

# 最短路径算法

6. 图

优先级搜索

邓俊辉

deng@tsinghua.edu.cn

## 通用算法

❖ 各种遍历算法的区别，仅在于选取顶点进行访问的次序

**广度** / **深度**：优先访问与**更早** / **更晚**被发现的顶点相邻接者

    ...

❖ 不同的遍历算法，取决于顶点的**选取策略**

❖ 不同的顶点选取策略，取决于存放和提供顶点的**数据结构——Bag**

❖ 此类结构，为每个顶点v维护一个**优先级数**  $\text{priority}(v)$

    每个顶点都有**初始**优先级数；并可能随算法的推进而**调整**

❖ 通常的习惯是，优先级数越**大/小**，优先级越**低/高**

    特别地， $\text{priority}(v) == \text{INT\_MAX}$ ，意味着v的优先级最低

## 统一框架

❖ template <typename Tv, typename Te> //顶点类型、边类型

template <typename PU> //优先级更新器（函数对象）

```
void Graph<Tv, Te>::pfs( int s, PU prioUpdater ) { //PU的策略，因算法而异
    priority(s) = 0; status(s) = VISITED; parent(s) = -1; //起点s加至PFS树中
    while (1) { //将下一顶点和边加至PFS树中
        /* ... 依次引入n - 1个顶点（和n - 1条边） ... */
    } //while
} //如何推广至非连通图?
```

## 统一框架

```
❖ while (1) { //依次引入n - 1个顶点 (和n - 1条边)
    for ( int w = firstNbr(s); -1 < w; w = nextNbr(s, w) ) //对s各邻居w
        prioUpdater( this, s, w ); //更新顶点w的优先级及其父顶点
    for ( int shortest = INT_MAX, w = 0; w < n; w++ )
        if ( UNDISCOVERED == status(w) ) //从尚未加入遍历树的顶点中
            if ( shortest > priority(w) ) //选出下一个
                { shortest = priority(w); s = w; } //优先级最高的顶点s
        if ( VISITED == status(s) ) break; //直至所有顶点均已加入
        status(s) = VISITED; type( parent(s), s ) = TREE; //将s加入遍历树
    } //while
```

- ❖ 执行时间主要消耗于内、外两重循环；其中两个内循环前、后并列
- ❖ 前一内循环的累计执行时间： 若采用邻接矩阵，为 $\Theta(n^2)$ ；若采用邻接表，为 $\Theta(n + e)$   
后一循环中，优先级更新的次数呈算术级数变化{ n, n - 1, …, 2, 1 }，累计为 $\Theta(n^2)$   
两项合计，为 $\Theta(n^2)$
- ❖ 后面将会看到：若采用优先队列，以上两项将分别是 $\Theta(e \log n)$ 和 $\Theta(n \log n)$   
两项合计，为 $\Theta((e + n) * \log n)$
- ❖ 这是很大的改进——尽管对于稠密图而言，反而是倒退 //已有接近于 $\Theta(e + n \log n)$ 的算法
- ❖ 基于这个统一框架，如何解决具体的应用问题…

**6. 图**

**最小支撑树**

**邓俊辉**

**deng@tsinghua.edu.cn**

## 最小 + 支撑 + 树

❖ 连通网络  $N = (V; E)$  的子图  $T = (V; F)$

1) 支撑/spanning

覆盖  $N$  中所有顶点

2) 树/tree

连通且无环,  $|V| = |F| + 1$

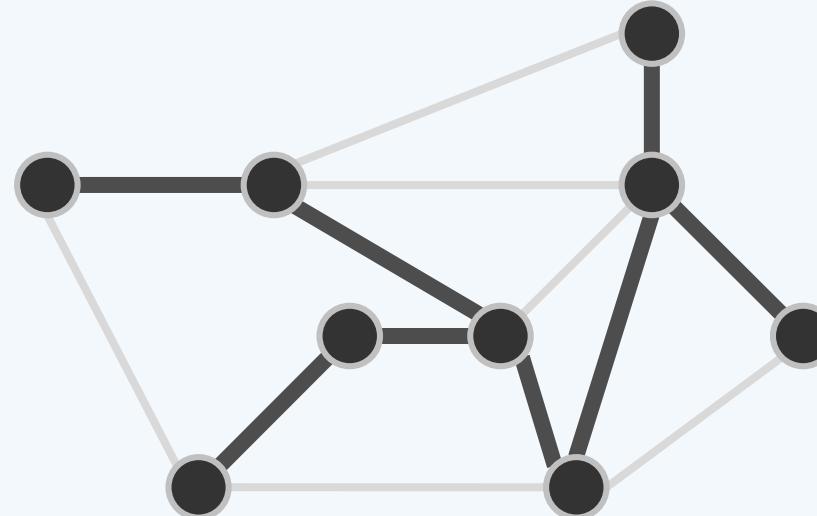
加边出单环, 再删同环边即恢复为树

删边不连通, 再加联接边即恢复为树

不难验证, 同一网络的支撑树不唯一

3) 最小/minimum

各边总权重  $wt(T) = \sum_{e \in F} wt(e)$  达到最小



❖ 谁感兴趣?

电信公司、网络设计师、VLSI布线算法设计师、...

❖ 为何重要?

应用中常见的共性问题，也是很多优化问题的**基本模型**

自身可**有效计算**

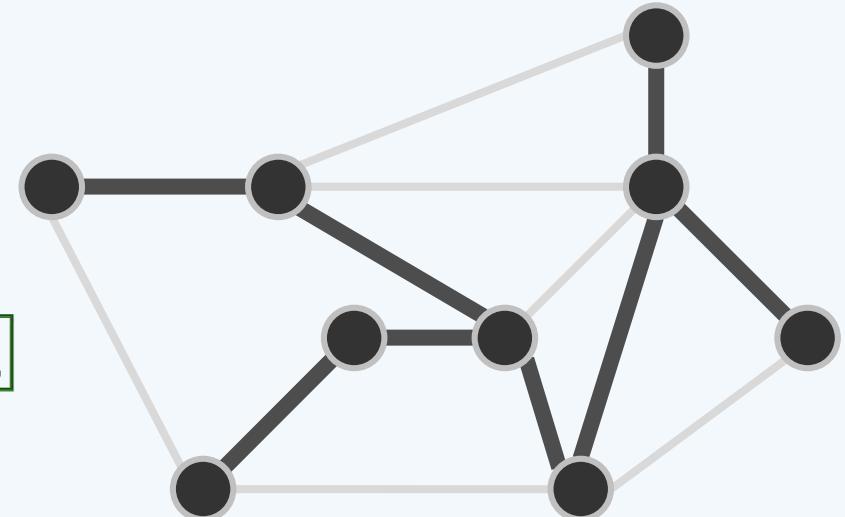
为许多NP问题提供足够好的**近似解** //比如，Euclidean TSP

❖ 算法

Boruvka-1926, Jarnik-1930/Prim-1956, Kruskal-1956, ...

