6.图

(h) Dijkstra算法

邓俊辉

deng@tsinghua.edu.cn

问题描述

❖ 给定：连通有向图G及其中的顶点u和v

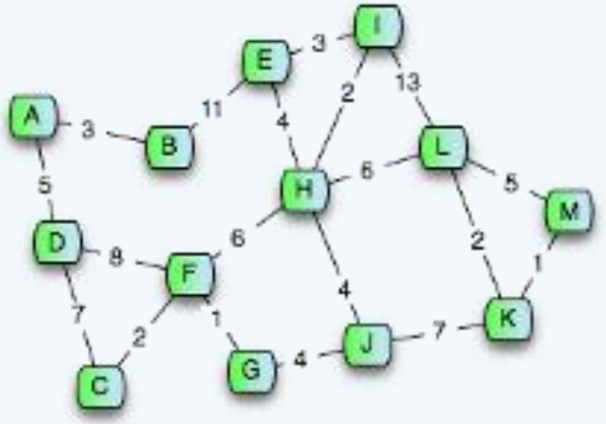
找到：从u到v的最短路径及其长度

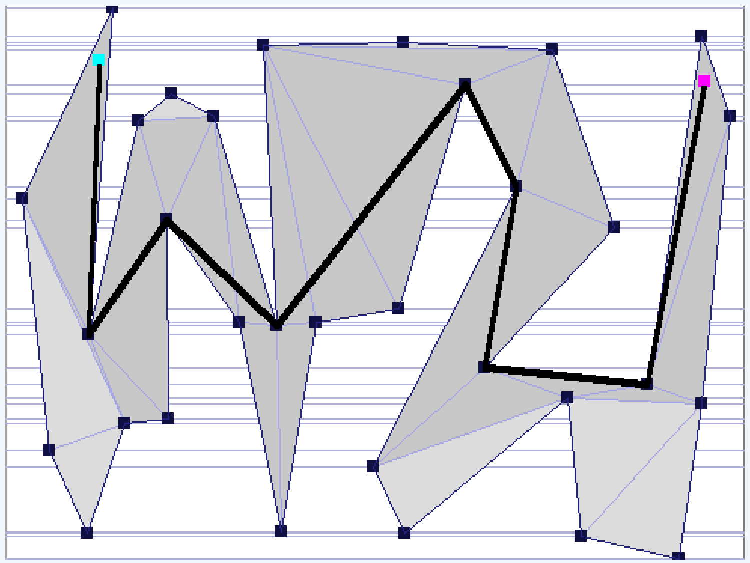
❖ 旅游者：最经济的出行路线

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

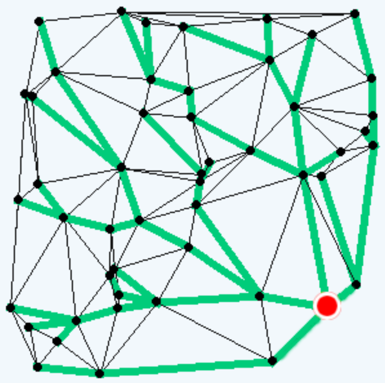
带权有向图 //负权值呢？

❖ 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

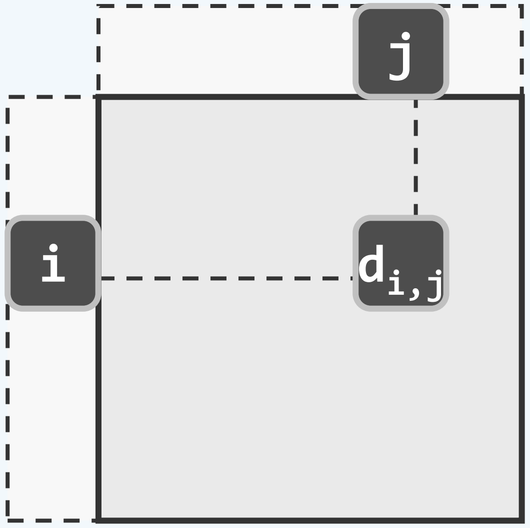


❖ 所有顶点对之间的最短路径

All shortest paths

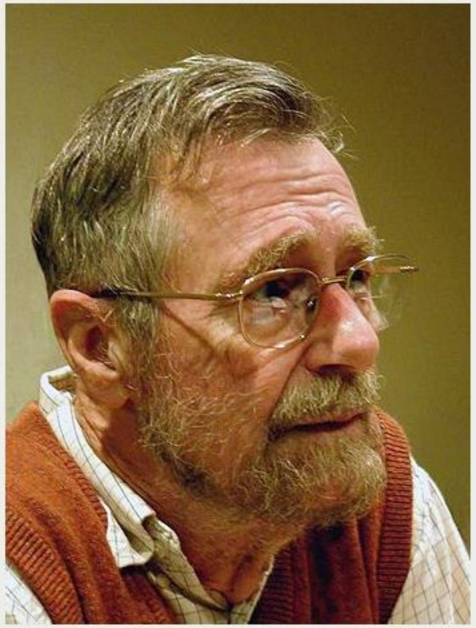
找出每对顶点i和j之间的最短路径及长度

Floyd-Warshall, 1962



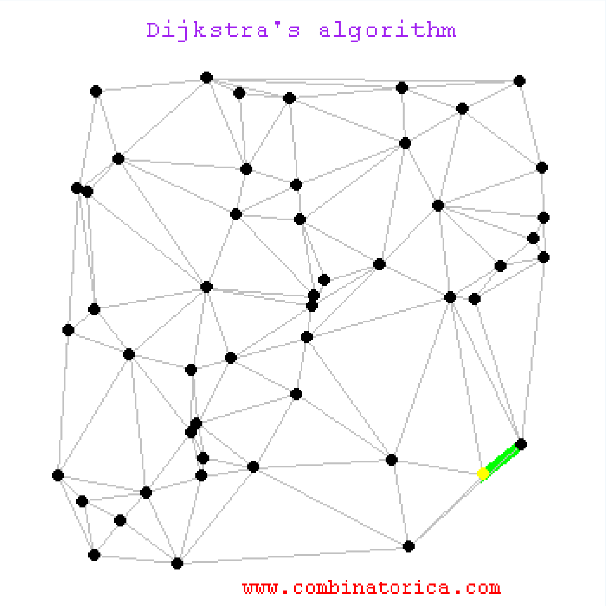
E. W. Dijkstra

❖ Turing Award, 1972



E. W. Dijkstra

(1930 ~ 2002)



