In the name of God
algorithm project report
number 31

Amir Mohamamd Pir Hossein Lou
9531014

Subject : Define an algorithm for optimal(min-coloring) coloring a graph in polynomial time.

Introduction :
By an article from www.geeksforgeeks.com , there is no polynomial time algorithm for min-coloring a graph . But there are some algorithms for min-coloring special types of graph like bipartite graph , planar graph , complete graph , regular graph and … .

for example:

For bipartite graphs , Cole, Ost & Schirra (2001) showed that an optimal edge coloring of these graphs can be found in the near-linear time bound O(*m* log Δ), where *m* is the number of edges in the graph.

Again for bipartite graphs , Alon (2003) algorithm solves problem in polynomial time .Alon (2003) begins by making the input graph regular, without increasing its degree or significantly increasing its size, by merging pairs of vertices that belong to the same side of the bipartition and then adding a small number of additional vertices and edges. Then, if the degree is odd, Alon finds a single perfect matching in near-linear time, assigns it a color, and removes it from the graph, causing the degree to become even. Finally, Alon applies an observation of Gabow (1976), that selecting alternating subsets of edges in an Euler tour of the graph partitions it into two regular subgraphs, to split the edge coloring problem into two smaller subproblems, and his algorithm solves the two subproblems recursively. The total time for his algorithm is O(*m* log *m*).

For planar graphs with maximum degree Δ ≥ 7, the optimal number of colors is again exactly Δ. With the stronger assumption that Δ ≥ 9, it is possible to find an optimal edge coloring in linear time (Cole & Kowalik 2008).

For d-regular graphs which are pseudo-random in the sense that their adjacency matrix has second largest eigenvalue (in absolute value) at most $d^{1-\varepsilon}$, d is the optimal number of colors (Ferber & Jain 2018).

as mentioned above, there are such algorithms which solve this problem in polynomial time for special graphs like bipartite , … . one of theses special graphs is complete graph.

A complete graph, K_N , is defined as a collection of N vertices and an edge from each vertex to every other vertex. In other words, each distinct pair of vertices in the graph has an edge connecting them. For any complete graph K_N , Δ = N – 1. Similar to cycles, if N is even, the graph is Class 1 and can be edge-colored with N – 1 colors. If N is odd, the graph is Class 2 and needs N colors .this can be proven with induction using the base cases of K_2 and K_3 . A K_2 trivially only needs one color since the graph has one edge and a K_3 needs three colors since each edge is adjacent to the other two.
Every complete graph K_N can be visualized as a K_{N-1} with an extra vertex on the outside and edges

connecting the outer vertex to every vertex in the original K_{N-1} . this will be demonstrated visually in the algorithm below. If K_{N-1} is Class 2, then each vertex will be missing a different color and that color can be used to color each edge to the new vertex for K_N . A simple algorithm to color a complete graph, K_N , is listed below :

Edge coloring algorithm:

I have implemented an algorithm which colors complete graphs in time complexity $O(E) = O(V^2)$.

Steps of algorithm:

N is degree of graph
If N is odd:
1) Arrange the vertices as an n-polygon and color each edge with a different color.

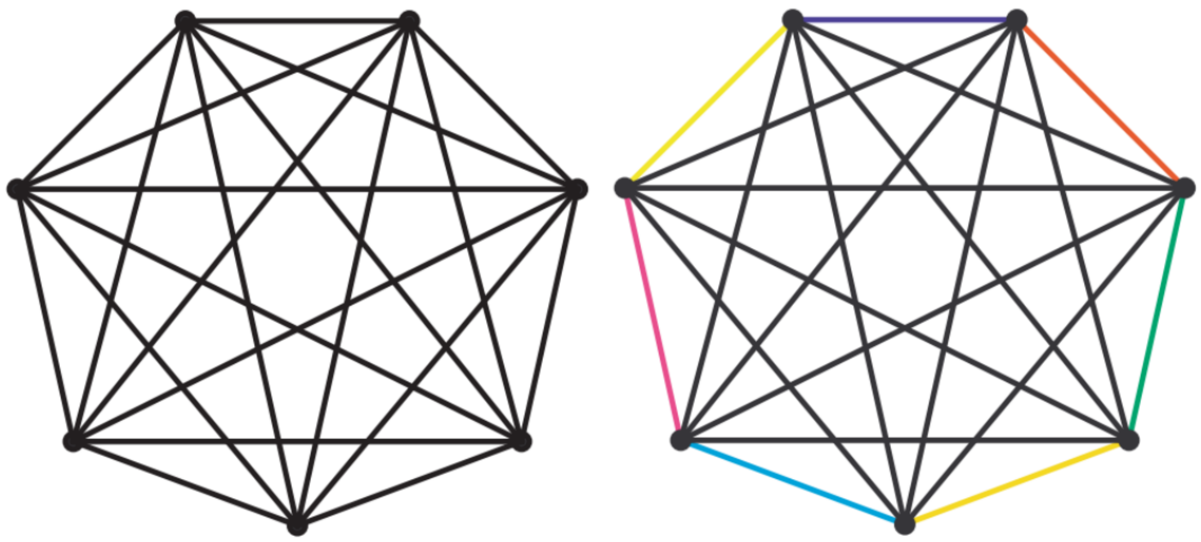


Figure 5: Coloring the outer edges of a K_7 graph.

