Spanning Trees (Ch. 23)

**Definition:** Given an undirected, unweighted graph \( G = (V, E) \), a **spanning tree** of \( G \) is any subgraph of \( G \) that is a tree.

![Diagram of a graph with labels a, b, c, d, e, f, g, h and edges with weights 1, 2, 3, 5, 6, 7, 8]

Weighting edges

Assign a weight (a numerical value) to each edge of the graph.

Examples:
1. a road network, the weights could represent the length of each road;
2. a network of connecting flights, weights could represent flight time;
3. a computer network, the weights could represent the bandwidth of each bus and link.
4. length of wire needed to connect gates in a circuit.

Minimum Spanning Trees

**Definition:** Given an undirected graph \( G = (V, E) \) with weights on the edges, a **minimum spanning tree** of \( G \) is a subgraph \( T \subseteq E \) such that:
- \( T \) is a spanning tree of \( G \)
- \( T \) has no cycles (i.e., is a tree), and
- \( T \) has a sum of edge weights that is minimum over all possible spanning trees of \( G \).

![Diagram of a graph with labeled edges and weights 1, 6, 3, 6, 2, 5, 4, 7, 8]

MST Property

**MST property:** Let \( G = (V, E) \) and let \( T \) be any spanning tree of \( G \). Suppose that for every edge \((u, v)\) of \( G \) that is not in \( T \), if \((u, v)\) is added to \( T \) it creates a cycle such that \((u, v)\) is a maximum weight edge on that cycle. Then \( T \) has the MST property.

If there are 2 spanning trees \( T_1 \) and \( T_2 \) on \( G \) that both have the MST property, then \( T_1 \) and \( T_2 \) have same total weight.

Minimum Spanning Trees

We will look at two “greedy algorithms” to find an MST of a weighted graph: **Kruskal’s** and **Prim’s** algorithms.

A greedy algorithm makes choices in sequence such that each choice is best according to some limited “short-term” criterion that is not too expensive to evaluate (no look-ahead is involved).

![Diagram of a graph with labeled edges and weights 1, 6, 3, 5, 2, 4, 8]

MST Uses

Finding an MST has proven to be a useful technique in finding maximum bandwidth channels, circuit design, and in finding neural pathways in the brain.

Finding maximum bandwidth channels in networks requires a simple change to Kruskal’s algorithm:

Kruskal's MST Algorithm

Idea:
- use a greedy strategy
- consider edges in increasing order of weight (sort edges)
- add edge to spanning forest $F$ if adding the edge doesn't create a cycle.

Algorithm MST-Kruskal $(G)$

1. $R = E$ // $R$ is initially set of all edges
2. $F = \emptyset$ // $F$ is set of edges in a spanning tree of a sub-graph of $G$
3. sort all edges of $R$ in increasing order of weight
4. while ($R$ is not empty)
5. remove the lightest-weight edge, $(v,w)$, from $R$
6. if $(v,w)$ does not make a cycle in $F$
7. add $(v,w)$ to $F$
8. return $F$

Kruskal's MST Algorithm

Complexity:
- line 1- Sorting edges = ?? time
- lines 2-5 - Keep edges in a structure with $O(1)$ time
- line 4 - Checking to see if edge creates a cycle = ?? time

Disjoint Sets (Ch. 21)

A disjoint-set data structure
- maintains a collection of disjoint subsets $C = s_1, s_2, \ldots, s_m$, where each $s_i$ is identified by a representative element (set id).

Operations on $C$:
- Make-Set($x$): creates singleton set {$x$}
- Union($x,y$): $x$ and $y$ are id’s of their resp. sets, $s_x$ and $s_y$; union operation replaces sets $s_x$ and $s_y$ with a set that is $s_x \cup s_y$ and returns the id of the new set.
- Find-Set($x$): returns the id of the set containing $x$.

Data Structures for Disjoint Sets

Comment 1: The Make-Set operation is only used during the initialization of a particular algorithm.