Karger-Klein-Tarjan-1995, Chazelle-2000

...是否存在 $\theta(n + e)$ 算法?

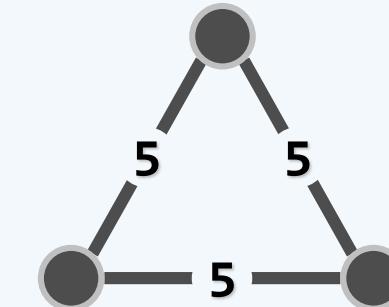
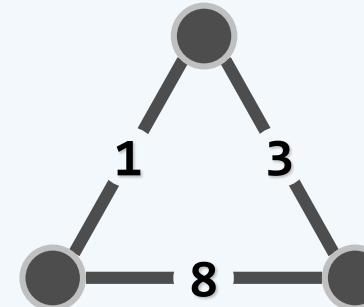
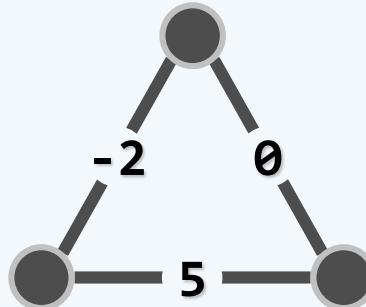


## 退化

❖ 权值必须是正数?

允许为零, 会有什么影响?

允许为负数呢?



❖ 所有支撑树所含的边数, 必然相等

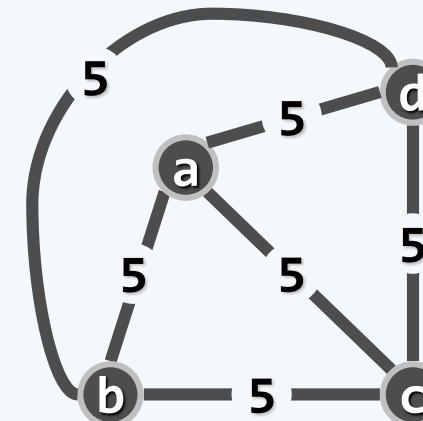
故可统一调整: `increase( 1 - findMin() )`

❖ The minimum?

A minimal

同一网络N可能有多棵MST

The minimum! 可强制消除歧义...



❖ 合成数 (composite number) : (  $w(u, v)$ ,  $\min(u, v)$ ,  $\max(u, v)$  )

$5ab < 5ac < 5ad < 5bc < 5bd < 5cd$

## 蛮力算法

❖ 枚举出N的所有支撑树，从中找出代价最小者

❖ 这一策略是否可行，取决于...

❖ n个互异顶点组成的图，可能有多少棵支撑树？

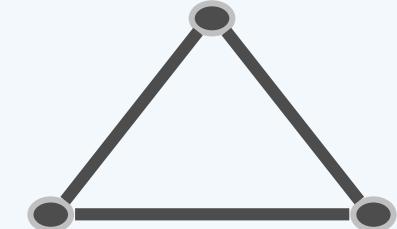
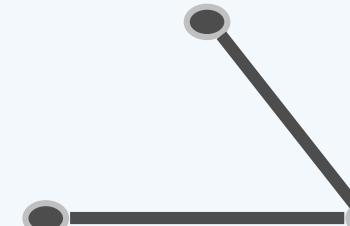
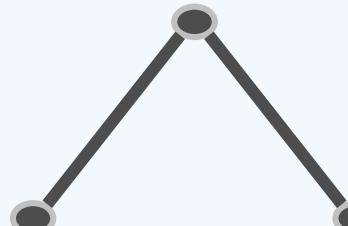
$n = 1$  1

$n = 2$  1

$n = 3$  3

$n = 4$  16

... ...



❖ Cayley公式：联接n个互异顶点的树共有 $n^{n-2}$ 棵；或等价地，完全图 $K_n$ 有 $n^{n-2}$ 棵支撑树

❖ 如何高效地构造MST呢？后续算法部分讲述

6. 图

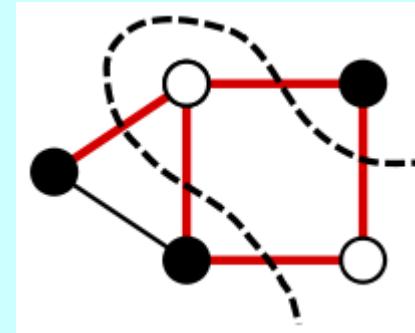
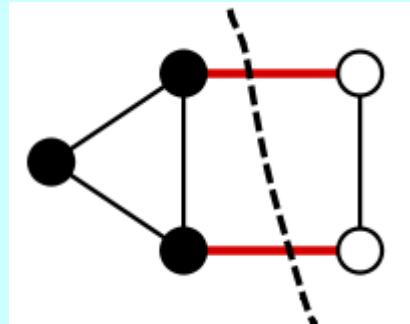
Dijkstra算法