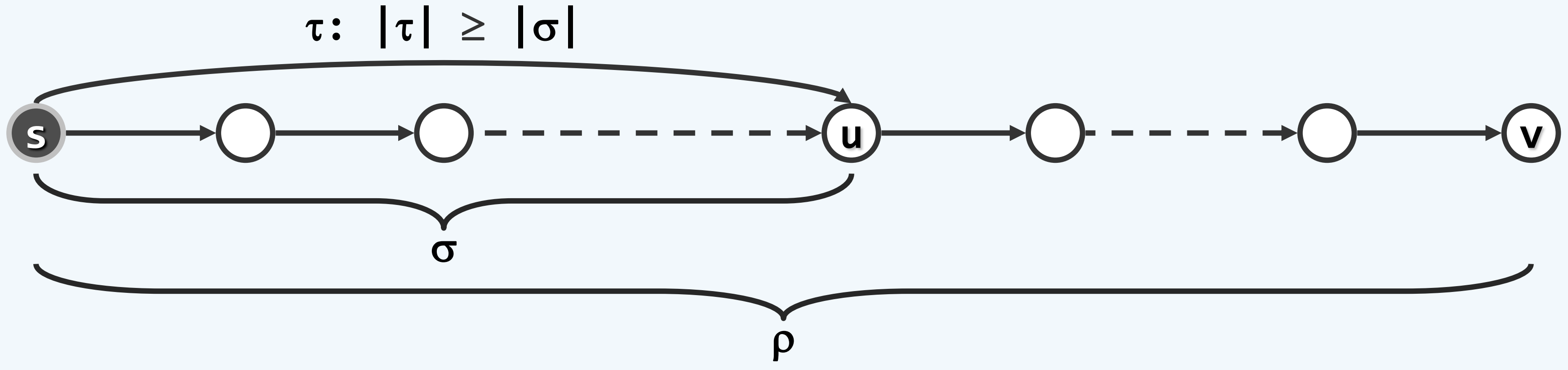
SPT

1)连通图中，s到每个顶点都有（至少）一条最短路径

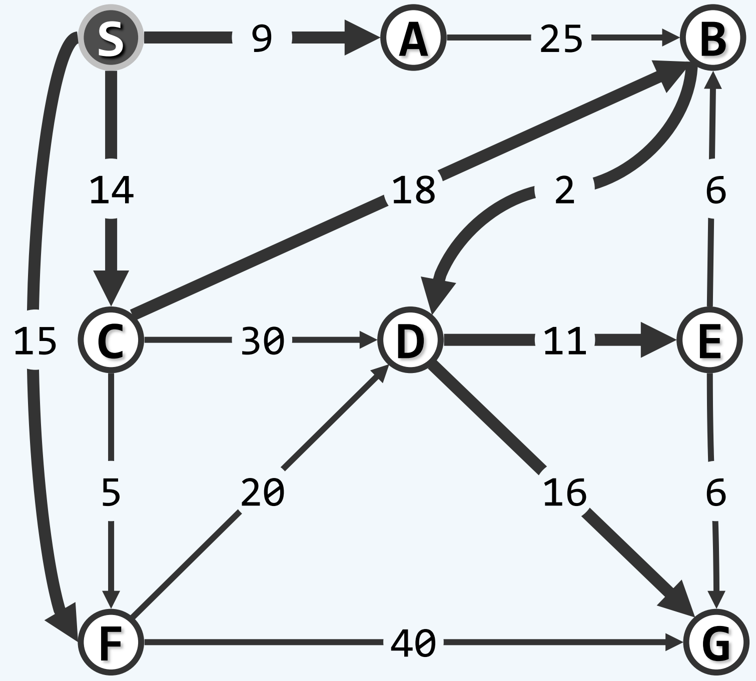
//非退化假设：每个顶点对应的最短路径唯一

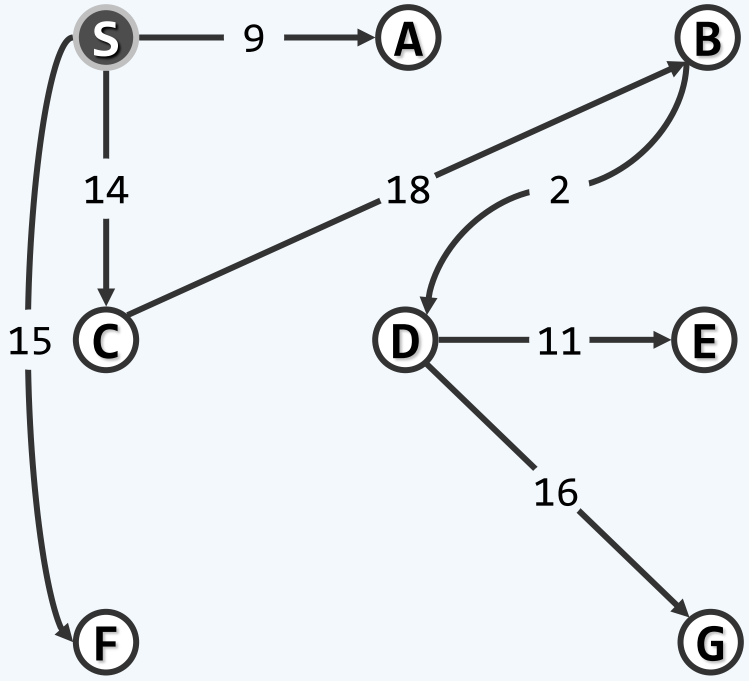
2)就同一起点s而言，任何最短路径的**前缀**，也是一条最短路径

