`tag` — Version 1.0 ***due:*** *tue, apr 3 @ 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 and solve two written problems about minimum spanning trees and maximal matching.

1.1 Submission

This assignment is distributed in a number of files in our `git` repository. Instructions on how to access that repository can be found at `http://www.cs.cmu.edu/~15210/resources/git.pdf`. This is how assignments will be distributed in this course.

This assignment requires that you submit both code and written answers.

Submit your solutions by placing you 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.

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

`BoruvkaMST.sml` and `TestMST.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 Naming Modules

The questions that follow ask you to organized your solutions in a number of modules. Your modules must be named exactly as stated in the handout. Correct code inside an incorrectly named structure, or a structure that does not ascribe to the specified signatures, *will not receive any credit*.

You may not modify any of the signatures or other library code that we give you. We will test your code against the signatures and libraries that we hand out, so if you modify the signatures your code will not compile and you will not receive credit.

1.3 The SML/NJ Build System

This assignment includes a substantial amount of library code spread over several files. The compilation of this is orchestrated by CM through the file `sources.cm`. Instructions on how to use CM can be found at on the website at `http://www.cs.cmu.edu/~15210/resources/cm.pdf`.

1.4 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:

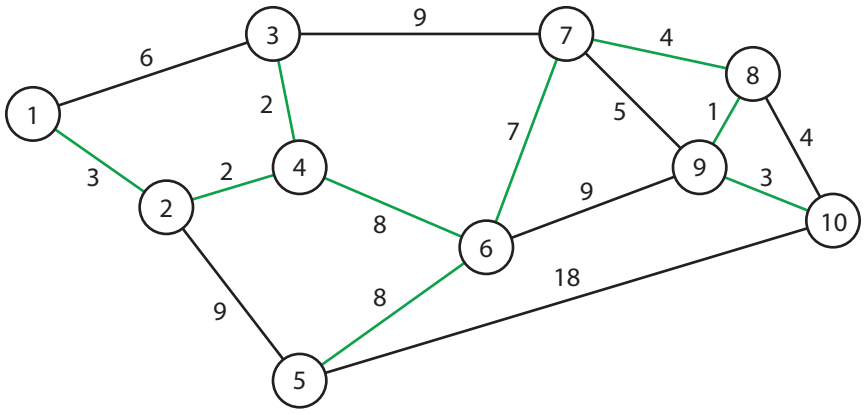
1. **If the purpose or correctness of a piece of code is not obvious, document it.** Ideally, your comments would convince a reader that your code is correct.
2. **You will be required to write tests for any code that you write.** In general, you should be in the habit of thoroughly testing your code for correctness, even if we do not explicitly tell you to do so. We have provided a file, `TestSML.sml` which you should put all your test code in.

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

For efficiency, you should use single-threaded or normal array sequences in your implementation for fast lookups and updates. Vertices will be labeled from 0 to $|V| - 1$. We will represent both the input and output graphs as edge sequences, where an edge is defined

```
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. For simplicity, the MST produced by your solution doesn't need to have edges in both directions (you need at least one direction of each edge in there).

Note that the `edge` type is for input/output specifications; your internal representation may differ.

2.1.2 `Random210`

In `lib/Random210.sml`, we have provided you with the randomization structure conveniently named `Random210`. In particular, you may find the following function useful:

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

where `flip r n` takes a seed value r and produces a n-length sequence of random 0/1 numbers. 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 $O(m \log n + n)$ work and $O(\log^k n)$ 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. 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

Task 2.2 (5%). Complete the structure `TestMST` in `TestSML.sml` that thoroughly tests your solution for correctness.

3 Written Problems

Task 3.1 (5%). **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.

Task 3.2 (15%). **Maximal Matching Max.** TA Max (not his real name, due to privacy concerns) is in charge of pairing students to work on a final programming project in SML. However, not all students are compatible with each other. Max has drawn a graph $G = (V, E)$ where V is the set of students and E is the set of edges such that $(s_1, s_2) \in E$ if the students s_1 and s_2 are willing to partner up.

Since he's running out of time, Max has come to you for help. He only cares to pair students until the rest are incompatible with each other (a maximal matching). The remaining students will be assigned 251 problems to do by themselves for being so inconvenient.

Design an algorithm that finds a maximal matching of the students in $O(|E| \log |V|)$ work and $O(\log^2 |V|)$ span. You must write your solution in SML or pseudocode, and prove that it meets the given cost bounds.

(*Hint:* A first attempt at a solution might be to run MIS (Maximal Independent Set) on the set of possible pairings, but constructing the graph itself can already take more than $\Omega(|E| \log |V|)$ work[1]).

[1] So don't construct it.