Comment 2: We assume there is an array of pointers to each $x \in U$ (so we never have to search for a particular element, just for the id of the set $x$ is in).

Thus the problems we’re trying to solve are how to join the sets (Union) and how to find the id of the set containing a particular element (Find-Set) efficiently.

Data Structures for Disjoint Sets

Applications include network algorithms such as finding the connected components of a graph.

PROBLEM IS HOW TO KEEP ELEMENTS IN A SET IN SUCH A WAY THAT IT IS A FAST (SUB-LINEAR TIME) OPERATION TO DETERMINE WHETHER A NEW EDGE WILL CREATE A CYCLE.

We determine membership using representative node names for each connected component.

Rooted Tree Representation of Sets

Idea: Organize elements of each set as a tree with id = element at the root, and a pointer from every child to its parent. Also assume we have an array of pointers to each element in the tree.

Make-Set($x$): (Initial) $O(1)$ time
- start at $x$ (using pointer provided to find $x$) and follow pointers up to the root.
- return id of root

Find-Set($x$):
- make $x$ a child of (roots of trees).
- return id of $y$ and return $y$ running time is $O(n)$

Comment 2: We assume there is an array of pointers to each element in the tree.
Weighted Union Implementation for Trees

Idea: Add `rank` field to each node `x` holding the number of nodes in subtree rooted at `x` (only care about weight field of roots, even though other nodes maintain rank value too). When doing a Union, make the smaller tree (with lower rank at the root) a subtree of the larger tree (with greater rank at the root).

- `Make-Set(x)`: $O(1)$
- `Find-Set(x)`: See next slide, $O(n)$ w.c.
- `Union(x, y)`:  
  - $x$ and $y$ are ids (roots of trees).  
  - make node ($x$ or $y$) with smaller rank the child of the other  
  - $O(1)$ time

```
Weighted Union
Theorem: Any $k$-node tree created by $k-1$ weighted Unions has height $O(\lg k)$ (assume we start with Make-Set on $k$ singleton sets). We want to show that trees stay "short".
Proof: By induction on $k$, the number of nodes in forest.
Basis: $k = 1$, height = 0 = $\lg 1$.\text{<true>}

Inductive Hypothesis: Assume true for all $i < k$.
Inductive Step: Show true for $k$. Suppose the last operation performed was union($x,y$) and that if $m = w(x)$ and $w(y) \leq w(x)$, that $m \leq k/2$.
```

Path Compression Implementation

Idea: extend the idea of weighted union (i.e., unions still weighted), but on a `Find-Set(x)` operation, make every node on the path from $x$ to the root (the node with the set id) a child of the root.

```
Path Compression Analysis
The running time for $m$ (find or union) disjoint-set operations on $n$ elements is $O(mlg^*n)$.

The full proof is given in our textbook.
```

Kruskal’s MST Algorithm

Idea: Make each node a singleton set.

Sort edges, then add the minimum-weight edge $(u,v)$ to the MST if $u$ and $v$ are not already in same sub-graph.

Use weighted union with path compression during find-set operations to determine when nodes are in same sub-graph.

```
Kruskal’s MST Algorithm
Running Time
* initialization (lines 1-3) – $O(1) + O(V) + O(E \lg E) = O(V + E \lg E)$
* $E$ iterations of for-loop (line 4)  
  - $2E$ finds – $O(E \lg^* E)$ time  
  - $O(V)$ unions = $O(V)$ time (at most $V - 1$ unions)
* total: $O(V + E \lg E) = O(E \lg V)$ time  
  - (note $\lg E = O(\lg V)$ since $E = O(V^2)$, so $\lg E = 2 \lg V$).
```

```
MS-T (G) = (V, E)  
1. $T = \emptyset$  
2. for each $v \in V$
    - make-set($v$)  
3. sort edges in $E$ by increasing (non-decreasing) weight
4. for each $(u,v) \in E$
    - if find-set($u$) $\neq$ find-set($v$)
        - $T = T \cup (\{u,v\})$  
        - *add edge to MST**
5. return $T$
```
Correctness of Kruskal’s Algorithm

Theorem: Kruskal’s algorithm produces an MST on $G = (V, E)$.

