CS 5633 Analysis of Algorithms – Fall 18

Exam 3

NAME: JISHNU BANERJEE

- 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

1 Greedy Algorithms (30 Points)

30

Suppose we are given a collection of n intervals $I_1, I_2, \ldots I_n$ with integer endpoints. Let I_j^ℓ denote the left endpoint of I_j, and let I_j^r denote the right endpoint of I_j. We say a set of integer points P *stabs* the n intervals if each interval contains some point of P. Give a greedy algorithm which computes a set of points which stabs the n input intervals of minimum cardinality. Why is the algorithm correct? What is the running time of your algorithm?

- Example: Let $I_1 = [1, 10], I_2 = [4, 15]$, and $I_3 = [8, 20]$. The set $P_1 = \{2, 14\}$ stabs the intervals (I_1 contains 2 and I_2 and I_3 contain 14). However the set $P_2 = \{9\}$ also stabs the intervals (all three intervals contain 9), and we would prefer P_2 to P_1 since it contains fewer points.

First, we will sort $I_1, I_2, I_3 \ldots I_n$ according to its right endpoints. ~~Then we will run a loop over all the intervals. In each loop we will check~~

```
for(i=0; i<I.size(); i++) {
    if cover in I[i]: continue;
    else:
        Result.push(I[i].endpoint);
        cover = I[i].endpoint;
} Return Result;
```

Initialization
cover = -infinity
Result = { }

Why correct: The algo can be wrong if I pick ~~any~~ more than one point for an interval. I'm not doing it. Again, if we miss any interval, it can be wrong. But I am looping through all the interval. It's not possible. Moreover, I am picking always the right most point of an interval (If necessary). This approach is ~~taking the high~~ covering the highest possible intervals as they are sorted.

Output: {10} is returned in the solution of me.

Complexity: Greedy Algo runs in $O(n)$ time. (If we consider the Intervals are already sorted).

2 Amortized Analysis (30 Points)

(30)

In the dynamic array data structure as discussed in class, once the array becomes full we doubled the length of the array and copied the previous elements to the new array. Now suppose that instead of doubling the size of the array, we *quadruple* it. So the sizes of the array would be 1, 4, 16, 64, etc. This question will consider the amortized cost of making n insertions into this data structure.

1. Perform an aggregate analysis to obtain a bound on the sum of the cost of n operations.

$$\sum_{i=0}^{n} cost[i] = n + \sum_{j=1}^{\log_4 n - 1} 4^j = n + \frac{4^{\log_4 n + 1} - 1}{3} = n + \frac{4n - 1}{3} = \frac{7n-1}{3}$$

$$= \frac{7}{3}n - \frac{1}{3}$$

✓

c[i]	1	2	1	1	5	1	1	1
sz[i]	1	4	4	4	16	16	16	16
i =	1	2	3	4	5	6	7	8

2. Use the accounting method to determine the amortized cost of a single operation.

So, per one single operation, the cost $= \frac{7}{3} - \frac{1}{3n}$

we just can drop the $\frac{1}{3n}$ as it is too small for bigger n's, and take the ceiling of $\frac{7}{3}$ which is 3.

So, Amortize cost for single operation = 3

✓

given[i]	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
c[i]	1	2	1	1	5	1	1	1	1	1	1	1	1	1	1	1	17	1
rem[i]	2	3	5	7	5	7	9	11	13	15	17	19	21	23	25	27	13	15
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Chart for upto i = 18.

3 Graph Algorithms (20 Points)

20

Suppose $G = (V, E)$ is a *connected*, undirected graph. That is, for any two vertices $u, v \in V$, there is a path that connects u and v. An edge $e \in E$ is called a *bridge* if its removal disconnects the graph. Give an algorithm that determines if G contains an edge. Make your algorithm as efficient as possible. What is the running time of your algorithm?

I will run a loop over all the edges. Remove the edge. Then, run (BFS/DFS) traversal from both the vertices connected to the edge each time. If in any pair of vertices, (x, y) we get no way to reach from x to y, we have a bridge. If we don't get so, we don't have any bridge.

Complexity: We are running traversal for every edge. So the running time would be

$$m(m+n) = m^2 + mn$$
$$= O(m^2) \quad [\text{As, } m > n \text{ in most cases}]$$

(19)

4 Shortest Path (20 Points)

Given a graph $G = (V, E)$ with weights on the edges (positive and negative), let $\delta(u, v)$ denote the length of a shortest path connecting u and v. The *diameter* of G is defined to be $\max_{u,v \in V} \delta(u, v)$ (note that the diameter is a number). Give a polynomial-time algorithm to compute the diameter of G. Your algorithm can use shortest path algorithms that we discussed in class as a subroutine of your algorithm. Make your algorithm as efficient as possible. What is the running time of your algorithm?

As there are some negative weight paths, we will use bellman-ford algorithm. I will run bellmanford algorithm for n times, from all the nodes. Then, we will get all the shortest possible paths from for every pair of nodes. I will just pick the maximum one and that is the diameter of the graph G.

Complexity: As I am running bell-manford algo for n times, the complexity will be $O(mn^2)$.

If we consider that there'll be no negative cycle, then we just can use the single pass bellmanford for DAG which will be $(m+n)n = n^2 + mn = O(mn)$.

[As $m > n$ in most cases]

Edmonds-karp faster