3)就同一起点s而言，所有最短路径的并，不含回路

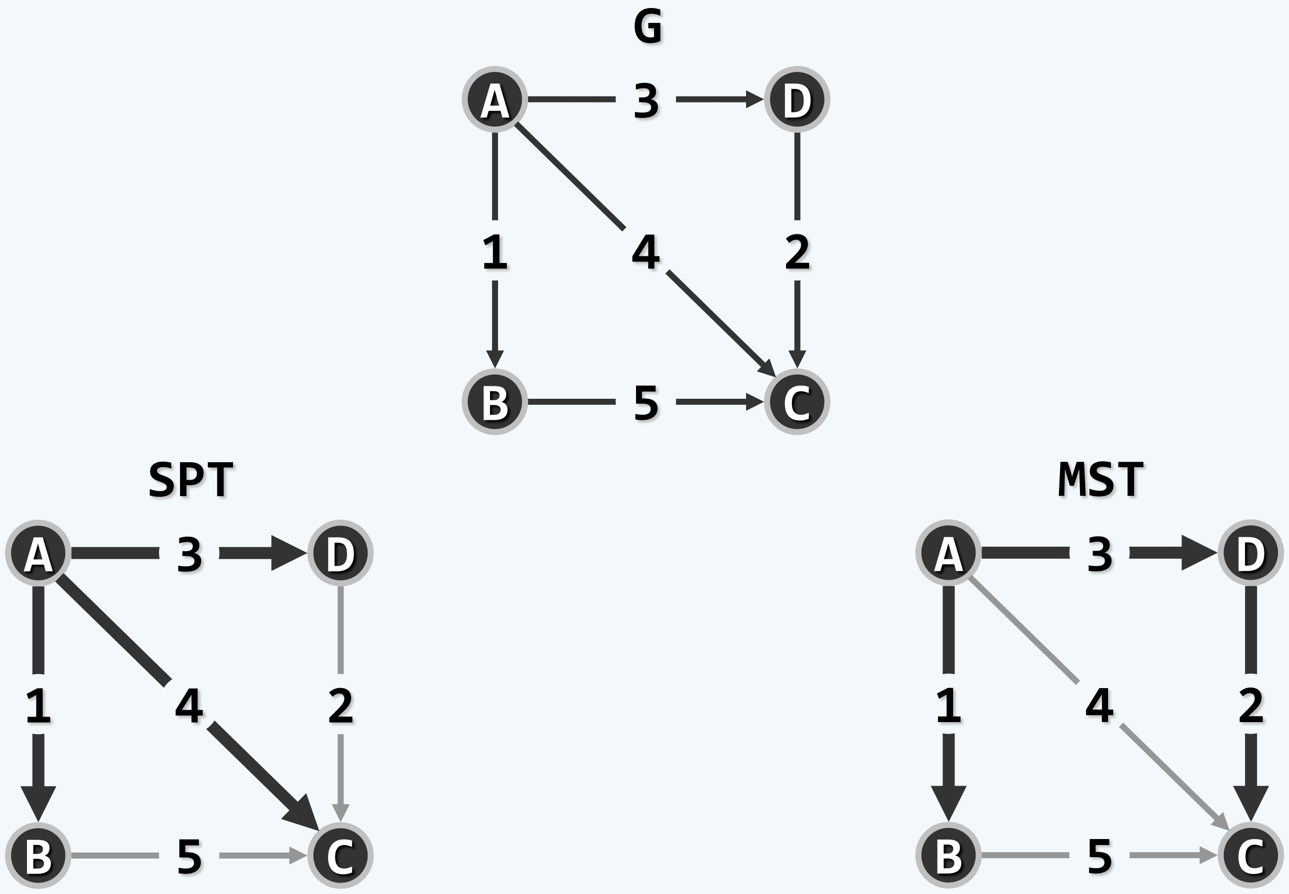


❖ 因此，所有最短路径的并，构成一棵树（shortest path tree）





SPT ≠ MST



u_1

- 按照到s的最短距离，对其余的顶点排序

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

