CS 5633 Analysis of Algorithms – Spring 23

Exam 3

NAME: Jyotirmay Nag Setu

- This exam is closed-book and closed-notes, and electronic devices such as calculators or computers are not allowed. You are allowed to use a cheat sheet (half a single-sided letter paper).
- Please try to write legibly – if I cannot read it you may not get credit.
- **Do not waste time** – if you cannot solve a question immediately, skip it and return to it later.

1) Greedy Algorithms		30
2) Amortized Analysis		30
3) Graph Algorithms		20
4) Shortest Paths		20
		100

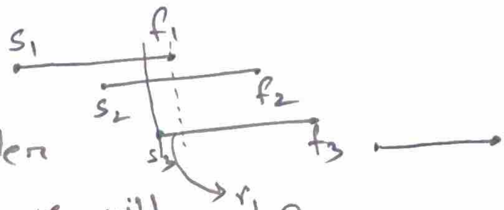
25

1 Greedy Algorithms (30 Points)

In class we considered the activity selection problem where we computed a maximum sized subset of non-conflicting activities that could use a shared resource (such as a lecture hall). Each activity a_i can be viewed as an interval $[s_i, f_i)$ where s_i is the *start time* of a_i and f_i is the *finishing time* of a_i. In the classic activity selection problem, there is only one resource. Suppose we consider a generalization of this problem were we are allowed to use more resources (e.g., we can use more than 1 lecture hall). In this generalization, we want *every* activity to be able to use some resource. The goal is to minimize the number of resources needed to accomodate this. More formally:

1. Let $r_1, \ldots, r_k$ be the resources (e.g., lecture halls) that you use.
2. Each activity a_i must be assigned to some resource r_j.
3. For each resource r_j, the activities assigned to it must be non-conflicting.
4. The goal is to minimize k (the number of resources you use).

Give a greedy algorithm that computes an optimal solution for any set of n activities. Argue why your algorithm is correct.



So, we will start by sorting the intervals in a non decreasing order based on their finish time. Then we will make a set of activity that overlaps the first activity that has the earliest finishing time. we will find the most late s_i from this set and $f_1 - s_i$ will be the first interval where we can assign our first resource.

function minResource (s_i f_i) {

a_1 = earliest finishing time.

A = { set of all the overlaping activity with a_1 }

s_i = most late starting activity from A

then pick the earliest finishing activity not in A

and repeat the steps

}

What is A second time?

30

2 Amortized Analysis(30 Points)

Consider a sequence of n operations on a data structure in which the cost c_i of the ith operation is defined as $c_i = 5i$ if i is a power of 2 and $c_i = 1$ otherwise.

1.

1. Use an aggregate analysis to get an upper bound on the cost of all n operations.
2. Use the accounting method to get an upper bound on the amortized cost of a single operation.

i	2^0 1	2^1 2	3	2^2 4	5	6	7	2^3 8	9	10	11	12	13	14	15	2^4 16
c_i	5	5.2	1	5.4	1	1	1	5.8	1	1	1	1	1	1	1	5.16

$$\sum_{i=1}^{n} c_i = (n - \log_2 n) + \sum_{i=0}^{\log_2 n} 5.2^i$$

$$= (n - \log_2 n) + 5 \frac{2(2^{\log_2 n + 1} - 1)}{2-1}$$

$$= n - \log_2 n + 10 (2n - 1)$$

$$\leq 2In - 10$$

upper bound: $O(n)$

2. For the accounting method we can use $\left(\frac{21n}{n}\right) = 21$ as the cost and we will calculate the table for

$n = 8$

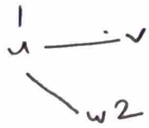
i	2^0 1	2^1 2	3	2^2 4	5	6	7	2^3 8
get	21	21	21	21	21	21	21	21
cost	5	10	1	20	1	1	1	40
rem	16	27	48	49	70	91	112	72

so, the balance is always ≥ 0

19

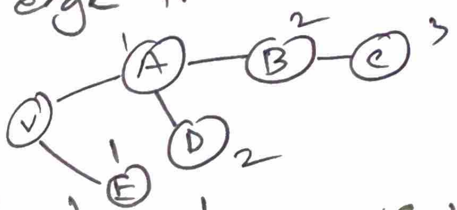
3 Graph Algorithms (20 Points)

Suppose we are given an undirected graph $G = (V, E)$ that represents a social media graph (vertices represent accounts and edges represent that the accounts are "friends"). For any vertex $v \in V$, we can consider how many *degrees of separation* v is from some other vertex. If a vertex u is friends with v, then v and u have one degree of separation. If a vertex w is not friends with v but is friends with some friend of v, then v and w have two degrees of separation. In this problem, the goal is that given any vertex v, we want to find a vertex in the graph that maximizes the degrees of separation from v. Give an algorithm that finds such a vertex and runs in $O(n + m)$ time.



we can run BFS and maintain a degreeCounter

We will run the BFS starting from v and mark on the every vertex that has a ~~edge~~ edge from v one degree friend.



