アルゴリズムの設計と解析

教授：　黄　潤和　（W4022）
rhuang@hosei.ac.jp

SA：　広野　史明　（A4/A8）
fumiaki.hirono.5k@stu.hosei.ac.jp

Contents (L13 – MST algorithm)

- Shortest path
- Minimal spanning tree
- Prim’s algorithm
- Kruskal’s algorithm

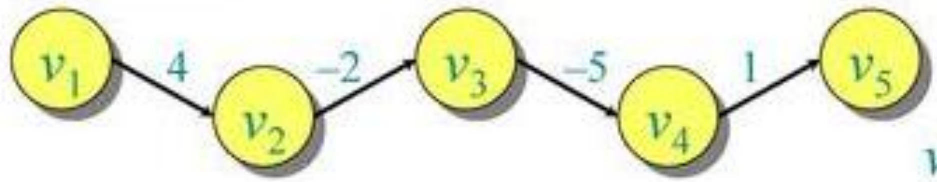
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Weighted Graph

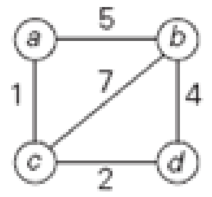
- Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

Example:



$w(p) = -2$



(a)

$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & \infty & 5 & 1 & \infty \\ b & 5 & \infty & 7 & 4 \\ c & 1 & 7 & \infty & 2 \\ d & \infty & 4 & 2 & \infty \end{array}$$

(b)

a	→ *b*, 5 → *c*, 1
b	→ *a*, 5 → *c*, 7 → *d*, 4
c	→ *a*, 1 → *b*, 7 → *d*, 2
d	→ *b*, 4 → *c*, 2

(c)

FIGURE 1.8 (a) Weighted graph. (b) Its weight matrix. (c) Its adjacency lists.

Shortest Path

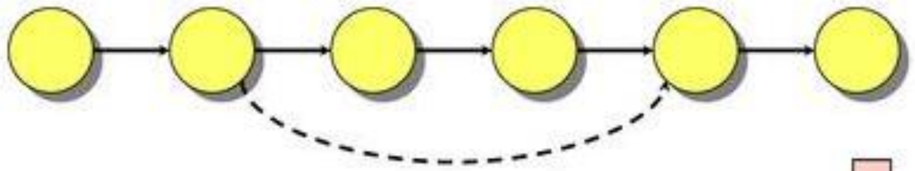
A ***shortest path*** from u to v is a path of minimum weight from u to v. The ***shortest-path weight*** from u to v is defined as

$$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}.$$

Note: $\delta(u, v) = \infty$ if no path from u to v exists.

Theorem. A subpath of a shortest path is a shortest path.

Proof. Cut and paste:



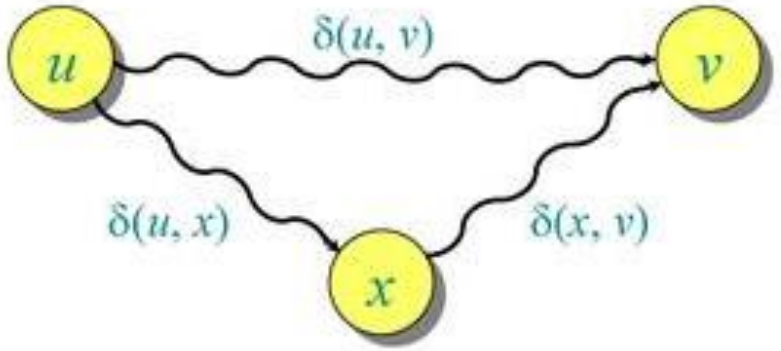
□

Triangle inequality

Theorem. For all $u, v, x \in V$, we have

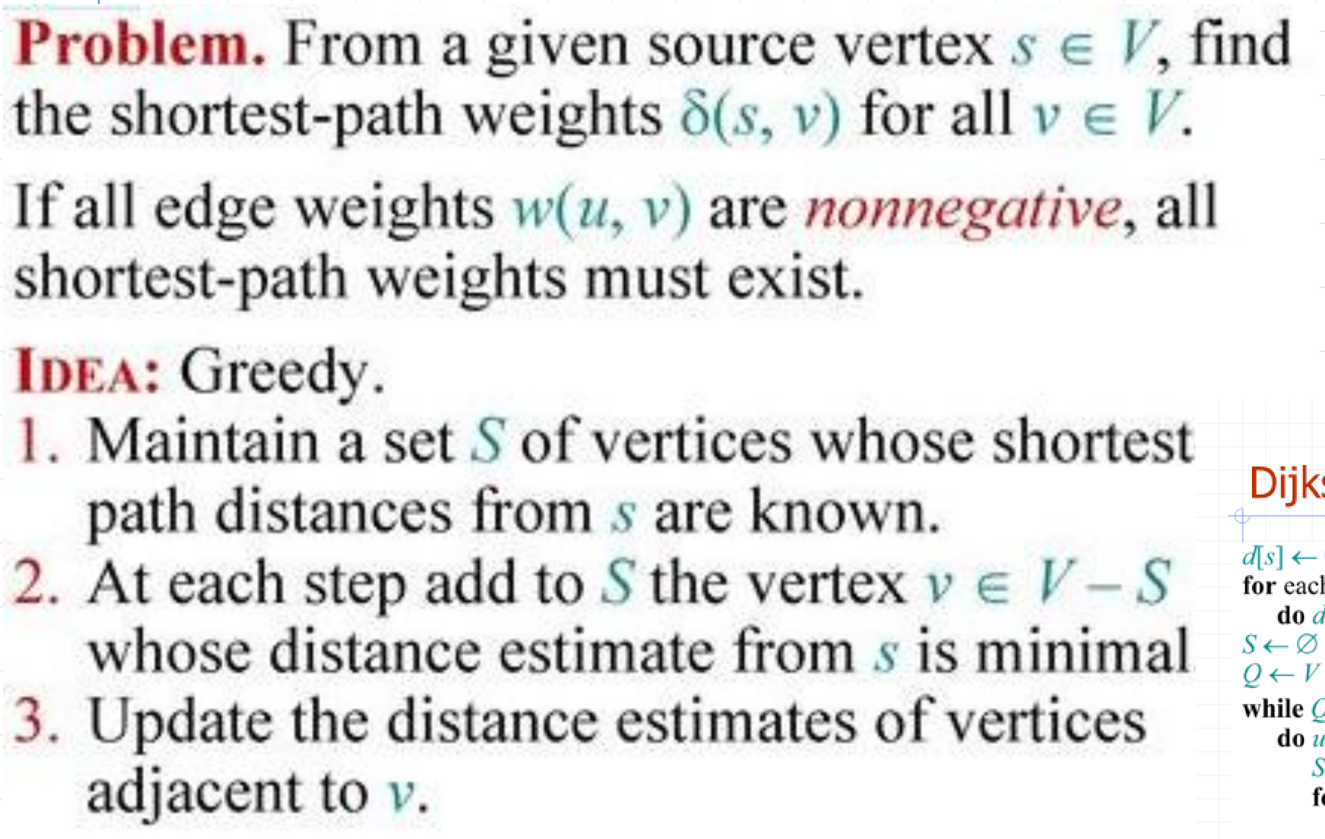
$$\delta(u, v) \le \delta(u, x) + \delta(x, v).$$

Proof.



□

Single source shortest paths

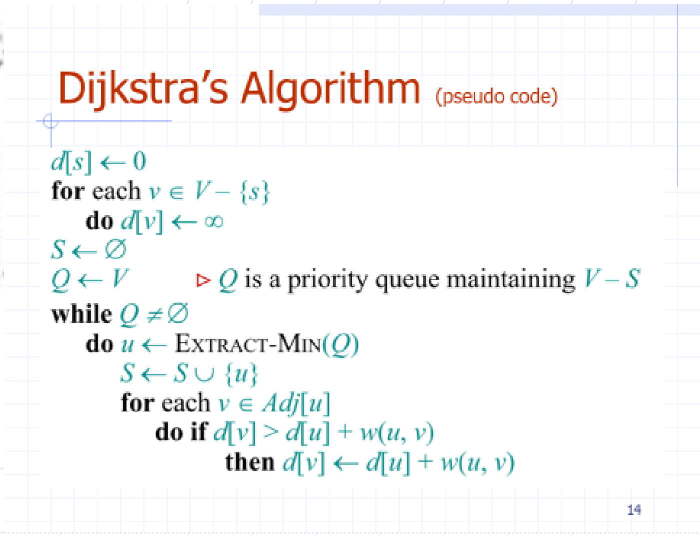


Problem. From a given source vertex $s \in V$, find the shortest-path weights $\delta(s, v)$ for all $v \in V$.

If all edge weights $w(u, v)$ are *nonnegative*, all shortest-path weights must exist.

IDEA: Greedy.

1. Maintain a set S of vertices whose shortest path distances from s are known.
2. At each step add to S the vertex $v \in V - S$ whose distance estimate from s is minimal
3. Update the distance estimates of vertices adjacent to v.



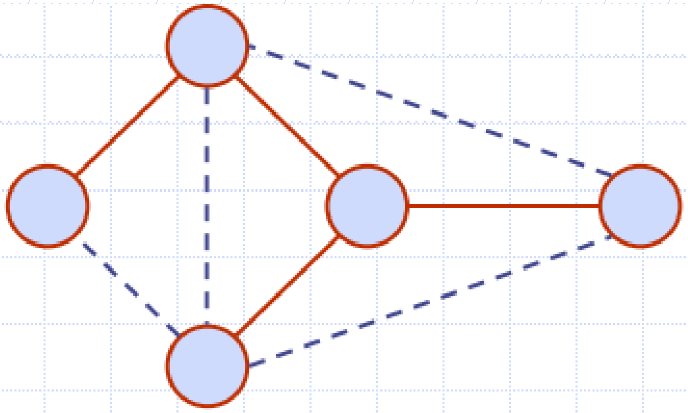
Dijkstra's Algorithm (pseudo code)

```
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$          $\triangleright$ $Q$ is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN($Q$)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$
```

14

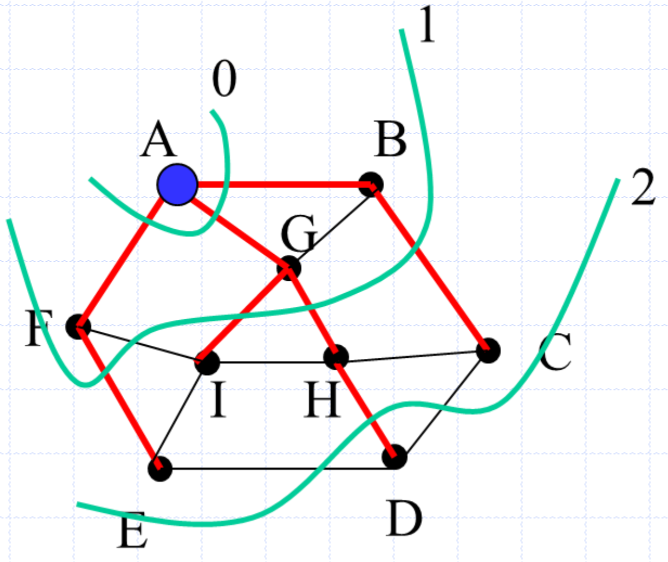
What are spanning trees?

