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 (L12 – Shortest Path/Dijkstra algorithm)

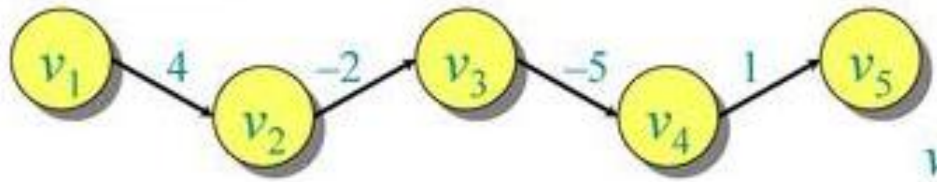
- Weighted Graph
- Shortest path problems
- Single-source shortest path problem
- Dijkstra's algorithm
- Dijkstra's algorithm versus BFS

Weighted Graph

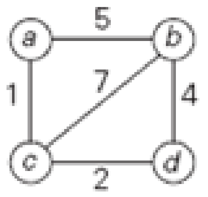
- Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

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

Example:



$w(p) = -2$



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

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

(a) (b) (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

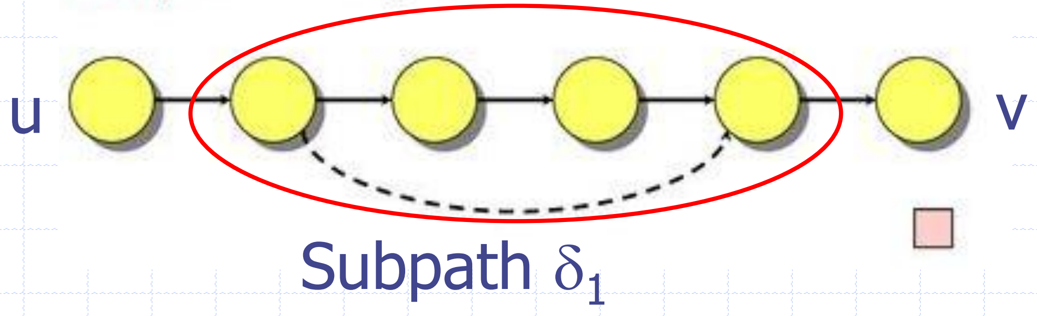
$$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}.$$

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:

If the subpath δ_1 is not a shortest path,
Then the path δ is not a shortest path.

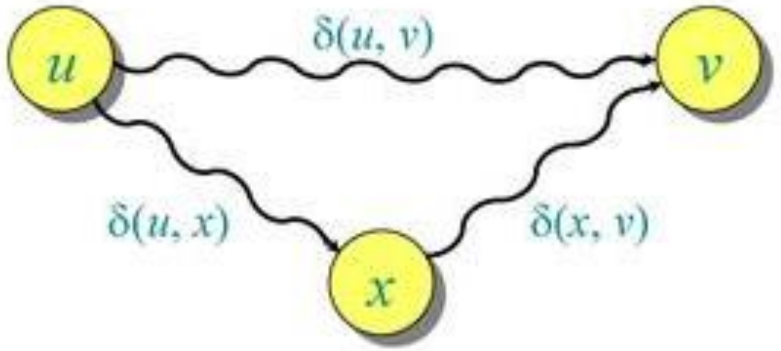


Triangle inequality

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

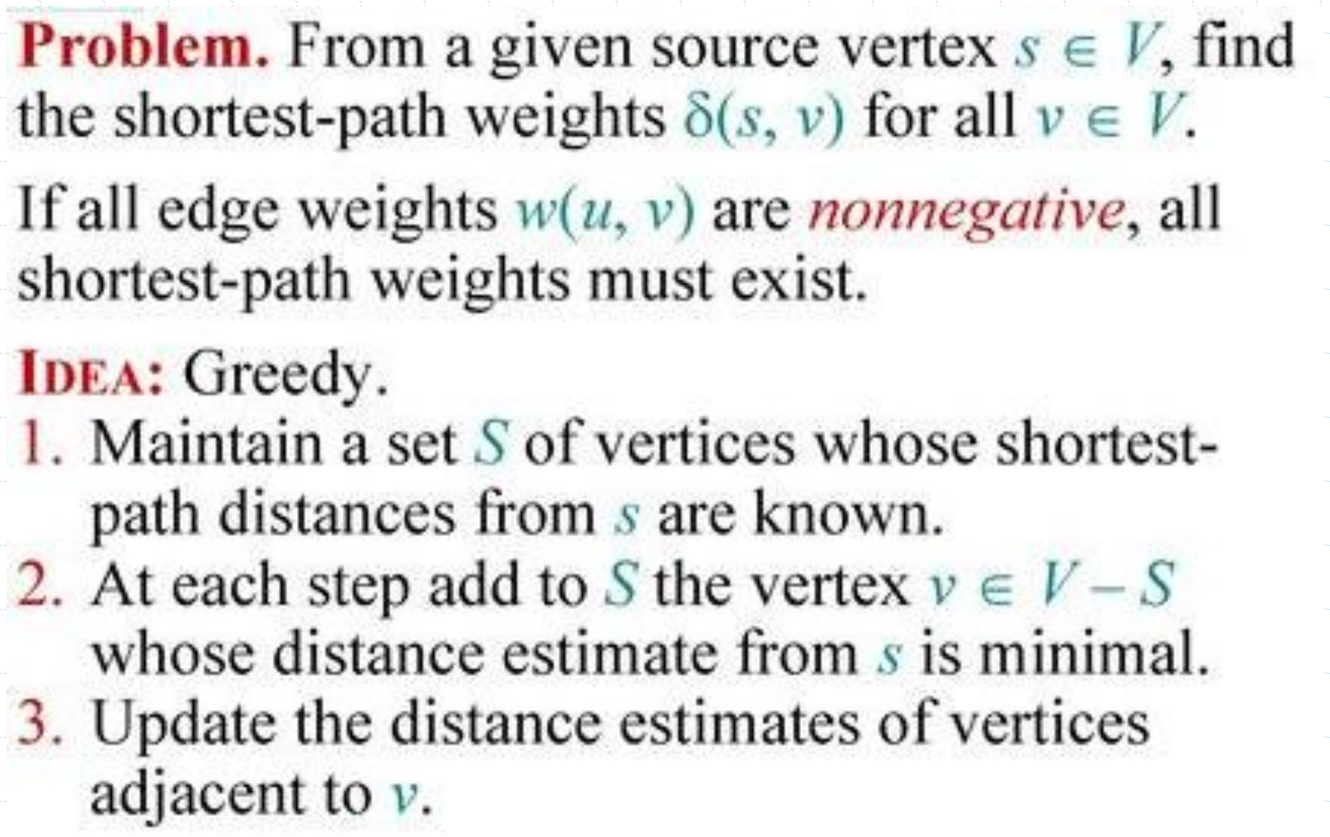
$$\delta(u, v) \le \delta(u, x) + \delta(x, v).$$

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.

Exercise 11-1 (work in class)

Let *G* be a graph whose vertices are integers 1 to 7, and let the adjacent list be given by the table below:
節の集合が整数1から7までのグラフ*G*また以下の隣接リストがある。

vertex	adjacent vertices
1	2 → 4 → 6
2	1 → 5
3	4 → 7
4	1 → 3 → 7
5	2 → 7
6	1 → 7
7	3 → 4 → 5 → 6

1. Draw *G*　　グラフ*G*を描きなさい
2. Show the adjacency matrix of *G*　グラフ*G*の隣接マトリクスを書きなさい
3. Show the spanning tree of *G* using a DFS traversal starting from vertex 1.
   節の集合1からDFSを使ってグラフ*G*の全域木を書きなさい。
4. Show the spanning tree of *G* using a BFS traversal starting from vertex 1.
   節の集合1からBFSを使ってグラフ*G*の全域木を書きなさい。

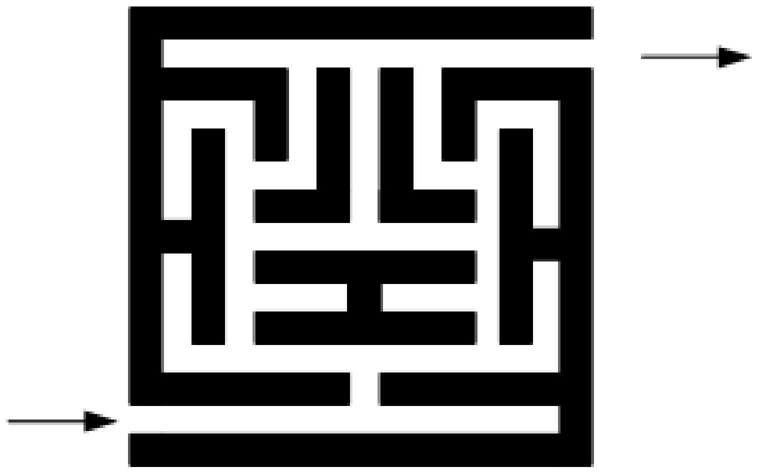
Exercise 11-2

Write an algorithm to check whether a graph *G* has cycles or not. What is the running time of your algorithm? (Hint: modify the DFS. Any back edge will create a cycle)
グラフ*G*にサイクルがあるかどうかをチェックするアルゴリズムを書いてください。
またそのアルゴリズムの実行時間は? (ヒント: DFSを変更してください)

Exercise 11-3

One can model a maze by having a vertex for a starting point, a finishing point, dead ends, and all the points in the maze where more than one path can be taken, and then connecting the vertices according to the paths in the maze.

