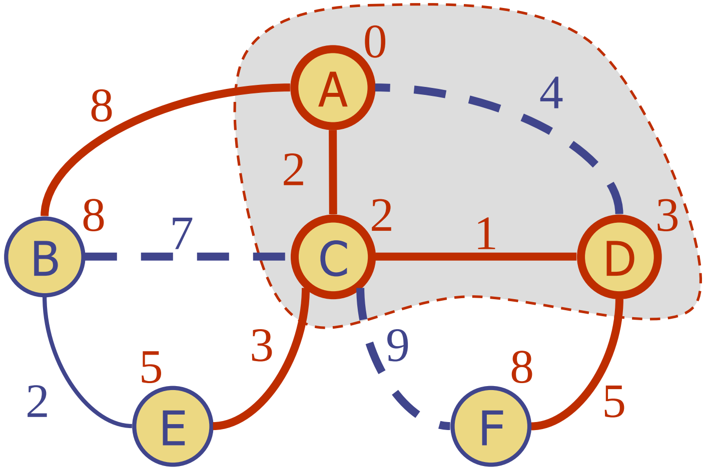
最短通路



weighted graph (加权图、有权图)

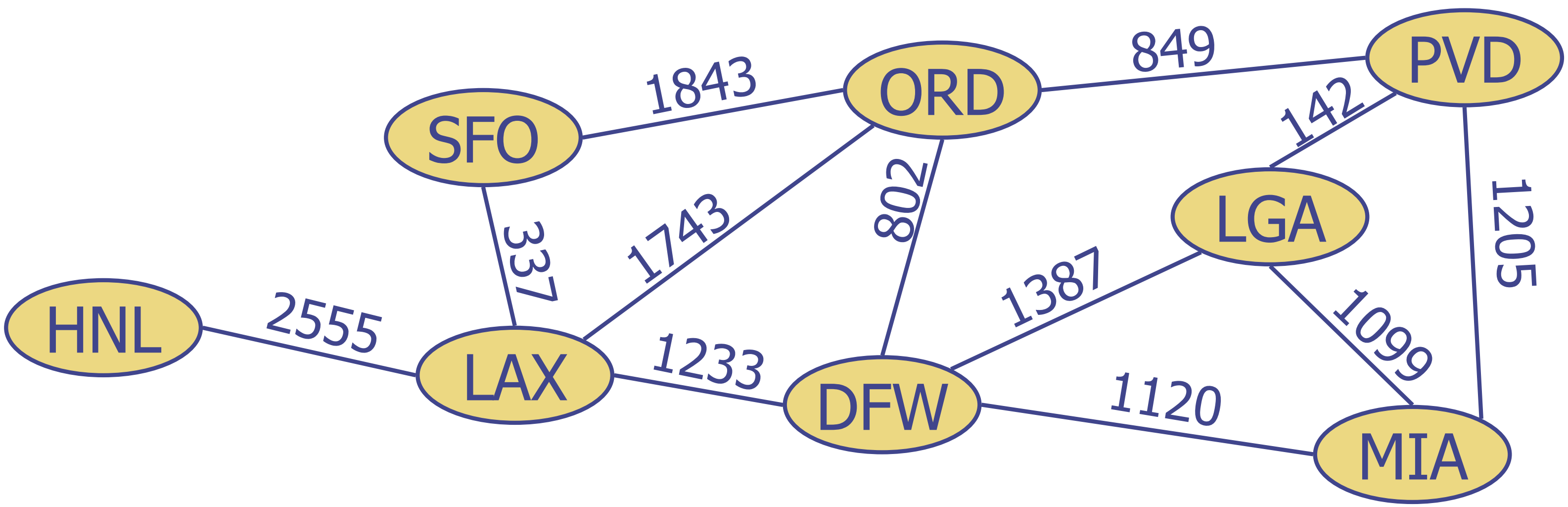
- 在某些时候某些场合，并非图的所有边都一样长。出于某些原因和目的，需要给图的每条边加权(某种意义上的值、长度)， 也即给边赋一个值，称为边的长度。

- 举例说明一下加权的必要性。

- Weights can also be attached to the vertices instead of the edges or can be attached to both vertices and edges. The resulting graph is called a *weighted graph.*

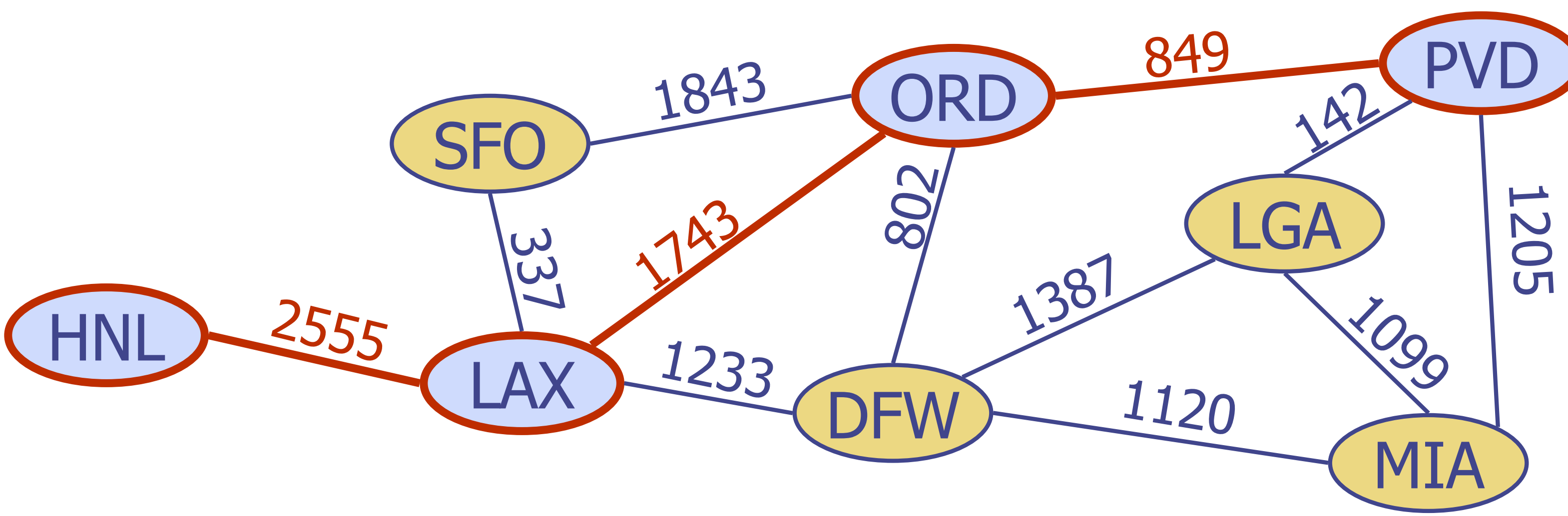
Weighted Graphs

- 加权图中的每条边被赋予一个数值（权）
- 权可以是距离，时间，成本，费用，带宽，吞吐量等等。
- Example:
  - In a flight route graph, the weight of an edge represents the distance in miles between the endpoint airports



Shortest Path Problem

- 最小通路问题：给定有权图中的两个不同结点，寻找两个点之间总权最小的通路
- E.G: Shortest path between Providence and Honolulu
- Applications
  - Internet packet routing
  - Flight reservations
  - Driving directions



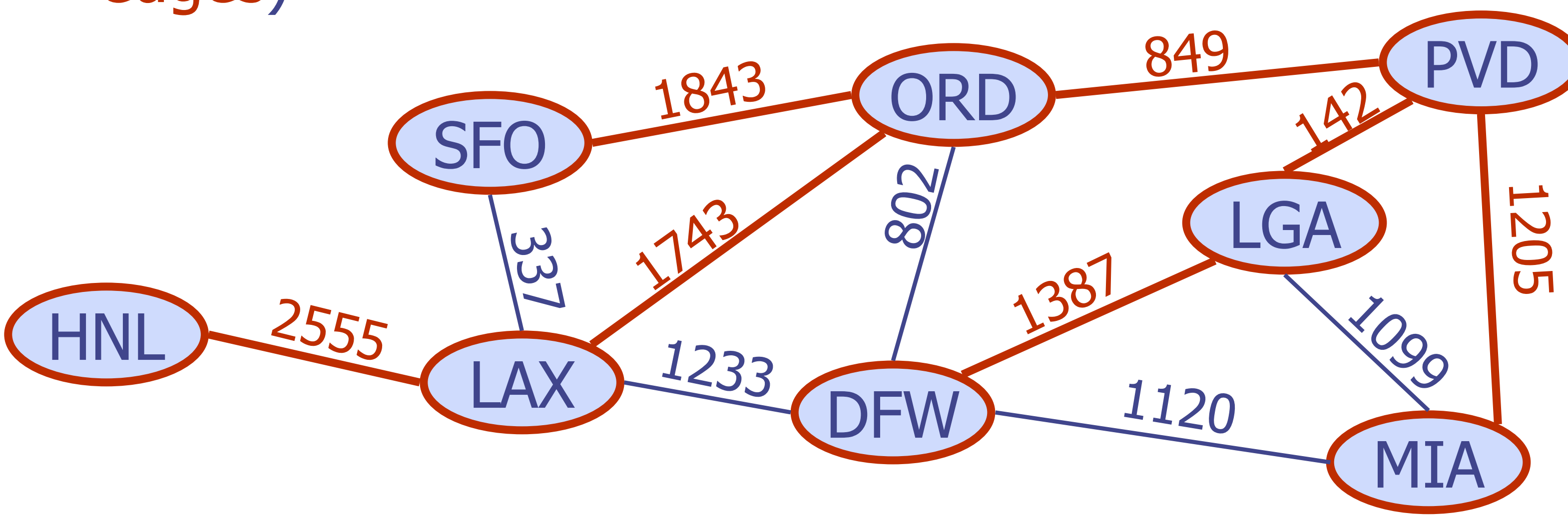
Shortest Path Properties

性质1：最短通路的子路本身也一定是一条最短通路。（Why?）

性质2：连通图中，存在一颗从一个起始结点到其它所有结点的最短路径的树。

Example:

Tree of shortest paths from Providence (those red edges)



Dijkstra's 算法 (最经典的、最常用的算法）

single-source shortest path problem in graph theory. 图论最短通路问题的单源算法 （给定一个点到其它所有连接的点的）

Works for both undirected and digraph. 但只适应非负权图。

算法Input: Weighted graph G=(V,E,f) and source vertex $s \in V$, such that all edge weights are nonnegative

算法Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $s \in V$ to all other vertices

Dijkstra's Algorithm (迪克斯特拉单源算法)

- 距离：一个结点到v到另一结点s的距离是v,s间的最短通路的长度，这里的长度是路的所有边的权之和。
- Dijkstra's algorithm 计算起点s到所有其它所有连接的点的距离。
- Assumptions:
  - 连通
  - 所有权值非负
  - 简单无向图

Dijkstra’s Algorithm (迪克斯特拉单源算法)

- “云”：结点集V的子集（从点s开始慢慢形成扩张）

- “云”算法，或者叫“水淹”算法：给每一个结点v保存一个临时值 ***d***(***v***)，表示在“云”以及云邻接的所有结点形成的子图中从起点s到v的距离。最终将这"云"扩大到整个图

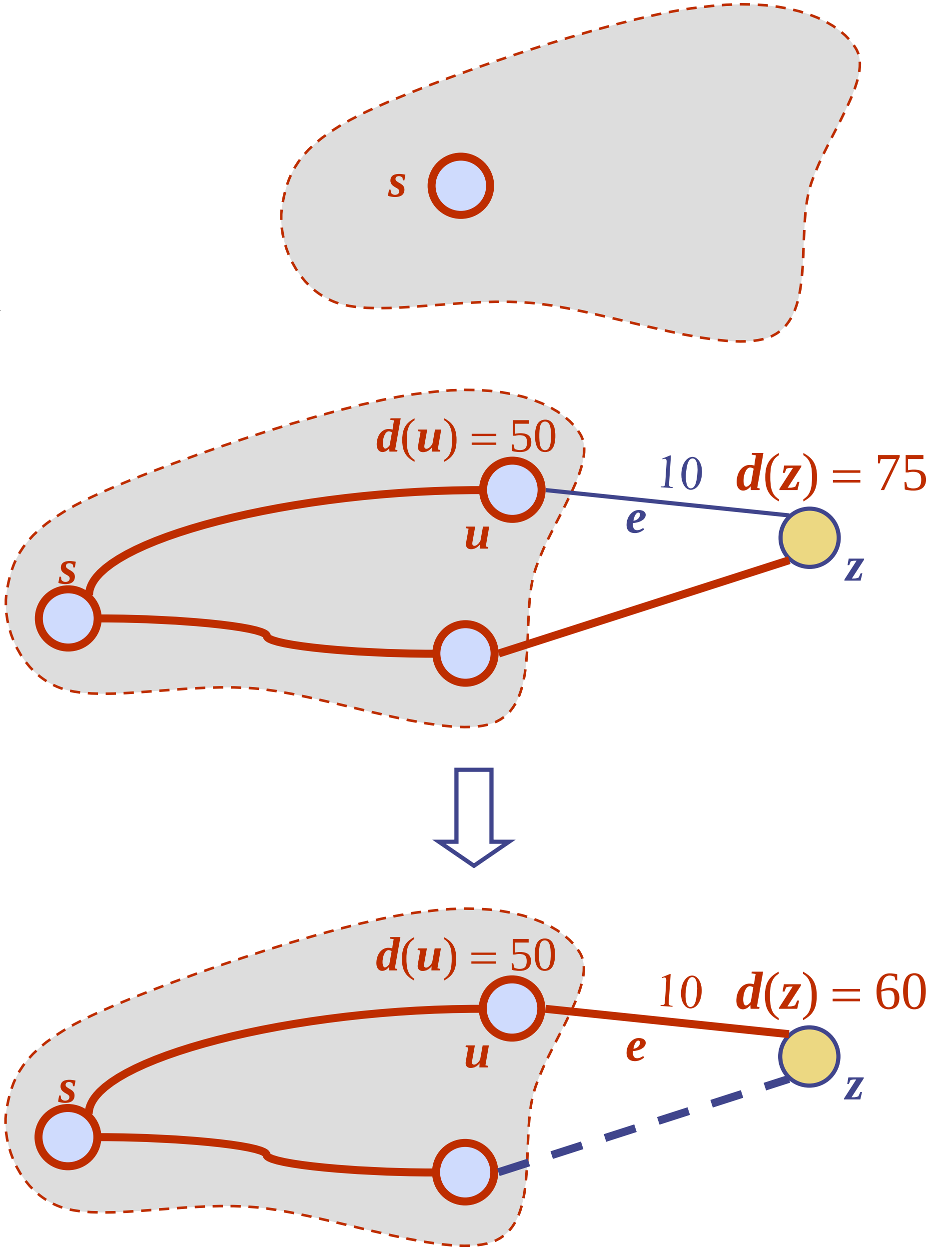
- At each step (“云”扩张过程，也即迭代的过程)
  - 开始“云”只包括结点s一个点 (d(s)=0)
  - 把云外的d(u)最小的点u加入到云中（也即离“云”最近的点）
  - 更新“云”外与u邻接的结点的的标记d(v).（关键搞清楚如何更新d(v)）
  - 当“云”扩张到了整个图，所有的d(v)都标注完，任务完成

Edge Relaxation

- 第一次给所有与起点s邻接的点中离s最近的点标注一个距离d(v)（也即相应的边长）。每一个与s邻接的结点标注成相应边的长度；其它所有外围的点标为∞

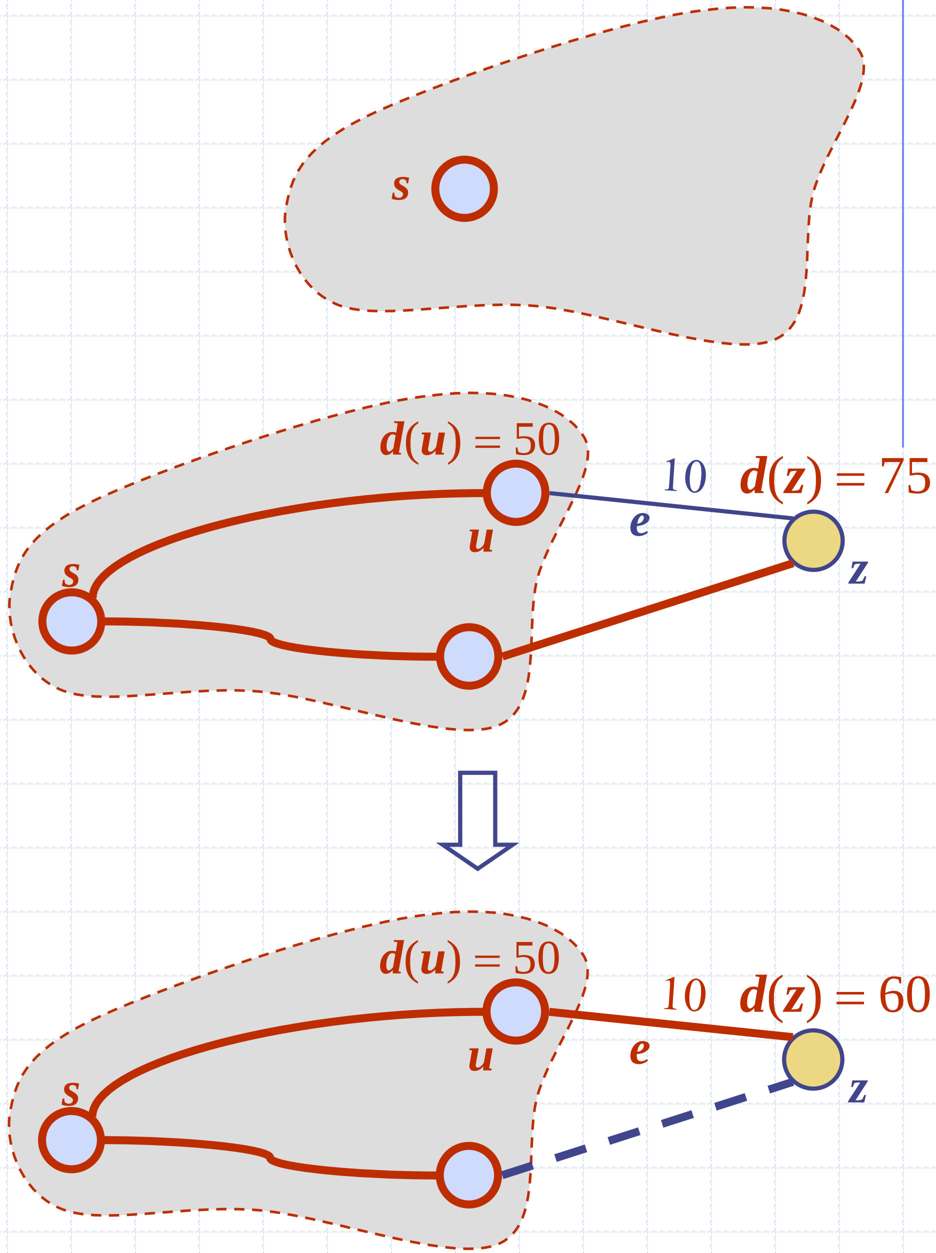
- Consider an edge $e = (u,z)$ such that
  - u 是最近加入到云中的结点
- The relaxation of edge e updates distance $d(z)$ as follows:

