In the name of God
algorithm project report
number 31
Amir Mohammad Pir Hossein Lou
9531014

Subject:

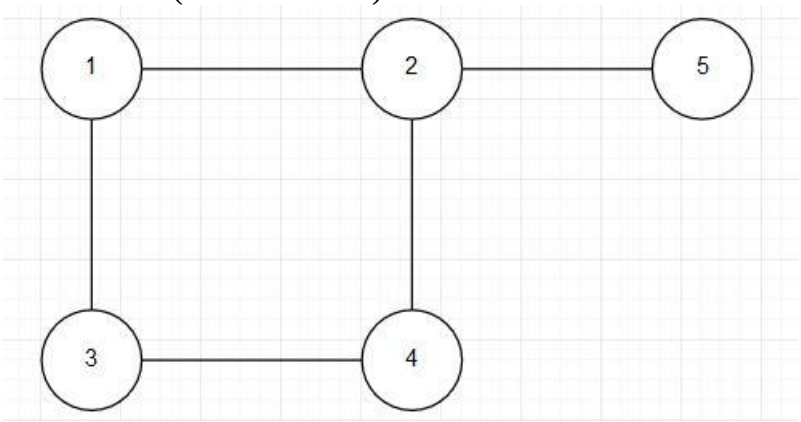
Define an algorithm for optimal(min-coloring) coloring a graph in polynomial time.

coloring method description:

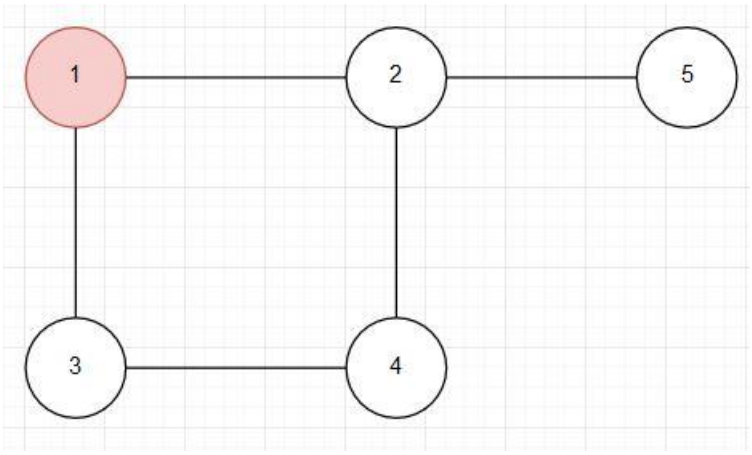
1- **Vertex coloring:**

At first, pick a vertex randomly and paint it, now you cannot paint its neighbors. Select one of the vertices which is not in set containing neighbors and paint it with same color. now you can pick vertices which are not adjacent to two previous vertices which selected and painted. Pick one of them and paint it with same color. Repeat this process until you could not pick any more vertices. now choose a vertex randomly which is not painted yet and paint it with new color. Continue in same manner until no vertex is remained.
Note that you can choose a vertex in above steps while it is not painted.

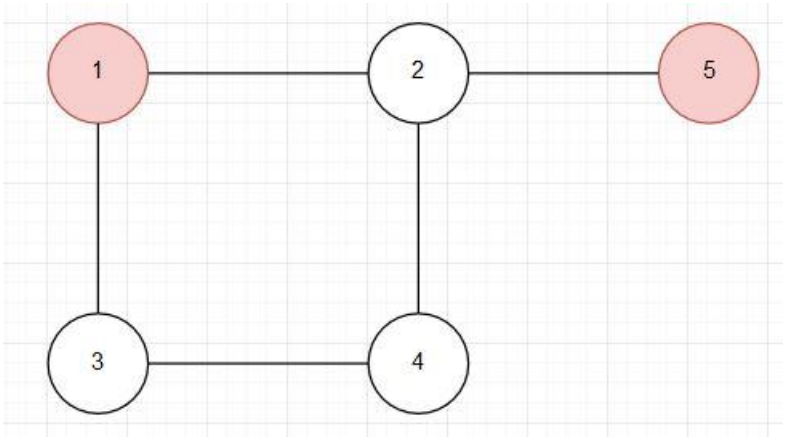
Example:
Pick vertex 1(random choice):