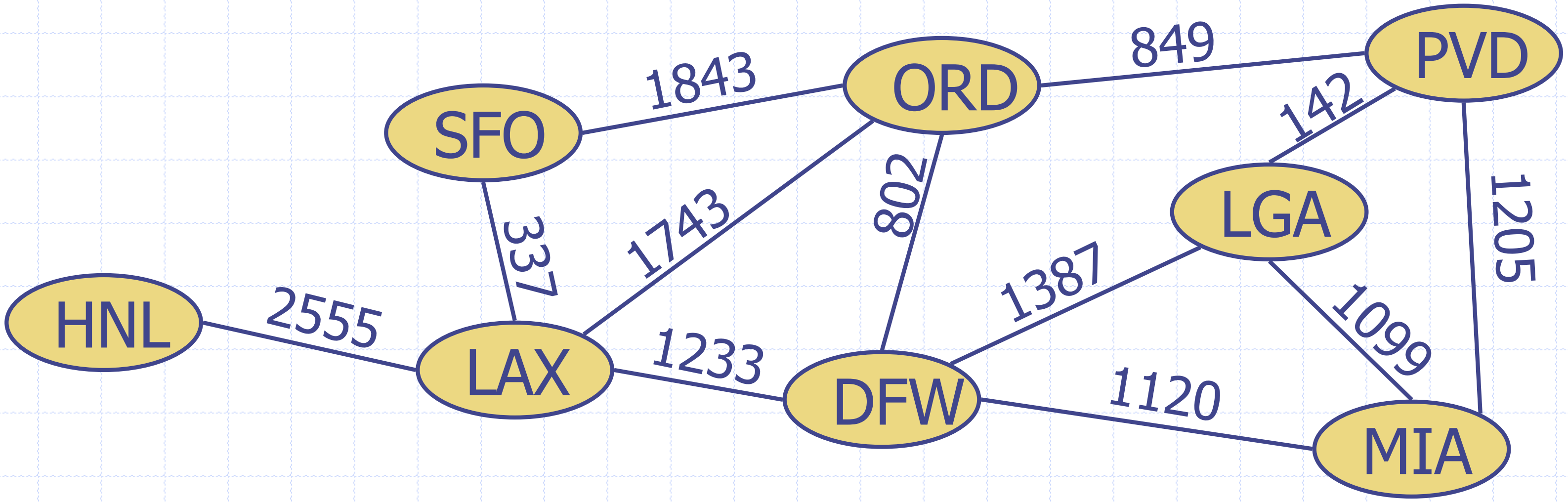
a. Construct such a graph for the following maze.



b. Which traversal— DFS or BFS— would you use if you found yourself in a maze and why?

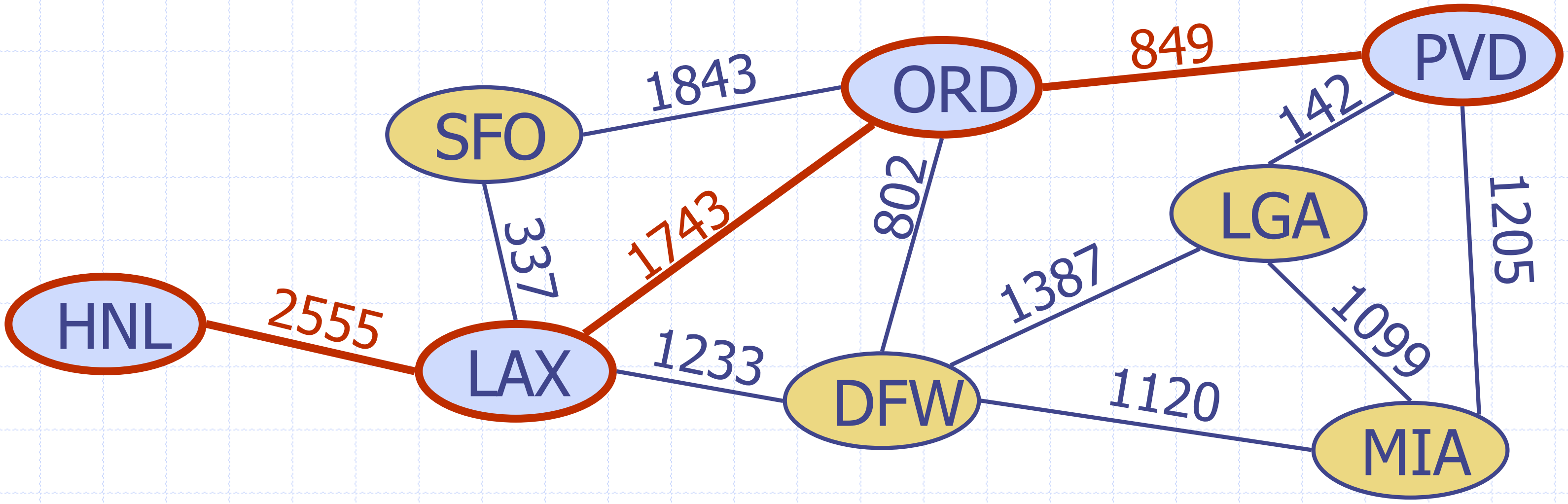
Weighted Graph
重み付きグラフ

- In a weighted graph, each edge has an associated numerical value, called the weight of the edge
  重み付きグラフでは、各枝に重みと呼ばれる数値をもっている。
- Edge weights may represent, distances, costs, etc.
  枝の重みは距離やコストなどを表している。
- Example:
  - In a flight route graph, the weight of an edge represents the distance in miles between the endpoint airports:各空港間の距離(マイル)



Shortest Path Problem
最短経路問題

- Given a weighted graph and two vertices u and v, we want to find a path of minimum total weight between u and v
  二つの頂点uとvからの重みの合計が最小となる経路を考える
- Applications
  - Flight reservations(飛行機予約)
  - Driving directions（ナビ、道案内）
  - Internet packet routing (インターネットパケットルーチン)
- Example:
  - Shortest path between Providence and Honolulu（プロビデンスーホノルル間）



Shortest Path Properties 最短経路の特徴

Property 1:

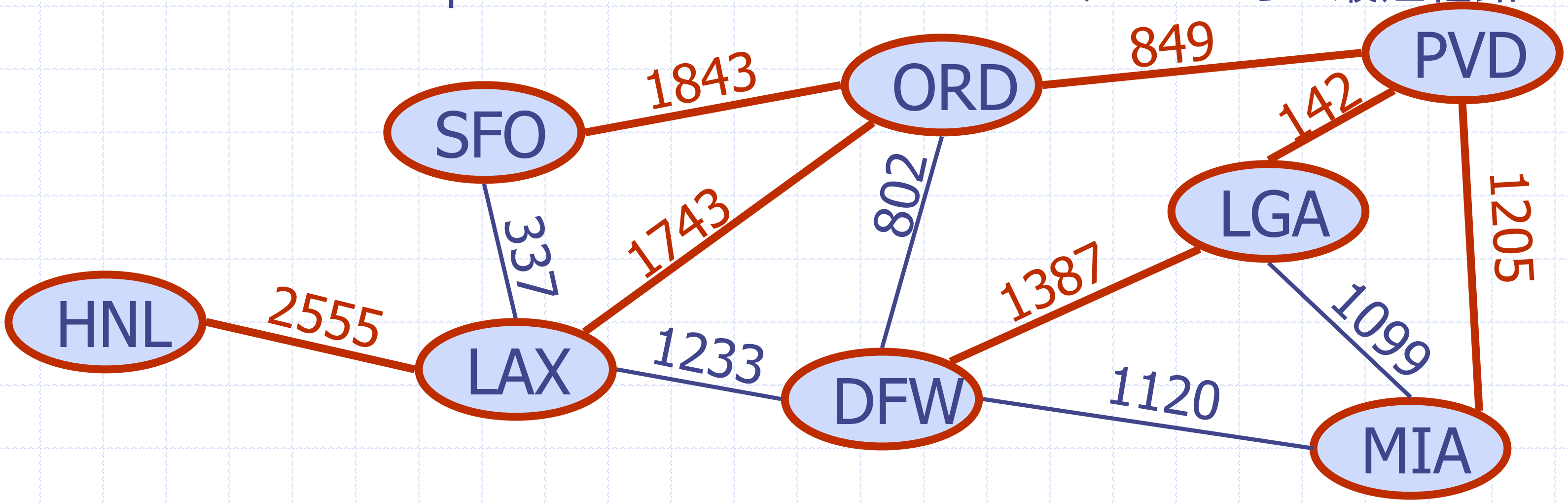
A subpath of a shortest path is itself a shortest path
最短経路の部分経路はそれ自体が最短経路である

Property 2:

There is a tree of shortest paths from a start vertex to all the other vertices　スタート地点から全ての点への最短経路の木

Example:

Tree of shortest paths from Providence　プロビデンスからの最短経路の木



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

- Algorithm　アルゴリズム
- Edge relaxation　枝の緩和
- Example　例
- Analysis　解析

Regarding Dijkstra’s Algorithm

(1)compute the distances of all the vertices from a given start vertex ***s***

ダイクストラのアルゴリズムは
スタート地点***s***から全ての頂点へ
の距離を計算する。

(2) grow a “cloud” of vertices, beginning with s and eventually covering all the vertices

頂点のクラウド(雲)をスタート地点から始め、徐々にすべての頂点へと広げていく。

(3) store each vertex ***v*** a label ***d***(***v***) representing the distance of ***v*** from ***s*** in the subgraph consisting of the cloud and its adjacent vertices

クラウド内のサブグラフとその隣接点はそれぞれスタート地点***s*** からの距離を表すラベル***d***(***v***) を持っている。

(4) The distance of a vertex v from a vertex s is the length of a shortest path between s and v

頂点sからvへの最短経路の長さを距離とする

Work in class:
Please write your version of Dijkstra’s algorithm (仮コード)

