Course Name: Analysis and Design of Algorithms

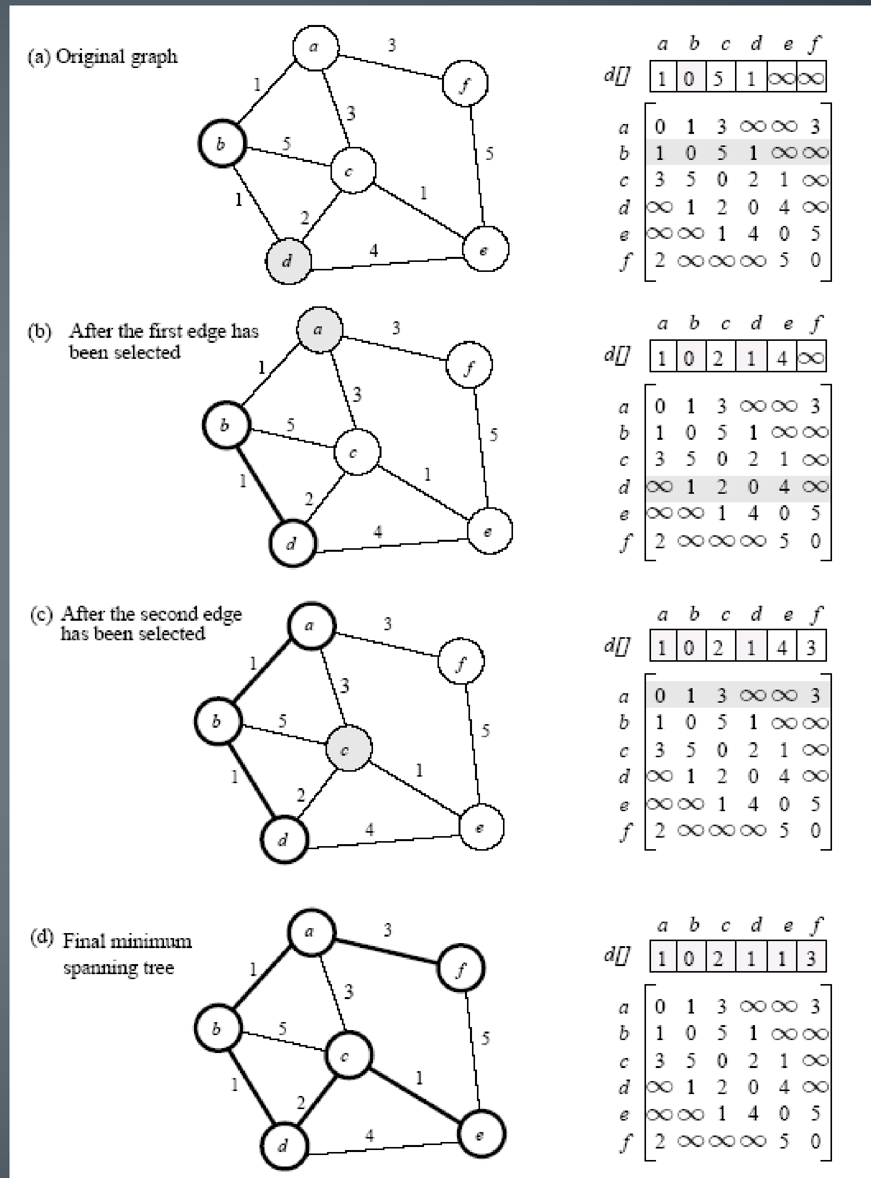
Topics to be covered

- Minimum Spanning Tree: Prim's Algorithm
- Single-Source Shortest Paths: Dijkstra's Algorithm

Minimum Spanning Tree: Prim's Algorithm

- Prim's algorithm for finding an MST is a greedy algorithm.
- Start by selecting an arbitrary vertex, include it into the current MST.
- Grow the current MST by inserting into it the vertex closest to one of the vertices already in current MST.

Minimum Spanning Tree: Prim's Algorithm



Prim's minimum spanning tree algorithm.