- 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直接相联

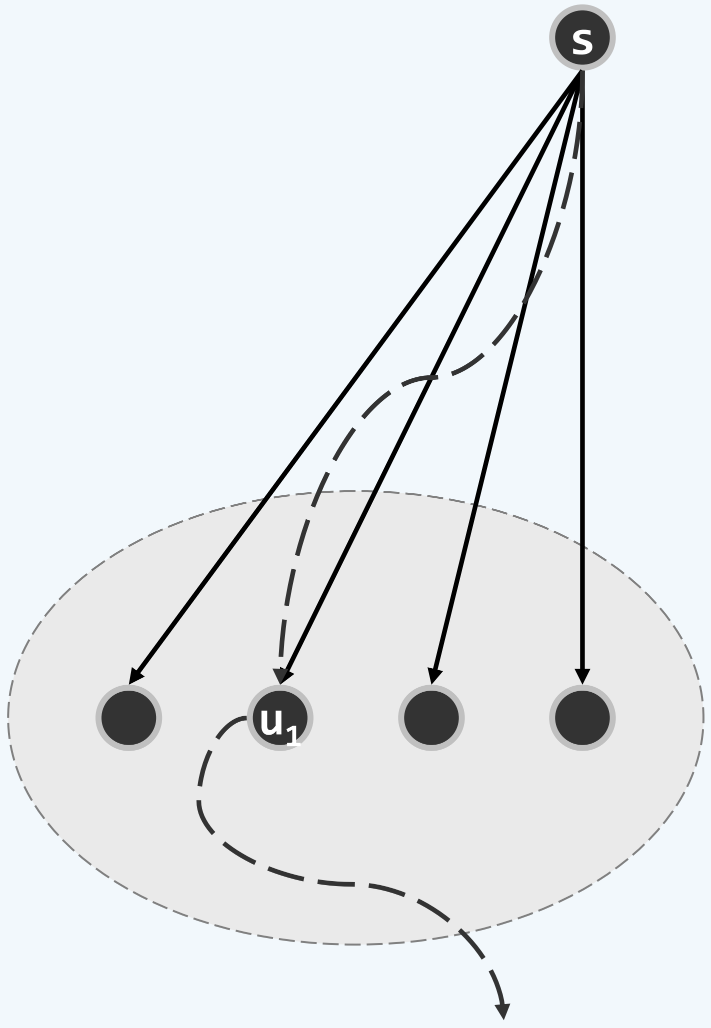
  $dist(s, s_1) = w(s, s_1) < \infty$

- $\forall u \ne s$,

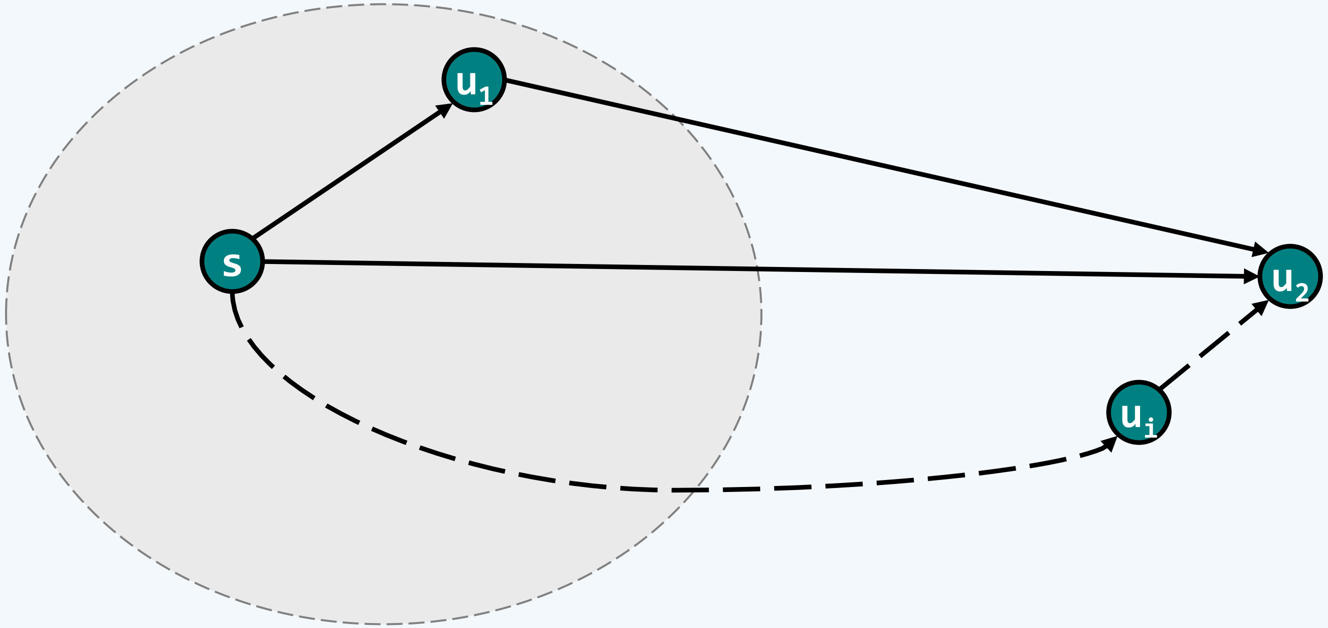
  $w(s, u) < \infty$ 仅当 $w(s, u_1) \le w(s, u)$

