Algorithms and Data Structures II

Lecture 4:

Weighted Graphs

https://elms.u-aizu.ac.jp

Weighted Graphs

- A weighted graph is a graph $G(V, E)$ with real valued weights assigned to each edge. Equivalently, a weighted graph is a triple $G(V, E, W)$, where V is the set of vertices, E is the set of edges, and W is a function that maps the elements of E into the reals. The weight on edges are also called distance or cost.

distance matrix

- A weighted graph $G(V, E, W)$ can be represented by a distance matrix

$$D_{n \times n}, \quad n = |V|,$$

where

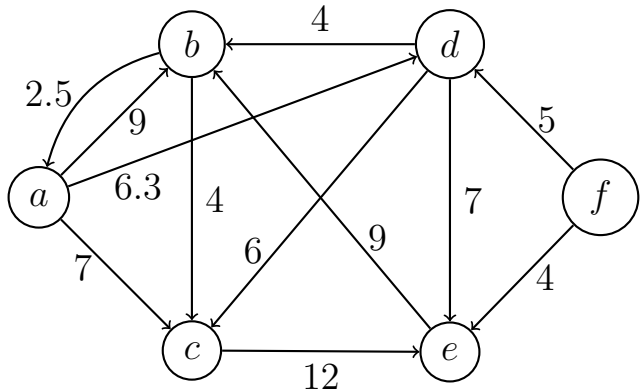
$$D[i, i] = 0,$$

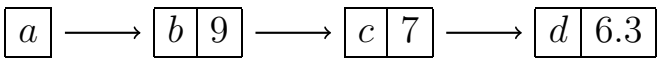
and for $1 \leq i \neq j \leq n$, if edge $(i, j) \in E$ then $D[i, j]$ is the weight of (i, j), otherwise $D[i, j]$ is infinite ∞ (a sufficient large number in practice).

Graph Representations

	a	b	c	d	e	f
a	0	9	7	6.3	M	M
b	2.5	0	4	M	M	M
c	M	M	0	M	12	M
d	M	4	6	0	7	M
e	M	9	M	M	0	M
f	M	M	M	5	4	0

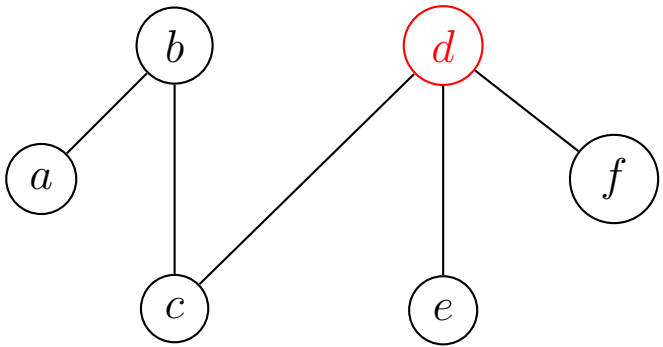
$M = \infty$





Minimum Spanning Trees

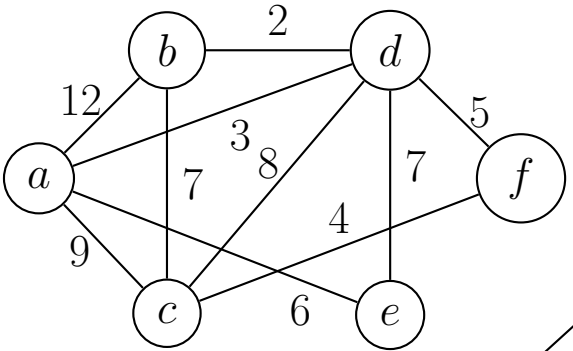
- A rooted tree is a graph in which for each vertex v, there is exactly one path of the form $(r, x)(x, y), \ldots, (z, v)$, where r is a special vertex called the root of the tree.