A spanning tree of a connected graph is a spanning subgraph that is a tree

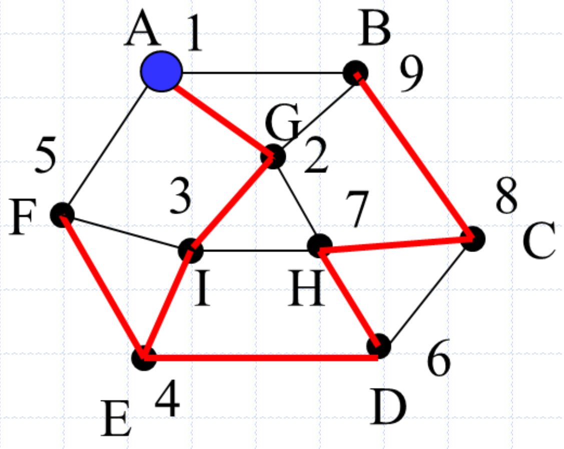


Spanning tree

Results from BFS and DFS?



Breadth-first Spanning Tree



Depth-first spanning tree

Note: A spanning tree is not unique unless the graph is a tree
グラフが木でない限り全域木は1つではない。

⇨ Q.
Exist a minimal spanning three?

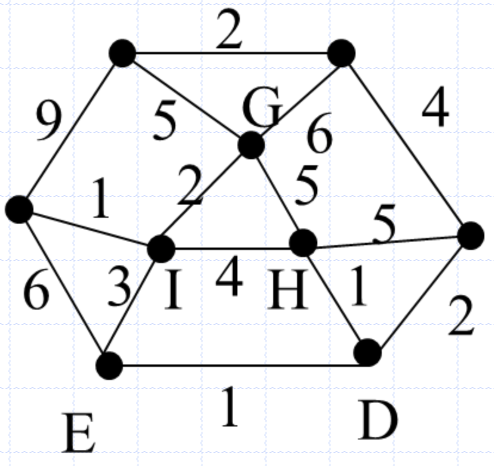
Q.

(1) What is a mimumal spanning tree?

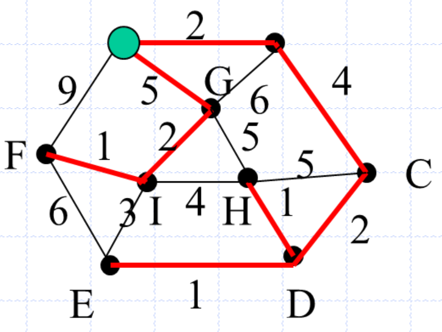
(2) Is the single source shortest path (Dijkstra algorithm) a minimal spanning?

What is a minimal spanning tree?

- Weighted spanning tree
- Weight of spanning tree = sum of tree edge weights
- Any spanning tree whose weight is minimal

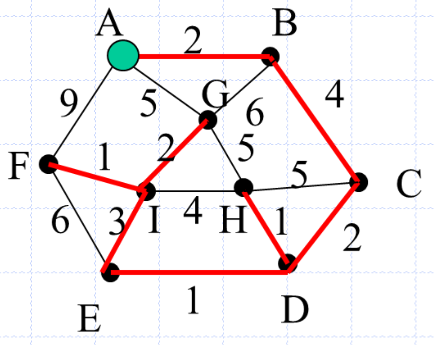


Graph



5+2+2+1+4+2+1+1
=18

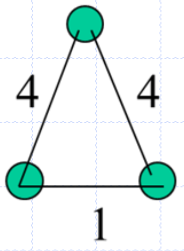
SSSP tree Single source shortest path tree

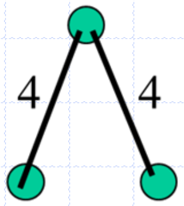


2+4+2+1+1+3+2+1
=16

Minimal spanning tree

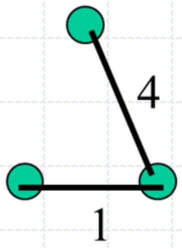
Caution: in general, SSSP tree is not MST





SSSP Tree

Spanning tree cost =8

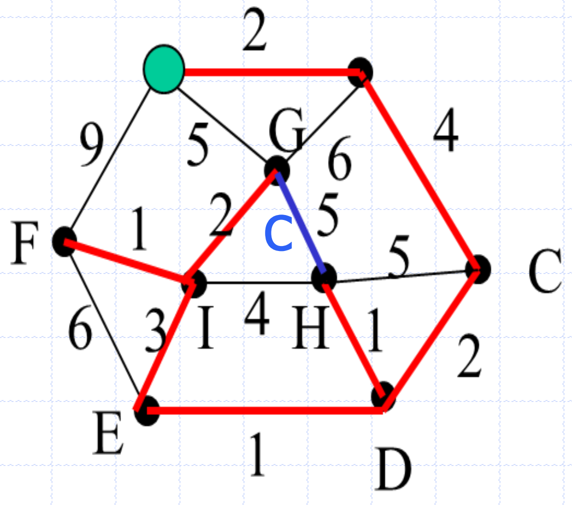


MSP

Spanning tree cost =5

- Intuition:
  - SSSP: fixed start node
  - MST: at any point in construction, we have a bunch of nodes that we have reached, and we look at the shortest distance from any one of those nodes to a new node

Property of MSTs



- For any edge: c (e.g. **c**=(G,H)(5) in blue color) in G but not in T (e.g. in red), there is a simple cycle Y ((H,D), (D,E), (E,I), (I,G)) containing only edge **c** and edges in spanning tree
- Moreover, weight of **c** must be greater than or equal to weight of any edge in this cycle.

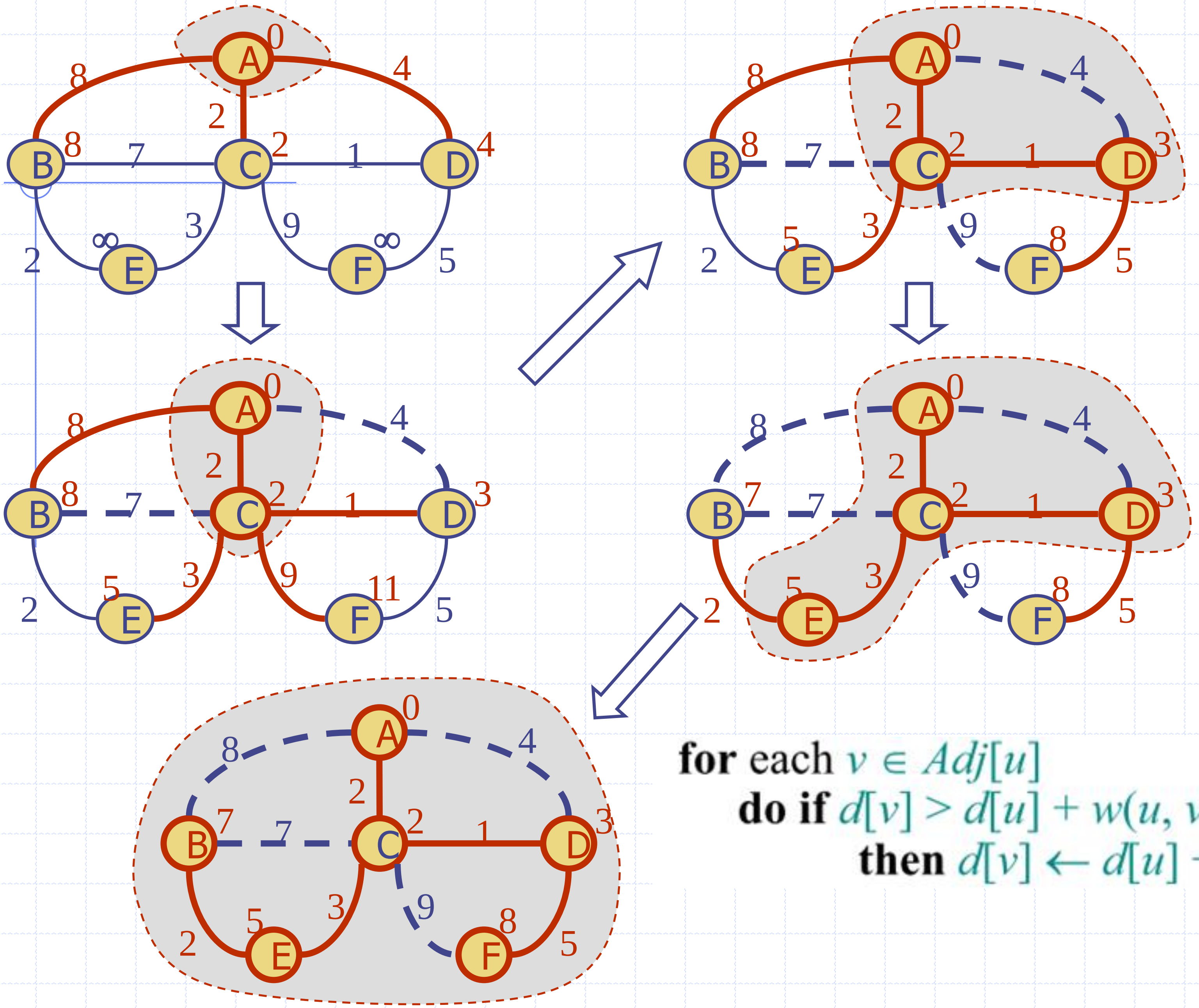
From SSSP to MST

From Dijkstra’s to Prim’s algorithms

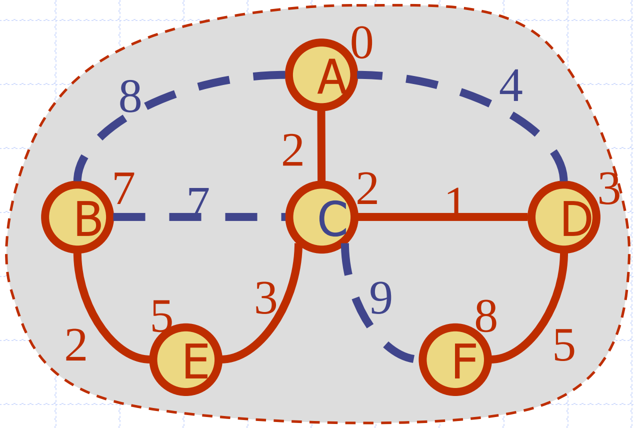
- Change Dijkstra’s algorithm so the priority of bridge (f→n) is length(f,n) rather than minDistance(f) + length(f,n)
- In other words:
  - starts with any node
  - keeps adding smallest edge to expand its path

