图

拓扑排序：零入度算法

11-A1

一吕二马三典韦，四关五赵六张飞，七许八黄九姜维

And as we walk, we must make the pledge that we shall always march ahead. We cannot turn back.

邓俊辉

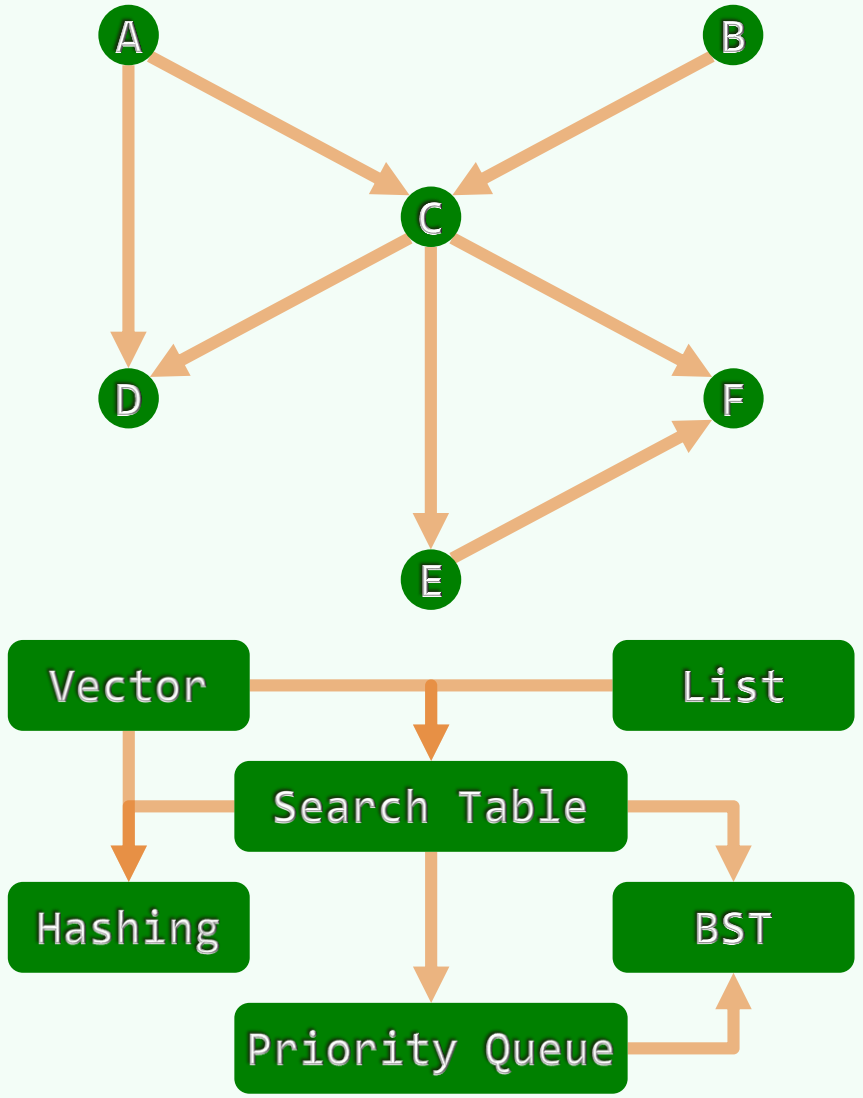
deng@tsinghua.edu.cn

有向无环图

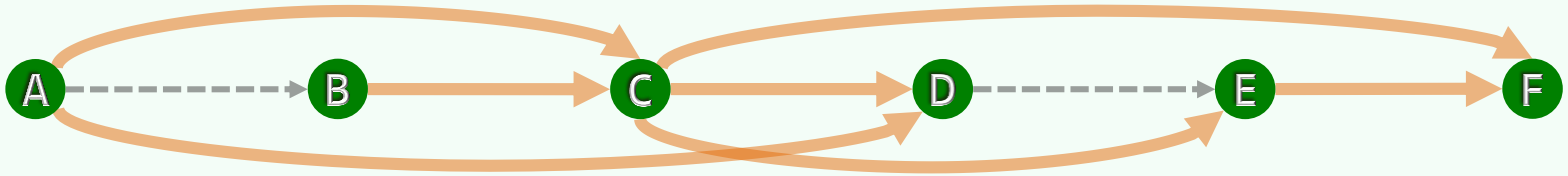
❖ Directed Acyclic Graph

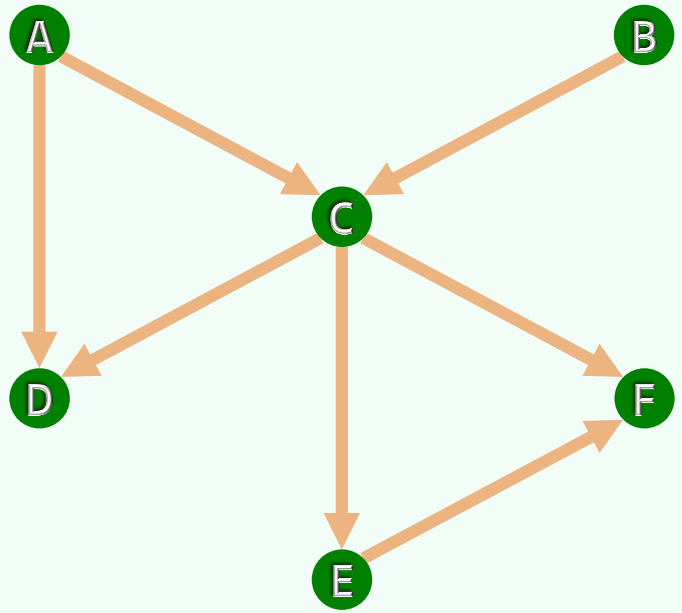
❖ 应用

- 类派生和继承关系图中，是否存在循环定义
- 操作系统中相互等待的一组线程，如何调度
- 给定一组相互依赖的课程，设计可行的培养方案
- 给定一组相互依赖的知识点，设计可行的教学进度方案
- 项目工程图中，设计可串行施工的方案
- email系统中，是否存在自动转发或回复的回路
- ...



拓扑排序





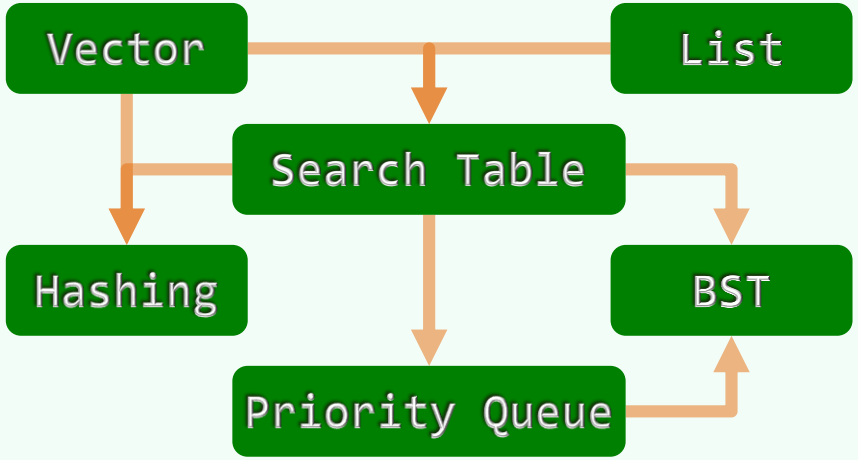
❖ 任给有向图G（不一定是DAG），尝试

将所有顶点排成一个**线性序列**，使其次序须与原图**相容**

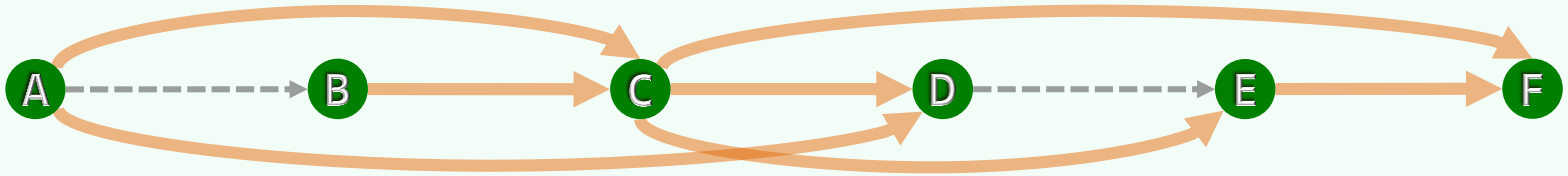
（每一顶点都不会通过边指向**前驱**顶点）

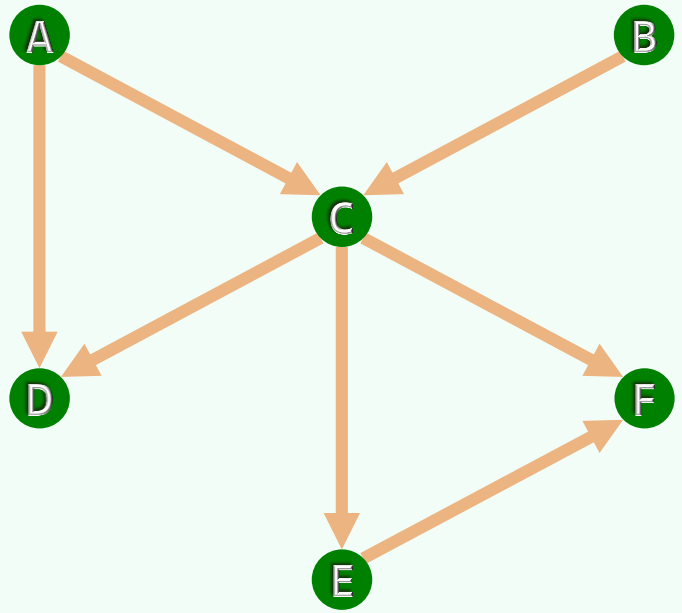
❖ 接口要求

- 若原图存在回路（即并非DAG），检查并报告
- 否则，给出一个相容的线性序列



偏序 ~ 极值





- 每个DAG对应于一个**偏序集**；拓扑排序对应于一个**全序集**

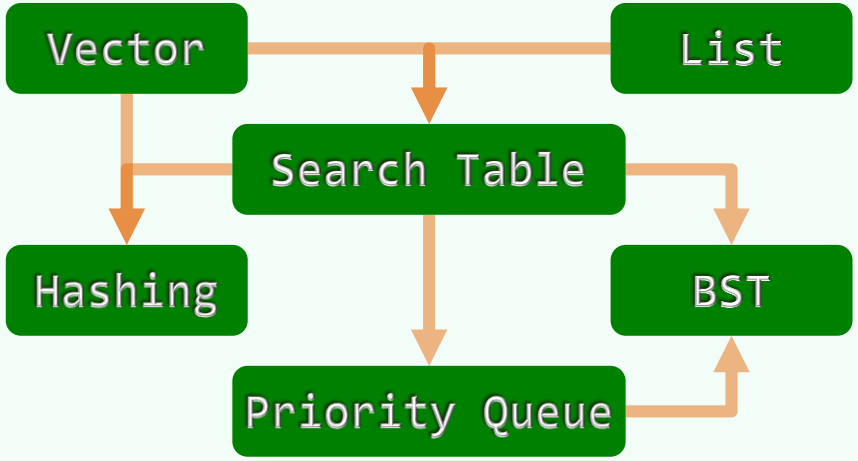
  所谓的拓扑排序，即构造一个与指定偏序集**相容**的全序集

- 可以拓扑排序的有向图，必定无环 //反之...

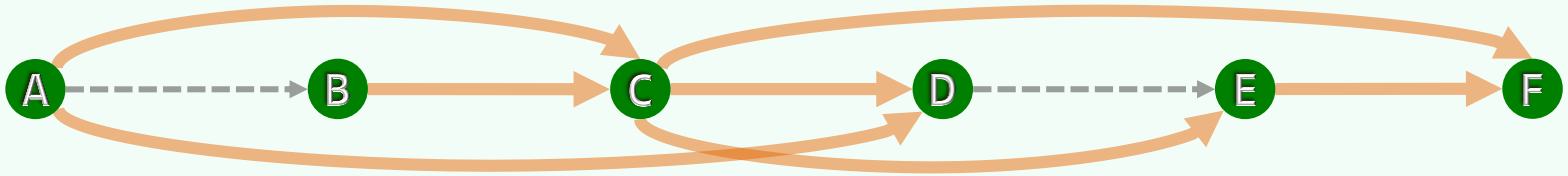
  任何DAG，都存在（至少）一种拓扑排序？是的！

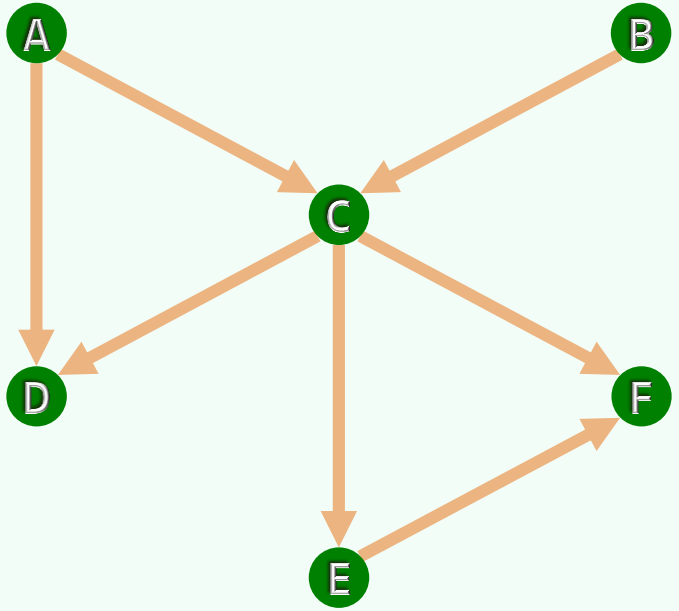
- 有限偏序集必有**极值**元素...

- 可归纳证明，并直接导出一个算法...



存在性





❖ 任何有向无环图 $\mathcal{G}$ 中

必有一个零入度的顶点 m

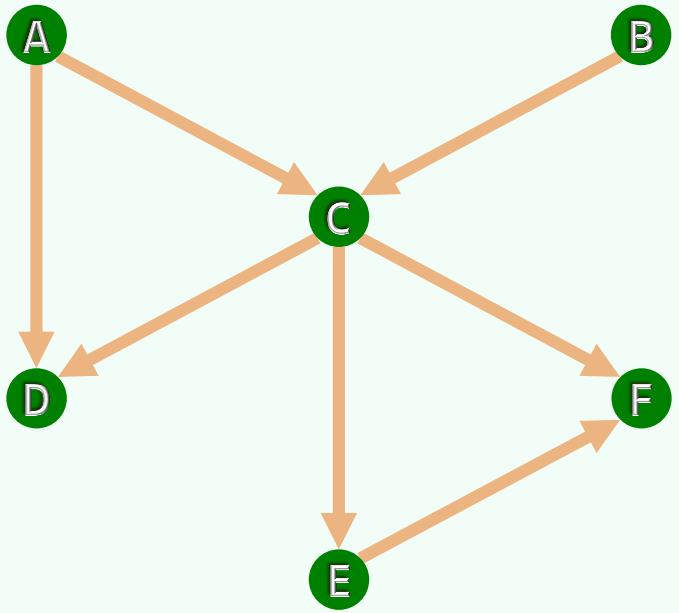
❖ 若 $\mathcal{G}\backslash\{m\}$ 存在拓扑排序 $S = \{ u_{k_1}, u_{k_2}, u_{k_3}, \ldots u_{k_{n-1}} \}$ //subtraction

则 $S' = \{ m, u_{k_1}, u_{k_2}, u_{k_3}, \ldots u_{k_{n-1}} \}$ 即为 $\mathcal{G}$ 的拓扑排序 //DAG子图亦为DAG

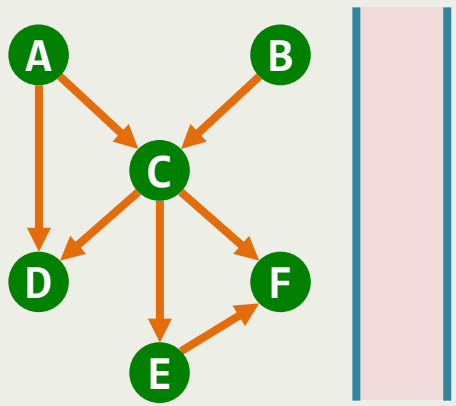
❖ 当然，只要 m 不唯一，拓扑排序也应不唯一 //反之呢?

策略：顺序输出零入度顶点

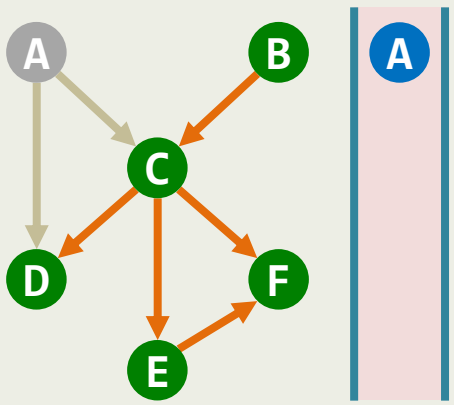
```
将所有入度为零的顶点存入栈S，取空队列Q //O(n)

while ( ! S.empty() ) { //O(n)

   Q.enqueue( v = S.pop() ); //栈顶v转入队列

   for each edge( v, u ) //v的邻接顶点u若入度仅为1

      if ( u.inDegree < 2 ) S.push( u ); //则入栈

   G = G \ { v }; //删除v及其关联边（邻接顶点入度减1）

} //总体O(n + e)

return |G| ? "NOT_A_DAG" :  Q; //残留的G空，当且仅当原图可拓扑排序
```



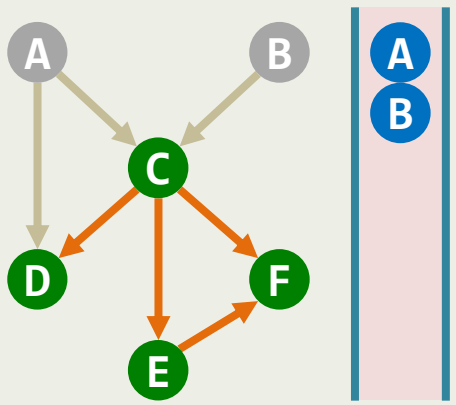
实例


A
B
C
D
F
E

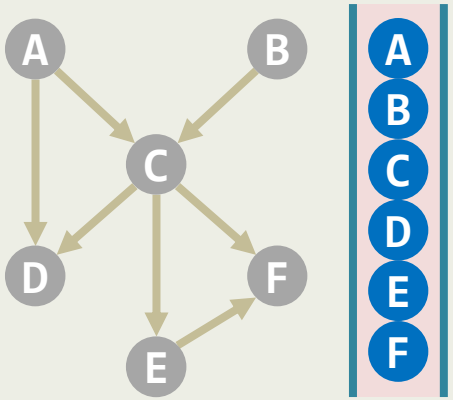
(a)


A
B
C
D
F
E
A

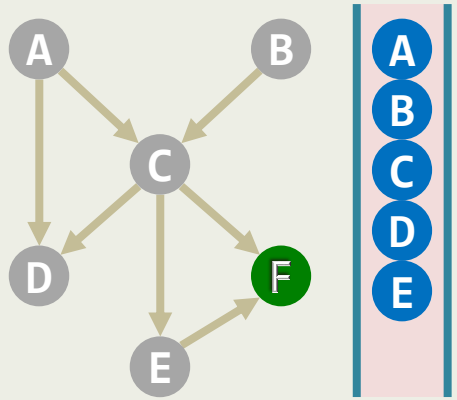
(b)


A
B
C
D
F
E
A
B

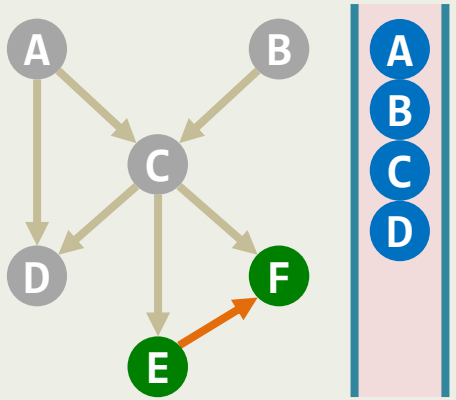
(c)


A
B
C
D
F
E
A
B
C
D
E
F

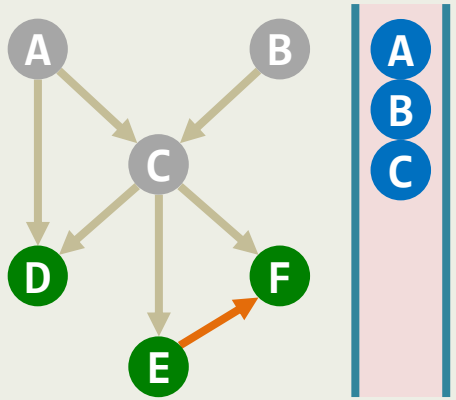
(g)


A
B
C
D
F
E
A
B
C
D
E

(f)


A
B
C
D
F
E
A
B
C
D

(e)


A
B
C
D
F
E
A
B
C

(d)

图

拓扑排序：零出度算法

11-A2

Begin with the end in mind.