  $d(z) = \min\{d(z), d(u) + weight(e)\}$ 逐个更新与云中点u邻接的在云外的结点z的标注值d(z)（“云”周边的）

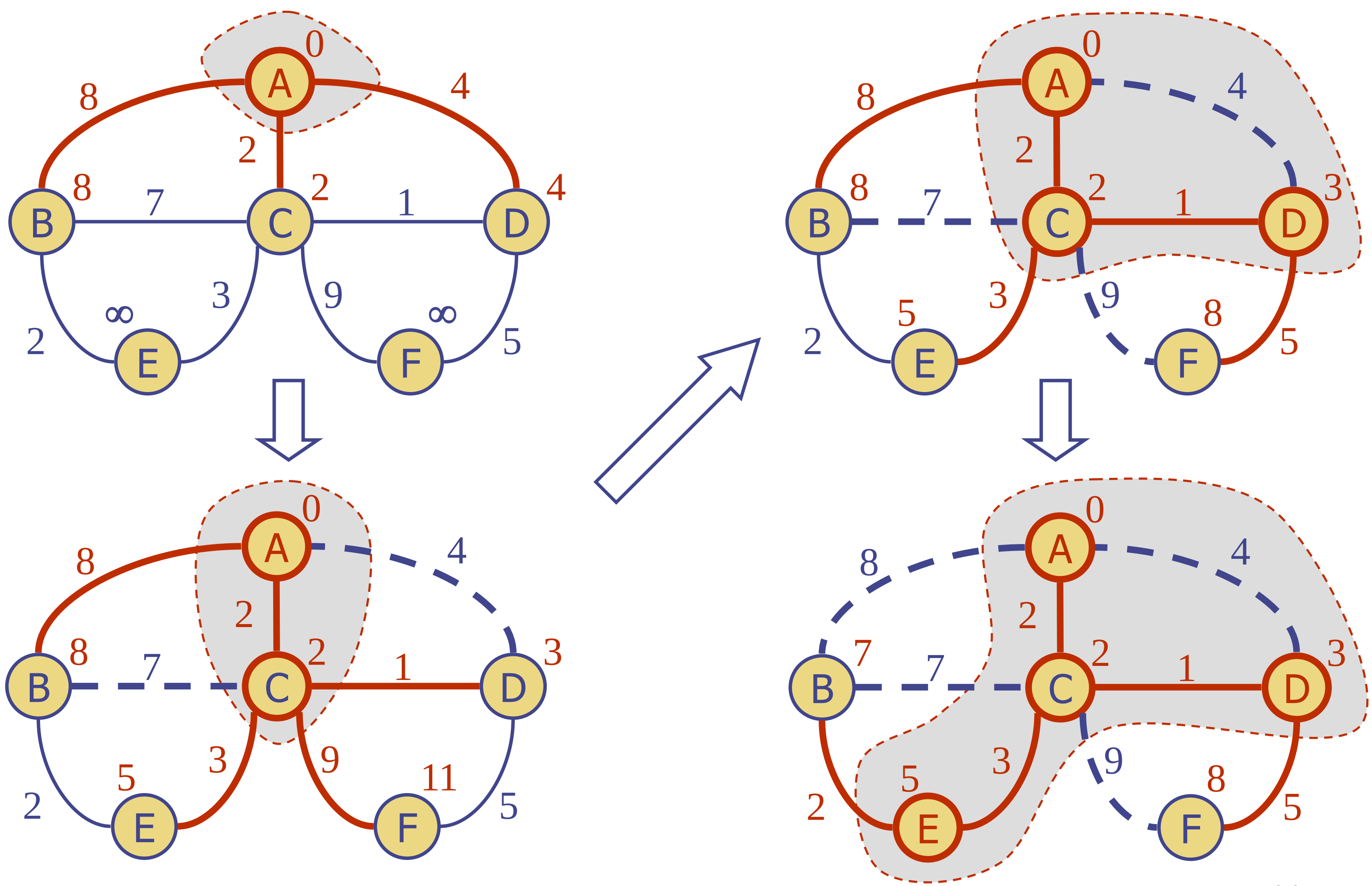


Edge Relaxation

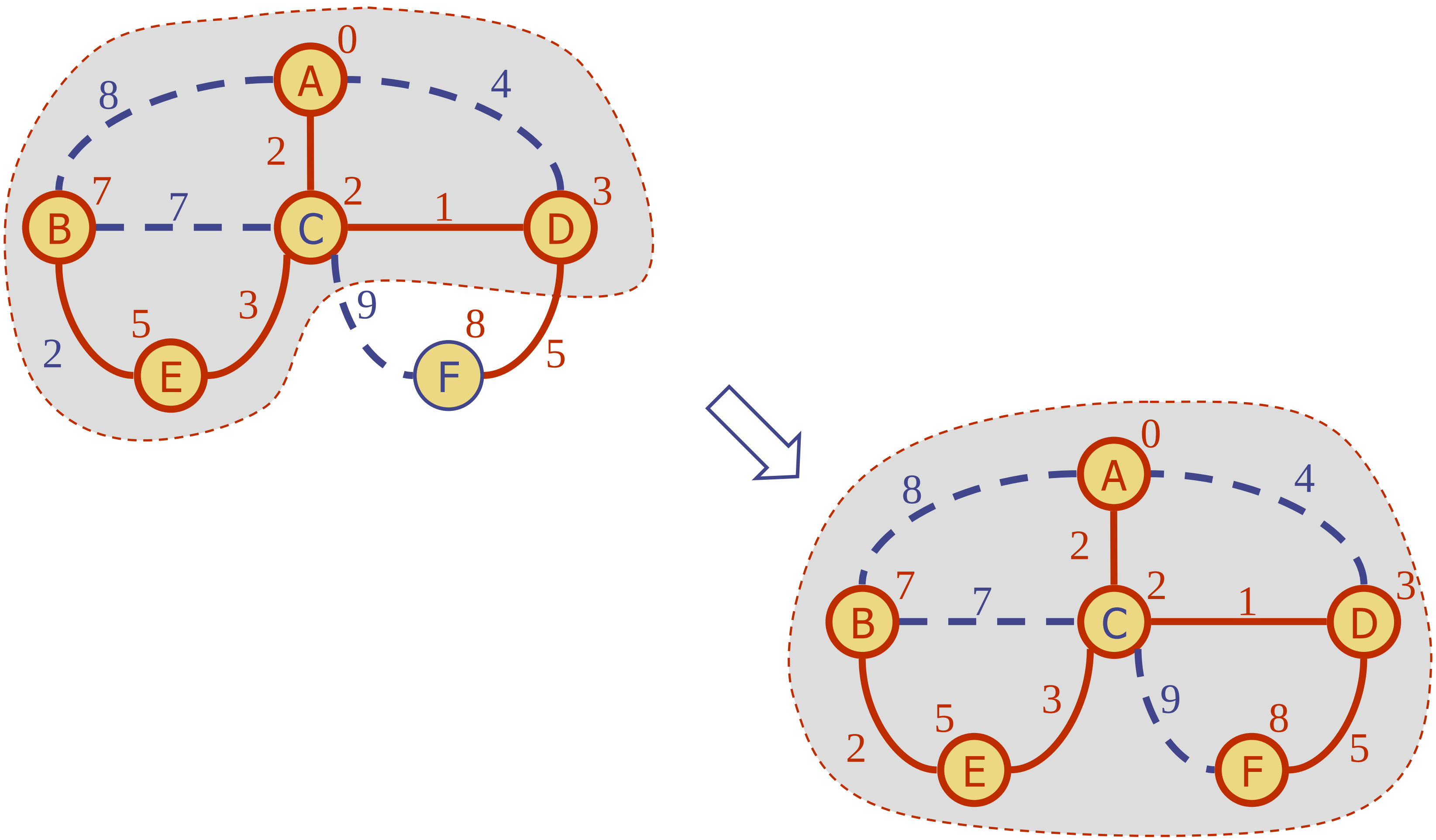
- 逐个更新与云中点邻接的在云外的结点z的标注值d(z)（“云”周边的）
- 将“云”周边邻接的点中距离（标注值）最小的点加入到“云”中
- 直到“云”包含了所有的结点
- *注：这里所谓的“云”其实是结点集V的一个子集。*



举例：观察云的扩张过程以及点的值d(v)的变化过程

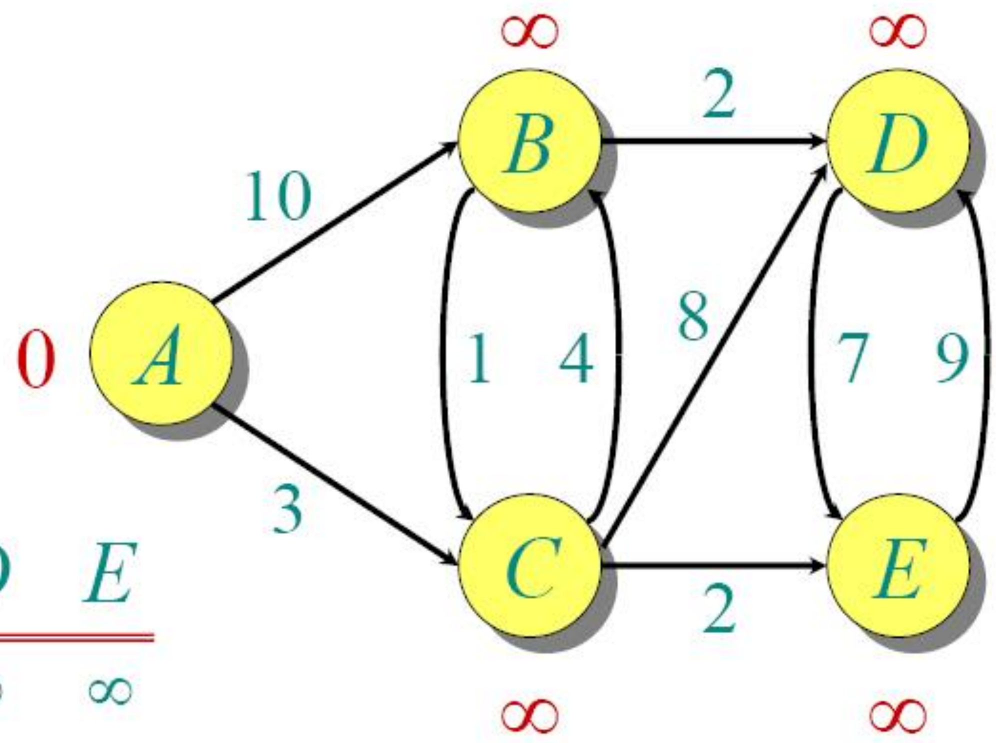


Example (cont.)



