ALGORITHMS AND DATA STRUCTURES II

Lecture 4
Spanning Tree,
Weighted Graphs,
Prim's and Kruskal's algorithms.

Lecturer: K. Markov
markov@u-aizu.ac.jp

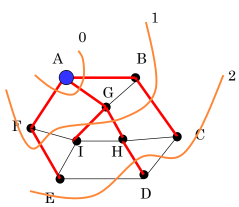
1/26

SPANNING TREE

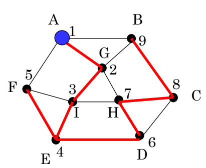
- Assume you have an **undirected graph** **G = (V,E)**
- Spanning tree of graph **G** is tree **T = (V,E$_T$ ⊆ E, R)**
  - Tree has same set of nodes.
  - All tree edges are graph edges.
  - Root of tree is R.
- **Think:** "smallest set of edges needed to connect everything together".

2/26

SPANNING TREE



Breadth-first Spanning Tree

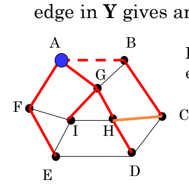


Depth-first spanning tree

3/26

SPANNING TREE

- **Properties:**
  - In any tree **T** = (**V**,**E**), |**E**|=|**V**| - 1
  - For any edge ***e*** in **G** but not in **T**, there is a simple cycle **Y** containing only edge ***e*** and edges in spanning tree.
  - Moreover, inserting edge ***e*** into **T** and deleting any edge in **Y** gives another spanning tree **T'**.

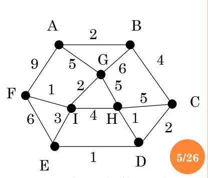


EXAMPLE:
edge (**H**,**C**):
simple cycle is (**H**,**C**,**B**,**A**,**G**,**H**)
adding (**H**,**C**) to **T** and deleting (**A**,**B**) gives another spanning tree

4/26

WEIGHTED GRAPHS

- **Definition:**
  - 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 the set of weights. The weights on edges are also called **distances** or **costs**.



Weighted Graph

5/26

WEIGHTED GRAPHS

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

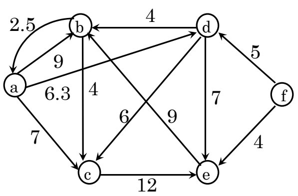
$$D_{n\times n} = \begin{bmatrix} d(1,1) & \dots & d(1,n) \\ \dots & \dots & \dots \\ d(n,1) & \dots & d(n,n) \end{bmatrix} \quad n=|V|$$

where $d(i,i) = 0$,
and for $1 \le i \ne j \le n$, if edge $(i,j) \in E$ then $d(i,j)$ is the weight of (i,j), otherwise $d(i,j)$ is infinite ∞ (a sufficiently large number in practice).

6/26

WEIGHTED GRAPHS

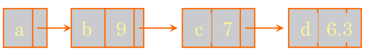
- **Representation:**



Distance Matrix

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

Adjacency list



MINIMUM SPANNING TREE (MST)

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

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

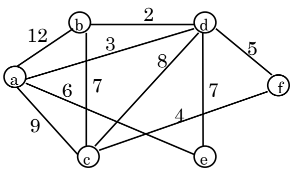
be the sum of weights of edges in T, where $W(v,w)$ denotes the weight of edge (v,w).

- A **minimum spanning tree** of G is a spanning tree T^m of G such that

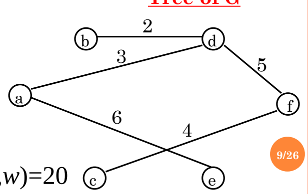
$$W(T^m) = \min_T \{W(T)\}.$$

MINIMUM SPANNING TREE (MST)

Weighted Graph G



Minimum Spanning Tree of G



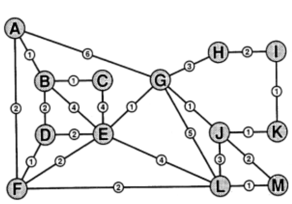
$W(T^m) = \Sigma_{(v,w) \in E'} W(v,w) = 20$

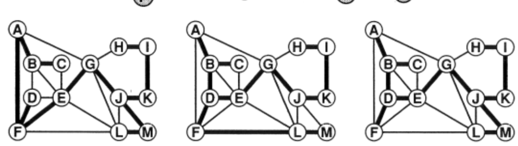
MINIMUM SPANNING TREE (MST)

- **Minimum spanning tree** is useful when we attempt to minimize the cost of connecting all the nodes.
- **Applications:**
  - Constructing electric power networks or telephone networks.
  - Making printed circuit boards (PCBs).
  - Etc.
- **Note:** Minimum spanning tree need not to be unique. (simple examples)

MINIMUM SPANNING TREE (MST)

Weighted Graph G





Multiple Minimum Spanning Trees of G

MINIMUM SPANNING TREE (MST)

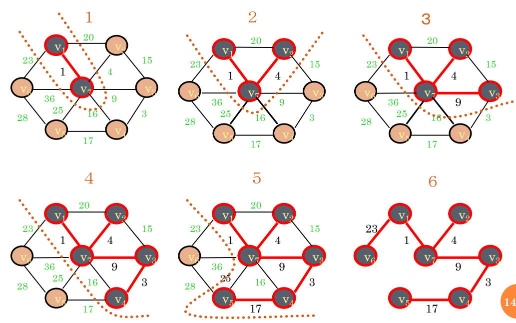
- **Building MST** – two strategies:
  - **Prim's algorithm** – start with a root node s and try to grow a tree from s outward. At each step, add the node that can be attached as cheaply as possible to the partial tree we already have.
  - **Kruskal's algorithm** – start with no edges and successively insert edges from E in order of increasing cost. If an edge makes cycle when added, skip this edge.