真正的农夫每天完成自己的劳动，并不要求地里产出的成品一股脑儿归他所有，他心里想的是，他奉献出的不仅是他的第一个果子，而且还有他的最后一个果子

邓俊辉
deng@tsinghua.edu.cn

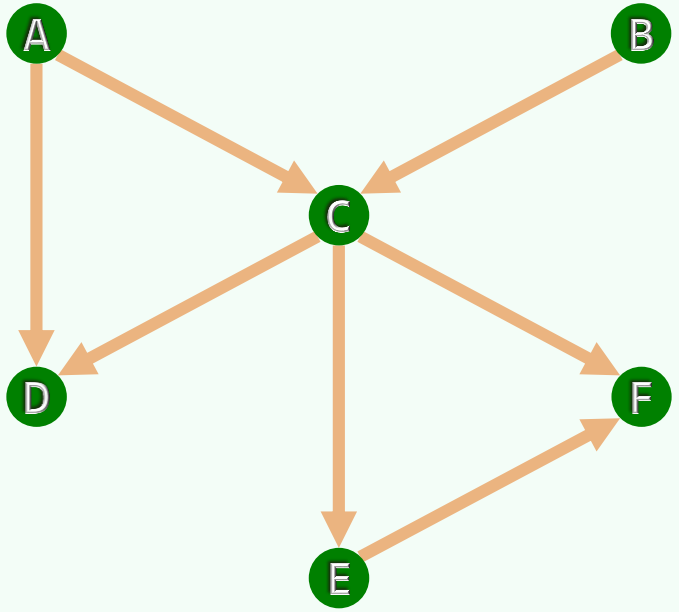
策略：逆序输出零出度顶点

❖ //基于DFS，借助栈S

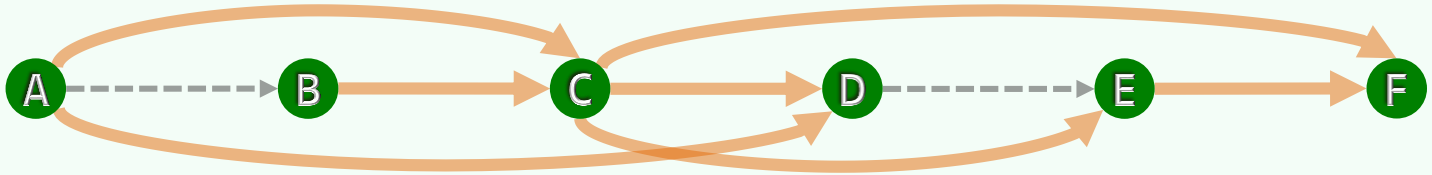
对图G做DFS，其间 //得到组成DFS森林的一系列DFS树

- 每当有顶点被标记为VISITED，则将其压入S
- 一旦发现有后向边，则报告“NOT_A_DAG”并退出

DFS结束后，顺序弹出S中的各个顶点

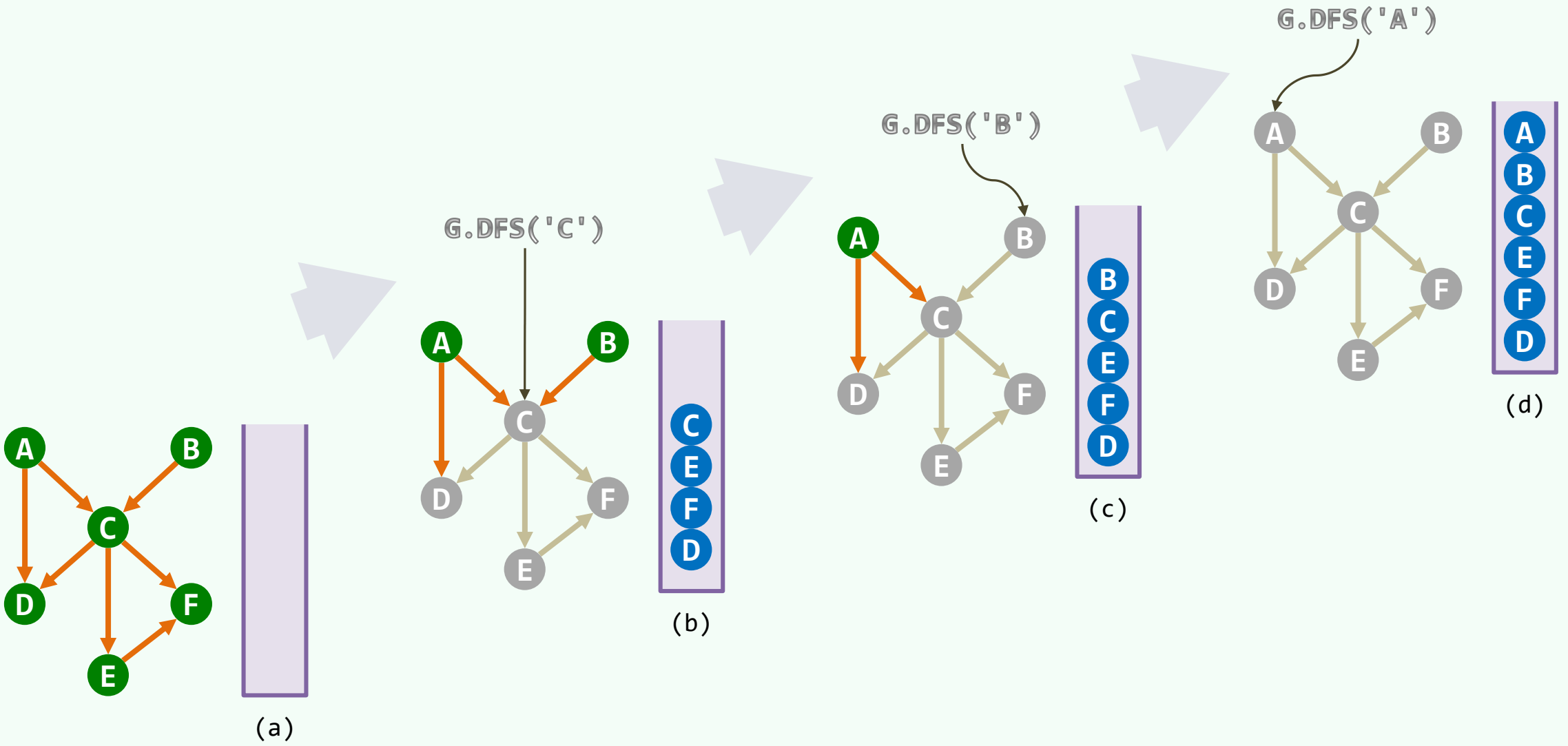


❖ 各节点按fTime逆序排列，即是拓扑排序



❖ 复杂度与DFS相当，也是 $\mathcal{O}(n+e)$

实例



实现（1/2）

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

bool Graph<Tv, Te>::TSort( Rank v, Rank& clock, Stack<Tv>* S ) {

   dTime(v) = ++clock; status(v) = DISCOVERED; //发现顶点v

   for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) ) //考查v的每一邻居u

      /* ... 视u的状态，分别处理 ... */

   status(v) = VISITED; S->push( vertex(v) ); //顶点被标记为VISITED时入栈

   return true;

}
```

实现（2/2）

```
for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) ) //考查v的每一邻居u
   switch ( status(u) ) { //并视u的状态分别处理
      case UNDISCOVERED:
         parent(u) = v; type(v, u) = TREE;
         if ( ! TSort(u, clock, S) ) return false; break; //从顶点u处深入
      case DISCOVERED: //一旦发现后向边（非DAG）
         type(v, u) = BACKWARD; return false; //则退出而不再深入
      default: //VISITED (digraphs only)
         type(v, u) = dTime(v) < dTime(u) ? FORWARD : CROSS; break;
   }
```

图应用

双连通分量：判定准则

11-B1

我自己则生活在几个相互之间没有联系的圈子里，我想，如果有人在创作时，能够同时处理在某一时期的不同地方圈子里发生的同等重要的多种故事，那创作出的这幅生活图景就会真实多了

邓俊辉

deng@tsinghua.edu.cn

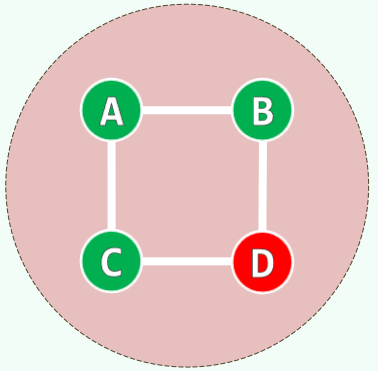
关节点 + 双连通分量

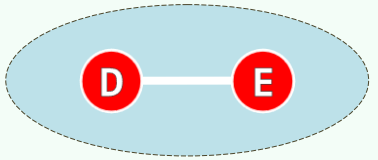
- **无向图的关节点：** //articulation point, cut-vertex

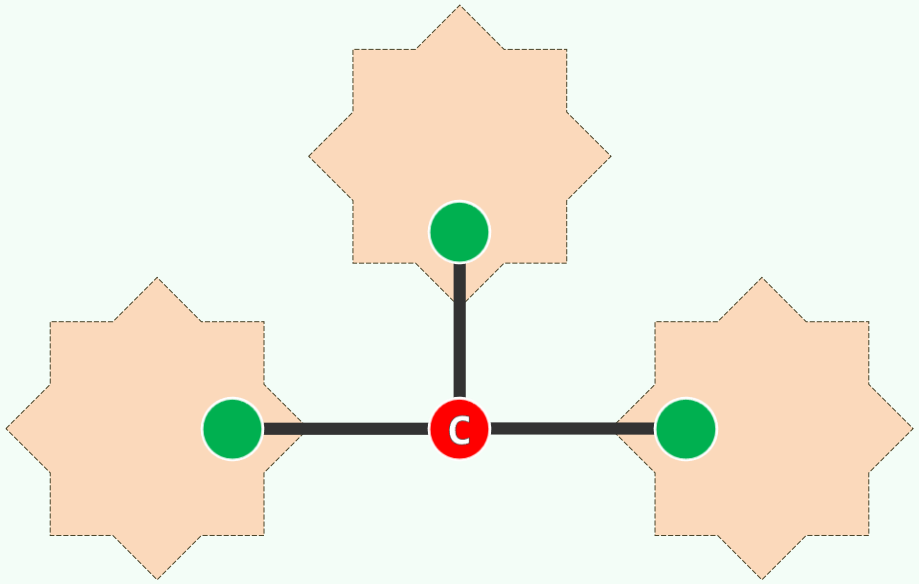
  其删除之后，原图的连通分量增多 //connected components

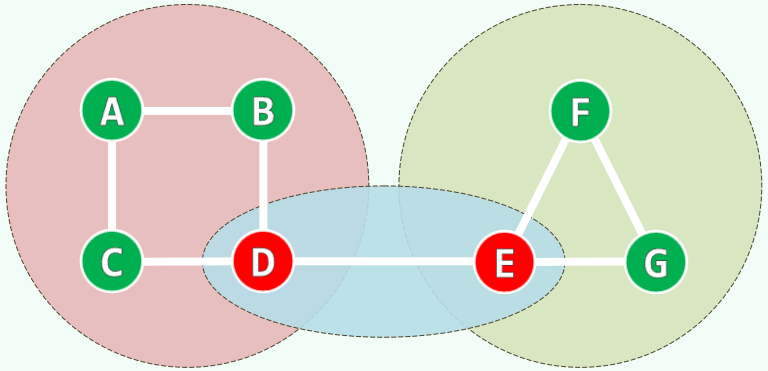
- **无关节点的图，称作双（重）连通图** //bi-connectivity

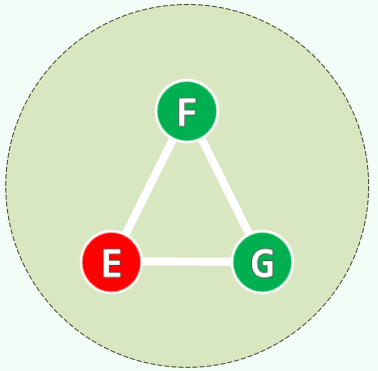
- **极大的双连通子图，称作双连通分量** //Bi-Connected Components









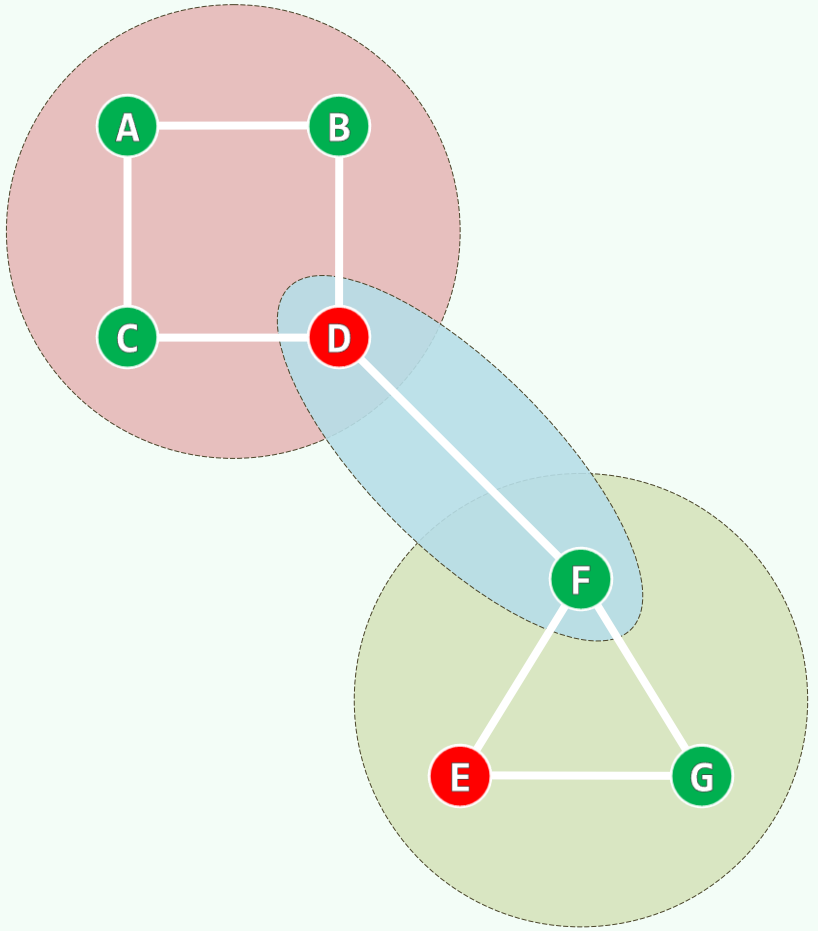


Brute-Force

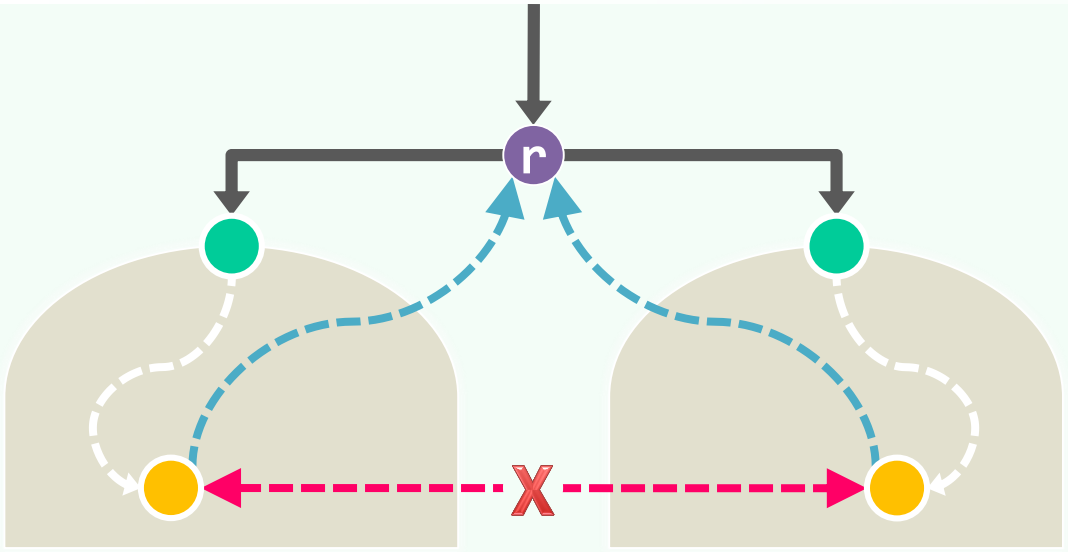
- 给定无向图，如何确定各BCC?
- 先考察简单的版本：如何确定关节点?
- 蛮力： 对每一顶点v，通过遍历检查G\{v}是否连通
- 共需 $\mathcal{O}(n \cdot (n+e))$ 时间，太慢!

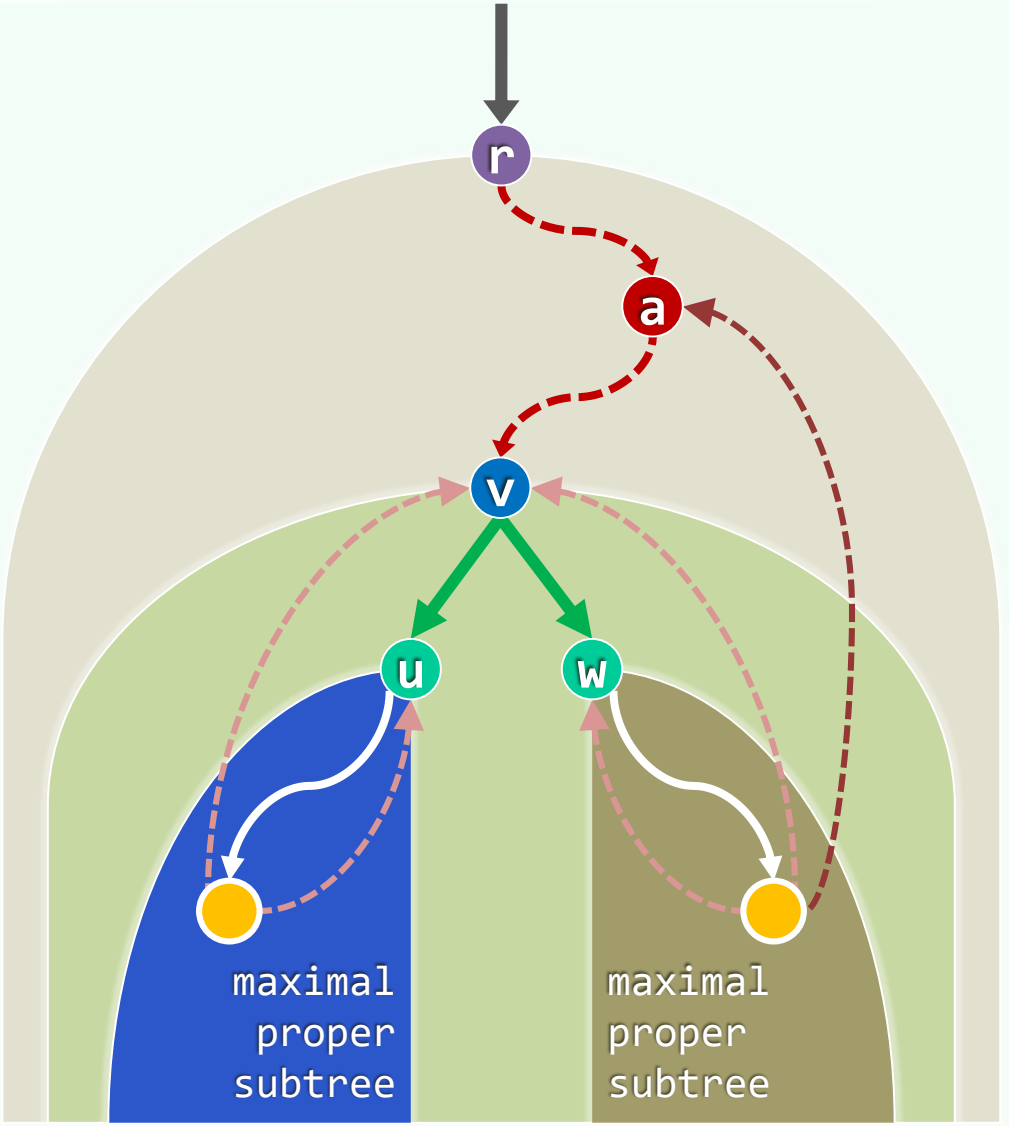
  而且，即便找出关节点，各BCC仍需确定
- 改进： 从任一顶点出发，构造DFS树

  根据DFS留下的标记，甄别是否关节点
- 比如，叶节点绝不可能是关节点 //为什么?



非叶节点





- 根r：必须至少有2棵子树
- 内部节点v：
  - 有某个孩子u，而subtree(u)不能经由BACKWARD边，联接到v的任何真祖先a
- 此时，{v} = BCC(u) ∩ BCC(parent(v))

Highest Connected Ancestor

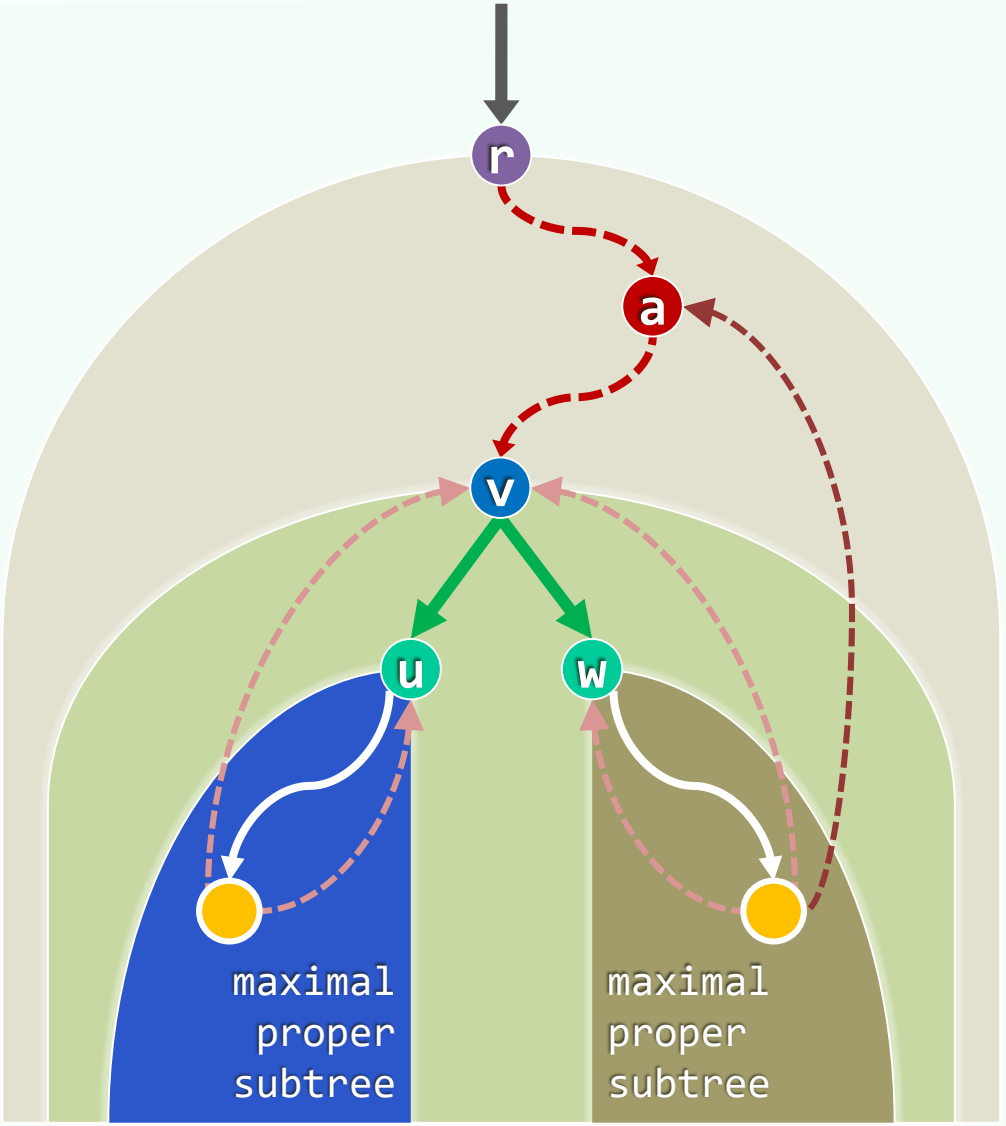
- hca(v) = subtree(v)经后向边能抵达的最高祖先
- 由括号引理：dTime越小的祖先，辈份越高
- DFS过程中，一旦发现后向边(v,u)

  即取：hca(v) = min(hca(v), dTime(u))
- DFS(u)完成并返回v时

  若有：hca(u) < dTime(v)

  即取：hca(v) = min(hca(v), hca(u))
- 否则，即可断定：v系关节点，且

  {v} + subtree(u)即为一个BCC



11-B2

图应用

双连通分量：算法

不管怎么说，我觉得今晚比以前任何时候跟祖父
都靠得近了。我想他老人家肯定会很高兴的

邓俊辉
deng@tsinghua.edu.cn

Graph::BCC()