Another Dijkstra Animated Example for directed graph （有向图距离）

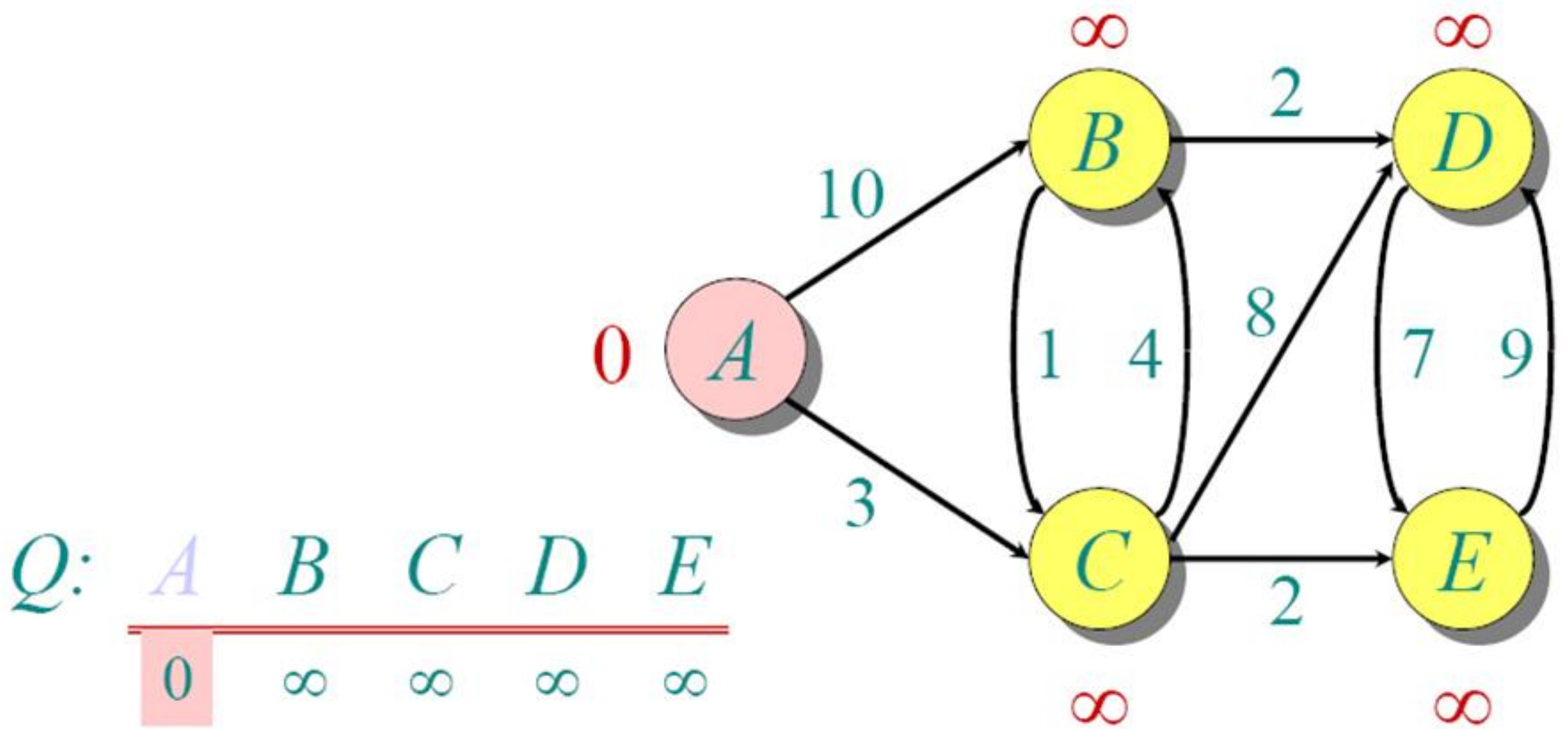
Initialize:



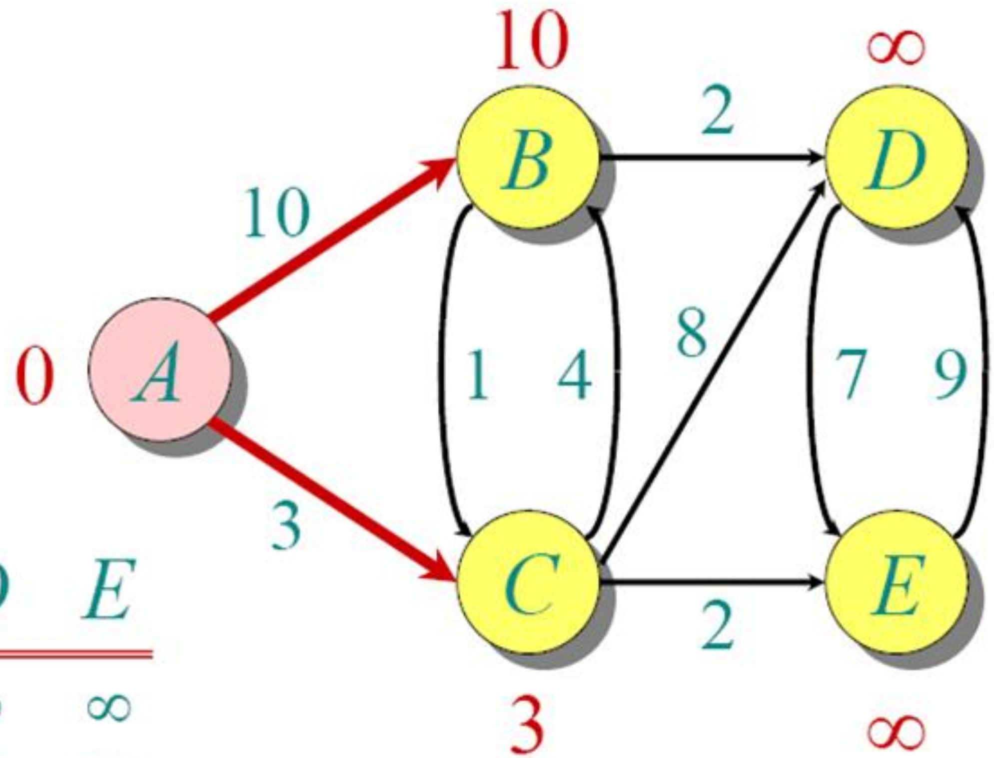
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Dijkstra Animated Example



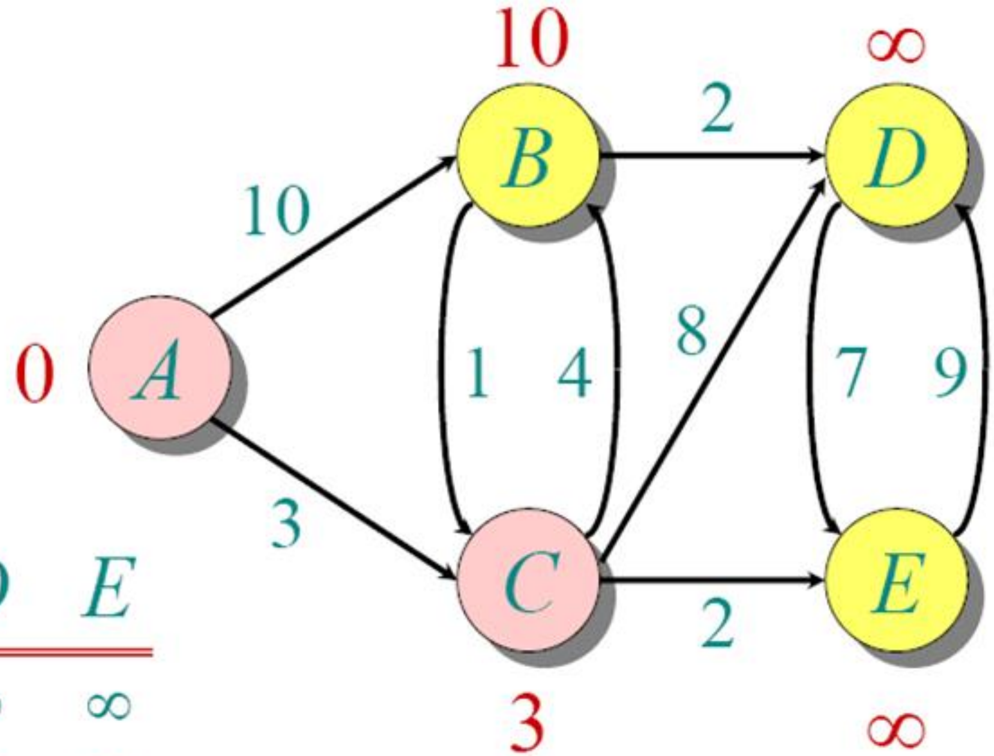
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