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 $d[v] \leftarrow d[u] + w(u, v)$
```

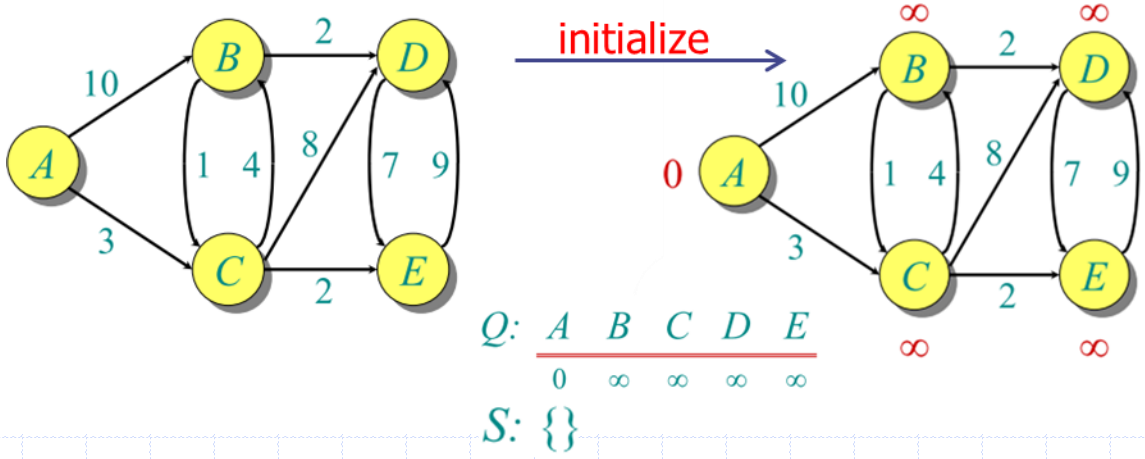
relaxation step

Implicit DECREASE-KEY

17

Dijkstra’s Algorithm (initialize)

```
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$          $\triangleright$ $Q$ is a priority queue maintaining $V - S$
```



Q: $A \quad B \quad C \quad D \quad E$

$0 \quad \infty \quad \infty \quad \infty \quad \infty$

S: $\{\}$

Dijkstra's Algorithm (pseudo code)

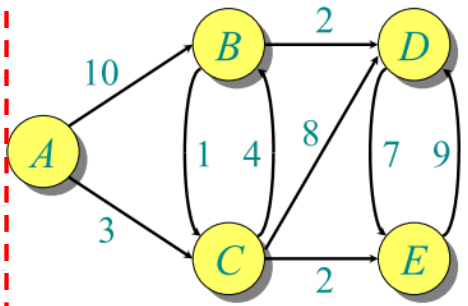
Initialize the Graph

```
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 $d[v] \leftarrow d[u] + w(u, v)$
```

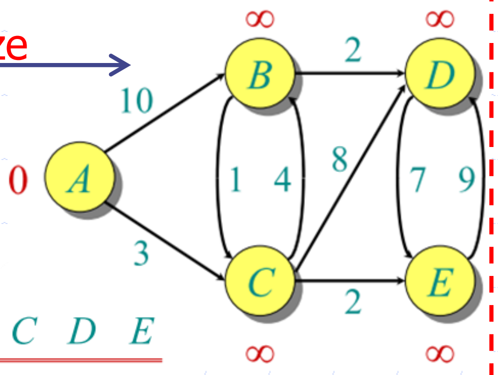
relaxation step

Implicit DECREASE-KEY

Example of Dijkstra’s algorithm



initialize

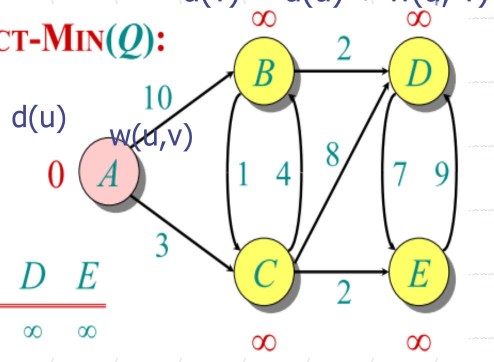


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

“A” ← Extract-Min(Q):

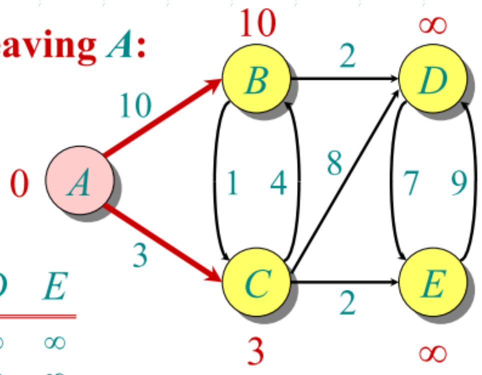
$d(v) > d(u) + w(u, v)$



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

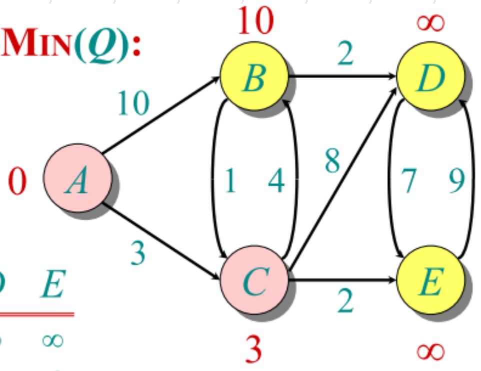
Relax all edges leaving A:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

“C” ← Extract-Min(Q):



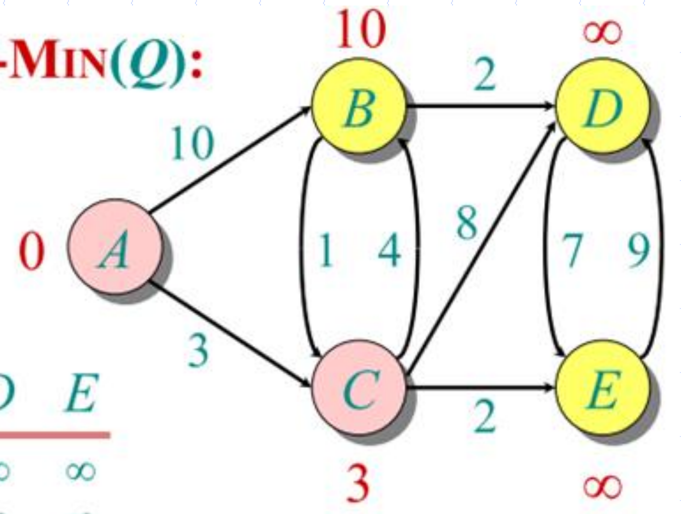
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

∞ infinity

Find a next node from Q to S

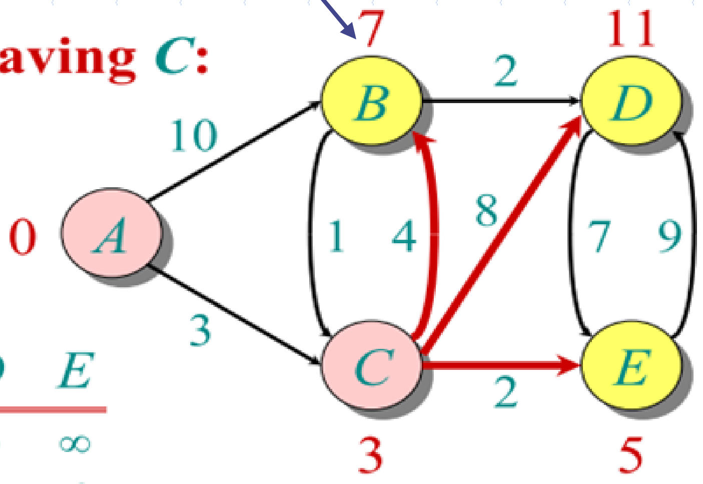
"*C*" ← EXTRACT-MIN(*Q*):



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

Update all path evaluation

Relax all edges leaving *C*:



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

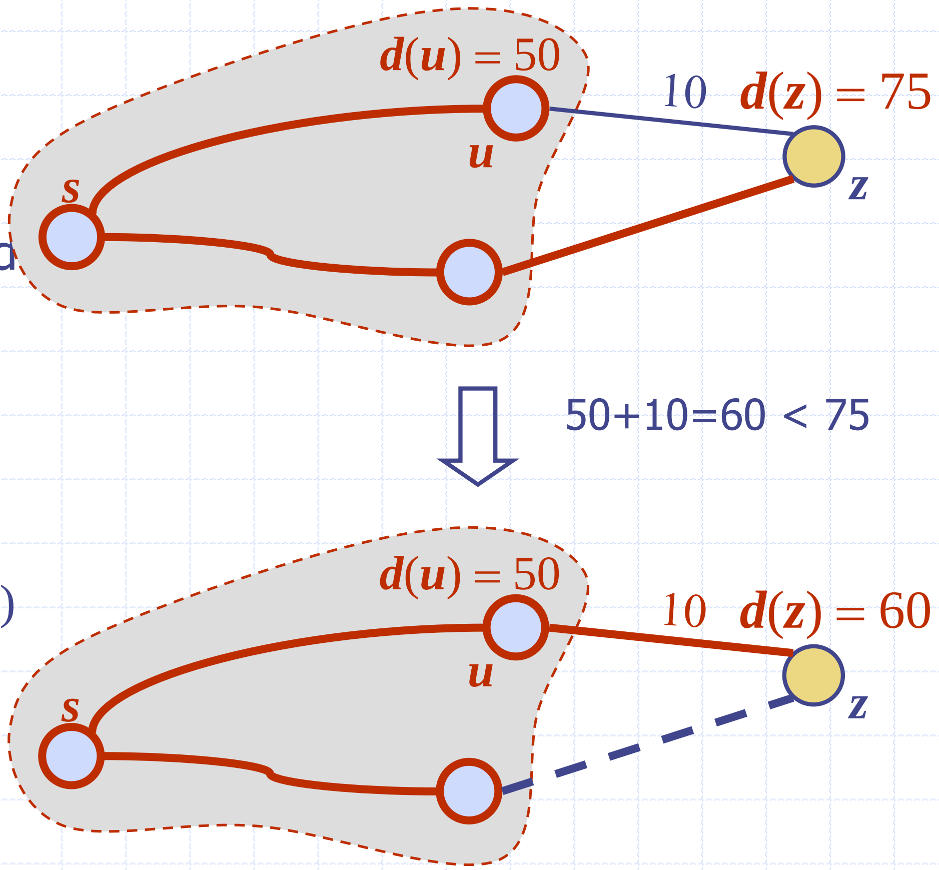
Edge Relaxation 枝の緩和

- Consider an edge $e = (u,z)$ such that
  - u is the vertex most recently added to the cloud
  - z is not in the cloud
- The relaxation of edge e updates distance $d(z)$ as follows

$$d(z) \leftarrow \min(d(z), d(u) + weight(e))$$

50+10=60 < 75

下の経路よりも上の経路を通ったほうが近いので、$d(z)$ の値を更新する。



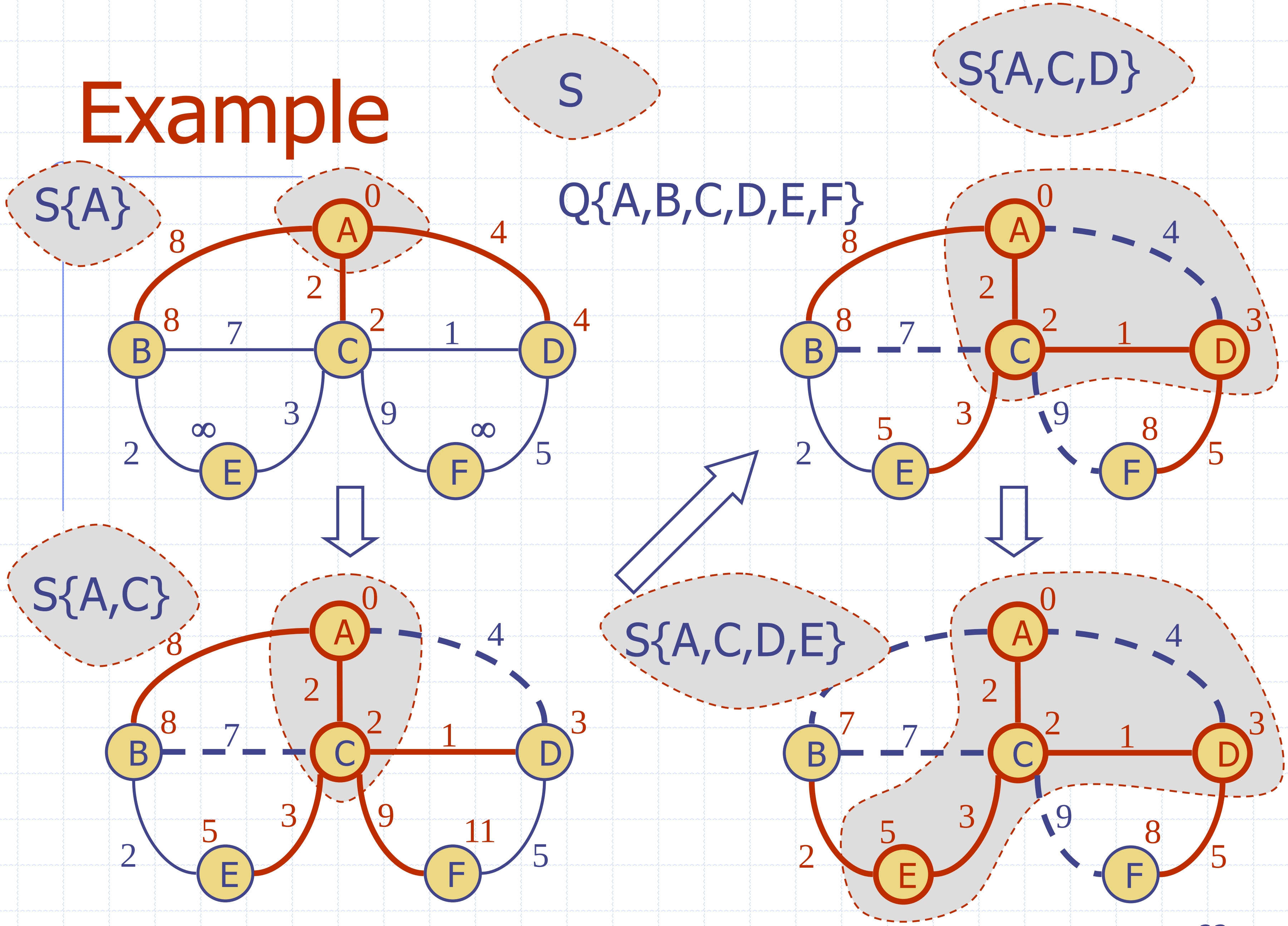
Dijkstra's Algorithm (pseudo code)

Work in class:
Please make comments to the algorithm

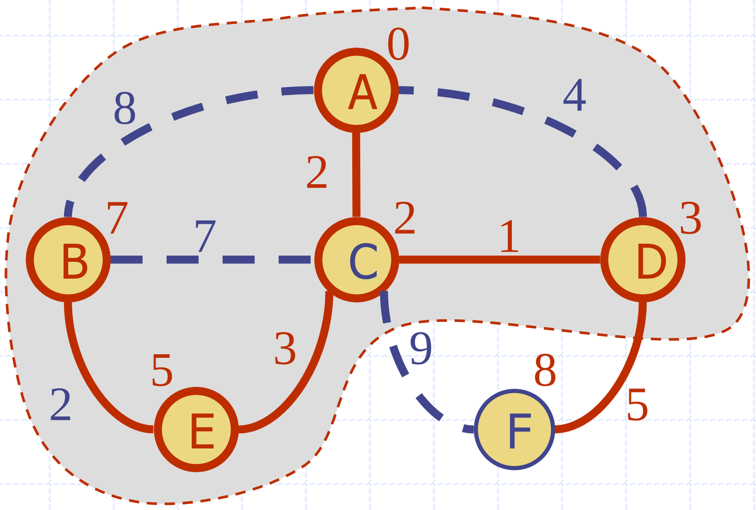
```
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$            ▷ $Q$ is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN$(Q)$
       $S \leftarrow S \cup \{u\}$
       for each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