```
#define hca(x) ( fTime(x) ) //利用此处闲置的fTime
template <typename Tv, typename Te>
void Graph<Tv, Te>::BCC( Rank v, int & clock, Stack<Rank> & S ) {
   hca(v) = dTime(v) = ++clock; status(v) = DISCOVERED; S.push(v);
   for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) )
      switch ( status(u) )
         { /* ... 视u的状态分别处理 ... */ }
   status(v) = VISITED; //对v的访问结束
}
#undef hca
```

switch (status(u))

```
case UNDISCOVERED:
   parent(u) = v; type(v, u) = TREE; //拓展树边
      BCC( u, clock, S ); //从u开始遍历，返回后...
   if ( hca(u) < dTime(v) ) //若u经后向边指向v的真祖先
      hca(v) = min( hca(v), hca(u) ); //则v亦必如此
   else //否则，以v为关节点（u以下即是一个BCC，且其中顶点此时正集中于栈S的顶部）
      while ( u != S.pop() ); //弹出当前BCC中（除v外）的所有节点
   break;
```

switch (status(u))

```
case DISCOVERED:
   type(v, u) = BACKWARD;
   if ( u != parent(v) )
   hca(v) = min( hca(v), dTime(u) ); //更新hca(v)，越小越高
   break;
default: //VISITED (digraphs only)
   type(v, u)  =  dTime(v) < dTime(u) ? FORWARD : CROSS;
   break;
```

复杂度

❖ 运行时间与常规的DFS相同，也是 $\mathcal{O}(n+e)$

自行验证：栈操作的复杂度也不过如此

❖ 除原图本身，还需一个容量为 $\mathcal{O}(e)$ 的栈存放已访问的边

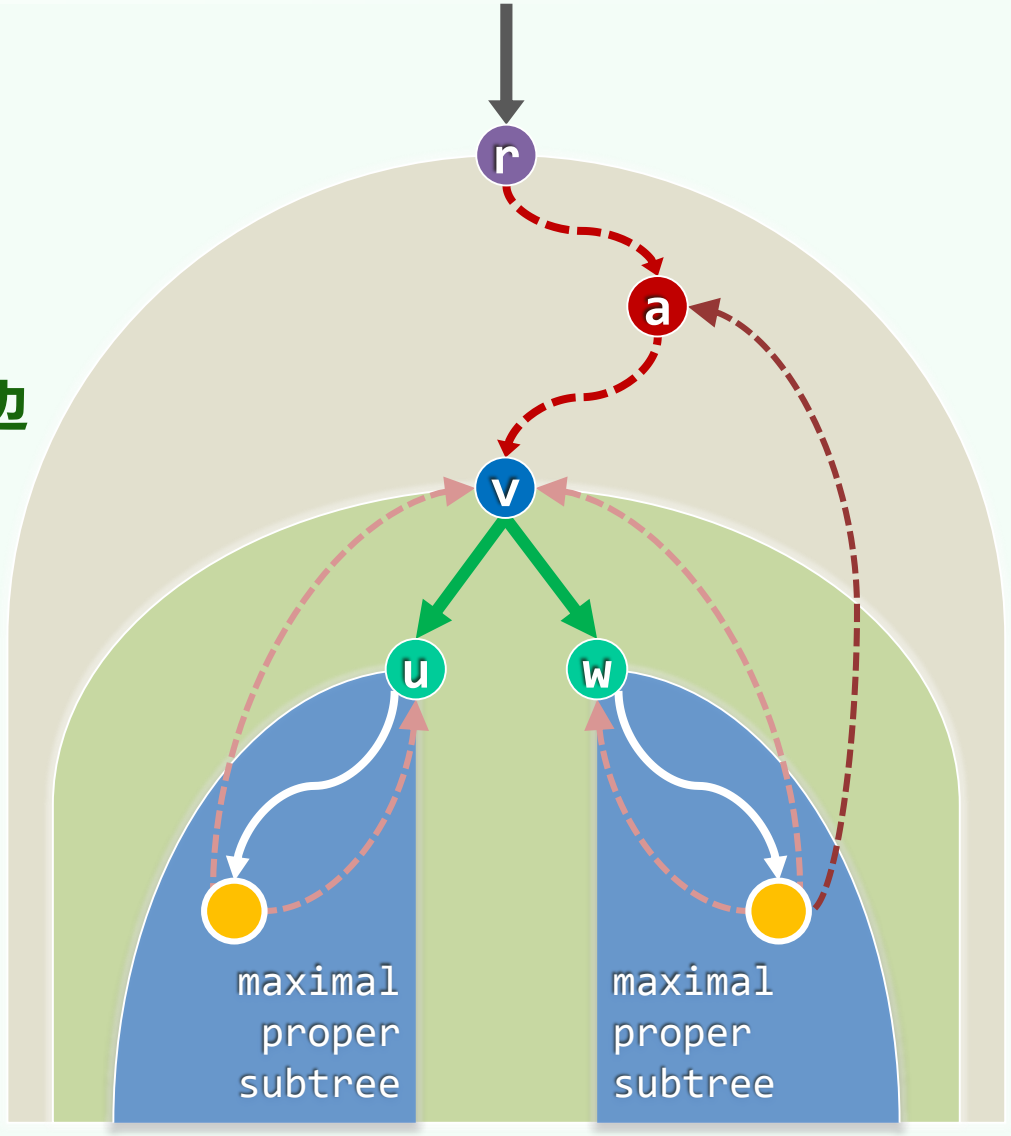
为支持递归，另需一个容量为 $\mathcal{O}(n)$ 的运行栈

❖ 如何推广至有向图的强连通分量

(Strongly-connected component)

- Kosaraju's algorithm

- Tarjan's algorithm



图应用

双连通分量：实例

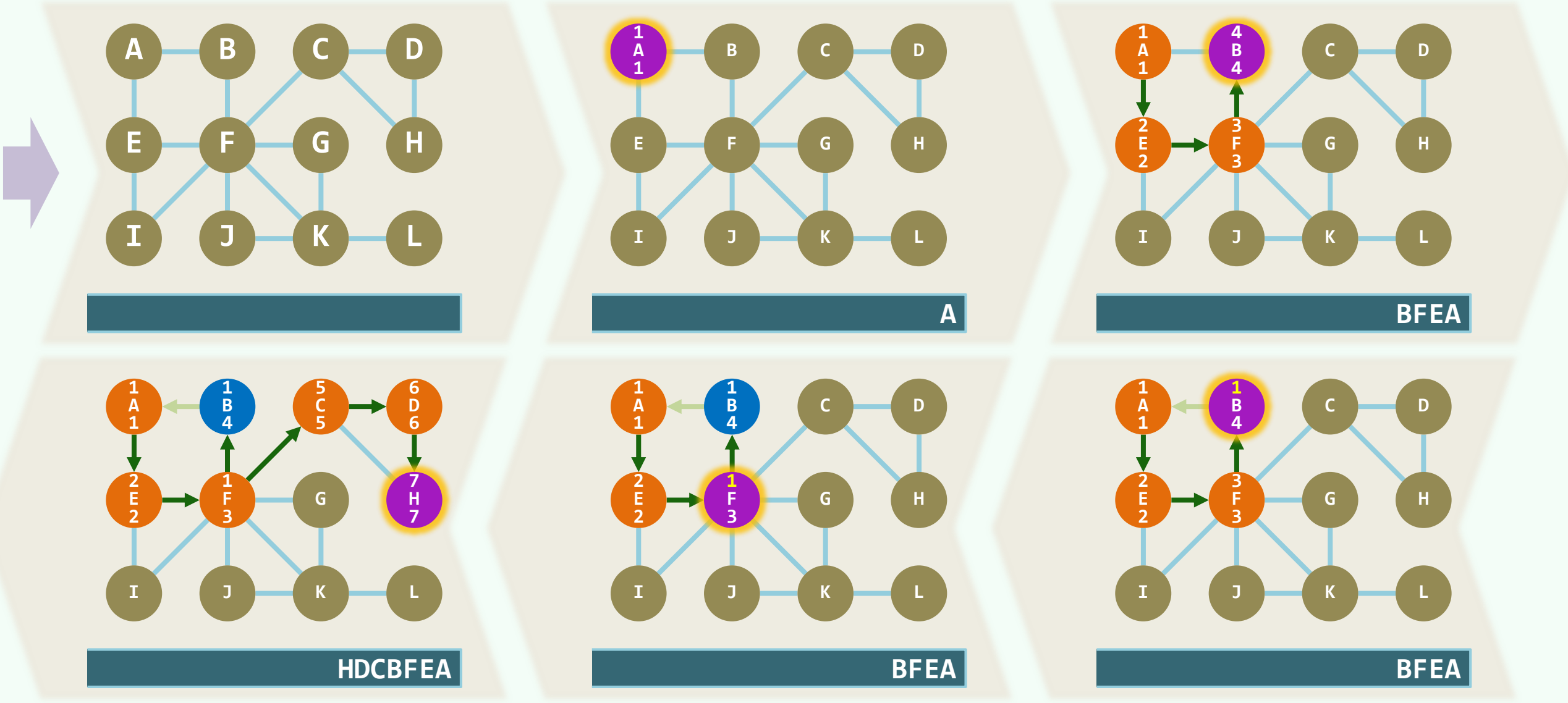
11-B3

邓俊辉

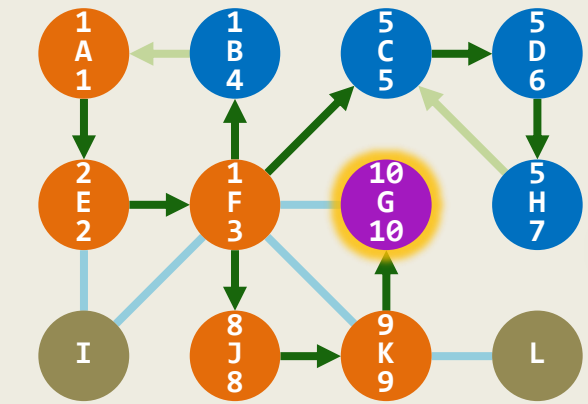
deng@tsinghua.edu.cn

彼节者有间，而刀刃者无厚；以无厚入有间，恢恢乎其于游刃必有余地矣

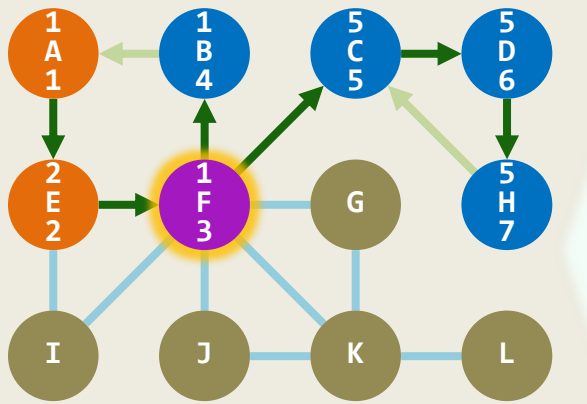
BCC: Example (1/5)



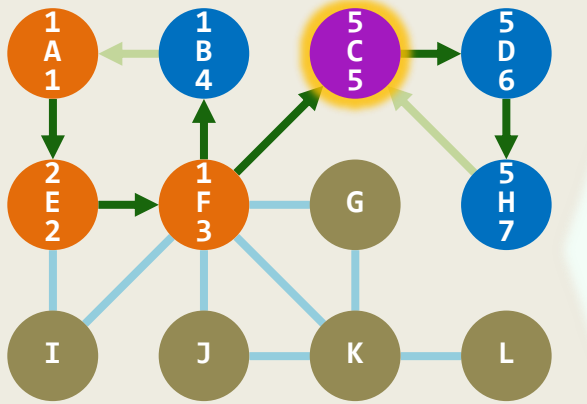
BCC: Example (2/5)



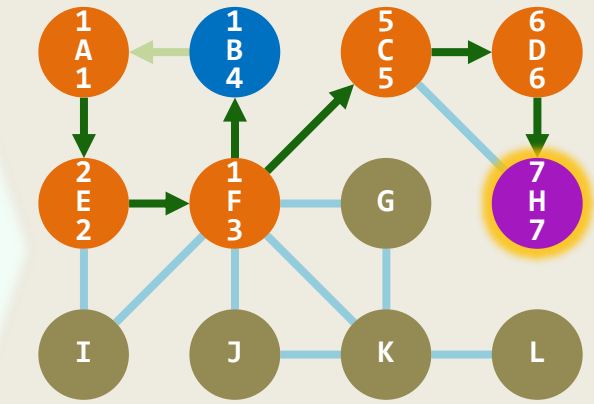
HDC CF GKJBFEA



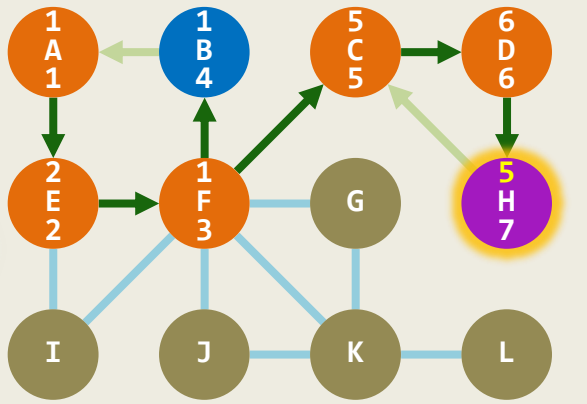
HDC CF BFEA



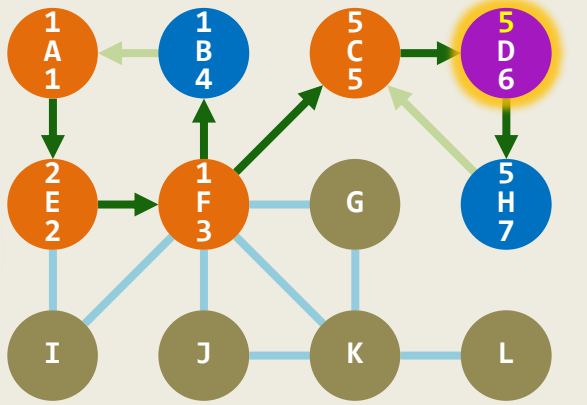
HDC CBFEA



HDCBFEA

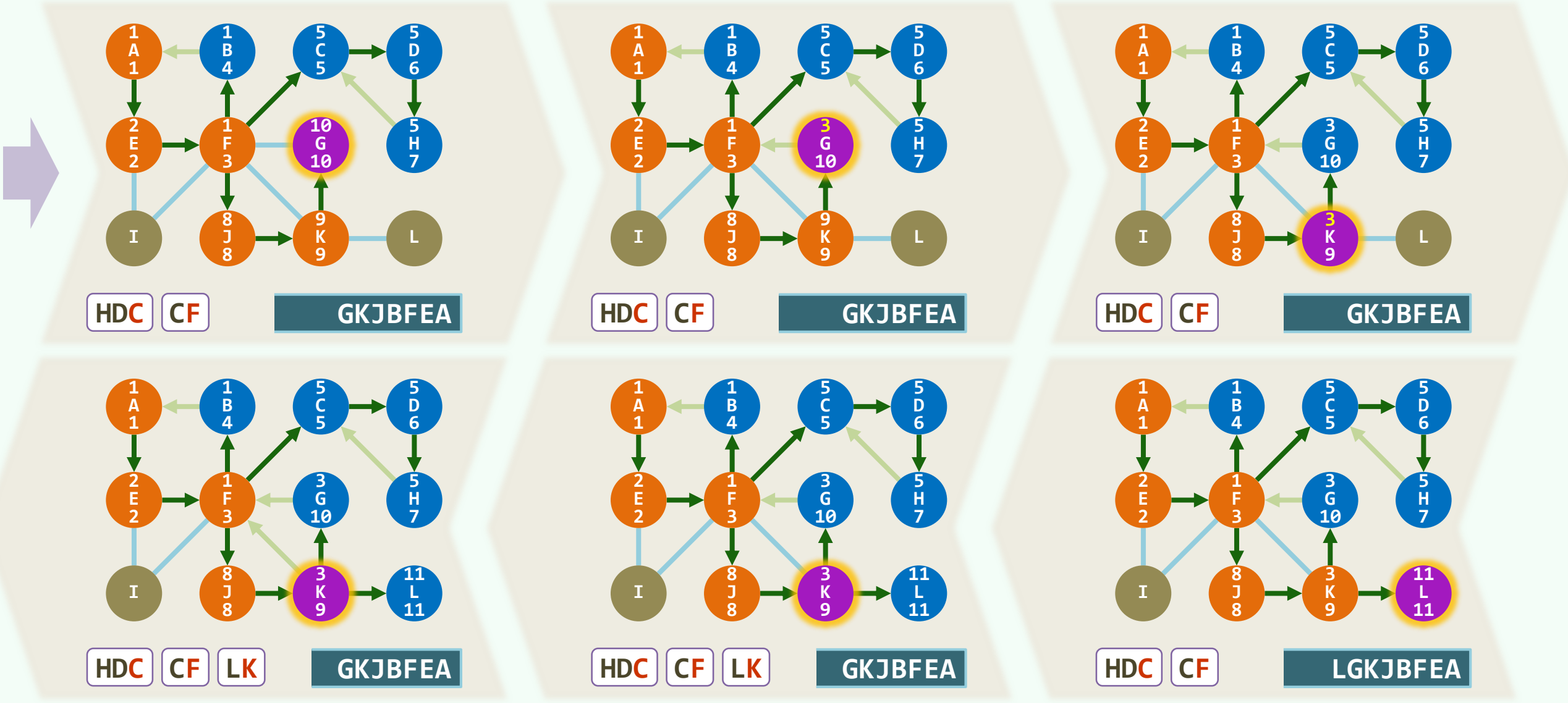


HDCBFEA

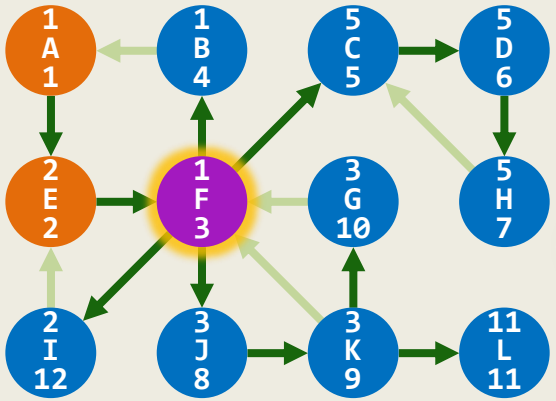


HDCBFEA

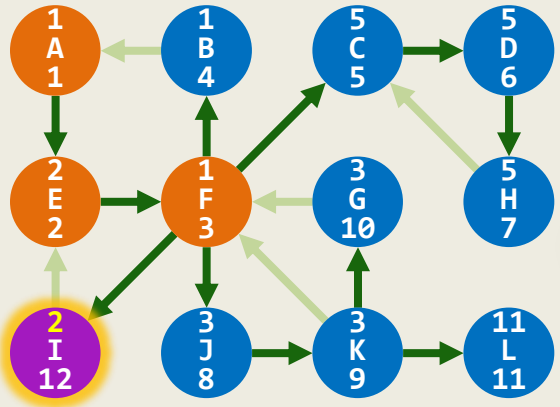
BCC: Example (3/5)



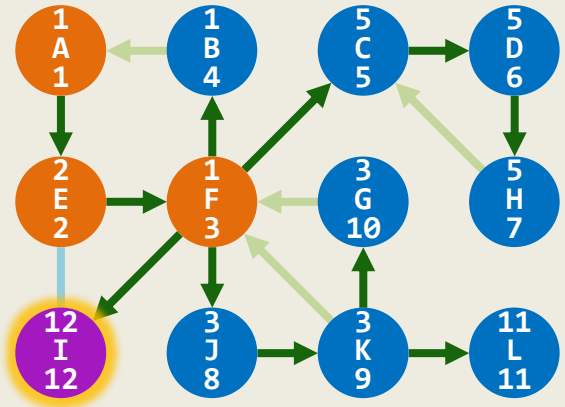
BCC: Example (4/5)



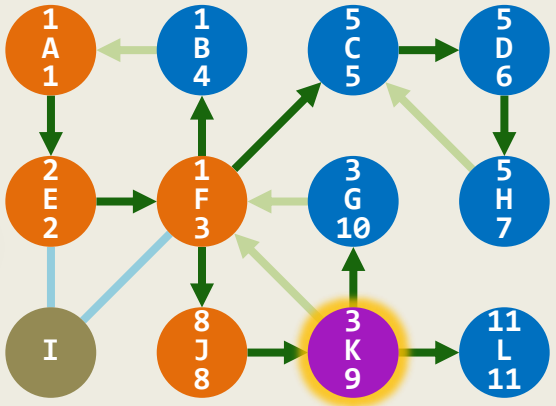
HDC CF LK GKJF IBFEA



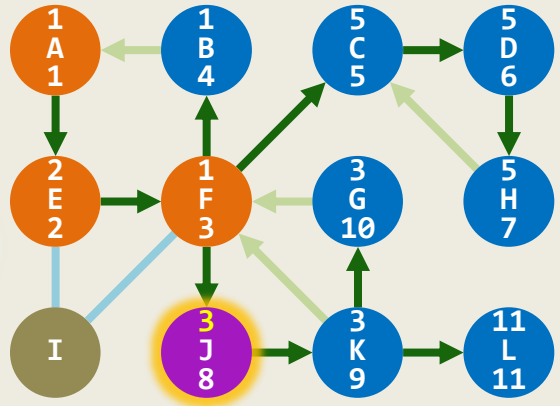
HDC CF LK GKJF IBFEA



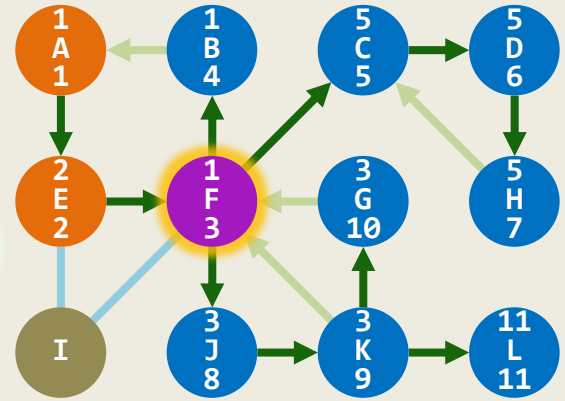
HDC CF LK GKJF IBFEA



HDC CF LK GKJBFEA

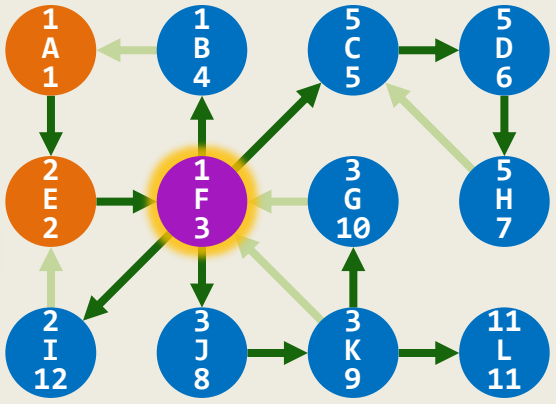


HDC CF LK GKJBFEA

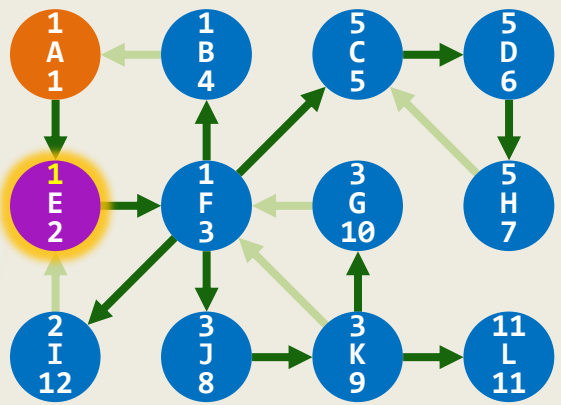


HDC CF LK GKJF BFEA

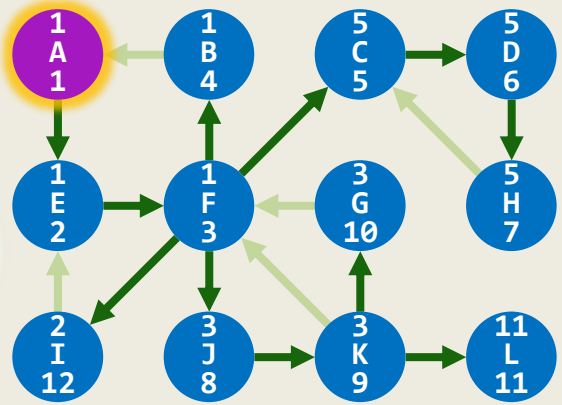
BCC: Example (5/5)



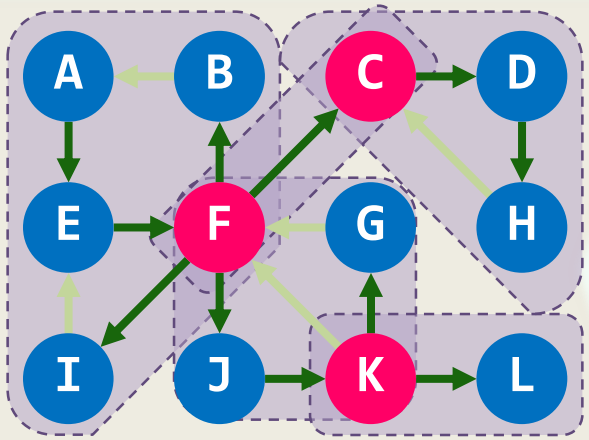
HDC CF LK GKJF IBFEA



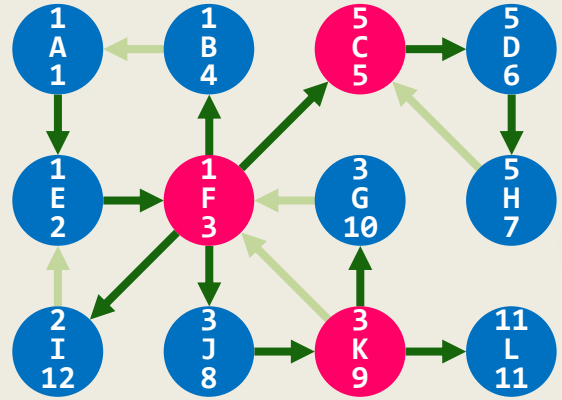
HDC CF LK GKJF IBFEA



HDC CF LK GKJF IBFEA



HDC CF LK GKJF IBFEA



HDC CF LK GKJF IBFEA

11-C

图应用

优先级搜索

邓俊辉

deng@tsinghua.edu.cn

尧舜之知而不遍物，急先務也；尧舜之仁不遍愛人，急親賢也

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——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，priority(v) == INT_MAX，意味着v的优先级最低

统一框架（1/2）

