30

1 Greedy Algorithms (30 Points)

Suppose we are organizing tours of the Alamo. Customers can reserve a time slot for a tour, and once one customer reserves that time slot, no other customer can take that time slot (no more than 1 tour going on at the same time). We need to ensure that there is a tour guide to lead the tour for each of our reservations. A tour guide can only stay for 5 time slots before they must leave. We would then need to bring in another tour guide for later tours.
Given a schedule with n time slots (where any subset of the time slots could actually have reservations), what is the minimum number of tour guides we need to cover all of the scheduled tours? Give a greedy algorithm to compute the optimal number of guards. Argue why your algorithm is guaranteed to produce an optimal solution.

Greedy Algorithm for time slots :-

Let time[ ] be the time slots booked.

⇒ We will assign the first booked slot to the guide, then we jump '+5' slots to the 7th, (guide stays for 5 time slots) then we check if the slot is booked or not. If booked, assign new guide and repeat process. If not booked, move to next slot until we find booked slot until slots completed.

Correctness :- Suppose, there is an optimal solution which have different selection methods, then maximum number of slots assigned would not be enforced as it does in my method. If so, then we can replace that solution with the above solution to get new optimal solution.

(30)

2 Amortized Analysis (30 Points)

Consider the dynamic array as we considered in class (like a C++ vector, Java ArrayList, etc.). In class, we supposed that we doubled the size of the array every time the array filled up. Now instead of doubling the size of the array (increase by a factor of 2), suppose we increase the size by a factor of 4.

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.

1) Aggregate analysis:- $(4^2 = 16)$

Array list:- 1 2 3 4 5 6 7 8 9 10 _ _ _ _ _ _ 17

1 4 4 4 16 16 16 16 16 16 _ _ _ _ _ _ 64

1 2 1 1 5 1 1 1 1 1 _ _ _ _ _ _ 16

$$\sum_{i=1}^{n} c_i = n + \sum_{j=1}^{\log_4 4n} 4^j = n + \frac{4^{(\log 4n + 1)} - 1}{4-1} < 3n$$

$$= n + \frac{4^{(\log 4n + 1)} - 1}{3} < 3n$$

2) Accounting method:-

Each operation pays \$3, \$1 for now and \$2 for later quadruple. On an expensive operation, 3/4 of the operations are new and t_i is old. The new operations will have saved each one. So, we can afford the operation without going below '0'.

(20)

3 Graph Algorithms (20 Points)

Suppose we have an undirected, unweighted graph G. For any two vertices u, v of G, let $\delta(u, v)$ denote the length of the shortest path between u and v where the length of the path is determined by the number of edges in the path. The *diameter* of a graph is defined to be $\max_{u,v} \delta(u, v)$. In other words, if we consider all pairs of vertices, it's the pair that maximizes their shortest path distance. Give an algorithm to compute the diameter of G. The algorithm can run in $O(nm)$ time.

Given, G is a undirected, unweighted graph.

→ $\delta(u, v)$ denote length of shortest path between u and v.

Algorithm :-

We can use BFS (Breadth first search) algorithm to determine the shortest path distances since edges are unweighted. Run BFS for each node or vertex, remembering the larger shortest path we encounter.

→ This is the $O(nm)$ time algorithm that computes the diameter of G.

10

4 Minimum Spanning Trees (20 Points)

Suppose we have a graph G with non-negative weights on the edges, and we wish to compute a minimum spanning tree for this graph. That is, the edge weights represent the cost to connect two vertices together, and we want to build a connected network that spans the graph. Now suppose that we have a "free edge" coupon, where we can get any edge for free. So for example, if a regular MST used edges that weighed 1, 2, 3, and 4, it would have a cost of 10. But in this setting, we could use the coupon on the edge that costs 4 and pay only 6. Give an algorithm to compute an MST in this setting, and briefly argue why your algorithm is correct.

Given, a graph G with non-negetive weights on the edges,

what order? Any?

⇒ Fix an ordering of edges. For each edge (u,v) in the order, check to see if (u,v) gives us a cheaper way to connect to v than we have seen before. If so, switch the edge to v to be (u,v). Then switch until the edges converge.

⇒ Then use the "free edge" coupon to the expensive edge so the cost will be ~~minimum~~ Minimum. We obtain a minimum spanning tree.

Okay, How many times through the order? Cycles?

⇒ Suppose if the largest weighted edge is removed (use free coupon) then the cost cannot be minimum because the path may be different. So, after finding the path and edge is converged, we can use "free edge" coupon to the expensive edge in the shortest path, that gives the minimum spanning tree.