Istanbul Technical University

Computer Engineering Department

Fatima Rahimova 150180905

Analysis of Algorithms 2

CRN: 21549

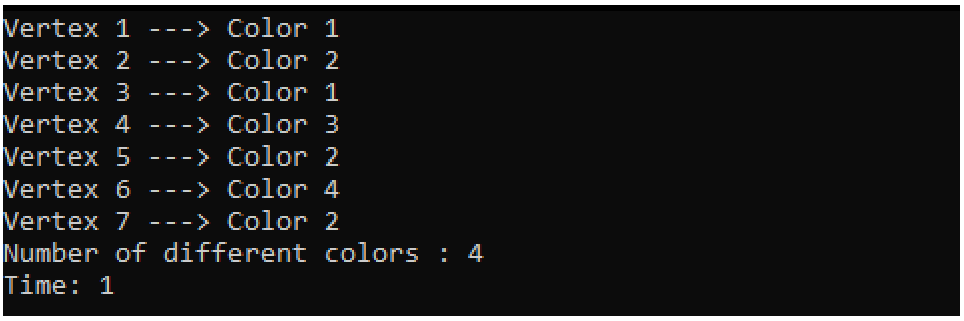
Assignment 2

Table of Contents

1) Adjecancy matrix for G2

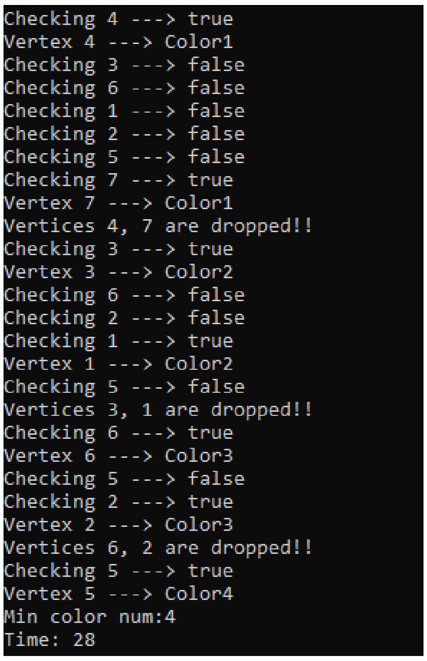
Regions	1	2	3	4	5	6	7
1	0	1	0	1	1	0	0
2	1	0	1	1	0	0	0
3	0	1	0	1	0	1	1
4	1	1	1	0	1	1	0
5	1	0	0	1	0	1	0
6	0	0	1	1	1	0	1
7	0	0	1	0	0	1	0

2) Screen shot (SS) of the output for Question-2.



```
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 1
Vertex 4 ---> Color 3
Vertex 5 ---> Color 2
Vertex 6 ---> Color 4
Vertex 7 ---> Color 2
Number of different colors : 4
Time: 1
```

3) SS of the output for Question-3 a)



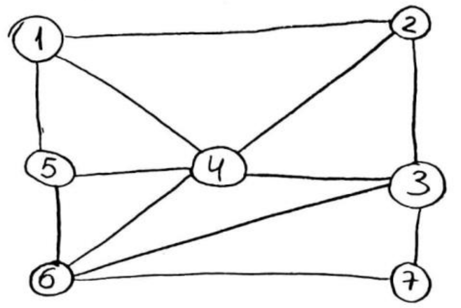
```
Checking 4 ---> true
Vertex 4 ---> Color1
Checking 3 ---> false
Checking 6 ---> false
Checking 1 ---> false
Checking 2 ---> false
Checking 5 ---> false
Checking 7 ---> true
Vertex 7 ---> Color1
Vertices 4, 7 are dropped!!
Checking 3 ---> true
Vertex 3 ---> Color2
Checking 6 ---> false
Checking 2 ---> false
Checking 1 ---> true
Vertex 1 ---> Color2
Checking 5 ---> false
Vertices 3, 1 are dropped!!
Checking 6 ---> true
Vertex 6 ---> Color3
Checking 5 ---> false
Checking 2 ---> true
Vertex 2 ---> Color3
Vertices 6, 2 are dropped!!
Checking 5 ---> true
Vertex 5 ---> Color4
Min color num:4
Time: 28
```

3B) Pseudo Code and Time Complexity

```
Step1 : Create a degree_list
Step2 : Assign number of the node’s degrees to this list
Step3: Sort them in descending order
Inside while1:
    Step4: Increasing color by 1
    Step5: Copy degree_list
    Step6: Delete colored ones from the copied list
    Inside while2:
        Step7: Increasing color by1
        Step8: Make first element colored
        Step9: Delete node’s adjacent nodes from the copied list
        Step10: Check list’s size
        Step11: If there is any element in the list go to Step 7
```

3B) cont'd…

How my code works?



