• **DISCLAIMER:** These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

• Main topics:
  - Homework 2 out.
  - Why does BFS run in \(O(m)\) Time.
  - Minimum Spanning Trees, cut rule.

• Comments from Last Time:
  - Weighted shortest path example went too quickly: example handout.
  - Why is running time of queue algorithm \(O(m)\).

We start with a few comments about why BFS is fast.

Outer and Inner loop: consider an algorithm of the form:

1. Loop through all vertices \(u\)
   
   (a) Loop through all neighbors of \(u, v \in N(u)\)

The inner loop can take \(O(n)\), while the outer is \(O(n)\) as well.

These loops combined do better than \(O(n^2)\): the true total cost is \(\sum_u |N(u)| = O(m)\), because the outer loop is not always expensive. For example consider the star: the inner loop does 1 for \(n - 1\) vertices and \(n - 1\) for 1 vertex, for a total of \(O(n)\).

We move onto a different problem: minimum spanning trees, often abbreviated as MST. Given a weighted undirected graph, we want to find a subset of edges of minimum total weight to make things connected. Note that maximum spanning tree has the same abbreviation. **we will use MST to denote minimum spanning tree unless specified otherwise.**

This subset is a tree. A tree is a subgraph meeting any two out of the following three conditions:

1. is connected,
2. \(n - 1\) edges,
3. has no cycles
For example, if the graph has no negative cycles, the union of all $s \rightarrow v$ shortest paths is a tree. The proof is via the ‘from’ edges in question 4 of your homework.

For simplicity we assume that all edge weights are distinct. The goal of this lecture is to show that the following greedy algorithm due to Kruskal works.

1. Initialize $H = \emptyset$.
2. Sort edges in increasing order of weights.
3. Loop through edges in increasing order of weights.
   (a) If endpoints of $e$ are not connected in $H$.
      i. Add $e$ to $H$.

The correctness of this algorithm crucially relies on the following fact, known as the cut property. To define this property, we need the definition of a cut. It refers to a set of edges, but is defined by a subset of vertices $S$.

**Definition 0.1.** A cut given by $S \subseteq V, E(S, V \setminus S)$, is the set of edges between $S$ and $V \setminus S$.

**Theorem 0.2.** For any cut $S$, the minimum weight edge on the cut is in the MST.

The proof relies on the following operation that transforms between trees.
1. Add an edge $e = uv$.
2. Identify the path in $T$ between $u$ and $v$
3. Remove any edge from this path.

This still gives a tree because the number of edges remains the same, and connectivity is preserved.

**Proof.** The proof is by contradiction.
Suppose $T$ is a minimum spanning tree that does not use $e = uv$. Without loss of generality by symmetry, assume $u \in S$ and $v \notin S$.

Then consider the path in $T$ between $u$ and $v$. This path goes from $S$ to $V \setminus S$, so must contain an edge $e'$ in the cut $E(S, V \setminus S)$.

Since $e$ is the minimum weight edge in $E(S, V \setminus S)$, we have

$$w_{e'} > w_e,$$

so adding $e$ to $T$ and removing $e'$ from the cycle gives a tree with smaller weight, a contradiction.

This gives the correctness of the above algorithm. Implemented directly, we spend $O(m \log n)$ sorting the edges, then $O(n)$ time to check connectivity between vertices in $H$ at each step. As we do this $m$ times, the total cost is $O(nm)$. 

2