CS/ECE 374 P25

Pengxu Zheng, Jiawei Tang

TOTAL POINTS

100 / 100

QUESTION 1

1 25.A. 20 / 20

✓ **- 0 pts** **Correct**

- 20 pts Incorrect algorithm (see comments below)

- 5 pts Correct algorithm, but running time depends on $$k$$ (e.g., $$O(k! m)$$)

- 10 pts Correct algorithm, but running time is $$\omega(m)$$ even for constant values of $$k$$

- 5 pts Running time incorrectly stated or missing running time analysis

- 5 pts Correct algorithm, but missing or largely incorrect justification/proof of correctness

- 2 pts Minor mistake in algorithm, proof of correctness or running time (see comments below)

- 15 pts IDK

- 60 pts IDK

You can further organize the runtime analysis.

QUESTION 2

2 25.B. 80 / 80

✓ **- 0 pts** **Correct**

- 80 pts Incorrect algorithm (see comments below)

- 20 pts A new graph is constructed, but description of construction is unclear or contains errors (see comments below)

- 20 pts Failed to state how to traverse newly construction graph (i.e., using DFS/BFS)

- 30 pts Algorithm, proof of correctness and runtime analysis is correct, but is slower than $$O(m)$$

- 20 pts Runtime or runtime analysis missing/incorrect

- 20 pts Missing or largely incorrect justification/proof of correctness

- 8 pts Minor mistake algorithm, proof of correctness or runtime analysis (see comments below)

HW Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- ≪**Jiawei Tang**≫: ≪**jiaweit2**≫
- ≪**Pengxu Zheng**≫: ≪**pzheng5**≫

25

Solution:

25.A.

Since here, a closed walk is that we start and end at the same vertex in a directed graph, such starting vertex must be in a cycle. Therefore, our plan is to first find every strongly connected component in the directed graph. Then we iterate through each strongly connected component to find out if there exists a closed walk that uses all k colors. Because within a SCC, every two vertices form a path so that a closed walk can contain all edges, we can directly iterate through all edges to check if they use all k colors. If all strongly connected component do not exists such closed walk, we can conclude that there is no closed walk that can use all k colors. Assume we have a directed graph G. If f(G) is True, then such walk exists. If f(G) is False, then otherwise.

$f(G)$:

```
Using Mar.14 slide 44 algorithm to find all SCCs in directed G //O(m+n)
for each c in all SCCs do
    s = set()
    for each edge e in c do
        s.add(c(e))
    end for
    if s contain all k colors then
        return True
    end if
end for
return False
```

The two for loops are basically iterate through all the edges in G so the running time is $O(m)$ where m is number of the edges of G. Also, we need to check if "s contain all k colors" $O(n)$ times. $O(n)$ here stands for the worst case when each vertex is a SCC so that there are n SCC. The total time complexity is linear, $O(2m + n + nk)$.

1 25.A. **20 / 20**

✓ **- 0 pts Correct**

- 20 pts Incorrect algorithm (see comments below)

- 5 pts Correct algorithm, but running time depends on $$k$$ (e.g., $$O(k! m)$$)

- 10 pts Correct algorithm, but running time is $$\omega(m)$$ even for constant values of $$k$$

- 5 pts Running time incorrectly stated or missing running time analysis

- 5 pts Correct algorithm, but missing or largely incorrect justification/proof of correctness

- 2 pts Minor mistake in algorithm, proof of correctness or running time (see comments below)

- 15 pts IDK

You can further organize the runtime analysis.

25.B. Our plan here is to first find all of the SCCs in the directed graph G, and then create a meta-graph G' from G. As we know G' is a DAG, therefore we want to find out the source node of the DAG and do DFS on the source node. For this meta-graph G', each vertex contains a set of colors. Also, because in the original graph G, we can have multiple edges connecting two SCCs, then for each edge in G', it should also contains a set of colors.
Then when we do DFS on the source node of G', we always keep an another set s' corresponding to each vertex. Because here each edge contains a set of colors but each time we can only use one of them, s' means the set of sets of colors that are still not used at this point. Because of the assumption that there are only a constant number of colors, s' is of constant length. Whenever s' contains an empty set, it means we have already used all of the 3 colors. Then, such walk exists. If there is no s' that contains empty set, then such walk doesn't exist. Assume we have the three colors 1,2,3.
$f(G)$:

```
Using Mar.14 slide 44 algorithm to find all SCCs in directed G
Create meta-graph G' from all SCCs of G (assign each vertex the set of colors that each SCC contains,
and assign each edge the set of colors that contains the colors of edge(s) which connect 2 SCCs)
//O(m+n) above
s = all source nodes in G' found by using topological sort
for each edge e in G' do
  Unmark e
end for
for each source in s do
  Initialize a stack
  stack.add(source)
  source.s' = {{1,2,3}− source's set of colors}//source's s' has some colors, representing some colors
  are not used
  while stack is not empty do
    v = stack.top
    pick an arbitrary unmarked out-going edge e = (v,u)
    if no such e exists then
      stack.pop()
    else
      u.s' = set()
      for each color in e do
        for each setOfColors in v.s' do
          u.s'.add({setOfColors − {color} − u's set of colors})
        end for
      end for
      if ∅ in u.s' then
        return True
      end if
      mark e
      stack.add(u)
    end if
  end while
end for
return False
```

The first for loop is going through all edges so it takes $O(m)$. In the second for loop, it is basically iterating through each vertices and edges. Note that within the while loop, there is one for loop with another for loop nested inside. These two loops are constant time because we only have 3 colors so that

there can only be at most 3 iterations for the outer loop. And there are only 8 combinations of 3 colors so that there can only be at most 8 iterations for the inner loop.

To conclude, the total runtime is $O(m+n) + O(m) + O(m+n) = O(2m+n)$.

2 25.B. **80 / 80**

✓ **- 0 pts Correct**

- 80 pts Incorrect algorithm (see comments below)

- 20 pts A new graph is constructed, but description of construction is unclear or contains errors (see comments below)

- 20 pts Failed to state how to traverse newly construction graph (i.e., using DFS/BFS)

- 30 pts Algorithm, proof of correctness and runtime analysis is correct, but is slower than $$O(m)$$

- 20 pts Runtime or runtime analysis missing/incorrect

- 20 pts Missing or largely incorrect justification/proof of correctness

- 8 pts Minor mistake algorithm, proof of correctness or runtime analysis (see comments below)

- 60 pts IDK