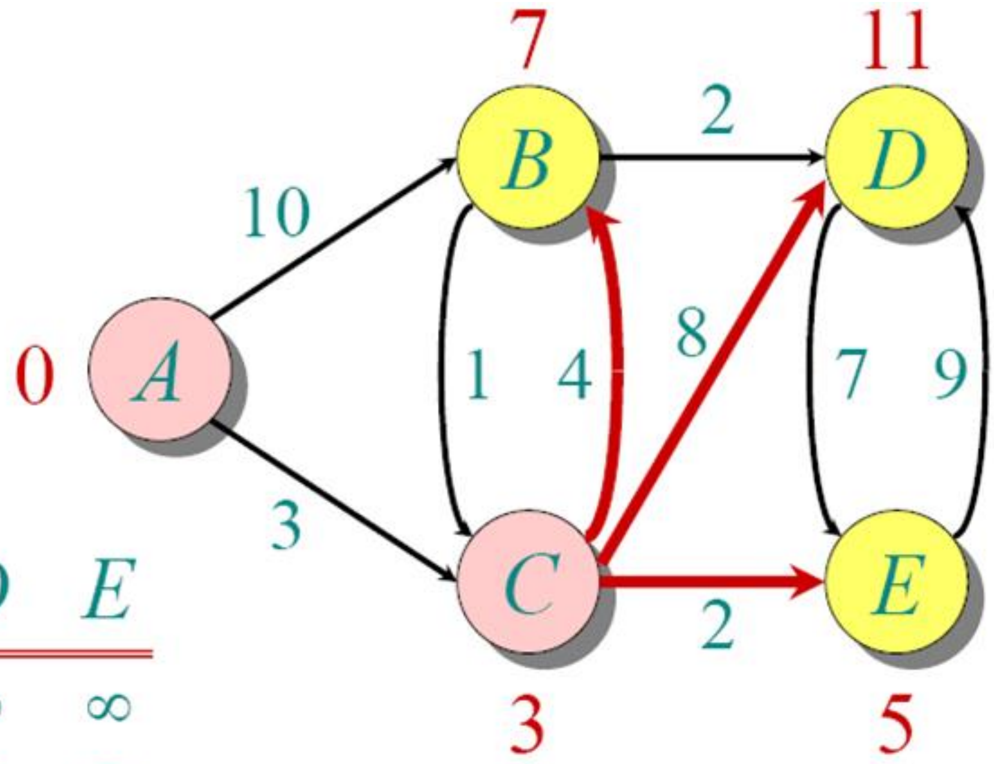
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

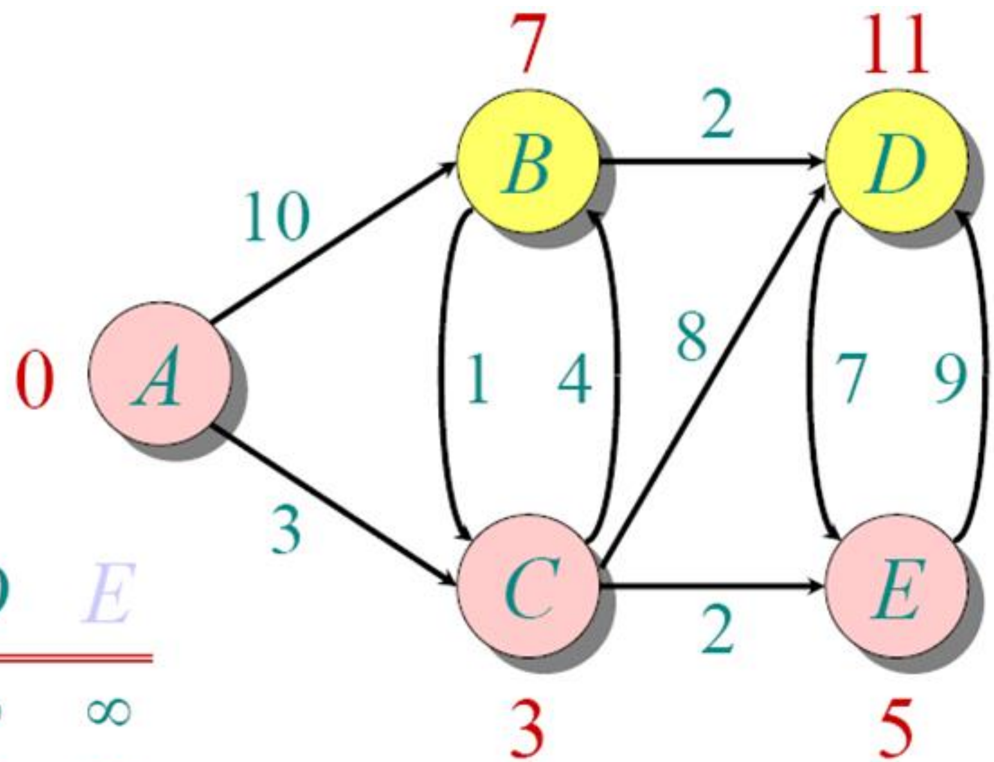
Dijkstra Animated Example



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

S: { *A*, *C* }

Dijkstra Animated Example

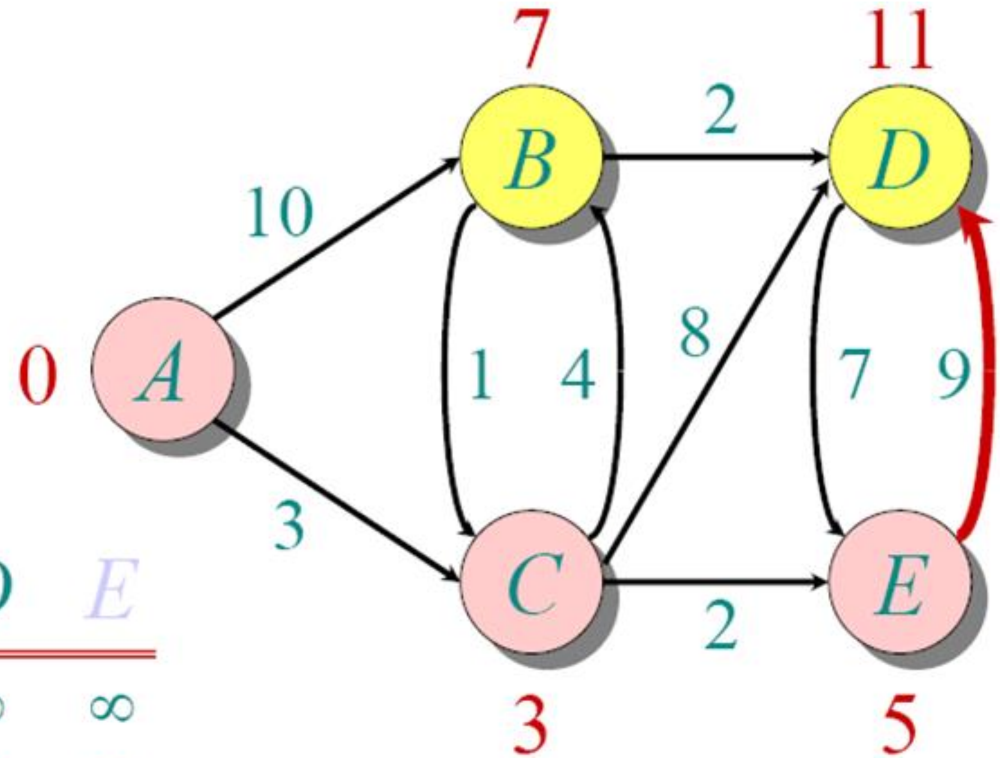


Q:

