CSC 505, Fall 2000: Week 7

In which our discussion of Minimum Spanning Tree Algorithms and their data Structures is brought to a happy Conclusion.
Prim’s algorithm without data structures

Attributed to Jarnik (1930), Dijkstra (1957), and Prim (1957)

Idea: Maintain one component (a single tree) throughout and repeatedly choose the lowest cost edge of the cutset that connects that tree to the remaining vertices.

function $\text{Simple-Prim}(G, w)$ is

▷ returns a minimum spanning tree of $G$ as a list of edges;
$T \leftarrow \emptyset$; $s \leftarrow$ an arbitrary vertex of $G$; $S \leftarrow \{s\}$

while $|T| < n - 1$ and there are edges between $S$ and $V - S$ do

$e = uv \leftarrow$ the lowest cost edge with $u \in S$, $v \in V - S$

$T \leftarrow T + e$; $S \leftarrow S + v$

if $|T| = n - 1$ then return $T$

else error: $G$ is not a connected graph

end $\text{Simple-Prim}$
A simple observation...

Each time we add an edge to the tree, the edge connects the tree to the “closest” outside vertex.

How does this help? We don’t need to look at all the edges if we maintain, for each vertex, its “distance” from the tree and the edge that achieves that distance.
...and an update strategy to match

**Initialize:** \( d[v] \leftarrow \infty \) for all \( v \in V \)

**Start Vertex:** \( d[s] \leftarrow 0 \)

for \( v \in V - S \) with \( sv \in E \) do

\[ d[v] \leftarrow w(s, v); \pi[v] \leftarrow s \]

**Add Tree Edge:** Choose \( v \) with minimum \( d[v] \) for all \( v \in V - S \) and add \( e = (\pi[v], v) \)

**Update Step:** When a new vertex \( v \) is added to \( S \),

for \( x \in V - S \) with \( vx \in E \) do

if \( w(v, x) < d[x] \) then

\[ d[x] \leftarrow w(v, x); \pi[x] \leftarrow v \]
Reminder: structures for storing graphs

Adjacency Matrix

\[ \Theta(n) \] to access all edges incident on a node.

Adjacency List

\( O(1) \) per edge to access all edges incident on a node.
Prim’s algorithm with a heap

function PRIM(G, w) is
  ▷ returns a minimum spanning tree of G as a list of edges;
  for all \( v \in V \) do \( d[v] \leftarrow \infty; \pi[v] \leftarrow \emptyset; \text{in}[v] \leftarrow \text{FALSE} \)
  \( T \leftarrow \emptyset; s \leftarrow \) an arbitrary vertex of \( G; d[s] \leftarrow 0 \)
  Initialize a heap \( H \) on \( V - S \) with \( d \) values as keys
  while \( |T| < n - 1 \) and \( d[\text{MIN}(H)] < \infty \) do
    \( v \leftarrow \text{EXTRACT-MIN}(H); \text{in}[v] \leftarrow \text{TRUE}; u \leftarrow \pi[v] \)
    if \( u \neq \emptyset \) then \( T \leftarrow T \cup \{uv\} \)
    for \( x \in V - S \) with \( vx \in E \) do
      if \( w(v, x) < d[x] \) then
        \( d[x] \leftarrow w(v, x); \pi[x] \leftarrow v \)
        \( \text{DECREASE-KEY}(H, x, d[x]) \)
    if \( |T| = n - 1 \) then return \( T \)
  else error: \( G \) is not a connected graph
end PRIM
Summary of MST algorithms

<table>
<thead>
<tr>
<th>algorithms</th>
<th>data structures</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>none</td>
<td>simple</td>
<td>sophisticated</td>
</tr>
<tr>
<td>Kruskal’s</td>
<td>$\Theta(m \lg n + n^2)$</td>
<td>$\Theta(m \lg n)$</td>
<td>$\Theta(m \lg n)$</td>
</tr>
<tr>
<td>Kruskal’s$^a$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(m + n \lg n)$</td>
<td>$\Theta(m \alpha(m, n))$</td>
</tr>
<tr>
<td>using...</td>
<td>linked lists</td>
<td>disjoint-set forests</td>
<td></td>
</tr>
<tr>
<td>Prim’s</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(m \lg n)$</td>
<td>$\Theta(m + n \lg n)$</td>
</tr>
<tr>
<td>using...</td>
<td>binary heaps</td>
<td>Fibonnaci heaps</td>
<td></td>
</tr>
</tbody>
</table>

