Overview

- General Algorithm
- Prim’s Algorithm
- Kruskal’s Algorithm
- Round Robin Algorithm
Problem Formulation

- \textit{Given:} undirected, connected, network \(G = (V, E, c)\).
- \textit{Find:} a tree \(T\) spanning \(V\), and such that its total cost (sum of its edge-costs) is minimized.
Main Observation

- \( C = \{X, \bar{X}\} \) is a cut in \( G \): a partition of the vertex set in two parts \( X \) and \( \bar{X} \). An edge crosses the cut if it has one endpoint in \( X \) and the other endpoint in \( \bar{X} \).
- \( e \) is a minimal cost edge across \( C \).
- at least one MST of \( G \) contains \( e \).

\[ \begin{align*}
X & \quad u \quad e \quad v \\
\bar{X} = V - X
\end{align*} \]

- Suppose that \( e \) is not in any MST of \( G \).
- Let \( T \) be one of the MSTs of \( G \).
- There is a path from \( u \) to \( v \) in \( T \).
- Let \( f \) be the first edge on this path crossing \( C \).
- \( T \setminus f \cup e \) is a tree spanning all vertices and \( c(T) \geq c(T \setminus f \cup e) \).
- Hence, \( T \setminus f \cup e \) is an MST, a contradiction.
Blue Rule

• Select a cut $C$ with no blue edges.
• Among all edges in $C$, select one of minimum cost and colour it blue.
• Repeat as long as possible.
Blue Rule - Correctness

Every MST can be obtained by repeated applications of the blue rule. Let $T$ be one of (possibly several) MSTs. Keep colouring edges of $T$ as long as the blue rule is not violated.

- If all edges of $T$ have been coloured, there is nothing to prove.
- $e$: one of the edges in $T$ that cannot be coloured by the blue rule.
  - $T - e$ falls into two parts, and vertices in these two parts form a cut.
  - No edge across the cut can be blue; only edges of $T$ have been coloured blue so far.
  - Since $e$ cannot be coloured blue, it is not of minimum cost among cut-edges.
  - This contradicts the assumption that $T$ is an MST.

$$\hat{X} = V - X$$
Blue Rule - Correctness

Only MSTs can be obtained by repeated applications of the blue rule.

- Blue rule generates a tree.
- Suppose that at some iteration the blue rule selects an edge $e$ such that $e$ together with previously coloured edges is in no MST.
- The edge set across the cut from which $e$ was selected must contain at least one uncoloured edge $e'$ in some MST $T$ containing all edges chosen before $e$.
- $c(e') = c(e)$.
- $T + e - e'$ is an MST. It contains $e$ as well as all edges coloured blue before $e$, a contradiction.
Round Robin Algorithm

- **Initialize**: $F$: a forest consisting of isolated vertices of $G$.
- **Terminate**: If $F$ is a tree then terminate.
- **Update**: Select any tree $T$ in $F$ (cut selection). Add to $F$ a minimum cost edge between $T$ and another tree in $F$ (edge selection). Go to the **Termination**.

- how to select a tree? can it be done in parallel?
- how to quickly identify the shortest “sticking out” edge?
- how to efficiently join trees?
- how to clean-up afterwards (if at all)?
Prim’s Algorithm

- **Initialize:** $F$: a forest consisting of isolated vertices of $G$. $T$: arbitrarily chosen tree in the forest.
- **Terminate:** If $F$ is connected then terminate.
- **Update:** Select an isolated vertex $v$ closest to $T$. Add the shortest edge between $v$ and $T$ to $F$. Go to the Termination.
Prim’s Algorithm - Data Structures

• Place all vertices on a heap. Let one vertex $s$ have 0-key. Let all other vertices have $\infty$-keys. Also, $\pi(s) = \text{nil}$.

• Delete the top vertex $v$ from the heap (using $\text{deletemin}$).

• Inspect every edge $(v, w)$ incident to $v$. If $w$ is on the heap and $|(v, w)| < \text{key}(w)$, then $\text{key}(w) := |(v, w)|$ and $\pi(w) := v$.

• $n \text{ deletemin}$ operations are required. Each requires $O(d \log_d n)$ time in $d$-heaps and $O(\log n)$ amortized time in Fibonacci heaps.

• $n \text{ insert}$ operations are required. Each requires $O(\log_d n)$ time in $d$-heaps and $O(1)$ amortized time in Fibonacci heaps.

• at most $m \text{ decrease}$ operations is required. Each requires $O(\log_d n)$ time in $d$-heaps and $O(1)$ amortized time in Fibonacci heaps.

• The worst-case time complexity for this implementation is
  
  - $d$-heaps: $O(nd \log_d n + m \log_d n)$
  - Fibonacci heaps: $O(n \log n + m)$
Kruskal’s Algorithm

- **Initialize:** $F$: forest consisting of isolated vertices of $G$. $E_s$: set of sorted edges.
- **Terminate:** If $F$ is connected then terminate.
- **Update:** Scan $E_s$ and delete edges with both end-vertices in the same tree. Let $e$ denote the first edge with end-vertices in different trees. Join the trees containing the end-vertices of $e$. Delete $e$ from $E_s$. Go to the Termination.
Kruskal’s Algorithm - Data Structures

• Sorting of edges requires $O(m \log m) = O(m \log n)$.

