Introduction

> Given a connected, undirected graph $G = (V, E)$, a spanning tree is an
> acyclic subset $T \subseteq E$ that connects all vertices in $V$
> $T$ acyclic $\Rightarrow$ a tree
> $T$ connects all vertices $\Rightarrow$ spans $G$
> If $G$ is weighted, then $T$’s weight is $w(T) = \sum_{(u,v) \in T} w(u,v)$
> A minimum weight spanning tree (or minimum spanning tree, or
> MST) is a spanning tree of minimum weight
> Not necessarily unique
> Applications: anything where one needs to connect all nodes with
> minimum cost, e.g. wires on a circuit board or fiber cable in a network

MST Example

Kruskal’s Algorithm

> Greedy algorithm: Make the locally best choice at each step
> Starts by declaring each vertex to be its own tree (so all nodes together
> make a forest)
> Iteratively identify the minimum-weight edge $(u, v)$ that connects two
> distinct trees, and add it to the MST $T$, merging $u$’s tree with $v$’s tree

MST-Kruskal($G, w$)

```plaintext
A = \emptyset
for each vertex $v \in V$ do
    Make-Set($v$)
end
sort edges in $E$ into nondecreasing order by weight $w$
for each edge $(u, v) \in E$, taken in nondecreasing order do
    if Find-Set($u$) $\neq$ Find-Set($v$) then
        $A = A \cup \{(u, v)\}$
        Union($u$, $v$)
    end
end
return $A$
```

More on Kruskal’s Algorithm

> Find-Set($u$) returns a representative element from the set (tree) that
> contains $u$
> Union($u$, $v$) combines $u$’s tree to $v$’s tree
> These functions are based on the disjoint-set data structure
> More on this later
Disjoint-Set Data Structure

Given a universe \( U = \{x_1, \ldots, x_n\} \) of elements (e.g., the vertices in a graph \( G \)), a DSDS maintains a collection \( S = \{S_1, \ldots, S_k\} \) of disjoint sets of elements such that

- Each element \( x_i \) is in exactly one set \( S_j \)
- No set \( S_j \) is empty
- Membership in sets is dynamic (changes as program progresses)
- Each set \( S \in S \) has a representative element \( x \in S \)

Chapter 21

Disjoint-Set Data Structure (2)

- DS implementations support the following functions:
  - \( \text{MAKE-SET}(x) \) takes element \( x \) and creates new set \( \{x\} \); returns pointer to \( x \) as set’s representative
  - \( \text{UNION}(x, y) \) takes \( x \)'s set \( S_x \) and \( y \)'s set \( S_y \), assumed disjoint from \( S_x \), merges them, destroys \( S_x \) and \( S_y \), and returns representative for new set from \( S_x \) \( \cup \) \( S_y \)
  - \( \text{FIND-SET}(x) \) returns a pointer to the representative of the unique set that contains \( x \)
- Section 21.3: can perform \( d \) D-S operations on \( e \) elements in time \( O(d \alpha(e)) \), where \( \alpha(e) = o(\log^* e) = o(\log e) \) is very slowly growing:

<table>
<thead>
<tr>
<th>( e )</th>
<th>( \alpha(e) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Analysis of Kruskal’s Algorithm

- Sorting edges takes time \( O(|E| \log |E|) \)
- Number of disjoint-set operations is \( O(|V| + |E|) \) on \( O(|V'|) \) elements, which can be done in time \( O((|V'| + |E|) \alpha(|V'|)) = O(|E| \alpha(|V'|)) \) since \( |E| \geq |V| - 1 \)
- Since \( \alpha(|V'|) = o(\log |V'|) = O(\log |E|) \), we get total time of \( O(|E| \log |E|) = O(|E| \log |V|) \) since \( \log |E| = O(\log |V|) \)
Prim's Algorithm

- Greedy algorithm, like Kruskal’s
- In contrast to Kruskal’s, Prim’s algorithm maintains a single tree rather than a forest
- Starts with an arbitrary tree root \( r \)
- Repeatedly finds a minimum-weight edge that is incident to a node not yet in tree

\[
\text{MST-Prim}(G, w, r)
\]

1. \( A = \emptyset \)
2. for each vertex \( v \in V \) do
   3. \( \text{key}[v] = \infty \)
   4. \( \pi(v) = \text{nil} \)
5. end
6. \( \text{key}[r] = 0 \)
7. \( Q = V \)
8. while \( Q \neq \emptyset \) do
   9. \( u = \text{Extract-Min}(Q) \)
   10. for each \( v \in \text{Adj}[u] \) do
       11. if \( v \in Q \) and \( w(u, v) < \text{key}[v] \) then
           12. \( \pi(v) = u \)
           13. \( \text{key}[v] = w(u, v) \)
   14. end
15. end

More on Prim’s Algorithm

- \( \text{key}[v] \) is the weight of the minimum weight edge from \( v \) to any node already in MST
- \text{Extract-Min} uses a minimum heap (minimum priority queue) data structure
  - Binary tree where the key at each node is \( \leq \) keys of its children
  - Thus minimum value always at top
  - Any subtree is also a heap
  - Height of tree is \( \mathcal{O}(\log n) \)
  - Can build heap on \( n \) elements in \( \mathcal{O}(n) \) time
  - After returning the minimum, can filter new minimum to top in time \( \mathcal{O}(\log n) \)
  - Based on Chapter 6

Example (1)

Example (2)

Analysis of Prim’s Algorithm

- \textbf{Invariant}: Prior to each iteration of the while loop:
  1. Nodes already in MST are exactly those in \( V \setminus Q \)
  2. For all vertices \( v \in Q \), if \( \pi[v] \neq \text{nil} \), then \( \text{key}[v] < \infty \) and \( \text{key}[v] \) is the weight of the lightest edge that connects \( v \) to a node already in the tree
- \textbf{Time complexity}:
  - Building heap takes time \( \mathcal{O}(V) \)
  - \( |V| \) calls to \text{Extract-Min}, each taking time \( \mathcal{O}(\log |V|) \)
  - For loop iterates \( \mathcal{O}(|E|) \) times
  - In for loop, need constant time to check for queue membership and \( \mathcal{O}(\log |V|) \) time for decreasing \( v \)'s key and updating heap
  - Yields total time of \( \mathcal{O}(|V| \log |V| + |E| \log |V|) = \mathcal{O}(|E| \log |V|) \)
  - Can decrease total time to \( \mathcal{O}(|E| + |V| \log |V|) \) using Fibonacci heaps