CS 161 W19: Recitation 5 Solutions

February 2019

Exercise 0

Solve the following recurrences — that is, get a tight bound of the form $T(n) = O(f(n))$ for the appropriate function f. You may use the master theorem if it applies. Ignore floors and ceilings, and assume $T(0) = T(1) = 1$.

(a) $T(n) = T(n/5) + 64n$

(b) $T(n) = 4T(n/4) + n$

(c) $T(n) = T(n-2) + 9n$

Solution 0

(a) Using the master theorem, we have $T(n) = T(n/5) + O(n)$ so $a = 1, b = 5, d = 1$, meaning $a < b^d$. Thus $T(n) = O(n^d) = O(n)$.

(b) Using the master theorem, we have $T(n) = 4T(n/4) + O(n)$, so $a = 4, b = 4, d = 1$, meaning $a = b^d$. Thus $T(n) = O(n^d \log n) = O(n \log n)$.

(c) In this case the master theorem doesn't apply. We solve by unrolling the recurrence.

$$\begin{aligned} T(n) &= T(n-2) + 9n \\ &= T(n-4) + 9(n-2) + 9n \\ &= T(0) + 9(2) + 9(4) + \cdots + 9(n-4) + 9(n-2) + 9n \\ &= 1 + \sum_{i=1}^{n/2} 9(2i) \\ &= 1 + 18\sum_{i=1}^{n/2} i = 1 + 18\frac{(n/2)(n/2+1)}{2} = 1 + 9(\frac{n^2}{4} + \frac{n}{2}) \end{aligned}$$

Thus $T(n) = 1 + 9(\frac{n^2}{4} + \frac{n}{2}) = O(n^2)$.

Exercise 1

Random sort (also known as bogosort) attempts to sort a list by first checking if it sorted, and if not it randomly permutes all the elements and then checks again. It repeats this process until it successfully generates the sorted list. In this problem, assume that the operation of randomly permuting n elements takes $O(n)$ time, and assume that the list we are attempting to sort consists of distinct integers. What is the expected runtime of the algorithm for worst-case input? Is the runtime of the algorithm bounded?

Solution 1

A worst-case input is any unsorted array. Let X be a random variable that is the number of times that the algorithm permutes the array. The algorithm chooses the right permutation with probability $\frac{1}{n!}$. Then, X is a geometric random variable with parameter $\frac{1}{n!}$. Thus, $E[X] = n!$

Each permutation takes $O(n)$ time (this happens $n!$ times in expectation). At the beginning of the algorithm and after each permutation, the algorithm checks if the array is sorted in time $O(n)$ (this happens $n! + 1$ times in expectation).

The total runtime is

$$(2 \cdot n! + 1) \cdot O(n) = O(n \cdot n!)$$

However, the runtime is not bounded because in the worst case the random permutation never sorts the list, i.e., the worst-case runtime is ∞.

Exercise 2

A review on hashing: Assuming you are using a uniformly random hash function,

1. if you were to hash n items into $b = n$ buckets, how many items would you expect to be hashed to the 3rd bucket? (Assume $n > 3$.)
2. in expectation, what is the number of items per bucket?
3. what is the probability that the 3rd bucket has exactly 3 items?
4. what is the big-O runtime of INSERT, DELETE, SEARCH, and SELECT where, for example, SELECT(5) finds the 5th smallest element in the hash table?

Solution 2

1. 1, because each item is hashed into bucket 3 with probability $1/n$.
2. 1, by symmetry.
3. This is the probability that for some choice of 3 items, they are hashed into bucket 3 and every other item is hashed into a different bucket, i.e., $\binom{n}{3}\left(\frac{1}{b}\right)^3\left(\frac{b-1}{b}\right)^{n-3}$.
4. INSERT, DELETE and SEARCH have expected runtime $O(1)$, but DELETE and SEARCH have worst-case runtime $O(n)$. SELECT is $O(n)$ by taking all of the elements out of the hash table and into an array, and using the median-of-medians-based SELECT algorithm from lecture 4.

