Randomized Graph Algorithms
so far, we saw perfect bipartite matching.

Min Cut - in an undirected graph, find the min. number of edges whose removal disconnects the graph.

![Diagram of a graph with a cut of 3 edges]

- can use max flow - min cut algorithms
- to find min cut with vertices s,t on opposite sides (try various t) - use network flow algorithms
- improved to $O(m \cdot n)$ Matula '87

Today: Karger's randomized alg. $O(n^4)$ runtime can be improved to $O(n^2 \text{ poly}(\log n))$
which beats best deterministic alg.
- also, it's very simple

Karger's Alg. '93
while $|V| > 2$
  I choose a random edge and contract it
output edges between the 2 vertices
Example

Note: the graph has multiple edges.

Idea for why this works: consider a min. cut $T \subseteq E$.

The alg. finds $T$ so long as it never contracts an edge of $T$ and we expect $T$ to be fairly small relative to $|E| = m$.

Thm. The alg. outputs a min. cut with $\Pr = \frac{2}{n(n-1)}$

Pf. will show alg. outputs $T$ with $\Pr \geq \frac{2}{n(n-1)}$

Let $|T| = k$.

Note that any vertex $v$ gives a cut of size $\deg(v)$.

Since $k$ is min cut, $\deg(v) \geq k \forall v$.

Thus $m \geq n \cdot \frac{k}{2}$.

$\Pr [\text{contract edge of } T \text{ in 1st iteration}] \leq \frac{k}{n \cdot \frac{k}{2}} = \frac{2}{n}$

In $i$th iteration we have $n-i+1$ vertices.

any cut is still a cut of original, so $\leq k$.

so vertex degrees $\geq k$.

and # edges $\geq \frac{(n-i+1)k}{2}$

$\Pr [\text{contract an edge of } T \text{ in } i^{th} \text{ iteration}] \leq \frac{2}{n-i+1}$
\[ \Pr [T \text{ survives}] \]
\[ \geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{3}\right) \]
\[ = \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{2}{4} \cdot \frac{1}{3} \]
\[ = \frac{2}{n(n-1)} > \frac{2}{n^2} \]

\[ \Pr [\text{failure}] < \left(1 - \frac{2}{n^2}\right) \]

How often should we repeat this process?
Recall \( (1 - \frac{1}{a})^a < \frac{1}{e} \) from \( e^{\frac{1}{x}} < \frac{1}{1-x} \)
\[ \frac{1}{e^x} > 1-x \]
So repeating \( \frac{n^2}{2} \) times

gives \( \Pr [\text{failure}] < \frac{1}{e} \)

Run Time: One execution can be implemented in \( O(n^2) \) time, so total is \( O(n^4) \)
Can be improved to \( O(n^2 \text{ poly}(\log n)) \)

How did we use randomness?

Just to pick random order of edges,
Can interpret the alg. as building spanning tree from that edge ordering
Last edge gives cut.
Random Sampling - for Min. Spanning Tree.

MST - Given (undirected) graph \( G = (V, E) \) with edge weights \( w: E \to \mathbb{R}^+ \), find a min. weight spanning tree.

- Connected acyclic
- Sum of edge weights
- Reaches all vertices

Assume distinct edge weights (just break ties consistently).
Can generalize to spanning forest of disconnected graph.

Two basic rules:

**Exclusion rule**
If cycle \( C \) has max weight edge \( e \), then \( e \notin \text{MST}(G) \) - delete \( e \) and continue.

**Example**

![](image)

**Inclusion rule**
If \( u \)'s min. weight incident edge is \( e = vu \), then \( vu \in \text{MST}(G) \)

Contract \( vu \) and continue with smaller graph.