```
template <typename Tv, typename Te>

template <typename PU> //优先级更新器（函数对象）

void Graph<Tv, Te>::PFS( Rank v, PU prioUpdater ) { //PU的策略，因算法而异

   priority(v) = 0; status(v) = VISITED; //起点v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（2/2）

```
for ( Rank k = 1; k < n; k++ ) { //逐步将n-1顶点和n-1条边加至PFS树中
   for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) ) //对v的每一个邻居u
      prioUpdater( this, v, u ); //更新其优先级及其父亲
   int shortest = INT_MAX;
   for ( Rank u = 0; u < n; u++ ) //从尚未加入遍历树的顶点中，选出下一个优先级
      if ( (UNDISCOVERED == status(u)) && (shortest > priority(u)) ) //最高的
         { shortest = priority(u); v = u; } //顶点v
   status(v) = VISITED; type( parent(v), v ) = TREE; //将v加入遍历树
} //for
```

复杂度

❖ 执行时间主要消耗于内、外两重循环；其中内循环有两个（类），前、后并列

❖ 前一类循环（更新）：若采用**邻接矩阵**，累计耗时 $\mathcal{O}(n^2)$ ；若采用**邻接表**，耗时 $\mathcal{O}(n+e)$

后一类循环（择优）：每次耗时 $\mathcal{O}(n)$ ，累计 $\mathcal{O}(n^2)$

两类合计，为 $\mathcal{O}(n^2)$

❖ 后面将会看到：若采用**优先级队列**，以上两项将分别是 $\mathcal{O}(e \cdot \log n)$ 和 $\mathcal{O}(n \cdot \log n)$ //保持兴趣

合计为 $\mathcal{O}((n+e) \cdot \log n)$

❖ 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O(e + n*logn)的算法

❖ 基于这个统一**框架**，如何解决具体的应用问题...

11-D1

图应用

Dijkstra算法：最短路径

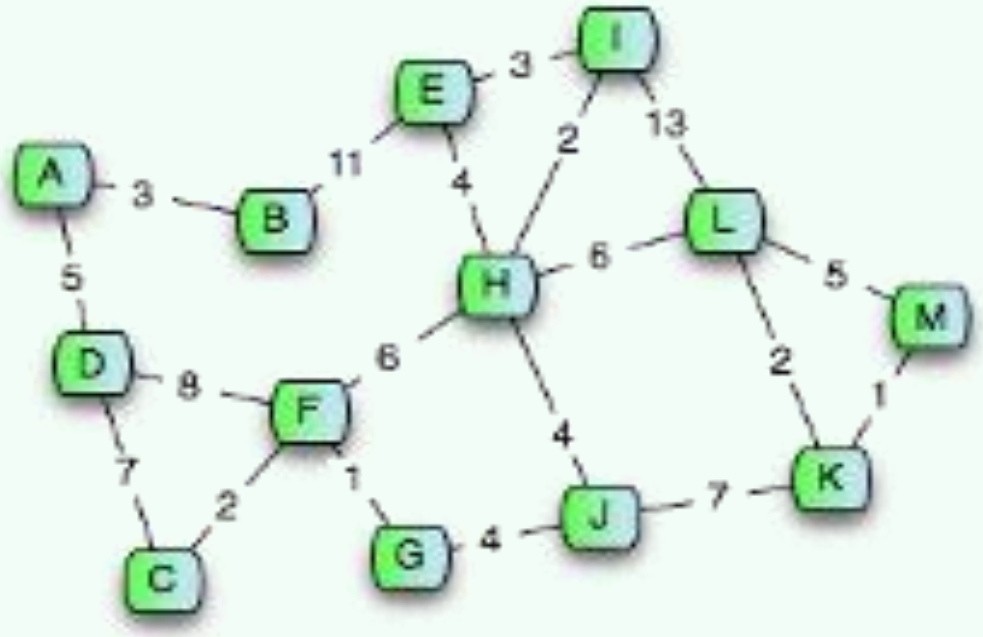
The hurricane seeks the shortest road by the no-road, and suddenly ends its search in the nowhere.

邓俊辉
deng@tsinghua.edu.cn

问题 + 应用

❖ 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

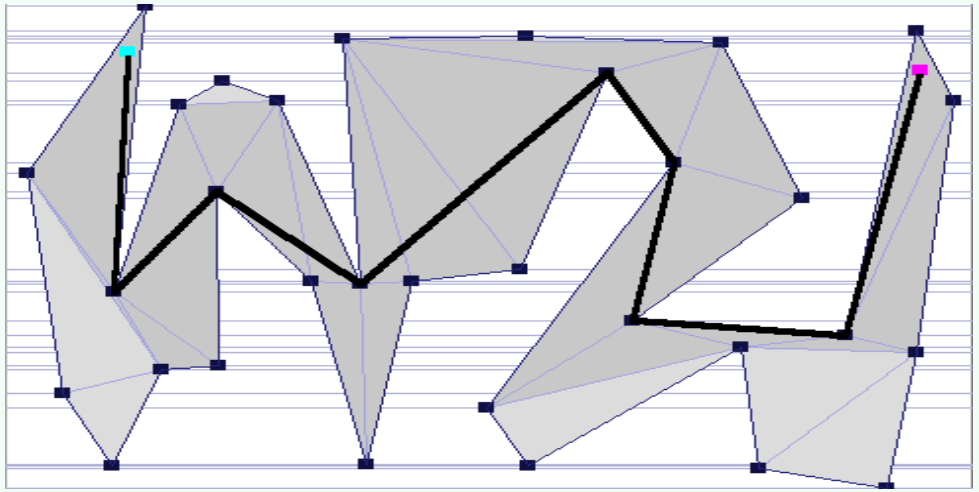


❖ 旅游者：最经济的出行路线

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

.....

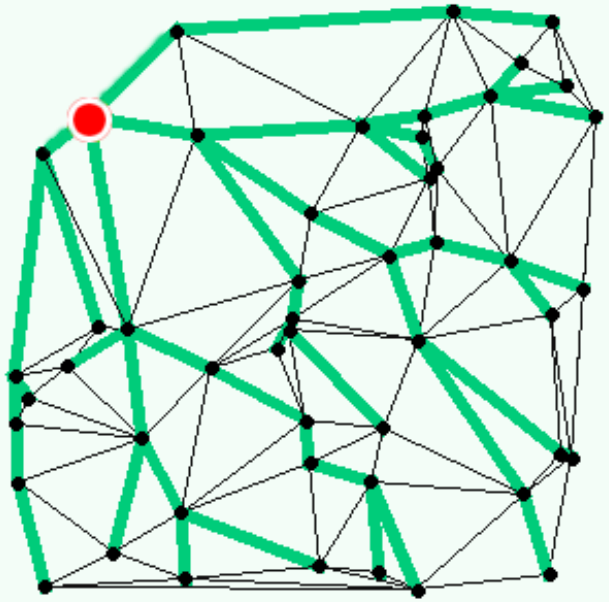


问题分类

❖ SSSP: Single-Source Shortest Path

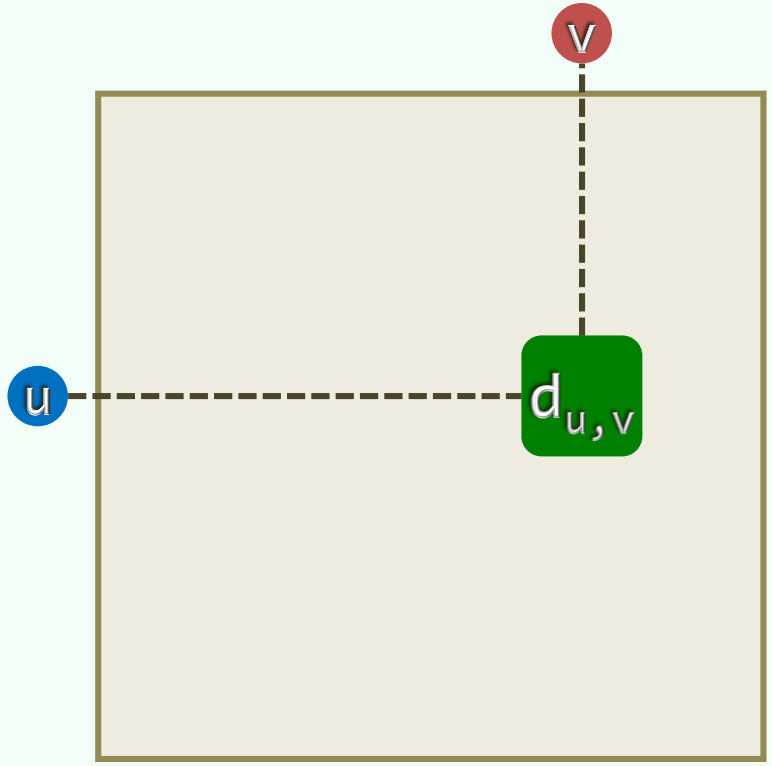
给定顶点s，计算s到

其余各个顶点的最短路径及长度

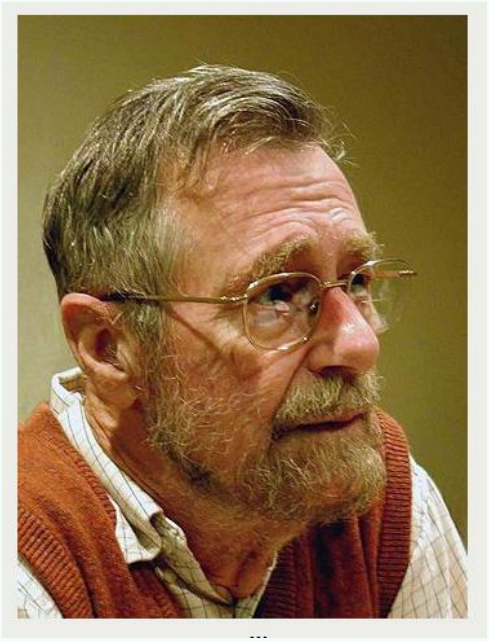


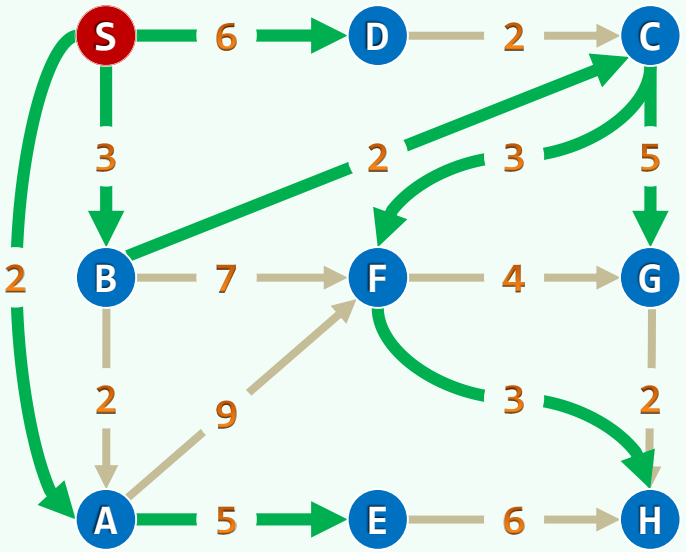
❖ APSP: All-Pairs Shortest Path

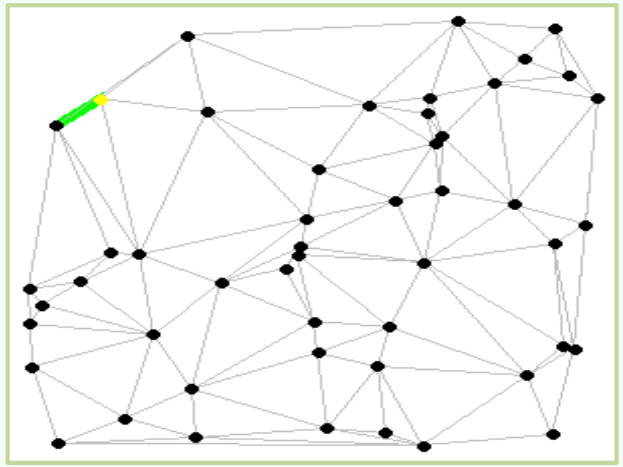
找出每对顶点u和v之间的最短路径及长度



E. W. Dijkstra







1972 ACM Turing Award Lecture

[Extract from the Turing Award Citation read by M.D. McIlroy, chairman of the ACM Turing Award Committee, at the presentation of this lecture on August 14, 1972, at the ACM Annual Conference in Boston.]

The working vocabulary of programmers everywhere is studded with words originated or forcefully promulgated by E.W. Dijkstra—display, deadly embrace, semaphore, go-to-less programming, structured programming. But his influence on programming is more pervasive than any glossary can possibly indicate. The precious gift that this Turing Award acknowledges is Dijkstra's *style*: his approach to programming as a high, intellectual challenge; his eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness; and his illuminating perception of problems at the foundations of program design. He has published about a dozen papers, both technical and reflective, among which are especially to be noted his philosophical addresses at IFIP,[1] his already classic papers on cooperating sequential processes,[2] and his memorable indictment of the go-to statement.[3] An influential series of letters by Dijkstra have recently surfaced as a polished monograph on the art of composing programs.[4]

We have come to value good programs in much the same way as we value good literature. And at the center of this movement, creating and reflecting patterns no less beautiful than useful, stands E.W. Dijkstra.