- 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

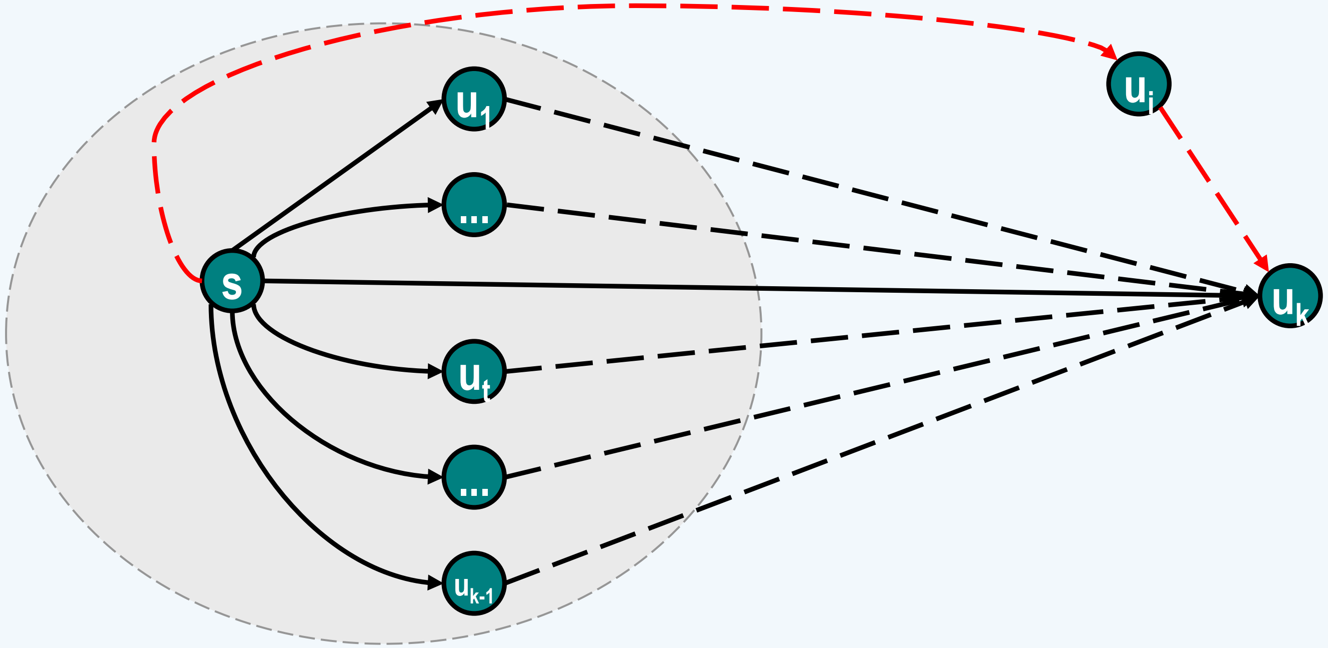


- 最短距离次小的顶点u_2 = ?
- $dist(s, u_2) = min\{ w(s, u_2), dist(s, u_1) + w(u_1, u_2) \}$



u_k

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



算法

- 从$T_1 = (\{v_1\};\ \varnothing)$开始，逐步构造$T_2$、$T_3$、...、$T_n$，其中

  $v_1 = s$

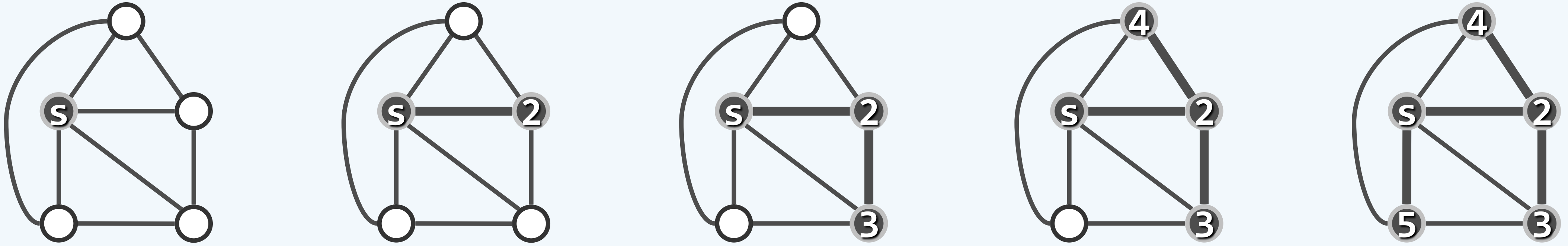
  $T_k = (V_k;\ E_k)$，$|V_k| = k$，$|E_k| = k-1$，$V_k \subset V_{k+1}$

