. In this function, the algorithm iterates every vertex with descending order according to the degree of vertices. While iterating, it gives the minimum color code that can give to the current vertex. After that, it iterates all vertices except for the current vertex and adjacencies of the current vertex. Then it gives the same color code with the current vertex's color to all other vertices if it can give. If it can not give the same color as the current vertex's color, it passes this vertex without painting. This process continues up until the iterator comes to the end in ***self.id_vertex*** dictionary.

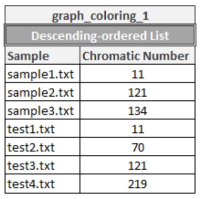
Finally, the algorithm creates the output file in the ***show_result()*** function.

-Main Function:

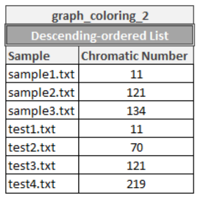
In the main function, our program takes results of all four algorithms and chooses the best result which is called chromatic number (χ(G)) and writes it as an output file.

CONCLUSION:

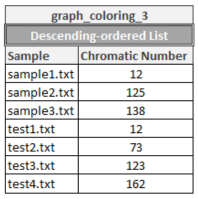
In this homework, we examined different algorithms. We searched for an efficient way to color the graph with a minimum number of colors called chromatic number. After implementing our algorithm we tried it with the given samples. Also we sorted them to ascending or descending orders by means of the number of their adjacent vertices. With coloring the vertices in descending order we get the best result. You can see the results down below.



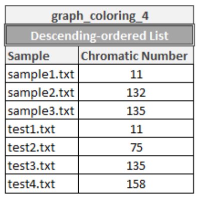
graph_coloring_1	
Descending-ordered List	
Sample	Chromatic Number
sample1.txt	11
sample2.txt	121
sample3.txt	134
test1.txt	11
test2.txt	70
test3.txt	121
test4.txt	219



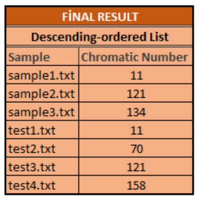
graph_coloring_2	
Descending-ordered List	
Sample	Chromatic Number
sample1.txt	11
sample2.txt	121
sample3.txt	134
test1.txt	11
test2.txt	70
test3.txt	121
test4.txt	219



graph_coloring_3	
Descending-ordered List	
Sample	Chromatic Number
sample1.txt	12
sample2.txt	125
sample3.txt	138
test1.txt	12
test2.txt	73
test3.txt	123
test4.txt	162



graph_coloring_4	
Descending-ordered List	
Sample	Chromatic Number
sample1.txt	11
sample2.txt	132
sample3.txt	135
test1.txt	11
test2.txt	75
test3.txt	135
test4.txt	158



FİNAL RESULT	
Descending-ordered List	
Sample	Chromatic Number
sample1.txt	11
sample2.txt	121
sample3.txt	134
test1.txt	11
test2.txt	70
test3.txt	121
test4.txt	158

We also found that different algorithms show different performance with different input files.

Who did what ? :

As a group we all worked together. While implementing the codes and writing the report we contributed equally.

REFERENCES:

- **https://en.wikipedia.org/wiki/Graph_coloring**
- **Introduction to the Design and Analysis of Algorithms - Levitin, Anany.**
- **https://www.researchgate.net/publication/332366008_Exact_Algorithms_for_the_Graph_Coloring_Problem**
- **https://dergipark.org.tr/en/download/article-file/254140**
- **https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/**
- **https://www.interviewbit.com/blog/graph-coloring-problem/**