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

S: { A, C, E }

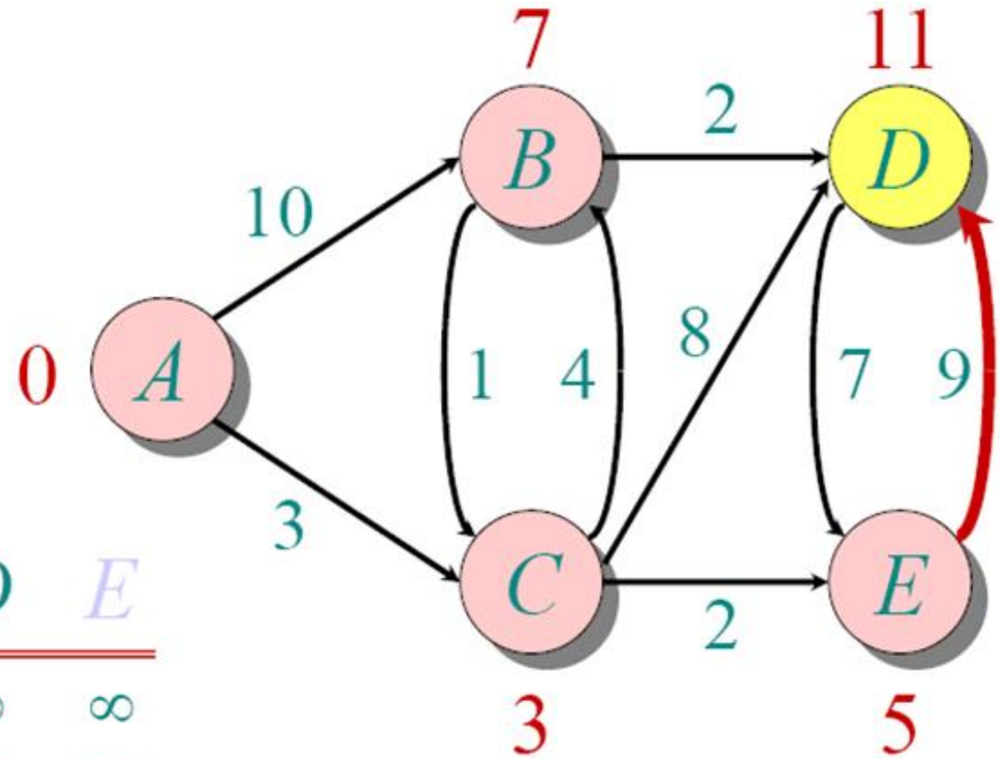
Dijkstra Animated Example



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

S: { A, C, E }

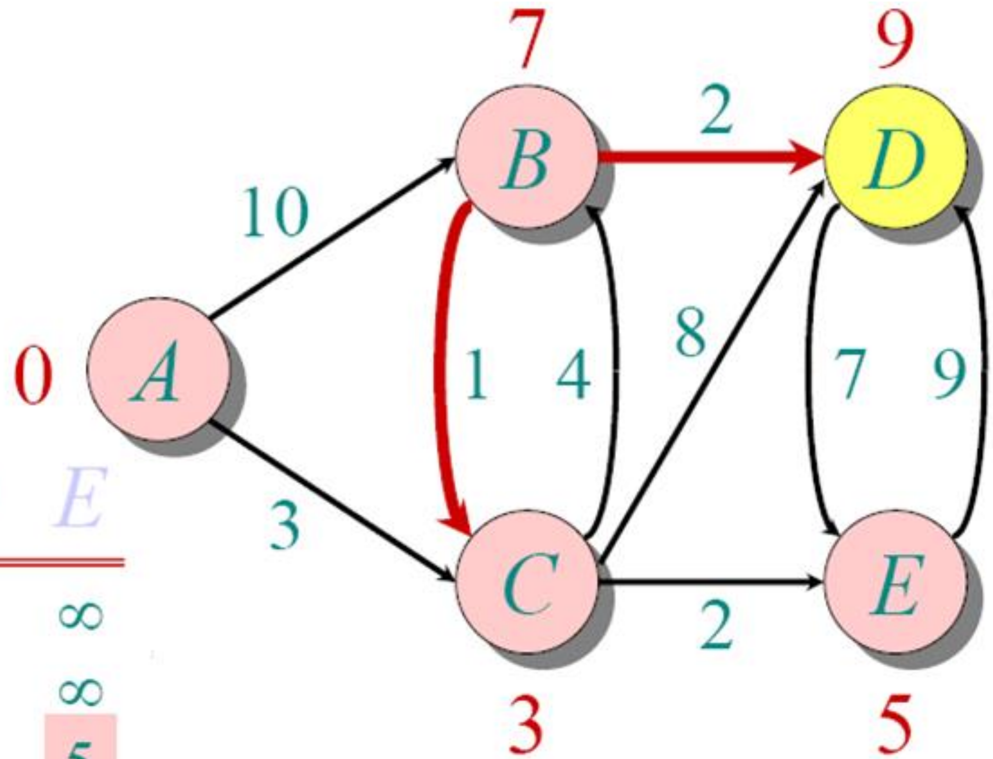
Dijkstra Animated Example



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

S: { A, C, E, B }

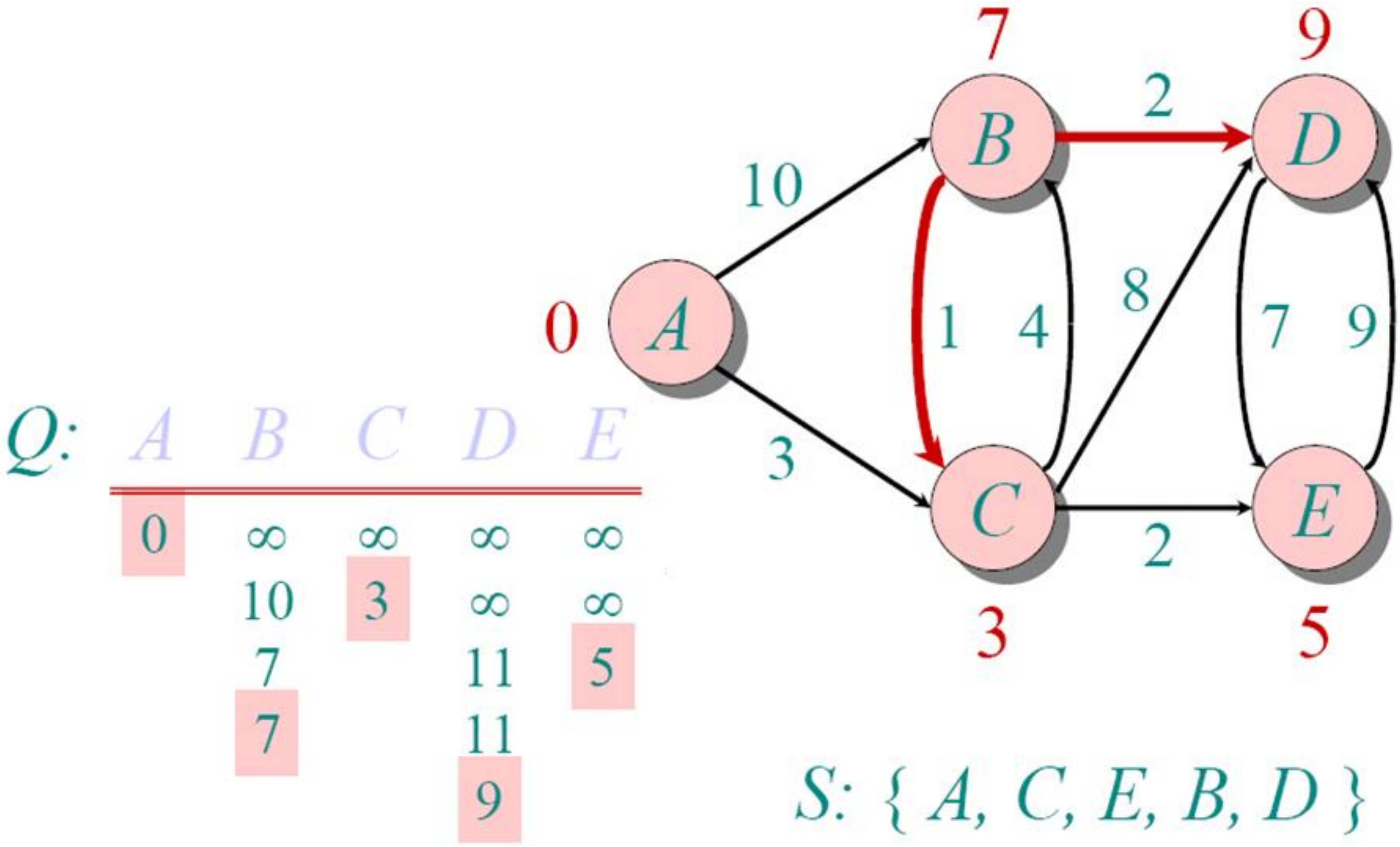
Dijkstra Animated Example



Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	
				9	

S: { A, C, E, B }

Dijkstra Animated Example



Dijkstra 算法每次迭代要做的两件事

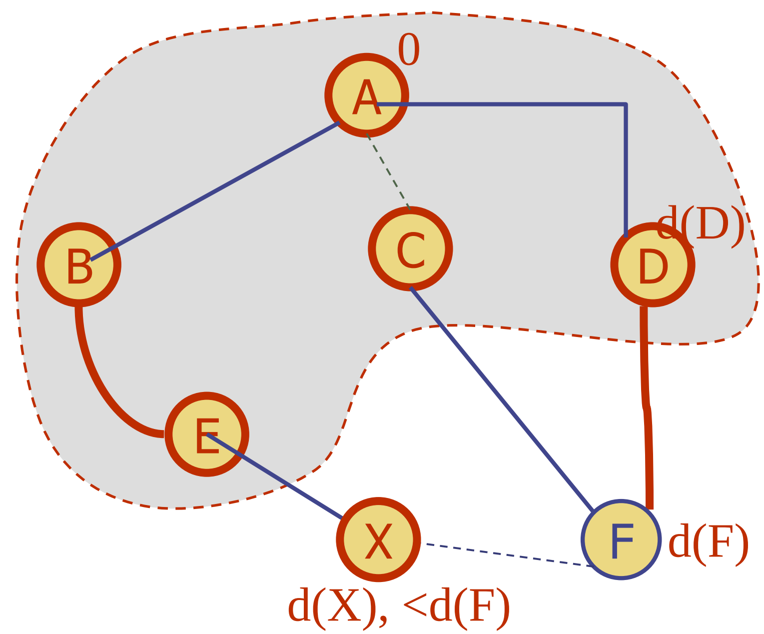
- 1. 从子集S外（“云”外）的所有与S中结点邻接的所有结点中选择标注值d(z)最小的结点u，加入到S中；
- 2. 考察对比与u结点邻接的在S之外的结点的标注值，做可能的修改。

- 注：一个结点z的标注值d(z)，当它≠∞时， 所代表的含义是：如果z在S中，则它是从起点到点z的最短路径的长度，也即距离；如果在S外则说明有某一条从起点到z的路，长度为d(z).这个值的不断修改过程就是寻找更短路的过程，直到找到最短的为止。

Why Dijkstra's Algorithm Works

- Dijkstra's algorithm is based on the greedy method. It adds vertices by increasing distance.

- Suppose it didn't find all shortest distances. Let F be the first wrong vertex the algorithm processed.
- When the previous node, D, on the true shortest path was considered, its distance was correct.
- But the edge (D,F) was **relaxed** at that time!
- Thus, so long as d(F)≥d(D)（非负边）, F's distance cannot be wrong. That is, there is no wrong vertex.

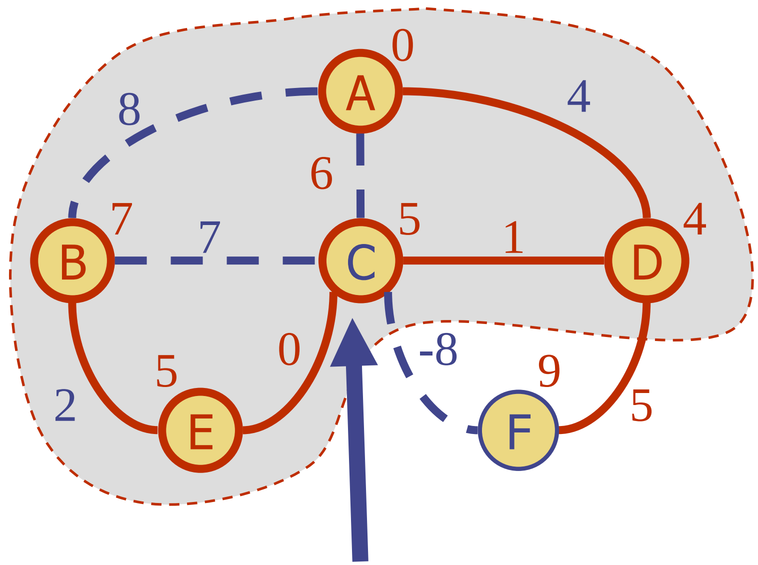


Why It Doesn't Work for Negative-Weight Edges

- Dijkstra's algorithm is based on the greedy method. It adds vertices by increasing distance.

