Homework 7 — 15-210: Parallel and Sequential Data Structures and Algorithms (Fall’12)

tag — Version 1.0 + ε **_due:_** *Mon, Nov 5th @ 11:59pm*

Disclaimer:
We will not grade non-compiling code.

1 Introduction

In this homework, you will implement the parallel Minimum Spanning Tree algorithm known as Borůvka’s algorithm, write a grader for it, and solve some written problems on MST and probability.

1.1 Submission

Submit your solutions by placing your solution files in your handin directory, located at

```
/afs/andrew.cmu.edu/course/15/210/handin/<yourandrewid>/assn7/
```

Name the files exactly as specified below. You can run the check script located at

```
/afs/andrew.cmu.edu/course/15/210/bin/check/07/check.pl
```

to make sure that you’ve handed in appropriate files. Do not submit any other files.

Your written answers must be in a file called `hw07.pdf` and must be typeset. You do not have to use LaTeX, but if you do, we have provided the file `defs.tex` with some macros you may find helpful. This file should be included at the top of your `.tex` file with the command `\input{<path>/defs.tex}`.

For the programming part, the only files you’re handing in are

```
BoruvkaMST.sml
MSTTest.sml
```

These files must contain all the code that you want to have graded for this assignment and must compile cleanly. If you have a function that happens to be named the same as one of the required functions but does not have the required type, it will not be graded.

1.2 Style

As always, you will be expected to write readable and concise code. If in doubt, you should consult the style guide at `http://www.cs.cmu.edu/~15210/resources/style.pdf` or clarify with course staff. In particular:

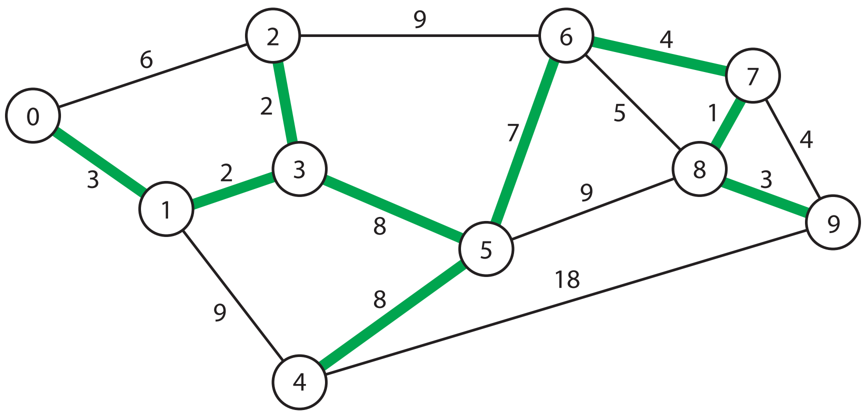
1. **Code is Art.** Code *can* and *should* be beautiful. Clean and concise code is self-documenting and demonstrates a clear understanding of its purpose.
2. **If the purpose or correctness of a piece of code is not obvious, document it.** Ideally, your comments should convince a reader that your code is correct.
3. **You will be required to write tests for any code that you write.** In general, you should be in the habit of thoroughly testing your code for correctness, even if we do not explicitly tell you to do so.
4. **Use the check script and verify that your submission compiles.** We are not responsible for fixing your code for errors. If your submission fails to compile, you will lose significant credit.

2 Borůvka's Algorithm

Recall that the minimum spanning tree (MST) of a connected undirected graph $G = (V, E)$ where each edge e has weight $w : E \to \mathbb{R}^+$ is the spanning tree T that minimizes

$$\sum_{e \in T} w(e)$$

For example, in the graph



the MST shown in green has weight 38, which is minimal.

You should be familiar with Kruskal's and Prim's algorithms for finding minimum spanning trees. However, both algorithms are sequential. For this problem, you will implement the parallel MST algorithm, otherwise known as Borůvka's algorithm.