Dijkstra's Algorithm (pseudo code)

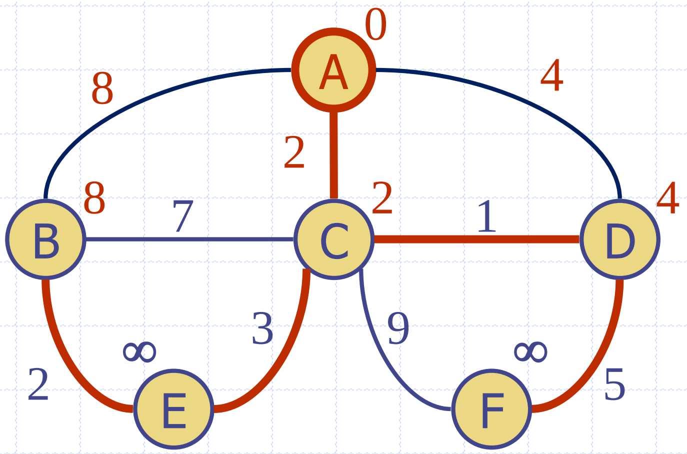
```
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$
$Q \leftarrow V$            ▷ $Q$ is a priority queue maintaining $V - S$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN($Q$)
       $S \leftarrow S \cup \{u\}$
       for each $v \in Adj[u]$
           do if $d[v] > d[u] + w(u, v)$
                 then $d[v] \leftarrow d[u] + w(u, v)$
```



```
for each $v \in Adj[u]$
    do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$
```

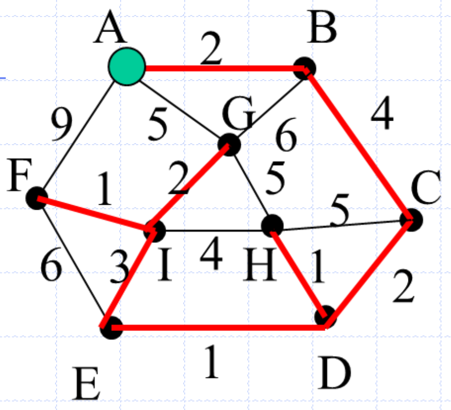


SSSP
cost = 13

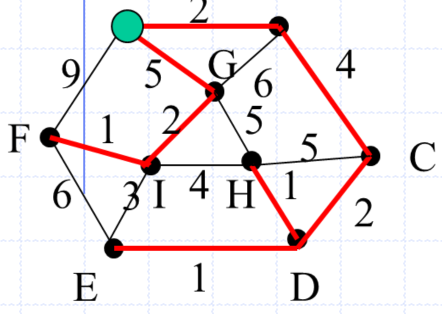


MST
cost = 13

An example



MST



SSSP tree

[((dummy→A), 0)]

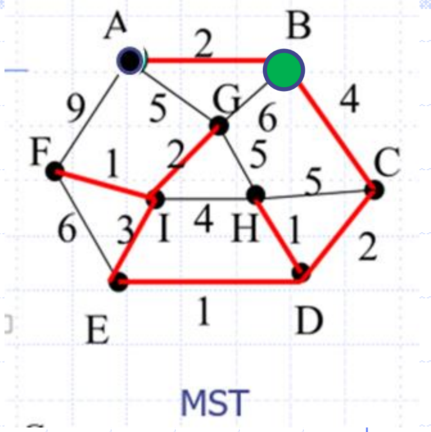
[] add (dummy→A) to MST
[((A→B),2), ((A→G),5),((A→F),9)]

[((A→G),5),((A→F),9)] add (A→B) to MST
[((A→G),5),((A→F),9), (B→G),6),((B→C),4)]

[((A→G),5),((A→F),9),((B→G),6)]
add (B→C) to MST
[((A→G),5),((A→F),9),((B→G),6),((C,H),5),
((C,D), 2)]

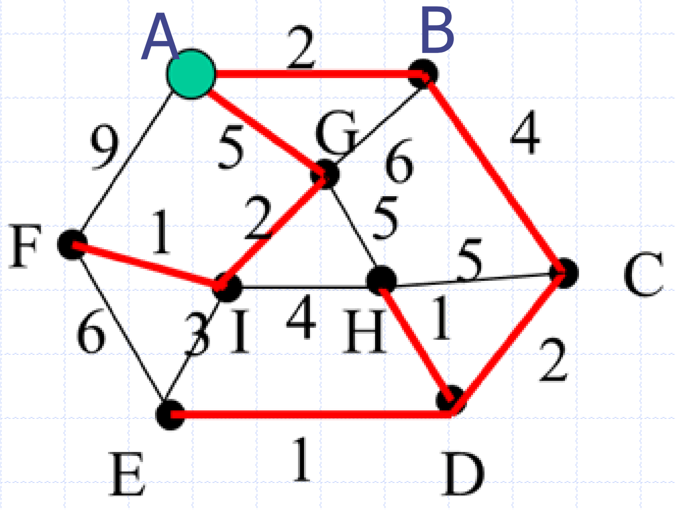
………..

Work in class:
Start from B,
(1) Draw a SSSP (Single source shortest path)
(2) Draw a MST (Minimal spanning tree)?