The Humble Programmer

by Edsger W. Dijkstra

As a result of a long sequence of coincidences I entered the programming profession officially on the first spring morning of 1952, and as far as I have been able to trace, I was the first Dutchman to do so in my country. In retrospect the most amazing thing is the slowness with which, at least in my part of the world, the programming profession emerged, a slowness which is now hard to believe. But I am grateful for two vivid recollections from that period that establish that slowness beyond any doubt.

After having programmed for some three years, I had a discussion with van Wijngaarden, who was then my boss at the Mathematical Centre in Amsterdam—a discussion for which I shall remain grateful to him as long as I live. The point was that I was supposed to study theoretical physics at the University of Leiden simultaneously, and as I found the two activities harder and harder to combine, I had to make up my mind, either to stop programming and become a real, respectable theoretical physicist, or to carry my study of physics to a formal completion only, with a minimum of effort, and to become . . . , yes what? A programmer? But was that a respectable profession? After all, what was programming? Where was the sound body of knowledge that could sup-

1,2,3,4 Footnotes are on page 866.

11-D2

图应用

Dijkstra算法：最短路径树

邓俊辉

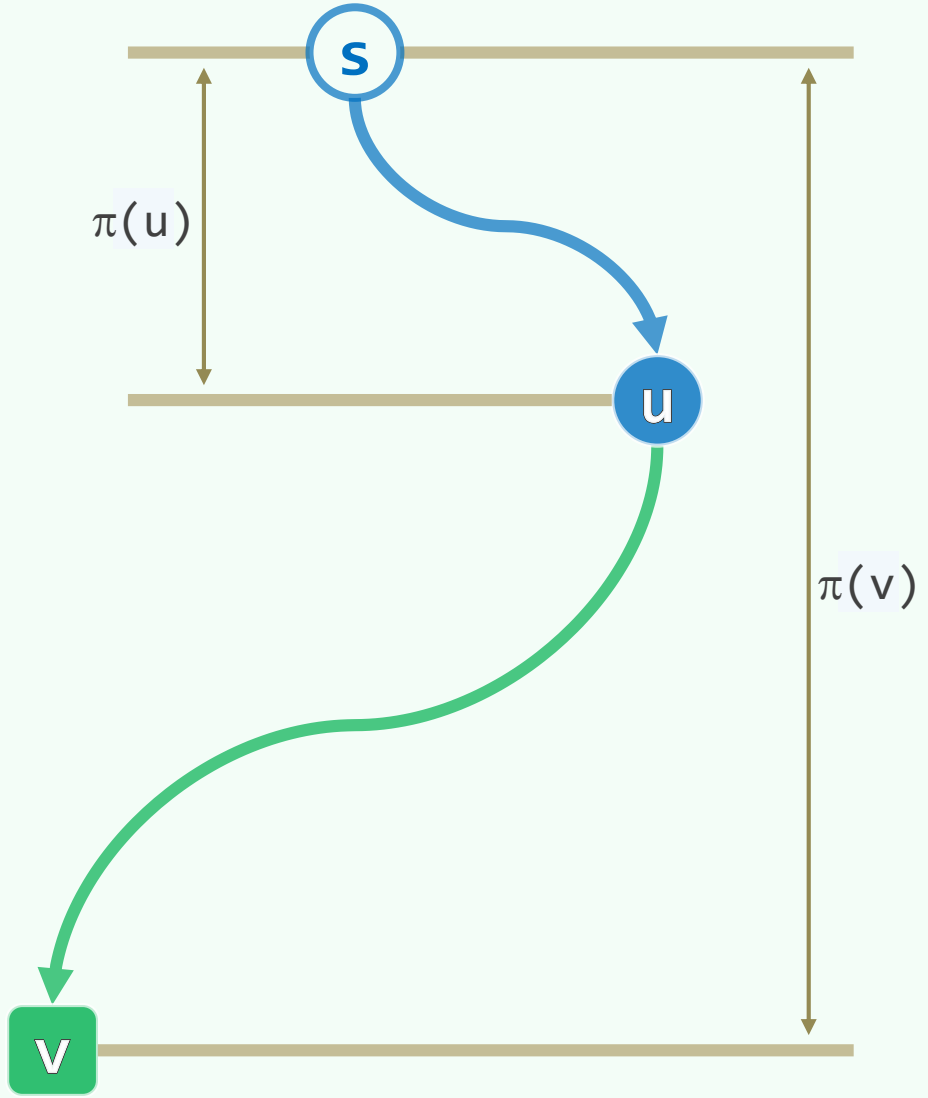
deng@tsinghua.edu.cn

有澹台灭明者，行不由径；非公事，未尝至于偃之室也

单调性 + 假想实验

❖ 任一最短路径的前缀，也是一条最短路径

$$u \in \pi(v) \quad \text{only if} \quad \pi(u) \subseteq \pi(v)$$

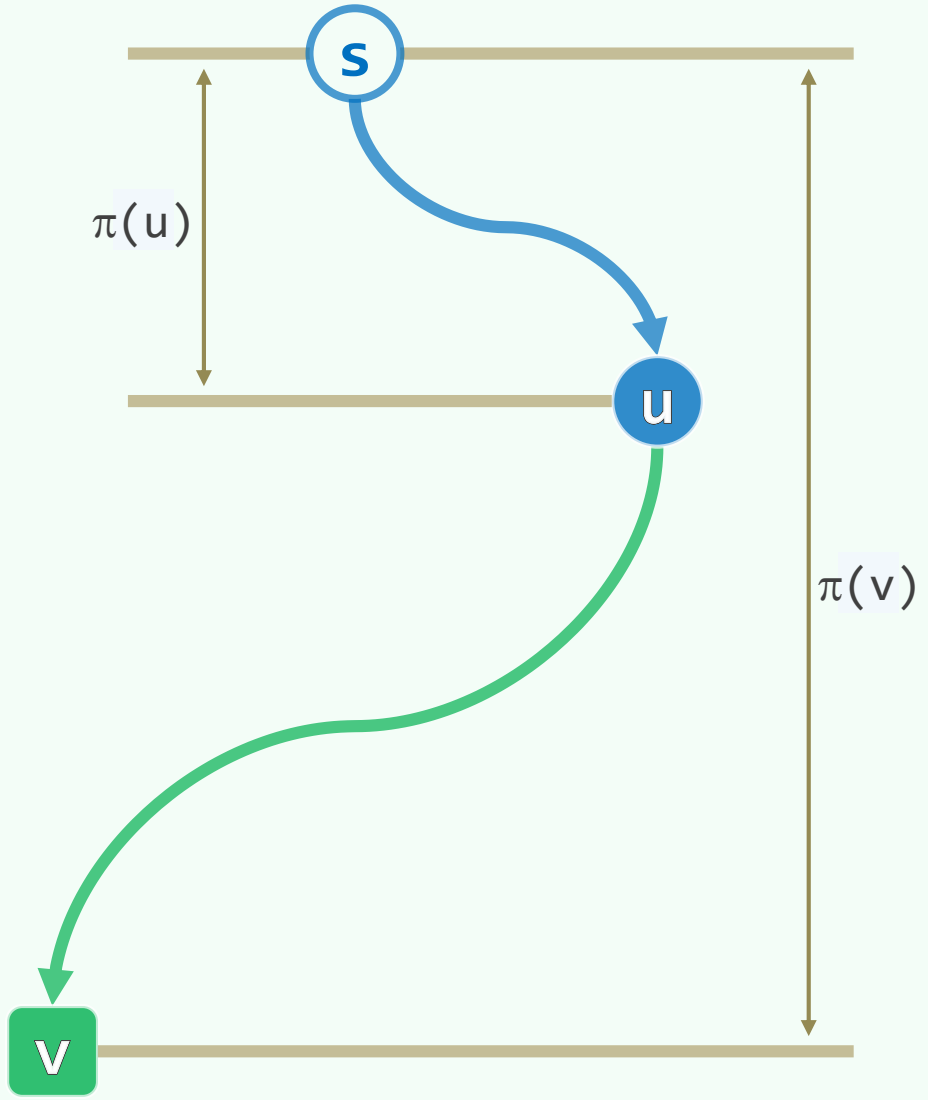


消除歧义

- 各边权重均为正，否则
  - 有可能出现总权重非正的环路
  - 以致最短路径无从定义
- 有负权重的边时，即便所有环路总权重皆为正

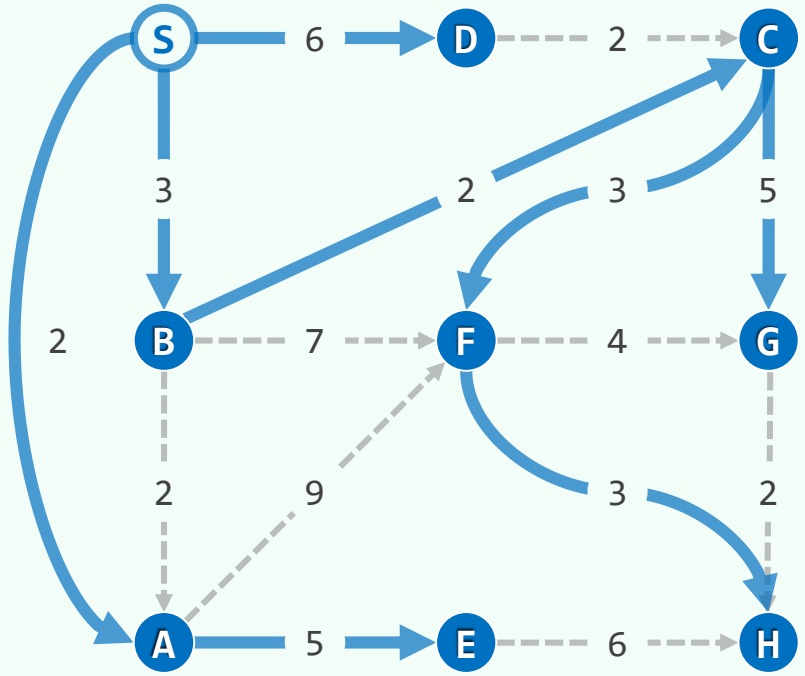
  以下将介绍的Dijkstra算法依然可能失效
- 任意两点之间，最短路径唯一
  - 在不影响计算结果的前提下

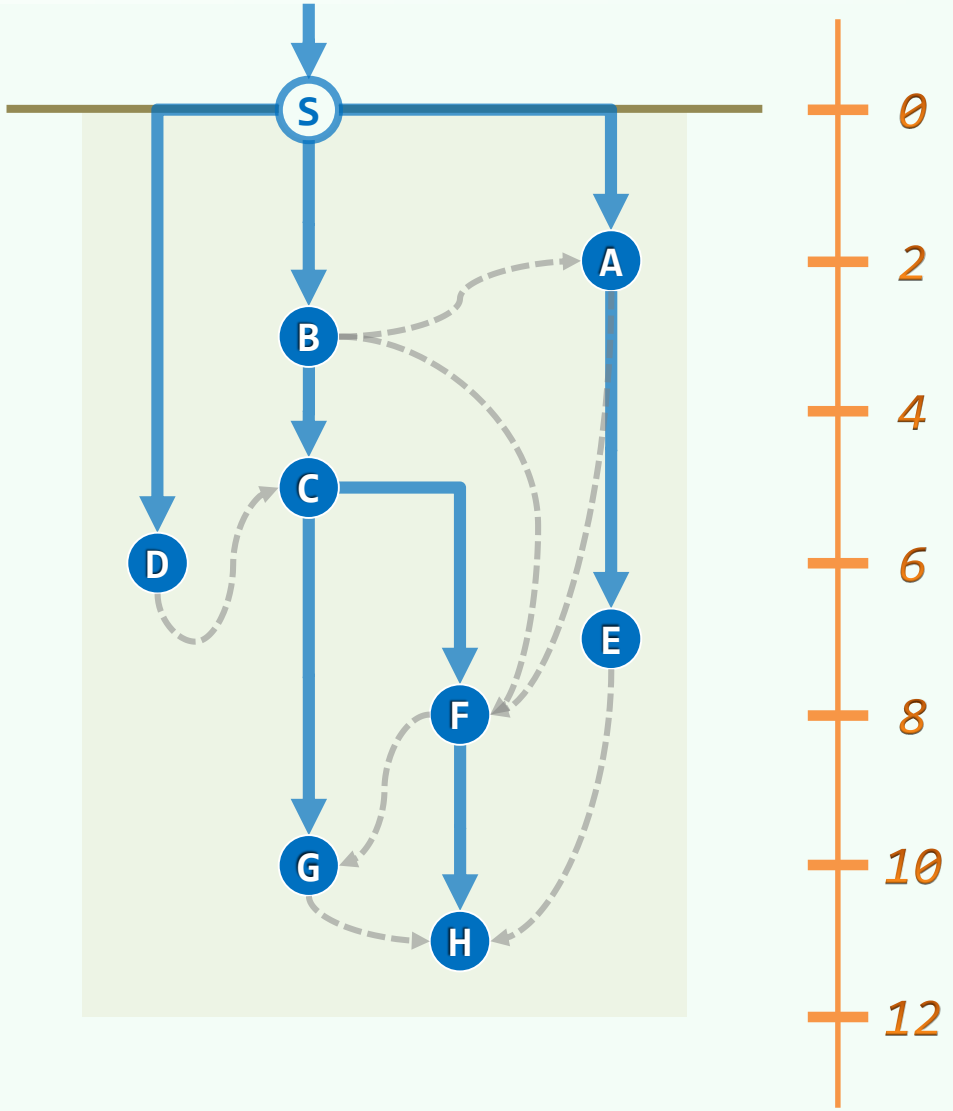
    总可通过适当扰动予以保证（习题[6-17]）



Shortest Path Tree

❖ 所有最短路径的并，既连通亦无环





❖ 于是，$\mathcal{T} = \mathcal{T}_{n-1} = \bigcup_{0 \le i < n} \pi(u_i)$ 构成一棵树

图应用

Dijkstra算法：算法

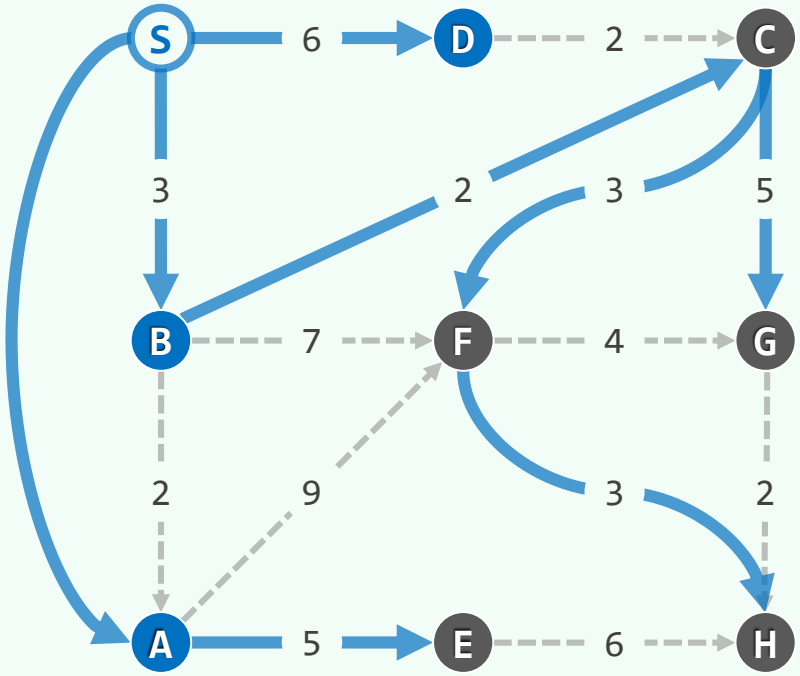
11-D3

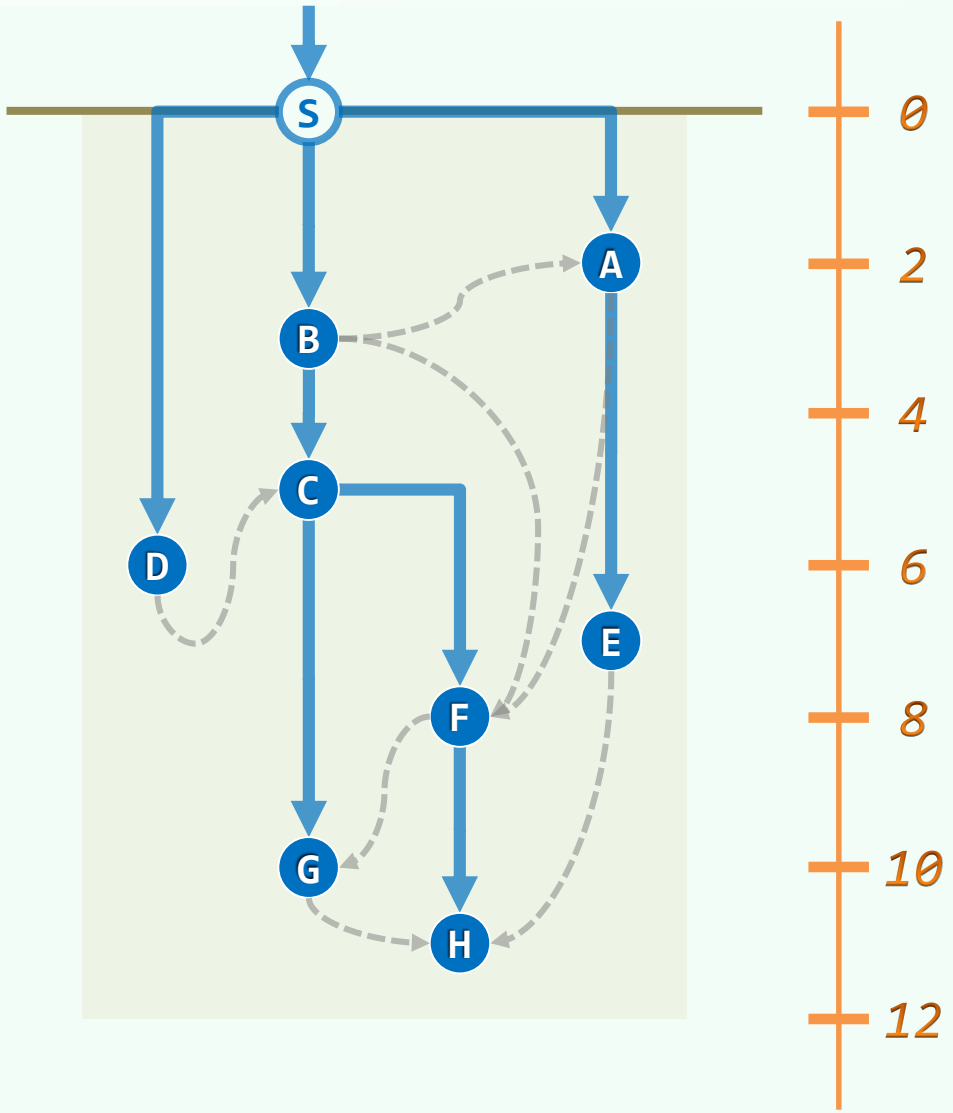
那好吧，我再告诉你一件事：赫尔辛根山靠着海，在山顶能看到默斯肯岛，默斯肯岛是距赫尔辛根山最近的一座海岛！