2) For each outer edge, color its “inner” parallel edges the same color. Repeat for each outer edge.

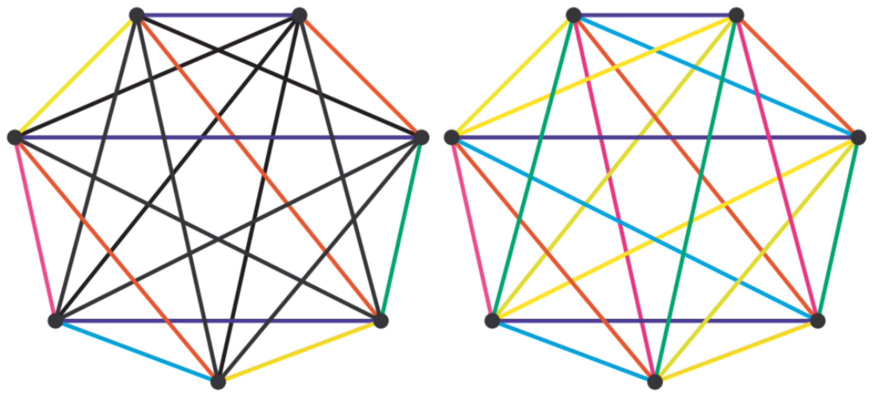


Figure 6: Coloring the first set of parallel edges red and blue (left) and the complete K_7 (right).

To clarify Step 2, the horizontal edge between the top vertices is assigned blue in Figure 6. The next lower horizontal edge is colored blue (the vertices adjacent to the initial pair) and so forth. The same parallel coloring is done for each outer edge of the polygon. This takes N colors and since $\Delta = N - 1$, this graph is Class 2.

If N is even:
1) Color a K_{N-1} using the algorithm shown above (N – 1 will be odd).

2) Add a new outer vertex and an edge between it and each vertex in K_{N-1}.
3) Color each edge with the missing color from its connected K_{N-1} vertex.

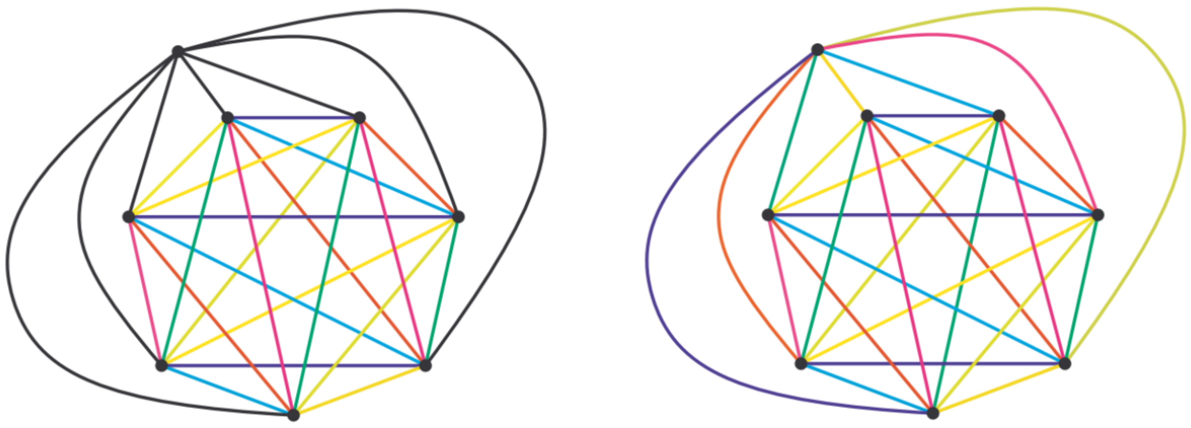
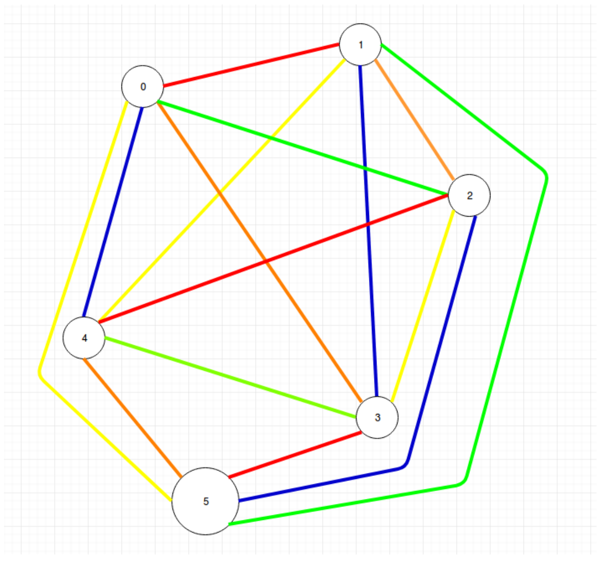