邓俊辉

deng@tsinghua.edu.cn

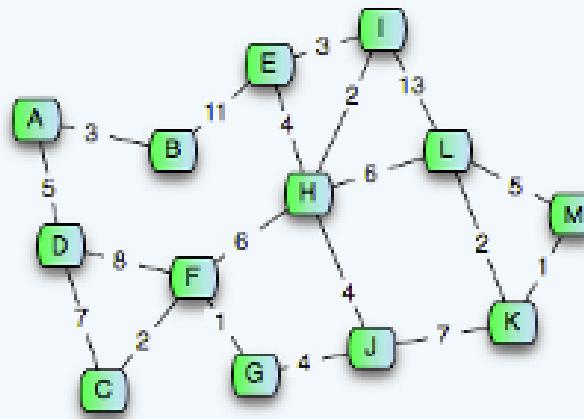
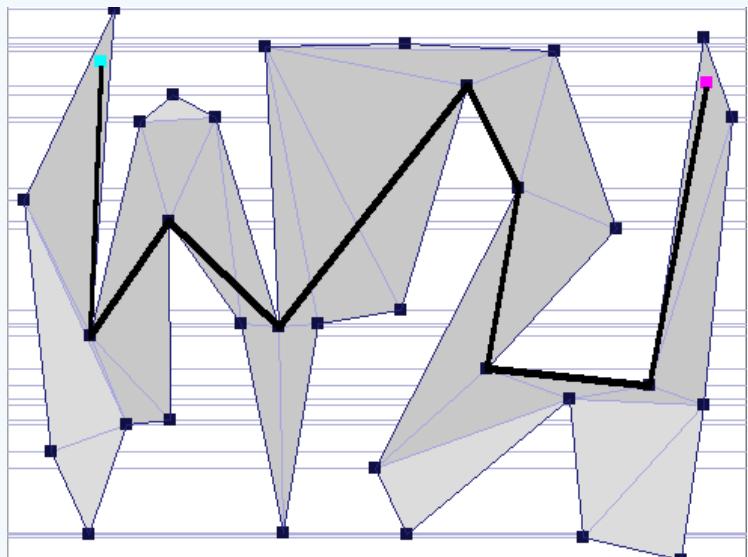
# 预备知识

- In graph theory, a **cut** (割) is a partition of the vertices of a graph into two disjoint subsets.
- Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. These edges are said to **cross** the cut.
- 图 $G=(V,E)$ 中， $V$ 的任一非平凡子集 $U$ 及其补集 $V \setminus U$ 都构成 $G$ 的一个割。记作 $(U: V \setminus U)$ 。若边 $(u,v)$ 满足 $u \in U, v \in V \setminus U$ ，则称跨越边(crossing edge)。



## 问题描述

- ◆ 给定：连通有向图G及其中的顶点u和v  
找到：从u到v的最短路径及其长度
- ◆ 旅游者：最经济的出行路线  
路由器：最快地将数据包传送到目标位置  
路径规划：多边形区域内的自主机器人



## 问题分类

### ❖ 按照图的类型

无权图/等权图: BFS

带权有向图 //负权值呢?

### ❖ 单源点到各顶点的最短路径

Single-source shortest paths

给定顶点x, 计算x到其余各个顶点的最短路径及长度

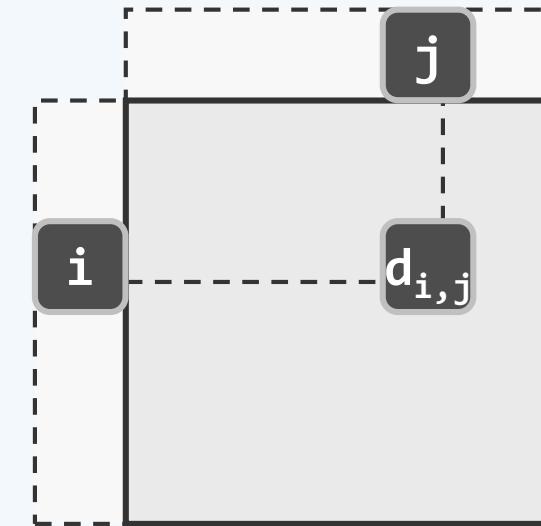
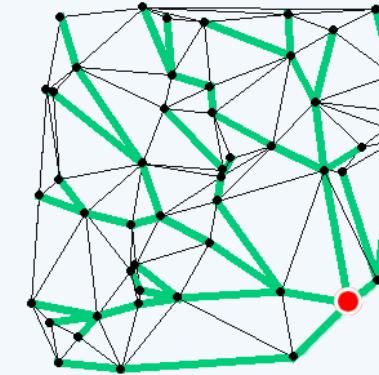
E. Dijkstra, 1959

### ❖ 所有顶点对之间的最短路径

All shortest paths

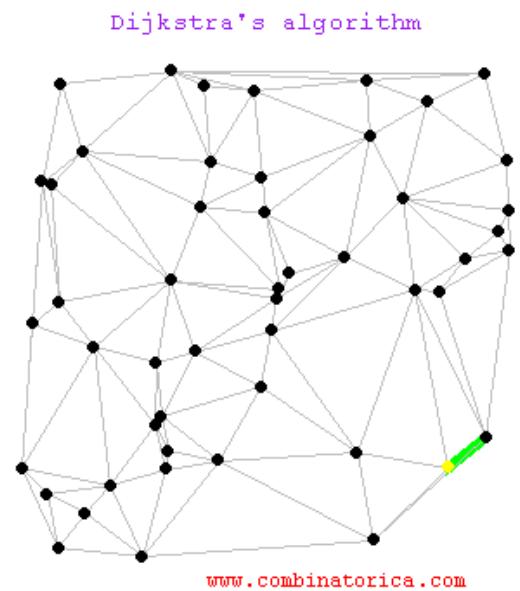
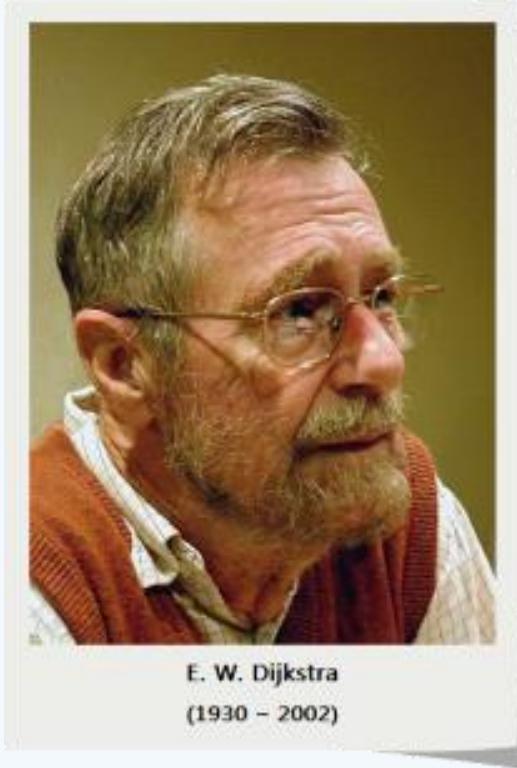
找出每对顶点i和j之间的最短路径及长度

Floyd-Warshall, 1962



# E. W. Dijkstra

❖ Turing Award, 1972



Numerische Mathematik 1, 269–271 (1959)

## A Note on Two Problems in Connexion with Graphs

By

E. W. DIJKSTRA

We consider  $n$  points (nodes), some or all pairs of which are connected by a branch; the length of each branch is given. We restrict ourselves to the case where at least one path exists between any two nodes. We now consider two problems.