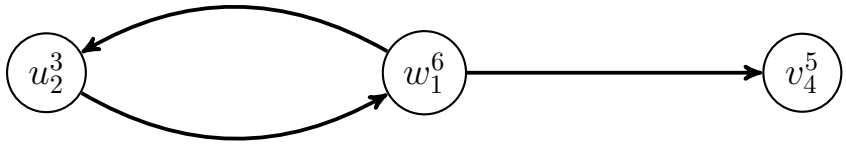
Exercise 3

True or false:

1. If (u, v) is an edge in an undirected graph and during DFS, $finish(v) < finish(u)$, then u is an ancestor of v in the DFS tree.
2. In a directed graph, if there is a path from u to v and $start(u) < start(v)$ then u is an ancestor of v in the DFS tree.

Solution 3

1. **True**. When we do DFS, once we visit a node, we immediately recurse on each of its unvisited neighbors before marking the node as finished. In this case, we must have visited u, then recursed on v before marking u as finished, making u an ancestor of v. The only other way we could have $finish(v) < finish(u)$ is if v were to be finished before u was visited at all. But since (u, v) is an edge, i.e., u was an unvisited neighbor of v, this is impossible.
2. **False**. Consider the following case:



Note that 1. had an edge between u and v while 2. does not specify such an edge (only a path).

Exercise 4

You have n boxes. The i-th box has dimensions $w_i \times h_i$. Box i can fit inside box j if and only if $w_i < w_j$ and $h_i < h_j$. A sequence of boxes $b_1, b_2, ..., b_k$ form a chain if box b_i fits inside box b_{i+1} for each $1 \leq i < k$. Design an algorithm which takes as input a list of dimensions $w_i \times h_i$ and returns a longest possible chain of boxes. You must construct a directed graph as part of your solution.

Solution 4

Construct a directed graph whose vertices are boxes, and such that there is an edge (v_i, v_j) iff box v_i fits inside box v_j. Notice that this graph is a DAG (you can only go one direction between boxes). Our goal is now to find the longest path.

Linearize (topologically sort) the graph, so whenever there is an edge from v_j to v_i , $j < i$. Recall that this can be accomplished by running DFS on the graph, and ordering the nodes in decreasing order of finish time.

For every node v_i, let ℓ_i be the length of the longest path ending at v_i. We can compute ℓ_i as follows:

$$\ell_i = 1 + \max_{(v_j, v_i) \in E} \ell_j$$

Because we have linearized the graph, ℓ_i depends only on ℓ_j for $j < i$. So we can compute the ℓ_i values in order. The answer is $\max_{i=1}^{n} \ell_i$.

Exercise 5

Suppose that G is a graph with $2n$ nodes and no triangles (cycles of length 3). G is a proper graph, *i.e.* it has no self-loops or multiple edges between the same pair of nodes. Use induction to prove that G has at most n^2 edges.

Solution 5

We prove by induction.

Base case: If $n = 1$, there are 2 nodes and clearly no more than 1 edge (since it is a proper graph).

Inductive hypothesis: Suppose that for $n = k$, any proper graph G with $2k$ nodes and no triangles has at most k^2 edges.

Inductive step: Consider the case $n = k + 1$. Let G be a proper graph with $2k + 2$ vertices and no triangles. If G has no edges, we are done. Otherwise, there exists an edge (u, v), where u and v are vertices in G. Let G' be the graph obtained from G by removing the vertices u and v and all edges adjacent to them. G' is a proper graph with no triangles since G is a proper graph with no triangles and we only removed edges and vertices. G' has $2k$ vertices, thus the inductive hypothesis applies to it. Hence, the number of edges in G between vertices that are not u or v is at most k^2. Now consider the edges that go from u and v to vertices in G'. There is no vertex w in G' that is connected to both u and v since that would form a triangle in G (remember that (u, v) is an edge in G). Therefore any vertex in G' can be connected to only one of u and v, which means that the total number of edges from u and v to vertices in G' is at most $2k$. Putting it all together, we see that the total number of edges in G is at most

$$k^2 + 2k + 1 = (k + 1)^2$$

Conclusion: A graph with $2n$ nodes and no triangles has at most n^2 edges.