Recitation 8 — Analyzing (Randomized) Algorithms

Parallel and Sequential Data Structures and Algorithms, 15-210 (Fall 2011)

October 19, 2011

Today's Agenda:
- Announcements
- Cost Aggregation
- Probability

1 Announcements

- Assignment 6 is coming out tomorrow and is due next Thursday.
- Questions about homework or class?

2 Cost Aggregation

Let's start out by looking at a familiar example: the `makeGraph` function from Homework 4. Remember that `makeGraph` is a function takes an edge sequence (`(vertex, vertex, edgeLabel) seq`) and produces a graph represented as a vertex-table table (`edgeLabel vertexTable vertexTable`). The specific details about types don't matter much here; we're going to focus mainly on the cost aspect of a particular implementation.

```
fun makeGraph (edges : 'a edge Seq.seq) =
  let
    val T = Table.collect (Seq.map (fn (u,v,w) => (u,(v,w))) edges)
  in
    Table.map Table.fromSeq T
  end
```

Q: What's the cost of `Table.collect` (assuming the comparison function has constant cost)?

A: On a sequence of length n, `Table.collect` does $O(n \log n)$ work and has $O(\log^2 n)$ span.

Q: What about the cost of `Table.map`? But this time, when we call `map f T`, the cost of running `f` v is $W(\texttt{f}\ \ v)$ Work and $S(\texttt{f}\ \ v)$ span.

A: The work is $O(\sum_{(k,v)\in T} W(\texttt{f}\ \ v))$ and the span is $O(\max_{(k,v)\in T} S(\texttt{f}\ \ v))$.

Which leads us to the question: *What's the work and span of `makeGraph`?*

Take #1: First, it's easy to see that for an input sequence of length m, `Table.collect` takes $O(m \log m)$ work and $O(\log^2 m)$ span.

Now let's examine what `Table.collect` returns. The variable `T` is a table where each key is a vertex and corresponding to that key is a (vertex,edgeLabel) sequence. Remember that an edge $u \to v$ with label ℓ is represented in the input sequence as (u, v, ℓ), so `Table.collect` simply groups all entries with the same u

together. This means that for each vertex u, `find T` u gives a (vertex, edgeLabel)-sequence for all edges that go out from u.

To simplify notation, define $n = |V|$.

What's the length of such a sequence? Clearly, it cannot be longer than m (since this is all the edges to begin with). How many keys can there be in T (i.e., what's the size of `domain T`)? At most n. Furthermore, `fromSeq S` has $O(|S| \log |S|)$ work and $O(\log^2 |S|)$ span.

So, we have that the cost of the `Table.map` step is $O(nm \log m)$. **Can we do better?**

Take #2: We have hopes to do better because we were really sloppy about estimating the length of the sequences on which we run `fromSeq`. Let's define the following quantity:

For each v, let F_v denote the sequence associated with the vertex v (i.e. $F_v =$ `find T v`).

Although we can't say much about the length of each F_v (that's why we were so pessimistic and said $|F_v| \leq m$), we do know something very concrete about their sum: since`Table.collect` simply partitions the entries into groups, we have

$$\sum_{v \in V} |F_v| = m.$$

Now running `Table.fromSeq` on F_v costs us $O(|F_v| \log |F_v|)$ work and $O(\log^2 |F_v|)$ span. Therefore, the total work for the map step is

$$\begin{aligned} W(\texttt{map fromSeq T}) &\leq c_1 \sum_{(k,v)\in T} W(\texttt{fromSeq } F_v) \\ &\leq c_1 c_2 \sum_{(k,v)\in T} |F_v| \log |F_v| \\ &\leq c_1 c_2 \sum_{(k,v)\in T} |F_v| \log m \\ &\leq c_1 c_2 \underbrace{\left(\sum_{(k,v)\in T} |F_v| \right)}_{=m} \log m, \end{aligned}$$

which is $O(m \log m)$ or $O(m \log n)$ because $m \leq n^2$. The span is $O(\log^2 n)$ and can be analyzed analogously.

3 Probability: Let's Flip Some Coins

We'll spend the rest of today's recitation on probability concepts, expanding what you saw in class. In particular, we'll revisit expectation and probabilistic recurrences.

3.1 Finding the Smallest k Values

Consider the following problem:

Input: S — a sequence of n real numbers (not necessarily sorted)

Output: a sequence of the smallest k numbers (could be in any order).

Requirement: $O(n)$ expected work and $O(\log^2 n)$ span.

Note that the linear-work requirement rules out the possibility of sorting the sequence. Here's where the power of randomization gives you a simple algorithm. For simplicity, we'll assume the elements are unique.

```
SmallestK (k, S) =
  1. If |S| <= k, return S.
  2. Pick p in S uniformly at random.
  3. Let L = { x in S: x <= p } and R = {x in S : x > p}
  4. If |L| >= k then return SmallestK(k, L)
     Else return append(L, SmallestK(k - |L|, R).
```