2.1 Logistics

2.1.1 Representation

Vertices will be labeled from 0 to $|V| - 1$. The input graph is both simple (no self-loops, at most one undirected edge between any two vertices) and connected. We will represent both the input and output graphs as edge sequences, where an edge is defined as

```
type edge = vertex * vertex * weight
```

such that the triple `(u, v, w)` indicates a directed edge from `u` to `v` with edge weight `w`. For the input, since we will be dealing with undirected graphs, for every edge `(u, v, w)` there will be another edge `(v, u, w)` in the sequence. **The MST produced by your solution should only have edges in one direction.** You may find the following function useful:

```
Random210.flip : Random210.rand -> int -> int seq
```

where `Random210.flip r n` takes a `rand` r and produces a n-length sequence of random 0/1 numbers. You may assume that `Random210.flip r n` has $O(n)$ work and $O(\log n)$ span. Use `Rand.fromInt` and `Rand.next` to generate seed values for each round of your algorithm.

2.2 Specification

Task 2.1 (25%). Implement the function

```
MST : edge seq * int -> edge seq
```

where `MST (E, n)` computes the minimum spanning tree of the graph represented by the input edge sequence E using Borůvka's Algorithm in `BoruvkaMST.sml`. There will be n vertices in the graph, labeled from 0 to $n - 1$. For full credit, your solution must have *expected* $O(m \log n + n)$ work and *expected* $O(\log^k n)$ span for some k, where n is the number of vertices and m is the number of edges.

You may find the pseudocode given in lecture or the component labeling code covered in recitation useful for your implementation. Keep in mind that directly translating pseudocode to SML will not result in the cleanest, nor most efficient solution. For reference, our solution is under 50 lines long and only has one recursive helper. As a hint, recall that the `SEQUENCE` library function

```
inject : (int * 'a) seq -> 'a seq -> 'a seq
```

takes a sequence of index-value pairs to write into the input sequence. If there are duplicate indices in the sequence, the *last* one is written and the others are ignored. Consider presorting the input sequence of edges `E` from largest to smallest by weight. Then injecting any subsequence of `E` will always give priority to the minimum-weight edges.

2.3 Testing

You will now implement a grader for your `BoruvkaMST` implementation in `MSTTest.sml`. To do this, you must verify that your results satisfy the following conditions:

1. **Spanning**. A simple BFS would work well here.
2. **A Tree**. It suffices to check condition 1 and that you have $|V| - 1$ edges.
3. **Minimum**. Implement a simple MST algorithm and compare the resulting weights.

Task 2.2 (6%). Write the function

```
spanning : mstresult * int -> bool
```

in `MSTTest.sml` where `spanning (E, n)` evaluates to true if and only if the graph described by the sequence of edges in `E` is spanning on the vertex set $\{0, \ldots, n-1\}$. That is to say, any vertex in $\{0, \ldots, n-1\}$ is reachable from any other vertex. As hinted above, you should use BFS to verify this.

Task 2.3 (4%). Complete the implementation of

```
mstWeight : mstinput -> int
```

in `MSTTest.sml` where `mstWeight (E, n)` computes the MST weight of a graph with Kruskal's algorithm. Most of the code has already been written for you, as adapted from the `UFLabel` code in recitation 9. You should replace the `raise NotYetImplemented` parts.

Task 2.4 (5%). Complete the test structure `MSTTest` in `MSTTest.sml` by writing the function

```
grader : mstinput * mstresult -> bool
```

which verifies the above MST conditions using `spanning` and `mstWeight`, as well as the standard function `all : unit -> bool` to run a suite of tests (a combination of hardcoded edge cases and generated graphs) on the argument `MST` structure.

3 Written Problems

For the following problems, typeset your solutions in `hw07.pdf`. You may assume the correctness of any algorithms discussed in lecture.

Task 3.1 (4%). **Second-best is good enough for my MST.** Let $G = (V, E)$ be a simple, connected, undirected graph $G = (V, E)$ with $|E| \geq 2$ and *distinct* edge weights. We know for a fact that the smallest (i.e., least heavy) edge of G must be in the minimum spanning tree (MST) of G. Prove that the 2^{nd} smallest edge of G must also be in the minimum spanning tree of G.

