15. A. CONCEPTS

Given a weighted, connected, undirected graph, find a minimum (total) weight free tree connecting the vertices. (AKA bottleneck shortest path tree)

Cut Property: Suppose S and T partition V such that

1. \( S \cap T = \emptyset \)
2. \( S \cup T = V \)
3. \(|S| > 0 \) and \(|T| > 0\)

then there is some MST that includes a minimum weight edge \( \{s, t\} \) with \( s \in S \) and \( t \in T \).

Proof:

Suppose there is a partition with a minimum weight edge \( \{s, t\} \).

A spanning tree without \( \{s, t\} \) must still have a path between \( s \) and \( t \).

Since \( s \in S \) and \( t \in T \), there must be at least one edge \( \{x, y\} \) on this path with \( x \in S \) and \( y \in T \).

By removing \( \{x, y\} \) and including \( \{s, t\} \), a spanning tree whose total weight is no larger is obtained.

Cycle Property: Suppose a given spanning tree does not include the edge \( \{u, v\} \). If the weight of \( \{u, v\} \) is no larger than the weight of an edge \( \{x, y\} \) on the unique spanning tree path between \( u \) and \( v \), then replacing \( \{x, y\} \) with \( \{u, v\} \) yields a spanning tree whose weight does not exceed that of the original spanning tree.
Proof: Including \{u, v\} in the set of chosen edges introduces a cycle, but removing \{x, y\} will remove the cycle to yield a modified tree whose weight is no larger.

The proof suggests a slow approach - iteratively find and remove a maximum weight edge from some remaining cycle:

15.B. **Prim’s Algorithm** – Three versions

Prim’s algorithm applies the cut property by having \(S\) include those vertices connected by a subtree of the eventual MST and \(T\) contains vertices that have not yet been included. A minimum weight edge from \(S\) to \(T\) will be used to move one vertex from \(T\) to \(S\)

1. “Memoryless” – Only saves partial MST and current partition.
   
   (http://ranger.uta.edu/~weems/NOTES2320/primMemoryless.c)

   Place any vertex \(x \in V\) in \(S\).
   
   \(T = V - \{x\}\)
   
   while \(T \neq \emptyset\)
   
   Find the minimum weight edge \(\{s, t\}\) over all \(t \in T\) and all \(s \in S\). (Scan adj. list for each \(t\))
   
   Include \(\{s, t\}\) in MST.
   
   \(T = T - \{t\}\)
   
   \(S = S \cup \{t\}\)

Since no substantial data structures are used, this takes \(\Theta(EV)\) time.

*Which edge does Prim’s algorithm select next?*
2. Maintains T-table that provides the closest vertex in S for each vertex in T. 
(http://ranger.uta.edu/~weems/NOTES2320/primTable.c traverses adjacency lists)

Eliminates scanning all T adjacency lists in every phase, but still scans the adjacency list of the last vertex moved from T to S.

Place any vertex \(x \in V\) in S.

\[ T = V - \{x\} \]

for each \(t \in T\)

- Initialize T-table entry with weight of \(\{t, x\}\) (or \(\infty\) if non-existent) and \(x\) as best-S-neighbor

while \(T \neq \emptyset\)

- Scan T-table entries for the minimum weight edge \(\{t, \text{best-S-neighbor}[t]\}\)

- over all \(t \in T\) and all \(s \in S\).

- Include edge \(\{t, \text{best-S-neighbor}[t]\}\) in MST.

\[ T = T - \{t\} \]

\[ S = S \cup \{t\} \]

for each vertex \(x\) in adjacency list of \(t\)

- if \(x \in T\) and weight of \(\{x, t\}\) < T-weight[x]

  - T-weight[x] = weight of \(\{x, t\}\)

  - best-S-neighbor[x] = t

What are the T-table contents before and after the next MST vertex is selected?

![Graph](image)

Analysis:

- Initializing the T-table takes \(\Theta(V)\).
- Scans of T-table entries contribute \(\Theta(V^2)\).
- Traversals of adjacency lists contribute \(\Theta(E)\).
- \(\Theta(V^2 + E)\) overall worst-case.
3. Replace T-table by a min-heap.

(http://ranger.uta.edu/~weems/NOTES2320/primHeap.cpp)

The time for updating for best-S-neighbor increases, but the time for selection of the next vertex to move from T to S improves.

Place any vertex \( x \in V \) in S.

\( T = V - \{x\} \)

for each \( t \in T \)

Load T-heap entry with weight (as the priority) of \( \{t, x\} \) (or \( \infty \) if non-existent) and \( x \) as best-S-neighbor

\[ \text{minHeapInit}(T\text{-heap}) \]  // a fixDown at each parent node in heap

while \( T \neq \emptyset \)

Use \( \text{heapExtractMin} /* \text{fixDown} */ \) to obtain T-heap entry with the minimum weight edge over all \( t \in T \) and all \( s \in S \).