• Checking for cycles and joining trees can be carried out in $O(m \alpha(m, n))$ time.

• Overall complexity is $O(m \log n)$.
Round Robin - Data Structures

- Components in the forest are represented as disjoint sets of their vertices. Data structures permitting \( m \) find-operations, and \( n-1 \) union-operations in \( O(m\alpha(m, n)) \) time, where \( \alpha \) is a very slowly growing function of \( m \) and \( n \). In particular, it is dominated by \( O(m \log n) \).

- Edges “sticking out” from each tree are placed on heaps.

- Trees are placed on a queue. The first tree \( T \) in the queue is picked up. The shortest edge to any other \( T' \) tree is found on its heap. Components \( T \) and \( T' \) are merged and placed at the rear of the queue. Heaps for \( T \) and \( T' \) are merged.

- If heaps are appropriately implemented (leftist heaps with lazy deletion and lazy meld), an \( O(m \log \log n) \) time algorithm can be obtained.
Round Robin - Complexity Analysis

- \( n \) makeheap operations require \( O(m) \) time.
- \( n - 1 \) lazymeld operations require \( O(n) \) time.
- findmin?
- \( T_i = i\)-th tree selected.
- \( m_i \) = number of edges (including deleted and dummy edges) on the heap associated with \( T_i \) when \( T_i \) is selected.
- \( k_i \) = number of deleted or dummy edges encountered during the \( i\)-th findmin operation.
- Each tree on the queue during pass \( j \) contains at least \( 2^{j-1} \) vertices.
- At most \( \lceil \log n \rceil \) passes of the queue are needed.
- \( \sum_{i=1}^{n-1} m_i \leq (2m + n - 1) \lfloor \log n \rfloor \).
- \( \sum_{i=1}^{n-1} k_i \leq 2m + n - 1 \).
- \( i\)-th findmin operation requires \( O(k_i \max\{1, \log \frac{m_i}{k_i+1}\}) \) time.
- \( i\)-th findmin is small if \( k_i \leq m_i/(\log n)^2 - 1 \) and large otherwise.
Round Robin - Complexity Analysis (cont.)

- Assume that all findmin are small. Total time for small findmin operations is $O(\sum_{i=1}^{n-1} k_i \max\{1, \log \frac{m_i}{k_i+1}\})$

$$\sum_{i=1}^{n-1} k_i \max\{1, \log \frac{m_i}{k_i+1}\} \leq \sum_{i=1}^{n-1} k_i \log m_i < \sum_{i=1}^{n-1} \frac{m_i}{(\log n)^2} \log m_i <$$

$$\sum_{i=1}^{n-1} \frac{m_i}{(\log n)^2} 2 \log n = 2 \sum_{i=1}^{n-1} \frac{m_i}{\log n} \leq 2 \frac{(2m + n - 1) \log n}{\log n} \leq 6m$$

- $n-1$ small findmin require $O(\sum_{i=1}^{n-1} k_i \max\{1, \log \frac{m_i}{k_i+1}\}) = O(m)$
Round Robin - Complexity Analysis (cont.)

- Assume that all findmin are large. Total time for large findmin operations is $O(\sum_{i=1}^{n-1} k_i \max\{1, \log \frac{m_i}{k_i + 1}\})$.

$$\sum_{i=1}^{n-1} k_i \max\{1, \log \frac{m_i}{k_i + 1}\} < \sum_{i=1}^{n-1} k_i \max\{1, \log \frac{m_i^2}{m_i}\} =$$

$$\sum_{i=1}^{n-1} k_i \max\{1, 2 \log \log n\} = 2 \sum_{i=1}^{n-1} k_i \log \log n \leq$$

$$2(2m + n - 1) \log \log n \leq 6m \log \log n$$

- $n - 1$ large findmin require

$$O(\sum_{i=1}^{n-1} k_i \max\{1, \log \frac{m_i}{k_i + 1}\}) = O((m \log \log n)\alpha(m \log \log n, n)) =$$

$$O(m \log \log n)$$

since each operation involves deletion check and in total they require

$$O(\alpha(m \log \log n, n)) < 2 \text{ time.}$$

- Asymptotically faster than any other algorithm for sparse graphs.

- Asymptotically faster algorithms for dense graph exist ($O(n^2)$) but the round robin bound is likely to be overly pessimistic.

- appropriate clean-ups can improve the performance of round robin for dense graphs.
Minimum Spanning Trees - Summary

- Round robin algorithm: $O(m \log \log n)$.
- Prim’s algorithm: $O(n \log n + m)$.
- Kruskal’s algorithm: $O(m \log n)$.

- Round robin algorithm: leftist heaps and find-union algorithm.
- Prim’s algorithm: Fibonacci heaps.
- Kruskal’s algorithm: find-union algorithm.