如果一个结点与一条带负权的边关联，被加入到“云”里，就可能导致混乱。

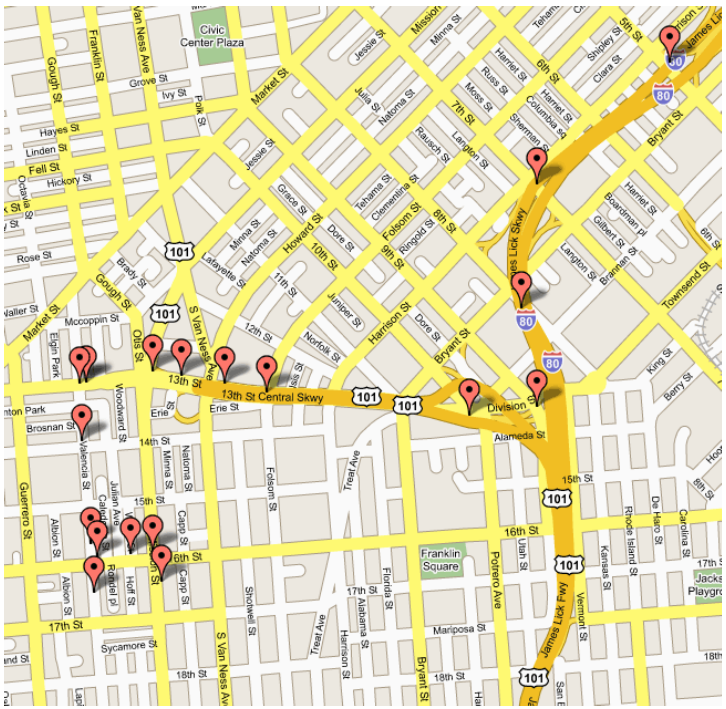
如右图所示：



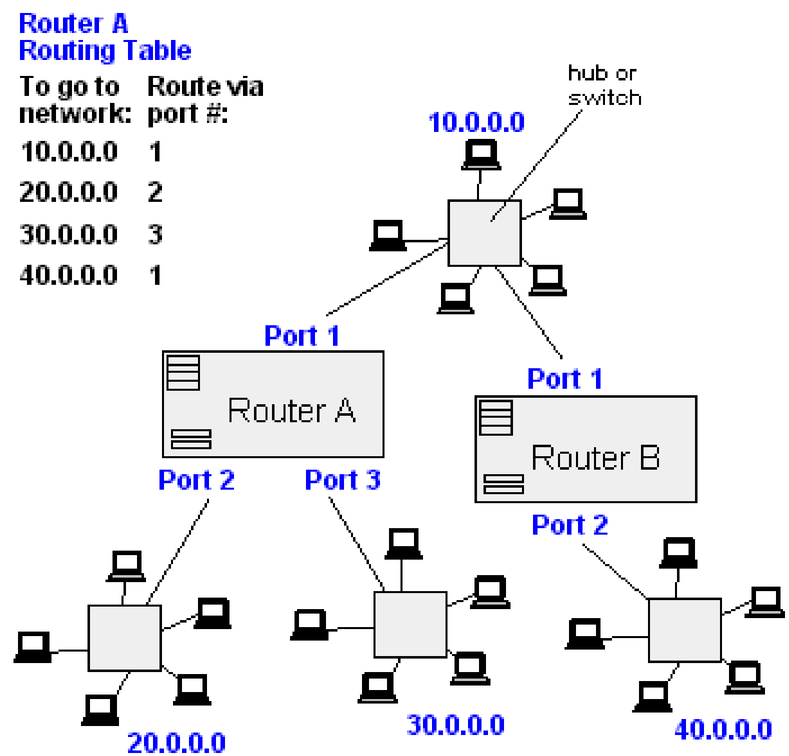
C's true distance is 1, but it is already in the cloud with d(C)=5!

Applications of Dijkstra's Algorithm

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems



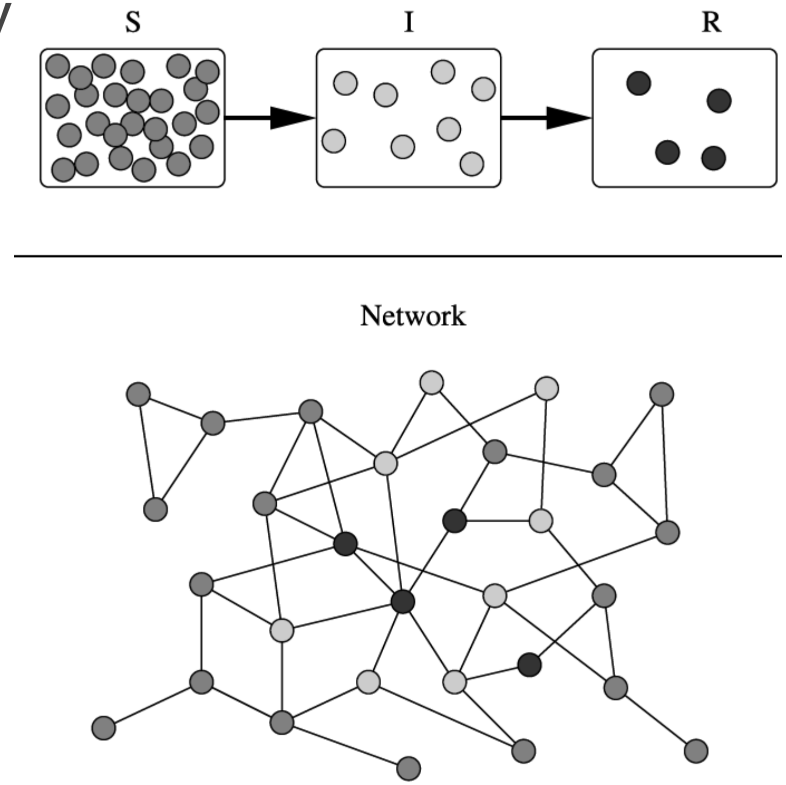
From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



Dijkstra's 算法应用

- One particularly relevant: epidemiology

- Prof. Lauren Meyers (Biology Dept.) uses networks to model the spread of infectious diseases and design prevention and response strategies. (传染疾病防控)



- Vertices represent individuals, and edges their possible contacts. It is useful to calculate how a particular individual is connected to others.

- Knowing the shortest path lengths to other individuals can be a relevant indicator of the potential of a particular individual to infect others.

- How to solve the shortest path problem when the graph has negative edges?

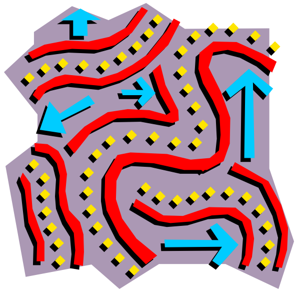
- Bellman-Ford Algorithm （求含负权图的单源最短路径算法，效率很低，但代码很容易写）

(http://blog.csdn.net/xu3737284/article/details/8973615)

- DAG-based Algorithm (http://blog.csdn.net/wall_f/article/details/8204747)

