CS 3100

BFS, MST
Announcements

- HW due today
- Next HW: oral grading, end of next week
- Midterm: Wednesday October 18
  review on Monday in class
Last time:
- Graph representations
- Graph traversals: DFS

Idea: determine connectivity - can we reach vertex u from vertex v?

(We're doing undirected, but all can be modified for directed - usually just by making sure edge lists have only outgoing edges.)
Pseudocode: two versions

**RecursiveDFS(v):**
- if v is unmarked
  - mark v
  - for each edge vw
    - RecursiveDFS(w)

**IterativeDFS(s):**
- Push(s) \(O(1)\)
- while the stack is not empty
  - \(v \leftarrow \text{Pop} \ O(1)\)
  - if v is unmarked
    - mark v
    - for each edge vw
      - Push(w) \(O(1)\)

\(O(m+n)\) total

Really, building a "tree":

DFS tree:
General traversal strategy in DFS, bag = stack

**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

Q: Can we use a different "bag"?
- Queue
BFS: use a queue

**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

"distance" traversal

$O(m+n)$
BFS vs. DFS.

- Both can tell if 2 vertices are connected.
- Both can be used to detect cycles.
  How? If revisit an edge, must have some cycle.
- Both run in $O(V+E) = O(n^2)$. 

**Difference:**

"long - thin"    "bushy"

A depth-first spanning tree and a breadth-first spanning tree of one component of the example graph, with start vertex $a$. 

- Both run in $O(V+E) = O(n^2)$. 

**Difference:**

"long - thin"    "bushy"
**Dfn:** A **tree** is a maximal acyclic graph, always with \( n-1 \) edges.

(DFS + BFS can both be used to get trees.)

**Dfn:** A **component** of a graph is a maximal connected subset of \( G \).

---

![Diagram of a tree and components](image)
New setting: a weighted graph

A graph $G = (V, E)$ together
with a weight function $w: E \rightarrow \mathbb{R}$
that gives a weight $w(e)$ to each edge.

In this setting, finding shortest paths is much more interesting!

We'll start with a more basic question:

What is the best tree contained in the graph?

acycles

\[\text{minimum}\]
Problem: Minimum Spanning Tree

Find a set of edges which connects all vertices and is as small as possible.

A weighted graph and its minimum spanning tree.

Applications: obvious
Strategy:

- We'll start by assuming edge weights are unique: $w(e) \neq w(e')$.

How to get started?

Idea: Choose Smallest Edge. (greedy!)
Intermediate stage

Now suppose we have a partial MST—a forest.

Better: Add min edge if not forming a cycle.
Lemma: Let $S$ be any subset of $V$ ($\neq \emptyset$ or $V$). Let $e$ be the edge of minimum weight between $S$ and $V - S$. Then $e$ is in any MST of $G$.

Proof:

Suppose $e$ is not in MST.

Let $T$ be MST not containing $e$. Some edge of $T$ must leave $S$, say $e'$. 
Prove $T - e' + e$ is better:

- minimum is clear:
  \[ w(e) < w(e') \]

$T$ was a tree: \( \forall u,v \), there was a path in $T$ from $u$ to $v$.

If path didn't use $e'$, still there.

If it did: let $e = xy$.

Use $u$-to-$x$ path $+ e'$, $+ y$ to $v$ path.

so $T - e' + e$ is still a tree. \( \blacksquare \)
A bit further: Take a forest $F$:

Define a safe edge for any component of $F$ as the minimum weight edge with only one endpoint in that component.

A useless edge is one not in $F$ with both endpoints in the same component.

Note: Prior lemma says any safe edge can be added to the MST!
Algorithm:
Start with \( n \) vertices.
Compute the safe edges.
Add them.
Recurse on the new forest.

Example:
This is Borůvka’s algorithm, from 1926.
(Also others—often called Sollin’s algorithm.)

Pseudo code:

**Borůvka**\((V, E)\):

\[
F = (V, \emptyset) \\
\text{count} \leftarrow \text{COUNTANDLABEL}(F) \\
\text{while count} > 1 \\
\quad \text{ADDALLSAFEEDGES}(E, F, \text{count}) \\
\text{count} \leftarrow \text{COUNTANDLABEL}(F) \\
\text{return } F
\]

**ADDALLSAFEEDGES**\((E, F, \text{count})\):

\[
\text{for } i \leftarrow 1 \text{ to count} \\
\quad S[i] \leftarrow \text{NULL} \quad \langle \text{sentinel: } w(\text{Null}) := \infty \rangle \\
\text{for each edge } uv \in E \\
\quad \text{if label}(u) \neq \text{label}(v) \\
\qquad \text{if } w(uv) < w(S[\text{label}(u)]) \\
\qquad \quad S[\text{label}(u)] \leftarrow uv \\
\qquad \text{if } w(uv) < w(S[\text{label}(v)]) \\
\qquad \quad S[\text{label}(v)] \leftarrow uv \\
\text{for } i \leftarrow 1 \text{ to count} \\
\quad \text{if } S[i] \neq \text{Null} \\
\qquad \text{add } S[i] \text{ to } F
\]

Essentially:

- Find min nbr for each vertex.
- Label each component:
  - use DFS/BFS
- Find min edge leaving + add to F
- repeat
Runtime:

Sort edges (once):

At each stage, get \( Y_2 \) many components (worst case)

\# stages:

\[ T(n) = T \left( \frac{n}{2} \right) + 1 \]

\( \Rightarrow \) \( O(\log n) \) stages

\( \Rightarrow \) \( O(m \log n) \) algorithm
Other algorithms:

Prim's algorithm: add a safe edge, one at a time
(Really Jarnik's from 1929)

How to implement?