**Problem 1.** Construct the tree of minimum total length between the  $n$  nodes. (A tree is a graph with one and only one path between every two nodes.)

In the course of the construction that we present here, the branches are subdivided into three sets:

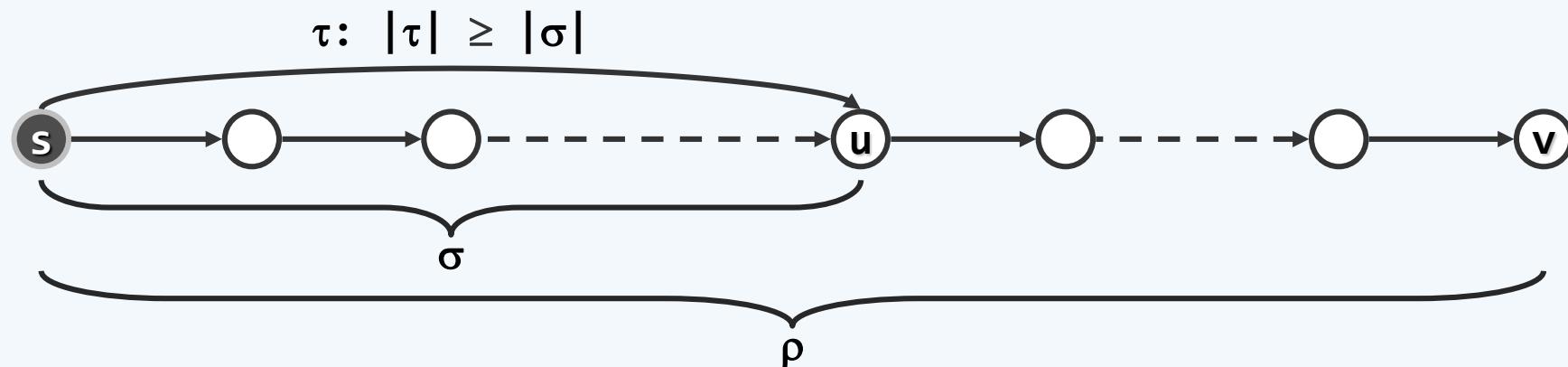
- I. the branches definitely assigned to the tree under construction (they will form a subtree);
- II. the branches from which the next branch to be added to set I, will be selected;
- III. the remaining branches (rejected or not yet considered).

1) 连通图中， $s$ 到每个顶点都有（至少）一条最短路径

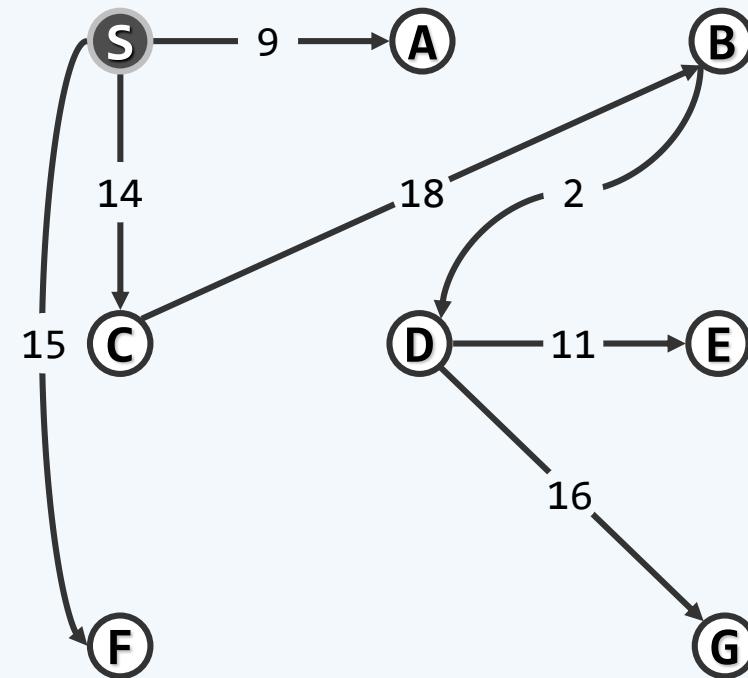
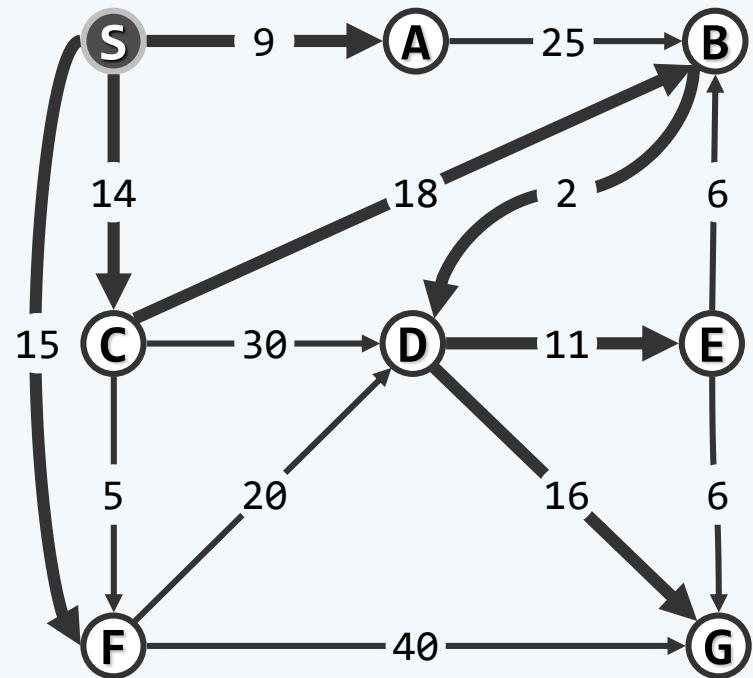
// 非退化假设：每个顶点对应的最短路径唯一