邓俊辉

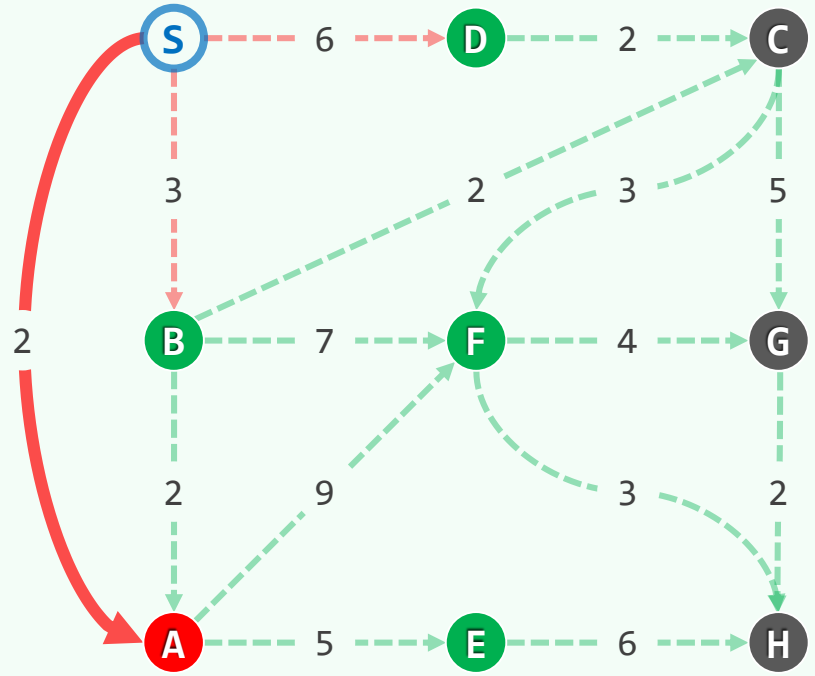
deng@tsinghua.edu.cn

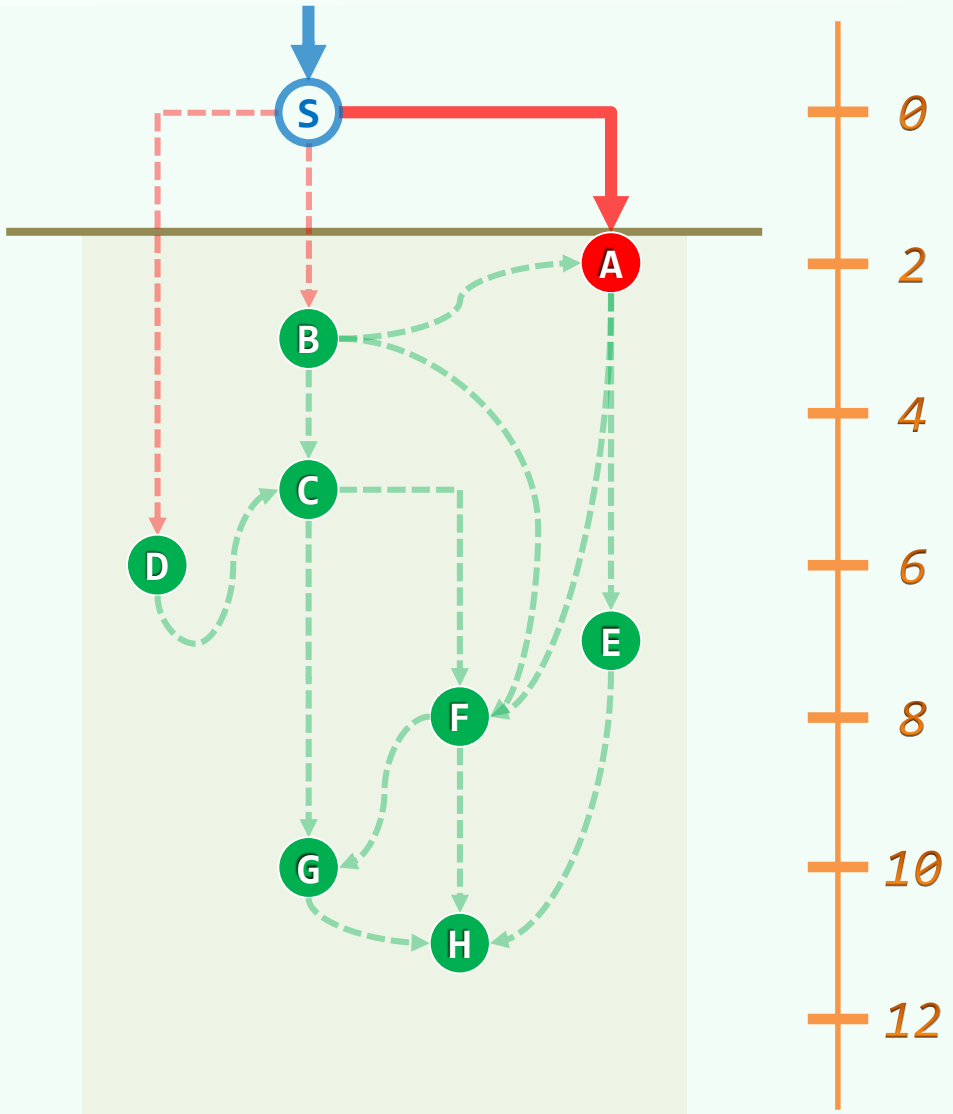
减而治之 = 遍历 = PFS



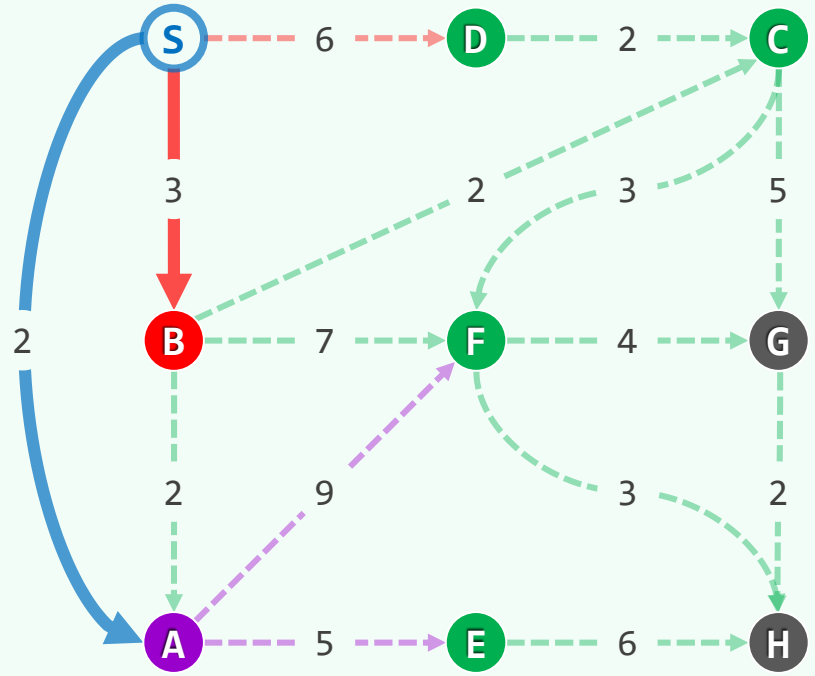


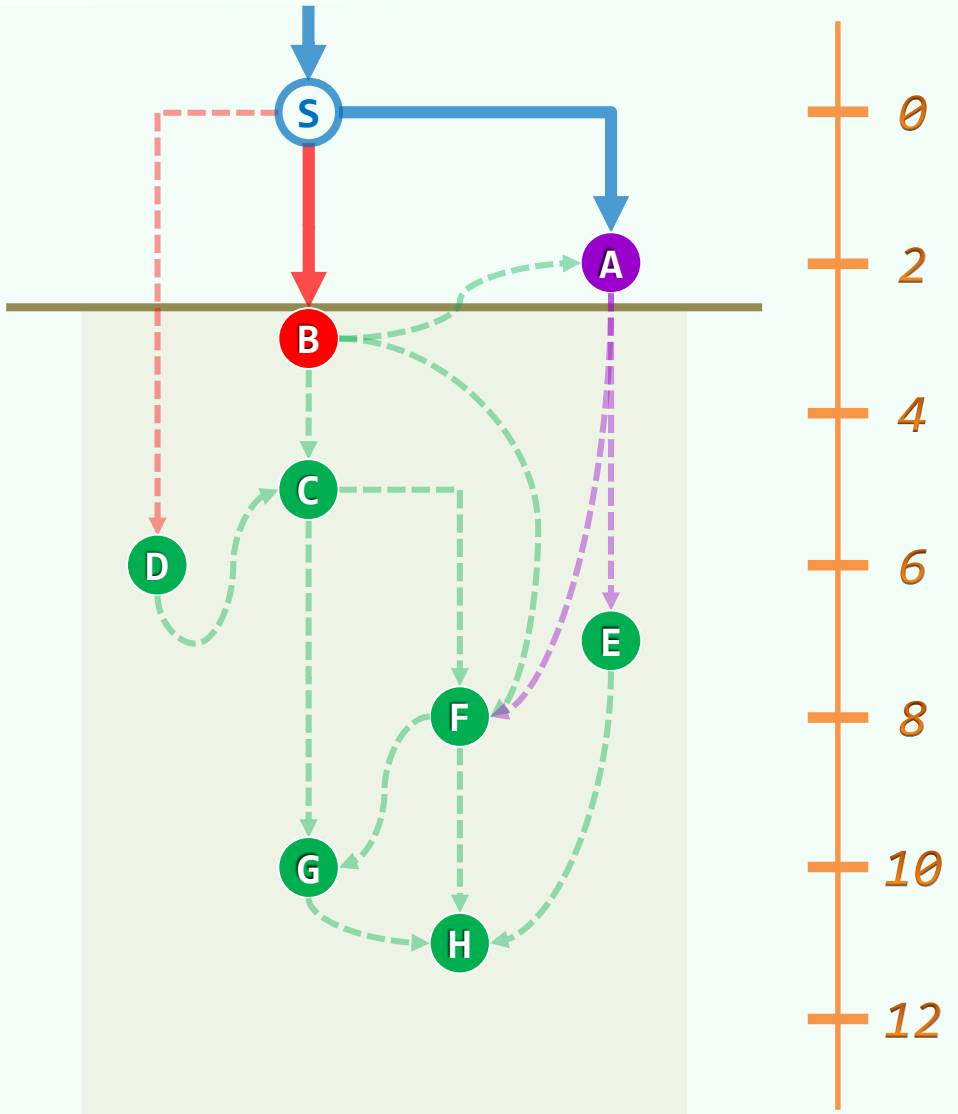
S起步，确定A



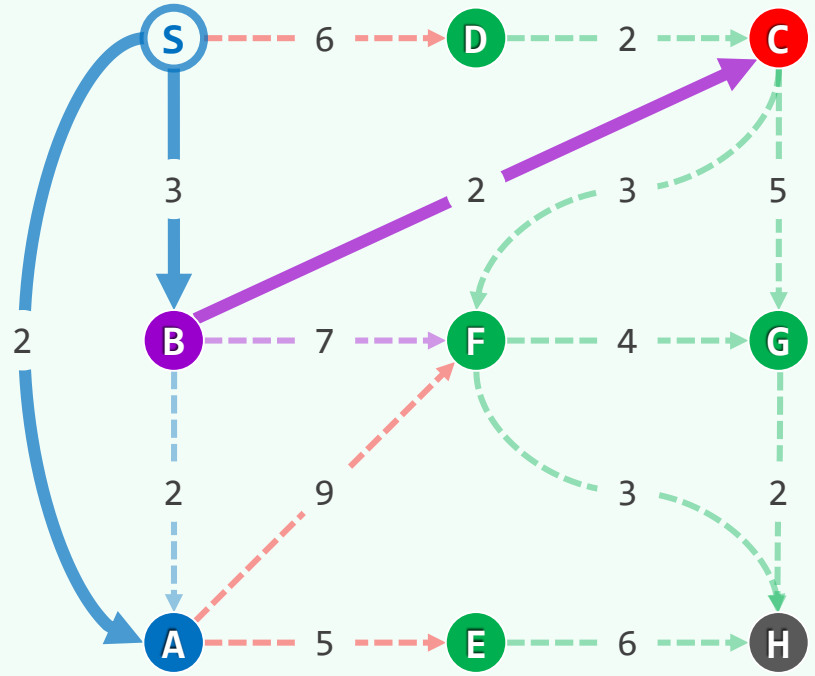


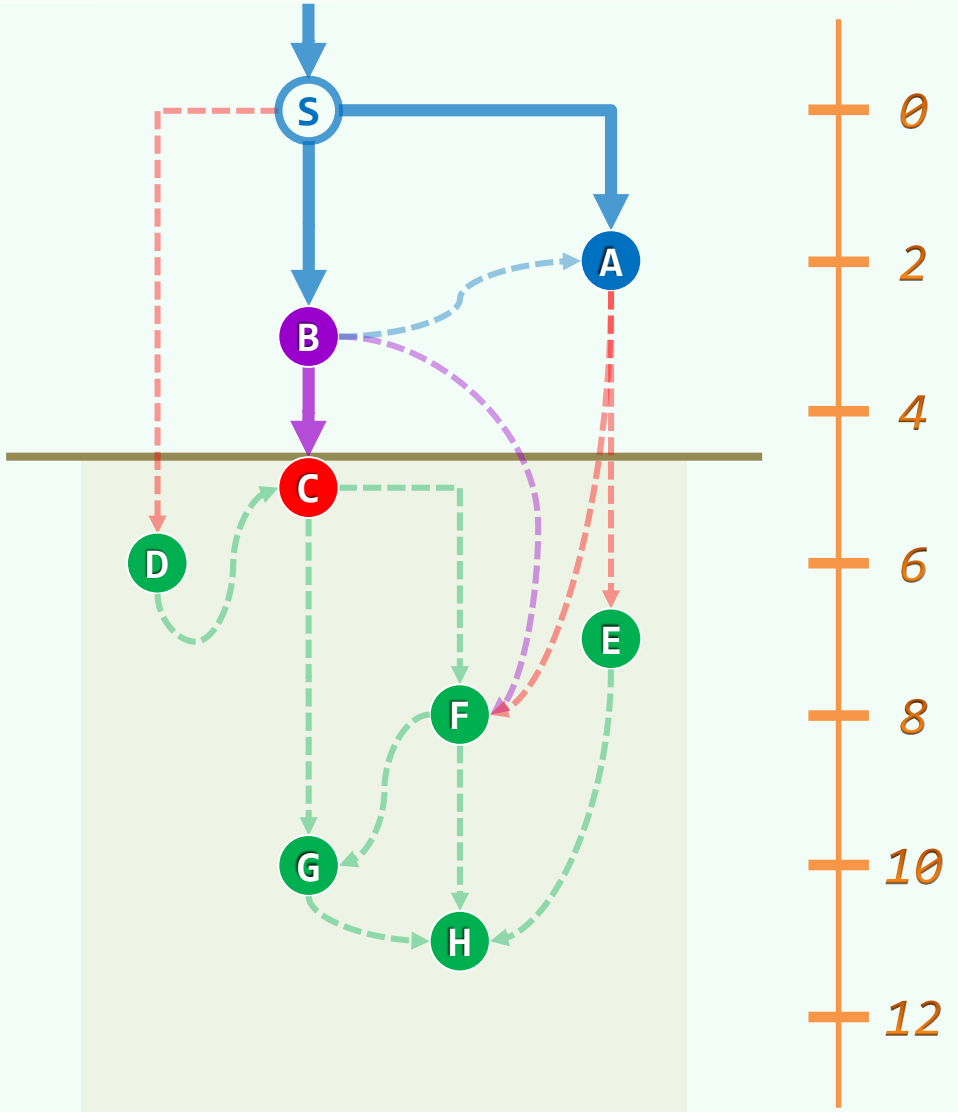
A入选，确定B



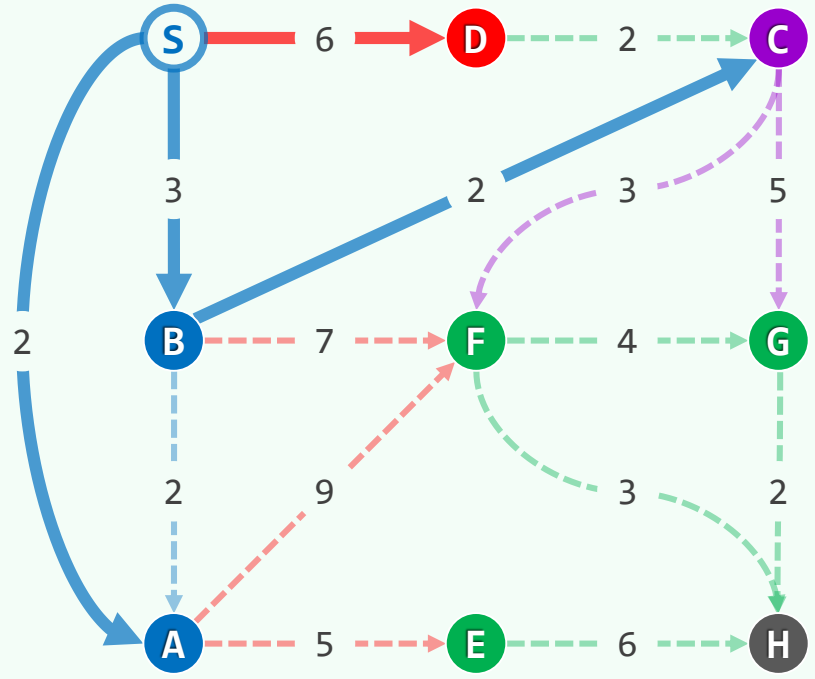


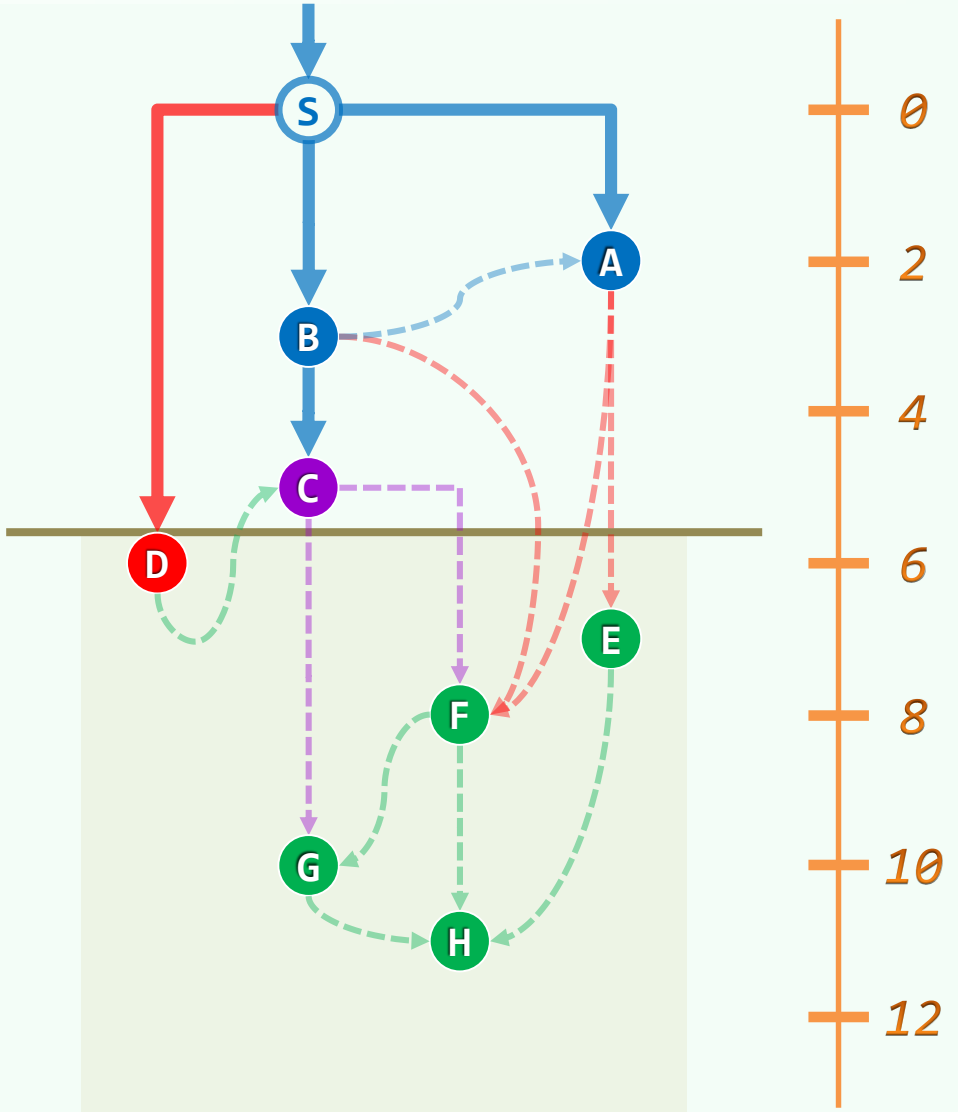
B入选，确定C



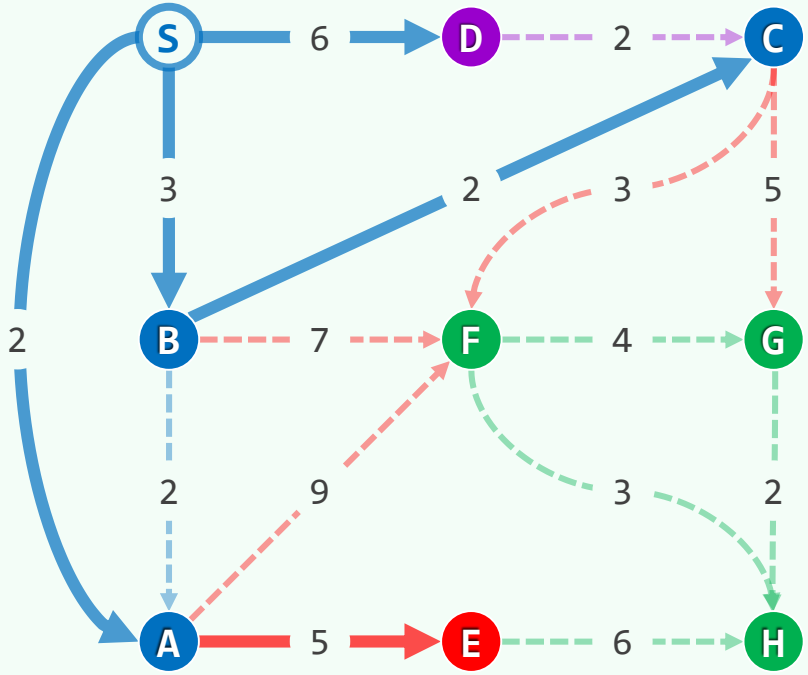


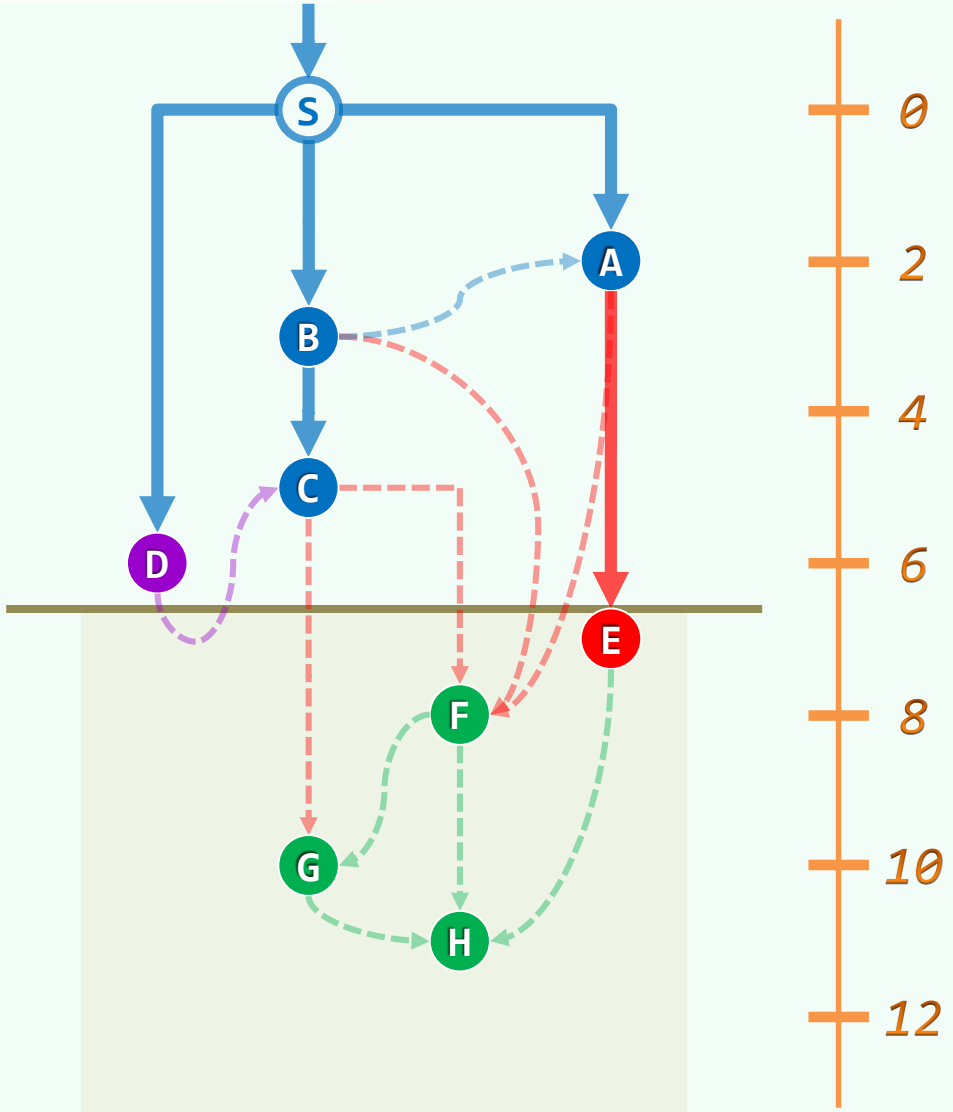
C入选，确定D





D入选，确定E





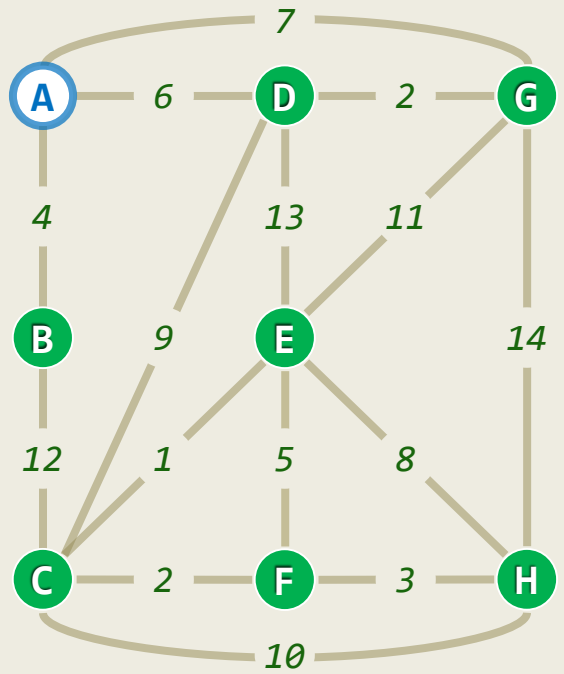
图应用

Dijkstra算法：实例

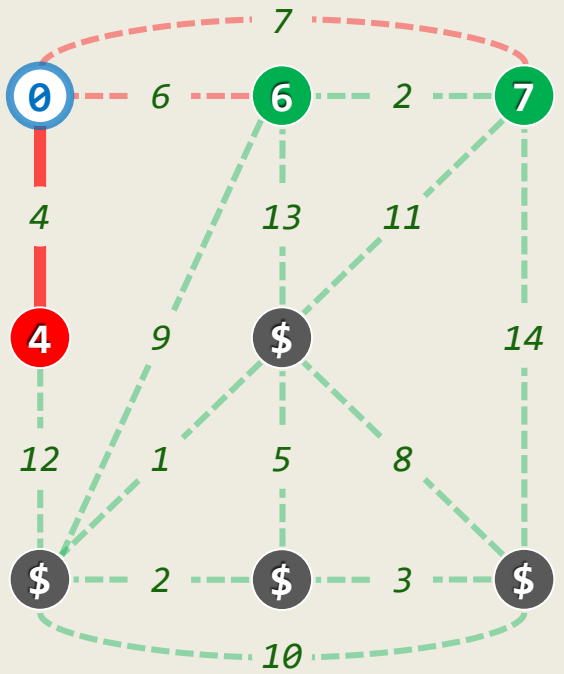
11-D4

我们旅行时不要像个信使一样，而应当像个探险家一样。我们不仅要考虑起点和终点，还得考虑起点和终点之间的距离

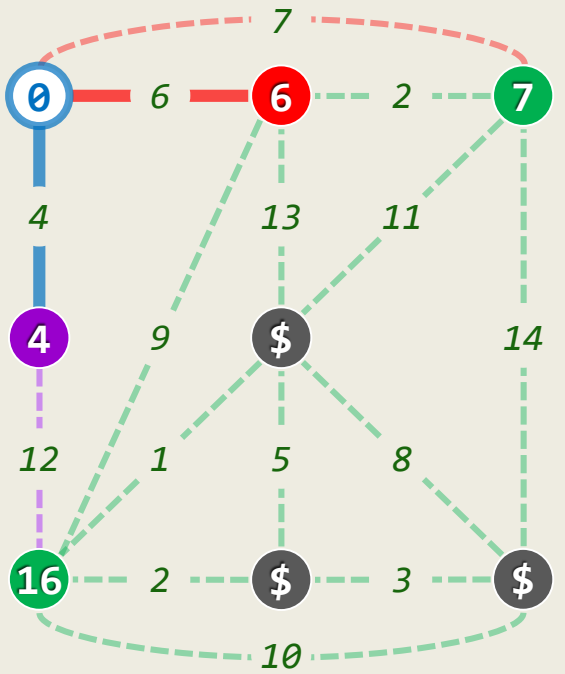
邓俊辉
deng@tsinghua.edu.cn


A
D
G
B
E
C
F
H
7
6
2
4
13
11
9
14
12
1
5
8
2
3
10

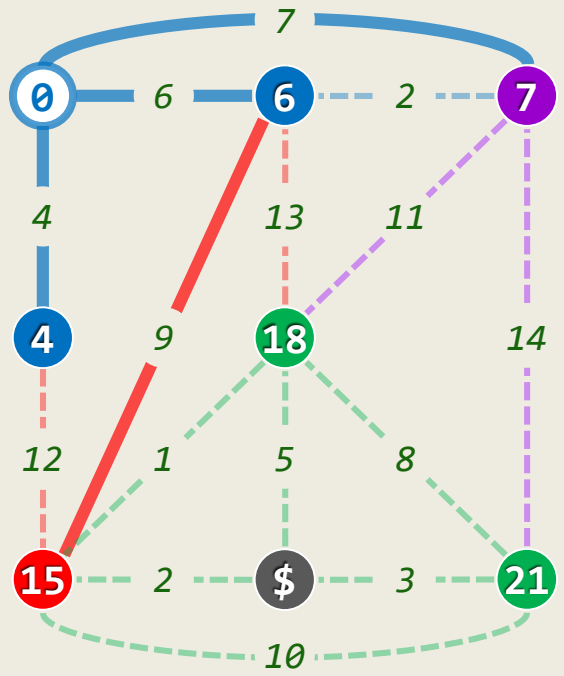
(a)


0
6
7
4
$
12
1
5
8
2
3
10

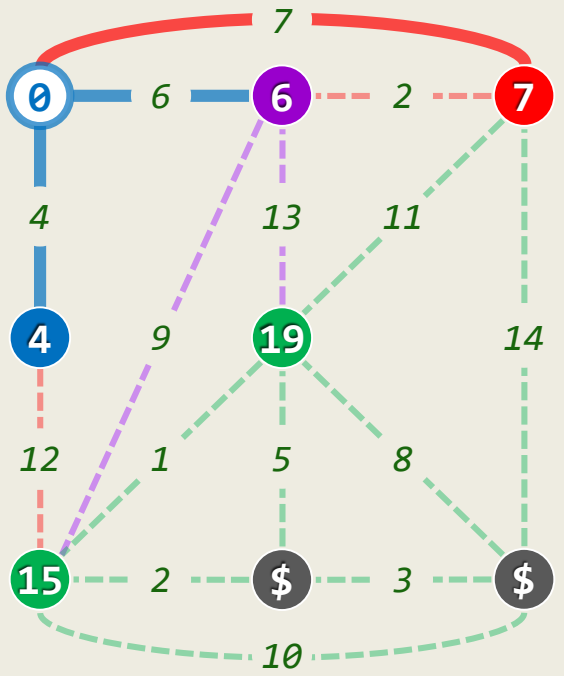
(b)


0
6
7
4
16
$
12
1
5
8
2
3
10

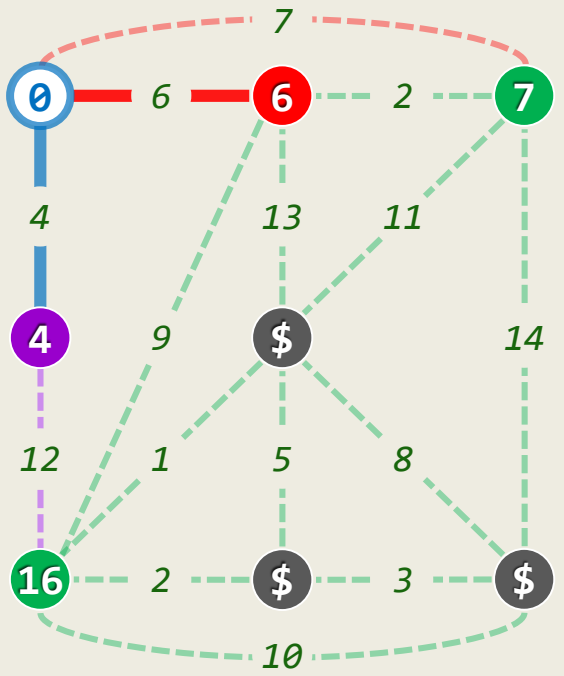
(c)


7
0
6
6
2
7
4
13
11
4
9
18
14
12
1
5
8
15
2
$
3
21
10

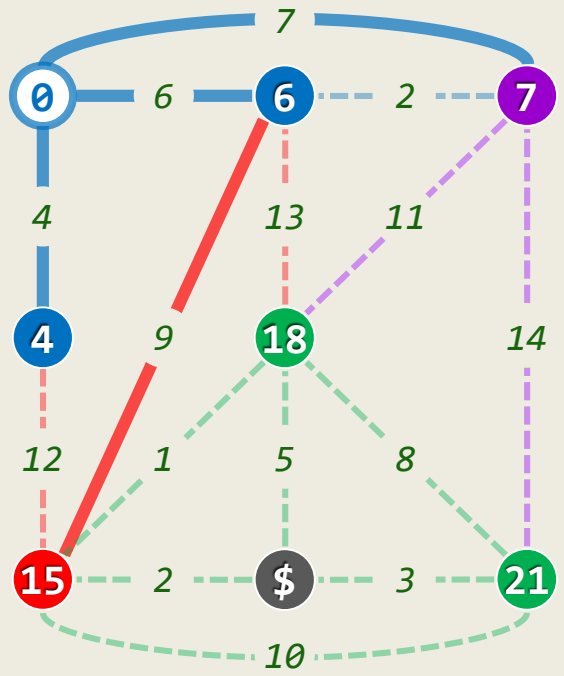
(e)


7
0
6
6
2
7
4
13
11
4
9
19
14
12
1
5
8
15
2
$
3
$
10

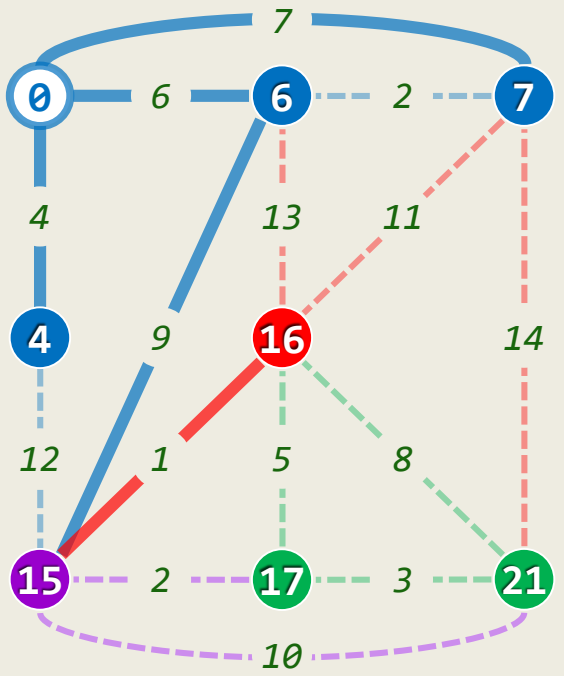