$\boxed{\textbf{Solution 3.1}}$ By Kruskal's algorithm, we consider edges in sorted order, adding an edge $\{u, v\}$ if u and v are in different components. Suppose for the sake of contradiction that the second edge is not in the MST. Then the two vertices are already in the same component. But that means that the second edge connects the same two vertices as the first edge, contradicting the simplicity of our graph.

Task 3.2 (8%). **I Prefer Chalk.** There is a very unusual street in your neighborhood. This street forms a perfect circle, and there are $n \geq 3$ houses on this street. As the unusual mayor of this unusual neighborhood, you decide to hold an unusual lottery. Each house is assigned a random number $r \in_R [0, 1]$ (drawn uniformly at random). Any house that receives a larger number than **both** of its two neighbors wins a prize package consisting of a whiteboard marker and two pieces of chalk in celebration of education. What is the expected number of prize packages given? Justify your answer.

$\boxed{\textbf{Solution 3.2}}$

The probability that a given house wins a prize is:

$$\textbf{Pr}\left[\text{house wins a prize}\right] = \textbf{Pr}\left[\text{house has largest number}\right] = \frac{1}{3}$$

Let random variable X be the number of prize packages given out to the households. To find the expected number of prizes given $\textbf{E}[X]$, define the random indicator variable X_i be 1 if house i wins a prize, and 0 if it does not. By Linearity of Expectation,

$$\textbf{E}[X] = \sum_{i=1}^{n} \textbf{E}[X_i] = \sum_{i=1}^{n} \textbf{Pr}\left[\text{house } i \text{ wins a prize}\right] = \sum_{i=1}^{n} \frac{1}{3} = \frac{n}{3}$$

Task 3.3 (8%). **It's *Probably* Linear.** Let f be a non-decreasing function satisfying

$$f(n) \leq f\left(X_n\right) + \Theta(n), \qquad \text{where } f(1) = 1.$$

Prove that if for all $n > 1$, $\textbf{E}\left[X_n\right] = n/3$, then $\overline{f}(n) = \textbf{E}\left[f(n)\right] \in O(n)$.
Hint: what is $\textbf{Pr}[X_n \geq 2n/3]$? Use Markov's Inequality, covered in recitation 8.

$\boxed{\textbf{Solution 3.3}}$

In recitation, we proved a super useful identity known as *Markov's inequality*. In the setting of the current problem, this implies that $\mathbf{Pr}\left[X_n \geq 2n/3\right] \leq \frac{\mathbf{E}\left[X_n\right]}{2n/3} = \frac{1}{2}$. By the definition of $\Theta(\cdot)$, our recurrence satisfies

$$f(n) \leq f(X_n) + c \cdot n$$

for some constant $c > 0$.

Therefore, we establish

$$\begin{aligned}
\mathbf{E}\left[f(n)\right] = \overline{f}(n) &\leq \overline{f}(X_n) + c \cdot n \\
&\leq \overline{f}(2n/3)\,\mathbf{Pr}\left[X_n < 2n/3\right] + \overline{f}(n)\,\mathbf{Pr}\left[X_n \geq 2n/3\right] + c \cdot n \\
&= p \cdot \overline{f}(2n/3) + (1-p) \cdot \overline{f}(n) + c \cdot n
\end{aligned}$$

where $p = \mathbf{Pr}\left[X_n < 2n/3\right]$. So then, collecting similiar terms, we get

$$(1 - (1-p)) \cdot \overline{f}(n) \leq c \cdot n + p \cdot \overline{f}(2n/3).$$

Dividing through by p gives

$$\overline{f}(n) \leq \frac{c}{p} \cdot n + \overline{f}(2n/3) \implies \overline{f}(n) \leq 2c \cdot n + \overline{f}(2n/3) \text{ since } p > 1/2.$$

This recurrence is root-dominated and solves to $\overline{f}(n) \in O(n)$.