**Example**

![](image)
Basically all MST alg. work via these rules: (in fact, mostly inclusion rule)

**Kruskal's Alg. '56**

repeat
- \( e = (u, v) = \text{min. weight edge} \)
- put \( uv \) in \( T \) and contract \( uv \). (implementations don't explicitly contract)

uses inclusion rule

implementation - sort edges by weight + union find:

\[ O(m \log n) \] time

\( n = \# \text{vertices} \quad m = \# \text{edges} \)

**Prim's Alg. '57**

- pick start vertex \( s \)

repeat
- find min weight edge leaving \( s \) \( e = su \)
- put \( e \) in \( T \), contract \( e \) (implementations don't explicitly contract)

uses inclusion rule

implementation: heap \( O(m \log n) \)

Fibonacci heaps give \( O(n \log n + m) \)

linear time if \( m > n \log n \)

**Boruvka's Alg. '26** - great parallel algorithm!

idea: in one step, apply inclusion rule

until every vertex has-participated in a contraction

then \( \# \text{vertices} \leq \frac{n}{2} \) so repeat \( \log n \) times

Result is \( O(m \log n) \) alg.

(can be implemented in \( O(m) \))
History of MST alg:
Yao '75 $O(m \log \log n)$
Cheriton (David R.) & Tarjan '76
Fredman, Tarjan '85 $O(m \log^* n)$
Chazelle '97 $O(m \alpha(n))$

OPEN - is there a linear time alg?

Karger '93 Las Vegas Alg. - linear expected time.
2 ideas - random sampling
- exclusion rule to reduce # edges.
  + Boruvka to reduce # vertices

$\text{MST}(E)$ - return MST of each connected comp. of $G = (V,E)$
  take random subset $R \subseteq E$ of size $r$ to be chosen
  $T = \text{MST}(R)$
  for each edge $uv \in E$ do
    classify $uv$ as heavy if $uv$ makes a cycle with
    edges of $T$ and $uv$ is heaviest in the cycle; else light
  $E = E \setminus \text{heavy edges}$
  return $\text{MST}(E)$

Note: correct by exclusion rule:
- $T$ is MST of whole graph iff all edges \& $T$ are heavy.
Classification of heavy edges can be done in \( O(m+n) \) time
- this includes "verification" - is \( T \) the MST?
  (this is a bit complicated).

**Sampling Lemma**
\[
E(\# \text{ light edges}) \leq \frac{m \cdot n}{r}
\]
(won't prove it. See

\[\text{Analysis: expected run time}\]
\[
T(m,n) = T(r,n) + O(m+n) + T\left(\frac{mn}{r}, n\right)
\]
(recursive call on \( R \), classify

with \( r = 2n \) this becomes
\[
T(m,n) = T(2n,n) + T\left(\frac{m}{2}, n\right) + O(m+n)
\]
with Kruskal/Baruvka in place of first recursive call, this
already gives \( O(n \log^2 n + m) \) (without Fibonacci heaps)

Final idea: Karger, Klein, Tarjan '94
on each recursive call, do 3 Baruvka steps first
this reduces \# vertices \( \leq \frac{n}{8} \) with \( O(n+m) \) work.

\[
T(m,n) \leq T\left(\frac{m}{4}, \frac{n}{8}\right) + T\left(\frac{m}{2}, \frac{n}{8}\right) + d(m+n), \quad d \text{ constant}
\]

Prove by induction \( T(m,n) \leq c(m+n) \)
\[
T(m,n) \leq c\left(\frac{m}{4} + \frac{n}{8}\right) + c\left(\frac{m}{2} + \frac{n}{8}\right) + d(m+n)
\]
\[
= (\frac{c}{2} + 2d)m + (\frac{c}{2} + d)n
\]
\[
\leq c(n+m) \quad \text{so long as} \quad \frac{c}{2} + d \leq c \quad \text{i.e.} \quad c \geq 2d
\]

So expected run time is \( O(m+n) \).
Summary of techniques for randomized algorithms

- Sampling: MST, quickselect, quicksort
- Abundance of witnesses: primality testing
- Fingerprinting: polynomial identities
- Backwards analysis of randomized incremental algs: linear programming
- Markov chain, random walk: SAT