Recitation 4 — MapReduce and Graphs

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

September 21, 2011

Today's Agenda:
- Announcements
- An Inductive Proof from HW2
- Building Index
- Graph Representations
- Test Your Code!

1 Announcements

- Assignment 3 is due tomorrow at 11:59pm. Unusual late-day policy: you have until Saturday at 11:59pm to hand in, at the cost of 2 late days.
- Assignment 4 will go out on Friday and will be due next Thursday 29th. However, Assignment 5 won't be due until after Midterm 1 on Oct 6—but there will be test-prep questions that you should attempt before the test.
- Questions about homework, class, life, universe?

2 An Inductive Proof from HW2

Student A's Proof: I have a short proof that $W(n) = 4W(n/4) + \Theta(\sqrt{n})$ with $W(1) = 1$ solves to $O(\sqrt{n})$. My proof is much shorter than any of your proof in class and does *not* have to carry around a boatload of constants. Why didn't you teach us this proof?

It goes like this. I claim that $W(n) \in O(\sqrt{n})$, and my **IH** is simply $W(n) \in O(\sqrt{n})$. You know, the best case is trivial: clearly, $W(1) = 1 \in O(\sqrt{n})$. For the inductive step, I assume that $W(m) \in O(\sqrt{m})$ for all $m < n$. Therefore, I have

$$\begin{aligned} W(n) &= 4W(n/4) + c \cdot \sqrt{n} && \textit{(recurrence defn and } \Theta \textit{ defn)} \\ &\leq 4(k\sqrt{n}/4) + c \cdot \sqrt{n} && \textit{(big-O defn)} \\ &= k \cdot \sqrt{n} + c \cdot \sqrt{n} = (k+c)\sqrt{n}. \end{aligned}$$

Since k and c are both constants, this is $O(\sqrt{n})$, which completes my proof. □

Is this proof correct? If not, what is wrong with this proof?

The proof cannot be correct because the recursion tree has n leaves each costing a constant. The work clearly is at least $\Omega(n)$.

But where did this proof break down? When we claim that $W(n) \in O(\sqrt{n})$, the precise statement I claim is

There exists a constant k such that for any $n \geq 1$, $W(n) \leq k\sqrt{n}$.

That is to say, we commit to a constant value before we even know what the value of n is—and this constant has to make the statement hold for all $n \geq 1$. The proof above falls apart because it could not find a constant k to stick to: It incorrectly says that it was k for $n/4$ and it will be $k + c$ for n. This is not correct because we cannot change our mind once we commit to a particular constant value.

Q: What is the correct complexity?

As already stated, because the recursion tree has n leaves, the work must be at least $\Omega(n)$. Therefore, let's try proving $W(n) = O(n)$. We know by definition of Θ that there is a constant $c > 0$ such that $W(n) \leq 4W(n/4) + c \cdot \sqrt{n}$. We'll claim that

> There exist constant $k_1 > 0$ such that for all $n \geq 1$, $W(n) \leq k_1 n$.

For the inductive case, we'll assume that our claim is true for all $1 \leq m < n$: $W(m) \leq k_1 m$ (these are the same k_i's). Therefore, we have

$$\begin{aligned} W(n) &\leq 4W(n/4) + c\sqrt{n} && \textit{(by defn of } \Theta\textit{)} \\ &\leq 4(k_1 n/4) + c\sqrt{n} && \textit{(by IH since } n/4 < n\textit{)} \\ &= k_1 n + c\sqrt{n} \\ &\not\leq k_1 n \end{aligned}$$

since $c > 0$. The proof breaks down. We have not shown that $W(n) \leq k_1 n$. But does that mean $W(n)$ is not $O(n)$? No, it just suggests that we need to strengthen the inductive hypothesis so that we can get rid of the pesky additive term $c\sqrt{n}$.

Q: What does a correct proof look like?

Again, by definition of Θ there exists a constant $c > 0$ such that $W(n) \leq 4W(n/4) + c \cdot \sqrt{n}$. This time in our inductive hypothesis we will subtract a $k_2\sqrt{n}$ term that will allow us to fix the above proof.

> **Inductive Hypothesis**: There exist constants $k_1 > 0$ and $k_2 > 0$ such that for all $n \geq 1$, $W(n) \leq k_1 n - k_2\sqrt{n}$.

That is, we need to find constants k_1 and k_2 (that might depend on c but not n, because c is a constant but n is what we vary). Note that the constant c is predetermined, as it depends, for example, on the implementation. We *cannot* set it. But we can set k_1 and k_2 in terms of c in order to get the proof to go through.

For the inductive case, we assume that for all $1 \leq m < n$, $W(m) \leq k_1 m - k_2\sqrt{m}$ (these are the same k_i's). Therefore, we have

$$\begin{aligned} W(n) &\leq 4W(n/4) + c\sqrt{n} && \textit{(by defn of } \Theta\textit{)} \\ &\leq 4(k_1 n/4 - k_2\sqrt{n/4}) + c\sqrt{n} && \textit{(by IH since } n/4 < n\textit{)} \\ &= k_1 n - k_2\sqrt{n} + (c\sqrt{n} - k_2\sqrt{n}) \\ &\leq k_1 n - k_2\sqrt{n}, \end{aligned}$$

because we can find a k_2 such that $(c\sqrt{n} - k_2\sqrt{n}) \leq 0$, namely $k_2 = c$.

