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

A graph structure can be extended by assigning a weight to each edge of the graph. Graphs with weights, or weighted graphs, are used to represent structures in which pairwise connections have some numerical values.

For example if a graph represents a road network, the weights could represent the length of each road; if a graph represents a computer network, the weights could represent the bandwidth of each connection.

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:
- $T$ has no cycles,
- $T$ connects all nodes in $V$, and
- $T$ has a sum of edge weights that is minimum over all possible 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 individual choice is best according to some limited "short-term" criterion that is not too expensive to evaluate.

Minimum Spanning Trees

Question: (True or False) Adding an edge to a spanning tree of a graph $G$ always creates a cycle.

Justify your answer.

Properties of MSTs

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.
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 it 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 subgraph 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 $F$

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).

- set of underlying elements make up the sets $U = \{1, 2, \ldots, n\}$

Operations on $C$:
- Make-Set ($x$): $x \not\in U$, creates singleton set $\{x\}$
- Union ($x,y$): $x, y \in U$ and are id’s of their resp. sets, $s_x$ and $s_y$; replaces sets $s_x$ and $s_y$ with a set that is $s_x + s_y$, and returns the id of the new set.
- Find-Set ($x$): $x \in U$, 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 which id the set has). 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.

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 (assuming we have an array of pointers to each element in the tree).

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

Weighted Union Implementation for Trees

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

- Make-Set ($x$): $O(1)$
- Find-Set ($x$):
  - ?? See next slide
- Union ($x, y$):
  - $x$ and $y$ are ids (roots of trees).
  - make $x$ a child of $y$ and return $y$
  - $O(1)$ time

Weighted Union

Theorem: Any $k$-node tree created by $k-1$ weighted Unions has height $O(\log k)$ (initially we start with Make-Set on $k$ singleton sets).

Proof: By induction on $k$, the number of nodes.

- Basis: $k = 1$, height = 0, $h_0 = 0$.
- Inductive Hypothesis: Inductive step: Suppose the set operation performed was Union($x,y$) and that $h_0 = \min(x), \max(x)$, and $m = \lfloor k/2 \rfloor$.

Show $h = \max(h_0, h_1) = k$. The IHLP must hold for trees $x$ and $y$.

- $h_0 + h_1 = h_x + h_y = h_x + \lfloor \log_2 m \rfloor + 1 = \lfloor \log_2 m \rfloor + 1 = \lfloor \log_2 k \rfloor$
- $h_0 + h_1 = h_x + \lfloor \log_2 m \rfloor = h_x + \log_2 k$ (for positive $m$)
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.

Find-Set(x) still has worst-case time of $O(\log n)$, but subsequent Find-Sets for nodes that used to be ancestors of x (or subsequent finds for x itself) will now be very fast – $O(1)$.

Path Compression Analysis

The running time for $m$ disjoint-set operations on $n$ elements is $O(m\alpha(n))$.

The full proof is given in our textbook.

You knew that $\alpha$ would turn up somewhere, right?

Kruskal’s MST Algorithm

Idea:
- Sort edges, then add the minimum-weight edge $(u,v)$ to the MST if $u$ and $v$ are not in same subgraph. Uses union-find operations to determine when nodes are in same subgraph.

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

Running Time
- Initialization (line 1-3) – $O(1) + O(|V|) + O(|E| \log |E|) = O(|V| + |E| \log |E|)$
- $|E|$ iterations of for-loop
  - $O(|V|)$ makes = $O(|V|)$ time
  - $O(|E|)$ unions = $O(|V|)$ time (at most $|V| - 1$ unions)
- $\log |E| = \log |V|$ since $|E| = O(|V|^2)$, so $\log |E| = O(2 \log |V|)$

Correctness of Kruskal’s Algorithm

**Theorem**: Kruskal’s algorithm produces an MST.

**Proof**: Assume the edge weights are distinct. Clearly, the algorithm produces a spanning tree. We need to argue that the spanning tree is an MST.

Suppose, in contradiction, the algorithm does not produce an MST. Suppose that the algorithm adds edges to the tree in order $e_1, e_2, \ldots, e_n$.

Let $T$ be the tree such that $e_1, e_2, \ldots, e_{i-1}$ is a subset of some MST $T_i$, but $e_i, e_{i-1}, \ldots, e_1$ is not a subset of any MST.

