NOTICE!

- Homework 4 theory is due tomorrow
- Lab 3 review
  - Make sure your code compiles.
  - Will provide a dummy test harness.
  - You should try compressing and decompressing txt files.
- Please do not make any assumptions about the lab specs
  - Do not throw any exceptions/errors
  - Your code must handle exceptions in a graceful manner
TODAY

- BW, MTF, Huffman, LZW
  - Seen most of it.
- DFA construction
- Graphs
  - Representations
  - Traversals
  - MST
DFA

- “Aho-corasick uses an NFA actually” oops
- Constructing DFA. Tips and trick
  - If you need to accept a string of length N, you would need N+1 states minimally
  - Constructing larger DFA’s from smaller DFA.
  - Try some small test cases.
GRAPHS

- Very very important in CS
- Generalized notation of trees and linked list
  - Give raise to the Pointer Model
- Properties
  - Direct/ Undirected
  - Weighted/ Unweighted
  - Simple/ Multi
GRAPHS

- **Representation**
  - Edge List
  - Adjacency Matrix
  - Which is more efficient?

- **Traversals**
  - DFS
  - BFS
PROBLEM: LAYING TELEPHONE WIRE
WIRING: NAÏVE APPROACH

Expensive!
WIRING: BETTER APPROACH

Minimize the total length of wire connecting the customers
**Minimum Spanning Tree (MST)**

- A subgraph of a weighted undirected graph \( G=(V,E) \)
- **Tree**
  - acyclic (no cycles)
- **Spanning**
  - the tree covers all the vertices \( V \)
  - \( |V| - 1 \) edges (Why?)
- **Minimum**
  - total cost associated with tree edges is the minimum among all possible spanning trees
APPLICATIONS OF MST

- Any time you want to visit all vertices in a graph at minimum cost.
- Wire routing on printed circuit boards
- Sewer pipe layout
- Road planning
- Provides a heuristic for the traveling salesman problems. The optimum traveling salesman tour is at most twice the length of the minimum spanning tree. (why??)
Recap
**RECAP: WHAT IS A TREE?**

**Directed**
A tree is a *connected* directed graph in which all vertices have *indegree* 1, except the *root* node which has *indegree* 0.

**Undirected**
A tree is any *connected* undirected graph that contains *no cycle*.

![Directed and Undirected Trees](image)
**Tree in undirected world**

- Any tree node can be thought of as the root!
How can we generate a MST?
THINK AGAIN: MST DEFINITION

- A subgraph of the given graph
- Tree
  - acyclic (no cycles)
- Spanning
  - the tree covers all the vertices
  - \(|V| - 1\) edges (Why?)
- Minimum
  - total cost associated with tree edges is the minimum among all possible spanning trees
HOW CAN WE GENERATE A MST?
PRIM’S ALGORITHM

- Pick a vertex
  - now we have a minimum spanning tree of the chosen vertex
- Grow the tree until all vertices are included
  - Each step, add the edge (u,v) s.t. the cost of (u,v) is minimum among all edges where u is in the tree and v is not in the tree
**Prim’s Algorithm**

- Let $V = \{1, 2, \ldots, n\}$ and $A$ be the set of vertices that makes the MST and $T$ be the MST
- Initially: $A = \{1\}$ and $T = \emptyset$
- while ($A \neq V$)
  - let $(u,v)$ be the lowest weight edge such that $u \in A$ and $v \in V - A$
  - $T = T \cup \{(u,v)\}$
  - $A = A \cup \{v\}$
PRIM’S ALGORITHM IMPLEMENTATION

Initialization

a. Pick a vertex \( r \) to be the root
b. Set \( D(r) = 0, \ parent(r) = \text{null} \)
c. For all vertices \( v \in V, v \neq r \), set \( D(v) = \infty \)
d. Insert all vertices into priority queue \( Q \), using distances as the keys

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
</tr>
</tbody>
</table>
PRIM’S ALGORITHM

While $Q$ is not empty:

1. Select the next vertex $u$ to add to the tree
   
   $u = P.deleteMin(Q)$

2. Update $D(w)$ for each vertex $w$ in $\text{adj}(u)$ which is not in the tree (i.e., $w \in Q$)
   
   If $\text{weight}(u,w) < D(w)$,
   
   a. $\text{parent}(w) = u$
   
   b. $D(w) = \text{weight}(u,w)$
   
   c. Update the priority queue to reflect new distance for $w$
PRIM’S ALGORITHM

While $Q$ is not empty:

1. Select the next vertex $u$ to add to the tree
   $u = P.deleteMin(Q)$

2. Update $D(w)$ for each vertex $w$ in $\text{adj}(u)$ which is not in the tree (i.e., $w \in Q$)
   
   If $\text{weight}(u,w) < D(w)$,
   
   a. $\text{parent}(w) = u$
   
   b. $D(w) = \text{weight}(u,w)$
   
   c. Update the priority queue to reflect new distance for $w$
PRIM’S ALGORITHM

The MST initially consists of the vertex $e$, and we update the distances and parent for its adjacent vertices.
PRIM’S ALGORITHM

Vertex Parent

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Vertex Parent

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>
PRIM’S ALGORITHM

Vertex Parent

d  c  a
4  5  9

Vertex Parent

e  -
b  e
c  e
d  e
a  b

Vertex Parent

ea  -
b  e
c  d
d  e
a  d
PRIM’S ALGORITHM
PRIM’S ALGORITHM

Vertex Parent

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>

Vertex Parent

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>
**Prim’s Algorithm Invariant**

- At each step, we add the edge \((u,v)\) s.t. the weight of \((u,v)\) is **minimum** among all edges where \(u\) is in the tree and \(v\) is not in the tree.

- Each step maintains a minimum spanning tree of the vertices that have been included thus far.

- When all vertices have been included, we have a MST for the graph!
**Initialization of priority queue** (array): $O(|V|)$

**Update loop**: $|V|$ calls
- Choosing vertex with minimum cost edge: $O(\log |V|)$
- Updating distance values of unconnected vertices: Each edge is considered only **once** during entire execution, for a **total** of $O(|E|)$ updates, each of $O(\log |V|)$.

**Overall time**:

$$O\left( (|E| + |V|) \log |V| \right)$$