Proof: Clearly, the algorithm produces a spanning tree. We need to argue that it is an MST.

Suppose, in contradiction, the algorithm does not produce an MST.

Suppose $T$ is the current tree (a greedy strategy).

- at each iteration, chooses the lowest weight edge that goes out from the current tree (a greedy strategy).

Let $e$ be the value such that $e \in E$ but $e \notin T$.

Then $PQ$ does not contain an edge $(u, v)$ with $v \in T$.

Idea: Associate with each node $v$ two fields:

- $\pi(v)$: if $v$ is not in $T$, then holds the name of the node $\pi(v)$ in $T$ such that $\delta(u, \pi(v))$ is $v$’s best edge to node in $T$.

- $\pi(v)$ and $\delta(u, \pi(v))$ are added to the MST by Kruskal’s algorithm.

Correctness of Kruskal’s Algorithm (cont.)

- let $e^*$ be the edge in $T \cup (e_i)$ that forms a cycle when $e_i$ is added to $T$ that is not in $e_1, e_2, …, e_i$, $e_i$ is not a subset of any MST.

Then $\delta(e_i) < \delta(e^*)$, otherwise the algorithm would have picked $e^*$ next in sorted order when it picked $e_i$ (by assumption that $T$, with $e^*$, is not an MST because the algorithm does not find an MST).

Claim: $T' = T - \{e^*\} \cup \{e_i\}$ is a MST

- $T'$ is a spanning tree since it contains all nodes and has no cycles.

- $\delta(T') < \delta(T)$, so $T$ is not a MST

This contradiction means our original assumption must be wrong and therefore the algorithm always finds an MST.$\blacksquare$

Prim’s MinST Algorithm

Algorithm starts by selecting an arbitrary starting vertex, and then “branching out” from the part of the tree constructed so far by choosing a new vertex and edge at each iteration.

Idea:
- always maintains one connected subgraph (different from Kruskal’s)
- at each iteration, chooses the lowest weight edge that goes out from the current tree (a greedy strategy).

Prim’s MST Algorithm

Idea: Use a min priority queue $PQ$ that uses the set field as a key.
Associate with each node $v$ two fields:

- $\pi(v)$: if $v$ isn’t in $T$, then holds the name of the node $\pi(v)$ in $T$ such that $\delta(u, \pi(v))$ is $v$’s best edge to node in $T$.

- $\pi(v)$ and $\delta(u, \pi(v))$ are added to the MST by Prim’s algorithm.

As min set edges are discovered they are added to $T$. 

MST-Kruskal $(G)$
1. $T = \emptyset$
2. for each $v \in V$
3. sort edges in $E$ by increasing weight
4. for each $(u, v) \in E$ sorted $E$
   if find $(u) \neq find(v)$ // doesn’t create a cycle **/
   $T = T \cup ((u, v))$ // ** add edge to $T$ **/
   union(find(u), find(v))
5. return $T$

Correctness of Prim’s Algorithm

Proof: Prim’s algorithm produces a spanning tree $T$. We need to argue that it is an MST.

Suppose, in contradiction, that the algorithm does not produce an MST.

Then $\delta(T)$, i.e., the minimum weight of an edge incident on some vertex in $V$ that is not in $T$.

Let $e^*$ be the edge of minimum weight that is not in $T$.

Claim: $T' = T \cup \{e^*\}$ is a MST

- $T'$ is a spanning tree since it contains all nodes and has no cycles.