Subgraph, spanning tree

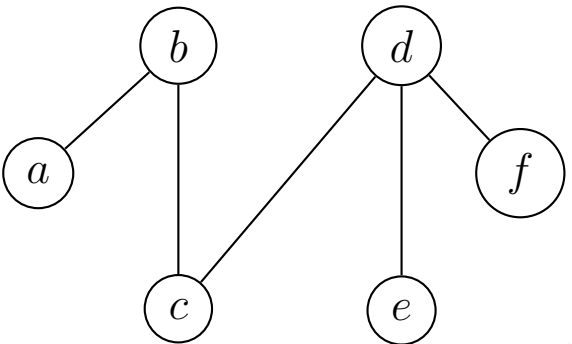
- A subgraph of a graph $G(V, E)$ is a graph $G'(V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$.
- A spanning tree of a graph $G(V, E)$ is a subgraph $T(V', E')$ such that T is a rooted tree and $V' = V$.
- A spanning tree of a graph can be found by DFS or BFS.

Weighted Graph and its Spanning Tree



A Weighted Graph G

A Spanning Tree of G



Minimum Spanning Tree

- Let $T(V', E')$ be a spanning tree of a weighted graph G and

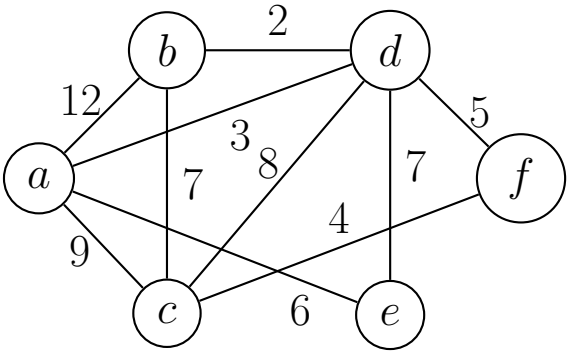
$$W(T) = \sum_{(v,w) \in E'} W(v, w)$$

be the sum of weights of edges in T, where $W(v, w)$ denotes the weight of edge (v, w).

- A minimum spanning tree (MST) of G is a spanning tree T' of G such that

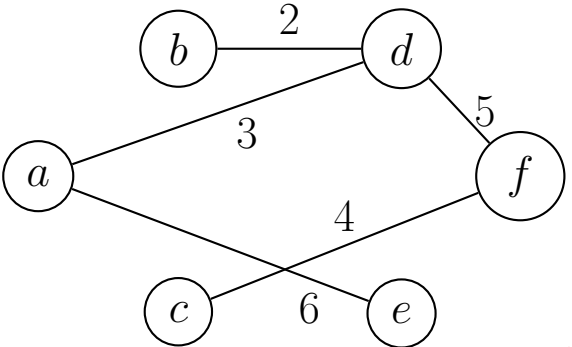
$$W(T') = \min\{W(T) | \ T \text{ is a spanning tree of } G\}.$$

Weighted graph and its MST



A Weighted Graph G

A Minimum Spanning Tree of G



MST example

- Minimum spanning tree problem has many applications such as in constructing electric power networks or telephone networks.

Notes on the MST

- There may be more than one MST (several trees with the same weight);
- If G is connected, every vertex is in the MST; otherwise, there is the minimum spanning forest;
- Naïve approach: list all spanning trees (how?) and compute the weights; then find the minimum. Inefficient....

Prim's algorithm

1. Pick an arbitrary vertex r of $G(V, E)$ as the root of the minimum spanning tree of G. Assume a partial solution (spanning tree) T has been obtained (initially, $T = \{r\}$).
2. Choose an edge (v, w) such that $v \in T$, $w \in V - T$, and the weight of edge (v, w) is the minimum among that of edges from the nodes of T to nodes of $V - T$.
3. Add the node w into T.
4. Repeat the above process until $T = V$.

Key Idea of Prim's algorithm

- The key point of the algorithm is how to determine the edge (v, w) with the minimum weight.
- For an edge (v, w), let $D[v, w]$ be the weight of (v, w). For each node w in $V - T$, let $d[w] = \min\{D[v, w] | v \in T\}$.
- If w is not adjacent to any node of T, $d[w] = \infty$. Initially, $T = \{r\}$ and for $w \neq r$, $d[w] = D[r, w]$.

