Lecture Notes CS:5360 Randomized Algorithms
Lectures 10, 11, and 12: Sep 20, Sep 25, and Sep 27, 2018
Scribe: Hankyu Jang

1 Minimum Spanning Tree (MST)

MST is another example of sampling.

- Input: An edge-weighted, connected graph $G = (V, E)$. Let $w(e)$ denote weight of edge e where $w(e) > 0$ for all e.
- Output: A spanning tree of G of minimum weight.

Example Some of the well known "greedy" algorithms are (1) Kruskal's algorithm, (2) Prom's algorithm, and (3) Boruvka's algorithm. Take a look at Figure 1 to understand how Kruskal's algorithm work.

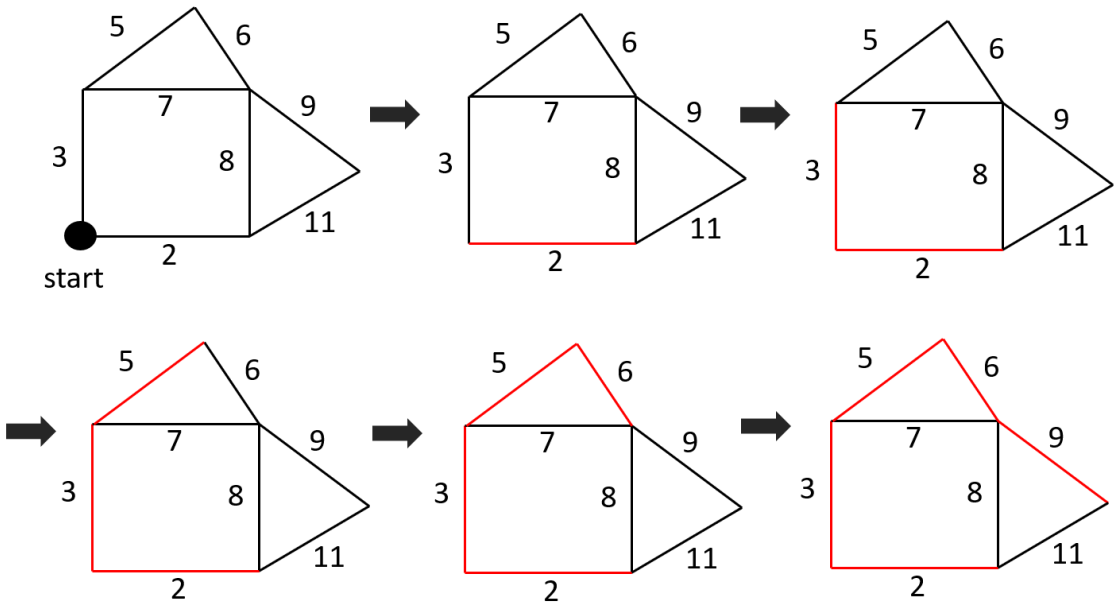

Figure 1: For each step of Kruskal's algorithm, it adds an edge to the tree that has the minimum weight and does not create a cycle. As depicted in the picture, the algorithm starts at the start node and then adds an edge respectively.

Let m be the number of edges and n be the number of vertices. Then, using simple date structures, these algorithms can be implemented in $O((m+n)\log n)$ time. (note that the state of the art, $\log n$ can be replaced with $\beta(m, n)$ which is an extremely slowly growing function.)

Question Does MST have an $O(m+n)$ time algorithm? In other words, does it have a linear time algorithm?

The answer to above question is that no one knows if there's a deterministic linear time algorithm. The problem is open if we focus on deterministic algorithm. There was a breakthrough from the early 1990's where Karger, Klein, and Tarjan showed that there is a Las Vegas algorithm for MST running in the expectation of $O(m+n)$ time. The key idea of this algorithm comes from sampling.

Before we move on, let's review Kruskal's algorithm.

Algorithm 1: KRUSKAL'S ALGORITHM

```
1 Sort the edges in increasing order of weight;
2 T ← (V, ∅) (containing all the vertices but no edges);
3 for each edge e = {u, v} considered in order do
4 |   if u and v are in distinct connected components of T then
5 |   |   add e to T;
6 |   end
7 end
```

Let's look at the typical step T in the Kruskal's algorithm in Figure 2.