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 $v \in T$, $w \in V\text{-}T$, 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 2) and 3) until $T = V$.

13/26

PRIM'S ALGORITHM



14/26

PRIM'S ALGORITHM

- If the graph is represented by an **adjacency (distance) matrix**, the time complexity of Prim's algorithm is $O(|V|^2)$.
- Prim's algorithm can be made more efficient 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 $O((|V|+|E|)\log|V|)$.

15/26

PRIM'S ALGORITHM

- Implementation:

```
def MST-PRIM (G, w, r)
// Graph G with set of nodes G.V, weight matrix w and
// root node r. MST is the edges set A={(v,v.π), v ∈ V-r}.
  for each u ∈ G.V:
     u.key = ∞
     u.π = NIL
  r.key = 0
  Q = Min-Priority-Queue (G.V)
  while Q ≠ ∅:
     u = Extract-Min (Q)
     for each v ∈ G.Adj[u]:
        if v ∈ Q and w(u,v) < v.key:
           v.π = u
           v.key = w(u,v)
```

16/26

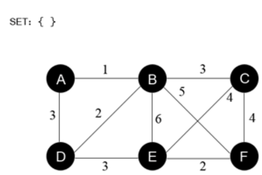
PRIM'S ALGORITHM

- Implementation notes:
  - During execution of the algorithm, all nodes that are **NOT** in the **MST**, reside in the **minimum priority queue** based on the *key* attribute.
  - For each node ***v***, the attribute ***v.key*** is the minimum weight of any edge connecting ***v*** to a node in the **MST**.
  - If there is no edge $v.key = \infty$.
  - The attribute $v.\pi$ names the parent of ***v*** in the **MST**.

17/26

PRIM'S ALGORITHM

- Animated example:



18/26

KRUSKAL'S ALGORITHM

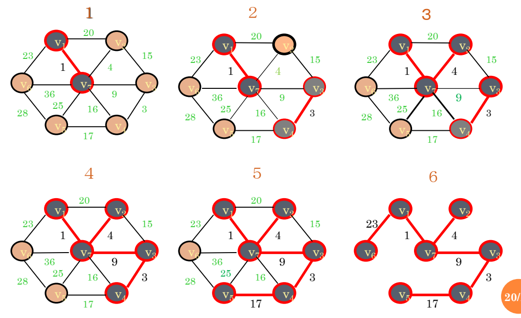
1) Pick the cheapest edge available and add it to the **MST**

   $e_0 = \min_{v,u} w(v,u)$, $\quad A=\{e_0\}$

2) Choose next cheapest edge $e=(v,w)$
3) If adding e to the A makes a cycle, do not add it.
4) Repeat the above 2) and 3) until all edges are chosen.

19/26

KRUSKAL'S ALGORITHM



20/26

KRUSKAL'S ALGORITHM

- Implementation:

```
def MST-KRUSKAL (G, w)
// Graph G with set of nodes G.V, weight matrix w.
// MST is the edges set A={}.
  A = ∅
  for each v ∈ G.V:
     MAKE-SET (v)
  Sort edges of G.E into non-decreasing order by weight w
  for each edge (u, v) ∈ G.E, taken in non-decreasing order of w:
     if FIND-SET (u) ≠ FIND-SET (v):
        A = A ∪ {(u, v)}
        UNION (u, v)
  return A
```

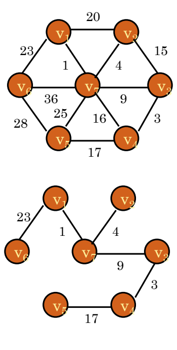
21/26

KRUSKAL'S ALGORITHM

- Implementation notes.
  - **UNION-FIND** data structure:
    - Given a node u the operation **FIND-SET (u)** will return the name of the set containing u.
    - To test if two nodes u and v are in the same set, we simply check if **FIND-SET(u) = FIND-SET(v)**
    - The operation **UNION (u, v)** will take two sets containing u and v respectively and will merge them into a single set.
    - To make a set from one or several nodes, we use the **MAKE-SET** () operation.

22/26

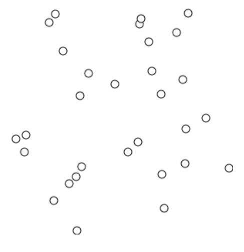
KRUSKAL'S ALGORITHM



Edge	Action	Sets
		$\{v_1\},\{v_2\},\{v_3\},\{v_4\},\{v_5\},\{v_6\},\{v_7\}$
(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	$\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}$

KRUSKAL'S ALGORITHM

- Animated example based on Euclidean distance:



24/26

KRUSKAL'S ALGORITHM

- Complexity.
  - Initializing set $\boldsymbol{A}$ takes $\boldsymbol{O}(1)$.
  - Making $|\mathbf{V}|$ sets takes $\boldsymbol{O}(\boldsymbol{V})$ time.
  - Time to sort the edges by weight is $\boldsymbol{O}(\boldsymbol{E}log\boldsymbol{E})$.
  - There are $|\mathbf{E}|$ FIND-SET and UNION operations taking $\boldsymbol{O}(\boldsymbol{E})$ time.
  - Since the graph is connected, $|\boldsymbol{E}| \geq |\boldsymbol{V}|$-1 and $|\boldsymbol{E}| < |\boldsymbol{V}|^2$, $log\ |\boldsymbol{V}|^2 = 2log|\boldsymbol{V}|$ which is $\boldsymbol{O}(log\boldsymbol{V})$.
  - Total running time is $\boldsymbol{O}(\boldsymbol{E}log\boldsymbol{V})$.

THAT'S ALL FOR TODAY!