(d)


7
0
6
6
2
7
4
13
11
4
9
$
14
12
1
5
8
16
2
$
3
$
10

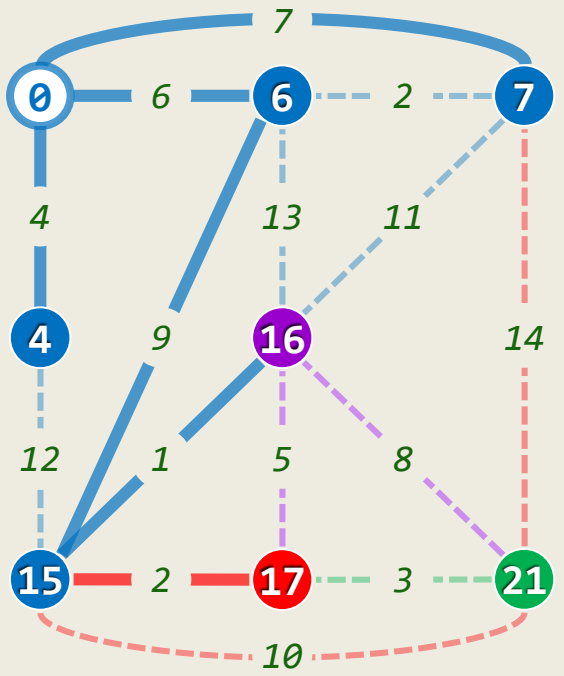
(c)


7
0
6
6
2
7
4
13
11
4
9
18
14
12
1
5
8
15
2
$
3
21
10

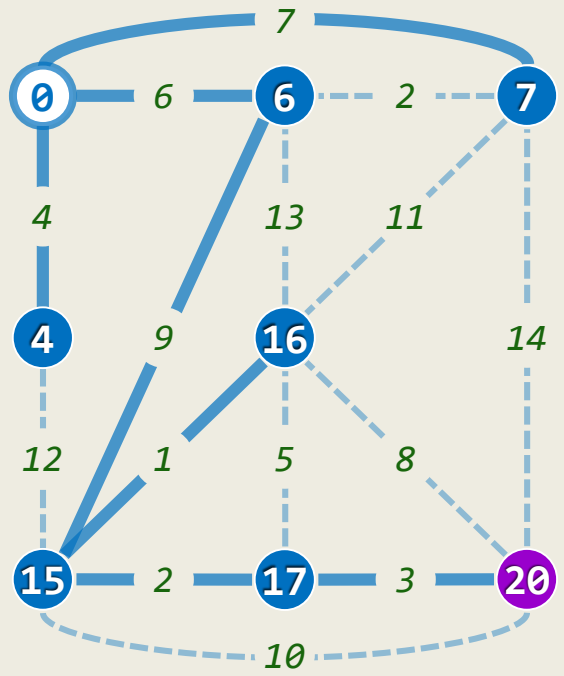
(e)


7
0
6
6
2
7
4
13
11
4
9
16
14
12
1
5
8
15
2
17
3
21
10

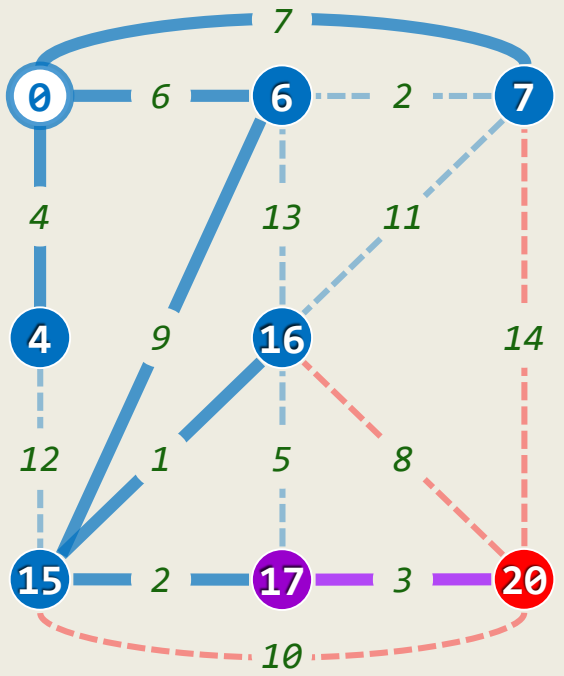
(f)


7
0
6
6
2
7
4
13
11
4
9
16
14
12
1
5
8
15
2
17
3
21
10

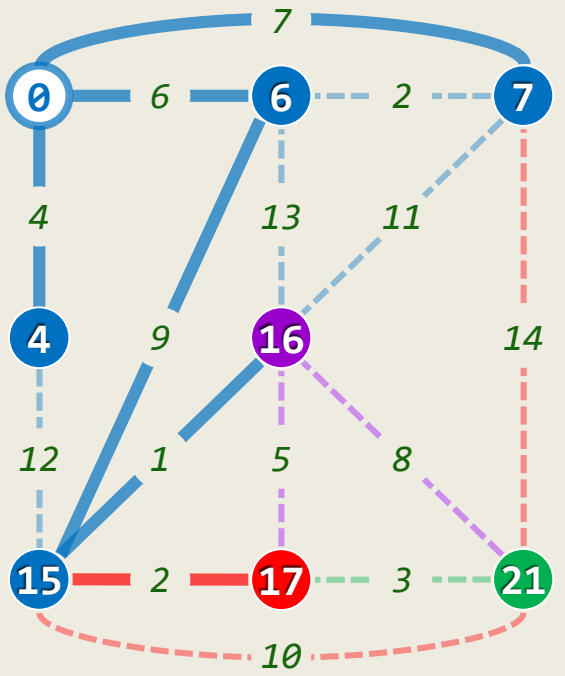
(g)


7
0
6
6
2
7
4
13
11
4
9
16
14
12
1
5
8
15
2
17
3
20
10

(i)


7
0
6
6
2
7
4
13
11
4
9
16
14
12
1
5
8
15
2
17
3
20
10

(h)


7
0
6
6
2
7
4
13
11
4
9
16
14
12
1
5
8
15
2
17
3
21
10

(g)

11-D5

图应用

Dijkstra算法：实现

他永远也不能了解，也不能向自己解释，他当时已经疲惫不堪、精疲力竭了，他理应走最短最直的路回家，而他为什么偏要走没有必要经过的干草市场回家去呢？虽然绕的路并不远，但是这显然完全没有必要

邓俊辉

deng@tsinghua.edu.cn

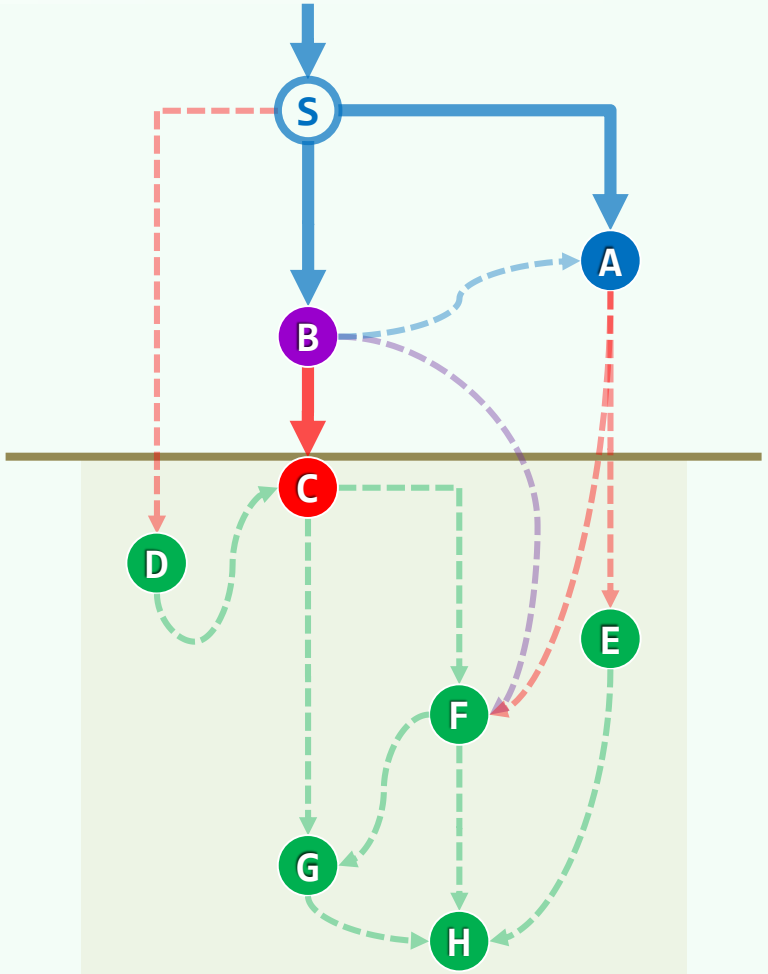
PFS

- $\forall v \notin V_k$, let $\text{priority}(v) = \|s, v\| \leq \infty$
- 于是套用PFS框架，为将 T_k 扩充至 T_{k+1}，只需
  - 选出优先级最高的跨边 e_k 及其对应顶点 v_k，并将其加入 T_k
  - 随后，更新 $V \backslash V_{k+1}$ 中所有顶点的优先级（数）
- 注意：优先级数随后可能改变（降低）的顶点，必与 v_k 邻接
- 因此，只需枚举 v_k 的每一邻接顶点 u，并取

$$\text{priority}(u) = \min(\text{priority}(u), \text{priority}(v_k) + \|v_k, u\|)$$

- 以上完全符合PFS的框架，唯一要做的工作无非是

  按照prioUpdater()规范，编写一个优先级（数）更新器...