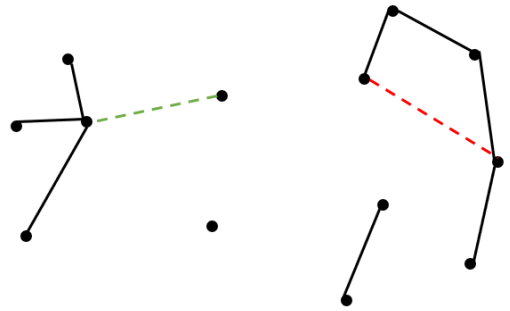

Figure 2: The black lines are the edges in T. If the next edge is the green dashed line, then Kruskal's algorithm will add the edge to T. If the next edge is the red dashed line, then Kruskal's algorithm will not add the edge to T.

Notation Let $F = (V, E')$, $E' \subseteq E$ be a spanning forest of G. For any pair of vertices $u, v \in V$, if u and v are in the same connected component of F, then $w_F(u,v)$ is the heaviest weight of an edge of the path between u and v. You can view this in Figure 3.

Definition An edge $\{u, v \in E\}$ is F-heavy if $w(u,v) > w_V(u,v)$. Otherwise, if $w(u,v) \leq w_V(u,v)$ then $\{u,v\}$ is F-light. Figure 4 depicts the different cases.

Note that every edge in F is F-light and other edges in E are partitioned into F-light or F-heavy edges.

Observation Let $F = (V, E')$ be an arbitrary spanning forest of G. Then if $\{u,v\} \in E$ is F-heavy, it is not in the MST.

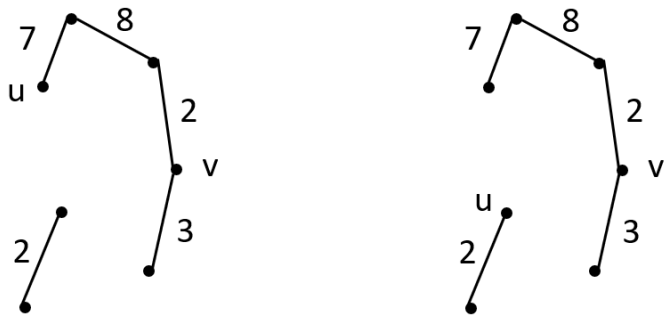

Figure 3: In the graph on the left, u and v are in the same connected component. Hence, $w_F(u, v) = 8$ is the heaviest weight of an edge of the path between u and v. In the graph on the right, u and v are in the different components. Therefore, $w_F(u, v) = \infty$.

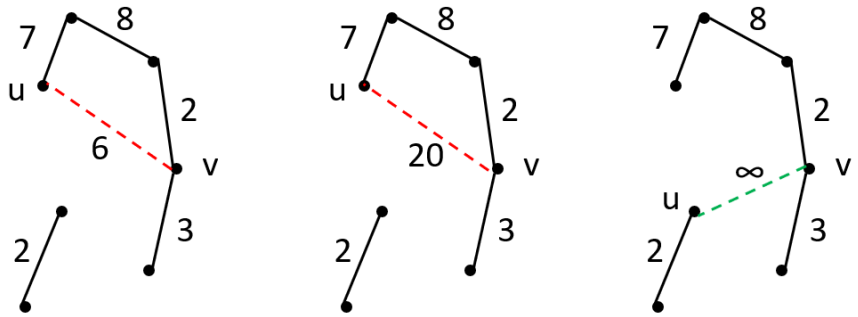

Figure 4: In the graph on the left, the edge connecting u and v is F-light because $w(u, v) = 6 < w_F(u, v) = 8$. In the graph in the middle, the edge connecting u and v is F-heavy because $w(u, v) = 20 > w_F(u, v) = 8$. In the graph on the right, the edge connecting u and v is also F-light because no matter weight of the edge, $w(u, v) < w_F(u, v) = \infty$.

Question How can we use randomization to find F quickly so that we can focus on computing MST on F-light edges?

1.1 KKT Sampling Theorem

For $0 \leq p \leq 1$, let $G(p)$ be the spanning subgraph of G in which each edge of G is placed by picking it independently with probability p. Size of $G(p)$ depends on p. Let F be a minimum weight spanning forest of $G(p)$. Then the expected number of F-light edges is at most $\frac{n}{p}$. KKT Sampling Theorem suggests the following MST algorithm:

1. Pick a value of p and compute $G(p)$
2. Find a Minimum Spanning Forest F of $G(p)$
3. Find the set L of F-light edges
4. Compute and return MST of (V, L)

There are known deterministic linear time algorithms for step (3). Steps (2) and (4) are MST computation step and step (3) is MST verification step. The higher you choose p, the more computation is needed in step (2); but $\frac{n}{p}$ is smaller so less the computation is needed in step (4). We need a balance!