注：这两个算法自己有兴趣的话，上网去搜索学习

Bellman-Ford Algorithm

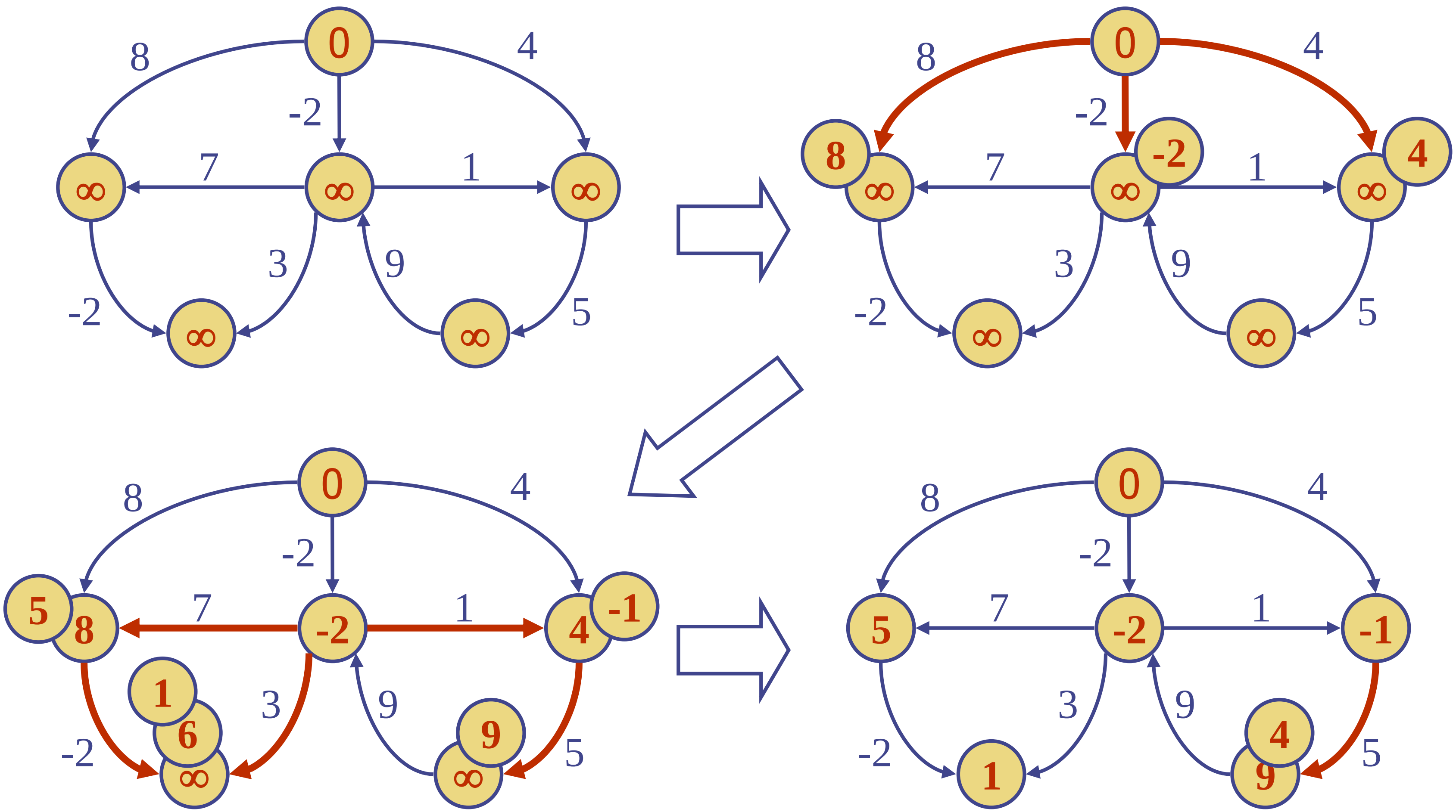


- Works even with negative-weight edges
- Must assume directed edges (有向无环) (for otherwise we would have negative-weight cycles)
- Iteration i finds all shortest paths that use i edges.
- Running time: O(nm).
- Can be extended to detect a negative-weight cycle if it exists
  - How?

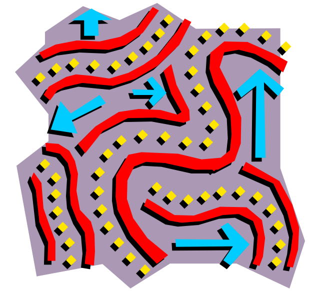
Algorithm *BellmanFord*(G, s)
  for all $v \in G.vertices()$
    if $v = s$
      setDistance(v, 0)
    else
      setDistance(v, ∞)
  for $i \leftarrow 1$ **to** $n\text{-}1$ **do**
    for each $e \in G.edges()$
      { relax edge ***e*** }
      $u \leftarrow G.origin(e)$
      $z \leftarrow G.opposite(u,e)$
      $r \leftarrow getDistance(u) + weight(e)$
      if $r < getDistance(z)$
        setDistance(z,r)

Bellman-Ford Example

Nodes are labeled with their d(v) values



DAG-based Algorithm

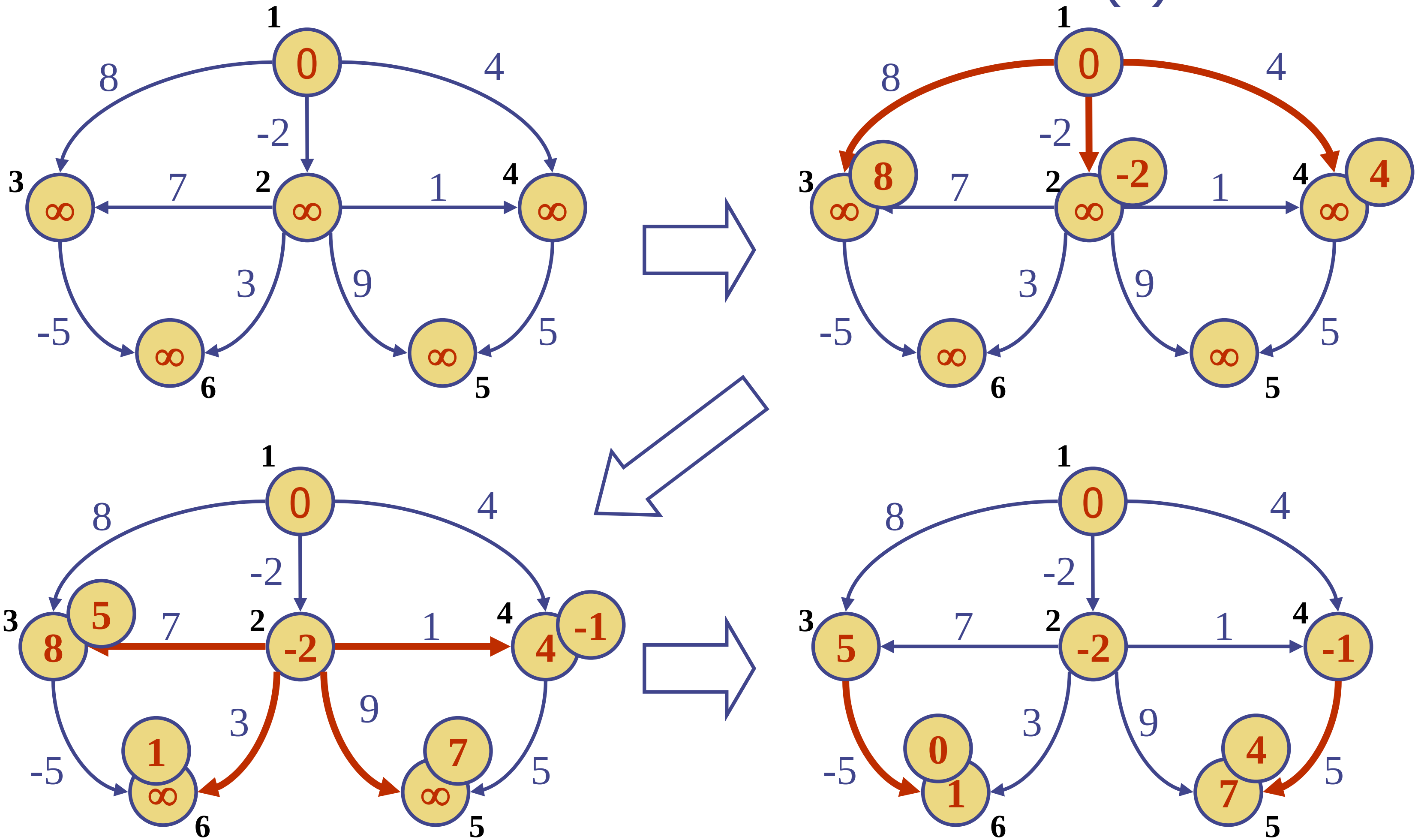


- Works even with negative-weight edges
- Uses topological order
- Doesn't use any fancy data structures
- Is much faster than Dijkstra's algorithm
- Running time: O(n+m).

```
Algorithm DagDistances(G, s)
    for all  v ∈ G.vertices()
        if  v = s
            setDistance(v, 0)
        else
            setDistance(v, ∞)
    Perform a topological sort of the vertices
    for u ← 1 to n do    {in topological order}
        for each  e ∈ G.outEdges(u)
            { relax edge e }
            z ← G.opposite(u,e)
            r ← getDistance(u) + weight(e)
            if  r < getDistance(z)
                setDistance(z,r)
```

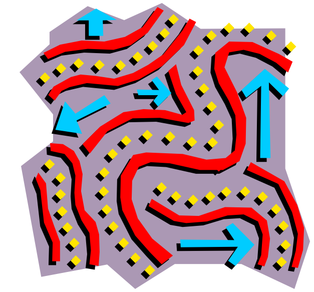
DAG Example

Nodes are labeled with their d(v) values



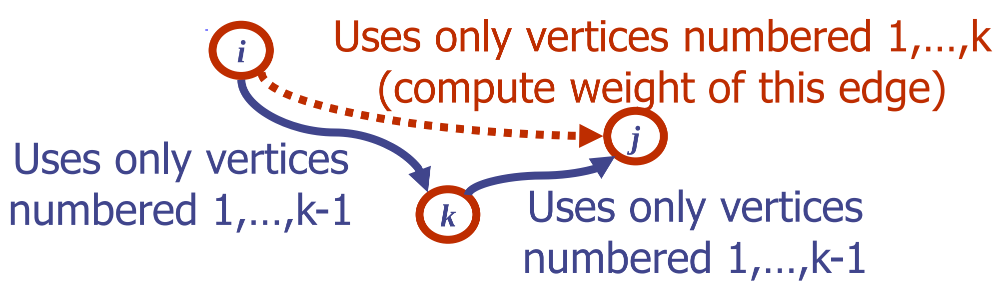
(two steps)

All-Pairs Shortest Paths



- Find the distance between every pair of vertices in a weighted directed graph G.
- We can make n calls to Dijkstra's algorithm (if no negative edges), which takes O(nmlog n) time.
- Likewise, n calls to Bellman-Ford would take $O(n^2m)$ time.
- We can achieve $O(n^3)$ time using dynamic programming (similar to the Floyd-Warshall algorithm).

```
Algorithm AllPair(G) {assumes vertices 1,…,n}
 for all vertex pairs (i,j)
   if i = j
      D0[i,i] ← 0
   else if (i,j) is an edge in G
      D0[i,j] ← weight of edge (i,j)
   else
      D0[i,j] ← + ∞
 for k ← 1 to n do
   for i ← 1 to n do
     for j ← 1 to n do
       Dk[i,j] ← min{Dk-1[i,j], Dk-1[i,k]+Dk-1[k,j]}
 return Dn
```



思考Question

- 对一个普通无向连通图，For a connected simple undirected graph (non-weighted), can you design an algorithm to calculate the distance between a start vertex v_0 to any other vertex v?

- *Solution: you can set weight 1 to all edges of the graph*

Traveling Salesman Problem

- Introduction to Traveling Salesman Problem
- 某售货员要到若干城市去推销商品，已知各城市之间的路程(或旅费)。他要选定一条从驻地出发，经过每个城市一次，最后回到驻地的路线，使总的路程(或总旅费)最小。
- 数学化的问题：在带权完全无向图里，求访问每个顶点一次只一次，且最后返回出发点，总权最小的路。这实质是求完全图里总权最小的哈密尔顿回路。
- 这是一个NP-问题。

练习

- 6.6节 T1， T2