**Spanning Trees (Lecture 4)**

**MST:** Classical Algorithms:
- Kruskal
- Prim (Jarník/Dijkstra) '57
- Borůvka

Modern:
- Yao '75 $O(m \log n)$
- Fredman Tarjan '84 $O(m \log^* n)$
- Gabow Galil Spencer Tarjan '85? $O(m \log^* n)$
- Karger (Karger Klein Tarjan) '95 [Randomized]
- Chazelle '97 $O(m \alpha(m,n))$ worse case

---

**Basic Idea of MST algorithms:** Given $G = (V,E)$, edge weights can be any $O(1)$ time

don't use facts about edge wts.

- $m$ edges, $n$ nodes, connected.
- can assume no parallel edges
- assume nodes are numbered 1..n.

Cut rule: take a cheapest edge across some cut.

- this is in an MST. (color it blue)

Cycle rule: take any cycle and drop the heaviest edge on the cycle (color it red).

Thus: Repeatedly apply cut and cycle rules in any order. Get the MST.

Most of simple algorithms (all the ones in Lecture 1) just use the cut rule.

These facts also extend to "matroids" [See HW 7] where we can define
the notion of a cut and cycle analogously. But also often use more of the
simple.
Algorithm 1: Kruskal

Sort edges. Add next edge if it does not create cycle (if it decreases the number of components) or if it crosses a cut in the current graph.

How to check if edge crosses a cut? Disjoint set union (aka Union-find) data structure.

Operations: createSet,
find (c) ← return "root" of the set containing c
Union (r₁, r₂) ← unify the sets whose roots are r₁, r₂.

Using union by rank and path compression:

\[
\begin{align*}
\text{any set of } k \text{ operations takes time } & \Theta(\alpha(n)) \\
\text{(on a universe of size } n) & = O(n + k \alpha(n)) \\
\end{align*}
\]

Various definitions but almost all essentially this:

\[A(m, n) = \begin{cases} 
2n & \text{for } m = 0 \\
A(m-1, A(m, n-1)) & \text{otherwise}
\end{cases}\]

\[\begin{align*}
A(0, n) &= 2^n \\
A(1, n) &= 2^n + 1 \\
A(2, n) &= 2^{2n} + 1 \\
&\vdots
\end{align*}\]

Runtime: \(O(m \log m) + O(n + \max(m, n)) = O(m \log m)\).
Prim's Algorithm: Pick a root vertex; Maintain some root component. C

Pick smallest edge out edge of root component and add it in. (Blue rule)

So want to maintain: for each vertex not in C, the length of shortest edge from C to v. (if any). And quickly want to find min.

Bin heap

| log n | Fib n |
|-------+-------|
| [O(1)] | [O(n)] |

Priority queue: each element has a key.

- insert (e, k)
- delete min (return element with smallest key, remove from priority)
- decrease key (e, k').

log n | delete (dkey + delete)

n | meld [the 2 q's and combine them into 1]

O(1) | O(1) | find min

n = #elements in queue at anytime


Amortized bounds:

So: for each vertex in work load maintain the set of edge min (cheapest) of C

1 decrement for each vertex v

now: update with all edges from v to rest : d(v).

decrease keys

Total time: O(mlog n) + O(m).

Using fibonacci heap: O(mlog n).
Borůvka: For each vertex, pick cheapest edge out.

// Unique edge of it ensures will pick a forest. (also undirected). F.

Contract the edges in F to set new graph G'. Repeat.

Fact: # of components in \((G', F)\) = # vertices of \(G' \leq \frac{|V(G')|}{2} \)

\[ \Rightarrow \text{round} = O(c\log n). \]

Time per round: scan all the edges in \(G_e\): time \(O(m). \) - forget \( E \)

Contract \( G_i \rightarrow G_{i+1} \): time \( O(n \log^2 m) \) [HW1].

\[ \Rightarrow O(m \log n). \]

Could be worse than Prim + Fib Heap? Yes.

Could be worse than \( O(n^2 \log n) \)? Yes.

[HW1]: Borůvka \( \leq O(n^2) \) time.

\[ \leq O(m \log (\log m + 1)) \]

Fredman + Tarjan: \( O(m \log^* n) \)

Very simple idea. work graph is sparse
the \( O(m + n \log n) \) of Prim is doing a lot of
more work in \( n \log n \) than on the \( O(m) \).