Edge Relaxation

Example



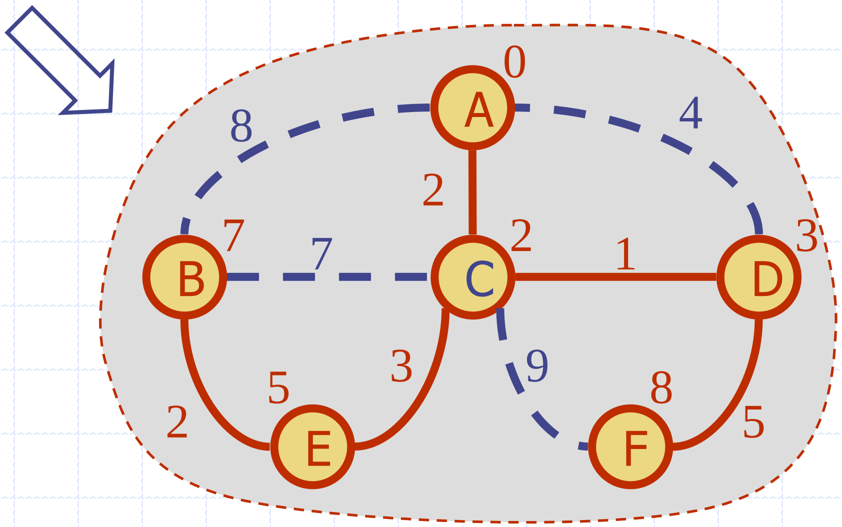
Example (cont.)



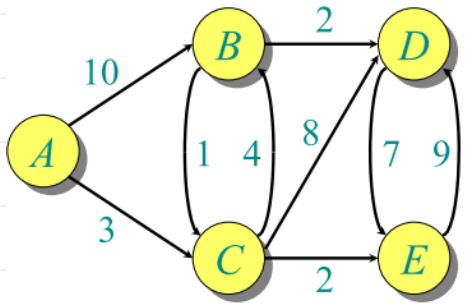
S{A,C,D,E,B}

S{A,C,D,E,B,F}

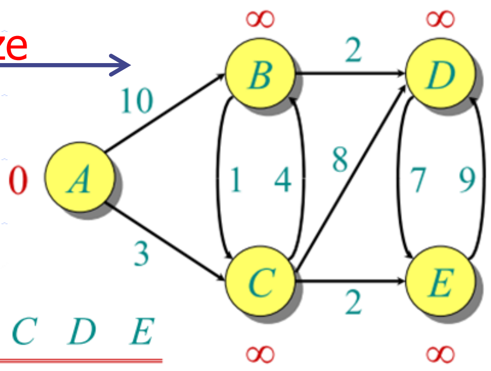
Q{}



Example of Dijkstra's algorithm



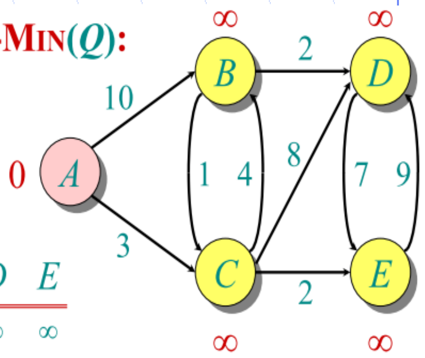
initialize $\longrightarrow$



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { }

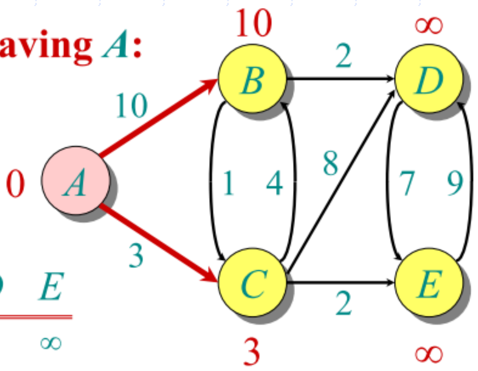
"A" ← EXTRACT-MIN(Q):