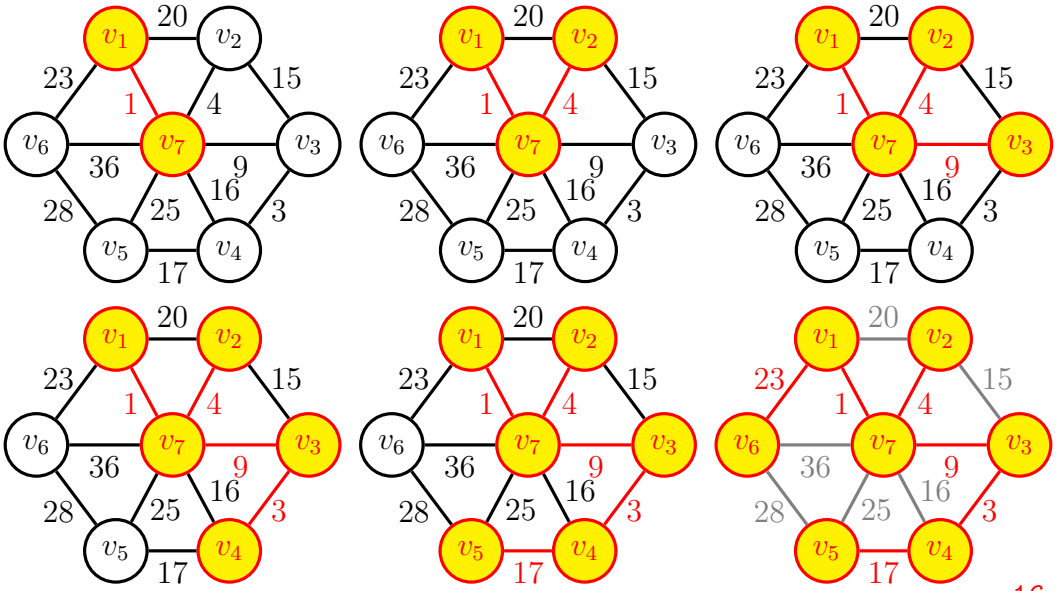
Key Idea of Prim's algorithm 2

- When a new vertex c is added to T, for $w \in V - T, d[w]$ is updated to the minimum of $d[w]$ and $D[c, w]$, i.e., $d[w] = \min\{d[w], D[c, w]\}$.
- An additional array adj$[w]$ is used for each $w \in V - T$ to indicate the node v in T such that $D[v, w] = d[w]$.
- Prim's algorithm may therefore be implemented as follows.

Prim's Algorithm

$X = \emptyset; T = \{r\}$ /* r is the root. */
for (all $w \in V - T$) $\{d[w] = D[r, w]; adj[w] = r;\}$
for ($i = 1; i < |V|; i++$) {
    Find $v \in V - T$ such that $d[v] = \min\{d[x] \mid x \in V - T\}$;
    /*Add $(adj[v], v)$ to the tree and adjust distance */
    $X = X \cup \{(adj[v], v)\}$; $T = T \cup \{v\}$;
    for (every node $w \in V - T$ adjacent to v) {
        if ($D[v, w] < d[w]$) {
            $d[w] = D[v, w]$; $adj[w] = v$;}
    }
}

Prim's Algorithm



Complexity of Prim's Algorithm

- If the graph is represented by an adjacent (distance) matrix, the time complexity of Prim's algorithm is $O(|V|^2)$.
- Prim's algorithm can be made more efficiently by maintaining the graph using adjacency lists and keeping a priority queue of the nodes not in T. Under this implementation, the time complexity of Prim's algorithm is $O((|V| + |E|) \log |V|)$.