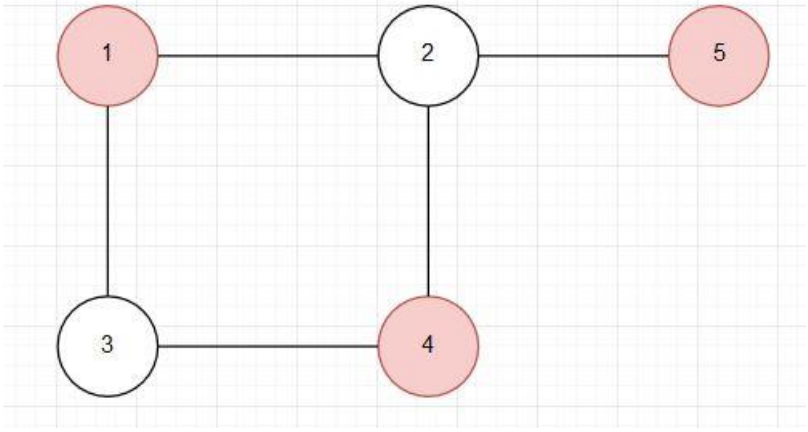
Paint it with red:



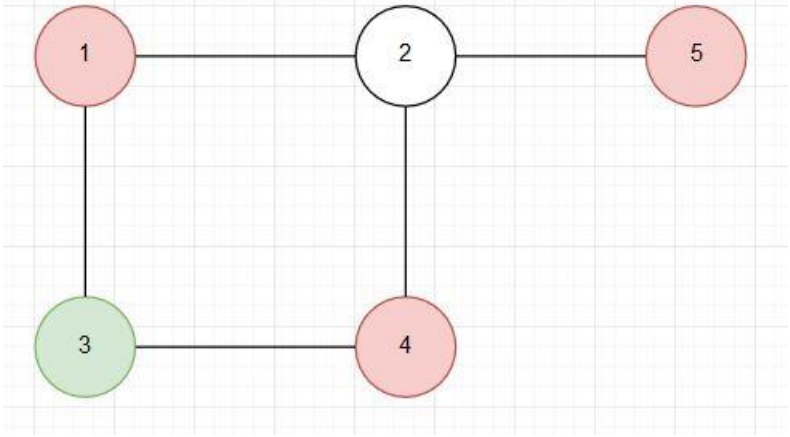
Now we can choose vertex 4 or 5.lets choose vertex 5 and paint it.



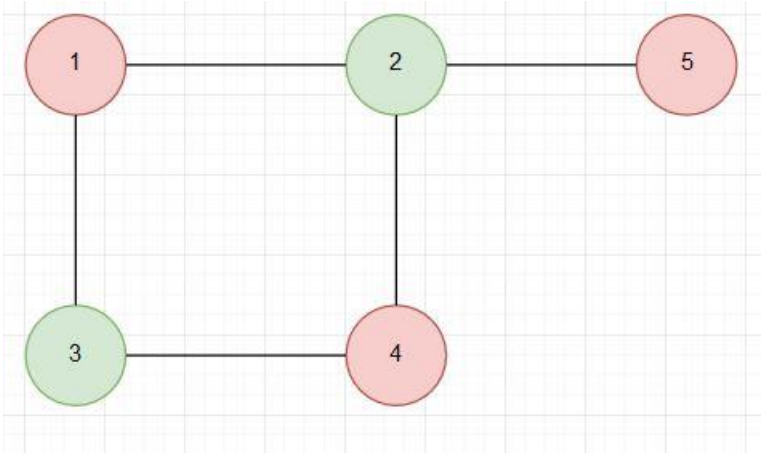
Now we can just choose vertex 4.so pick it and paint it with red.



With selected vertices, there is no vertices which can be paint with red.so choose vertex 2 or 3 (random, here we choose vertex 2) and paint it with new color (here green).



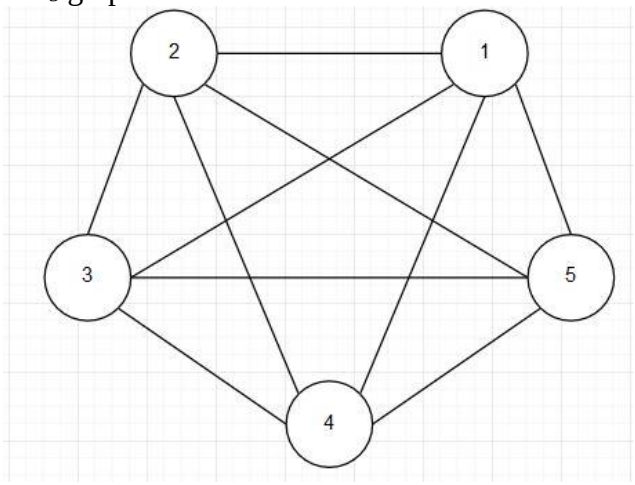
Vertex 2 is not adjacent to vertex 3 and is not colored yet.so selected vertex 2 and color it with green.