Example To balance work in steps (2) and (4), set $mp = \frac{n}{p}$ where n is number of edges in step (4). This lead to choosing $p = \sqrt{\frac{n}{m}}$. Suppose $m = \Theta(n^2)$ (very dense graph). Then,

$$\begin{aligned} mp &= n^2\sqrt{\frac{n}{n^2}} \\ &= n^2\frac{1}{\sqrt{n}} \\ &= n^{1.5} \end{aligned}$$

1.1.1 Proof of KKT sampling theorem

Consider edges of G e_1, e_2, $\cdots$, e_m in increasing order of weight. In step i, we process e_i. e_1, e_2, e_3, $\cdots$, e_{i-1} are already processed.

- Edges get included or excluded from $G(p)$
- Some edges are added to F
- Edges are classified as F-light or F-heavy

Algorithm 2: KKT SAMPLING AT e_i (step i)

```
if coin toss = H then
    add e_i to G(p);
    if e_i connects to different components of F then
        add e_i to F;
        classify e_i as F-light;
    end
    classify e_i as F-heavy;
    if e_i connects two different components in F then
        classify e_i as F-light;
    end
    classify e_i as F-heavy;
end
```

The table below illustrates Algorithm 2. At the end of Phase 0, e_i is added to F since it's the first head. Likewise, at the end of Phase 1 e_j is added to F. One thing to notice that in Phase 2, coin toss is H for an edge e_{j+1} but this edge was not added to F because the edge is F-heavy which would create a cycle. Only edges that are F-light with respect to the F in the previous phase can be added to F when the coin toss is Head. In general, at a typical phase, the first appearing F-light edge can be added to F only if the coin toss is Head.

KKT Sampling			
-	Phase 0	Phase 1	Phase 2
Coin Toss	T T $\cdots$ T H	T T $\cdots$ T H	H T ... T H
Edge selected	$e_1e_2\cdots e_{i-1}e_i$	$e_{i+1}e_{i+2}\cdots e_{j-1}e_j$	$e_{j+1}e_{j+2}\cdots e_{l-1}e_l$
Edges in F	0	1	2

1.2 KKT Sampling Lemma

Let G be an edge-weighted graph. For $0 \leq p \leq 1$, let $G(p)$ be the spanning sub-graph obtained by sampling each edge in G independently with probability p. Let F be a minimum weight spanning forest of $G(p)$. Then the expected number of F-light edges of G is $\leq \frac{n}{p}$

Proof We consider the m edges $e_1, e_2, \cdots, e_m$ of G in order of increasing weight. We process edges in this order and in step i, we decide if

1. $e_i \in G(p)$ by tossing a biased coin
2. $e_i \in F$
3. e_i is F-light F-heavy

We partition the processing of $e_1, e_2, \cdots, e_m$ into phases. Let Phase k = subsequence of edges processed while waiting to add the $(k+1)^{th}$ edge to F. Let's take a look at Phase k where the sequence of edges is following: $e_p, e_{p+1}, \cdots, e_q$. At the end of Phase k-1, F has k edges. Throughout Phase k, F stays same which means at the start of Phase k, edges in Phase k are fixed as F-light or F-heavy. So, it's not dynamic within this phase.

Example e_p is F-heavy, coin toss is H. What happens? In this case, do not add the edge to F, but add it o $G(p)$.

F-heavy edges in Phase k are not added to F independent of coin tosses. So we are essentially just processing F-light edges until we get the first F-light edge for which we have heads as the coin toss outcome.

Let X_k = number of F-light edges in Phase k. What is the distribution of X_k? $X_k \sim Geom(p)$. We know that $E[X_k] = \frac{1}{p}$

Let X = number of F-light edges. Note that if an edge is F-light when it is processed, it remains F-light forever. Therefore, $X = \sum_{k \geq 0} X_k$

Suppose we add s edges to F for some $0 \leq s \leq n-1$. The random variables X_0, X_1, $\cdots$, X_0 follow $Geom(p)$. Hence, $E[X] = \frac{s-1}{p} \leq \frac{n-1}{p}$.

What about X_S? In other words, what about the remaining parts?

To deal with this issue, imagine that we add infinitely many dummy F-light edges to e_1, e_2, $\cdots$, e_m. Let the dummy edges be e_{m+1}, e_{m+2}, e_{m+3}, $\cdots$, and let's say these are all F-light edges. Also, we toss coins until we have tossed n Heads for F-light edges.