Node	Its adjacent nodes	Degree
1	2, 4, 5	3
2	1, 3, 4	3
3	2, 4, 6, 7	4
4	1, 2, 3, 5, 6	5
5	1, 4, 6	3
6	3, 4, 5, 7	4
7	3, 6	2

degree_list = {4, 3, 6, 1, 2, 5, 7}

↳ node ids

↳ according to their degree

First element → 4 → color1 → {~~4~~, ~~3~~, ~~6~~, ~~1~~, ~~2~~, ~~5~~, 7} → {7}

removing node itself and its adjacent nodes from the list

First element → 7 → color 1 → {~~7~~} → { }

First element → 3 → color 2 → {~~4~~, ~~3~~, ~~6~~, 1, ~~2~~, 5, ~~7~~} → {1, 5}

First element → 1 → color 2 → {~~1~~, 5} → {~~5~~} → { }

{~~4~~, ~~3~~, 6, ~~1~~, 2, 5, ~~7~~}

↳ deleting already colored nodes

First element → 6 → color 3 → {~~6~~, 2, ~~5~~} → {2}

First element → 2 → color 3 → {~~2~~} → { }

{~~4~~, ~~3~~, ~~6~~, ~~1~~, ~~2~~, 5, ~~7~~}

↳ deleting already colored nodes

First element → 5 → color 4 → {~~5~~} → { }

↳ NO element left in list

Graph coloring is done

Since there is 2 nested while loops, the complexity of this algorithm will be O(n^2).

4) Analysis of GA3

GA3 does not perform well for every possible scenario. One of the big problems in this algorithm is this algorithm does not return back to the nodes. It does not use stack, queue, or any other beneficial data structure, that is why if there is any adjacent node that does not visited algorithm does not work. If there is not any unvisited adjacent node algorithm does not work.

The counter scenario of GA3 failing:
Step1: Choose vertex1 and color it with color1.

Step2: Then select vertex5 and color it with color2.

Step3: Then select one of the vertex5's adjacent nodes, vertex4 and color it with color3.

Step 4: Select vertex 6 and then color it with color1.

Step5: Select vertex3 and color it with color2.

Step6: Select vertex2 and color it with color4.

So we already visited all adjacent nodes of vertex2 and that is why we cannot visit to node 7. That is why we are NOT able to color vertex 7 and therefore we cannot color whole graph. It shows that this algorithm does not perform well for every possible scenario.