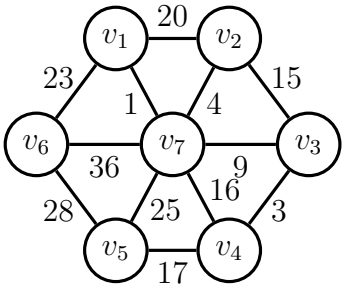
Kruskal's Algorithm

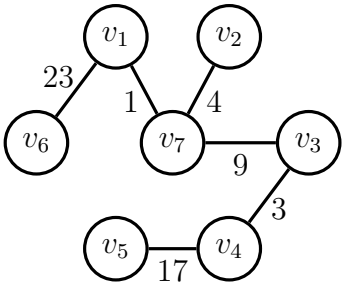
- Kruskal's Algorithm is another algorithm for finding a minimum spanning tree of a graph.
- The algorithm finds the minimum spanning tree by selecting the edges in order of smallest weight and including an edge into the tree if the edge does not cause a cycle.
- In the algorithm, Q is a priority queue (a heap).

Kruskal's Algorithm

$X = \{\{v\} | v \in V\}; T = \emptyset;$
Construct the edges of E into a minimum heap Q;
while $(Q \neq \emptyset)\{$
    Delete an edge (v, w) of minimum weight from Q
        and restore the heap condition;
    if ($v \in V_i$ and $w \in V_j$ with $V_i \neq V_j$ and $V_i, V_j \in X)\{$
        Replace V_i and V_j in X by $V_i \cup V_j$;
        $T = T \cup \{(v, w)\}$;
    }
}

Kruskal's process





Edge	Action	Sets in X(connected components)
		$\{v_1\}, \{v_2\}, \{v_3\}, \{v_4\}, \{v_5\}, \{v_6\}, \{v_7\}$
(v_1, v_7)	Add	$\{v_1, v_7\}, \{v_2\}, \{v_3\}, \{v_4\}, \{v_5\}, \{v_6\}$
(v_3, v_4)	Add	$\{v_1, v_7\}, \{v_2\}, \{v_3, v_4\}, \{v_5\}, \{v_6\}$
(v_2, v_7)	Add	$\{v_1, v_2, v_7\}, \{v_3, v_4\}, \{v_5\}, \{v_6\}$
(v_3, v_7)	Add	$\{v_1, v_2, v_3, v_4, v_7\}, \{v_5\}, \{v_6\}$
(v_2, v_3)	Reject	
(v_4, v_7)	Reject	
(v_4, v_5)	Add	$\{v_1, v_2, v_3, v_4, v_5, v_7\}, \{v_6\}$
(v_1, v_2)	Reject	
(v_1, v_6)	Add	$\{v_1, v_2, v_3, v_4, v_5, v_7, v_6\}$

Complexity of Kruskal's algorithm

- The time complexity of Kruskal's algorithm is $O(|E| \log |E|)$ if adjacency lists and priority queue are used.

Boruvka's algorithm

- Boruvka's algorithm (1926): perhaps the easiest one of the three classical algorithms for computer implementation (no complicated data structures are needed). It does the steps like Prim's algorithm, but in parallel all over the graph at the same time.
- Boruvka's algorithm:
  1. Make a list L of n trees, each a single vertex;
  2. While (L has more than one tree)
     - **2.1** for each T in L, find the smallest edge connecting T to $G - T$;
     - **2.2** add all these edges to the MST (merge);
- Time complexity: $O(|E| \log |V|)$
- Hybrid between Boruvka and Prim: $O(|E| \log \log |V|)$

Maggs & Plotkin Algorithm (1994)

- Weighted Graph $G(V, E, W)$, $|V| = n$

  `Initialization:` $d[i,j]^{(0)} = d[i,j], 1 \le i,j \le n$;

  `for` $k = 1$ `to` n `do`

  `for` $i = 1$ `to` n `do`

  `for` $j = 1$ `to` n `do`

  $$d[i,j]^{(k)} = \min\{d[i,j]^{(k-1)}, \max(d[i,k]^{(k-1)}, \ d[k,j]^{(k-1)})\};$$

- Edges $d[i,j]^{(n)} = d[i,j]^{(0)}$ are from MST
- Complexity is $O(|V|^3)$