Let Y = number of F-light edges we process in order to get n Heads. Then, $E[X_0 + X_1 + \cdots + X_{s-1} + X_s] \leq E[Y]$. Note that until Phase s-1, there are up to $n-1$ heads corresponding to F-light edges. In Phase s (X_s) we have bunch of heads corresponding to F-light edges.

Definition A random variable Y has negative binomial distribution with parameters n (number of successes we want) and p (probability of head) if it equals the number of coin tosses of a biased coin needed to get n Heads.

So far, $y = n, n+1, \cdots$

$P(Y = y) = \binom{y-1}{n-1} p^n (1-p)^{y-n}$

Facts $E[Y] = \frac{n}{p}, Var[Y] = \frac{n(1-p)}{p^2}$

Since $Y \sim NegBin(n,p), E[Y] = \frac{n}{p}$, expected number of F-light edges $\leq \frac{n}{p}$

We will now use the KKT Sampling Lemma to get a randomized Las Vegas $O(m+n)$ MST algorithm.

1.3 MST Verification

- Input: An edge weighted graph G and a spanning forest F
- Output: Edges in G that are F-heavy

If F is MST, output is all edges beside the edges in F. However, if F is a spanning forest, output is all edges beside the edges in F, which is depicted in Figure 5.

There are several deterministic linear time algorithms for this problem. The simplest one is due to Valerie King in the early 1990s. Another ingredient is Boruvka's MST algorithm.

Algorithm 3: BORUVKA'S MST ALGORITHM

```
1 while there is more than one connected component do
2 |   For each vector v, pick a minimum weight edge e_v incident on v;
3 |   Add all e_v's to solution;
4 |   Contract connected components to supervertices;
5 end
```

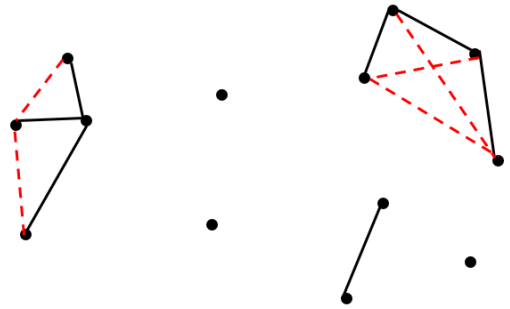

Figure 5: The lines are the edges in a spanning forest F. If this is the case, red dashed lines are the output of MST verification.

Figure 6 shows an applying Boruvka's MST algorithm on a graph.

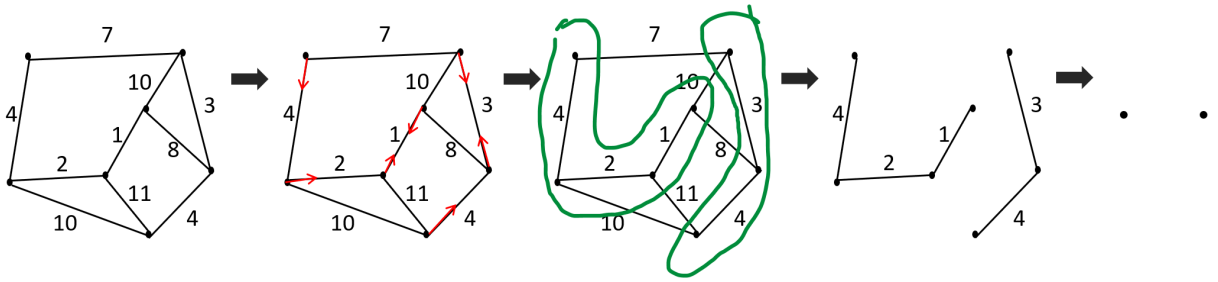

Figure 6: Red arrows on the second graph are e_vs picked per node. These red arrows are all added to solution. Then, the connected components are contracted to super-vertices.

There are two facts about Boruvka's algorithm. (1) If we start a Boruvka iteration with n vertices, we have $\leq \frac{n}{2}$ vertices after the iteration, and (2) Each Boruvka iteration can be implemented in $O(m+n)$ time.

1.4 KKT MST algorithm

So far, we talked about three ingredients which are (1) KKT Sampling lemma, (2) Deterministic linear-time algorithm for MST verification and (3) Boruvka's algorithm. With these ingredients in place, we can now describe the KKT MST algorithm.

