MST Algorithm Review

Minimal Cost Spanning Trees

Minimal Cost Spanning Tree (MST)
Problem:
Input: An undirected, connected graph G.
Output: The subgraph of G that
1) has minimum total cost as measured by
   summing the weight of all the edges in the
   subset, and
2) keeps the vertices connected.

Partition Property

Partition Property:
- Consider a partition of the vertices of G into subsets U and V
- Let e be an edge of minimum weight across the partition
- There is a minimum spanning tree of G containing edge e
Prim’s MST Algorithm

As with Dijkstra’s algorithm, uses a priority queue.

Running time is identical to Dijkstra’s algorithm implementations.

Kruskal’s MST Algorithm

Initially, each vertex is in its own MST.

Merge two MST’s that have the shortest edge between them.
  – Use a priority queue to order the unprocessed edges. Grab the minimum one at each step.

How to tell if an edge connects two vertices already in the same MST?
  – Use parent-pointer representation.

Kruskal's MST Algorithm

Cost is dominated by the time to remove edges from the heap.
  – Can stop processing edges once all vertices are in the same MST

Total cost: $\Theta((n + m) \log m)$. 
Baruvka’s MST Algorithm

- Create a forest of n trees
- Loop while (there is > 1 tree in the forest)
  - For each tree \( T_i \) in the forest
    - Find the smallest edge \( e = (u,v) \), in the edge list with \( u \) in \( T_i \) and \( v \) in \( T_j \neq T_i \)
    - \( e \) connects 2 trees from the forest into one.
- end loop

Baruvka’s Algorithm

- Similar to Kruskal’s Algorithm, but no Priority queue.

```plaintext
Algorithm BaruvaMST(G):
    T ← V (just the vertices of G)
    while T has fewer than \( n-1 \) edges do
        for each connected component C in T do
            let edge \( e \) be the smallest-weight edge from C to another component in T.
            if \( e \) is not already in T then
                add edge \( e \) to T
    return T
```

- Each iteration of the while-loop halves the number of connected components in \( T \).
- The running time is \( O(m \log n) \).

Network Flow Problem
Flow Network

- A network \( N \) consists of
  - A weighted digraph \( G \) with nonnegative integer edge weights, where the weight of an edge \( e \) is called the capacity \( c(e) \) of \( e \)
  - Two distinguished vertices, \( s \) and \( t \) of \( G \), called the source and sink or target, respectively, such that \( s \) has no incoming edges and \( t \) has no outgoing edges.
- Example:

---

Flow

- A flow \( f \) for a network \( N \) is an assignment of an value \( f(e) \) to each edge \( e \) that satisfies the following properties:
  1. Capacity Rule: For each edge \( e \), \( 0 \leq f(e) \leq c(e) \)
  2. Conservation Rule: For each vertex \( v \neq s,t \)
     \[ \sum_{e \in \text{in}(v)} f(e) = \sum_{e \in \text{out}(v)} f(e) \]

---

Flow

- The value of a flow \( f \), denoted \( |f| \), is:
  - the total flow from the source which is the same as,
  - the total flow into the sink
Maximum Flow

- A flow for a network $N$ is said to be maximum if its value is the largest of all flows for $N$.
- The maximum flow problem consists of finding a maximum flow for a given network $N$.
- Applications:
  - Hydraulic systems
  - Electrical circuits
  - Traffic movements
  - Freight transportation

A Simple Flow Algorithm

- Initialize all edges of the flow graph with zero flow.
- Loop while there is a path in $G_f$ from $s$ to $t$:
  - Find a path in $G_f$ from $s$ to $t$ (augmenting path).
  - Add to the flow graph the minimum residual capacity from this path.
  - Reduce the residual capacity of the edges.

Residual Graph ($G_r$)
A Simple Flow Algorithm

• Initialize all edges of the flow graph with zero flow
• Loop while ∃ a path in $G_f$ from s to t
  • find a path in $G_f$ from s to t (augmenting path)
  • Add to the flow graph the minimum residual capacity from this path
  • Reduce the residual capacity of the edges

• end Loop
Ford-Fulkerson’s Algorithm

- Initialize all edges of the flow graph with zero flow
- Loop while \( \exists \) a path in \( G_r \) from \( s \) to \( t \)
  - find a path in \( G_r \) from \( s \) to \( t \) (augmenting path)
  - Add to the flow graph the minimum residual capacity from this path
  - Reduce the residual capacity of the edges
  - Add a reversed path in the residual graph
- end Loop
Analysis

- Each augmenting path increases the flow value by at least 1.
- Let $f^*$ be a maximum flow, then in the worst case, Ford-Fulkerson’s algorithm performs $|f^*|$ flow augmentations.
- Finding an augmenting path and augmenting the flow takes $O(n + m)$ time.
- The running time of Ford-Fulkerson’s algorithm is $O(|f^*|(n + m))$. 

Classic worst case example:

Flow Graph ($G_f$)  
Residual Graph ($G_r$)