Dijkstra algorithm from B

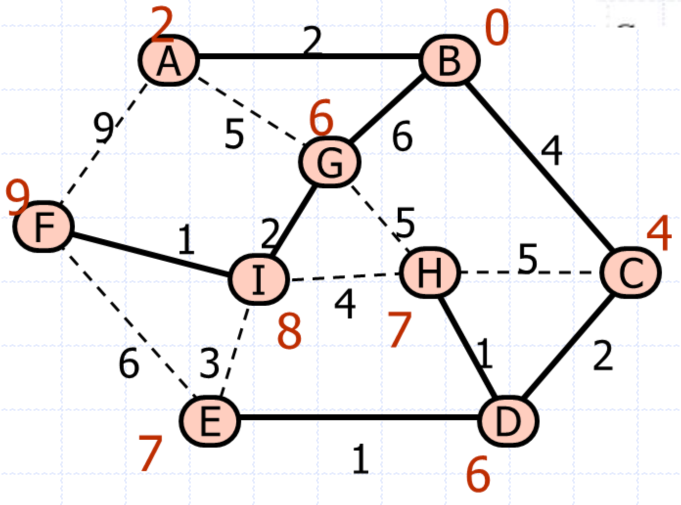
Start from A



SSSP tree

Cost = 18

Start from B



SSSP tree
Cost = 19

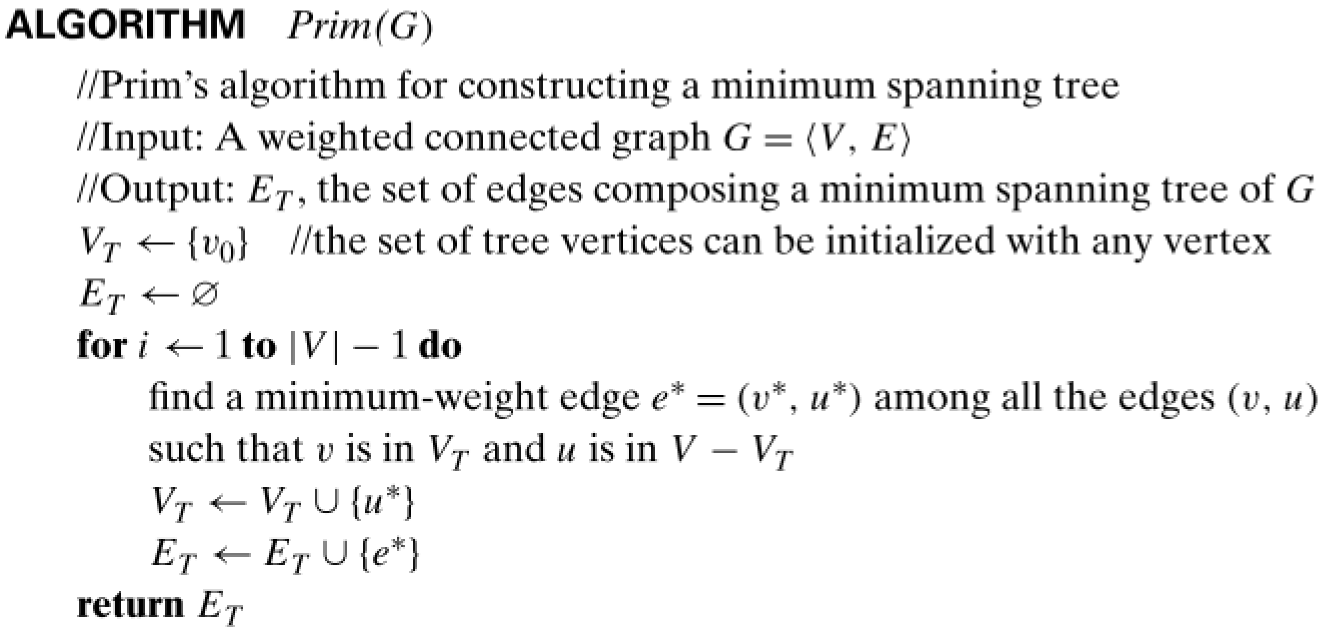
Results are different

MST cost = 16

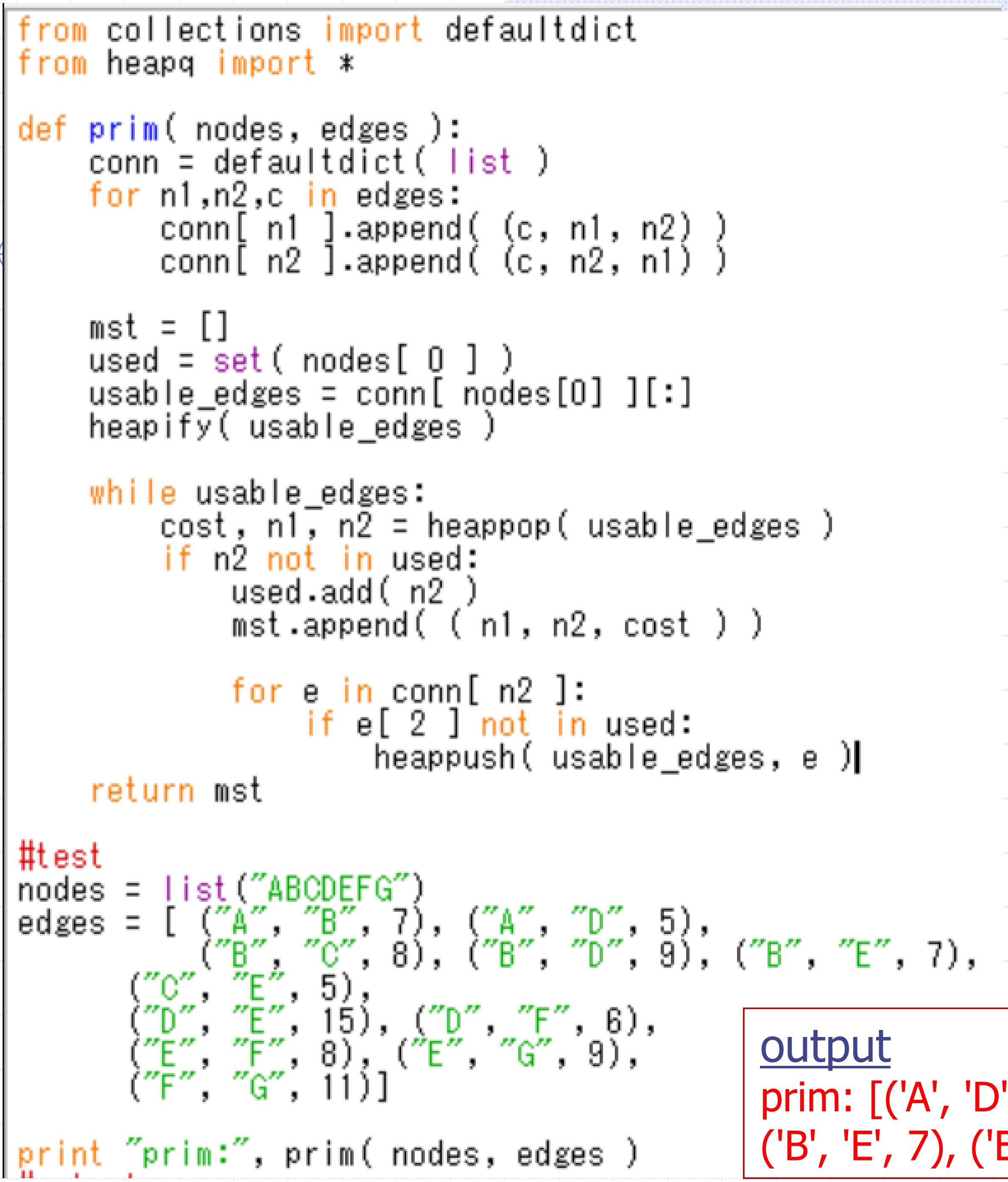
Prim’s algorithm

```
{
    T = $\phi$;
    U = { };
    while (U$\neq$V)  {
        let (u, v) be the lowest cost edge
        such that u$\in$U and v$\in$V - U;
        T = T $\cup${(u, v)}
        U = U $\cup${v}
      }
}
```

Complexity = O(|E|log(|E|))



```
ALGORITHM  Prim(G)
    //Prim’s algorithm for constructing a minimum spanning tree
    //Input: A weighted connected graph $G = \langle V, E \rangle$
    //Output: $E_T$, the set of edges composing a minimum spanning tree of $G$
    $V_T \leftarrow \{v_0\}$   //the set of tree vertices can be initialized with any vertex
    $E_T \leftarrow \varnothing$
    for $i \leftarrow 1$ to $|V| - 1$ do
        find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges $(v, u)$
        such that $v$ is in $V_T$ and $u$ is in $V - V_T$
        $V_T \leftarrow V_T \cup \{u^*\}$
        $E_T \leftarrow E_T \cup \{e^*\}$
    return $E_T$
```

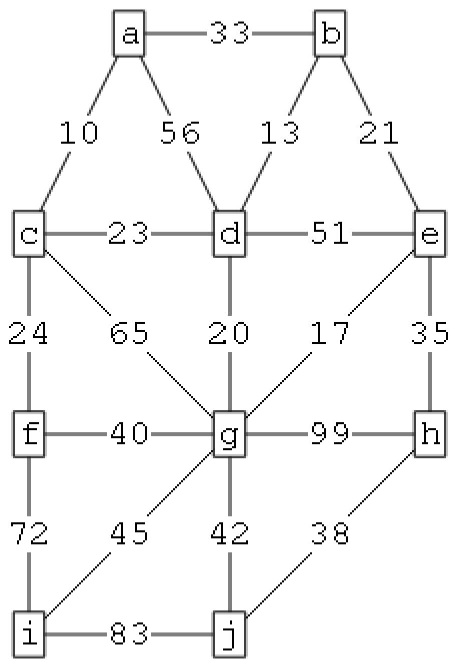


```
from collections import defaultdict
from heapq import *

def prim( nodes, edges ):
    conn = defaultdict( list )
    for n1,n2,c in edges:
        conn[ n1 ].append( (c, n1, n2) )
        conn[ n2 ].append( (c, n2, n1) )

    mst = []
    used = set( nodes[ 0 ] )
    usable_edges = conn[ nodes[0] ][:]
    heapify( usable_edges )

    while usable_edges:
        cost, n1, n2 = heappop( usable_edges )
        if n2 not in used:
            used.add( n2 )
            mst.append( ( n1, n2, cost ) )

            for e in conn[ n2 ]:
                if e[ 2 ] not in used:
                    heappush( usable_edges, e )
    return mst

#test
nodes = list("ABCDEFG")
edges = [ ("A", "B", 7), ("A", "D", 5),
          ("B", "C", 8), ("B", "D", 9), ("B", "E", 7),
      ("C", "E", 5),
      ("D", "E", 15), ("D", "F", 6),
      ("E", "F", 8), ("E", "G", 9),
      ("F", "G", 11)]

print "prim:", prim( nodes, edges )
```

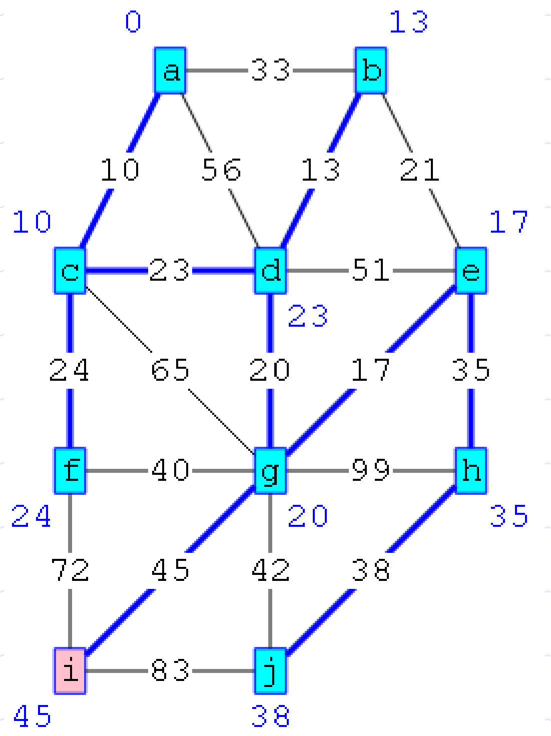
output

prim: [('A', 'D', 5), ('D', 'F', 6), ('A', 'B', 7), ('B', 'E', 7), ('E', 'C', 5), ('E', 'G', 9)]

Work in class: draw the middle steps

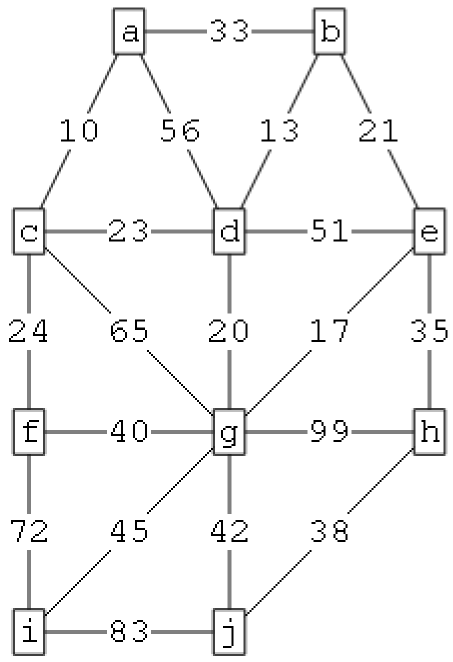


(a)

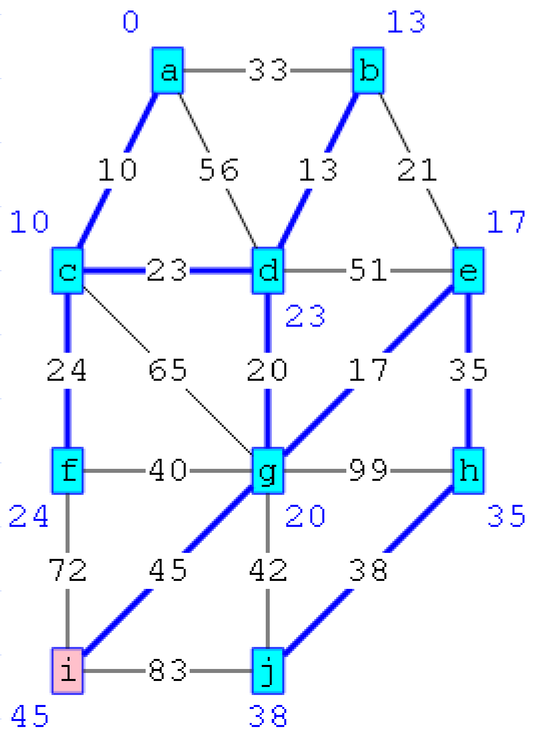


(b)

Work in class: draw the middle steps

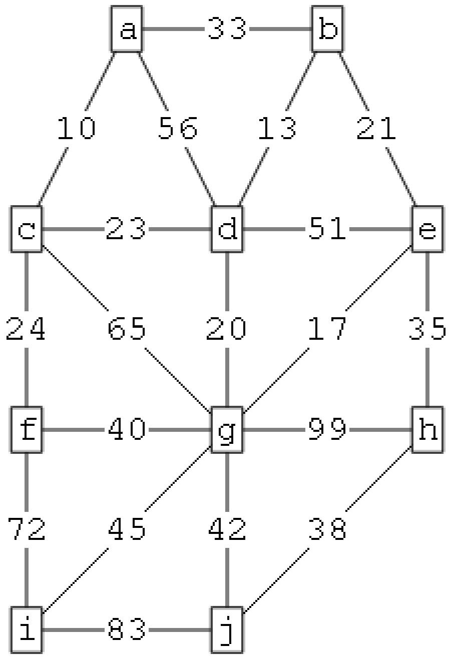


(a)