- 由以上分析，为由T_k构造T_{k+1}，只需

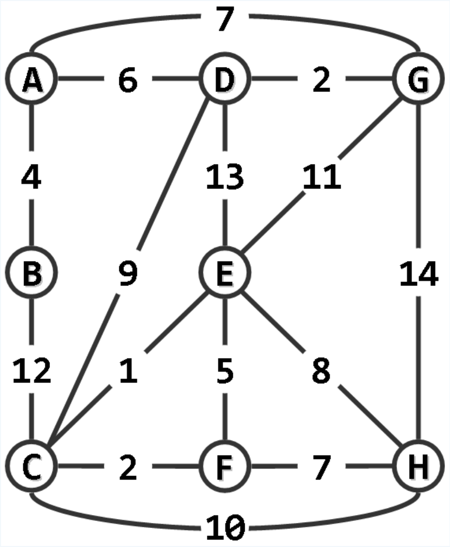
  将$(V_k : V \backslash V_k)$视作原图的一个割

  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 ）

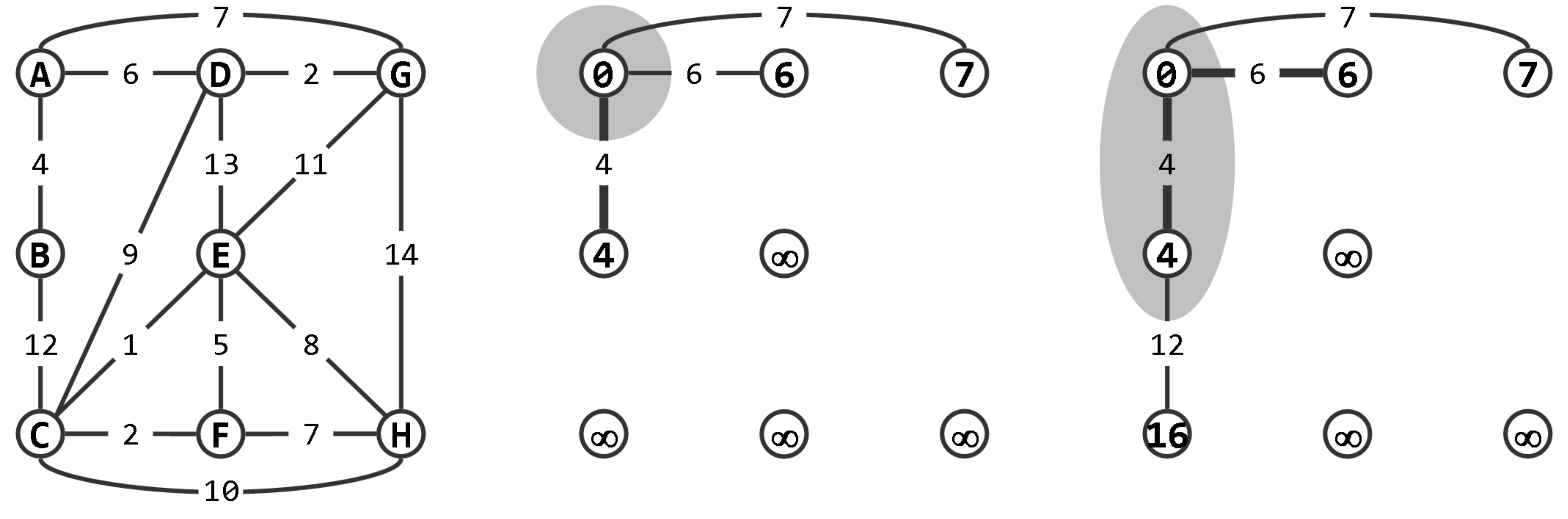
  令$T_{k+1} = (V_{k+1};\ E_{k+1}) = (V_k \cup \{u_k\};\ E_k \cup \{e_k\})$