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

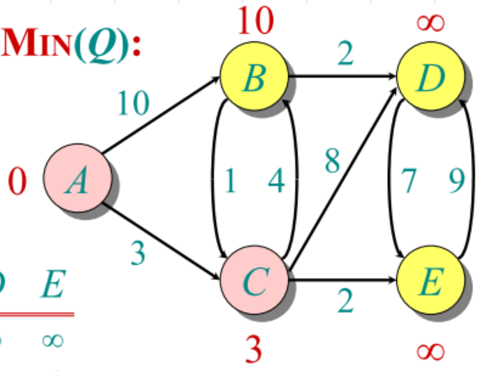
Relax all edges leaving A:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

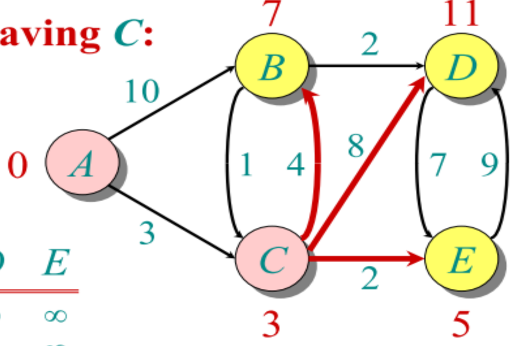
"C" ← EXTRACT-MIN(Q):



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

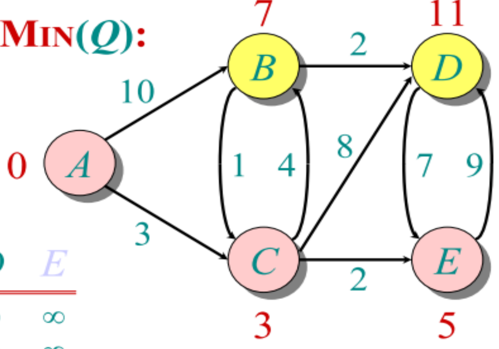
Relax all edges leaving *C*:



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

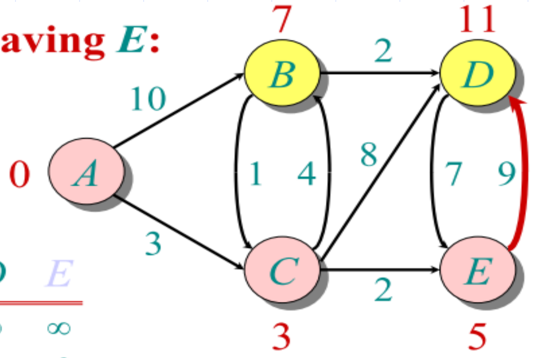
“*E*” ← EXTRACT-MIN(*Q*):



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C*, *E* }

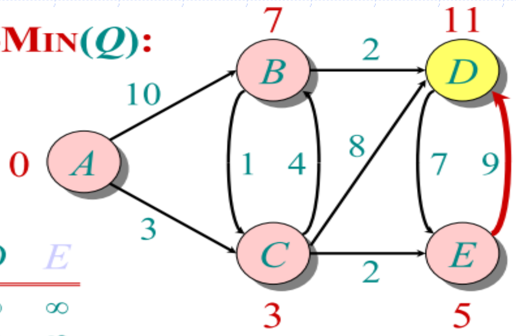
Relax all edges leaving *E*:



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E* }

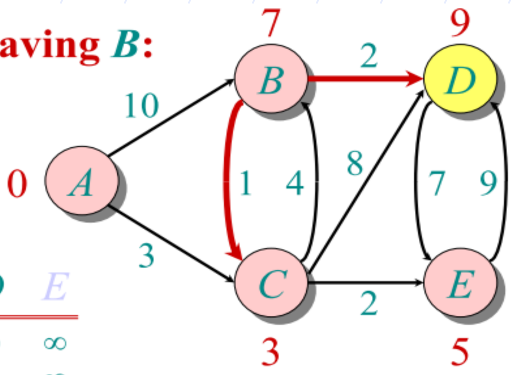
“*B*” ← EXTRACT-MIN(*Q*):



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E*, *B* }

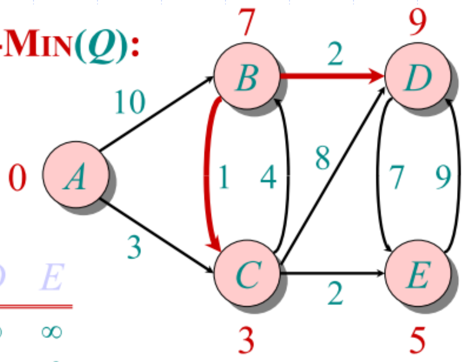
Relax all edges leaving *B*:



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B* }

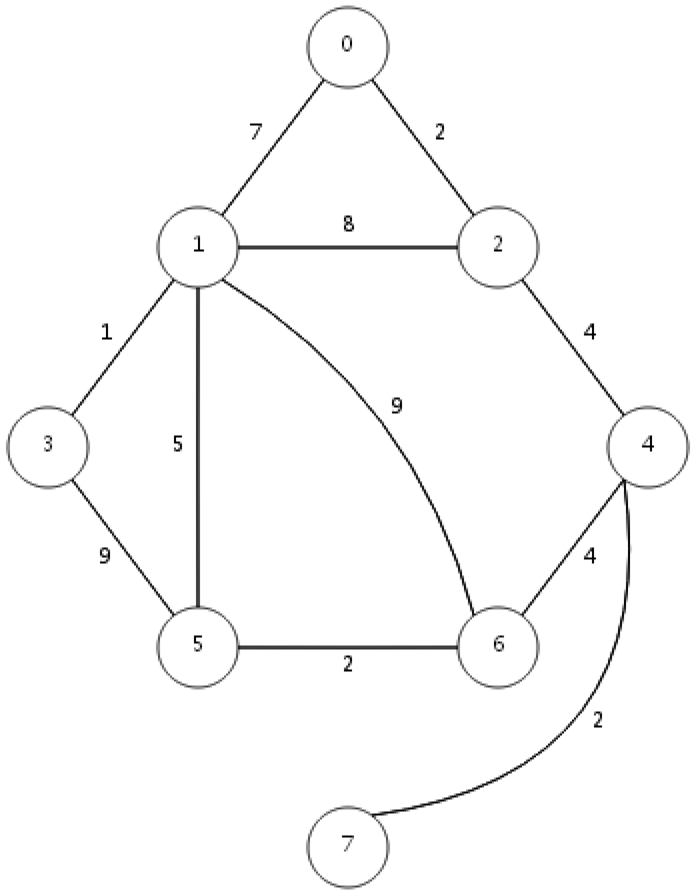
“*D*” ← EXTRACT-MIN(*Q*):



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B*, *D* }

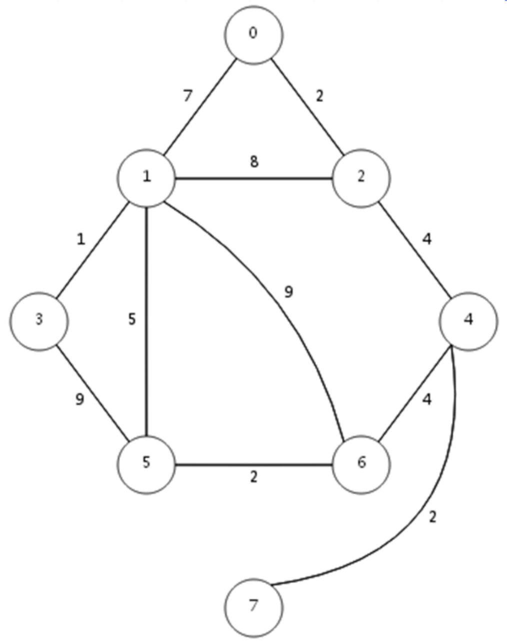
Work in class:
Start from vertex 0



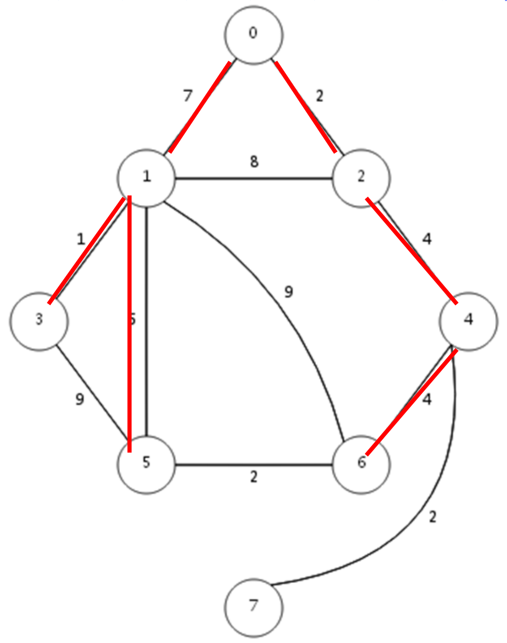
Vertex	Known	Total Cost	the previous vertex in the Path
0			
1			
2			
3			
4			
5			
6			
7			

Total the previous vertex in the

Vertex	Known	Cost	Path	
0	T	0	-1	0
1	T	7	0	0 1
2	T	2	0	0 2
3	T	8	1	0 1 3
4	T	6	2	0 2 4
5	T	12	1	0 1 5
6	T	10	4	0 2 4 6
7	T	8	4	0 2 4 7



Vertex	Known	Cost	Path	
0	T	0	-1	0
1	T	7	0	0 1
2	T	2	0	0 2
3	T	8	1	0 1 3
4	T	6	2	0 2 4
5	T	12	1	0 1 5
6	T	10	4	0 2 4 6
7	T	8	4	0 2 4 7



