CSE 421 Section 6

Midterm Review

Administrivia

Announcements & Reminders

- HW4
  - If you think something was graded incorrectly, submit a regrade request!

- HW5
  - Was Due Yesterday, Wednesday Nov 1 @ 11:59 pm

- Midterm Exam: **Wednesday November 8@ CSE2 G20 @ 6-7:30 pm**
  - Make sure you have it saved on your calendar!
  - If you can't make it, let us know and we will schedule a conflict exam!
  - If you are sick on the day of, let us know and we will schedule a conflict exam!

Greedy Algorithms

Problem 1 – Interval Covering

You have a set, $\mathcal{X}$, of (possibly overlapping) intervals, which are (contiguous) subsets of $\mathbb{R}$. You wish to choose a subset $\mathcal{Y}$ of the intervals to cover the full set. Here, cover means for all $x \in \mathbb{R}$ if there is an $X \in \mathcal{X}$ such that $x \in X$ then there is a $Y \in \mathcal{Y}$ such that $x \in Y$.

Describe (and prove correct) an algorithm which gives you a cover with the fewest intervals.

Work through this problem with the people around you, and then we'll go over it together!

Problem 1 – Interval Covering

Correctness:

Problem 1 – Interval Covering

Running Time:

Divide and Conquer

Problem 2 – Binary Search Variant

Let $A[1..n]$ be an array of ints. Call an array a **mountain** if there exists an index i called "the peak", such that:

$$\forall 1 \leq j < i(A[j] < A[j+1])$$
$$\forall i \leq j < n(A[j] > A[j+1])$$

Intuitively, the array increases to the "peak" index i, and then decreases. Note that either of these conditions could be vacuous if the peak is index 1 or n (e.g., a decreasing array is still a mountain).

Problem 2 – Binary Search Variant

a) Given an array $A[1..n]$ that you are promised is a mountain, find the index peak index.
b) Can you design an algorithm with the same running time that also determines whether a given array is a mountain (and if it is, finds the peak)?

Work through this problem with the people around you, and then we'll go over it together!

Problem 2 – Binary Search Variant

a) Given an array $A[1..n]$ that you are promised is a mountain, find the index peak index.

Problem 2 – Binary Search Variant

b) Can you design an algorithm with the same running time that also determines whether a given array is a mountain (and if it is, finds the peak)?

Dynamic Programming

Problem 3.1 – Write a Recursive Solution

a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation

Problem 3.1 – Write a Recursive Solution

b) Write a recurrence for solving the problem you defined in the last part (the recurrence is for the answer, not the running time).

Problem 3.1 – Write a Recursive Solution

c) What is your final answer (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

Problem 3.1 – Write a Recursive Solution

d) Give a brief justification for why your recurrence is correct. You do not need a formal inductive proof, but your intuition will likely resemble one.

Problem 3.2 – Write and Analyze the Dynamic Program

a) Describe the parameters for the subproblems in the recursive calls for your algorithm and how you can store their solutions.

b) Describe a computation order for those subproblems that allows an iterative solution.

c) Write the pseudocode for an iterative algorithm

d) State and justify the running time of your iterative solution.

Work through this problem with the people around you, and then we'll go over it together!

Problem 3.2 – Write and Analyze the Dynamic Program

a) Describe the parameters for the subproblems in the recursive calls for your algorithm and how you can store their solutions.

Problem 3.2 – Write and Analyze the Dynamic Program

b) Describe a computation order for those subproblems that allows an iterative solution.

Problem 3.2 – Write and Analyze the Dynamic Program

d) State and justify the running time of your iterative solution.

Graph Algorithms

Problem 4 – Running Out of Rooms

a) Describe an algorithm to solve this problem.

Problem 4 – Running Out of Rooms

b) Give some intuition for why your algorithm is correct. (Don’t write a full proof of correctness).

Problem 4 – Running Out of Rooms

c) If your list has n people, what is the worst-case running time. Briefly (1-2 sentences) explain.

Stable Matching

Problem 5 – Practice a Reduction

You have a set of r riders and h horses, but unfortunately, $2h < r < 3h$, i.e. there are many more riders than horses. You wish to setup a set of 3 rides which will give each rider exactly one chance to ride a horse. To keep things fair among the horses, you wish for each to have exactly 2 or 3 rides.

Because it's winter, by the time the third ride starts it will be very dark, so every rider would prefer any horse on the first two rides over being on the third ride. Between the first two rides, each rider doesn't have a preference over time of day, and has the same fixed preference over horses. If a rider must be on the third ride, it has the same preference list for that ride as well. Each horse has a single list over riders, which doesn't change by ride. Since horses love their jobs, they prefer to be one of the horses on the third ride instead of not being on a ride.

Design an algorithm which calls the following library exactly once and ensures there are no pairs r, h which would both prefer to change the matching and get a better result for themselves.

`BasicStableMatching`
Input: A set of 2k agents in two groups of k agents each. Each agent has an ordered preference list of all k members of the other group.
Output: A stable matching among the 2k agents.

Problem 1 – Practice a Reduction

a) Give a 1-2 sentence summary of your idea.
b) Give the algorithm you're going to run.
c) Give a 1-2 sentence summary of the idea of your proof.
d) Write a proof of correctness.
e) Give the running time of your algorithm; briefly justify (1-3 sentences). You can assume `BasicStableMatching` has a runtime of $\theta(k^2)$.

Work through this problem with the people around you, and then we'll go over it together!

Problem 1 – Practice a Reduction

(a) Give a 1-2 sentence summary of your idea.

Problem 1 – Practice a Reduction

b) Give the algorithm you're going to run.

Problem 1 – Practice a Reduction

c) Give a 1-2 sentence summary of the idea of your proof.

Problem 1 – Practice a Reduction

d) Write a proof of correctness.

Problem 1 – Practice a Reduction

e) Give the running time of your algorithm; briefly justify (1-3 sentences). You can assume `BasicStableMatching` has a runtime of $\theta(k^2)$.

That's All, Folks!

Thanks for coming to section this week!
Any questions?