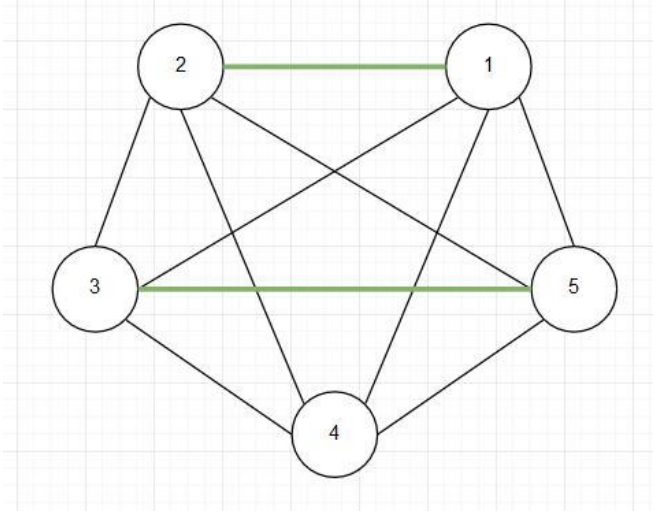
2- Edge coloring:

My method for edge coloring in optimal mode only works for complete graphs.
At first pick an edge and paint it with a color, then paint all edges which are parallel to colored edge with same color. Now pick another edge which is not colored yet and do same process for it. Continue in this manner until all edges being colored.

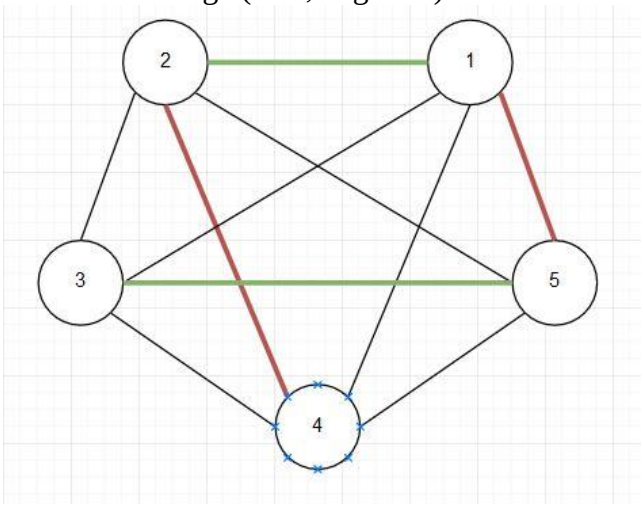
Example:
A k_5 graph:



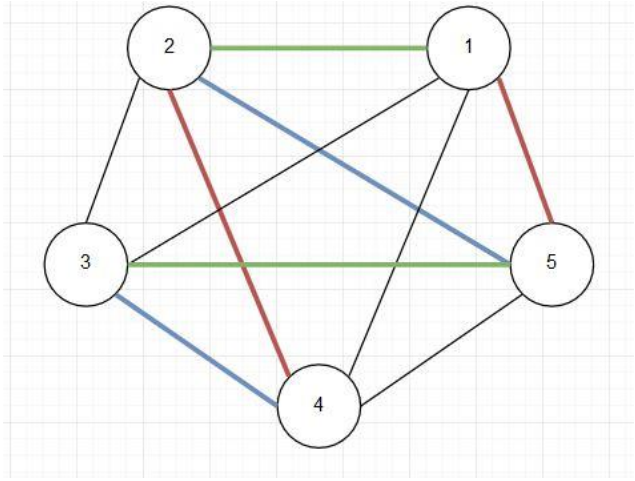
Pick edge 2-1 and color it with green. Then color all parallel edges to it with green.