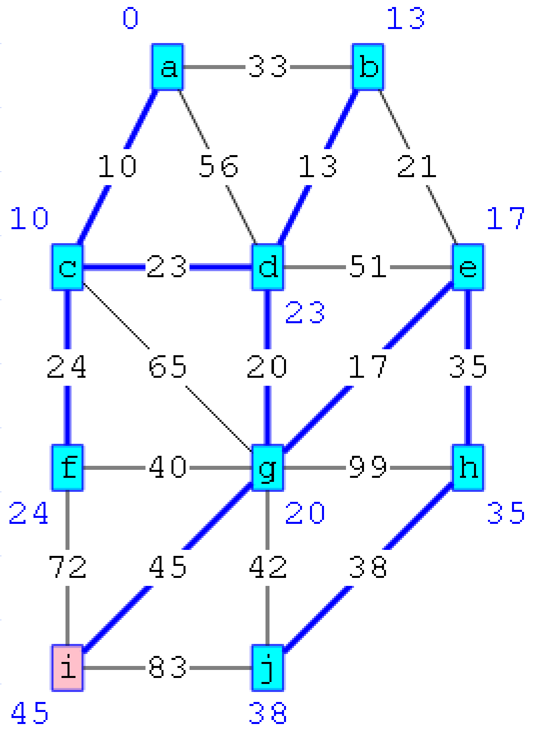


(b)

http://weierstrass.is.tokushima-u.ac.jp/ikeda/suuri/dijkstra/PrimApp.shtml?demo3

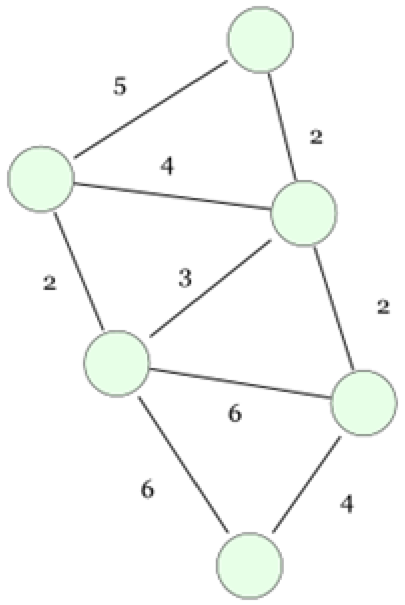


(a)



(b)

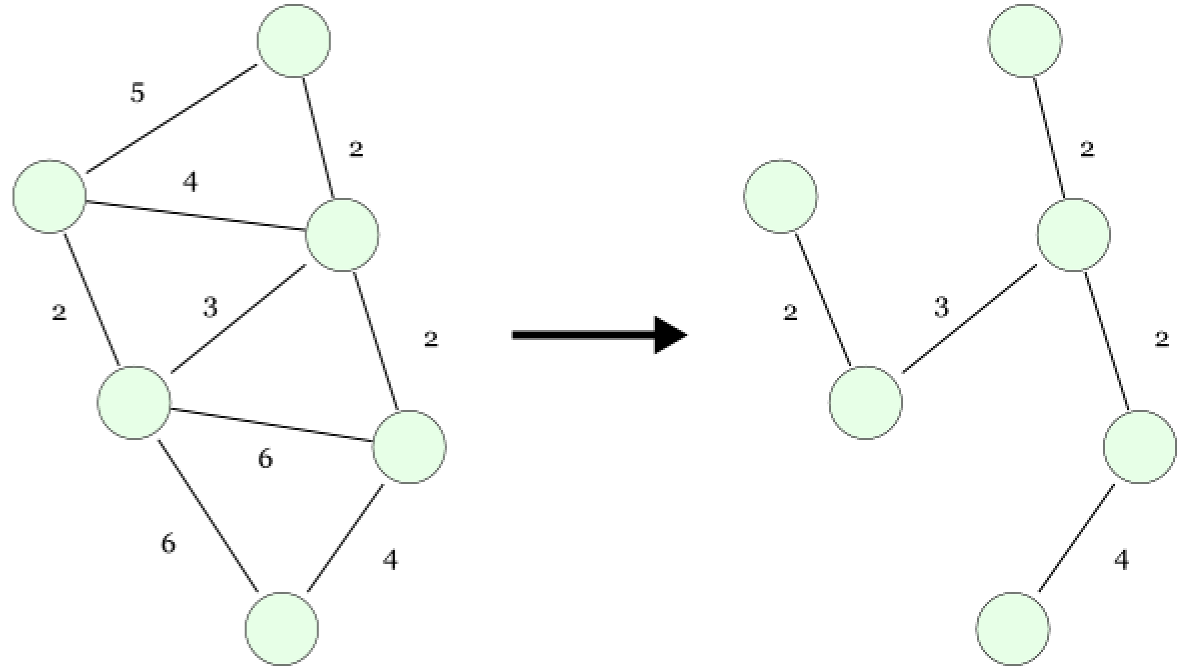
Work in class (prim's algorithm)



⇨

```
{
    T = $\phi$;
    U = { };
    while (U≠V)  {
        let (u, v) be the lowest cost edge
        such that u∈U and v∈V - U;
        T = T ∪{(u, v)}
        U = U ∪{v}
    }
}
```

http://www.deqnotes.net/acmicpc/prim/



https://www.cs.usfca.edu/~galles/visualization/Prim.html

Prim’s algorithm animation examples

方法まとめ:

1. 初期化: X = {a}, Y = V − X とする (ここでノード a は何でも良い), エッジのコストの総和を0とする
2. Y が空になるまで以下のループを繰り返す:
    1. 「Xの要素 → Yの要素」を繋ぐエッジのうち, 最小コストを持つエッジ e を探す (この実装方法は後で詳述)
    2. X = X + {eのY側の要素}, Y = Y − {eのY側の要素} とする (実際にはノードの確定フラグを立てればよい)
    3. エッジのコストの総和にeのコストを加算

アルゴリズムの実装方法によって、計算量は

素朴な方法

まず, 素朴な方法 (naive method) です。XとYを繋ぐ最小コストのエッジを見つける必要があるたびに (|V|回のループ), Xの要素から伸びているエッジを毎回全て調べ, その中で最小コストのエッジを見つけることができます。ループの内側では, これは最大でも毎回全エッジを調べれば済むので, $O(|V|^2)$ もしくは $O(|E|)$ になります。これが |V| 回ループで回るので, 全体の計算量は $O(|V|^3)$ もしくは $O(|V|\cdot|E|)$ になります。

ノード数が100であれば $|V|^3$ = 1,000,000 = 1M で解けそうですが, ノード数が1000になると $|V|^3$ = 1,000,000,000 = 1G で解けなくなります。

優先度付き待ち行列

上記の方法でもたいていの場合は十分高速ですが，ダイクストラ法と同様に，プリム法でも優先度付き待ち行列（priority queue）を使うことでさらなる高速化が可能です。

実装も比較的簡単で，まず最初に初期化フェーズでノード a を選択後，a から伸びているエッジを全て優先度待ち行列に追加します。優先度付き待ち行列はエッジのコストでソートされるようにしておきます（内部構造にヒープ（heap）を使うと追加は O(log n) になります）。次に，優先度付き待ち行列から取り出した（popした）エッジの接続先ノードを，現在のMSTに追加します（pop も O(log n) で可能です）。そして，そのノードから伸びているエッジをまた全て優先度付き待ち行列に追加していきます。なお，pop したエッジの接続先がすでにMSTに追加済みなら気を取り直してもう一度popし直します。後はこれを繰り返すだけで動きます。

この場合，ループ回数はすなわち優先度付き待ち行列に挿入される回数なので，上のアルゴリズムではエッジは全て各2回ずつ挿入されるので |E| になります。ループの内側では，挿入 = O(log n) とpop = O(log n) だけなので，二つの操作を足して O(log n) になります。ここで n は待ち行列の長さですが，最大 |E| を超えることはないので，O(log |E|) とみなせます。結局，全体としては計算量は O(|E|･log |E|) となります。

Demonstration

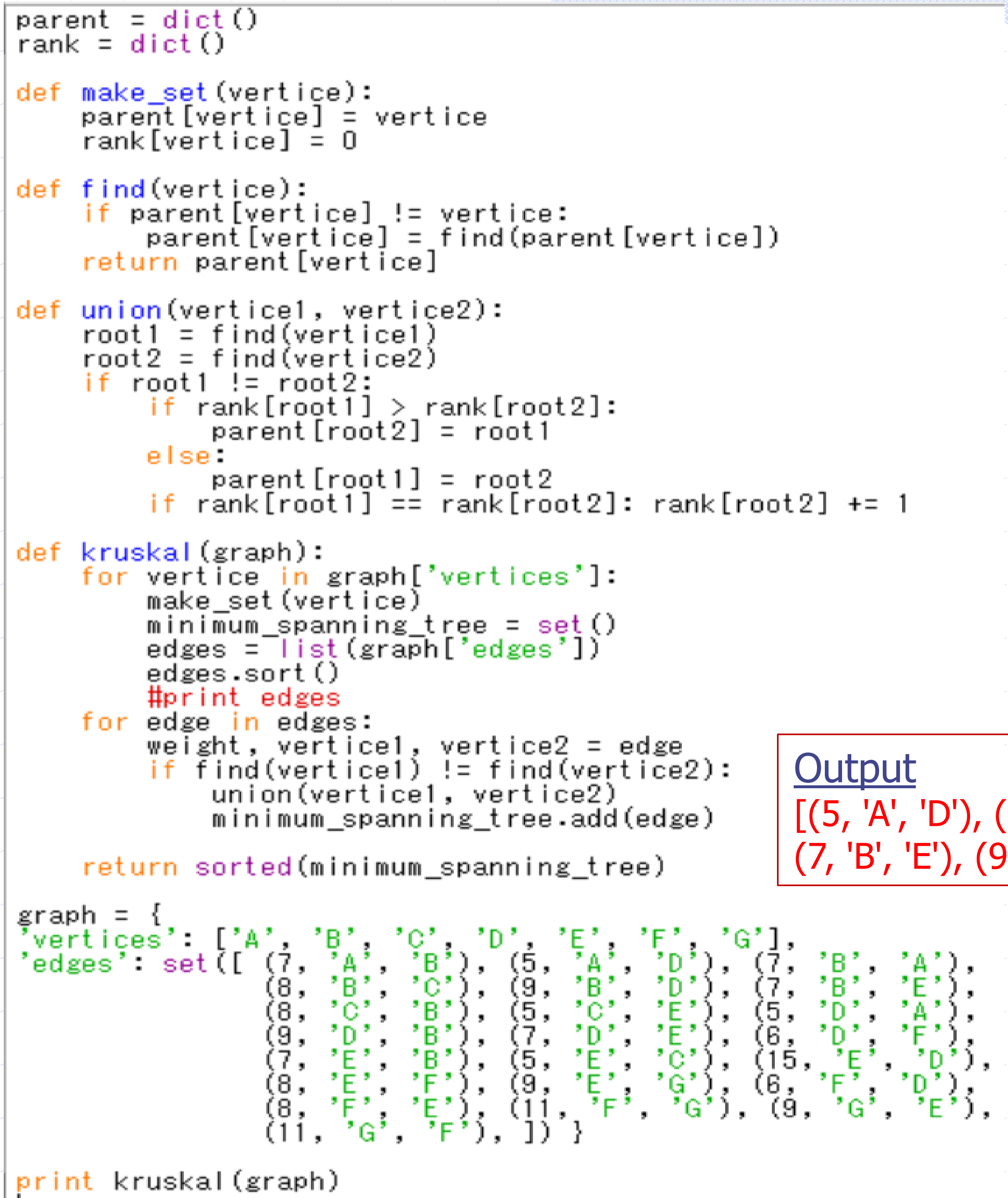
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Algorithms in Java

http://weierstrass.is.tokushima-u.ac.jp/ikeda/suuri/dijkstra/Prim.shtml

Kruskal's algorithm (basic part)

```
(Sort the edges in an increasing order)
A:={}
**while** E is not empty **do {**
    take an edge (u, v) that is shortest in E
    and delete it from E
    if  u and v are in different components
    then
        add (u, v) to A
}
```

