Recitation 13

Minimum Spanning Trees

13.1 Announcements

- *SegmentLab* has been released, and is due **Friday, November 17**. It’s worth 135 points.

13.2 Prim's Algorithm

Minimum spanning trees are useful for a variety of applications in computer science, such as resource allocation, clustering, and image processing. They exhibit certain "greedy" properties that allow fast computation.

In particular, the *light-edge property*, or *cut property*, states that when a graph $G = (V, E)$ has its vertices cut into two partititions $(U, V \setminus U)$, then the edge with minimum weight that crosses from U into $V \setminus U$ is in the MST of G.

Prim's algorithm allows us to exploit this property to greedily insert edges into the partial MST until the full tree is built. The algorithm performs a priority-first search in a fashion similar to Dijkstra's algorithm. A partial implementation for connected, weighted, undirected graphs is given below.

Algorithm 13.1. *Prim's Algorithm (Partial)*

```
fun Prim G =
  let
    fun prim (X,T,Q) =
      case PQ.deleteMin Q of
        (NONE, _) ⇒ ______________
      | (SOME (d,(u,v)),Q') ⇒
          if $v \in X$ then ______________ else
          let
            val X' = ______________
            val T' = case u of
              NONE ⇒ ______________
            | SOME u' ⇒ ______________
            fun relax (Q,(v',w)) = ______________
            val Q'' = iterate relax Q' ($N_G^+(v)$)
          in
            prim (X',T',Q'')
          end
  in
    prim ({},[],PQ.singleton (0,(NONE,0)))
  end
```

Task 13.2. *Complete the implementation above by filling in the blanks. The similarity of Prim's and Dijkstra's algorithms should yield a $O(m \log n)$ work and span bound as for Dijkstra's algorithm.*

Algorithm 13.3. *Prim's Algorithm (Complete)*

```
fun Prim G =
  let
    fun prim (X,T,Q) =
      case PQ.deleteMin Q of
        (NONE, _) => T
      | (SOME (d,(u,v)),Q') =>
          if $v \in X$ then prim (X,T,Q') else
          let
            val X' = $X \cup \{v\}$
            val T' = case u of
              NONE => T
            | SOME u' => (u',v) :: T'
            fun relax (Q,(v',w)) =
              PQ.insert (Q,(w,(SOME v,v')))
            val Q'' = iterate relax Q' ($N_G^+(v)$)
          in
            prim (X',T',Q'')
          end
  in
    prim ({},[],PQ.singleton (0,(NONE,0)))
  end
```

Task 13.4. *In Dijkstra's algorithm, it was possible to terminate the algorithm early when we first reached some vertex t to obtain the shortest path from s to t. Can we make a similar optimization in Prim's algorithm? If so, what is it?*

Yes. Since the MST has at most $n - 1$ edges, we may stop when $|T| = n - 1$.

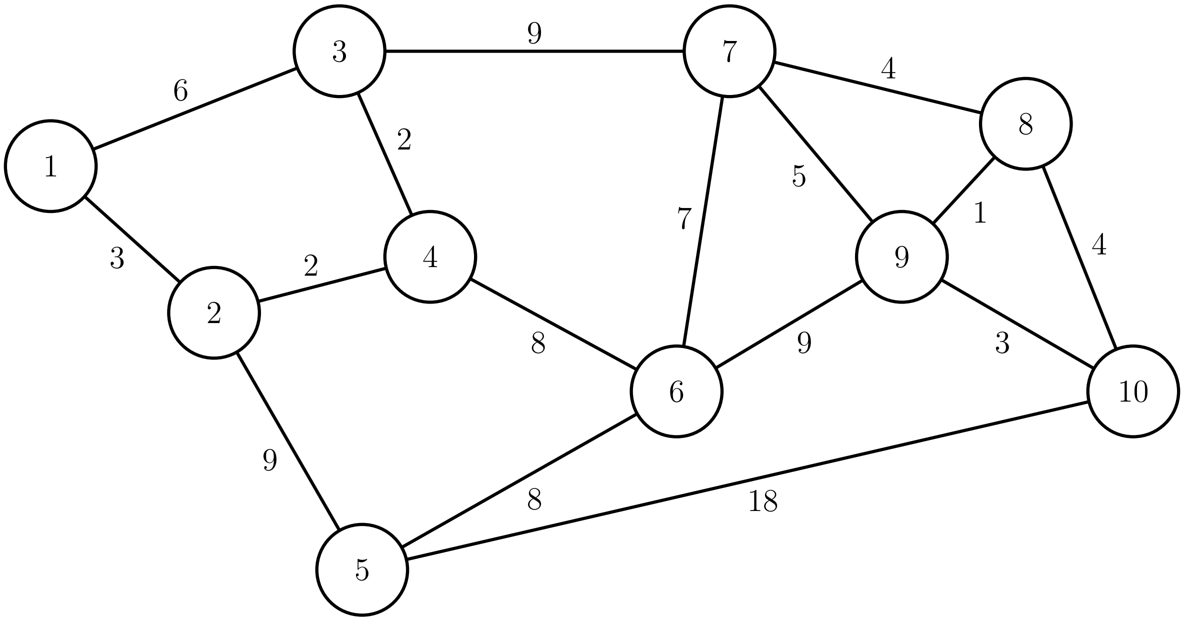
Remark 13.5. *Just as we generalized DFS to solve problems ranging from connectivity to bridge edges, we see it is possible to generalize Dijkstra's algorithm to obtain BFS, A* search, or Prim's algorithm. How versatile!*

