CS 721: Advanced Algorithms & Analysis

Homework – 2

Chakradhar Reddy Donuri

E949F496

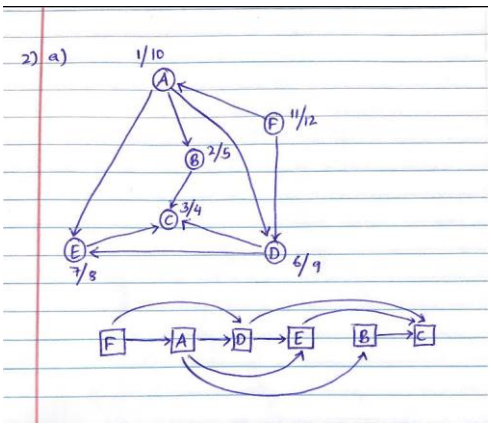
Q1. (15 points) Suppose all golf players are either professionals or amateurs and between any pair of golf players there may or may not be a rivalry. Suppose we are given n golf players and a list of r pairs of golf players for which there are rivalries (note that, you can represent this information as an undirected graph). Give an O(n + r)-time algorithm that determines whether it is possible to designate some of the golf players as professionals and and the remainders as amateurs such that each rivalry is between a professional and an amateur. If it is possible to perform such a designation, your algorithm should produce it. You do not need to provide pseudo-code. Explain in details how your algorithm will work. Hint: You can perform BFS to visit all vertices and compute distances. What would odd and even distances mean?

Answer :

- Let the Undirected graph be G which represents the given information
- The graph G consists of 'n' golf players and 'r' edges i.e. edges depicts rivalry
- Now, perform BFS to visit all the vertices and assign even distance from vertex to professional and odd distance to amateurs.
- Assign any golf player as professional and then assign its neighbor golf player to be amateur or vice-versa.
- If the golf player (professional or amateur) is assigned to same category level then report that designation is not possible.
- It takes O(n) to designate each player as professional or amateur and O(r) to check the edges. Therefore it gives to O(n + r).

Q2.

(a) Consider the following directed acyclic graph.
Compute pre and post number for each node of the above graph starting from node A and draw graph after it is topologically sorted (linearized).



(b) (10 points) Give a linear-time algorithm that takes as input a directed acyclic graph G = (V, E) and two vertices s and t and returns the number of simple paths (i.e., does not contain cycles) from s to t in G. For example, in the above graph there are four simple paths from vertex A to C: A → B → C, A → E → C, A → D → C and A → D → E → C. Your algorithm needs only to count the number of paths not list them.

Hint: First linearize the graph and locate nodes s and t. Which nodes will appear in simple paths from s to t? You need to consider only those nodes that may appear in simple paths between s and t (including s and t) in linearized order, maintain an array for whose indices are linearized ordering of the nodes and update the array elements as you process each node in the linearized order to get your final answer. What should this array contain? How will you update this array?

Answer:

Algorithm:

- Consider DAG G=(V,E) and two vertices s (starting node) and t (ending node).
- If ‘s’ equal to ‘t’ the return 1
- Now count the path staring from s and at each point count of ‘s’ is updated but value of ‘t’ is fixed.
- Now sum the number of paths recursively, the nodes which are adjacent to ‘s’
- Return the number of paths which have been counted starting from vertex ‘s’ to ‘t’

Pseudo Code:

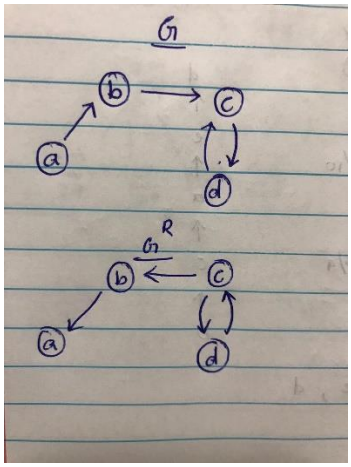
```
PATH_COUNT (s,t)

if (s ==t)
```

```
        Return 1;
 else
{
        {
        Foreach u ∈ adj[v]
                {
                count(s) = count(s) + PATH_COUNT (u,t);
                }
        }
 return count(s);
}
```

Q3. We have seen that algorithm for finding strongly connected components of a directed graph G = (V;E) works as follows. In the first step, compute DFS on the reverse graph GR and compute post numbers, Then run the undirected connected component algorithm on G, and during DFS, process the vertices in decreasing order of their post number from step 1. Now professor Smart Joe claims that the algorithm for strongly connected component would be simpler if it runs the undirected connected component algorithm on GR (instead of G), but during DFS, process the vertices in increasing order of their post number from step 1.
(a) (10 points) Explain when the algorithm proposed by prof. Smart Joe might produce incorrect answer