- $\delta(T') = \delta(T) + \delta(e^*)$, so $T$ is not a MST

This contradiction means our original assumption must be wrong and therefore the algorithm always finds an MST.$\blacksquare$
Prim's MST Algorithm

Start at node a: PQ contains all nodes:

iteration 1: PQ = PQ – {a}

iteration 2: PQ = PQ – {b}

iteration 3: PQ = PQ – {e}

iteration 4: PQ = PQ – {d}

iteration 5: PQ = PQ – {g}

iteration 6: PQ = PQ – {c, d, g, h} due to (c,e), (d,e), (g,e), and (c,h)

iteration 7: PQ = PQ – {h} = ∅

T = $(a,b), (a,e), (e,g), (c,e), (f,g), (g,h))$ (no change to wt field at any node)

DONE

Running Time of Prim's MST Algorithm

• Assume PQ is implemented with a binary min-heap

• How can we tell if $v \notin PQ$ without searching heap?

Keep an array of booleans indexed by the nodes indicating if node is in heap

MST-Prim (G, r)
1. insert each $v \in V$ into PQ with $wt(v) = \infty$ or $\infty$ if $v=r$
2. cost = 0 / cost of MST
3. while PQ ≠ ∅
4. if $v \in PQ$ and $v \neq r$
5. add edge $(u, v)$ to T
6. for each node $u$ of $T$
7. if $v \notin PQ$ and $wt(u, v) < \infty$
8. $v = u$
9. $v \notin w(u, v)$

Running Time of Prim's MST Algorithm

• Assume PQ is implemented with a binary min-heap

• How can we tell if $v \notin PQ$ without searching heap?

Keep an array of booleans indexed by the nodes indicating if node is in heap

MST-Prim (G, r)
1. insert each $v \in V$ into PQ with $wt(v) = \infty$ or $\infty$ if $v=r$
2. cost = 0 / cost of MST
3. while PQ ≠ ∅
4. if $v \in PQ$ extract-min()
5. add edge $(u, v)$ to T
6. for each node $u$ of $T$
7. if $v \notin PQ$ and $wt(u, v) < \infty$
8. $v = u$
9. $v \notin w(u, v)$
Running Time of Prim’s MST Algorithm

Running time:
* Initialize PQ: $O(V)$ time
* while loop... in each of $V$ iterations of while loop:
  - extract min = $O(\lg V)$ time
  - update $T = O(1)$ time
  $\Rightarrow$ $O(V \lg V)$ total over all iterations (combined):
  - check neighbors of $u$ (line 6-9): $O(E)$ iterations
  - condition test and update $\pi = O(1)$ time
  - decreasing $v$’s $\text{wt} = O(\lg V)$ time
$\Rightarrow$ $O(E \lg V)$

So, the grand total is:
$O(V \lg V + E \lg V) = O(E \lg V)$ (asymptotically, the same as Kruskal’s)

Correctness of Prim’s Algorithm

Let $T_i$ be the tree after the $i$th iteration of the while loop

Lemma: For all $i$, $T_i$ is a subtree of some MST of $G$.

Proof: by induction on $i$, the number of iterations

Basis: when $i = 0$, $T_0 = \emptyset$, ok - because empty is trivial MST subtree

IHOP: Assume $T_i$ is a subtree of some MST $M$

Induction Step: Show that $T_{i+1}$ is a subtree of some MST $M$

Maximum Spanning Trees

Definition: Given an undirected graph $G = (V, E)$ with weights on the edges, a maximum spanning tree of $G$ is a subgraph $T \subseteq E$ such that:

- connects all nodes in $V$
- has no cycles (i.e., is a tree), and
- has a sum of edge weights that is maximum over all possible spanning trees of $G$. 

-\[ \begin{align*} 
\text{a} & \quad 1 \quad \text{b} \quad 6 \\
\text{c} & \quad \text{d} \quad \text{e} \quad \text{f} \quad \text{g} \quad \text{h} \\
\text{a} & \quad 1 \quad \text{b} \quad 6 \\
\text{c} & \quad \text{d} \quad \text{e} \quad \text{f} \quad \text{g} \quad \text{h} \\
\end{align*} \]