Since the algorithm is so similar to Dijkstra's algorithm, we would expect the same work and span bounds, or possibly faster using a more advanced heap structure. However, Prim's algorithm has no parallelism, while Borůvka's algorithm does.

13.3 Borůvka's Algorithm

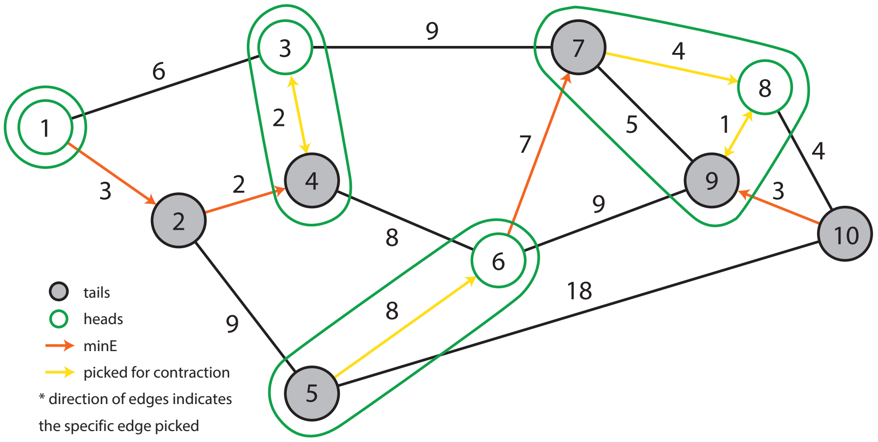
The textbook describes two versions of Borůvka's algorithm: one which performs tree contraction at each round, and another which performs a single round of star contraction at each round. We will be using the latter, since it has better overall span, $O(\log^2 n)$ rather than $O(\log^3 n)$.

Task 13.6. *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.*

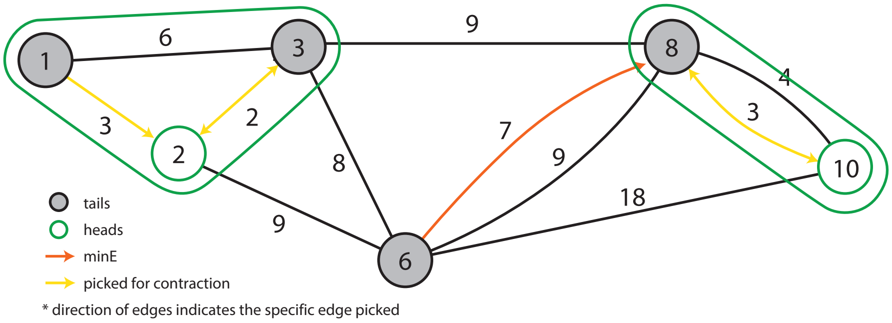


	Vertices									
Round	1	2	3	4	5	6	7	8	9	10
0	H	T	H	T	T	H	T	H	T	T
1	T	H	T			T		T		H
2		T				H				T

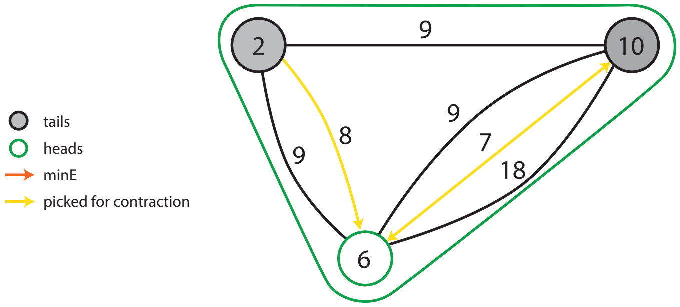
Round 0:



Round 1:



Round 2:



13.4 Additional Exercises

Exercise 13.7. *The vertex-joiners selected in any round of Borůvka's algorithm form a forest when no two edge weights are equal. Prove this fact.*
Hint*: a forest, by definition, has no cycles.*

Exercise 13.8. *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!