2) 就同一起点 $s$ 而言，任何最短路径的前缀，也是一条最短路径

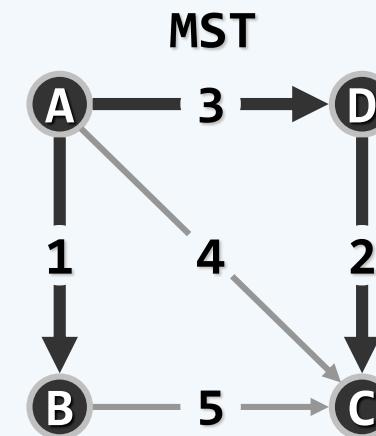
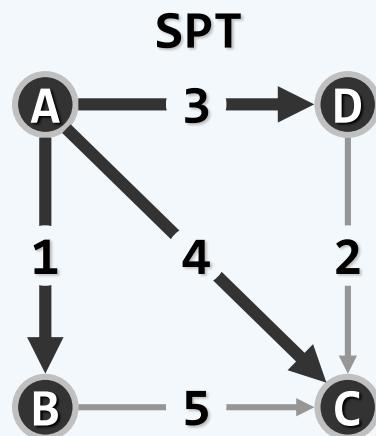
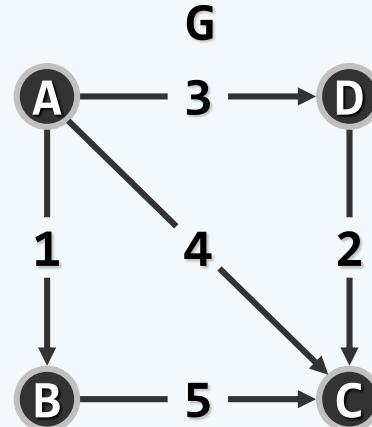
3) 就同一起点 $s$ 而言，所有最短路径的并，不含回路



❖ 因此，所有最短路径的并，构成一棵树（shortest path tree）



**SPT  $\neq$  MST**



$u_1$

❖ 按照到  $s$  的最短距离，对其余的顶点排序

$$\text{dist}(s, u_1) \leq \text{dist}(s, u_2) \leq \dots \leq \text{dist}(s, u_{n-1})$$

❖ 最短距离最短者  $u_1 = ?$

❖ 沿任一最短路径，各顶点到  $s$  的最短距离单调变化

❖  $u_1$  必与  $s$  直接相连

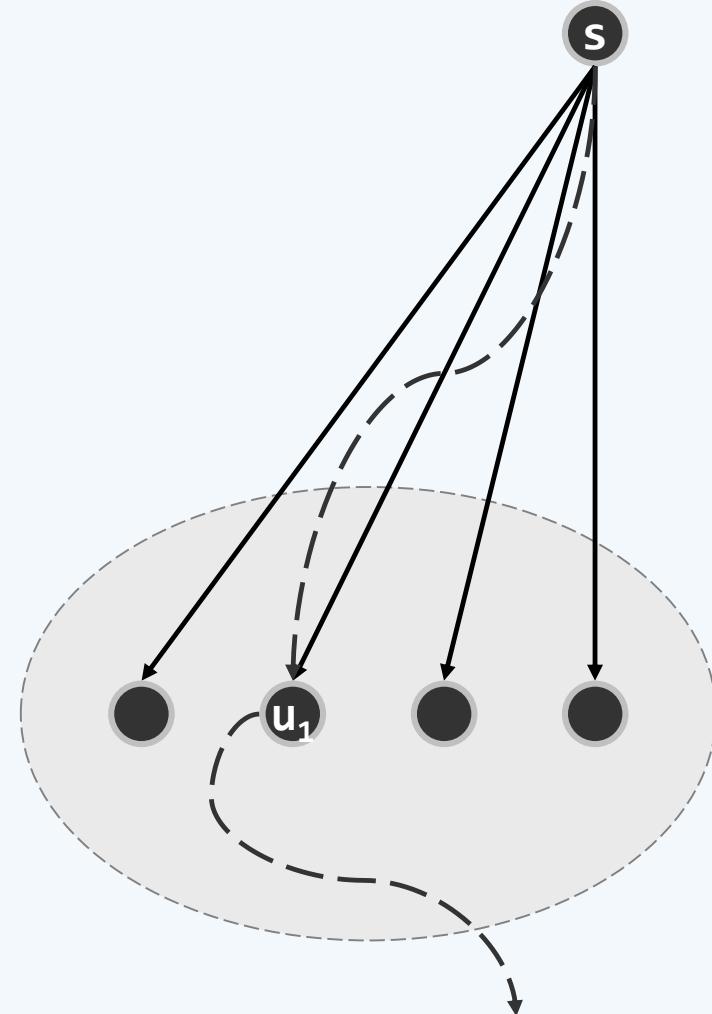
$$\text{dist}(s, s_1) = w(s, s_1) < \infty$$

