HKN CS 374 Midterm 2 Review

By: Tim Klem, Mahir Morshed, Noah Mathes

Writing Proofs in an Exam

- Figure out which techniques allow you to focus more on solving the problem without much overhead over how to deliver it
- Know what your rubrics want
- Use the list guides for DP and graphs as templates
    - DP - JeffE's March 2 lab
    - Graphs - JeffE's March 14 lab
- Lead off with your intuition - you might catch errors while writing!
- Write pseudocode *last* (or maybe not at all, if appropriate)

Solving Recurrences

Use recursion trees! Three things to determine:

1. How much work is done in the root
2. The relationship (ratio, difference) between the work done on a node and the sum of the work done on its children
3. Total number of levels in the recursion tree
   a. May need to bound this quantity

A paragraph synthesizing these three pieces should suffice.

Divide and Conquer

- Partition your problem into complementary problems
    a. That are smaller instances of the same problem
    b. That you know how to solve
    c. Just small enough so that you can use algorithms of low time complexity
    d. Just large enough so that you can bite the bullet and still achieve good runtime
- Partitions we’ve seen so far:
    a. Given in the data structure (children of nodes in trees)
    b. The middle of a sorted array (binary search)
    c. Making your own via linear-time select (deterministic quicksort)
    d. Choosing convenient element(s) to constrain the input (mergesort)
- Usually these exploit some sort of ordering in the data

NETID: NAME:

3 (20 PTS.) Let $A[1 \,..\, n]$ be an array of n distinct numbers that is not sorted, but it is k-scrambled. Here, being k-*scrambled* means that for any i, we have that $i - k \leq \mathrm{rank}(A[i]) \leq i + k$, where $\mathrm{rank}(A[i])$ is the location of $A[i]$ in the sorted version of A.

You are given the value of k as part of the input, and a number x. Describe an algorithm, as fast as possible, that reports whether or not x is stored in A somewhere.

What is the running time of your algorithm? Explain (shortly) why your algorithm is correct.

(Hint: What can you can say about the relation between $\mathrm{rank}(A[i])$ and $\mathrm{rank}(A[i + 3k])$?)

Dynamic Programming

- Clearly identify the subproblems
  - May require reformulation of the original problem
- Identify recurrence based on these subproblems
  - Often the base case is trivial while multiple formulations may exist of the recursive step
- Identify the specific term in the recurrence corresponding to the original problem
- *Optionally* unroll the recurrence into an iterative form
  - Usually having the recurrence and the specific term to be calculated are enough for exam purposes
  - Requires defining memoization array dimensions
- *Optionally* write pseudocode

From A lecture’s Piazza #919 (credits to Philip Shih)

You need 6 things for a dynamic programming problem:

1. An English description of the underlying recurrence you're going to evaluate
2. The details of that recurrence
3. The top level call to your function to get the final answer
4. The memoization data structure
5. An evaluation order
6. Runtime analysis

You can either give these things individually or embed them in pseudocode. Pseudocode covers numbers 2,3,4, and 5. If you don't give pseudocode, you need to **explicitly** say what each of those things are. You still need to do 1 and 6 separately even with pseudocode.

NETID: NAME:

4 (20 PTS.) There are n people living along Purple street in Shampoo-Banananana. The n people live in locations $1, \ldots, n$ along Purple street (which is as straight as a ruler),

It is time for redistricting Purple street. A district can have between Δ and 2Δ people living in it, for some prespecified parameter Δ (where $n/3 > \Delta > 0$). A district is a consecutive interval along Purple street. Every person is assigned to a district containing it. The districts are disjoint.

For every person in Purple street, you know their vote in the last election. Specifically, you are given an array $v[1 \,..\, n]$, where the vote of the ith person is $v[i]$, which is either equal to 0 or 1. A set $S \subseteq \{1, \ldots, n\}$ of people is t-good, if $|\#_0(S) - \#_1(S)| \leq t$, where $\#_0(S)$ (resp. $\#_1(S)$) is the number of people in S that voted for 0 (resp 1) in S.

Describe an algorithm, as fast as possible, that given $\Delta, v[1 \,..\, n]$ and t as input, outputs TRUE if there is a way to redistrict Purple street so that every district is t-good (the algorithm outputs FALSE otherwise). What is the running time of your algorithm? The algorithm you provide should be iterative (i.e., please do not use dictionary/hashing or recursion).

Graphs - Key Algorithms

- Whatever-first search - $O(V + E)$ time
- Dijkstra's - single-source shortest path tree in $O(E + V \log V)$ time,
  - for graphs with only non-negative weight edges
  - greedy search using a cost function
- Bellman-Ford - single-source shortest path tree in $O(EV)$ time, for graphs of any real-valued weight edge
  - DP formulation
- Floyd-Warshall - all-pairs shortest path in $O(V^3)$ time
- Topological sort - sort the vertices of a DAG so that for every edge $u \rightarrow v$, u is before v in the array, done in $O(V + E)$ time
- Tarjan's algorithm - reduce a graph to a DAG of connected components in $O(V + E)$ time

NETID: NAME:

2 (20 PTS.) (**Seen in lab**) You are given a directed graph G with n vertices and m edges. Every edge $x \to y \in \mathsf{E}(\mathsf{G})$ has a weight $w(x \to y)$ associated with it. Here, exactly *one* edge $u \to v$ has negative weight (all other weights are positive numbers). Describe an algorithm, as fast as possible, that decides if there is a negative cycle in G, and if not, it computes the shortest path between the two given vertices s and t in G. What is the running time of your algorithm? Argue that your algorithm is correct.

NETID: NAME:

5 (20 PTS.) You are given a graph G with n vertices and m edges (with $m \geq n$), and a parameter k. In addition, for every vertex $v \in \mathsf{V}(\mathsf{G})$, you are given a flag $r(v)$, which is 1 if v is a *router*, and 0 otherwise. A vertex is ***safe***, if there are at least k distinct routers it can reach (here, think about k as being a small number compared to n).

[Do not use hashing or dictionary data-structure in solving this problem.]

5.A. (10 PTS.) For the case that G is a DAG, describe an algorithm, as fast as possible, that computes all the safe vertices of G. What is the running time of your algorithm? Argue (shortly) that your algorithm is correct.

5.B. (10 PTS.) Describe an algorithm, as fast as possible, for the case that G is a general directed graph. What is the running time of your algorithm?

Hyperlinks

https://courses.engr.illinois.edu/cs374/sp2018/A/labs/lab7bis.pdf

https://courses.engr.illinois.edu/cs374/sp2018/A/labs/lab9.pdf