$^a$if edges are already sorted or can be sorted in linear time

Best bets in practice: Prim’s with no data structures for dense graphs. Prim’s with binary heaps for all others
Disjoint set union (underlying problem in Kruskal’s)

Operations:
- Make-Set(x) -- create a new set with x as its only member.
- Find-Set(x) -- return the name of the set containing x.
- Union(x,y) -- create a new set that is the union of the one containing x and the one containing y (discard the two previously existing sets).

In Kruskal’s algorithm, we do
- $n$ Make-Set operations (one per vertex initially)
- $m$ Find-Set operations (one per edge in the main loop)
- $n - 1$ Union operations (one per spanning tree edge)
A reminder of Kruskal’s algorithm

function Naive-Kruskal(G, w) is
\[
\text{// returns a minimum spanning tree of } G \text{ as a list of edges}
\]
\[
L \leftarrow \text{a list of edges of } G, \text{ sorted by increasing } w
\]
\[
T \leftarrow \emptyset
\]
\[
\text{while } L \neq \emptyset \text{ and } |T| < n - 1 \text{ do}
\]
\[
e = uv \leftarrow \text{the first edge in } L
\]
\[
\text{if } u \text{ and } v \text{ are not in the same component of } T \text{ then}
\]
\[
T \leftarrow T + e; \text{ update components of } T
\]
\[
L \leftarrow L - e
\]
\[
\text{if } |T| = n - 1 \text{ then return } T
\]
\[
\text{else error: } G \text{ is not a connected graph}
\]
end Naive-Kruskal.
Suppose cost of a Union depends on smaller set

<table>
<thead>
<tr>
<th>Union</th>
<th>Assume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union(1, 2)</td>
<td>$n = 2^k$ 1</td>
</tr>
<tr>
<td>Union(3, 4)</td>
<td>1</td>
</tr>
<tr>
<td>Union(n – 1, n)</td>
<td>1</td>
</tr>
<tr>
<td>Union(2, 4)</td>
<td>2</td>
</tr>
<tr>
<td>Union(6, 8)</td>
<td>2</td>
</tr>
<tr>
<td>Union(n – 2, n)</td>
<td>2</td>
</tr>
<tr>
<td>Union(n/4, n/2)</td>
<td>(n/4)</td>
</tr>
<tr>
<td>Union(3n/4, n)</td>
<td>(n/4)</td>
</tr>
<tr>
<td>Union(n/2, n)</td>
<td>(n/2)</td>
</tr>
<tr>
<td>total</td>
<td>((n/2) \log n)</td>
</tr>
</tbody>
</table>
Implementation notes

• Implement each set as a linked list of elements, with pointers to front and rear (so that concatenation is constant time) and store the cardinality.

• Let each element have a pointer to the set it belongs to (typically, elements are integers and can be used as array indices).

• Make-Set and Find-Set are trivial to implement in constant time.

• Union: Go through the list for the smaller of the two sets, changing each pointer to point to the larger set. Then concatenate the lists and update the cardinality. Time is proportional to the size of the smaller set.
Analysis

Naive analysis says $O(n)$ UNION’s, each having at most $n/2$ elements in the smaller set, so $O(n^2)$.

But let’s look at it from the point of view of a single element:

- Each time an element is accessed within a list, it’s involved in a UNION that at least doubles the size of the set it’s in.
- After being accessed $k$ times an element is therefore in a set of size $\geq 2^k$.
- Since no set can have more than $n$ elements, $2^k \leq n$ or the number of accesses of each individual element is no more than $\lceil \lg n \rceil$.