Minimum Spanning Tree: Prim's Algorithm

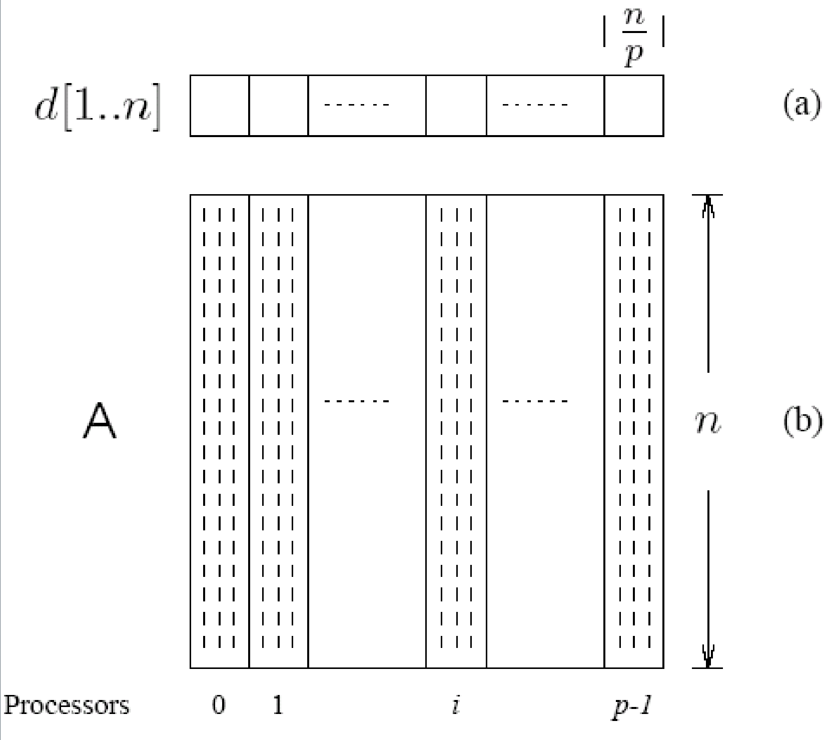
procedure PRIM_MST(V, E, w, r)
begin
    $V_T := \{r\};$
    $d[r] := 0;$
    for all $v \in (V - V_T)$ **do**
        if edge (r, v) exists set $d[v] := w(r, v);$
        else set $d[v] := \infty;$
    while $V_T \neq V$ **do**
    begin
        find a vertex u such that $d[u] := \min\{d[v] | v \in (V - V_T)\};$
        $V_T := V_T \cup \{u\};$
        for all $v \in (V - V_T)$ **do**
            $d[v] := \min\{d[v], w(u, v)\};$
    endwhile
end PRIM_MST

Prim's sequential minimum spanning tree algorithm.

Prim's Algorithm: Parallel Formulation

- The algorithm works in *n* outer iterations - it is hard to execute these iterations concurrently.
- The inner loop is relatively easy to parallelize. Let *p* be the number of processes, and let *n* be the number of vertices.
- The adjacency matrix is partitioned in a 1-D block fashion, with distance vector *d* partitioned accordingly.
- In each step, a processor selects the locally closest node, followed by a global reduction to select globally closest node.
- This node is inserted into MST, and the choice broadcast to all processors.
- Each processor updates its part of the *d* vector locally.

Prim's Algorithm: Parallel Formulation



The partitioning of the distance array *d* and the adjacency matrix *A* among *p* processes.

Prim's Algorithm: Parallel Formulation

- The cost to select the minimum entry is $O(n/p + \log p)$.
- The cost of a broadcast is $O(\log p)$.
- The cost of local updation of the d vector is $O(n/p)$.
- The parallel time per iteration is $O(n/p + \log p)$.
- The total parallel time is given by $O(n^2/p + n \log p)$.
- The corresponding isoefficiency is $O(p^2 \log^2 p)$.

Single-Source Shortest Paths

- For a weighted graph $G = (V,E,w)$, the *single-source shortest paths* problem is to find the shortest paths from a vertex $v \in V$ to all other vertices in V.
- Dijkstra's algorithm is similar to Prim's algorithm. It maintains a set of nodes for which the shortest paths are known.
- It grows this set based on the node closest to source using one of the nodes in the current shortest path set.

Single-Source Shortest Paths: Dijkstra's Algorithm

procedure DIJKSTRA_SINGLE_SOURCE_SP(V, E, w, s)
begin
    $V_T := \{s\}$;
    for all $v \in (V - V_T)$ **do**
        if (s, v) exists set $l[v] := w(s, v)$;
        else set $l[v] := \infty$;
    while $V_T \neq V$ **do**
    begin
        find a vertex u such that $l[u] := \min\{l[v] | v \in (V - V_T)\}$;
        $V_T := V_T \cup \{u\}$;
        for all $v \in (V - V_T)$ **do**
            $l[v] := \min\{l[v], l[u] + w(u, v)\}$;
    endwhile
end DIJKSTRA_SINGLE_SOURCE_SP

Dijkstra's sequential single-source shortest paths algorithm.

Dijkstra's Algorithm: Parallel Formulation

- Very similar to the parallel formulation of Prim's algorithm for minimum spanning trees.
- The weighted adjacency matrix is partitioned using the 1-D block mapping.
- Each process selects, locally, the node closest to the source, followed by a global reduction to select next node.
- The node is broadcast to all processors and the *l*-vector updated.
- The parallel performance of Dijkstra's algorithm is identical to that of Prim's algorithm.