Recitation 7 — Midterm Feedback and Homework 5

Parallel and Sequential Data Structures and Algorithms, 15-210 (Fall 2012)

October 10, 2012

1 Topics

- Assignment 5 will be out soon. You will be required to make some modifications to Dijkstra's Algorithm.
- The first midterm has been graded. We will be coving common mistakes and fielding questions.
- DFS and Topo-sort are really cool.
- Questions? Comments? Pearls of wisdom?

2 Exam feedback

The average and median on the first midterm were 62 and 63, respectively. There were two 90's, four 80's, 21 70's, 27 60's, 20 50's, 19 40's, and four 30's. As many of you have guessed, this exam was pretty tough. Before jumping in to the main topics of this recitation we will go over some of the common mistakes that were made on each of the questions.

Short Answers :

Many people had difficulty analyzing the cost of a call to `reduce` with a non-trivial combine function. The proper way to approach this problem was to transform `reduce`'s call tree into a recurrence, which could then be solved. For a combining function, f, with $O(\sqrt{n_1 + n_2})$ work, the recurrence for `reduce` f b S would be $W(n) = 2W(n/2) + O(\sqrt{n/2 + n/2})$, which means that $W(n) \in O(n)$.

Another common mistake was assuming that the work and span of `iter` f b S are the same. This is not true: Even though each call to f is made sequentially, the span of f itself must also be accounted for if f is non-trivial.

Many students incorrectly stated that only one BFS level would be target of a vertex's out-neighbors or that only one level could supply a vertex with in-neighbors. This is not true (consider the case of an undirected graph implemented under the hood as a directed graph for a counter-example). In general, edges in directed graphs can go from any BFS level to the same or a level closer to the source.

The short answers are usually easier than the algorithm questions, but you still need to approach them carefully!

Coffee Shops :

A common mistake was to run BFS from each source in parallel. Since each BFS explores the whole graph separately, it duplicates work and does not meet the $O(m \log n)$ work bound. The correct way was to put all the coffee shops on the initial frontier. In the second part, many students reversed the roles of coffee shops with intersections and used intersections as the source vertices. This incorrectly finds the path lengths from the closest intersection to each coffee shop; in most cities that intersection would be one block away. The correct way was to transposed the graph and keep the coffee shops as the source vertices. You also needed to give the cost bounds for transposing a graph; a few students incorrectly claimed it could be done in linear work.

Voting Season :

A number of people tried to solve this problem without using the given function, `checkMajority`, during the combine step. The idea was to use the recursive calls to `checkMajority` to limit the pool of possible winners to at most two candidates using divide and conquer and to then check if either was the majority winner. Some students tried to map `checkMajority` over the input seqeunce. This had $O(n^2)$ work and did not meet the cost bounds.

Finding Your Mate :

Most students recognized that a scan was the appropriate way to solve `parenDepth`, but many did it with a function that was not associative, or even worse, one which was not of the type `'a * 'a -> 'a`. Remember, make your scan functions as simple as possible and handle as much work as you can in the pre or post-processing steps.

Median :

Finding the closed form of the span was, as the test suggested, non-trivial. The correct way to approach this was to assume that the answer would be a power law (there isn't a rigorous way to justify this guess other than noting that span doesn't seem to be decreasing fast enough to warrant a log). The inductive step of our (somewhat informal) substitution proof would then look like this:
Assume that $S(m) = k_0 m^x - k_1 \log(m) - k_2$ is true for all $m < n$. By this assumption,

$$\begin{aligned} S(n) &= \kappa \log(n) + S\left(\frac{n}{5}\right) + S\left(\frac{7n}{10}\right) \\ &= \kappa \log(n) + \left(k_0 \frac{n^x}{5^x} - k_1 \log\left(\frac{n}{5}\right) - k_2\right) + \left(k_0 \frac{7^x n^x}{10^x} - k_1 \log\left(\frac{7n}{10}\right) - k_2\right) \\ &= k_0 \left(\frac{7^x}{10^x} + \frac{1}{5^x}\right) n^x + (\kappa - 2k_1)\log(n) + (k_1 \log(5) - k_1 \log(7) + k_1 \log(10) - 2k_2) \end{aligned}$$

