CS 303, Paul Cao,
Plan: Greedy Algorithm (MST Construction)
Review:

Graphs review (App. B)

<u>Digraph:</u> (directed graph) $G=(V,E)$

- set V of vertices
- set E of edges $\in V \times V$ (ordered pair of vertices)

<u>undirected graph:</u> E contains undirected pair of vertices

- $|E| \in O(V^2)$
  $\hookrightarrow$ because ${}_2C_n$ at most
- if G is connected, then $|E| > |V|-1$
- $\lg |E| \in \Theta(\lg |V|)$ for connected undirected graphs

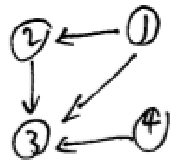
<u>Graph representation</u>

① Adjacency Matrix of $G=(V,E)$
where $V=\{1,2,\cdots,n\}$ is the $n \times n$ matrix of A
given by

$$A[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

- for weighted graph. 1 will be replaced w/ weight

e.g.



A

	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

storage for adjacency matrix : $\Theta(V^2)$

good for dense representation

↳ such as a complete graph

② adjacency list of $v \in V$ is the list adj[v] of vertices adjacent to v

e.g. Adj[1] = {2, 3}

2 = {3}

3 = { }

4 = {3}

good for sparse graph

↳ such as trees, internet nodes,

Q:/ what's the length of the list.

$$|Adj[v]| = \begin{cases} \text{degree}(v) & \text{for undirected} \\ \text{out-degree}(v) & \text{for directed} \end{cases}$$

handshaking Lemma (undirected graph)

$$\sum_{v \in V} \text{degree}(v) = 2|E|$$

Thus for undirected graph, adjacency list cost $\Theta(|V|+|E|)$ storage. same for digraphs

Adjacency list usually out-performs adjacency matrix but the matrix is used some of the very important algorithms due to the easiness of matrix operations.

CS 303, Paul Cao,
Minimum spanning tree (MST)

Why?
Given a set of points (cities), how do you use the minimum amount of wire to connect these points?

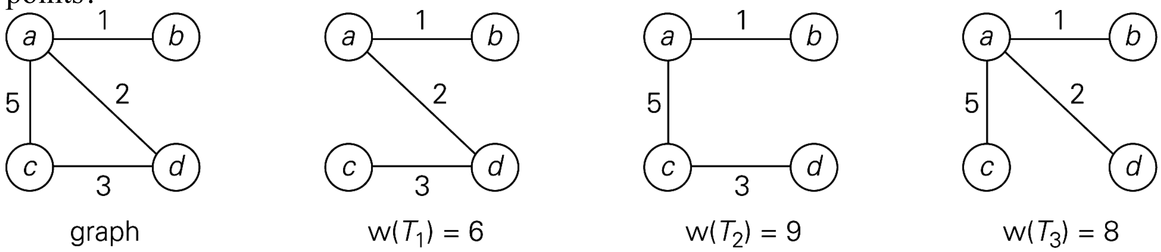


FIGURE 9.1 Graph and its spanning trees; T_1 is the minimum spanning tree

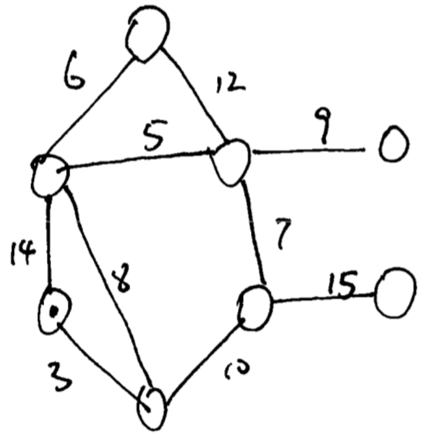
Definitions
<u>Spanning tree</u> of a connected graph *G*: a connected acyclic subgraph of *G* that includes all of *G*'s vertices

<u>Minimum spanning tree</u> of a weighted, connected graph *G*: a spanning tree of *G* of minimum total weight

Problem:
Input: Connected undirected graph G=(V,E) with weight function N:E → R
For simplicity, all weights are distinct (i.e. the function is one to one)
Output: A spanning tree T(connects all vertices) of minimum weight

$$W(T) = \sum_{(u,v)\in T} w(u,v)$$

There are many applications of MST on clustering, and point connecting problem
e.g.



edges have to be in

- include 9 & 15
- 3 has to be there because the dot node has two choices to be connected.
- 6 is there for a similar reason
- 5, 7, 8 have to be there because we have 4 connected components to be connected.