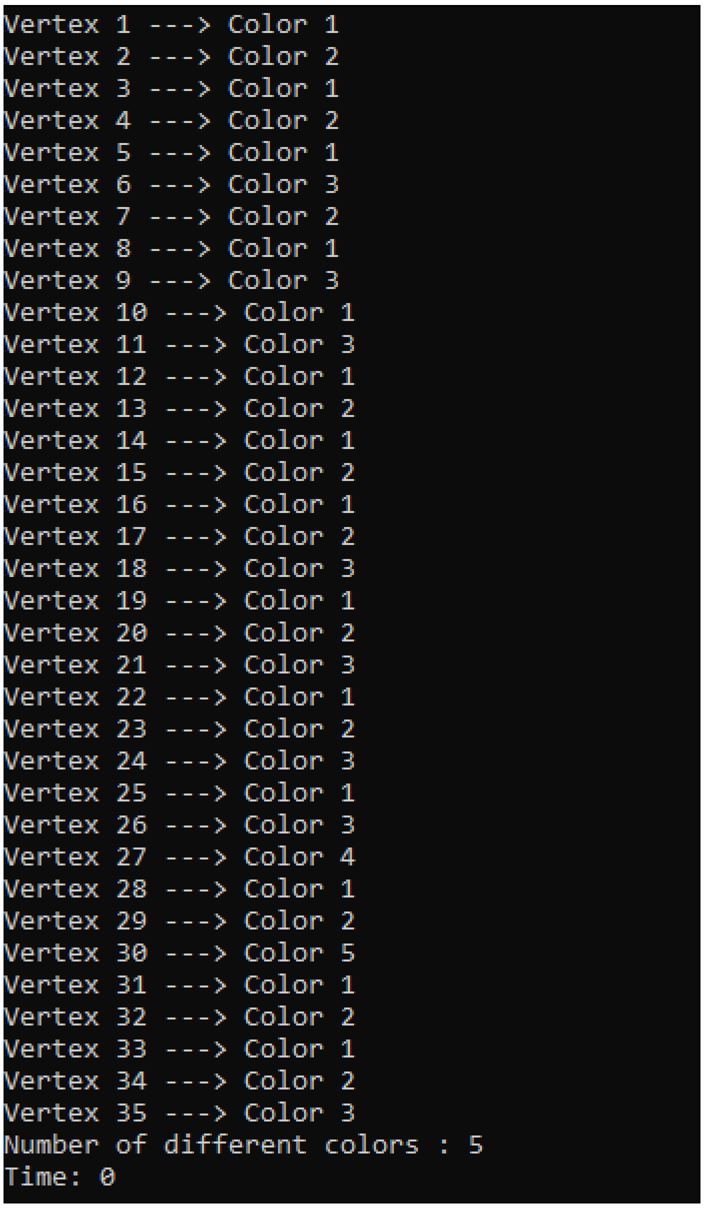
5A) Adjecancy matrix for SNG

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1

23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	1	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0

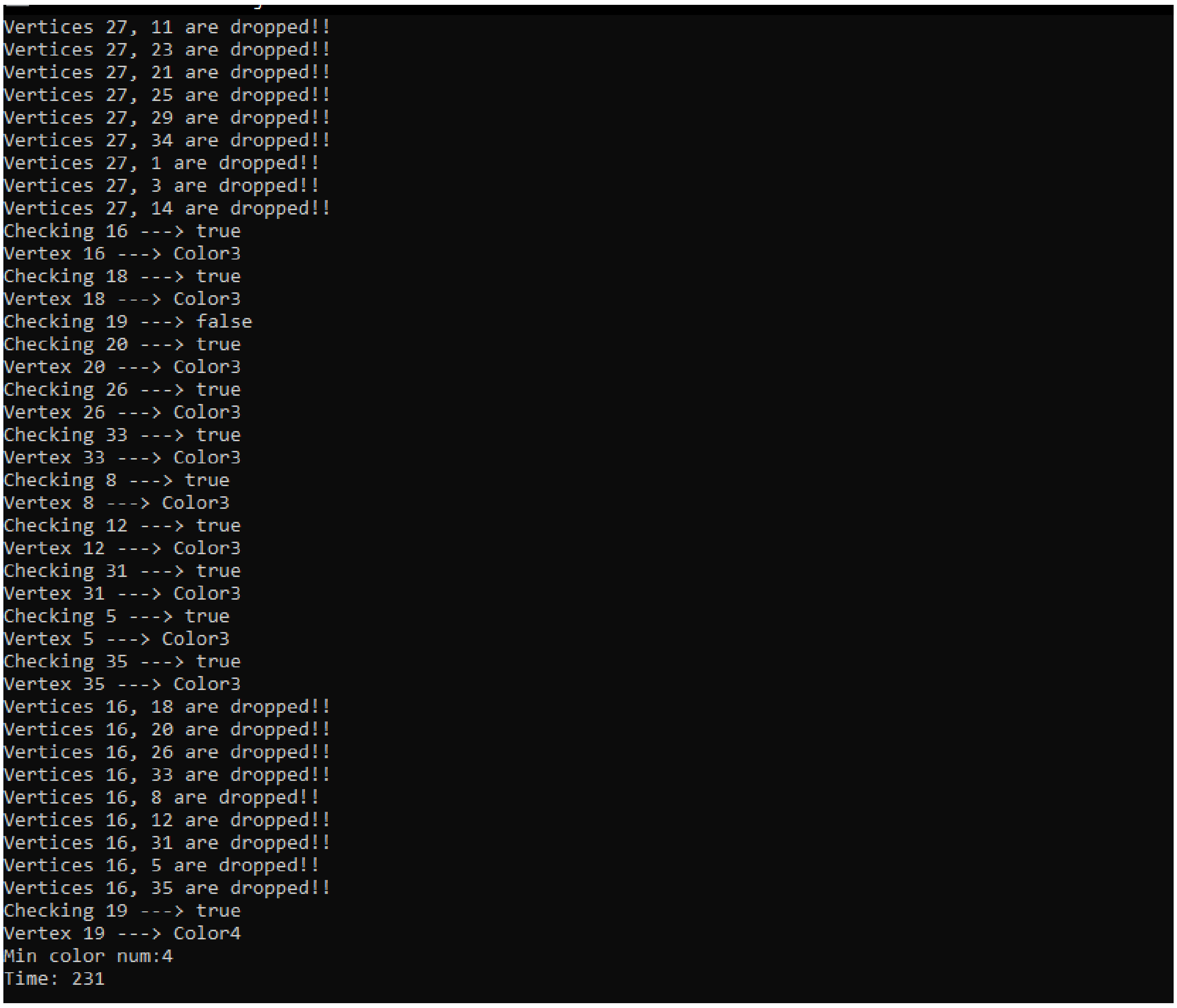
5C) Which greedy algorithm performed better in terms of execution time?