As we move to the adjacent vertexes we will update the degreeCounter for his adjacent vertexes by adding 1 with the current vertexes degree of separation.

The runtime for this algorithm is O(V+E) or O(m+n).

0 1 1
V: A, E
1 2 2
A: B, D
2 3
B: C

what do you return?

20

* We can figure out every path from s to t and delete the maximum value after $100 mark, and then take the minimum weight path.

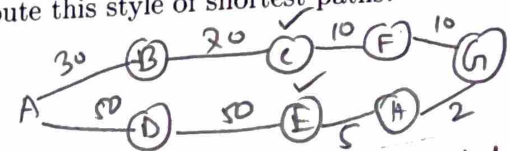
4 Shortest Paths (20 Points)

Suppose we have a directed graph $G = (V, E)$ where for each edge $(u, v) \in E$ we have a positive weight on the edge $w(u, v)$ that denotes how much money it costs to travel from u to v where the units are dollars (e.g., if $w(u, v) = 20$ then this means it costs \$20 to travel from u to v). We are interested in traveling from a vertex s to a vertex t for as cheaply as possible. This problem can be solved with Dijkstra's Algorithm as all of the edge weights are positive.

Then where do we need dijkstra?

Now suppose we have a setting where there is a promotion where if you have already spent \$100 on the path, then you can take one free trip (essentially you can make a single edge's weight be worth 0 on this path anytime after \$100 has been spent). If you have a given path p from s to t, then it is fairly easy to figure out the best way to use the free edge: you would use it on the most expensive edge after \$100 has been exceeded. It is more challenging to compute the path that leads to the minimum total cost in this setting.

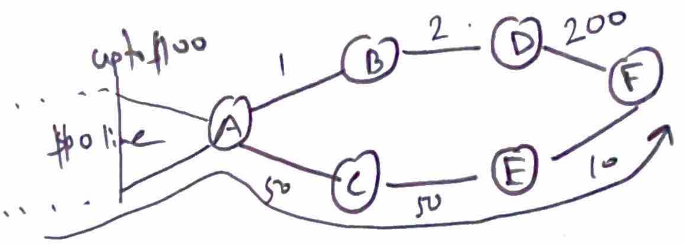
A reasonable idea to design an algorithm for this problem would be to start with something similar to Dijkstra's Algorithm and consider what modifications need to be made to handle the new setting. Why might Dijkstra's algorithm fail on this problem without modification, and what modifications need to be made to be able to compute this style of shortest paths?



let us consider a node that is in the path from s to t. let's call it t'. t' will be the node that is $100 cost way from s. we will run dijkstra again from all the t' and figure out the shortest path from t' to t. we will keep a max variable to record the maximum weighted edge and just subtract that from the total cost to get the ~~sto~~ minimum cost.

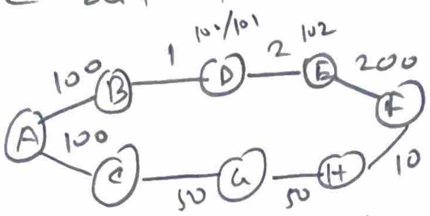
× this would avoid heavy edges but optimal may want one heavy edge

So, run dijkstra to find the shortest path from s to t. then find t' where the cost become >$100 and run dijkstra again from t' to t.



if dijkstra giving path that includes A→C→E→F we will just make one 50 go away. but the shortest path would have been A→B→D→F after deleting that 200 edge. This is where dijkstra fails. OK good

So, the possible solution would be to run dijkstra from t' making ~~each~~ one edge 0 in each iteration. and figure out the shortest path that way.



BD 0 CG 0 (make this choice
DE 0 GH 0 in every iteration
EF 0 HF 0 one at a time)

OK.

We will first find t' [B,C] using dijkstra. then run dijkstra (remaining edge time) each time making one edge 0.

We can optimize this by running a BFS initially making the weights 1: we will figure out all the paths that lead from s to t. then find t' using dijkstra, then from t' to t consider every path that leads to t.

Another optimized way would be save two values when your dijkstra goes beyond the $100 mark: One value will be the weight considering the previous edge's original weight and another value would be considering the value as 0.