Prim's algorithm

Idea: build the MST by adding in the **nearest** node into the tree in each iteration. (the greedy idea comes from the nearest)

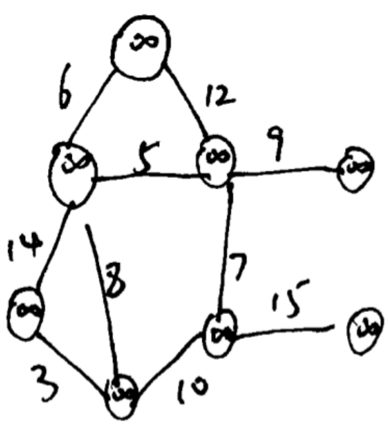
Procedure

- Start with tree T_1 consisting of one (any) vertex and "grow" tree one vertex at a time to produce MST through a series of expanding subtrees $T_1, T_2, \ldots, T_n$
- On each iteration, construct T_{i+1} from T_i by adding vertex not in T_i that is closest to those already in T_i (this is a "greedy" step!)
- Stop when all vertices are included

Pseudo-code

```
PRIM(G, w, r)
  Q = Ø
  for each u ∈ G.V
      u.key = ∞
      u.π = NIL
      INSERT(Q, u)
  DECREASE-KEY(Q, r, 0)        // r.key = 0
  while Q ≠ Ø
      u = EXTRACT-MIN(Q)
      for each v ∈ G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = u
              DECREASE-KEY(Q, v, w(u, v))
```

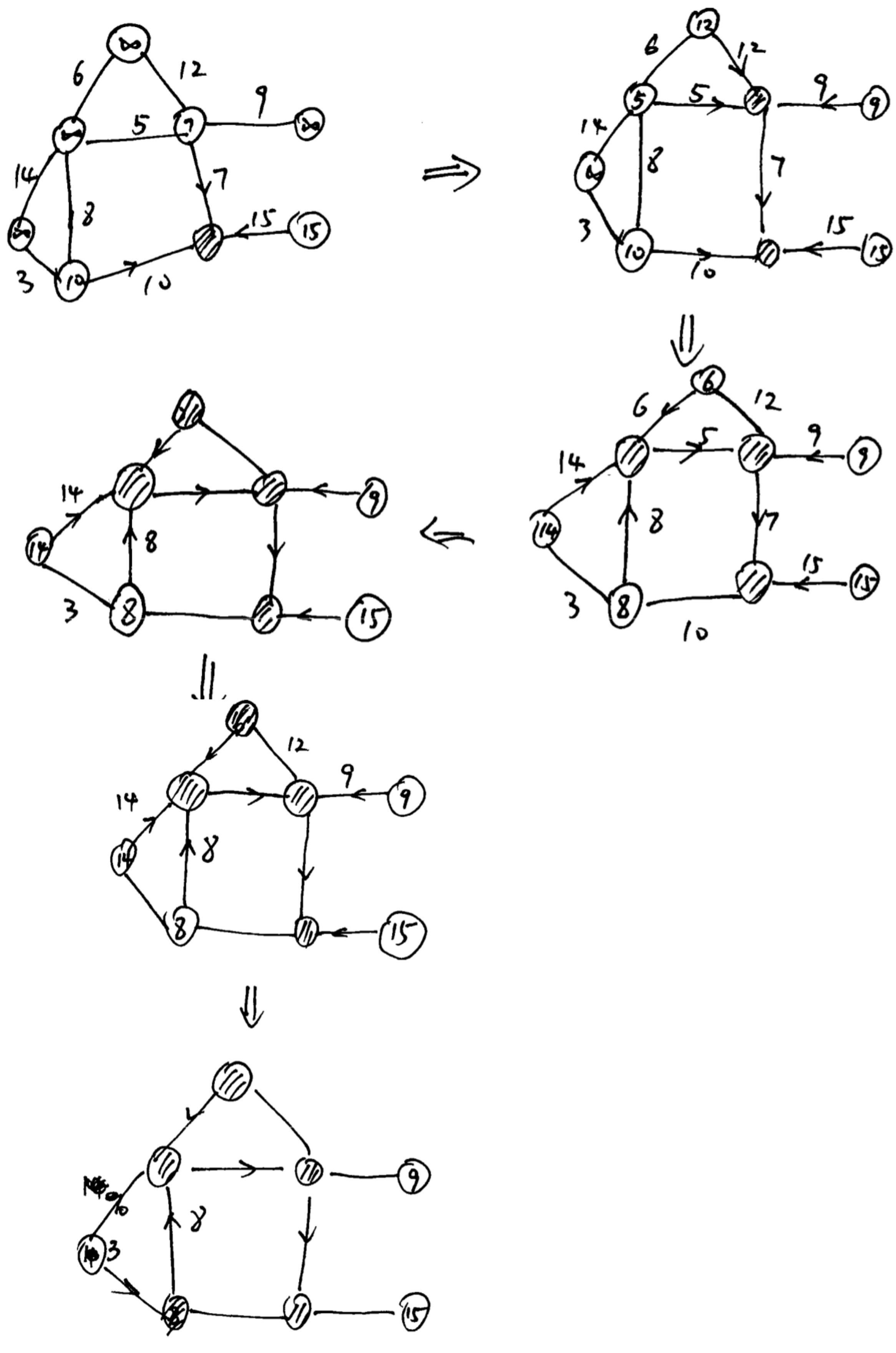
E.g. Find the MST on the example graph above