实例



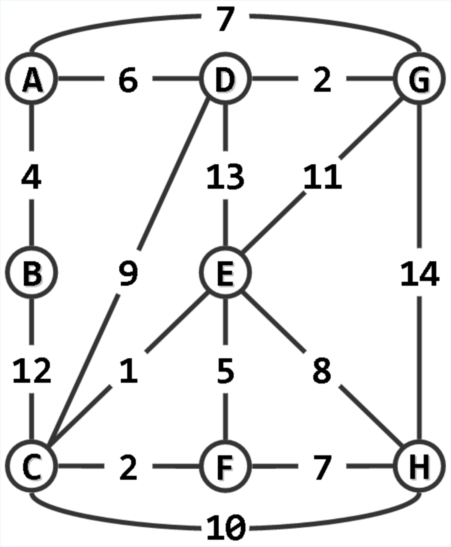
A
B
C
D
E
F
G
H
7
6
2
4
13
11
9
14
12
1
5
8
2
7
10



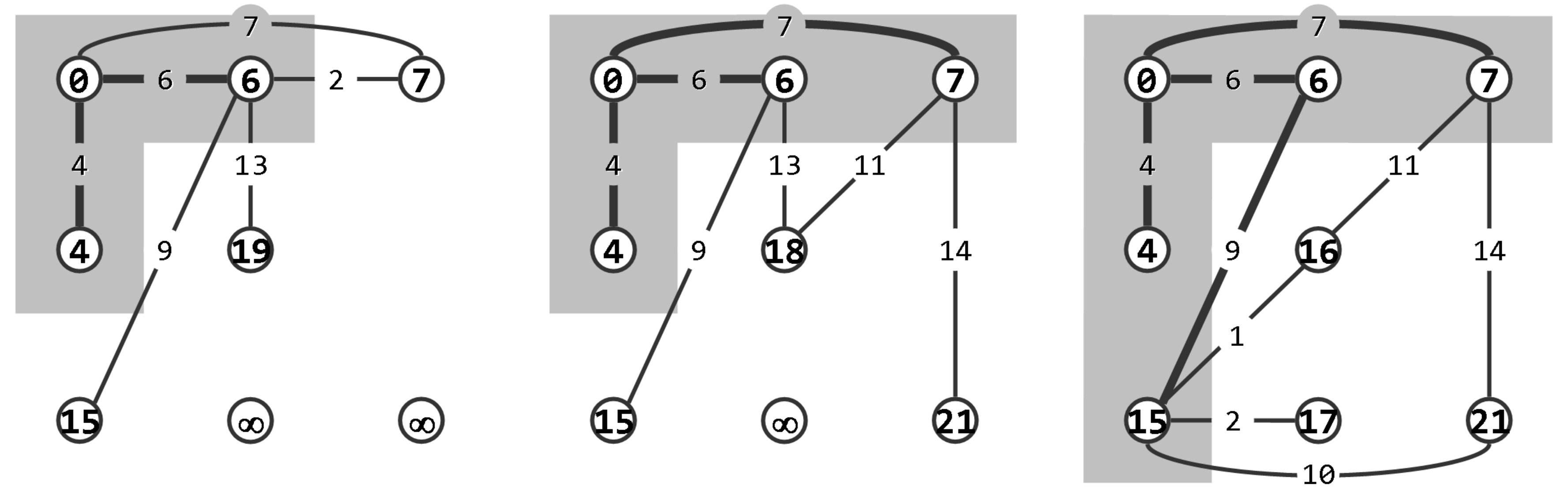
A
B
C
D
E
F
G
H
7
6
2
4
13
11
9
14
12
1
5
8
2
7
10
0
6
7
4
∞
∞
∞
∞
0
6
7
4
∞
16
∞
∞
12