More animation
https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

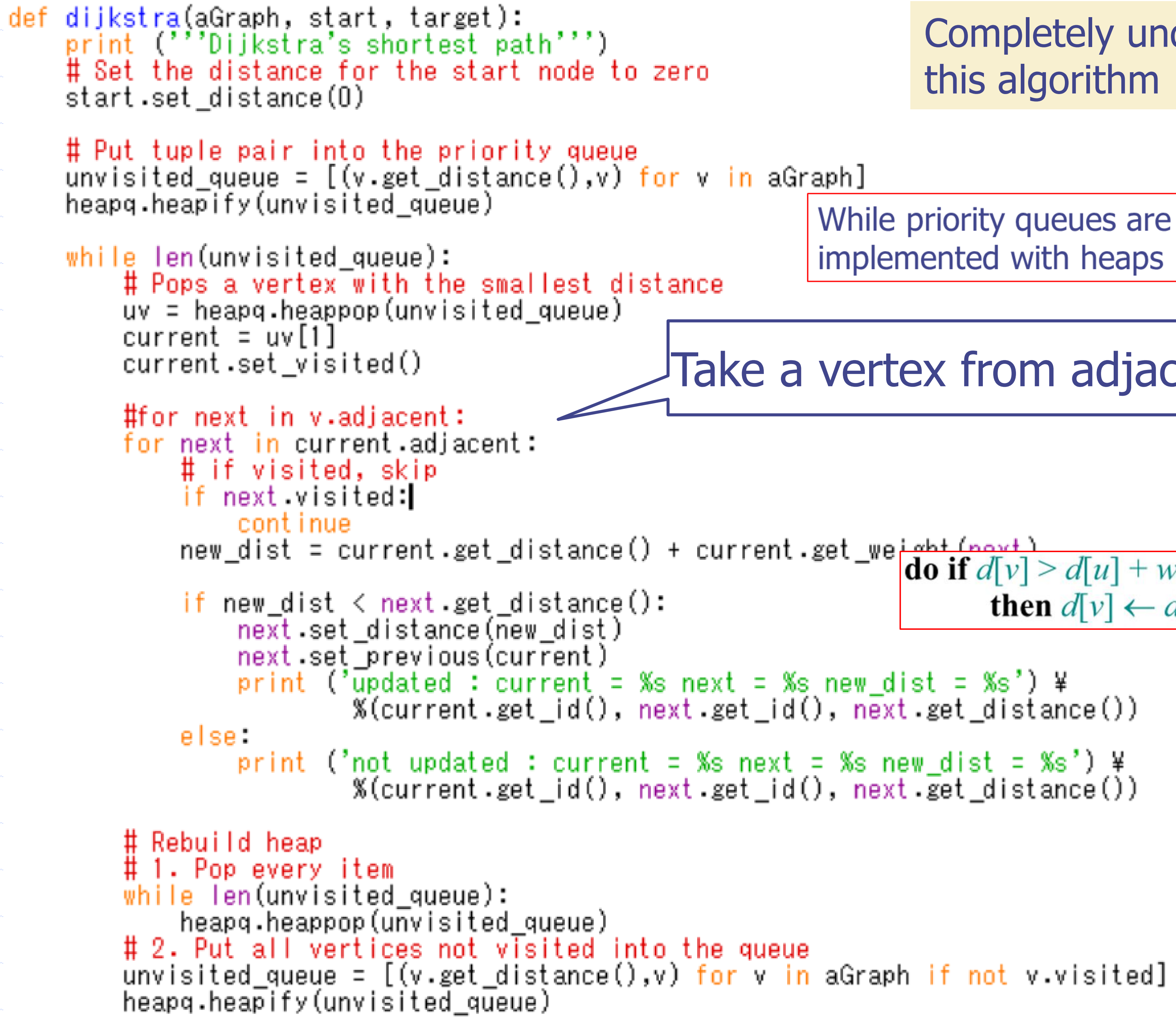
Dijkstra Implementation in Python

dijkstra-3.py

```
class Vertex:
    def __init__(self, node):
        self.id = node
        self.adjacent = {}
        # Set distance to infinity for all nodes
        self.distance = sys.maxint
        # Mark all nodes unvisited
        self.visited = False
        # Predecessor
        self.previous = None
```

Completely understand this algorithm

While priority queues are often implemented with heaps



```
def dijkstra(aGraph, start, target):
    print ('''Dijkstra's shortest path''')
    # Set the distance for the start node to zero
    start.set_distance(0)

    # Put tuple pair into the priority queue
    unvisited_queue = [(v.get_distance(),v) for v in aGraph]
    heapq.heapify(unvisited_queue)

    while len(unvisited_queue):
        # Pops a vertex with the smallest distance
        uv = heapq.heappop(unvisited_queue)
        current = uv[1]
        current.set_visited()

        #for next in v.adjacent:
        for next in current.adjacent:
            # if visited, skip
            if next.visited:
                continue
            new_dist = current.get_distance() + current.get_weight(next)

            if new_dist < next.get_distance():
                next.set_distance(new_dist)
                next.set_previous(current)
                print ('updated : current = %s next = %s new_dist = %s') ¥
                        %(current.get_id(), next.get_id(), next.get_distance())
            else:
                print ('not updated : current = %s next = %s new_dist = %s') ¥
                        %(current.get_id(), next.get_id(), next.get_distance())

        # Rebuild heap
        # 1. Pop every item
        while len(unvisited_queue):
            heapq.heappop(unvisited_queue)
        # 2. Put all vertices not visited into the queue
        unvisited_queue = [(v.get_distance(),v) for v in aGraph if not v.visited]
        heapq.heapify(unvisited_queue)
```

Take a vertex from adjacency list

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

```
def shortest(v, path):
    ''' make shortest path from v.previous'''
    if v.previous:
        path.append(v.previous.get_id())
        shortest(v.previous, path)
    return
```

Dijkstra’s algorithm algorithm

source_code

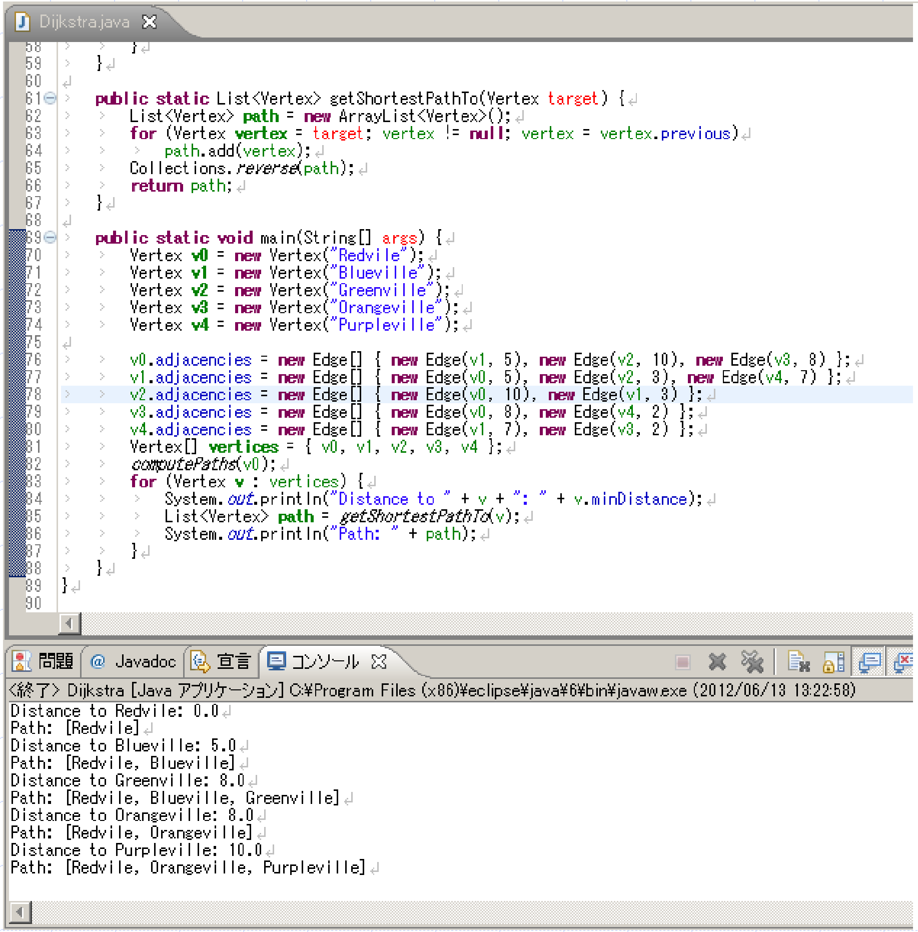
```java
public static void computePaths(Vertex source)
{
    source.minDistance = 0.;
    PriorityQueue<Vertex> vertexQueue = new PriorityQueue<Vertex>();
    vertexQueue.add(source);

    while (!vertexQueue.isEmpty()) {
        Vertex u = vertexQueue.poll();

        // Visit each edge exiting u
        for (Edge e : u.adjacencies)
        {
            Vertex v = e.target;
            double weight = e.weight;
            double distanceThroughU = u.minDistance + weight;
            if (distanceThroughU < v.minDistance) {
                vertexQueue.remove(v);
                v.minDistance = distanceThroughU ;
                v.previous = u;
                vertexQueue.add(v);
            }
        }
    }
```

poll() メソッドは、キューの先頭を削除および返します

```java
class Edge
{
    public final Vertex target;
    public final double weight;
    public Edge(Vertex argTarget, double argWeight)
    { target = argTarget; weight = argWeight; }
}
```