❖  $\forall u \neq s,$

$$w(s, u) < \infty \text{ 仅当 } w(s, u_1) \leq w(s, u)$$

❖ 为找到  $u_1$ ，只需

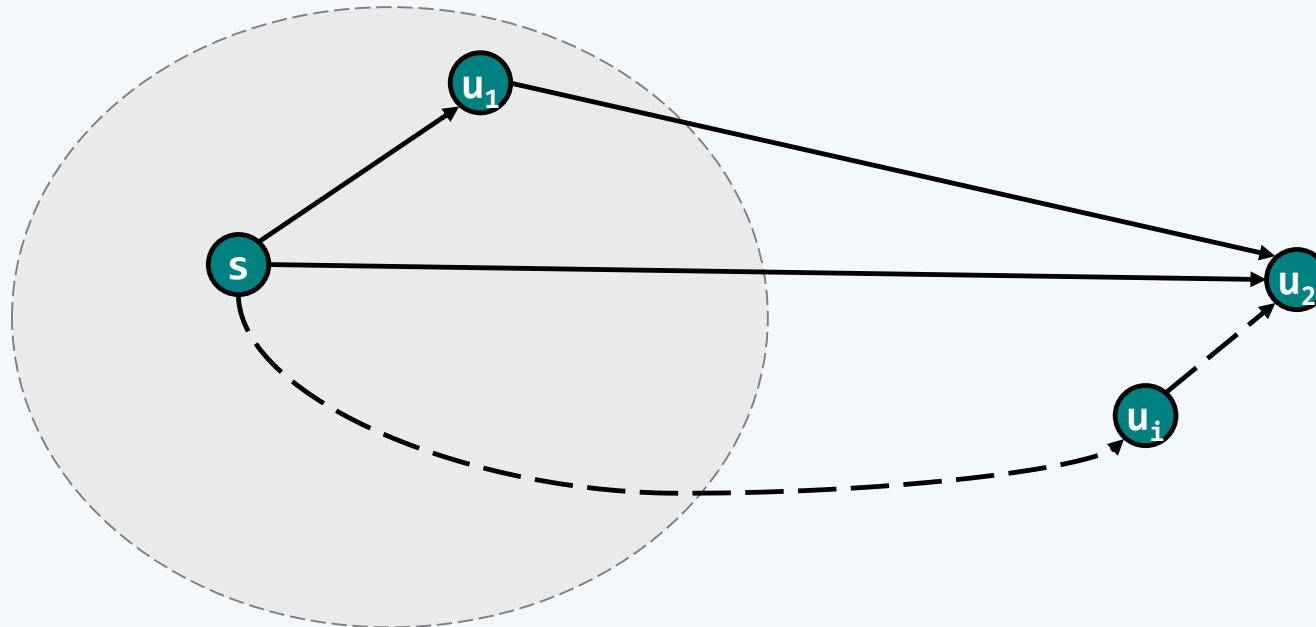
在与  $s$  关联的各顶点中，找到对应边权值最小者



$u_2$

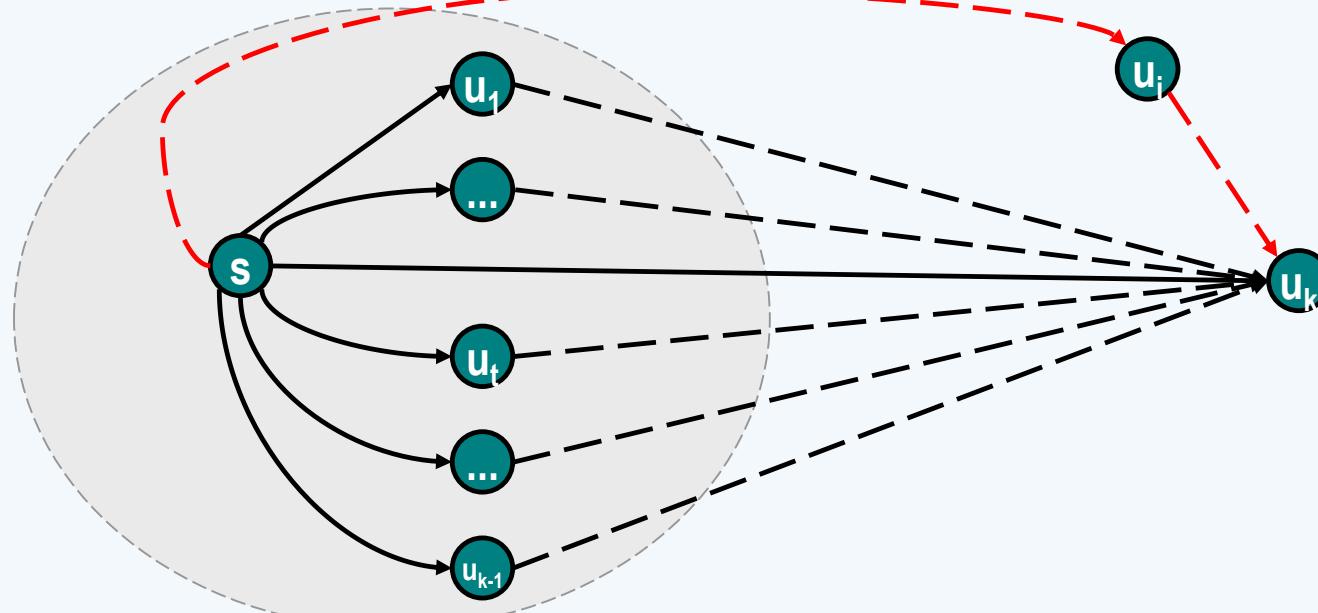
❖ 最短距离次小的顶点 $u_2 = ?$

❖  $\text{dist}(s, u_2) = \min\{ w(s, u_2), \text{dist}(s, u_1) + w(u_1, u_2) \}$



$u_k$

- ❖  $u_3 = ?, u_4 = ?, \dots, u_k = ?$
- ❖ 三角不等式:  $\text{dist}(s, v) \leq \text{dist}(s, u) + w(u, v)$
- ❖ 若记  $u_0 = s$ , 则有:  $\text{dist}(s, u_k) = \min\{ \text{dist}(s, u_i) + w(u_i, u_k) \mid 0 \leq i < k \}$
- ❖ 算法?



## 算法

❖ 从  $T_1 = (\{v_1\}; \emptyset)$  开始，逐步构造  $T_2, T_3, \dots, T_n$ ，其中

$$v_1 = s$$

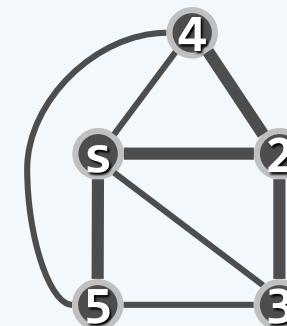
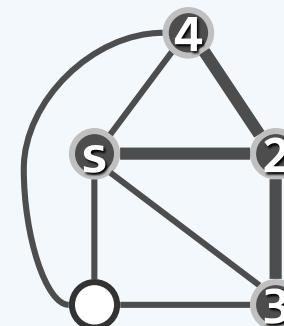
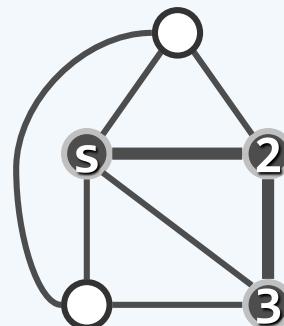
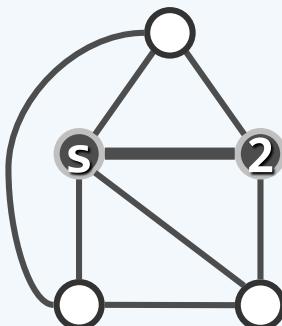
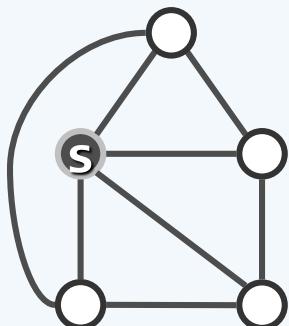
$$T_k = (V_k; E_k), |V_k| = k, |E_k| = k-1, V_k \subset V_{k+1}$$