: already part of MST

: in Q

→ pointer



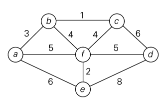
Then just add 9 and 15 to the tree

Notes

- Proof by induction that this construction actually yields MST
- Needs priority queue for locating closest fringe vertex

- Efficiency
    - $O(n^2)$ for weight matrix representation of graph and array implementation of priority queue
    - $O(m \log n)$ for adjacency list representation of graph with n vertices and m edges and min-heap implementation of priority queue

Exercise



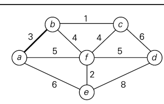
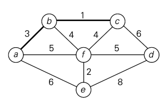
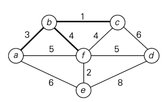
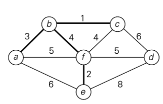
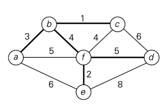
Tree vertices	Remaining vertices	Illustration
a(−, −)	**b(a, 3)** c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(−, ∞) e(a, 6) f(b, 4)	
c(b, 1)	d(c, 6) e(a, 6) **f(b, 4)**	
f(b, 4)	d(f, 5) **e(f, 2)**	
e(f, 2)	**d(f, 5)**	
d(f, 5)		

FIGURE 9.2 Application of Prim's algorithm. The parenthesized labels of a vertex in the middle column indicate the nearest tree vertex and edge weight; selected vertices and edges are shown in bold.

Kruskal's algorithm

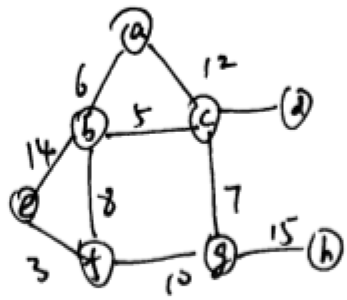
Idea: Since MST is a tree and we can build this tree, why not order the edges and build up the tree with edges that don't make circles.

Steps

- Sort the edges in nondecreasing order of lengths
- "Grow" tree one edge at a time to produce MST through a series of expanding forests F_1, F_2, …, F_{n-1}

- On each iteration, add the next edge on the sorted list unless this would create a cycle. (If it would, skip the edge.)

e.g. same graph

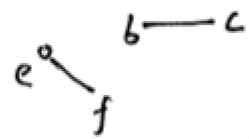


Sort by edge: ef bc ab cg bf cd be gh

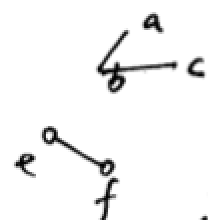
grow by edge:

ef ✓

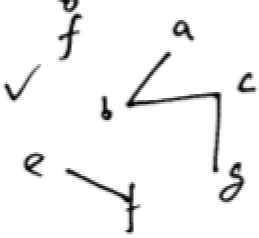
ef bc ✓



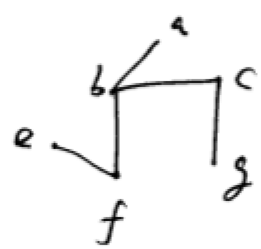
ef bc ab ✓



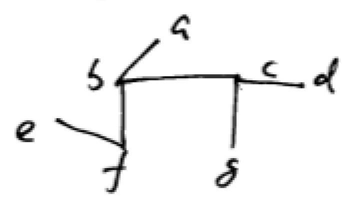
ef bc ab cg ✓



ef bc ab cg bf ✓

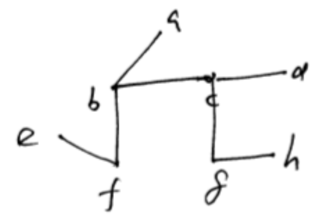


ef bc ab cg bf cd ✓



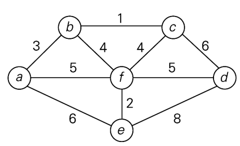
ef bc ab cg bf cd ~~be~~ X
↳ will introduce a cycle

ef bc ab cg bf cd gh ✓



done

Exercise



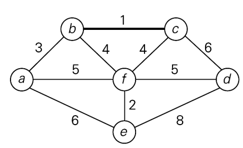
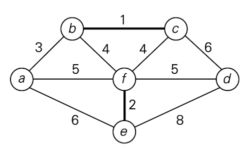
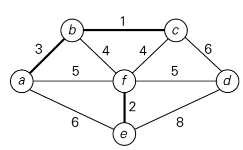
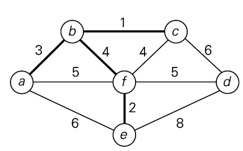
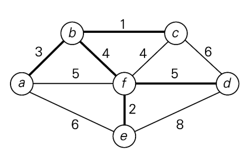
Tree edges	Sorted list of edges	Illustration
	bc ef ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bc 1	bc **ef** ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ef 2	bc ef **ab** bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ab 3	bc ef ab **bf** cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bf 4	bc ef ab bf cf af **df** ae cd de 1 2 3 4 4 5 5 6 6 8	
df 5		

FIGURE 9.4 Application of Kruskal's algorithm. Selected edges are shown in bold.