Since there are $n$ elements, the total time spent in accessing them during UNION’s is $O(n \lg n)$, so the overall time for any sequence of $m$ operations on $n$ elements is $O(m + n \lg n)$.
Another approach: disjoint set forest

Each set is a tree; each element points to its parent in the tree. The root of the tree points to itself (or has a null pointer).

Make-Set(x) sets x’s pointer to itself (or S[x] = x if array is used). Find-Set(x) traces path from x to the root of its tree and returns the root (the name of each set is the name of its root). Union(x,y) does Link(Find-Set(x), Find-Set(y)), where Link(r,s) [Pre: r and s are roots] makes r’s pointer point to s
A very bad situation

\[ \text{Union}(1, 2) \]
\[ \text{Union}(1, 3) \]
\[ \ldots \]
\[ \text{Union}(1, n-1) \]
\[ \text{Union}(1, n) \]

\( \Omega(n^2) \) for the Union’s and as bad as \( \Omega(mn) \) total.

Need some discretion when linking (link small to large).
Union by rank

Each node (element) has a rank, initially 0.
- When two nodes of unequal rank are linked, the one of smaller rank points to the one of larger rank.
- When two nodes of equal rank $r$ are linked, either one can point to the other and the new root has rank $r + 1$.

Lemma. If a node $x$ in a disjoint set forest with union by rank has rank $r$, then (a) the longest path ending at $x$ has no more than $r$ edges, and (b) the (sub)tree rooted at $x$ has at least $2^r$ nodes.

Proof. Easy induction on the number of UNION operations.

Conclusion: Total time for $m$ operations on $n$ elements is $O(m \log n)$ because no FIND-SET takes more than $\log n$. 
### Lower bound on union by rank

Assume $n = 2^k$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost (# of pointers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNION(2, 1)</td>
<td>2</td>
</tr>
<tr>
<td>UNION(4, 3)</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>UNION(n, n - 1)</td>
<td>2</td>
</tr>
<tr>
<td>UNION(4, 2)</td>
<td>4</td>
</tr>
<tr>
<td>UNION(8, 6)</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>UNION(n, n - 2)</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>UNION(n/2, n/4)</td>
<td>2(lg n − 2)</td>
</tr>
<tr>
<td>UNION(n, 3n/4)</td>
<td>2(lg n − 2)</td>
</tr>
<tr>
<td>UNION(n, n/2)</td>
<td>2(lg n − 1)</td>
</tr>
</tbody>
</table>

Total cost of the UNION’s is $\Theta(n)$, but after that we have one element a depth $k = \lg n$, $k$ at depth $k − 1$, $\binom{k}{2}$ at depth $k − 2$ and, in general, $\binom{k}{i}$ at depth $k − i$. 
Path compression

Suppose FIND-SET(x) encounters $x = x_0, x_1, \ldots, x_k$ on the path to the root of the tree ($x_k$ is the root). Then the FIND-SET operation modifies the tree so that $x_0, \ldots, x_k$ all point directly to the root $x_k$ (obviously there is no change for $x_{k-1}$ and $x_k$).

Even without union by rank

$$\text{UNION}(1, 2)$$
$$\text{UNION}(2, 3)$$
$$\ldots$$
$$\text{UNION}(n - 2, n - 1)$$
$$\text{UNION}(n - 1, n)$$
$$\text{FIND-SET}(1)$$

Bad situations do not recur
Time bounds for various combinations

Recall: \( n \) is number of elements, \( m \) is number of operations. For the MST problem this happens to coincide with \( n \) vertices and \( m \) edges.

**Worst-case time bounds**

- Array with direct pointer \( \Theta(m + n^2) \)
- Array plus linked list \( \Theta(m + n \lg n) \)
- Forest, no heuristics \( \Theta(mn) \)
- Forest, union by rank only \( \Theta(m \lg n) \)
- Forest, path compression only \( \Theta(m \frac{\lg n}{\lg(m/n)}) \)
- Forest, union by rank and path compression \( \Theta(m \alpha(m, n)) \)