Result of GA1:



```
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 1
Vertex 4 ---> Color 2
Vertex 5 ---> Color 1
Vertex 6 ---> Color 3
Vertex 7 ---> Color 2
Vertex 8 ---> Color 1
Vertex 9 ---> Color 3
Vertex 10 ---> Color 1
Vertex 11 ---> Color 3
Vertex 12 ---> Color 1
Vertex 13 ---> Color 2
Vertex 14 ---> Color 1
Vertex 15 ---> Color 2
Vertex 16 ---> Color 1
Vertex 17 ---> Color 2
Vertex 18 ---> Color 3
Vertex 19 ---> Color 1
Vertex 20 ---> Color 2
Vertex 21 ---> Color 3
Vertex 22 ---> Color 1
Vertex 23 ---> Color 2
Vertex 24 ---> Color 3
Vertex 25 ---> Color 1
Vertex 26 ---> Color 3
Vertex 27 ---> Color 4
Vertex 28 ---> Color 1
Vertex 29 ---> Color 2
Vertex 30 ---> Color 5
Vertex 31 ---> Color 1
Vertex 32 ---> Color 2
Vertex 33 ---> Color 1
Vertex 34 ---> Color 2
Vertex 35 ---> Color 3
Number of different colors : 5
Time: 0
```

Result of GA2:



```
Vertices 27, 11 are dropped!!
Vertices 27, 23 are dropped!!
Vertices 27, 21 are dropped!!
Vertices 27, 25 are dropped!!
Vertices 27, 29 are dropped!!
Vertices 27, 34 are dropped!!
Vertices 27, 1 are dropped!!
Vertices 27, 3 are dropped!!
Vertices 27, 14 are dropped!!
Checking 16 ---> true
Vertex 16 ---> Color3
Checking 18 ---> true
Vertex 18 ---> Color3
Checking 19 ---> false
Checking 20 ---> true
Vertex 20 ---> Color3
Checking 26 ---> true
Vertex 26 ---> Color3
Checking 33 ---> true
Vertex 33 ---> Color3
Checking 8 ---> true
Vertex 8 ---> Color3
Checking 12 ---> true
Vertex 12 ---> Color3
Checking 31 ---> true
Vertex 31 ---> Color3
Checking 5 ---> true
Vertex 5 ---> Color3
Checking 35 ---> true
Vertex 35 ---> Color3
Vertices 16, 18 are dropped!!
Vertices 16, 20 are dropped!!
Vertices 16, 26 are dropped!!
Vertices 16, 33 are dropped!!
Vertices 16, 8 are dropped!!
Vertices 16, 12 are dropped!!
Vertices 16, 31 are dropped!!
Vertices 16, 5 are dropped!!
Vertices 16, 35 are dropped!!
Checking 19 ---> true
Vertex 19 ---> Color4
Min color num:4
Time: 231
```

As you can see in the screenshots in above, GA1 takes almost 0 milliseconds (so close time to 0) for coloring whole SNG, but meanwhile GA2 requires 231 milliseconds for coloring same SNG. It shows that GA1 performs much better in terms of execution time. When we try to implement time-efficient graph coloring algorithm, we should choose GA1 not GA2.

5D) Which greedy algorithm performed better in terms of finding the minimum number of colors?

We can use same screenshots that I already added for question 5C in above. As you can observe in these screenshots, GA1 can color whole SNG with 5 colors, meanwhile with GA2 we can color whole graph with 4 colors. It shows that GA2 performs better in terms of finding the minimum number of colors. If we try to impelement an algorithm that helps us coloring a graph with minumum number of colors, we should choose GA2 not GA1.

Discussion: We already saw that GA1 does not always gives minimum number of colors. If we analyze GA2 carefully, we will observe that GA2 is not stable algorithm. So, if we will sort the nodes that have same degree in ascending order, we will get minimum number of colors as 3 which is less than 4. So, both GA1 and GA2 will not always give minimum number of colors for all graphs.

Additionally, we alrady saw that GA1 is faster than GA2 in terms of execution time.

The big problem related with GA3 is it does not guaranteed that whole graph will be colored or not. If all adjacent nodes of a node are colored, GA3 does not check that all nodes in graph is colored or not. If all adjacent nodes of a node are colored it just stops.