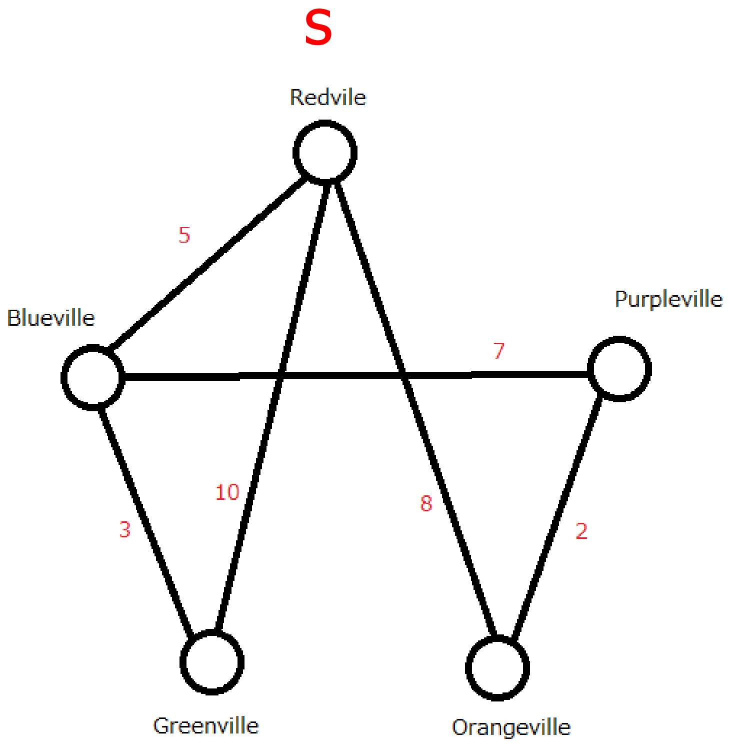
```java
class Vertex implements Comparable<Vertex>
{
    public final String name;
    public Edge[] adjacencies;
    public double minDistance = Double.POSITIVE_INFINITY;
    public Vertex previous;
    public Vertex(String argName) { name = argName; }
    public String toString() { return name; }
    public int compareTo(Vertex other)
    {
        return Double.compare(minDistance, other.minDistance);
    }
}
```

実行例：



```java
        }
    }

    public static List<Vertex> getShortestPathTo(Vertex target) {
        List<Vertex> path = new ArrayList<Vertex>();
        for (Vertex vertex = target; vertex != null; vertex = vertex.previous)
            path.add(vertex);
        Collections.reverse(path);
        return path;
    }

    public static void main(String[] args) {
        Vertex v0 = new Vertex("Redvile");
        Vertex v1 = new Vertex("Blueville");
        Vertex v2 = new Vertex("Greenville");
        Vertex v3 = new Vertex("Orangeville");
        Vertex v4 = new Vertex("Purpleville");

        v0.adjacencies = new Edge[] { new Edge(v1, 5), new Edge(v2, 10), new Edge(v3, 8) };
        v1.adjacencies = new Edge[] { new Edge(v0, 5), new Edge(v2, 3), new Edge(v4, 7) };
        v2.adjacencies = new Edge[] { new Edge(v0, 10), new Edge(v1, 3) };
        v3.adjacencies = new Edge[] { new Edge(v0, 8), new Edge(v4, 2) };
        v4.adjacencies = new Edge[] { new Edge(v1, 7), new Edge(v3, 2) };
        Vertex[] vertices = { v0, v1, v2, v3, v4 };
        computePaths(v0);
        for (Vertex v : vertices) {
            System.out.println("Distance to " + v + ": " + v.minDistance);
            List<Vertex> path = getShortestPathTo(v);
            System.out.println("Path: " + path);
        }
    }
}
```

```
<終了> Dijkstra [Java アプリケーション] C:¥Program Files (x86)¥eclipse¥java¥6¥bin¥javaw.exe (2012/06/13 13:22:58)
Distance to Redvile: 0.0
Path: [Redvile]
Distance to Blueville: 5.0
Path: [Redvile, Blueville]
Distance to Greenville: 8.0
Path: [Redvile, Blueville, Greenville]
Distance to Orangeville: 8.0
Path: [Redvile, Orangeville]
Distance to Purpleville: 10.0
Path: [Redvile, Orangeville, Purpleville]
```



Analysis of Dijkstra’s Algorithm

```
                              while $Q \neq \emptyset$
                                  do $u \leftarrow$ EXTRACT-MIN($Q$)
 $|V|$                               $S \leftarrow S \cup \{u\}$
 times                               for each $v \in Adj[u]$
            $degree(u)$                  do if $d[v] > d[u] + w(u, v)$
              times                           then $d[v] \leftarrow d[u] + w(u, v)$
```

$$\text{Time} = \Theta(V \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{DECREASE-KEY}})$$

Correctness (1)

Lemma. Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.

Proof. Suppose not. Let v be the first vertex for which $d[v] < \delta(s, v)$, and let u be the vertex that caused $d[v]$ to change: $d[v] = d[u] + w(u, v)$. Then,

$d[v] < \delta(s, v)$	supposition
$\leq \delta(s, u) + \delta(u, v)$	triangle inequality
$\leq \delta(s,u) + w(u, v)$	sh. path $\leq$ specific path
$\leq d[u] + w(u, v)$	v is first violation

Contradiction. □

Correctness (2)

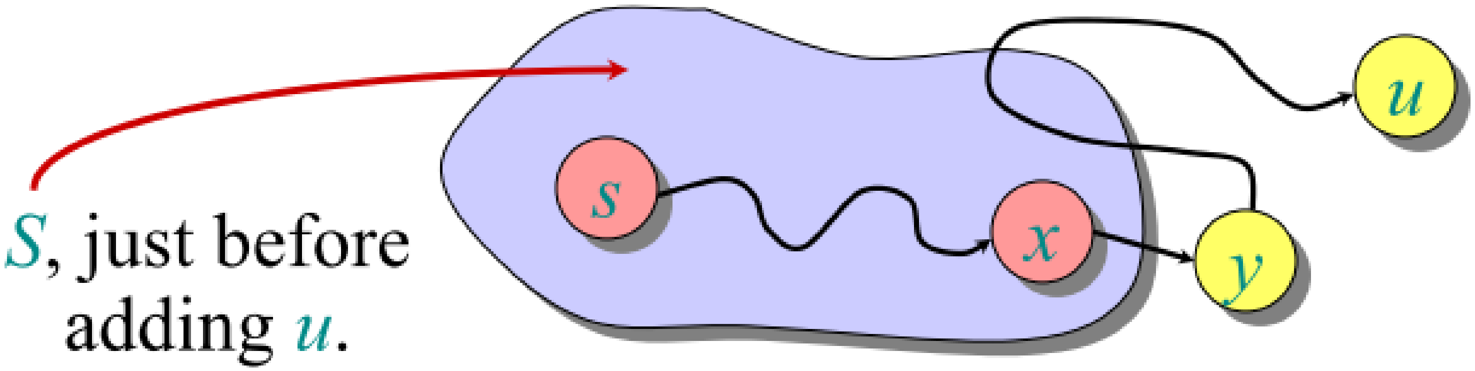
Lemma. Let u be v's predecessor on a shortest path from s to v. Then, if $d[u] = \delta(s, u)$ and edge (u, v) is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.

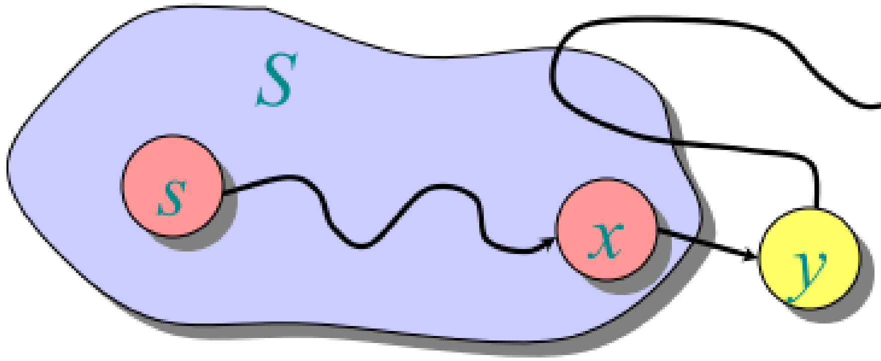
Proof. Observe that $\delta(s, v) = \delta(s, u) + w(u, v)$. Suppose that $d[v] > \delta(s, v)$ before the relaxation. (Otherwise, we're done.) Then, the test $d[v] > d[u] + w(u, v)$ succeeds, because $d[v] > \delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$, and the algorithm sets $d[v] = d[u] + w(u, v) = \delta(s, v)$. □

Correctness (3)

Theorem. Dijkstra's algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

Proof. It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when v is added to S. Suppose u is the first vertex added to S for which $d[u] > \delta(s, u)$. Let y be the first vertex in $V - S$ along a shortest path from s to u, and let x be its predecessor:





Since u is the first vertex violating the claimed invariant, we have $d[x] = \delta(s, x)$. When x was added to S, the edge (x, y) was relaxed, which implies that $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$. But, $d[u] \leq d[y]$ by our choice of u. Contradiction. □

Analysis of Dijkstra's Algorithm

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$

BFS on unweighted graphs

Suppose that $w(u, v) = 1$ for all $(u, v) \in E$. Can Dijkstra's algorithm be improved?

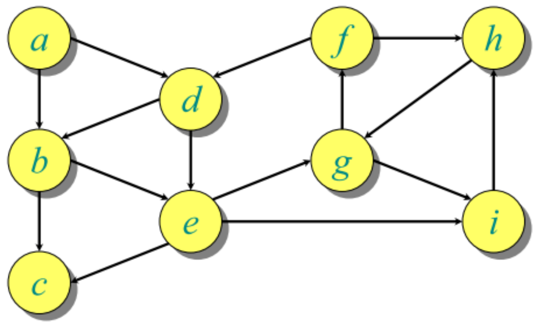
- Use a simple FIFO queue instead of a priority queue.

Breadth-first search

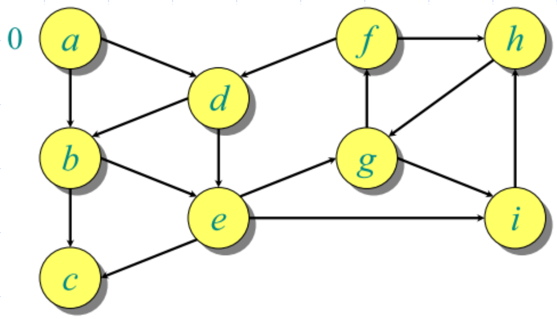
```
while $Q \neq \emptyset$
    do $u \leftarrow$ DEQUEUE($Q$)
        for each $v \in Adj[u]$
            do if $d[v] = \infty$
                then $d[v] \leftarrow d[u] + 1$
                     ENQUEUE($Q, v$)
```

W(u, v)

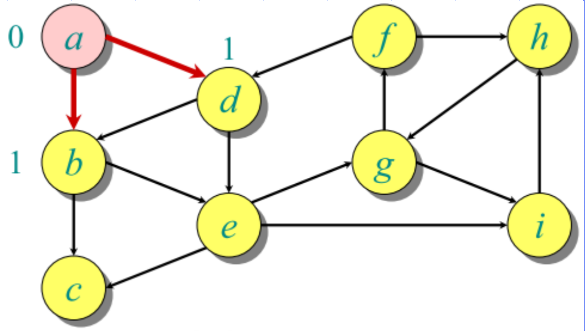
Analysis: Time = $O(V + E)$.