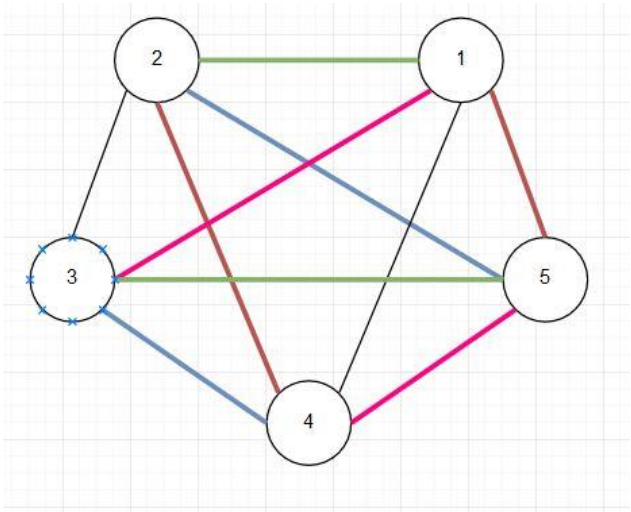


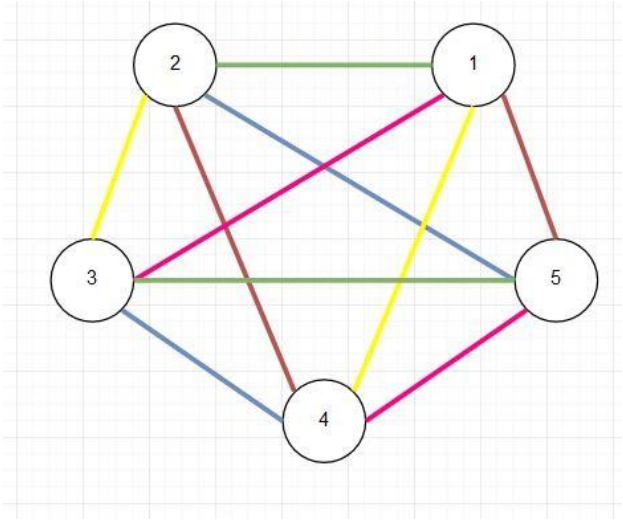
Pick another edge (here, edge 5-1). Color it and all parallel edges to it with red.



In the following:







Limitations:

In edge coloring, method works only for complete graphs.

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, $\Delta = N - 1$. 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.

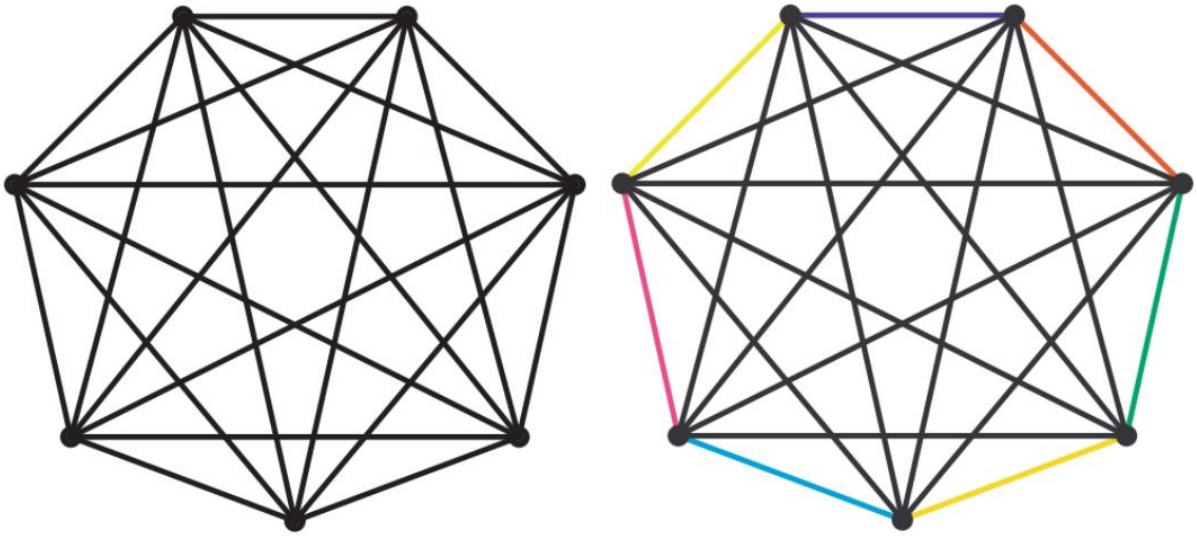


Figure 5: Coloring the outer edges of a K_7 graph.