❖ 由以上分析，为由  $T_k$  构造  $T_{k+1}$ ，只需

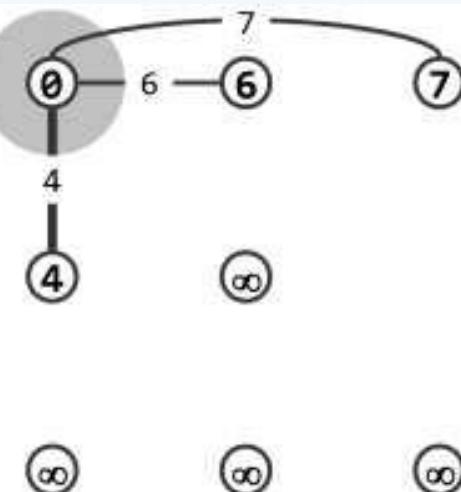
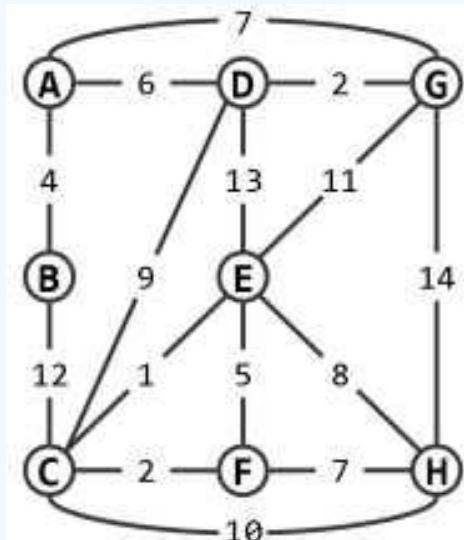
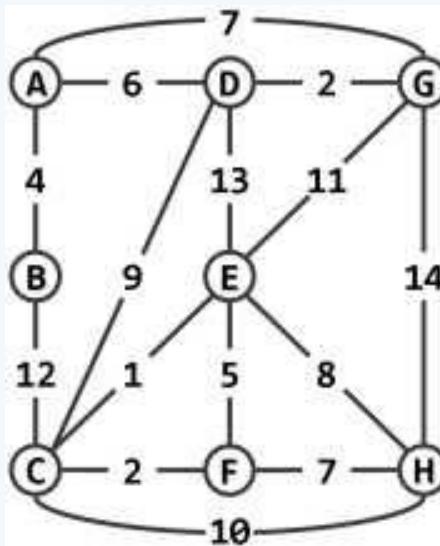
将  $(V_k : V \setminus V_k)$  视作原图的一个割

在该割的所有跨边中，找出极近者  $e_k = (v_k, u_k)$  ( $u_k$  到  $s$  距离极近)

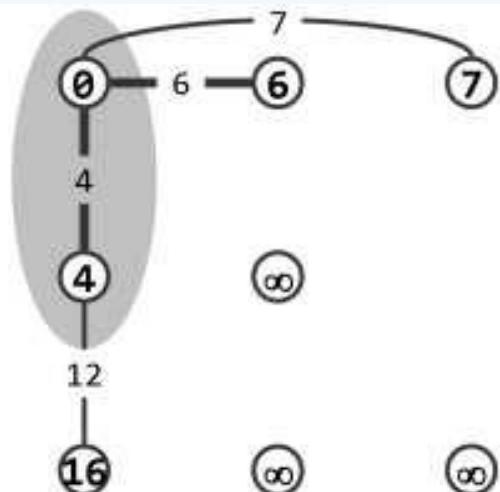
$$\text{令 } T_{k+1} = (V_{k+1}; E_{k+1}) = (V_k \cup \{u_k\}; E_k \cup \{e_k\})$$



## 实例

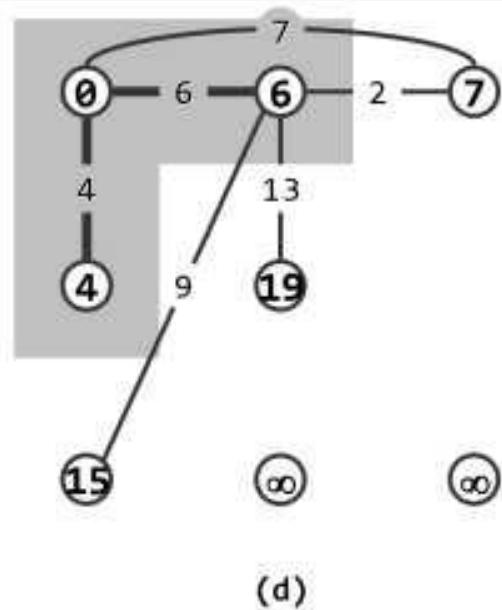
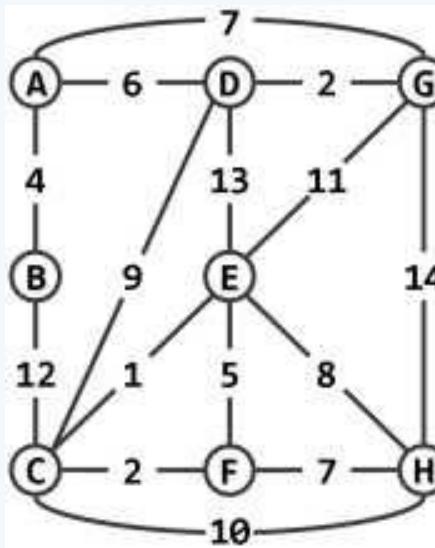


(b)

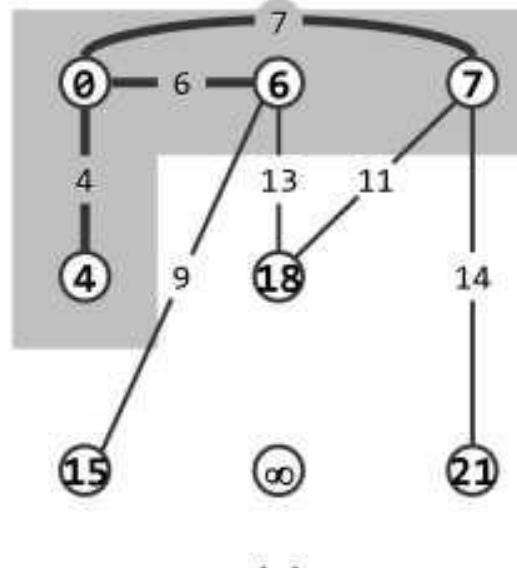


(c)

