Outline

Minimum Spanning Tree (MST), CP3 Section 4.3
• Motivating Example & Some Definitions

Two Algorithms to solve MST (you have a choice!)
• Prim’s (greedy algorithm with PriorityQueue)
  – PriorityQueue is discussed in Lecture 03-04 (not just Lecture 02)
• Kruskal’s (greedy algorithm, uses sorting and UFDS)
  – UFDS is discussed in Lecture 05
Definitions that we have learned before

- **Tree T**
  - $T$ is a *connected graph* that has $V$ vertices and $V-1$ edges
  - Important: One unique path between any two pair of vertices in $T$

- **Spanning Tree** $ST$ of connected graph $G$
  - $ST$ is a tree that spans (covers) every vertices in $G$
  - Recall the **BFS and DFS Spanning Tree**

Sorting problem & several sorting algorithms

- Rearrange set of objects so that every pair of objects $(a, b; a < b)$ in the final arrangement satisfies that $a$ is before $b$
Is This A Tree?

1. Yes, why __________
2. No, why __________
Are the edges highlighted in red part of a spanning tree of the original graph?

1. Yes, why __________
2. No, why __________
Motivating Example

Government Project

• Want to link rural villages with roads
• The cost to build a road depends on the terrain, etc
• You only have limited budget
• How are you going to build the roads?
More Definitions (1)

• **Vertex set** $V$ (e.g. street intersections, houses, etc)

• **Edge set** $E$ (e.g. streets, roads, avenues, etc)
  - Generally undirected (e.g. bidirectional road, etc)
  - Weighted (e.g. distance, time, toll, etc)

• **Weight function** $w(a, b): E \rightarrow \mathbb{R}$
  - Sets the weight of edge from $a$ to $b$

• **Weighted Graph**: $G(V, E), w(a, b): E \rightarrow \mathbb{R}$

• **Connected** undirected graph $G$
  - There is a path from any vertex $a$ to any other vertex $b$ in $G$
More Definitions (2)

- **Spanning Tree** $\text{ST}$ of $\text{G}$
  - Let $w(\text{ST})$ denotes the total weight of edges in $\text{ST}$
    \[ w(\text{ST}) = \sum_{(a,b) \in \text{ST}} w(a,b) \]

- **Minimum Spanning Tree (MST)** of connected undirected weighted graph $\text{G}$
  - $\text{MST}$ of $\text{G}$ is an $\text{ST}$ of $\text{G}$ with the minimum possible $w(\text{ST})$
More Definitions (3)

- **The (standard) MST Problem**
  - **Input:** A connected undirected weighted graph \( G(V, E) \)
  - **Select some edges of** \( G \) **such that the graph is still connected, but with minimum total weight**
  - **Output:** Minimum Spanning Tree (MST) of \( G \)
Example

The Original Graph

A Spanning Tree
Cost: $4+4+6+6 = 20$

An MST
Cost: $4+6+6+2 = 18$
Are the edges highlighted in red part of an MST of the original graph?

1. No, we must replace edge 0-3 with edge 2-3
2. No, we must replace edge 1-2 with 0-2
3. Yes
MST Algorithms

MST is a well-known Computer Science problem

Several efficient (polynomial) algorithms:

• Jarnik’s/Prim’s greedy algorithm
  – Uses PriorityQueue Data Structure taught in Lecture 02-04

• Kruskal’s greedy algorithm
  – Uses Union-Find Data Structure taught in Lecture 05

• Boruvka’s greedy algorithm (not discussed here)

• And a few more advanced variants/special cases...
Do you still remember Prim’s/Kruskal’s algorithms from CS1231?