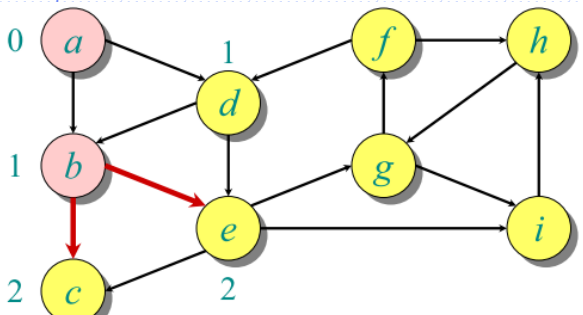
Q:



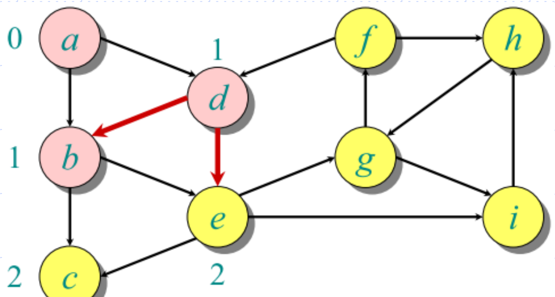
0
Q: *a*



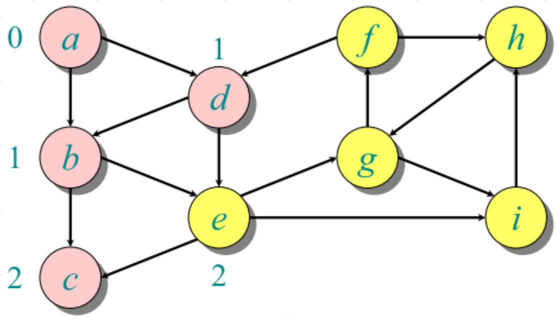
1 1
Q: *a b d*



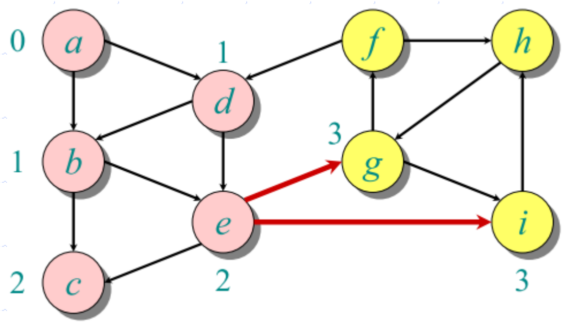
1 2 2
Q: *a b d c e*



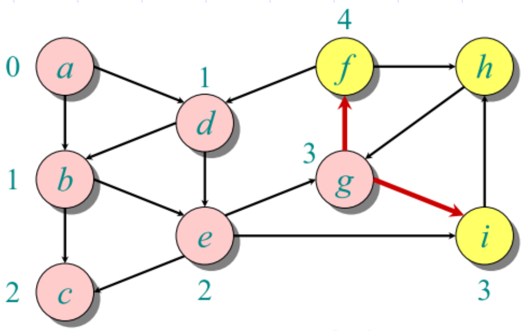
2 2
Q: *a b d c e*



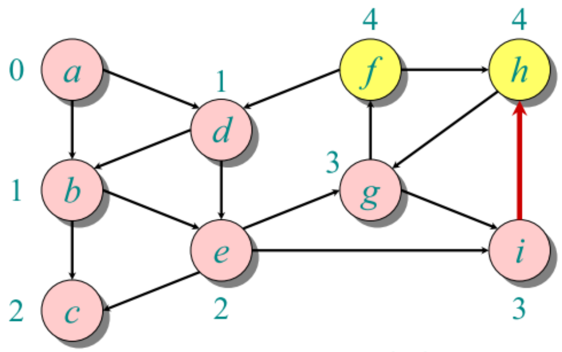
2
Q: *a b d c e*



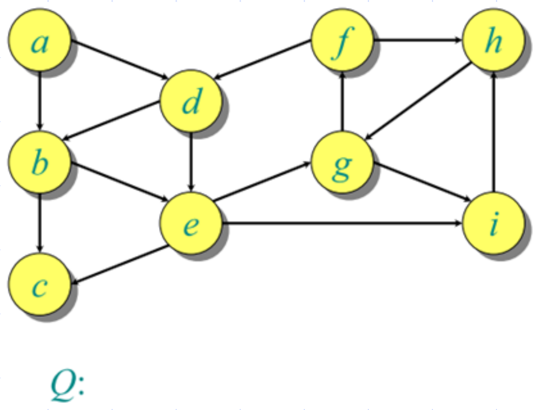
3 3
Q: *a b d c e g i*



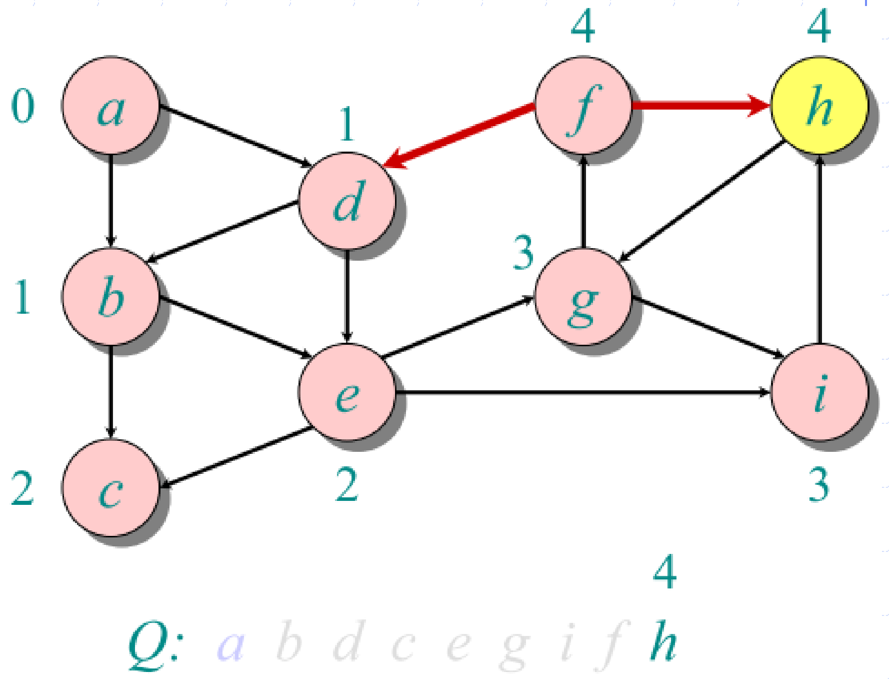
3 4
Q: *a b d c e g i f*



4 4
Q: *a b d c e g i f h*



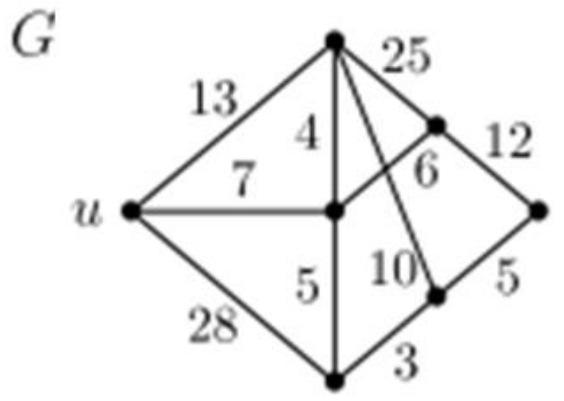
BFS



Exercises 12-1

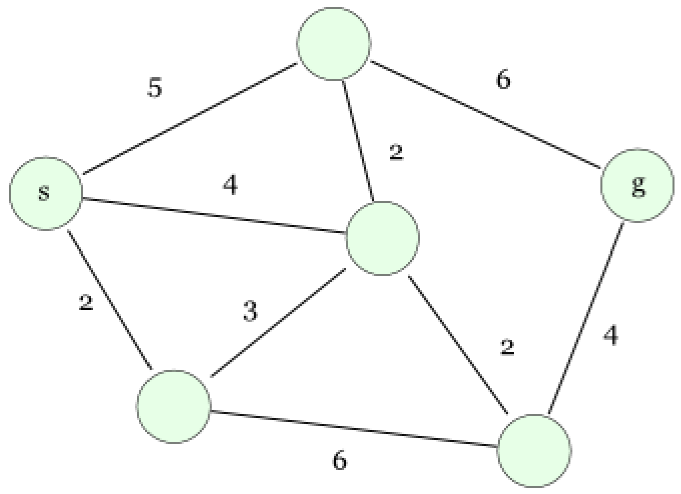
Use Dijkstra's algorithm to find a shortest-distance tree rooted at u in the graph G above, labelling the vertices clearly at each stage of the algorithm, and naming the vertices other than u as $a, b, \ldots$ in the order in which you process them. (When a label is superseded at a later stage of the algorithm, cross it out in such a way that it is still legible beneath the crossing.)

ダイクストラアルゴリズムを用いてグラフGのノードuを根とする最短経路木を見つけなさい。アルゴリズムの各段階において各頂点の重みと名前を記すこと。



Exercises 12-2

ダイクストラアルゴリズムを用いてグラフGのノードsを根とする最短経路木を見つけなさい。アルゴリズムの各段階において各頂点の重みと名前を記すこと。



Exercises 11-3

Please add comments to the Dijkstra’s algorithm in Java. (source_code)