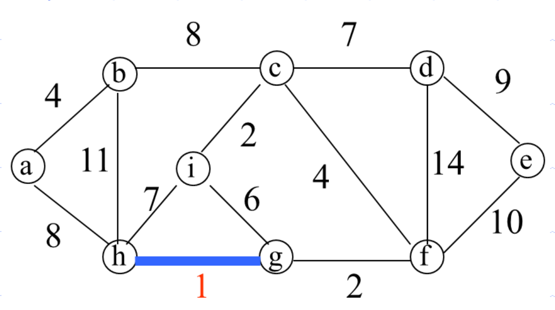


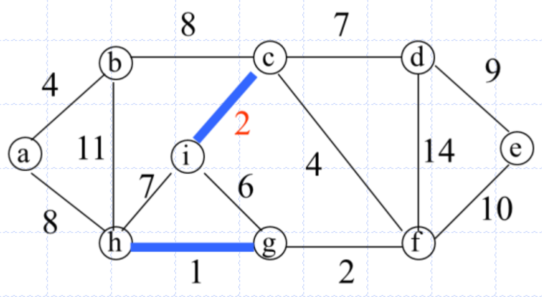
```
parent = dict()
rank = dict()

def make_set(vertice):
    parent[vertice] = vertice
    rank[vertice] = 0

def find(vertice):
    if parent[vertice] != vertice:
        parent[vertice] = find(parent[vertice])
    return parent[vertice]

def union(vertice1, vertice2):
    root1 = find(vertice1)
    root2 = find(vertice2)
    if root1 != root2:
        if rank[root1] > rank[root2]:
            parent[root2] = root1
        else:
            parent[root1] = root2
        if rank[root1] == rank[root2]: rank[root2] += 1

def kruskal(graph):
    for vertice in graph['vertices']:
        make_set(vertice)
        minimum_spanning_tree = set()
        edges = list(graph['edges'])
        edges.sort()
        #print edges
    for edge in edges:
        weight, vertice1, vertice2 = edge
        if find(vertice1) != find(vertice2):
            union(vertice1, vertice2)
            minimum_spanning_tree.add(edge)

    return sorted(minimum_spanning_tree)

graph = {
'vertices': ['A', 'B', 'C', 'D', 'E', 'F', 'G'],
'edges': set([ (7, 'A', 'B'), (5, 'A', 'D'), (7, 'B', 'A'),
               (8, 'B', 'C'), (9, 'B', 'D'), (7, 'B', 'E'),
               (8, 'C', 'B'), (5, 'C', 'E'), (5, 'D', 'A'),
               (9, 'D', 'B'), (7, 'D', 'E'), (6, 'D', 'F'),
               (7, 'E', 'B'), (5, 'E', 'C'), (15, 'E', 'D'),
               (8, 'E', 'F'), (9, 'E', 'G'), (6, 'F', 'D'),
               (8, 'F', 'E'), (11, 'F', 'G'), (9, 'G', 'E'),
               (11, 'G', 'F'), ]) }

print kruskal(graph)
```

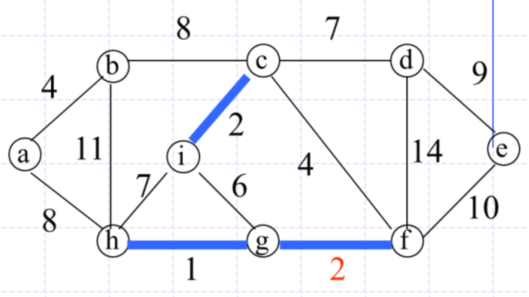
Output

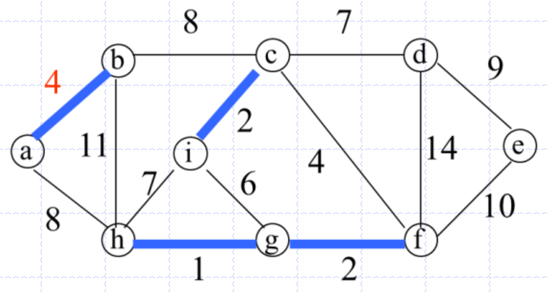
```
[(5, 'A', 'D'), (5, 'C', 'E'), (6, 'D', 'F'), (7, 'A', 'B'),
(7, 'B', 'E'), (9, 'E', 'G')]
```

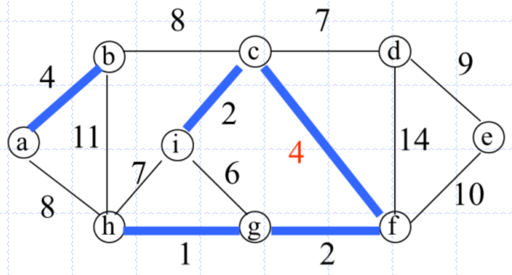
An example

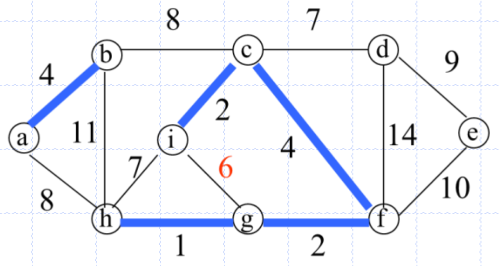

8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2


8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

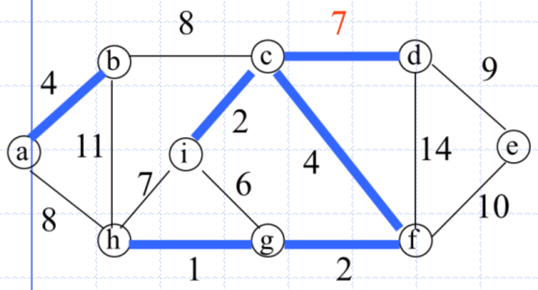

8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

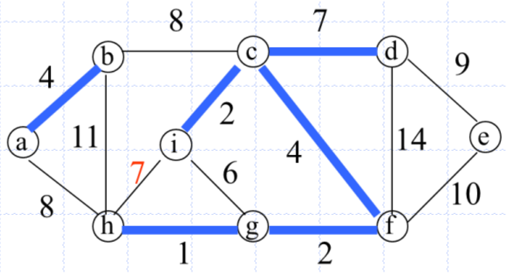

8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

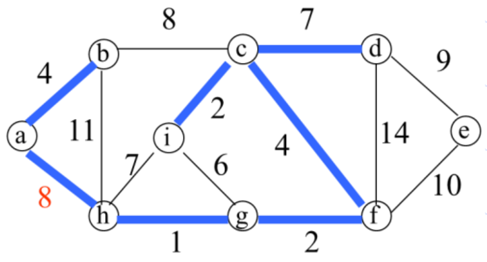

8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

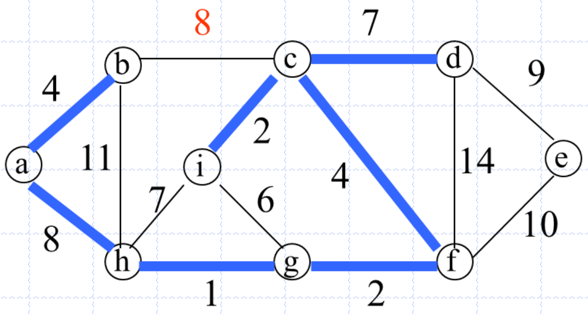

8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

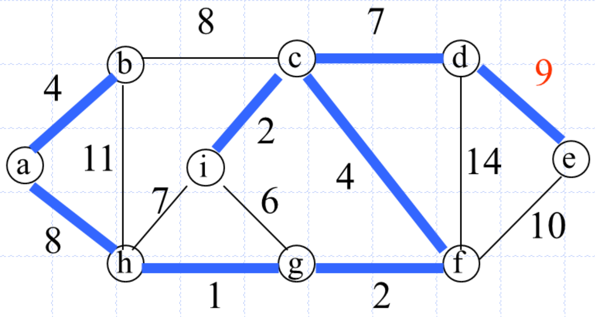
An example







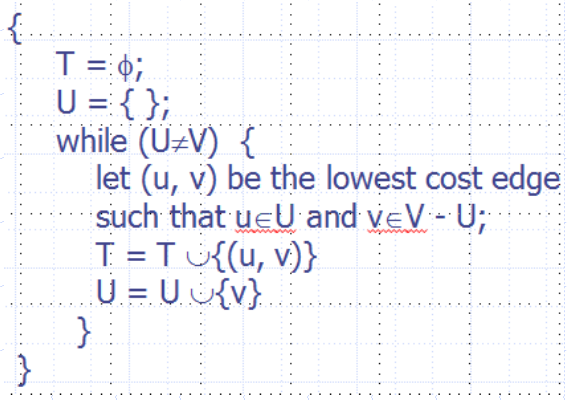




Kruskal’s v. s. Prim’s algorithms

- *In Kruskal's algorithm,*
  - *The set A is a forest.*
  - *The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.*
- *In Prim's algorithm,*
  - *The set A forms a single tree.*
  - *The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.*

```
A:={}
while E is not empty do {
   take an edge (u, v) that is shortest in E
   and delete it from E
   if  u and v are in different components
   then
       add (u, v) to A
}
```



```
{
    T = $\phi$;
    U = { };
    while ($U \neq V$)  {
        let (u, v) be the lowest cost edge
        such that $u \in U$ and $v \in V - U$;
        T = T $\cup$ {(u, v)}
        U = U $\cup$ {v}
    }
}
```

Kruskal's algorithm (disjoint set)