Include edge \( \{t, \text{best-S-neighbor}[t]\} \) in MST.

\( T = T - \{t\} \)

\( S = S \cup \{t\} \)

for each vertex \( x \) in adjacency list of \( t \)

if \( x \in T \) and weight of \( \{x, t\} < \text{T-weight}[x] \)

\( \text{T-weight}[x] = \text{weight of} \{x, t\} \)

\( \text{best-S-neighbor}[x] = t \)

\[ \text{minHeapChange}(T\text{-heap}) \]  // fixUp

Analysis:

Initializing the T-heap takes \( \Theta(V) \).

Total cost for \( \text{heapExtractMin} \)s is \( \Theta(V \log V) \).

Traversals of adjacency lists and \( \text{minHeapChange} \)s contribute \( \Theta(E \log V) \).

\( \Theta(E \log V) \) overall worst-case, since \( E > V \).

Which version is the fastest?

<table>
<thead>
<tr>
<th></th>
<th>Theory</th>
<th>Sparse ( (E = O(V)) )</th>
<th>Dense ( (E = \Omega(V^2)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( \Theta(EV) )</td>
<td>( \Theta(V^2) )</td>
<td>( \Theta(V^3) )</td>
</tr>
<tr>
<td>2.</td>
<td>( \Theta(V^2 + E) )</td>
<td>( \Theta(V^2) )</td>
<td>( \Theta(V^2) )</td>
</tr>
<tr>
<td>3.</td>
<td>( \Theta(E \log V) )</td>
<td>( \Theta(V \log V) )</td>
<td>( \Theta(V^2 \log V) )</td>
</tr>
</tbody>
</table>
15.C. **Union-Find Trees to Represent Disjoint Subsets**

**Abstraction:**

Set of \( n \) elements: \( 0 . . n - 1 \)

Initially all elements are in \( n \) different subsets

- **find(i)** - Returns integer ("leader") indicating which subset includes \( i \)
  
  \( i \) and \( j \) are in the same subset \( \iff \) find\((i) = \) find\((j) \)

- **union(i, j)** - Takes the set union of the subsets with leaders \( i \) and \( j \).

  Results of previous finds are invalid after a union.

**Implementation 1:** ([http://ranger.uta.edu/~weems/NOTES2320/uf1.c](http://ranger.uta.edu/~weems/NOTES2320/uf1.c))

**Initialization:**

\[
\text{for (i=0; i<n; i++)}
\]
\[
\quad \text{id[i]=i;}
\]

**find(i):**

\[
\text{return id[i];}
\]

**unionFunc(i, j):**

\[
\text{for (k=0; k<n; k++)}
\]
\[
\quad \text{if (id[k]==i)}
\]
\[
\quad \quad \text{id[k]=j;}
\]

**Implementation 2:** ([http://ranger.uta.edu/~weems/NOTES2320/uf2.c](http://ranger.uta.edu/~weems/NOTES2320/uf2.c))

**find(i):**

\[
\text{while (id[i]!=i)}
\]
\[
\quad \text{i=id[i];}
\]
\[
\quad \text{return i;}
\]

**unionFunc(i, j):**

\[
\text{id[i]=j;}
\]

**Implementation 3:** ([http://ranger.uta.edu/~weems/NOTES2320/uf3.c](http://ranger.uta.edu/~weems/NOTES2320/uf3.c))

**Initialization:**

\[
\text{for (i=0; i<n; i++)}
\]
\[
\quad \{
\quad \quad \text{id[i]=i;}
\quad \quad \text{sz[i]=1;}
\quad\}
\]

```c

```
find(x):
  for (i=x;
       id[i]! = i;
       i=id[i])
  ;
  root=i;
  // path compression – make all nodes on path
  // point directly at the root
  for (i=x;
       id[i]! = i;
       j=id[i],id[i]=root,i=j)
  ;
  return root;

unionFunc(i,j):
  if (sz[i]<sz[j])
  {
    id[i]=j;
    sz[j]+=sz[i];
  }
  else
  {
    id[j]=i;
    sz[i]+=sz[j];
  }

Best-case (shallow tree) and worst-case (deep tree) for a sequence of unions?

15.D. KRUSSLAL’S ALGORITHM – A Simple Method for MSTs Based on Union-Find Trees
(http://ranger.uta.edu/~weems/NOTES2320/kruskal.c)

Sort edges in ascending weight order.

Place each vertex in its own set.

Process each edge \{x, y\} in sorted order:

a=Find(x)
b=Find(y)
if a \ne b
  Union(a,b)
  Include \{x, y\} in MST
Time to sort, $\Theta(E \log V)$, dominates computation