Consider $T \cup (e_i)$
- $T \cup (e_i)$ must have a cycle $c$ involving $e_i$, $T \cup (e_i)$ must have a cycle $c$ involving $e_i$.
- If $c$ is not the cycle $e_i$, $e_{i-1}, \ldots, e_1$ (since the algorithm didn’t pick an edge that creates a cycle and it failed $e_i$).
Correctness of Kruskal’s Algorithm (cont.)

- Let e’ be the edge with minimum weight on c that is not in e₁, e₂, …, eₘ. Then wt(e’) < wt(e₁), otherwise the algorithm would have picked e’ next in sorted order instead of e₁.

Claim: T = T – (e’), U {eₘ} is an MST
- T is a spanning tree since it contains all nodes and has no cycles.
- wt(T) = wt(T) – wt(e’) + wt(eₘ) = wt(U) < wt(T), so T is not an MST (contradiction)

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

Prim’s MST Algorithm

Data: use a priority queue Q: associate with each node a two fields
- key[v]: to prioritize node u in T, the key[v] is the weight of the heaviest edge in the current tree.
- wt(u,v): for each neighbor v of u

T is the spanning tree that is being incrementally constructed in Prim’s Algorithm. When a node is extracted from the queue in line 4, it is added to T and never again considered.

Prim’s MST 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.

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

Start at node r:

Iteration 1: Q = {c}

Iteration 2: Q = Q – {c}

Iteration 3: Q = Q – {b}

Iteration 4: Q = Q – {d}

Iteration 5: Q = Q – {g}

Iteration 6: Q = Q – {f}

Iteration 7: Q = Q – {e}
Running Time of Prim’s MST Algorithm

- Assume $Q$ is implemented with a binary min-heap (heapsort)
- How can we tell if $v \in Q$ without searching heap? (line 6)
  - keep an array for nodes with broken flag indicating if still in heap
  - initialize $Q$: $O(|V|)$ time

Running time:
- Initialize $Q$: $O(|V|)$ time
- Decrease $v$’s key (line 2) $= O(|E|)$ time
- While loop...
- Decrease $u$’s key (line 15) $= O(|E|)$ time
- Extract min from $Q$ (line 6) $= O(|E|)$ time

So, the total time is:

$O(|V|) + |E| \log |V| = O(|E| \log |V|)$

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 subset of some MST of $G$.

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

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

Induction: Assume $T_i$ is a subset of some MST $M_i$.

Induction Step: Show that $T_{i+1}$ is a subset of some MST (possibly different from $M_i$).

Let $(w, v)$ be the edge added in iteration $i + 1$. Since $(w, v)$ is selected, it must be the minimum-weight edge outgoing from $T_i$. 

Correctness of Prim’s Algorithm

- Case 1: $(u, v)$ is an edge of $M$.
  - Then clearly $T_{i+1}$ is a subset of $M$.
- Case 2: $(u, v)$ is not an edge of $M$.
  - We know there is a path $p$ in $M$ from $u$ to $v$ (because $M$ is a ST).
  - Let $(u, y)$ be the first edge in $p$ with $x \in T_i$. We know this edge exists because the algorithm will not add edge $(u, v)$ to a cycle.
  - $M' = M - \{u, v\} \cup \{(u, y)\}$ is another spanning tree.
  - Now we note that
    $w(M') = w(M) - w(u, v) = w(u, y) + w(M)$
    since $(u, v)$ is the minimum-weight outgoing edge from $T_i$.
  - Therefore, $M'$ is also a MST of $G$ and $T_{i+1}$ is a subset of $M'$. 

HW 8

Due Tuesday, November 29th

1. Explain why adding an edge to a spanning tree of a graph $G$ always creates a cycle.
2. Argue that if all edge weights of a graph are positive, then any subset of edges that connects all vertices and has minimum total weight must be a tree. Give an example to show that the same conclusion does not follow if we allow some weights to be nonpositive.
3. Indicate whether the following statements are true or false. If your answer is false, provide a counter-example:
   - a) If $e$ is a minimum-weight edge in a connected weighted graph, it must be among the edges of at least one MST of the graph.
   - b) If $e$ is a minimum-weight edge in a connected weighted graph, it must be among the edges of each MST of the graph.