Answer:

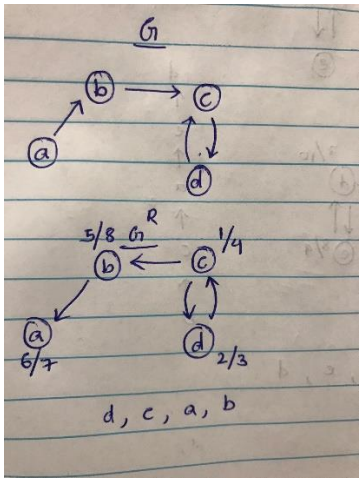
Consider a Strongly connected Components of a Directed Graph G = (V , E)



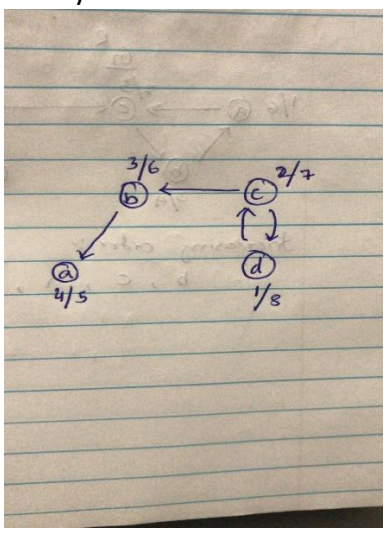
- As shown in the above picture we have G and G^R with a,b,c,d vertices and edges connecting them
- We have a strongly connected components {c,d}
- Consider writing pre/post numbers to the graph and the topological order is given as d,c,a,b
- Consider the starting node as ‘c’ then it has to explore ‘d’ then ‘a’ and then ‘b’
- It depicts that post value of ‘d’ should be low than ‘a’, ‘c’, ‘b’.

- Applying DFS staring at ‘d’, we reach all the vertices, but from the above picture it is not possible as only {c, d} are strongly connected.
- Hence, Professor’s assumption is wrong, because the algorithm will return the graph which has single strongly connected component whereas we only have ‘c’, ’d’ which are strongly connected.

(b) (10 points) With an example, (along with post numbers) show that professor Smart Joe is wrong.
Answer:



- Consider G^R in the above picture in which we ‘c’ as starting node to calculate pre/post numbers
- As per the Professor’s assumption of increasing order (d, c, a, b) i.e. ‘d’ is explored first and then followed by ‘c’ then ‘a’ and then ‘b’



- Therefore, if we start from node ‘d’ as per the order, we have to reach all the vertices but in this case it returns whole path in a single strongly connected component.

- Hence, professor's assumption is incorrect.

Q4. We are given a directed graph G = (V;E) on which each edge (u; v) 2 E has an associated value r(u; v) which is a real number in the range 0 _ r(u; v) _ 1 that represents the reliability of a communication channel from vertex u to vertex v. We interpret r(u; v) as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent.
(a) (10 points) Give an efficient algorithm to find the most reliable path between two given vertices of this graph.
Hint: Since probabilities are independent, this means we want to find a path for which product of the reliabilities will be maximum.

Algorithm to find most reliable path between two vertices

Dijkstra(G, l, s)
Input : Graph G=(V ; E) A directed or undirected
Output: For all the vertices 'u' reachable from 's', dist(u) is set to distance from 's' to 'u'
For all u ∈ V
    dist(u)= ∞
    prev(u) = NIL
dist(s) = 0
H= makequeue (V) (using dist-values as keys)
While H ≠ Ø:
    u = deleteMin(H)
    for all edges (u,v) ∈ E:
        if dist(v) > dist(u) + l(u,v):
        dist(v) = dist(u) + l(u,v)
        prev(v) = u
        decreaseKey(H,v)

- In order to get the most reliable path, we aim to find a path 'p' such that the product of the probabilities on that path is maximized.
- Let s be the source and t be the terminal
  let p = (v_0, v_1, · · · ,v_k) where v_0 = s and v_k = t then
    p = arg max **product** (v_{i-1}, v_i) where **i= 0 to k**
- Run Dijkstra's algorithm by letting the weight on edge (u, v) be w(u, v) = − log r(u, v)
- Now, the minus logarithm will convert a minimization problem into a maximization problem, a shortest path p on the converted graph satisfying
    p = arg min **sum**(w(v_{i-1}, v_i) where **i=1 to k**
- As it is a Dijkstra's algorithm, the initialization of weights takes O(E) time.

- The most reliable path is the shortest path from s to t and that paths reliability is the product of the reliabilities of its edges. Hence, the algorithm runs in O (E log V)

4)(b) (10 points) If the directed graph does not contain a cycle, can you give a better algorithm?
Explain how your algorithm will work. What is the running time of your algorithm?