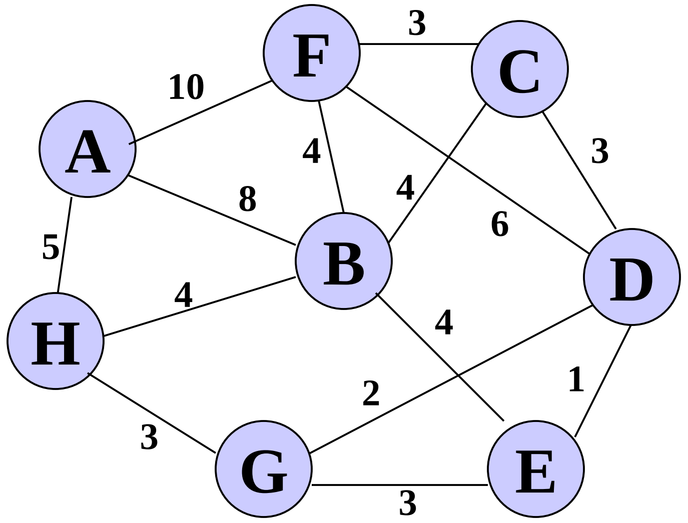
```
MST_KRUSKAL(G,w)
    A:={}
    for each vertex v in V[G]
        do MAKE_SET(v)
    sort the edges of E by non-decreasing weight w
    for each edge (u,v) in E, in order by non-decreasing weight
        do if FIND_SET(u) != FIND_SET(v)
            then A:=A∪{(u,v)}
                UNION(u,v)
    return A
```

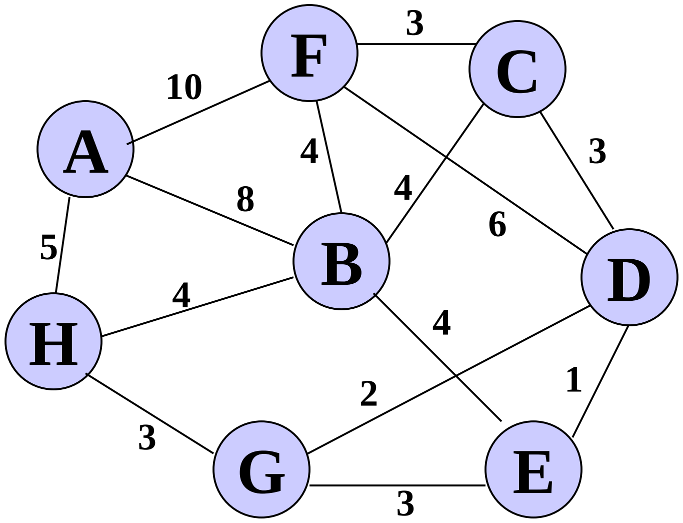
- Two steps:
  - Sort edges by increasing edge weight
  - Select the first |V| – 1 edges that do not generate a cycle

Animation of Kruskal’s algorithm
http://f.hatena.ne.jp/mickey24/20090614234556

Walk-Through

Consider an undirected, weight graph

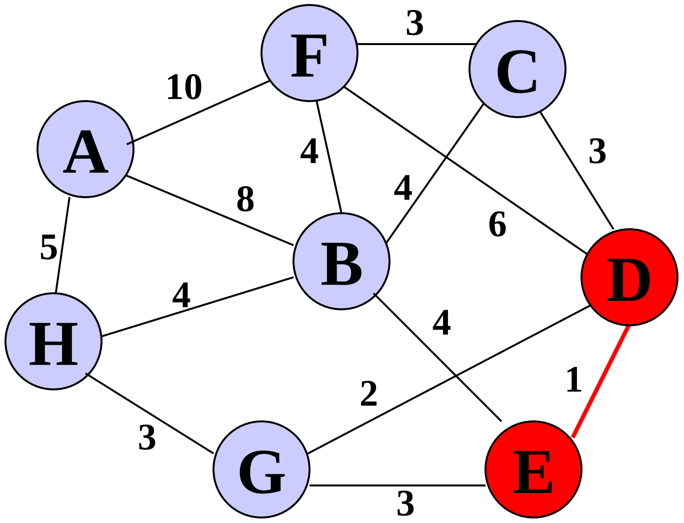




Sort the edges by increasing edge weight

edge	d_v	
(D,E)	1	
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

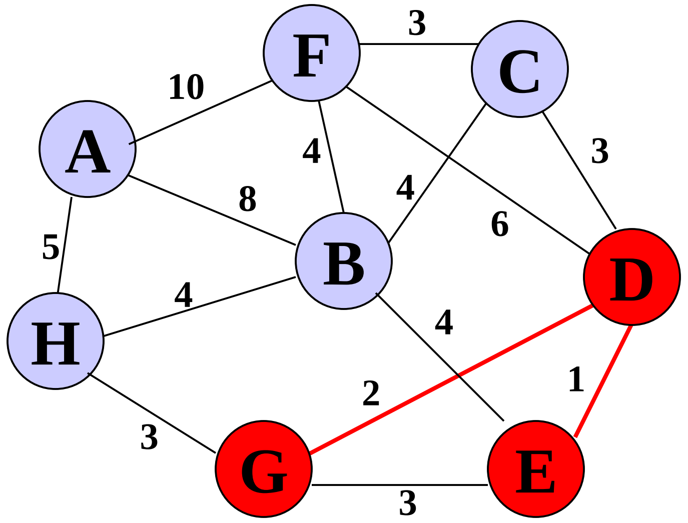
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

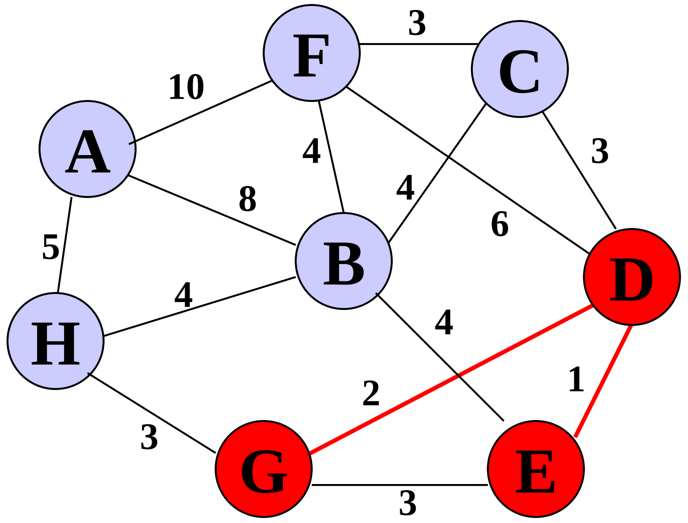
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

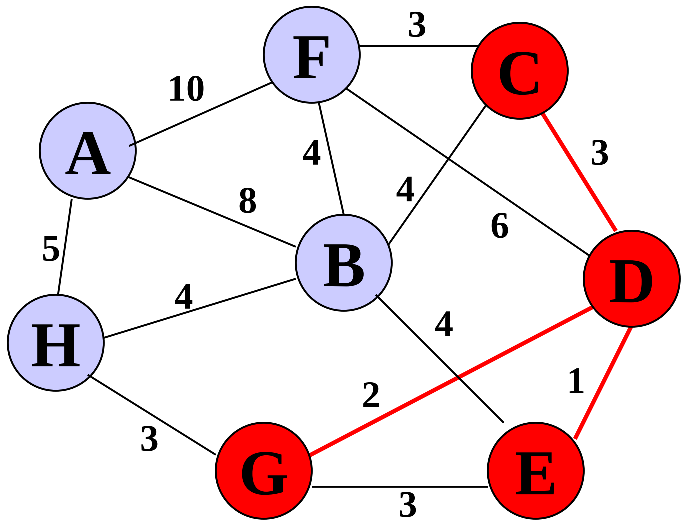


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

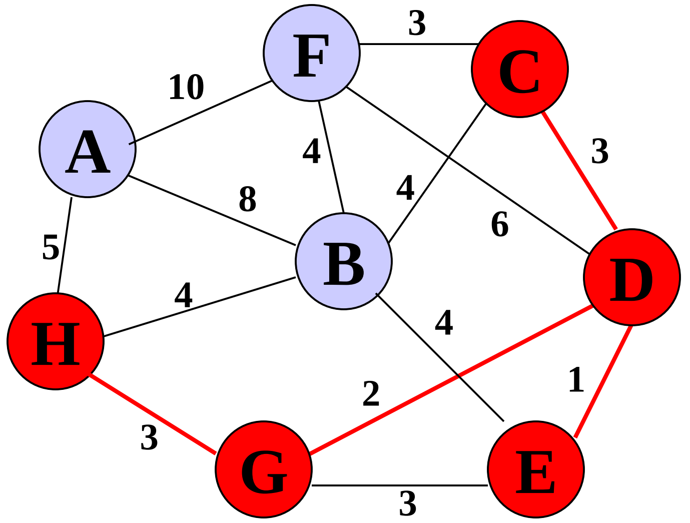
Accepting edge (E,G) would create a cycle



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

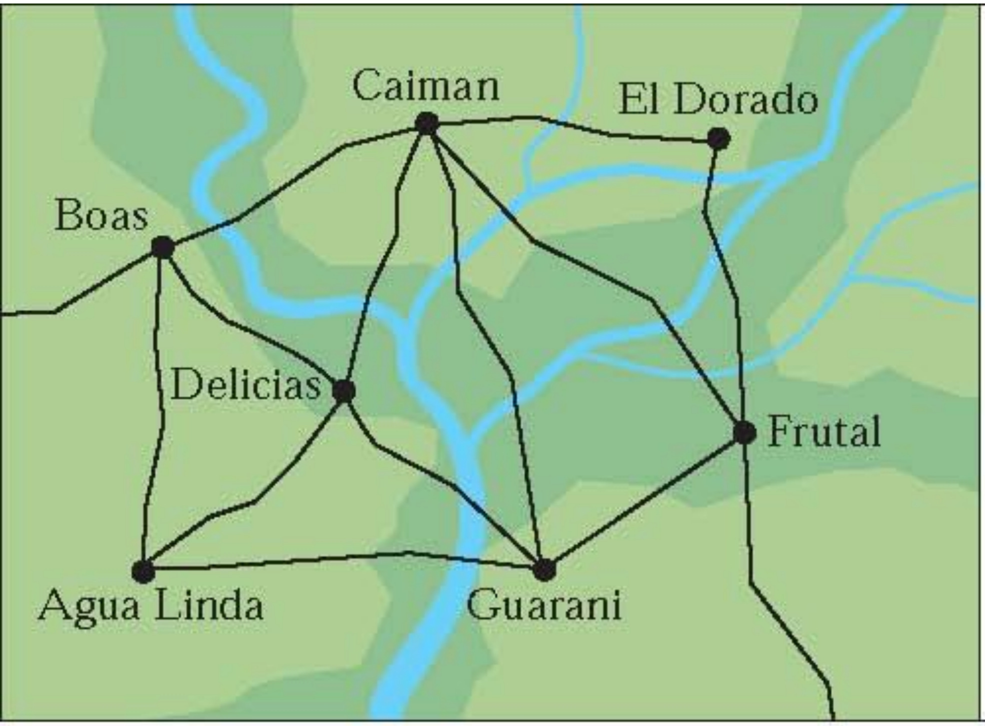
edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