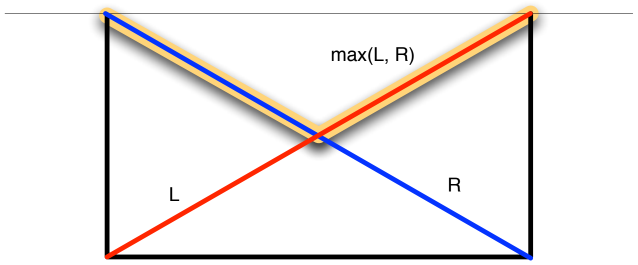
We'll try to analyze the work and span of this algorithm. First, note that the recursive call is on a sequence of length at most $\max\{|L|, |R|\}$. To simplify notation, let $X_n = \max\{|L|, |R|\}$. Since Step 3 is simply two `filter` calls, we have the following recurrences:

$$\begin{aligned} W(n) &= W(X_n) + O(n) \\ S(n) &= S(X_n) + O(\log n) \end{aligned}$$

Let's first look at the work recurrence. Specifically, we are interested in $\mathbf{E}[W(n)]$. First, let's try to get a sense of what happens in expectation.

Q: What is $\mathbf{E}[X_n]$?

A: First, let's take a look at a pictorial representation:



The probability that we land on a point on the curve is $1/n$, so

$$\mathbf{E}[X_n] = \sum_{i=1}^{n-1} \max\{i, n-i\} \cdot \tfrac{1}{n} \le \sum_{j=n/2}^{n-1} \frac{2}{n} \cdot j \le \frac{3n}{4}$$

(Recall that $\sum_{i=a}^{b} i = \frac{1}{2}(a+b)(b-a+1)$.)

Aside: This is a counterexample showing that $\mathbf{E}[\max\{X, Y\}] \neq \max\{\mathbf{E}[X], \mathbf{E}[Y]\}$.

This computation tells us that in expectation, X_n is a constant fraction smaller than n, so we should have a nice geometrically decreasing sum, which works out to $O(n)$. Let's make this idea concrete.

Suppose we want each recursive call to work with a constant fraction fewer elements than before, say at most $\frac{3}{4}n$.

Q: What's the probability $\mathbf{Pr}\left[X_n \le \frac{3}{4}n\right]$?

A: Since $|R| = n - |L|$, $X_n \le \frac{3}{4}n$ if and only if $n/4 < |L| \le 3n/4$. There are $3n/4 - n/4$ values of p that satisfy this condition. As we pick p uniformly at random, this probability is

$$\frac{3n/4 - n/4}{n} = \frac{n/2}{n} = \frac{1}{2}.$$

Notice that given an input sequence of size n, how the algorithm performs in the future is irrespective of what it did in the past. Its cost from that point on only depends on the random choice it makes after that. So, we'll let $\overline{W}(n) = \mathbf{E}\left[W(n)\right]$ denote the expected work performed on input of size n.

Now by the definition of expectation, we have

$$\begin{aligned}
\overline{W}(n) &\leq \sum_{i:X_n=i} \mathbf{Pr}\left[X_n = i\right] \cdot \overline{W}(i) + c \cdot n \\
&\leq \mathbf{Pr}\left[X_n \leq \tfrac{3n}{4}\right] \overline{W}(3n/4) + \mathbf{Pr}\left[X_n > \tfrac{3n}{4}\right] \overline{W}(n) + c \cdot n \\
&= \tfrac{1}{2}\overline{W}(3n/4) + \tfrac{1}{2}\overline{W}(n) + c \cdot n \\
&\implies \boxed{\overline{W}(n) \leq \overline{W}(3n/4) + 2c \cdot n.}
\end{aligned}$$

In this derivation, we made use of the fact that with probability $1/2$, the instance size shrinks to at most $3n/4$—and with probability $1/2$, the instance size is still larger than $3n/4$, so we pessimistically upper bound it with n. Note that the real size might be smaller, but we err on the safe side since we don't have a handle on that.

Now we know that the recurrence $\overline{W}(n) \leq \overline{W}(3n/4) + 2cn$ unfolds to at most $2cn(1 + \frac{3}{4} + (\frac{3}{4})^2 + \dots) \in O(n)$.

3.2 Span

Let's now turn to the span analysis. As before, we call each time we invoke the function `SmallestK` an *iteration*. To bound the number of iterations, we recall the following lemma from class:

Lemma 3.1 (Karp-Upfal-Widgerson)**.** *Let $T(n) = 1 + T(n - R_n)$ where for each $n \in \mathbb{Z}_+$, R_n is an integer-valued random variable satisfying $0 \leq R_n \leq n - 1$ and $T(1) = 0$. Let $\mathbf{E}\left[R_n\right] \geq \mu(n)$ for all $n \geq 1$, where μ is a positive non-decreasing function of n. Then,*

$$\mathbf{E}\left[T(n)\right] \leq \int_1^n \frac{1}{\mu(t)} dt.$$

We can massage our span recurrence to be in this form by making the following substitution: $n - R_n = X_n$, so $R_n = n - X_n$. By linearity of expectation, we have $\mathbf{E}\left[R_n\right] = n - \mathbf{E}\left[X_n\right] \geq n - 3n/4 = n/4$. Therefore, we could use $\mu(n) = n/4$. Applying the lemma, we have

$$\mathbf{E}\left[T(n)\right] \leq \int_{t=1}^n \frac{dt}{t/4} = 4 \ln n.$$

Since each iteration has $O(\log n)$ span, $S(n)$ solves to $O(\log^2 n)$ in expectation.