CS 374 HW 8 Problem 3

Aldo Sanjoto, Hieu Huynh, quddus2

TOTAL POINTS

100 / 100

QUESTION 1

1 Halloween! 100 / 100

✓ **- 0 pts** Correct

- 10 pts Minor mistake in computing max number of candies

- 40 pts Incorrect algorithm for computing max number of candies

- 15 pts Running time for computing max numbers of candies is not linear

- 10 pts Running time for walk is incorrect (should be O(n^2))

- 10 pts Minor mistake in computing the walk

- 20 pts Unclear explanation on walk within each SCC

- 35 pts Didn't consider walk within each SCC

- 35 pts Failed to distinguish walk from path or tree (e.g. DFS)

- 50 pts Incorrect walk

- 10 pts Incorrect walk Length

- 75 pts IDK

- 87.5 pts Almost completely wrong

Question 3

PART1 Algorithm

- Compute the metagraph G^{scc} } $O(m+n)$
- Create array svals[1...k] initialized to zero where metagraph gives us k components
- For S_i in G^{scc}:

  for v in S_i:

  svals[S_i] += v

  } $O(n)$ v = value of vertex in component S_i

To get the value of S_i (component of G^{scc}) we have to sum all the values of the verticies in the component S_i. In G^{scc} we can think of S_i as a vertex of the metagraph that has the total # of candies of all the verticie that particular S_i component holds.

$\underbrace{O(l+k)}$ where k = number of verticies in G^{scc}, l = number of edges in G^{scc}

- Topologically sort the verticies of G^{scc} (i.e $S_1, S_2 \ldots S_k$) & let the first element of the order be the S_i component that contains the s vertex

- Initialize array totals[1...k] to 0 & array successor[1...k] to null

• For $i \leftarrow 1$ to n:

For s_j in $Out(s_i)$:

if totals[s_j] < totals[s_i] + svals[s_j]:

totals[s_j] = totals[s_i] + svals[s_j]

Successor[s_i] = s_j

$\underbrace{O(n + \sum_{i=1}^{n} \deg(s_i)}_{O(l+n)}$

$s_i \in G^{scc}$ vertex

• Retrieve the max value from total's array which gives the max amount of candy collected from the optimal walk

Worst case running time for algorithm in part 1 is => $O(m+n)$

Correctness of algorithm:

The following ~~an~~ Algorithm yields max possible amount of candy that can be yield because we see that the totals array ~~that~~ gives us the max ~~total~~ amount of ~~your~~ candy that can be collected up to s_i component totals[s_i]. This is because after all the iteration of the for loop we chose out of all the incomming edges into s_i the ~~totals[s_n]~~ s_h component that has the higest possible amound candy collect up to the s_h component & by selecting totals[s_h] to use for the total of s_i by total[s_h] + ~~va~~ sval[s_i] we guarantee that the total[s_i] is the max amount of candy that can be collected up to the s_i component & we can continueing tracing the total max sum [s_i] candies value from s_h all the way back to the source ~~vertex~~

~~[illegible] which has the ver~~

Q3) • First of all, we design an algorithm to compute the path of a vertex u to all other vertices in the SCCi that contain u.

• By doing DFS(u), we can find path from u to any vertices in same SCC. We record the path in an array of link list, called Path_foward_i

```
DFS(u) {
    Mark u as visited;
      Path_foward_i[u]. insert_at_the_end (u);
    For each edge (u,v) in Out (u){
        If (v is not visited){
            DFS(v)
        }
    }
}
```

• Next, we compute path from any vertices in SCCi to vertex u. We store these paths in an array of link list, called Path_backward_i[u]

```
My_DFS(u){
    Mark u as visited;
    Path_backward_i[u]. insert_at_the_front (u);
    For each edge (v,u) in In (u) {
        If (v is not visted){
            My_DFS (v);
        }
    }
}
```

- We compute all of these 2 arrays of link list for all SCCs in G.
- From part (1), we have the sequence of SCCs that we need to visit to get optimal walk
- Let call this sequence $SCC_1, SCC_2, SCC_3, \ldots SCC_k$
- We have that starting vertex $s \in SCC_1$.
- Let's array Final_Path be optimal walk.
- We denote vertex v_i be the random node that we chose to compute paths from v_i to all other vertices in SCC_i and paths from other vertices in SCC_i to v_i (we discussed this before)

```
For i ← 1 to k {
    if (i = 1){
        si ← s;
    }
    . Traverse through link list Path_Backward_i [si]
      to get back to vi .. When visit a vertex, mark
      it as visited, and add it to back of Final_Path.
    For j ← 1 to l: // l is the number of vertices in this SCC
        if(vj is not visited){
            Mark vj as visited;
            Traverse through link list Path_Foward_i [vj]
            to get to vertex vj. when visit a vertex,
            mark it as visited, and add it to Final_Path
            Traverse through link list Path_Backward_i [vj]
            to get back to vertex vi when visit a vertex,
            mark it as visited, and add it to Final_Path
```

For $i \leftarrow 1$ to k {

......

For $j \leftarrow 1$ to l {.........

...

...

This is the part on previous page

}

- Traverse through list of vertices in SCC_i, find the vertex V_a that has the outgoing edge to $SCC_{(i+1)}$
- Let say $(V_a, V_b) \in E(G)$ and $V_b \in SCC_{(i+1)}$
  $V_a \in SCC_i$

- Traverse through Link list Path_Forward_i [V_a] to get to V_a. when visit a vertex, add it to Final_Path.

- $S_{i+1} \leftarrow V_b$

}

- The path to get maximum candy is the path that is stored in array Final_Path
- To compute all Path_Forward and Path_Backward arrays of link list for all SCC, it take

$$\sum_{i=1}^{k} O(m_i + n_i) = O\left(\sum_{i=1}^{k} m_i + \sum_{i=1}^{k} n_i\right) = O(m+n)$$

// m_i and n_i are # of edges and # of vertices of SCC_i

- To compute sequence of $SCC_1, SCC_2, \dots SCC_k$ takes $O(m+n)$ discussed in previous part.
- To compute the Final_Path, it takes $O(m+n)$ because each vertex we visit a constant time

Therefore, the total running time is $O(m+n)$

- the length of walk in the worst case is $3m$ because for each SCC_i, we need to traverse from s_i to v_i, which takes $O(m_i)$ in worst case. Then we visit all vertices in SCC_i, which take $O(m_i)$. Finally, we need to traverse from v_i to the transition to the next SCC_i, which take $O(m_i)$

$\Rightarrow$ Worst case: $\sum_{i=1}^{k} 3m_i = 3m$

1 Halloween! **100 / 100**

- ✓ **- 0 pts** Correct
- **- 10 pts** Minor mistake in computing max number of candies
- **- 40 pts** Incorrect algorithm for computing max number of candies
- **- 15 pts** Running time for computing max numbers of candies is not linear
- **- 10 pts** Running time for walk is incorrect (should be O(n^2))
- **- 10 pts** Minor mistake in computing the walk
- **- 20 pts** Unclear explanation on walk within each SCC
- **- 35 pts** Didn't consider walk within each SCC
- **- 35 pts** Failed to distinguish walk from path or tree (e.g. DFS)
- **- 50 pts** Incorrect walk
- **- 10 pts** Incorrect walk Length
- **- 75 pts** IDK
- **- 87.5 pts** Almost completely wrong