CS 3100

MST, shortest path trees (SSP)
Announcements

- next hw - due next Friday, oral grading
- Midterm: 2 weeks from today
Last time: MST

Key idea: for any vertex cut OS \& V-S, the smallest edge between S \& V-S will be in the MST.

Borůvka's (or Sollin's) algorithm:

- safe

Add all such edges for each component left in G.

Recurse.

Borůvka's algorithm run on the example graph. Thick edges are in F. Arrows point along each component's safe edge. Dashed (gray) edges are useless.
More precisely:
- Count components of \( G \) using BFS/DFS
- As we go, label each vertex \( u \) with its component #

While (\# components > 1):
- How many iterations?
- Compute array \( S[1..n] \)
  where \( S[i] = \min \text{ weight edge with one endpoint in component } i \)

How?
- Consider each edge \( uv \):
  - if endpoints have same label, ignore
  - if not, check if \( w(uv) \) beats current \( S[\text{label}(u)] \)
    or \( S[\text{label}(v)] \)
Runtime:

how many iterations?

\[ \Omega \rightarrow \Omega \rightarrow \Omega \rightarrow \Omega \rightarrow \Omega \]

\(\leq\) worst case, \# components divides by 2

\[ T(n) \leq T(\frac{n}{2}) + 1 \]

(One iteration reduce \# comp by half)

\# rounds = \(O(\log_2 n)\)

Total: \(O(m \log n)\)
Other algorithms: (Prim)

Grow a single $T$:

- start from any vertex $v$
- set $T = \{v\}$
- while $|T| = n$:
  - find safe edge from $T$ to $V - T$ and add

(Really Jarník’s from 1929)

How to implement?

heap!
\[ \log x^y = y \log x \]

**Traverse(s):**
- put \( s \) into the bag
- while the bag is not empty
  - take \( v \) from the bag
  - if \( v \) is unmarked
    - mark \( v \)
    - for each edge \( vw \)
      - put \( w \) into the bag

**Runtime:**
- If heap is size \( m \)
  - \( O(\log m) \) \( \leq \) \( O(\log n^2) \)
  - \( = O(\log n) \)
- Each vertex \( v \) gets added \( d(v) \) times:
  - \( \sum d(v) \cdot \log n = \log n \cdot (\sum d(v)) \)
  - \( = O(m \log n) \)
Kruskal’s algorithm (1956, motivated by Boruvka)

Scan all edges in increasing order.

If edge is safe, add it.

Implementation:
A bit more complex - uses Union-Find data structure (more to come...)
Next problem: Shortest paths

Goal: Find shortest path from s to v.

We'll think directed, but really could do undirected w/no negative edges.

Motivation:
- maps
- routing

Usually, to solve this, need to solve a more general problem:

Find shortest paths from s to every other vertex. Called the Single-Source Shortest Path Tree.

SSSP
Some notes:

- Why a tree?

If $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow v$ and $s \rightarrow a \rightarrow x \rightarrow y \rightarrow d \rightarrow u$ are shortest paths, then $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow u$ is also a shortest path.

- Negative edges?

If $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow v$ and $s \rightarrow a \rightarrow x \rightarrow y \rightarrow d \rightarrow u$ are shortest paths, then $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow u$ is also a shortest path.
Important to realize: \( \text{MST} \neq \text{SSSP} \)

Why? \( \text{SSSP} \) is rooted and directed.
- \( \text{SSSP} \) for every vertex (these are different!)
Computing a SSSP:
(Ford 1956 & Dantzig 1957)

Each vertex will store 2 values.

(Think of these as tentative shortest paths)

- \( dist(v) \) is length of tentative shortest \( s \rightarrow v \) path
  (or \( \infty \) if don't have an option yet)

- \( pred(v) \) is the predecessor of \( v \) on that tentative path \( s \rightarrow v \)
  (or NULL if none)

Initially:
We say an edge \( uv \) is tense if \( \text{dist}(u) + w(u \rightarrow v) < \text{dist}(v) \)

If \( u \rightarrow v \) is tense:

use the better path!

So, relax!

\[
\text{RELAX}(u \rightarrow v): \\
\text{dist}(v) \leftarrow \text{dist}(u) + w(u \rightarrow v) \\
\text{pred}(v) \leftarrow u
\]
Algorithm:
Repeatedly find tense edges and relax them. When none remain, the \( \text{Pred}(v) \) edges form the SSSP tree.

\[\begin{array}{|c|}
\hline
\text{InitSSSP}(s):
\begin{align*}
\text{dist}(s) & \leftarrow 0 \\
\text{pred}(s) & \leftarrow \text{Null}
\end{align*}
\text{for all vertices } v \neq s
\begin{align*}
\text{dist}(v) & \leftarrow \infty \\
\text{pred}(v) & \leftarrow \text{Null}
\end{align*}
\hline
\end{array}\]

\[\begin{array}{|c|}
\hline
\text{GenericSSSP}(s):
\begin{align*}
\text{InitSSSP}(s)
\text{put } s \text{ in the bag}
\text{while the bag is not empty}
\text{take } u \text{ from the bag}
\text{for all edges } u \rightarrow v
\text{if } u \rightarrow v \text{ is tense}
\text{ RELAX}(u \rightarrow v)
\text{put } v \text{ in the bag}
\hline
\end{array}\]

To do: which “bag”?
Dijkstra (59)
(Actually, Leyzorek et al. '57, Dantzig '58)

Make the bag a priority queue:
Keep "explored" part of the graph, $S$.

Initially, $S = \{s\}$ and $\text{dist}(s) = 0$ (all others $\text{NULL} + \infty$)

While $S \neq V$:
1. Select node $v \notin S$ with one edge from $S$ to $v$
   with:
   $$\min_{e=(u,v), u \in S} \text{dist}(u) + w(u \rightarrow v)$$
2. Add $v$ to $S$, set $\text{dist}(v) = \text{pred}(v)$
Four phases of Dijkstra’s algorithm run on a graph with no negative edges. At each phase, the shaded vertices are in the heap, and the bold vertex has just been scanned. The bold edges describe the evolving shortest path tree.
Correctness

Thm: Consider the set $S$ at any point in the algorithm. For each $u \in S$, the distance $\text{dist}(u)$ is the shortest path distance (so $\text{pred}(u)$ traces a shortest path).

Pf: Induction on $|S|$: 

Base case:
**TH:** Spps claim holds when \(|S| = k - 1\).

**TS:** Consider \(|S| = k\): algorithm is adding some \(v\) to \(S\).
Back to implementation and run time.

For each $v \in S$, could check each edge and compute $D[v] + w(e)$ runtime?
Better: a heap!

When v is added to S:
- look at v's edges and
  either insert w with key \( \text{dist}(v) + w(v \rightarrow w) \)
  or update w's key
  if \( \text{dist}(v) + w(v \rightarrow w) \) beats current one

Runtime:
- at most m ChangeKey operations in heap
- at most n inserts/removes