(a) (b) (c)

实例

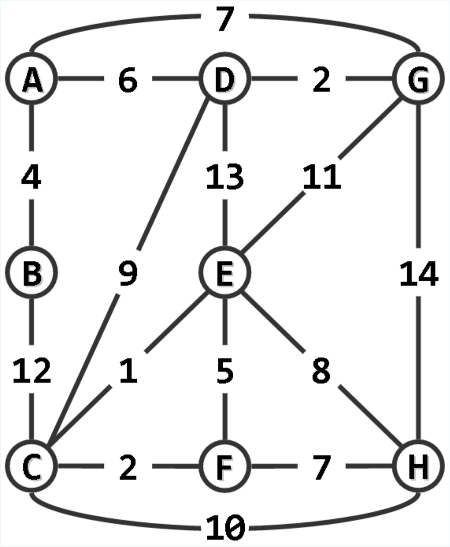


7
A 6 D 2 G
4 13 11
B 9 E 14
12 1 5 8
C 2 F 7 H
10

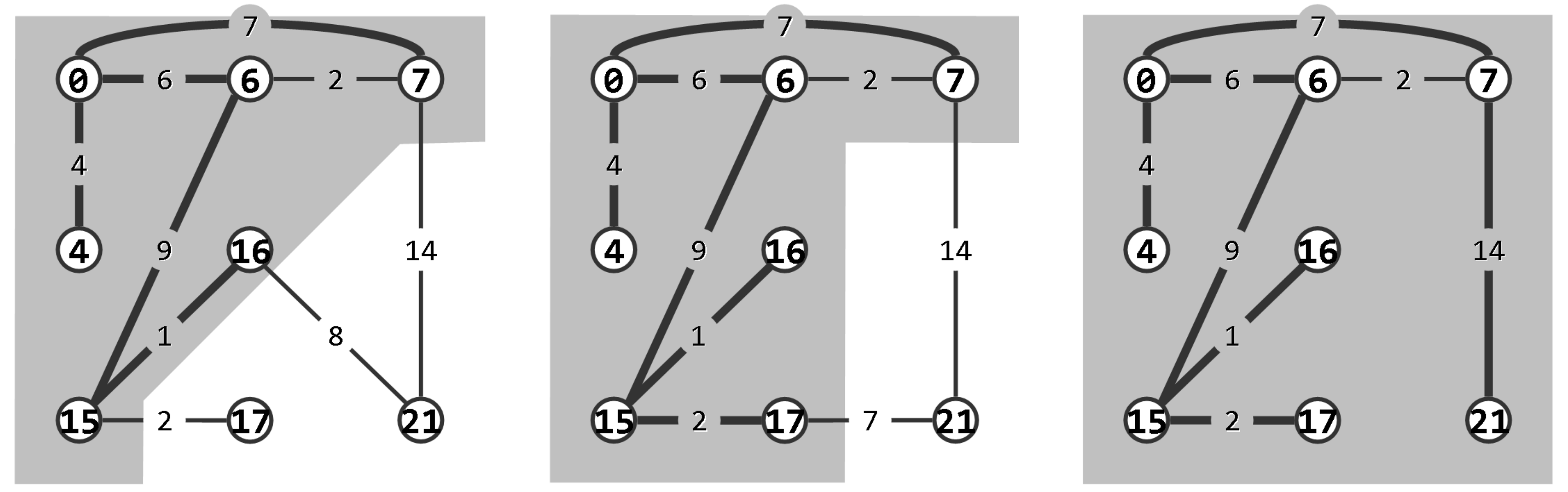


7
0 6 6 2 7
4 13
4 9 19
15 ∞ ∞
(d)
7
0 6 6 7
4 13 11
4 9 18 14
15 ∞ 21
(e)
7
0 6 6 7
4 11
4 9 16 14
1
15 2 17 21
10
(f)

实例



7
A
6
D
2
G
4
13
11
B
9
E
14
12
1
5
8
C
2
F
7
H
10



7
0
6
6
2
7
4
4
9
16
14
1
8
15
2
17
21

(g) (h) (i)

实现

❖ 对于V_k外各顶点v，令`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

priority(v) = min(priority(v), 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;
      }
   }
};
```