Recitation 11

Graph Contraction and MSTs

11.1 Announcements

- *SegmentLab* has been released, and is due **Monday afternoon**. It’s worth 135 points.
- *Midterm 2* is on **Friday, November 11**.
11.2 Contraction

In the textbook, we presented an algorithm for counting the number of connected components in a graph:

**Algorithm 11.1. (Algorithm 17.22 in the textbook.)**

```plaintext
1 countComponents (V,E) =
2   if |E| = 0 then |V| else
3     let (V',P) = starPartition (V,E)
4       E' = { (P[u],P[v]) : (u,v) ∈ E \ P[u] ≠ P[v] }
5     in
6       countComponents (V',E')
7   end
```

with `starPartition` implemented as follows:

**Algorithm 11.2. (Algorithm 17.15 in the textbook.)**

```plaintext
1 starPartition (V,E) =
2   let
3     TH = { (u,v) ∈ E \ ¬heads(u)\∧heads(v) }
4     P = \bigcup_{(u,v) ∈ TH} \{ u ↦ v \}
5     V' = V \ domain(P)
6     P' = \{ u ↦ u : u ∈ V' \}
7     in
8     (V',P' ∪ P)
9   end
```

Now, suppose we implemented star partitioning for enumerated graphs as follows:

```plaintext
val enumStarPartition : (int * int) Seq.t * int → int Seq.t
```

Specifically, given a graph represented as a sequence of edges `E` where every vertex is labeled `0 ≤ v < n`, `enumStarPartition (E,n)` returns a mapping `P` where `P[v]` is the super-vertex containing `v`. (If `v` was a star center or was unable to contract, then `P[v] = v`.)

**Task 11.3. Implement a function `enumCountComponents` which counts the number of components of an enumerated graph. It should take in a graph represented as `(E, n)` and use `enumStarPartition` internally.**
A direct but incorrect translation of the original code might look like this:

```ml
fun incorrectCountComponents (E, n) = 
  if |E| = 0 then n else 
  let 
    val P = enumStarPartition (E, n) 
    val E' = \((\langle P[u], P[v] \rangle : (u, v) \in E \mid P[u] \neq P[v] \rangle \) 
  in 
  incorrectCountComponents (E', n) 
end
```

The problem with this code is that it doesn’t actually count the number of connected components, despite performing the contraction correctly. This is because we never modify the value `n`.

A first step in fixing the issue is to add a line after line 5 which counts the number of distinct vertices in `E'`. Specifically, we use `P` to identify which vertices no longer exist, filter them out, then simply take the length of the resulting sequence:

```ml
val n' = |\langle v : 0 \leq v < n' \mid P[v] = v \rangle |
```

We could then pass `n'` in to the recursive call rather than `n`. However, we now notice an even bigger problem: not all vertices in `E'` are labeled `0 \leq v < n'`.

What we really need to do is construct a new labeling within the range `\[0, n'\)`. We can do so by marking each each contracted vertex with a 0 and each remaining vertex with a 1 and running a +-scan. This determines a sequence `P'` which maps each remaining vertex to a unique label in the range `\[0, n'\)`. This step also conveniently calculates `n'`. At the end of the round, when we promote edges by relabeling their endpoints, we have to further relabel them according to `P'`. The code is as follows.

```
Algorithm 11.4. Counting connected components in an enumerated graph.

fun enumCountComponents (E, n) = 
  if |E| = 0 then n else 
  let 
    val P = enumStarPartition (E, n) 
    fun isAlive v = if P[v] = v then 1 else 0 
    val (P', n') = Seq.scan + 0 \langle isAlive(v) : 0 \leq v < n \rangle 
    val E' = \((\langle P'[u], P'[v] \rangle : (u, v) \in E \mid P[u] \neq P[v] \rangle \) 
  in 
  enumCountComponents (E', n') 
end
```

Built: November 7, 2016
11.2.1 Cost Bounds

**Task 11.5.** Recall that a forest is a collection of trees. What are the work and span of `enumCountComponents` when applied to a forest? Assume that `enumStarPartition(E,n)` requires $O(n + |E|)$ work and $O(\log n)$ span.

Line 6 of `enumCountComponents` clearly requires $O(n)$ work and $O(\log n)$ span. Line 7 is just a map followed by a filter, and therefore requires $O(m)$ work and $O(\log n)$ span. But how do $n$ and $m$ change, round-to-round?

Regarding $n$, we recall that star-partitioning removes at least $n/4$ vertices in expectation, and therefore we expect the number of vertices to decrease geometrically.

For general graphs, we can’t say that $m$ decreases geometrically. However, a tree has $n - 1$ edges, and therefore $m$ is initially upper bounded by $n - 1$. Furthermore, on each round, exactly one edge is deleted for every vertex which is deleted. Therefore, for forests and trees, $m$ decreases geometrically during contraction. Therefore the total work and span of this algorithm for an input forest of $n$ vertices are $O(n)$ and $O(\log^2 n)$, respectively.
11.3 Borůvka’s Algorithm

The textbook describes two versions of Borůvka’s algorithm: one which performs tree contraction at each round, and another which performs a single round of star contraction at each round. We will be using the latter, since it has better overall span ($O(\log^2 n)$ rather than $O(\log^3 n)$).

**Task 11.6.** Run Borůvka’s algorithm on the following graph. Draw the graph at each round, and identify which edges are MST edges. Use the coin flips specified.

![Graph Diagram]

<table>
<thead>
<tr>
<th>Round</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>H</td>
<td>T</td>
<td>H</td>
<td>T</td>
<td>T</td>
<td>H</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>H</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>H</td>
<td>T</td>
<td>T</td>
<td>H</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>H</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Round 0:

Round 1:

Round 2:
Exercise 11.7. In graph theory, an independent set is a set of vertices for which no two vertices are neighbors of one another. The maximal independent set (MIS) problem is defined as follows:

For a graph \((V, E)\), find an independent set \(I \subseteq V\) such that for all \(v \in (V \setminus I)\), \(I \cup \{v\}\) is not an independent set.\(^a\)

Design an efficient parallel algorithm based on graph contraction which solves the MIS problem.

\(^a\)The condition that we cannot extend such an independent set \(I\) with another vertex is what makes it “maximal.” There is a closely related problem called maximum independent set where you find the largest possible \(I\). However, this problem turns out to be NP-hard!