1. Yes and I also know how to implement them
2. Yes, but I have not try implementing them yet
3. I forgot that particular CS1231 material... but I know it exists
4. Eh?? These two algorithms were covered before in CS1231??
5. I haven’t took CS1231 😞
Grid MST, ICPC SG Prelim 2015

https://open.kattis.com/problems/gridmst/
https://open.kattis.com/problems/gridmst/statistics

If you know basic MST algorithm..., you still can **NOT** solve this problem

But you can solve the simplified form when \( N \) is small (\( 1 \leq N \leq 1000 \))
Prim’s Algorithm

Very simple pseudo code

T ← {s}, a starting vertex s (usually vertex 0)
enqueue edges connected to s (only the other ending
vertex and edge weight) into a priority queue PQ
that orders elements based on increasing weight

while there are unprocessed edges left in PQ
    take out the front most edge e
    if vertex v linked with this edge e is not taken yet
        T ← T ∪ v (including this edge e)
        enqueue edges connected to v (as above)

T is an MST
MST Algorithm: Prim’s

Ask VisuAlgo to perform Prim’s **from various sources** on the sample Graph (CP3 4.12), **then try other graphs**

In the screen shot below, we show the start of **Prim(0)**
Easy Java Implementation

You just need to use two known Data Structures to be able to implement Prim’s algorithm:

1. A priority queue (we can use Java PriorityQueue), and
2. A Boolean array (to decide if a vertex has been taken or not)

With these DSes, we can run Prim’s in $O(E \log V)$

- We process each edge once, $O(E)$
  - Each time, we Insert/ExtractMax from a PQ in $O(\log E)$
  - As $E = O(V^2)$, we have $O(\log E) = O(\log V^2) = O(2 \log V) = O(\log V)$

Let’s have a quick look at PrimDemo.java
Why Prim’s Works? (1)

First, we have to realize that Prim’s algorithm is a **greedy algorithm**

This is because **at each step**, it always try to select the next valid edge e with **minimal weight** (greedy!)

Greedy algorithm is usually simple to implement

- However, it usually requires “proof of correctness”
- You will see such proof like this again in CS3230
- Here, we will just see a quick proof
Why Prim’s Works? (2)
see visual explanation in the next two slides

Let $T$ be the spanning tree of graph $G$ generated by Prim’s algorithm and $T^*$ be the spanning tree of $G$ that is known to have minimal cost

- If $T == T^*$, we are done
- If $T != T^*$
  - Let $e_k = (u, v)$ be the first edge chosen by Prim’s algorithm at the $k$-th iteration that is not in $T^*$
  - Let $P$ be the path from $u$ to $v$ in $T^*$, and let $e^*$ be an edge in $P$ such that one endpoint is in the tree generated at the $(k-1)$-th iteration of Prim's algorithm and the other is not
    - i.e. one endpoint of $e^*$ is $u$ or one endpoint is $v$, but the endpoints are not $u$ and $v$
Why Prim’s Works? (3)
see visual explanation in the next slide

• If $T \neq T^*$ (continued)
  – If the weight of $e^*$ is less than the weight of $e_k$, then Prim's algorithm would have chosen $e^*$ on its $k$-th iteration
    • So, it is certain that $w(e^*) \geq w(e_k)$
    • When $e^*$ has weight equal to that of $e_k$, the choice between the $e^*$ or $e_k$ is arbitrary
    • Whether the weight of $e^*$ is greater than or equal to $e_k$, $e^*$ can be substituted with $e_k$ while preserving minimal total weight of $T^*$
  – This process can be repeated until $T^*$ is equal to $T$
    • Thus we can show that the spanning tree generated by any instance of Prim's algorithm is a minimal spanning tree
Our Prim’s algorithm reports this MST $T$

Suppose that this is the optimal MST $T^*$

$e_1 = (0-1)$ at iteration 1
$P = 0-2-1$ in $T^*$
$e^* = (0-2)$
If we substitute $e_1$ with $e^*$, we can transform $T$ to $T^*$
Coming up next: Kruskal’s algorithm

5 MINUTES BREAK
Kruskal’s Algorithm

Very simple pseudo code

```
sort the set of E edges by increasing weight
T ← {}
while there are unprocessed edges left
    pick an unprocessed edge e with min cost
    if adding e to T does not form a cycle
        add e to T
T is an MST
```
MST Algorithm: Kruskal’s

Ask VisuAlgo to perform Kruskal’s on the sample Graph (CP3 4.11), then try other graphs

In the screen shot below, we show the start of Kruskal (there is no parameter for this algorithm)
Why Kruskal’s Works? (1)

Kruskal’s algorithm is also a greedy algorithm

Because at each step, it always try to select the next unprocessed edge $e$ with minimal weight (greedy!)