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

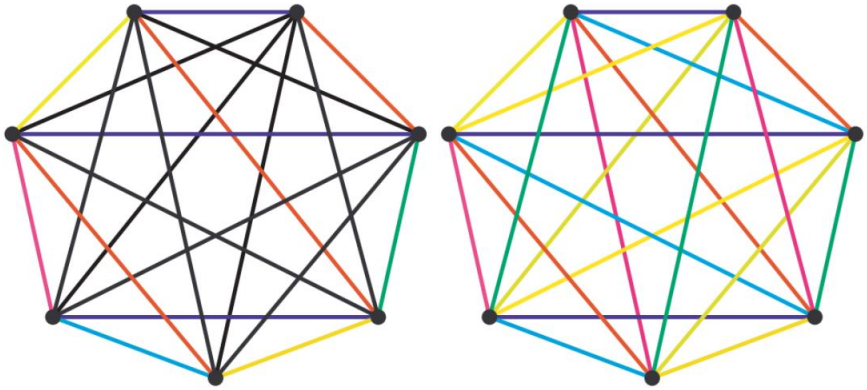


Figure 6: Coloring the first set of parallel edges red and blue (left) and the complete K_7 (right).

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.

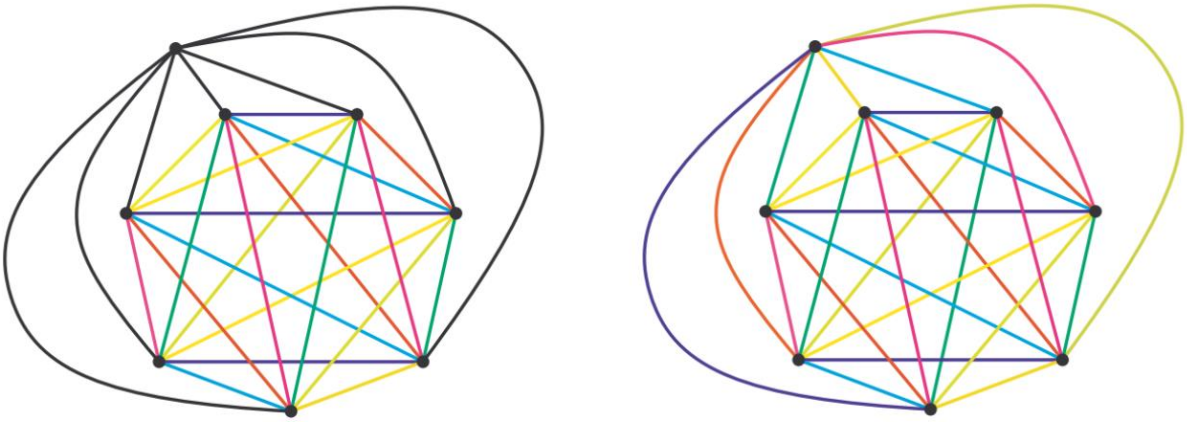


Figure 7: Coloring a K_8 using the inner K_7 from Figure 6.

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.

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.

Limitations of algorithms:

Vertex coloring:

No limitation, it works very good for all type of graphs but sometimes its coloring is not optimal. Note that it works only for undirected graphs.

Edge coloring:

It only works for complete graphs.

Time complexity of algorithms:

Vertex coloring algorithm:

Its time complexity is $O(V^3)$ because of 3 for loop in code.

Edge coloring algorithm:

Its time complexity is $O(E) = O(V^2)$ because each edge is processed only once. Number of edges in complete graph is V^2. So $O(E) = O(V^2)$.

Input and Output:

Input is a file with .csv format. Pass the address of files as the argument of main class.
Each line in input must contain 2 number which are separated with comma. First number is source of an edge and second one is destination.
Note: for edge coloring, there is no need to input file, just need to K, the degree of graph. You can set the degree in code in line 6 in main class.
Output is a file named 'output'.
For vertex coloring, each line in output contains 2 numbers. First number is vertex number and second one is its color.
For edge coloring, each line in output contains 3 numbers. In other words, each line contains an edge and corresponding color. First number is source of edge and second one is destination and third one is color of the edge.
Note: for the convenience of work, at first the code converts the vertices numbers of graph to continuous numbers. For example:
Input:
1,2
5,6
95,2
6,36
135,95
95,1024
The input will be converted to:
1,2
5,6
4,2
6,7
3,4
4,0
This conversion does not affect in algorithm and output. Just number of vertices are changed and the graph structure will not be changed.
Pros: with this manner, we can search vertices in O (log n) time with binary search against O(n) time with linear search.