For the base case, we need to check that $W(1) = 1 \leq k_1(1) - k_2(1) = k_1 - c$.

That is, for $k_1 = c + 1 > 0$, $k_2 = c > 0$ we have shown the inductive hypothesis is true for all $n \geq 1$.

Notice that in the inductive case, the series of equations always either equals or is less than or equal to the previous equation. The last equation must be exactly the same equation as in the inductive hypothesis. Notice also that we cannot set the constant c, as it depends on the implementation. But we do set the constants k_1 and k_2 in terms of c.

3 Building Index Using MapReduce

MapReduce, as well as its open-source counterpart Hadoop, is a popular distributed computing framework, where the user supplies two functions a mapper and a reducer

$$\texttt{mapF} : \alpha \to (\textit{key} \times \beta)\,\textit{seq}$$
$$\texttt{reduceF} : \textit{key} \times (\beta\,\textit{seq}) \to \gamma.$$

Then, on input an α-sequence, the framework performs an equivalent of the following:

```
fun MR mapF reduceF (input :  'a seq) =
  let
    val pairs = flatten (map mapF input)
    val groups = collect keyCmp pairs
  in map reduceF groups
  end
```

We have seen a basic form of document indexing that supports logical queries involving words, *and* and *or*, etc. We have seen how to build such an index structure. Again, we'll look at the problem of building a document index, but we'll use MapReduce to solve it.

Let's recall the index building problem. Given a sequence of strings representing a collection of documents, the *index building* problem is to produce a table of (word, document-set) pairs. For example, the collection of documents ⟨ "210 is fun", "210 is brutal" ⟩ should have the following output:

$$\Big[(\text{“210”},\{0, 1\}), (\text{“is”},\{0,1\}), (\text{“fun”}, \{0\}), (\text{“brutal”},\{1\})\Big].$$

In class, we have seen an implementation that uses `tokens`, `map`, etc. In this recitation, we're interested in using MapReduce to solve this problem. Any thoughts?

Clearly, the input is a `string seq`, so then what should the mapper function and the reducer function look like? What are their types? What should they do?

We'll also have a new helper function, whose behavior should be intuitive.

```
val mapi:  (int * 'a -> 'b) -> 'a seq -> 'b seq
fun mapi f s =
let val f' = fn i => f(i, nth s i)
in tabulate f' (length s)
end


fun isSpace c = not (Char.isAlphaNum c)
fun docToWordIDPair (id, doc) = map (fn word => (word, id)) (tokens isSpace doc)

val mapper = docToWordIDPair
fun reducer docSeq = docSeq  (* identity reducer *)

fun makeIndex s = Table.fromSeq(MR mapper reducer (mapi (fn x=>x) s))
```

4 Graph Representations

Suppose you're given an undirected graph $G = (V, E)$, where V is a set of vertices (also known as nodes) and $E \subseteq \binom{V}{2}$ is a set of edges. Notice that in this definition, an edge $e = \{x, y\}$ is a set of size two representing the endpoints. Thus, the edge $\{x, y\}$ represents the same edge as $\{y, x\}$. Following this, we can think of representing a graph simply as an edge set. But the edge set representation doesn't provide an efficient means to access the neighbors of a vertex. That's why in class we looked at a different representation which we call an adjacency set. In the following, we'll look at a simple problem that will illustrate the differences in complexity between these two common representations.

Consider writing a simple routine to compute the degree of a vertex v. First, how would we do it in the edge set representation? Let the edge set E be represented as a set.

```
fun degree E v =
  let val nbrs = filter (fn (x,y) => v=x orelse v=y) E (** cost?  **)
  in size nbrs
  end
```

What's the cost of this function? Ans: linear work and $O(\log n)$ span.

Now suppose we're given a graph represented in the edge set form. Can we convert it into an adjacency set? What is the type of an adjacency set? Ans: `int set IntTable.table`.

```
fun makeAdjSet E =
  let
    val biDir = flatten (map (fn (x,y) => %[(x,y), (y,x)]) (toSeq E))
    val nbrsSeq = collect Int.compare biDir
  in Table.fromSeq (map (fn (u, nbrs) => (u, Set.fromSeq nbrs)) nbrsSeq)
  end
```

What's the cost of makeAdjSet? Ans: $O(n \log n)$ work and $O(\log^2 n)$ span.

Finally, in this adjacency set representation, it's super easy to compute the degree.

```
fun degree' G v =
  let val nbrs = find G v (** cost?  **)
  in size nbrs
  end
```

What's the cost? Ans: $O(\log n)$ cost (both work and span).

5 Test Your Code!

"Beware of bugs in the above code;
I have only proved it correct, not tried it."

— Donald Knuth

There is no substitute for testing your code even when you have carefully reasoned it to be correct. It is equally easy to write a buggy proof as it is to write buggy code. The bottom line is *test your code!* And if you can, use your proof to guide the testing and use the test to double check your proof.