"balance it out"?

Start a Prim at vertex. if heap gets "too big", stop.

Recall: \( \min \) bottleneck \( \leq \) deltemin \( \leq O(\log \text{size of heap}). \)

So if heapsize \( \leq K \) then pay \( O(\log K) \) per extract min.
total cost if we keep doing this:

- each edge explored \( \leq 2 \text{ times} \) \( \Rightarrow \) \( O(m) \) decreases by \( \text{deg} \)
- each vertex can be deleted at most \( \text{deg} \) times \( \Rightarrow \) total \# deletions \( \leq O(n) \times \text{deg} \).

N.b. may make loops together if they meet \( O(1) \) time.

In this case stop the process.

even if the size of neighborhood has dropped below \( K \).

Crucial fact: each node deleted pays \( O(\log k) \).

How many trees eventually? Body of each component \( \geq K \Rightarrow \) \# trees \( \leq 2m/n_k \).

So from \( G_i \) with \( m_i \) edges, \( n_i \) nodes

\[ k_i \]

\[ G_{kt} \text{ with } m_{kt} \text{ edges, } n_{kt} \leq \frac{2m_{kt}}{k_i} \]

What is \( k_i \)? set \( k_i \) so that \( O(m_i) = O(m_{kt}) \Rightarrow k_i = 2 \)

\[ 2m_i = n_i \log k_i \leq O(m_{kt}) \text{ nodes per component.} \]

\[ \Rightarrow n_i \text{ nodes, now } n_{i+i} \leq \frac{2m_{i+i}}{2^{m_i/n_i}} \Rightarrow k_i = \frac{2m_{i+i}}{n_{i+i}} \geq 2 \]

\[ \Rightarrow k_i \text{ increases exponentially } \Rightarrow 2^{\log k_i} \Rightarrow 2^{m_i/n_i} \]

\[ 2m_i/n_i \rightarrow 2^{m_i/n_i} \rightarrow 2^{2^{m_i/n_i}} \ldots \text{ in } 2^{m_i/n_i} \text{ rounds } k_i=2^{m_i/n_i}. \]
Q: What about my folly; can we use that to set improvements?

Just be greedy we may not remove edges fast enough.

Eg.

very heavy edges here. All the work happens in
the improvements, not across it.

We should use the red rule (cycle rule)
to kill some of the edges. But that
is next time.

Can come another day , and set \( O(\text{only } \beta(nm)) \) runtime, but won't
do it here.

History after this: KKT, (randomized) and then

Chapelle (97) \( O(\text{non } \alpha(nm)) \) - initially with an extra log.

have [soft heaps] may corrupt some of the keys in
the heap!

Petttie & Ramachandran. an optimal MST algorithm:

Shows that if \( \text{MST}^*(nm) \) is optimal decision (near optimality of

MST on merge node graphs, then

explicit algorithm that takes \( O(\text{MST}^*(nm)) \) time on the pointer machine

model.

on all m & n vertex graphs.

Big Q: does I a deterministic \( O(\text{mmn}) \) time MST algorithm?

Enough to show that \( \text{MST}^*(mn) = O(\text{m}) \).

just a tree that maps
just counts # of comparisons
of edges done by algo?
Aside: (a) models of computation.
(b) is amortization necessary for (i) set union-find
(ii) Fibonacci heaps?

Union find: a lower bound in the cell probe model that
needs \( \max(m, n) \) time for \( m \) finds over \( n \) elements
[Alon et al.]

Also if worst-case complexity, on set \( \mathcal{O}(\log \log n) \) in cell probe model
[Fredman, Saks].

Fibonacci Heaps: [Brodal, Legenhius, Tarjan SODA 96] bounds match Fibheaps
but in the worst case [and pointer machine], highly complicated
RAM-machine soln by Brodal [SODA 96].

Cell probe: memory cells, size \( w \), we pay only for memory accesses.

Transdichotomous RAM: words of size \( w \) (typically \( \geq \log n \)). Each operation
manipulates \( O(1) \) words at a time; finite set of operations. Based on what OPs:
allowed
- Word RAM: ops standard to C: +, -, *, /, mod, shift etc.
- \( AC^0 \) ram: all ops implementable in \( AC^0 \) (constant depth circuits)

Pointer Machine: Memory is a giant DAG. With constant branchy factor.
Input to operation is a pointer. No pointer arithmetic allowed.