Simple proof on how this greedy strategy works

• Let’s define a loop invariant: Every edge $e$ that is added into $T$ by Kruskal’s algorithm is part of the MST
Loop invariant: Every edge $e$ that is added into $T$ by Kruskal’s algorithm is part of the MST.

```
sort $E$ edges by increasing weight
$T \leftarrow \{\}$
while there are unprocessed edges left
    pick an unprocessed edge $e$ with min cost
    if adding $e$ to $T$ does not form a cycle
        add $e$ to $T$
$T$ is an MST
```

Kruskal’s algorithm has a special cycle check before adding an edge $e$ into $T$. Edge $e$ will never form a cycle.

At the start of every loop, $T$ is always part of MST.

At the end of the loop, we have selected $V-1$ edges from a connected weighted graph $G$ without having any cycle. This implies that we have a Spanning Tree.
Why Kruskal’s Works? (3)

Connect 0 and 3
The next smallest edge

Loop invariant: Every edge \( e \) that is added into \( T \) by Kruskal’s algorithm is part of the MST.

sort \( E \) edges by increasing weight
\( T \leftarrow \{\} \)
while there are unprocessed edges left
    pick an unprocessed edge \( e \) with min cost
    if adding \( e \) to \( T \) does not form a cycle
        add \( e \) to \( T \)
\( T \) is an MST

By keep adding the next unprocessed edge \( e \) with min cost, \( w(T \cup e) \leq w(T \cup \text{any other unprocessed edge that does not form cycle}) \).

At the start of every loop, \( T \) is always part of \( \text{MST} \).

At the end of the loop, the Spanning Tree \( T \) must have minimal weight \( w(T) \), so \( T \) is the final \( \text{MST} \).
Kruskal’s Implementation (1)

sort $E$ edges by increasing weight // $O(E \log E)$

$T \leftarrow \{\}$

while there are unprocessed edges left // $O(E)$
    pick an unprocessed edge $e$ with min cost // $O(1)$
    if adding $e$ to $T$ does not form a cycle // $O(\?)$
        add $e$ to the $T$ // $O(1)$

$T$ is an MST

To sort the edges:

• We use **EdgeList** to store graph information
• Then use “any” sorting algorithm that we have seen before

To test for cycles:

• We use **Union-Find Disjoint Sets**
Sorting Edges in Edge List

Adjacency Matrix/List that we have learned previously are not suitable for edge-sorting task!

To sort EdgeList, we use one liner Java Collections.sort :O

- Yeah, you don’t have to use merge/quick sort in CS1020... :O
Kruskal’s Implementation (2)

sort E edges by increasing weight // O(E log E)
T ← {}
while there are unprocessed edges left // O(E)
    pick an unprocessed edge e with min cost // O(1)
    if adding e to T does not form a cycle // O(α(V)) = O(1)
        add e to the T // O(1)
T is an MST

To sort the edges, we need O(E log E)
To test for cycles, we need O(α(V)) – small, assume constant O(1)
In overall
• Kruskal’s runs in O(E log E + Eα(V)) // E log E dominates!
• As E = O(V^2), thus Kruskal’s runs in O(E log V^2) = O(E log V)

Let’s have a quick look at KruskalDemo.java
If given an MST problem, I will...

1. Use/code Kruskal’s algorithm
2. Use/code Prim’s algorithm
3. No preference...
Summary

Re-introducing the MST problem (covered in CS1231)

Discussing the implementation of Prim’s algorithm
– Revisiting the PriorityQueue ADT

Discussing the implementation of Kruskal’s algorithm
– Revisiting the EdgeList and showing technique to sort edges
– Revisiting the Union-Find Disjoint Sets DS

You may learn MST/Prim’s/Kruskal’s again in CS3230
PS4 should now be doable 😊

It will not due until Sat, 17 Oct 2015, 8am, because I want to give you time off from CS2010 in Week 07

But it is always better to attempt it earlier than later

*Reminder to self (re-discuss TopoSort if still have time)*