Input An edge-weighted graph $G(V, E)$

Algorithm 4: KKT MST ALGORITHM

1 We apply 3 iterations of Boruvka's algorithm. Let G_1 be the resulting graph. Let C be edges selected to be in MST by these iterations of Boruvka's algorithm;
2 If G_1 contains a single vertex then return C and exit;
3 Set $p = \frac{1}{2}$ and compute $G_2 = G_1(p)$;
4 By recursively calling the KKT MST algorithm, compute a minimum spanning forest F_2 of G_2;
5 Call MST verification algorithm to compute F_2-light edges of G_1. (MST verification algorithm will give the heavy edges. Use the complement of this);
6 Recursively call the KKT MST algorithm to compute MST of ($V(G_1)$, F_2-light edges);
7 **return** $C\cup$ (edges picked in step (6));

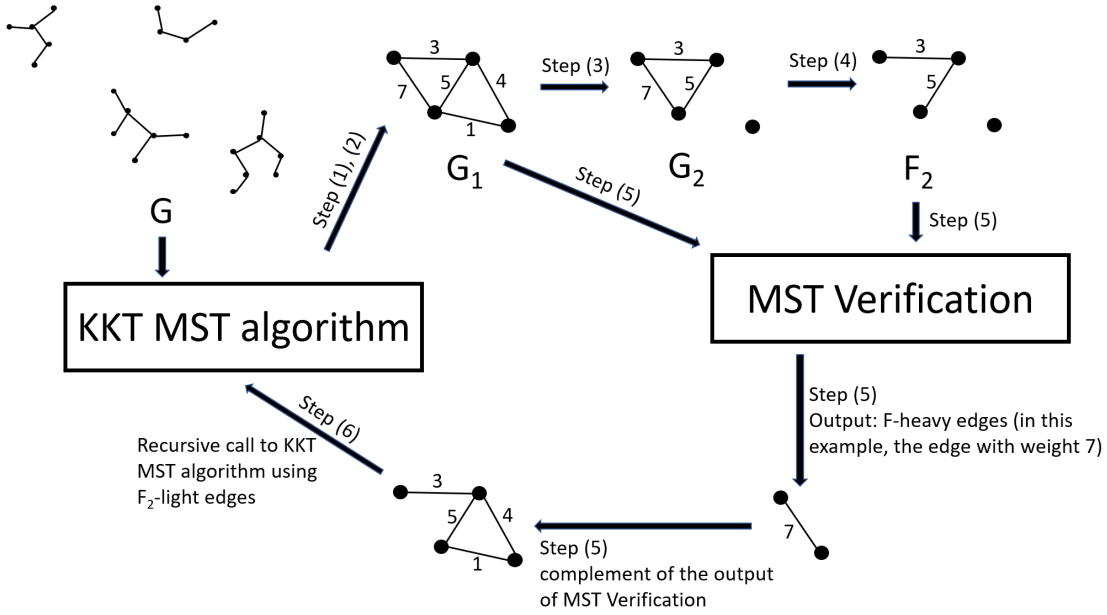

Figure 7: This figure depicts the KKT MST algorithm for a unconnected graph G.

Running time analysis Let $T(m, n)$ denote the expected running time of the KKT MST algorithm on an n-vertex graph with m edges. Here are the running time of the KKT MST algorithm per step:

- Step (1): $O(m + n)$
- Step (3): $O(m + n)$ (visiting every edge and then sampling it)
- Step (4): $T(\frac{m}{2}, \frac{n}{8})$
- Step (5): $O(m + n)$
- Step (6): $T(\frac{n}{4}, \frac{n}{8})$

The running time of the KKT MST algorithm is the summation of the running time in the above steps:

$$T(m,n) = O(m+n) + T(\frac{m}{2}, \frac{n}{8}) + T(\frac{n}{4}, \frac{n}{8})$$

We upper bound $O(m+n)$ by $c(m+n)$. Then,

$$T(m,n) \leq c(m+n) + T(\frac{m}{2}, \frac{n}{8}) + T(\frac{n}{4}, \frac{n}{8})$$

Claim Solution to this recurrence is $T(m,n) \leq 2c(m+n)$.

Proof Ignoring the base cases, we get

$$\begin{aligned} c(m+n) + T(\frac{m}{2}, \frac{n}{8}) + T(\frac{n}{4}, \frac{n}{8}) &\leq c(m+n) + 2c(\frac{m}{2}, \frac{n}{8}) + 2c(\frac{n}{4}, \frac{n}{8}) \\ &= c(m+n) + c(m + \frac{n}{4} + \frac{n}{2} + \frac{n}{4}) \\ &= c(m+n) + c(m+n) \\ &= 2c(m+n) \end{aligned}$$
