A directed acyclic graph is often called a DAG for short. DAG's arise in many applications where there are precedence or ordering constraints.

In general a precedence constraint graph is a DAG in which vertices are tasks and the edge \((u,v)\) means that task \(u\) must be completed before task \(v\) begins.
Example

Initially we only have a huge graph with dependencies:

When is a Graph a DAG

A directed graph $G$ is acyclic if a DFS of $G$ yields no back edge.
Topological Sort

- If G is a DAG then we can build a topological order of its vertices:
  - perform a DFS on G.
  - as each vertex is finished, insert it in front of a linked list
  - Return the linked list of the vertices.

Minimum Spanning Tree (MST) in a Weighted Graph

- Let $G=(V,E)$ be a graph on $n$ vertices and $m$ edges, and a weight function $w$ on edges in $E$.
- A sub-graph $T$ of $G$ through all vertices which avoids any cycle is a spanning tree.
- The weight of $T$ is define as sum of the weights of all edges in $T$: $w(T) = \sum_{(u,v) \in T} w(u,v)$
**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 $P=\{(u,v)| u \text{ and } v \text{ in } N\}$ 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)!=$\text{Find-Set(v)}$
      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.

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, $2^0, 2^1, 2^2, ..., 2^k = m$, 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.

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

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. \( \text{Make} \cdot \text{Set}(v) \)
4. \( \text{Sort Edges in } E_G \)
5. for each \( (u, v) \in E_G \) do 
   \( \text{(In order of increasing weights)} \)
6. \( \text{if } \text{Find} \cdot \text{Set}(u) \neq \text{Find} \cdot \text{Set}(v) \)
7. \( A \leftarrow A \cup \{(u, v)\} \)
8. \( \text{Union}(u, v) \)
9. Return \( A \)
Example
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 \(A\).
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 $\text{key}(u)$, 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 $\text{Adj}[\cdot]$ list for all elements in $Q$, the the algorithm take $O(n \log n + m \log n)=O(m \log n)$

<table>
<thead>
<tr>
<th>Prim’s MST</th>
<th>Prim’s MST Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MST – Prim($G,W,r$)</td>
<td></td>
</tr>
<tr>
<td>1. $Q \leftarrow V_G$</td>
<td></td>
</tr>
<tr>
<td>2. For each $u \in Q$</td>
<td></td>
</tr>
<tr>
<td>3. \hspace{1cm} do $\text{key}[u]=\infty$</td>
<td></td>
</tr>
<tr>
<td>4. $\text{key}[r] \leftarrow 0$</td>
<td></td>
</tr>
<tr>
<td>5. $p(r) = \text{NIL}$</td>
<td></td>
</tr>
<tr>
<td>6. While $Q \neq \emptyset$ do</td>
<td></td>
</tr>
<tr>
<td>7. \hspace{1cm} $u \leftarrow \text{Extract-Min}(Q)$</td>
<td></td>
</tr>
<tr>
<td>8. \hspace{1cm} For each $v \in \text{Adj}[u]$ do</td>
<td></td>
</tr>
<tr>
<td>9. \hspace{2cm} If $v \in Q$ and $w(u,u) &lt; \text{key}[v]$</td>
<td></td>
</tr>
<tr>
<td>10. \hspace{3cm} then $p(v) \leftarrow u$</td>
<td></td>
</tr>
<tr>
<td>11. \hspace{3cm} $\text{key}[v] \leftarrow w(u,v)$</td>
<td></td>
</tr>
</tbody>
</table>
Example
Example

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.