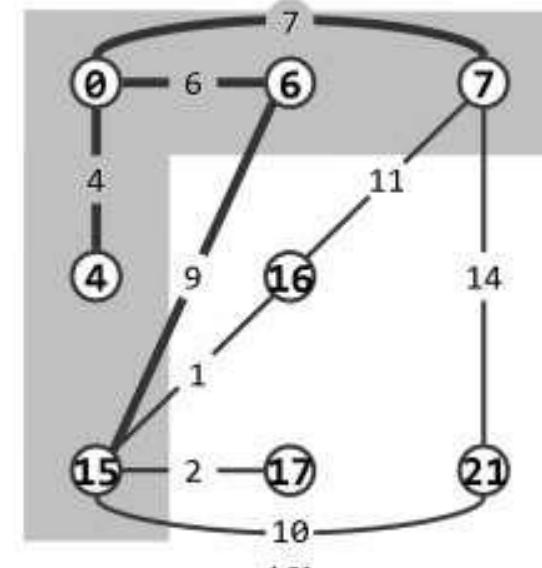
## 实例



(d)

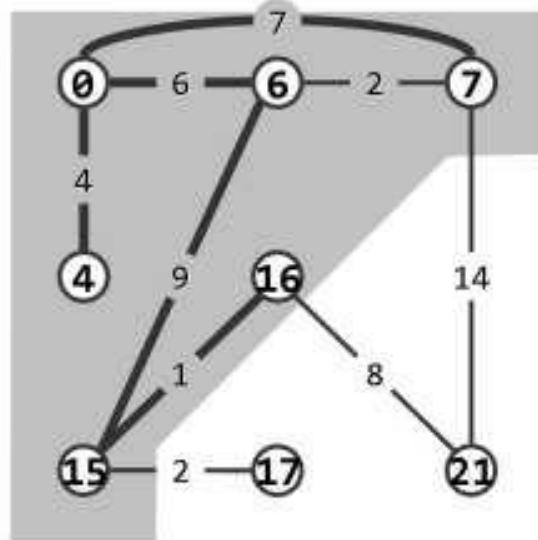
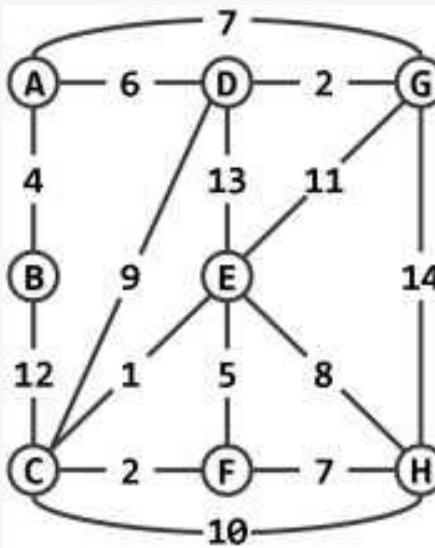


(e)

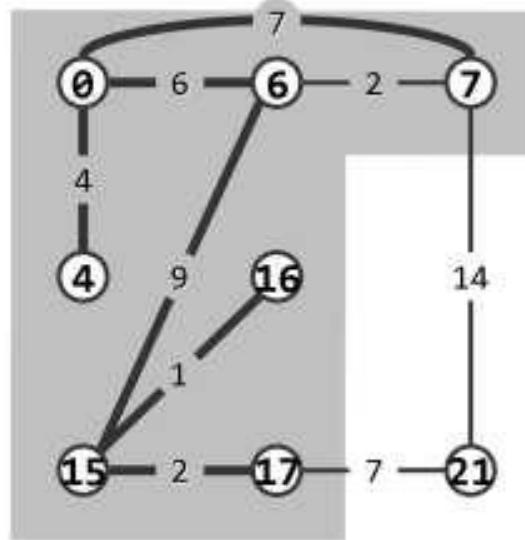


(f)

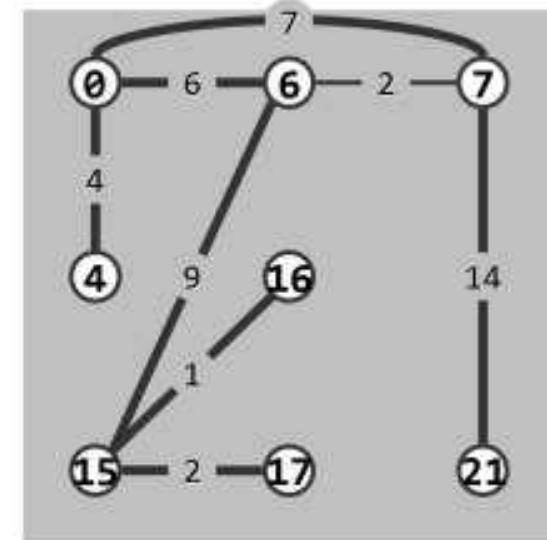
## 实例



(g)



(h)



(i)

## 实现

- ❖ 对于 $v_k$ 外各顶点 $v$ , 令 $\text{priority}(v) = v$ 到 $s$ 的距离  
于是套用优先级遍历算法框架...
- ❖ 为将 $T_k$ 扩充至 $T_{k+1}$ , 可以  
选出优先级最高的跨边 $e_k$ 及其对应顶点 $u_k$ , 并将其加入 $T_k$   
随后, 更新 $v_{k+1}$ 外所有顶点的优先级 (数)
- ❖ 注意, 优先级数随后可能改变 (降低) 的顶点, 必与 $u_k$ 邻接
- ❖ 因此, 只需枚举 $u_k$ 的每一邻接顶点 $v$ , 并取  
$$\text{priority}(v) = \min(\text{priority}(v), \text{priority}(u_k) + |u_k, v|)$$
- ❖ 为此, 需按照`prioUpdater()`模式, 编写一个优先级 (数) 更新器...

## 实现

```
❖ g->pfs( 0, DijkstraPU<char, int>() ); //从顶点0出发, 启动Dijkstra算法

❖ template <typename Tv, typename Te> //顶点类型、边类型

struct DijkstraPU { //Dijkstra算法的顶点优先级更新器

    virtual void operator()( Graph<Tv, Te>* g, int uk, int v ) {

        if ( UNDISCOVERED != g->status(v) ) return;

        // 对uk的每个尚未被发现的邻居v, 按Dijkstra策略做松弛

        if ( g->priority(v) > g->priority(uk) + g->weight(uk, v) ) {

            g->priority(v) = g->priority(uk) + g->weight(uk, v);
            g->parent(v)   = uk;
        }
    }
};
```

# Next

- 串匹配
- 数据结构（C++语言版）第三版 Chapter 11