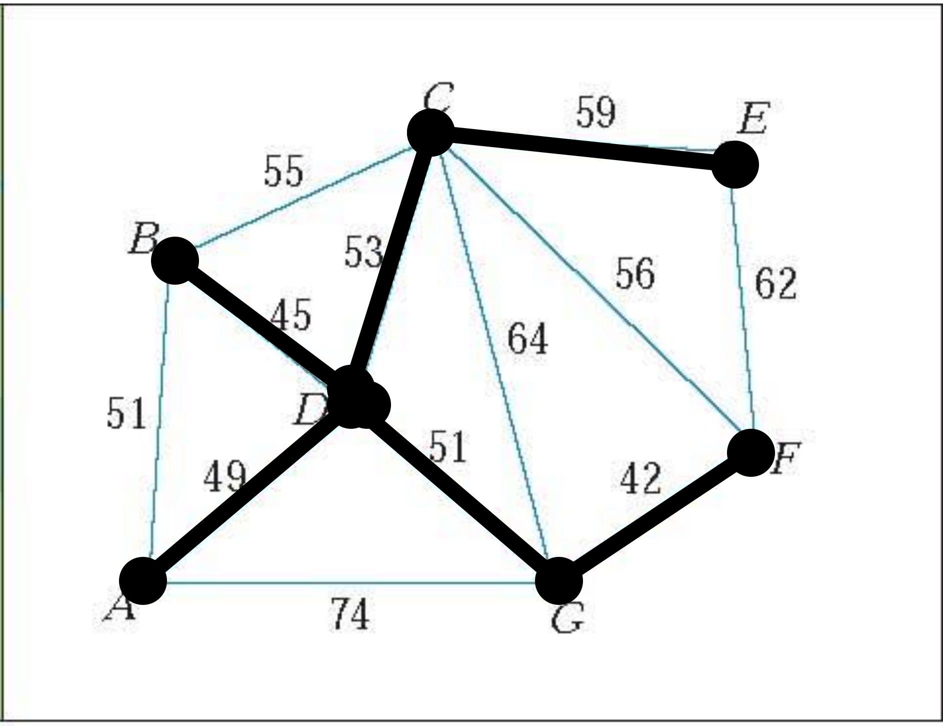
Work in class:

Please continue to finish this algorithm and draw the resulting tree.

Apply Kruskal’s algorithm to find the minimum spanning tree



(a)



(b)

Minimum Connector Algorithms

Kruskal's algorithm

1. Select the shortest edge in a network
2. Select the next shortest edge which does not create a cycle
3. Repeat step 2 until all vertices have been connected

Prim's algorithm

1. Select any vertex
2. Select the shortest edge connected to that vertex
3. Select the shortest edge connected to any vertex already connected
4. Repeat step 3 until all vertices have been connected

Demonstration

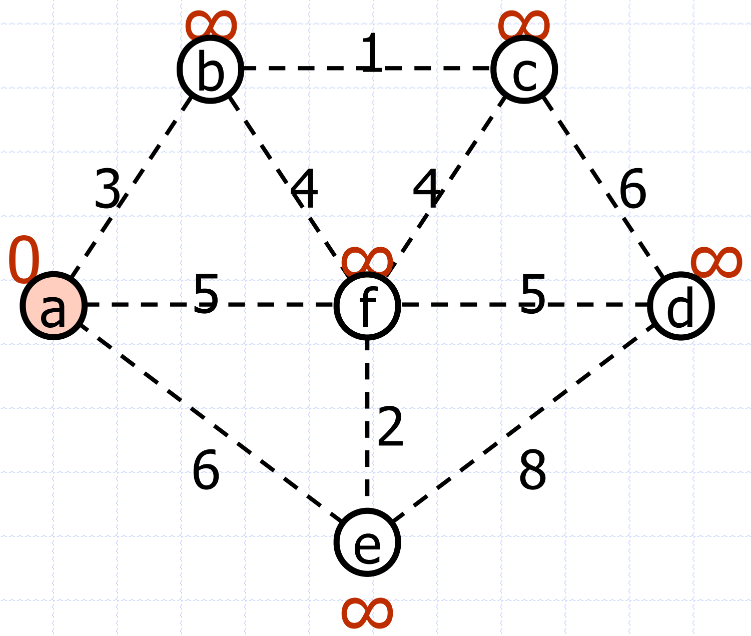
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Algorithms in Java

http://weierstrass.is.tokushima-u.ac.jp/ikeda/suuri/dijkstra/Prim.shtml

Ex. 12-1 Dijkstra algorithm from vertex *a*

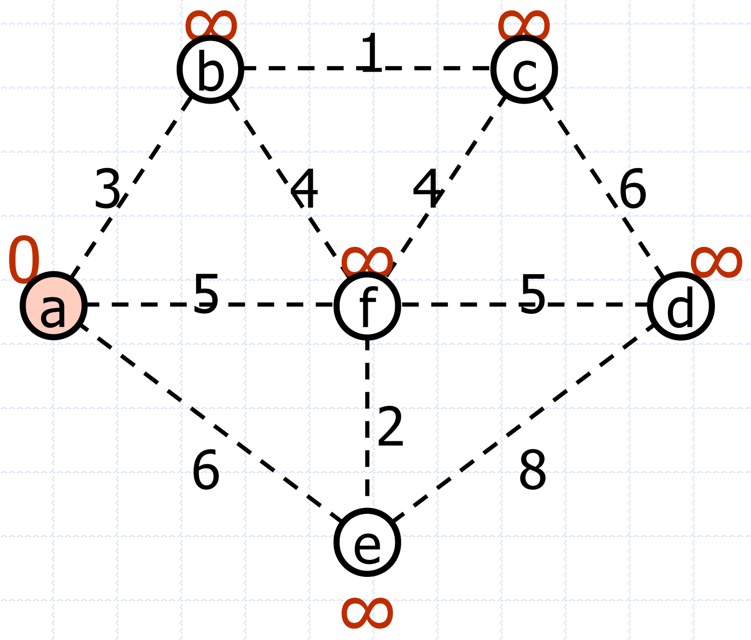
→ Single Source Shortest Path

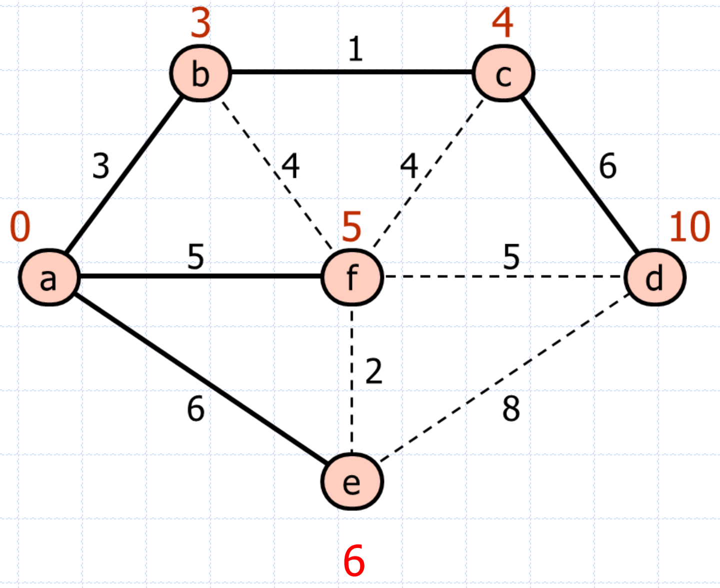


⇨

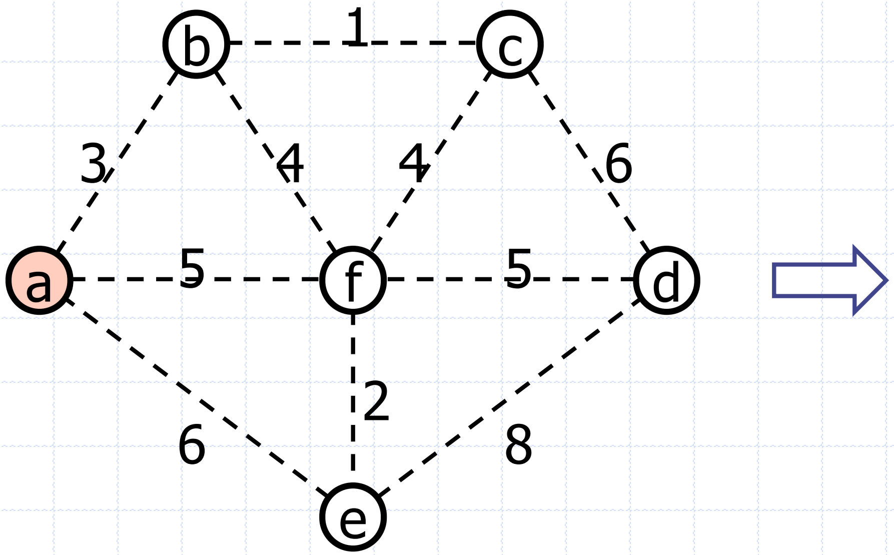
6

Ex. 12-1 Dijkstra algorithm from vertex *a* → Single Source Shortest Path

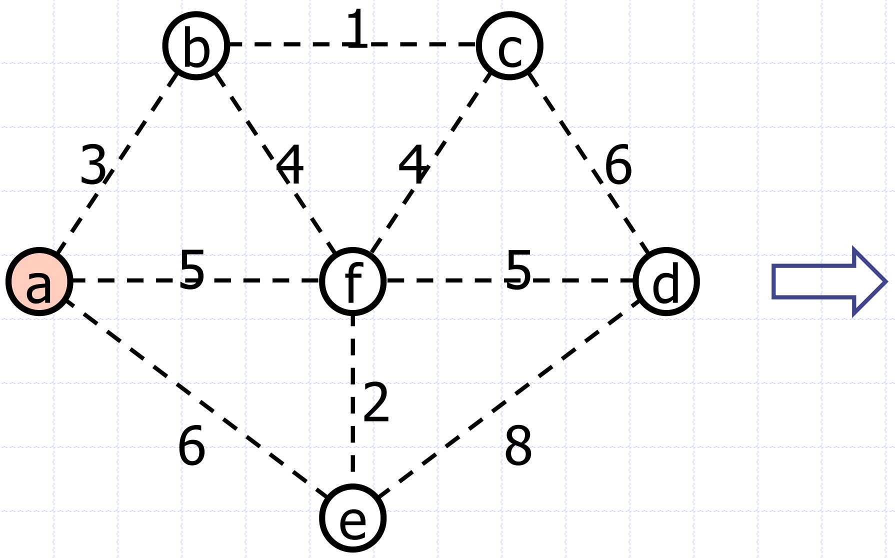




Ex12-2 prim's algorithm → Minimal Spanning Tree (MST) write down the process



Ex12-3 Kruskal's algorithm → Minimal Spanning Tree (MST) write down the process



b
1
c
3
4
4
6
a
5
f
5
d
2
6
8
e