Figure 7: Coloring a K_8 using the inner K_7 from Figure 6.

Each vertex in the inner K_{N-1} has a missing color because its edge chromatic number is $\Delta + 1$ but each vertex has degree Δ. By introducing the new vertex for K_N , a different color can be used for each edge and satisfy the edge coloring conditions.

Time complexity of algorithm is O(E) because each edge is processes only once.

Output for edge coloring of a 5-complete graph with degree :
red = 1 , orange = 2 , yellow = 3 , green = 4 , blue = 5
source,destination : color



```
0,0 : 0
0,1 : 1
0,2 : 4
0,3 : 2
0,4 : 5
0,5 : 3
1,0 : 1
1,1 : 0
1,2 : 2
1,3 : 5
1,4 : 3
1,5 : 4
2,0 : 4
2,1 : 2
2,2 : 0
2,3 : 3
2,4 : 1
2,5 : 5
3,0 : 2
3,1 : 5
3,2 : 3
3,3 : 0
3,4 : 4
3,5 : 1
4,0 : 5
4,1 : 3
```

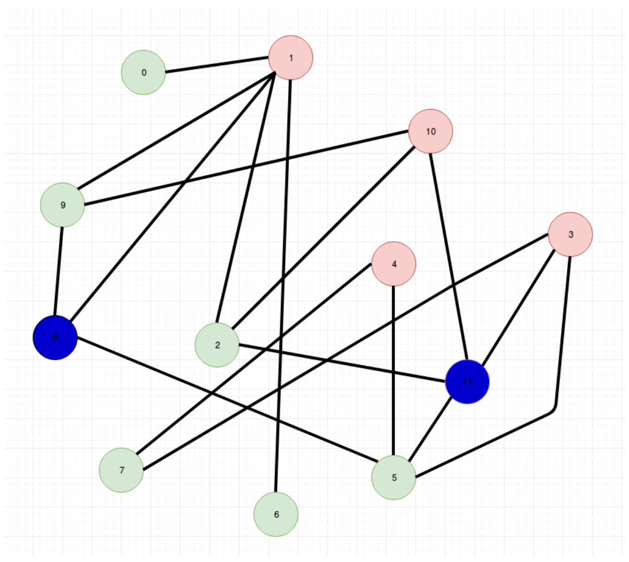
4,2 : 1
4,3 : 4
4,4 : 0
4,5 : 2
5,0 : 3
5,1 : 4
5,2 : 5
5,3 : 1
5,4 : 2
5,5 : 0

Vertex coloring algorithm:

I have implemented Welsh–Powell Algorithm for vertex coloring .It may not always give the best solution, but it usually performs better than just coloring the vertices without a plan will.

Steps of algorithm:
1- find the degree for each vertex.
2- list the vertices in order of descending valence(degree) .
3- color the first vertex in the list which is the highest valence with color 1.
4- go down the list and color every vertex not connected to the colored vertices above the same color. Then cross out all colored vertices in the list.
5- repeat the process on the uncolored vertices with a new color – always working in descending order of valence until all the vertices have been colored.

Output for vertex coloring of below graph :
vertex : color
0 : 2
1 : 1
2 : 2
3 : 1
4 : 1
5 : 2
6 : 2
7 : 2
8 : 3
9 : 2
10 : 1
11 : 3



In worst case, time complexity of this algorithm is O(V^3), because in worst case we visit all vertices, then for each vertex, we visit vertices which are not adjacent to it and not have being colored. If we reach a vertex like that, again we visit its neighbors to check that is it possible to color it with the desired color or not. Thus time complexity is O(V^3).

References :

http://mrsleblancsmath.pbworks.com/w/file/fetch/46119304/vertex coloring algorithm.pdf
https://en.wikipedia.org/wiki/Edge_coloring
http://www.eecs.tufts.edu/~tho01/edgecoloringreport_tho.pdf