Minimal Spanning Tree (MST)

- **Spanning tree** of an undirected graph G
  - A tree that is a sub-graph of G containing ALL vertices

- **Minimal spanning tree** of a weighted graph G
  - Spanning tree with minimal total weight

- **G** must be a connected graph

- **Applications**
  - Lowest cost set of roads connecting a set of towns
  - Shortest cable connecting a set of computers
Prim’s Algorithm for MST

- Pick an arbitrary vertex
- Grow MST by choosing a new vertex \( v \) and edge \( e \)
  - such that they are guaranteed to be in the final, correct MST
  - Select least-cost (minimal) edge \( e(u,v) \) such that
    - \( u \) is already in MST
    - \( v \) is not in MST as yet
- Keep doing this until all vertices are in MST
- This is a GREEDY algorithm
  - a locally optimizing strategy leading to a global optimum
Properties of any tree hence MST

- Path between two nodes $a$ and $b$ in MST is unique
- Cycles in MST
  - there are no cycles
  - If $a$ and $b$ are non-adjacent, adding the edge $(a,b)$ creates a cycle
  - Removing any edge on that cycle makes it a tree again
How greedy works for MST

- Consider each stage M with partial MST
  - Add the least-cost edge to M to obtain the next stage M'
  - The resulting MST will be minimal.

- Exchange Argument
  - Suppose we can create an MST by not taking the minimal cost edge
  - Call the minimal edge e, and the non-minimal edge taken e'
  - Build the rest of the spanning tree
  - We can now make a lower cost spanning tree by removing e’ and adding e
  - Hence the spanning tree with e’ in it was not minimal
Prim's Algorithm Code Structure

// Pick vertex r and initialize V_t, E_t, d and e
V_t = { r } ; E_t = { } // MST in construction
d[r] = 0 ; // d is a heap
∀ v ∈ V if ((r,v) ∈ E) { d[v] = w(r,v) ; e[v] = r ; }
else d[v] = ∞ ;

// grow the MST
while V_t != V
    Select vertex u from V-V_t with minimal d[u] ;
    V_t = V_t + u; E_t = E_t + (u,e[u]) ;
    // update d and e
    ∀ v ∈ V-V_t if (w(u,v) < d[v]) d[v]=w(u,v);e[v]=u
Complexity, parallelization of Prim

- while-loop executed n-1 times
- Loop-body $O(n)$ if arrays are used
- Sequential complexity: $O(m \log n)$
- while-loop is sequential in nature, because of the data dependencies in $V_t, E_t, d$ and $e$
- $\forall$ loops can be parallelized
Parallel Implementation of Prim

- **Data distribution**
  - Each PE has data for $n/p$ vertices
  - Adjacency matrix $A$ is block striped (column-wise)
  - $d$ and $e$ block striped

- PEs compute a local minimum $u_l$
- Local minima accumulate to give global minimum in PE$_0$
- PE$_0$ broadcasts global minimum $u_g$
- PE owning $u_g$ updates $V_t,E_t$
- All PEs update their partition of $d$ and $e$ using their columns of $A$
Single Source Shortest Path - SSSP

- Given a vertex \( s \) and weighted graph \( G \), find the shortest distances from \( s \) to each vertex
- Dijkstra’s algorithm (very much like Prim)

\[
V_t = \{ s \} \\
\forall v \in V-V_t \text{ if } ((s,v) \in E) \ l[v] = w(s,v); \text{ else } l[v] = \infty; \\
\text{while } V_t \neq V \\
\quad \text{Select vertex } u \text{ from } V-V_t \text{ with minimal } l[u] \\
\quad V_t = V_t + u \\
\forall v \in V-V_t \ l[v] = \min(l[v], l[u]+w(u,v))
\]
Parallel SSSP

- Very similar to Prim
  - Difference: MST selects minimal edge, SSSP takes minimal path length

- Data distribution
  - n/p vertices per PE
  - Column distribute A
  - block distribute l

- Find minimal $u_l$ locally
- Accumulate to obtain global minimum $u_g$
- Broadcast global minimum $u_g$
- Every PE updates its $l$ block using its column-block of A
All pairs shortest paths - APSP

- Find length of shortest path between all vertex pairs
  - n*n distance matrix D: D\(ij\) is shortest distance for \(v_i \rightarrow v_j\)
- Algorithm: Floyd’s APSP
Dynamic Programming approach

- Formulate the problem in a recursive fashion

- Reverse this formulation to create a BOTTOM UP solution
  - Use solutions for smaller problems to create solutions for larger ones

- There can be multiple recursive formulations
  - Recurrence on path length (Matrix Multiply formulation. $n^3 \log n$)
  - Recurrence on node set (Floyd’s algorithm, $n^3$)
Floyd’s APSP

Terms used
- node (sub)set $V_k = \{v_1, v_2, \ldots, v_k\}$
- $P_{ij}^k =$ minimal length path from $v_i$ to $v_j$ passing through nodes in $V_k$
- $d_{ij}^k =$ length of the path $P_{ij}^k$

Recursion: based on node sets
- Two possibilities: $v_k$ in $P_{ij}^k$ or not
  - $v_k$ not in $P_{ij}^k$: $P_{ij}^k = P_{ij}^{k-1}$ and $d_{ij}^k = d_{ij}^{k-1}$
  - $v_k$ in $P_{ij}^k$: $P_{ij}^k = P_{ik}^{k-1} + P_{kj}^{k-1}$ and $d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$
- $d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$ for $k > 0$
  
  $= w(v_i, v_j)$ for $k = 0$

Solution $D = D^n$
Floyd’s APSP (Sequential)

\[ D^0 = A \]

for \( k = 1 \) to \( n \)

\[ \text{for } i = 1 \text{ to } n \]

\[ \text{for } j = 1 \text{ to } n \]

\[ D_{ij}^k = \min(D_{ij}^{k-1}, D_{ik}^{k-1} + D_{kj}^{k-1}) \]

\( O(n^3) \) sequential time complexity

\( O(n^2) \) space complexity
Floyd Parallel

- Mesh checkerboard partitioning
- Iteration k: Broadcast k-th row and k-th column of D

for $k = 1$ to $n$

- each PE having a segment of row $k$ of $D^{k-1}$ broadcast it in its column
- each PE having a segment of column $k$ of $D^{k-1}$ broadcasts it in its row
- each PE waits to receive the needed segments of $D^{k-1}$
- each PE computes its part of $D^k$

- Note that this algorithm can be pipelined like Gaussian elimination or LUD
Floyd Parallel: update in place

- In the kth iteration
- $D_{ik}$ and $D_{kj}$ are broadcast and do not change
- other elements $D_{ij}$ depend on $D_{ik}$ and $D_{kj}$ and themselves (no other elements depend on $D_{ij}$)

So there are no data hazards, the $D_{ij}$s are independent, and can be updated in place
Transitive Closure

- Given: graph G=(V,E)
  Transitive closure: G*=(V,E*)
  - \( E^* = \{(v_1,v_2) \mid \exists \text{ path from } v_1 \text{ to } v_2 \text{ in } G\} \)

- Connectivity matrix A*
  - \( A_{ij}^* = 1 \) if \((v_i,v_j)\) in E* or \(i = j\)
    - = 0 otherwise

- Use Floyd
  - replacing \textit{min} by \textit{or} and \textit{sum} by \textit{and}