'MyList' data structure:

This data structure is very similar to java List data structure with additional two methods:
1- sort: sorts the list with heap-sort in O (n log n) time.
2- binarySearch: searches specific item in list in O (log n) time. Before calling it, you must call sort method.

Test cases for vertex coloring:

Test 1:

Input:
1,7
1,5
1,6
1,2
1,4
4,10
4,2

10,9
11,3
7,8
5,3
3,8
2,3
9,2
6,5
2,6

Output:
0:1
1:1
2:2
3:3
4:3
5:2
6:3
7:2
8:1
9:1
10:2
Time: 13 milliseconds.

Test 2:

Input:
1,2
4,6
5,8
8,14
9,10
4,7
2,11
9,7
4,14
13,3
3,11
5,10
5,14
6,10
1,12
2,13

Output:
0:1
1:1
2:2
3:1
4:2
5:2
6:1
7:1
8:3

9:2
10:3
11:3
12:2
13:3
Time: 14 milliseconds.

Test 3:

Input:
1,4
1,3
1,1000
135,3
135,9
135,8
88,1000
1000,90
88,44
8,15
800,44
24,3
33,8
15,44
4,44
90,800
15,9

Conversion of input:
1,4
1,3
1,0
5,3
5,9
5,8
7,0
0,6
7,10
8,13
2,10
12,3
11,8
13,10
4,10
6,2
13,9

Output:
0:1
1:2
2:2
3:1
4:3
5:2

6:3
7:2
8:1
9:1
10:1
11:2
12:2
13:2
Time: 64 milliseconds.

Test cases for edge coloring:

Note: for coloring a K_N graph which N is odd, the algorithm will perform task for K_{N+1} graph without losing any generality. When N is odd, you can add a vertex to graph and assume that this vertex is redundant is connected to all other vertices.in this manner, one color will be added which is redundant and you can delete it.

Color 0 in output is meaningless. You can remove it (its line).
Label of vertices starts form zero.

Test 1:

N = 4:
Output:
0,0:0
0,1:1
0,2:3
0,3:2
1,0:1
1,1:0
1,2:2
1,3:3
2,0:3
2,1:2
2,2:0
2,3:1
3,0:2
3,1:3
3,2:1
3,3:0
Time: 5 milliseconds.

Test 2:

N = 5:
Output: (This is actually output of N= 6 too. In this case, you can delete vertex 5 and color 5.)

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
Time: 10 milliseconds.

Test 3:

N = 7:
Output: (This is actually output of N= 8 too. In this case, you can delete vertex 7 and color 7.)

0,0: 0
0,1: 1
0,2: 5
0,3: 2
0,4: 6
0,5: 3
0,6: 7
0,7: 4
1,0: 1
1,1: 0
1,2: 2
1,3: 6
1,4: 3
1,5: 7
1,6: 4
1,7: 5
2,0: 5
2,1: 2
2,2: 0

2,3: 3
2,4: 7
2,5: 4
2,6: 1
2,7: 6
3,0: 2
3,1: 6
3,2: 3
3,3: 0
3,4: 4
3,5: 1
3,6: 5
3,7: 7
4,0: 6
4,1: 3
4,2: 7
4,3: 4
4,4: 0
4,5: 5
4,6: 2
4,7: 1
5,0: 3
5,1: 7
5,2: 4
5,3: 1
5,4: 5
5,5: 0
5,6: 6
5,7: 2
6,0: 7
6,1: 4
6,2: 1
6,3: 5
6,4: 2
6,5: 6
6,6: 0
6,7: 3
7,0: 4
7,1: 5
7,2: 6
7,3: 7
7,4: 1
7,5: 2
7,6: 3
7,7: 0

Time: 16 milliseconds.

Strengths:

'MyList' data structure: Search in O (log n) instead of search in O(n).
Conversion of inputs vertices to continuous labels: causes direct access to array of vertices in O (1) instead of search whole array in O(n).

Weak points:

Edge coloring algorithm only works for complete graphs.
Vertex coloring algorithm only works for undirected graphs.

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