Considering acyclic graph (DAG) then we can apply DFS algorithm
Algorithm:-

Directed Acyclic Graph G=(V, E)

```
for all u ∈ V {
        dist(u) = ∞
        prev(u) = nil
        dist(s) = 0
        Linearize G
                For each u∈ V, in linearized order
                For all edges (u,v) ∈ V
                        Update(u,v)
}
```

Vertices in DAG appear in increasing linearized order, hence it can be sorted in a topological order by DFS. Now the shortest path in the original graph will be the longest path in the converted graph. Therefore, We need to perform sorting for V-1 times for V vertices to find the path and The Runtime is given as O((V-1)Log(V+E))

Q5. Let G = (V;E) be an undirected connected graph, where all edge weights are positive and equal. Describe an algorithm (no need to provide pseudo-code) that finds MST of G and is asymptotically more efficient than Prim's and Kruskal's algorithm. What is the running time of your algorithm?

Answer:

- Given an Undirected connected graph G = (V,E) with positive and equal edge weights.
- Let the weight of an edge be 'x' and 'V' be the vertices in the tree then the whole weight of the tree 'T' would be
  weight(T) = x * (V-1)
- As the weights are equal and positive we can apply BFS (Breadth First Search) because it will not visit any node or vertex twice.
- By this application of BFS we can get a tree 'T' that will be a part of MST (Minimum Spanning Tree)
- Runtime of this process will be O(V + E)

Q6. Suppose all edge weights in a graph are integers in the range 1 to |V|. How fast can you make Kruskal's algorithm run? What if the edge weights are integers in the range 1 to W for some constant W?
Hint: You can use the fact that counting sort can sort n integers in the range 0 to k in (n + k) time.

Answer :
Kruskal's algorithm sorts edges in non-decreasing order by weight. If the edge weights are integers in the range 1 to |V|, we can use Counting-Sort to sort the edges in (V +E) time (i.e. Counting-Sort can sort n integers in the range 0 to k in (n + k) time).

Therefore, Kruskal's algorithm will run in O(V +E+V log V) = O(E+V log V) time.

If the edge weights are integers in the range from 1 to W for some constant W, we can use Counting-Sort to sort the edges in (W + E) time and Kruskal's algorithm will run in O (W + E + V log V) time

Q7. (15 points) Let T be an MST of a graph G = (V ; E). Suppose we decrease the weight of one of the edges of T. Show that T (with decreased edge weight) is still an MST for G. More formally, let T be a minimum spanning tree for G with edge weights given by a weight function w. Choose one edge (x; y) ∈ T and a positive number k and define a new weight function w^l by

$$w'(u,v) = \begin{cases} w(u,v), & \text{if } (u,v) \neq (x,y) \\ w(x,y) - k, & \text{if}(u,v) = (x,y) \end{cases}$$

Show that T is also an MST for G with edge weights given by w^l.
Hint: First, find out how w(T) and w^l (T) are related (their values differs in a single edge). In particular, which one is larger among these two? Next, consider any other spanning tree T^l (different from T). You need to show that $w^l(T) < w^l(T^l)$, that is even with the new edge weights w^l, T is MST and T^l is not. To show this, you need to consider two cases, either the edge (x; y) ∈ T^l, that is the edge (x; y) whose weight differs under new edge weight w^l is a member of T or (x; y) ∈ T^l. Show that in both these cases, $w^l(T) \leq w^l(T^l)$.

Answer:

Consider any other spanning tree **T^l**
Let
weight(T) = **w(T)**
new edge weight(T) = **$w^l(T)$**
weight(T^l) **= $w(T^l)$**
new edge weight(T^l) **= $w^l(T^l)$**

let **w(T) = Sum (w (x , y))** where (x , y) ∈ T

Now, $\mathbf{w^l(T) = w(T) - k}$
$\mathbf{w(T) > w^l(T)}$ (w(T) is larger than $w^l(T)$)

Now for other spanning tree $\mathbf{T^l}$, so that $\mathbf{w(T) \leq w(T^l)}$

If (x, y) $\notin \mathbf{T^l}$, then $\mathbf{w^l}(\mathbf{T^l}) = w(\mathbf{T^l}) \geq w(\mathbf{T}) > \mathbf{w^l}(\mathbf{T})$
If (x, y) $\in \mathbf{T^l}$, then $\mathbf{w^l}(\mathbf{T}) = w(\mathbf{T^l}) - k \geq w(\mathbf{T}) - k$
In both the situations $\mathbf{w^l}(\mathbf{T}) \leq \mathbf{w^l}(\mathbf{T^l})$
So **T** is also an MST for G with edge weights given by $\mathbf{w^l}$