Greedy MST

- The greedy algorithm tries to solve the MST problem by making locally optimal choices:
  1. Sort the edges by weight.
  2. For each edge on sorted list, include that in the tree if it does form a cycle with the edges already taken; Otherwise discard it
- The algorithm can be halted as soon as \( n-1 \) edges have been kept.
- Step 1. takes \( O(m \log m) = O(m \log n) \).
- Today, we will see that Step 2 can be done in \( O(n \log n) \) time, later we will present an linear time implementation from this step.
Set Operation

- In the proof of running time for our MST algorithm we will use the following set operations:
  - **Make-Set**(v): creates a set containing element v, {v}.
  - **Find-Set**(u): returns the set to which v belongs to.
  - **Union**(u,v): creates a set which is the union of the two sets, one containing v and one containing u.

- As an example, we can use a pointer to implement a set system: **Make-Set**(v) will create a single node containing element v. 
  **Find-set**(u) will return the name of the first element in the set that contains u, and finally the **union**(u,v) will concatenate the sets containing u and v.

Example of a set Operations

- Use linked list to show a set
- **Make-Set**(w):
- **Find-Set**(u): (will return w)
- **Union**(u,v):
Running Time of Set Operations

- The Make-Set and Find-Set will run in $O(1)$-time.
- How fast can we compute the union.
- Let us ask a different question. Let $N = \{1, \ldots, n\}$ be a set of $n$ integers, and let $P = \{(u, v) \mid u$ and $v$ in $N\}$ be a subset of pairs from $n \times n$.
- For $u = 1$ to $n$ Make-Set$(u)$;
  For every pair $(u, v)$ in $P$
    If Find-Set$(u)$ $\neq$ Find-Set$(v)$
      Union$(u, v)$
- Question: How many times does the pointer for an element get redirected?

Union Operation

- Each merge of two sets might take linear number of pointer changes.
- We might have up $O(n^2)$ pointer changes.
- Let us keep a number associated with each set in its root, $\text{Rank}(u)$, which tell how many elements a set has.
- When merging two lists, always change the pointers in the list with smaller rank.
Union Operation

- Now each time a pointer changes its corresponding set doubles in the size.
- During the whole process the maximum set can become of size at most \( n \).
- For a specific pointer this happen at most \( \log n \) times, 
  \[ 2^0, 2^1, 2^2, ..., 2^k = m, \text{ which means } k = \log n \]
- Over all \( n \) elements, this will result in an \( O(n \log n) \) number of pointer updates.

Kruskal’s MST Algorithm

- It is directly based on Generic MST.
- At each iteration, it finds a light edge, which is also safe, and adds it to an ever growing set, \( A \), which will eventually become the MST.
- During the course of algorithm, the structure generated by algorithm is a forest.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( A \leftarrow \emptyset )</td>
</tr>
<tr>
<td>2.</td>
<td>for each ( v \in V_G ) do</td>
</tr>
<tr>
<td>3.</td>
<td>Make-Set((v))</td>
</tr>
<tr>
<td>4.</td>
<td>Sort Edges in ( E_G )</td>
</tr>
<tr>
<td>5.</td>
<td>for each ((u, v)) in ( E_G )</td>
</tr>
<tr>
<td>6.</td>
<td>if Find-Set((u)) ( \neq ) Find-Set((v))</td>
</tr>
<tr>
<td>7.</td>
<td>( A \leftarrow A \cup {(u, v)} )</td>
</tr>
<tr>
<td>8.</td>
<td>Union((u, v))</td>
</tr>
<tr>
<td>9.</td>
<td>Return ( A )</td>
</tr>
</tbody>
</table>
Running time of Kruskal’s Algorithm

- Step 1: $O(1)$
- Steps 2,3: $O(n)$
- Step 4: $O(m \log m)$
- Steps 5-8: $O(m \log n)$

1. $A \leftarrow \emptyset$
2. for each $v \in V_G$ do
3. Make - Set$(v)$
4. Sort Edges in $E_G$
5. for each $(u, v) \in E_G$
   (In order of increasing weights)
6. if Find - Set$(u) \neq$ Find - Set$(v)$
7. $A \leftarrow A \cup \{(u, v)\}$
8. Union$(u, v)$
9. Return $A$

Example
**Example**

![Graph Example Image]

**Why Does the MST Work?**

- Cuts in graphs: A cut $(S,V-S)$ of an undirected graph $G=(V,E)$ is a partition of $V$:
  - A **Cross edge** is an edge with one endpoint in $S$ and the other in $V-S$.
  - We say a cut $(S,V-S)$ respects the set $A$ if no edge of $A$ is cross edge.
  - An edge is a **light edge** crossing the cut $(S,V-S)$ if it has the minimum weight among all crossing edges.
Correctness:

- Let $G=(V,E)$ be a connected undirected graph with real-valued weight function $w$ defined on $E$. Let $A$ be a subset of $E$ that is included in some minimum spanning tree for $G$, let $(S,V-S)$ be any cut that respects $A$, and let $(u,v)$ be a light edge crossing $(S,V-S)$. Then edge $(u,v)$ is a safe edge for $A$.

![Diagram of a graph with a minimum spanning tree and a cut](image)

### Prim’s MST Algorithm

- At each step the set $A$ is one connected component.
- Start at an arbitrary vertex $r$ (root of the tree).
- At each step adds a new vertex which is connected to $A$ through a minimum weight edge.
- The growth starts at $r$ and continue till all vertices are covered, each vertex $u$ has a parent $p(u)$, which represents its parent in the tree.
- Also, each vertex $u$ has a $key(u)$, which represents the cost of adding $u$ to $A$ at each point of algorithm.
### Prim’s MST

- To implement the priority queue $Q$, we can use a binary heap.
- The steps 1-5 can be done in $O(n)$-time.
- Step 7 take $O(\log n)$-time.
- Step 11 can be implemented with decrease key which takes $O(\log n)$-time.
- Since there are at most $m=|E|$ elements in all $Adj[]$ list for all elements in $Q$, the the algorithm take $O(n \log n + m \log n) = O(m \log n)$.

### Example

- **MST – Prim($G,W,r$)**
  1. $Q \leftarrow V_g$
  2. For each $u \in Q$
  3.     do $key[u] = \infty$
  4.     $key[r] \leftarrow 0$
  5.     $p(r) = \text{NIL}$
  6.     While $Q \neq \emptyset$ do
  7.         $u \leftarrow \text{Extract-Min}(Q)$
  8.         For each $v \in Adj[u]$ do
  9.             If $v \in Q$ and $w(u,v) < key[v]$
  10.                then $p(v) \leftarrow u$
  11.                $key[v] \leftarrow w(u,v)$
Example

11/6/2003
Dept. of Computer Science
Correctness

- The theorem 24.1 holds here.
  - At each step the cut is \((Q, V-Q)\).
  - Extract-min will return the light edge of the current cut.
  - The \(key[v]=w(u,v)\) will ensure that the cost of adding all new vertices in next iteration is up to date.
  - The algorithm stops after \(Q\) is empty, which happens after \(n\) iterations.