This might look disheartening, since we want that mess to be equal to $k_0 n^x - k_1 \log(n) - k_2$ but let's see it through to its conclusion. By splitting apart our basis functions we see that we need choose our free parameters such that the following is true:

$$\begin{aligned} k_0 &= k_0 \left(\frac{7^x}{10^x} + \frac{1}{5^x}\right) \\ k_1 &= 2k_1 - \kappa \\ k_2 &= 2k_2 - k_1(\log(5) - \log(7) + \log(10)) \end{aligned}$$

We solve the bottom two equations and find that $k_1 = \kappa$ and $k_2 = \kappa(\log(5) - \log(7) + \log(10))$. We also note that the top equation is only true when $\frac{7^x}{10^x} + \frac{1}{5^x} = 1$. This is only true for one value of x (roughly 0.8).
Since k_0 is still a free parameter, we can choose it such that we pass the base case.

MKCSS :

Aside from the apocalypse recurrence in the previous question, MKCSS was probably the most difficult question on the exam. The solution revolved around noticing that fewer than n subsequence sums needed to be evaluated and that, if you preprocess the input sequence with scan using addition, the sum of a subsequence can be evaluated in $O(1)$. Many students attempted a divide and conquer solution with several calls to `subseq` during the combine step. If these approaches returned correct answers, they would invariably have $O(nk)$ work.
An important skill to have when taking exams in 210 is the ability to realize when a problem *can't* be solved by a certain approach. If a certain approach seems to have unavoidable costs associated with it, try a different approach instead of trying to make a square peg fit in a round hole.
On a more superficial note, several students tried to solve this problem with `iter`. In general, you should avoid `iter` unless you are sure that the sequential cost is acceptable. Often times using `iter` will avoid the challenging parts of a problem, meaning even an approach that returns correct answers will receive few points.

3 Homework 5

For Homework 5 you will write four variations of Dijkstra's algorithm:

- Dijkstra's algorithm with multiple source vertices.
- Dijkstra's algorithm with multiple target vertices.
- Dijkstra's algorithm with an additional distance metric used when selecting the next vertex to search. This is know as the A* heuristic.
- Dijkstra's algorithm modified to accept negative edge weights.

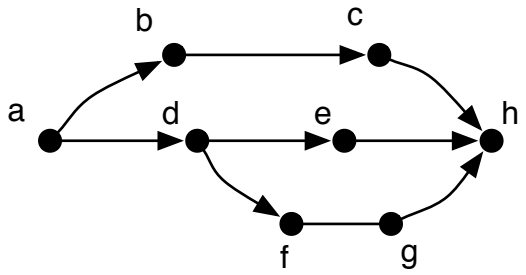
A* is the most interesting (and practical!) part of this homework and deserves a second mention. As stated above, when selecting the next vertex to add to the set of visited vertices a distance metric is used to estimate which candidates are closer to the target. An intuitive example would be a search across the Euclidean plane with nodes representing points and edge weights representing distances. The estimation metric would just be the straight-line distance between a given node and the target. This metric will make Dijkstra's algorithm prioritize nodes that are close to the goal over those that are far away.

4 DFS

TA note: Use pictures to prove that when `exit` is called for some vertex, v, in a DAG, `exit` has been called by all the vertices reachable from v. This was already done in the lecture notes.

5 Topological Sort Example

Below is an example of a DAG that we might want to do a topological sort over:



A possible call order would be to start the DFS on the top path. This hits all the vertices down to h with no branch-offs. When we exit each vertex we add it to our list of vertices, meaning that when we get back to a, our list is `[b, c, h]`. Before exiting a we need to search each of its children, so we move down to d. If the next path that we search down is the middle path, only e will be added to our list because our DFS has already touched h. We do the same process for the bottom path, so right before d exits, our list is `[f, g, e, b, c, h]`. Next we exit d and add it to the list, then we exit a and add it to the list, too. Our final list is `[a, d, f, g, e, b, c, h]`