Priority Updater ~ DijkPU

```
g->pfs( 0, DijkPU<char, Rank>() ); //从顶点0出发，启动Dijkstra算法

template <typename Tv, typename Te> struct DijkPU { //Dijkstra算法的优先级更新器
   virtual void operator()( Graph<Tv, Te>* g, Rank v, Rank u ) { //对v的每个
      if ( UNDISCOVERED != g->status(u) ) return; //尚未被发现的邻居u，按
      if ( g->priority(u) > g->priority(v) + g->weight(v, u) ) { //Dijkstra
         g->priority(u) = g->priority(v) + g->weight(v, u); //策略
         g->parent(u)   = v; //做松弛
      }
   }
};
```

11-E1

图应用

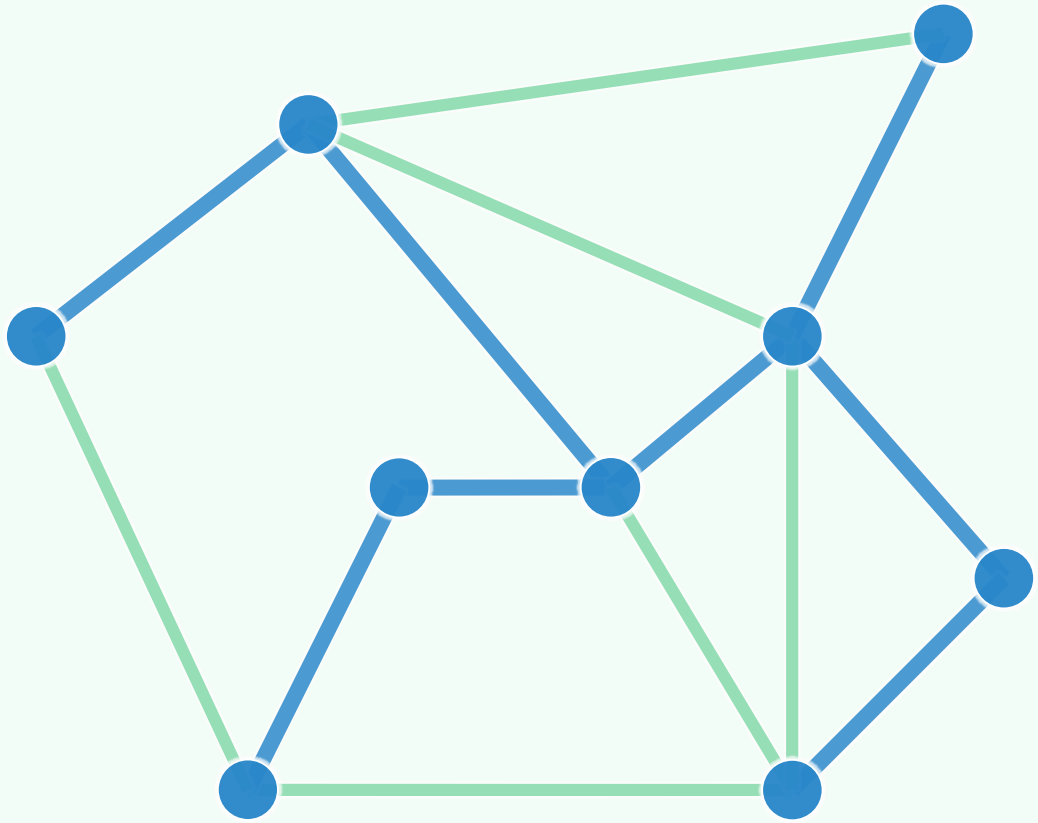
Prim算法：最小支撑树

还有更荒唐的事呢，他要在普济造一条风雨长廊，把村里的每一户人家都连接起来，哈哈，他以为，这样一来，普济人就可免除日晒雨淋之苦了

邓俊辉
deng@tsinghua.edu.cn

最小 + 支撑 + 树

- 连通网络 $N = (V; E)$ 的
  子图 $T = (V; F)$
- 支撑/spanning = 覆盖N中所有顶点
- 树/tree = 连通且无环，$|F| = |V| - 1$
- 同一网络的支撑树，未必唯一
- minimum = optimal:
  总权重 $\mathrm{w}(T) = \sum_{e \in F} \mathrm{w}(e)$ 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、...

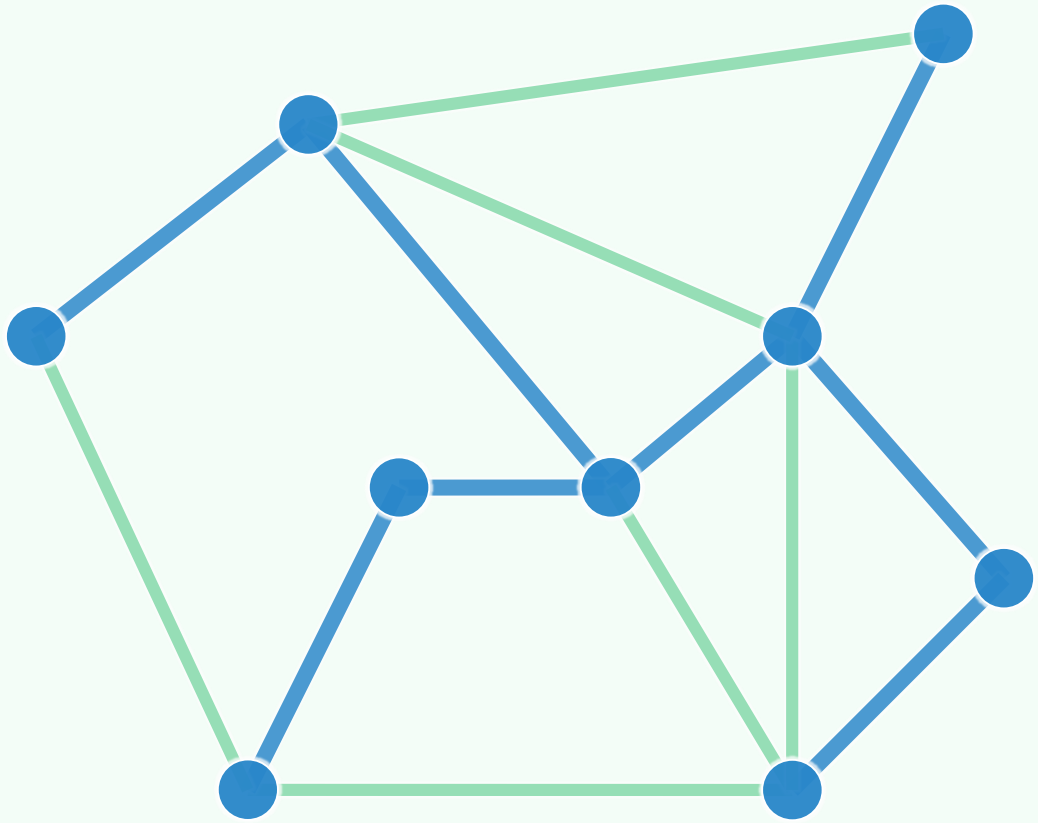


MST

❖ 为何重要?

- 自身可有效计算
- 众多优化问题的基本模型
- 为许多NP问题提供足够好的近似解

比如，Euclidean TSP



❖ 延伸问题： Proximity Graphs //Ω(nlogn)

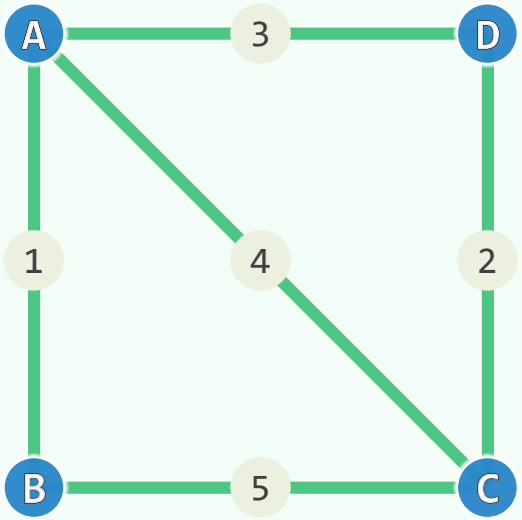
Steiner MST //NP-hard

❖ 众多算法： Boruvka-1926、Jarnik-1930、

Prim-1956、Kruskal-1956

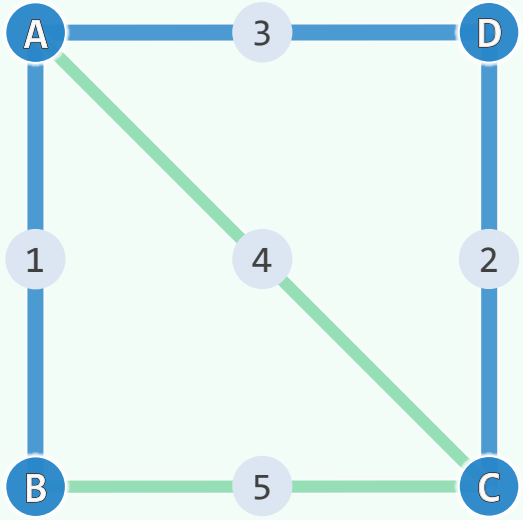
Karger-Klein-Tarjan-1995、Chazelle-2000、...

MST ≠ SPT

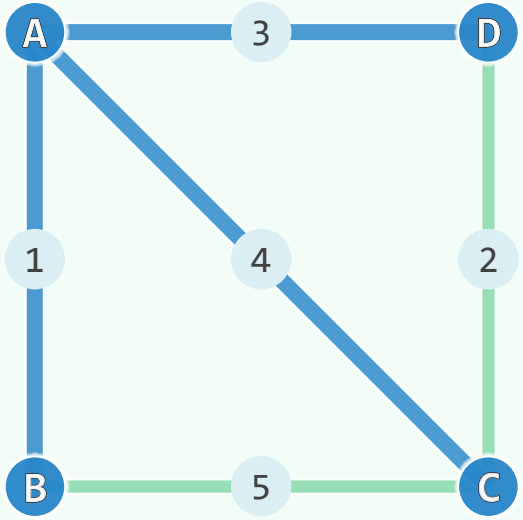


G

MST



SPT



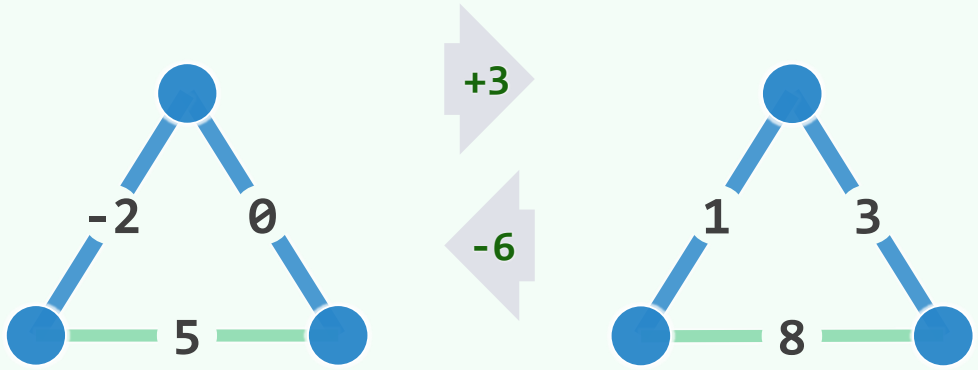
负权 & 退化

❖ 权值必须是正数?

- 允许为零，有何影响?
- 允许为负数呢?

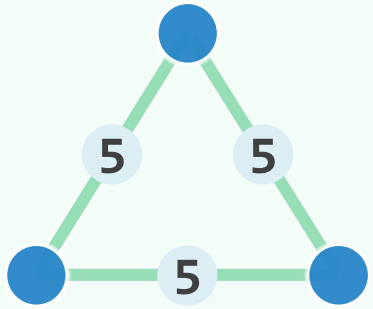
❖ 支撑树所含边数必相等，故可统一调整:

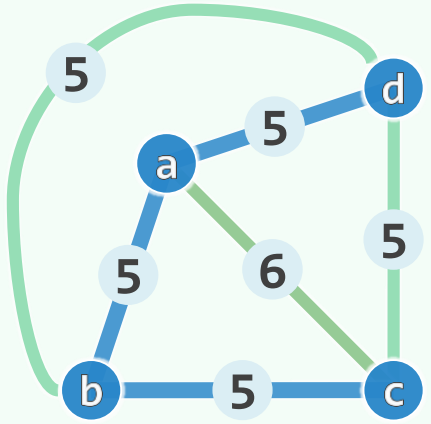
```
increase(1 - findMin())
```



❖ The minimum? A minimal!

同一网络，或有多棵MST: 可强制消除歧义...





❖ 合成数 (composite number) :

$$(\mathbf{w}(u,v),\ \min(u,v),\ \max(u,v))$$

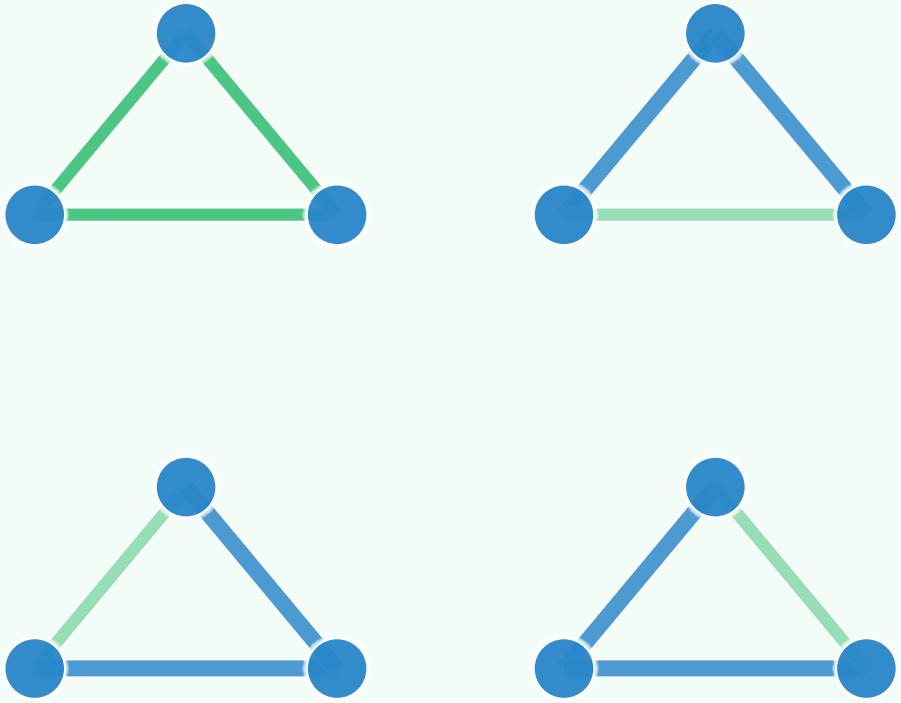
5ab < 5ad < 5bc < 5bd < 5cd < 6ac

蛮力算法

❖ 枚举出N的所有支撑树，从中找出代价最小者

❖ Cayley公式：完全图 $\mathcal{K}_n$ 有 n^{n-2} 棵支撑树

n = 1	1
n = 2	1
n = 3	3
n = 4	16
n = 5	125
...	...



❖ 如何高效地构造MST呢?

11-E2

图应用

Prim算法：极短跨边

从邻枝上切下的一根枝条，必定也是从整个树上切下的。所以，
一个人若同另一个人分离，他也是同整个社会分离

邓俊辉
deng@tsinghua.edu.cn

Excluding The Longest Edge Along A Cycle

- 任何环路C上的最长边f，都不会被MST采用

  否则...

- 在移除f之后，MST将分裂为两棵树

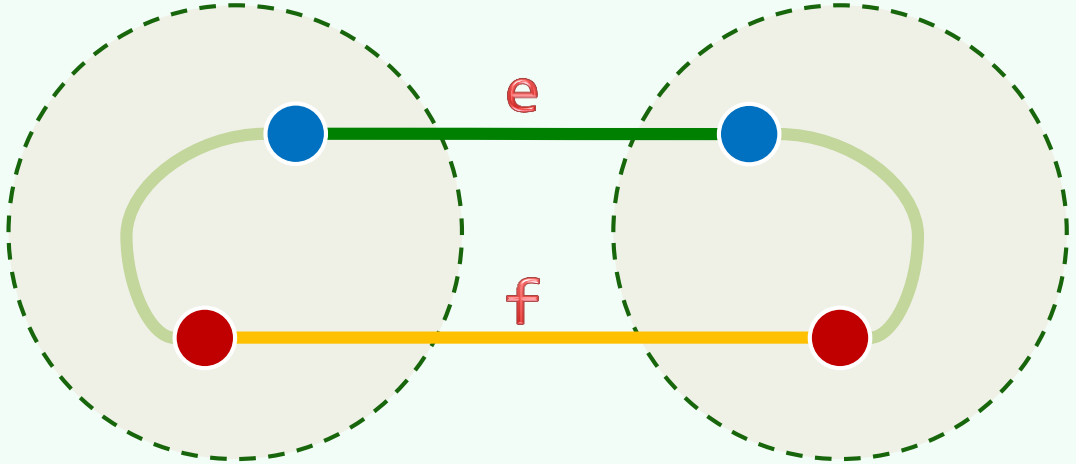
  将其视作一个割，则C上必有该割的另一跨边e

  既然|e|<|f|，那么只要用e替换f，就会...

  ...得到一棵总权重更小的支撑树

- 这也是Kruskal算法的依据（稍后细解）

- 下面这个准则，才是Prim算法的依据...



Including The Shortest Edge Crossing A Cut

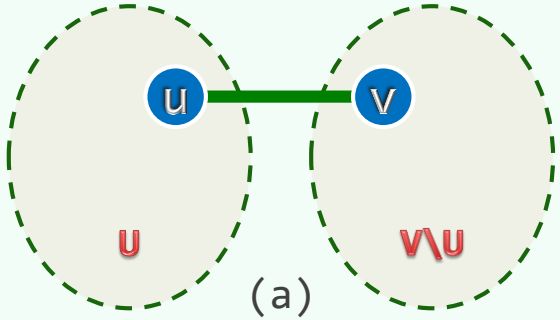
- 设$(U : V\backslash U)$是N的一个割
- 若uv是该割的一条极短跨边

  则必存在一棵包含uv的MST
- 反证：假设uv未被任何MST采用...

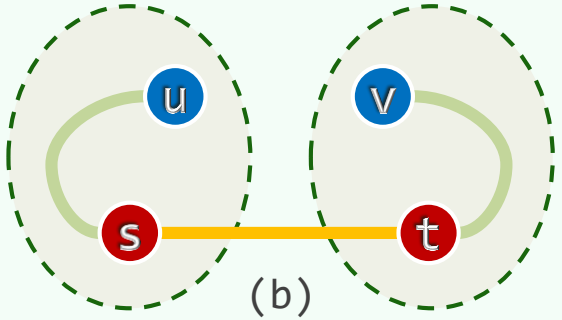
  任取一棵MST，将uv加入其中，于是
  - 将出现唯一的回路，且该回路
  - 必经过uv以及至少另一跨边st

  接下来，摘除st后...

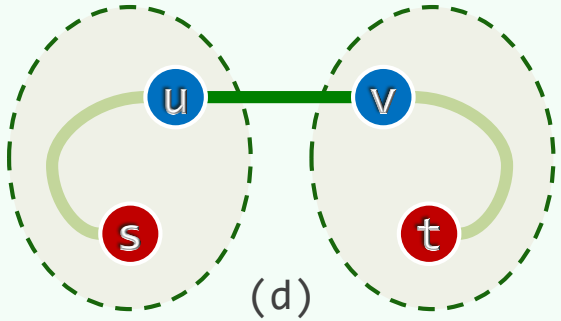
  恢复为一棵支撑树，且总权重不致增加
- 反之，任一MST都必经由极短跨边联接每一割



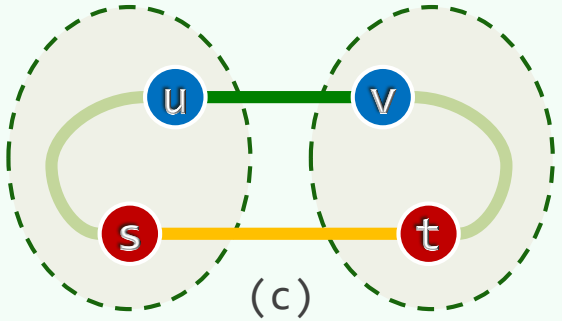
(a)



(b)



(d)



(c)

递增式构造

❖ 首先，任选：$T_1 = (\{v_1\}; \varnothing)$

❖ 以下，不断地将 T_k 拓展为树 T_{k+1}

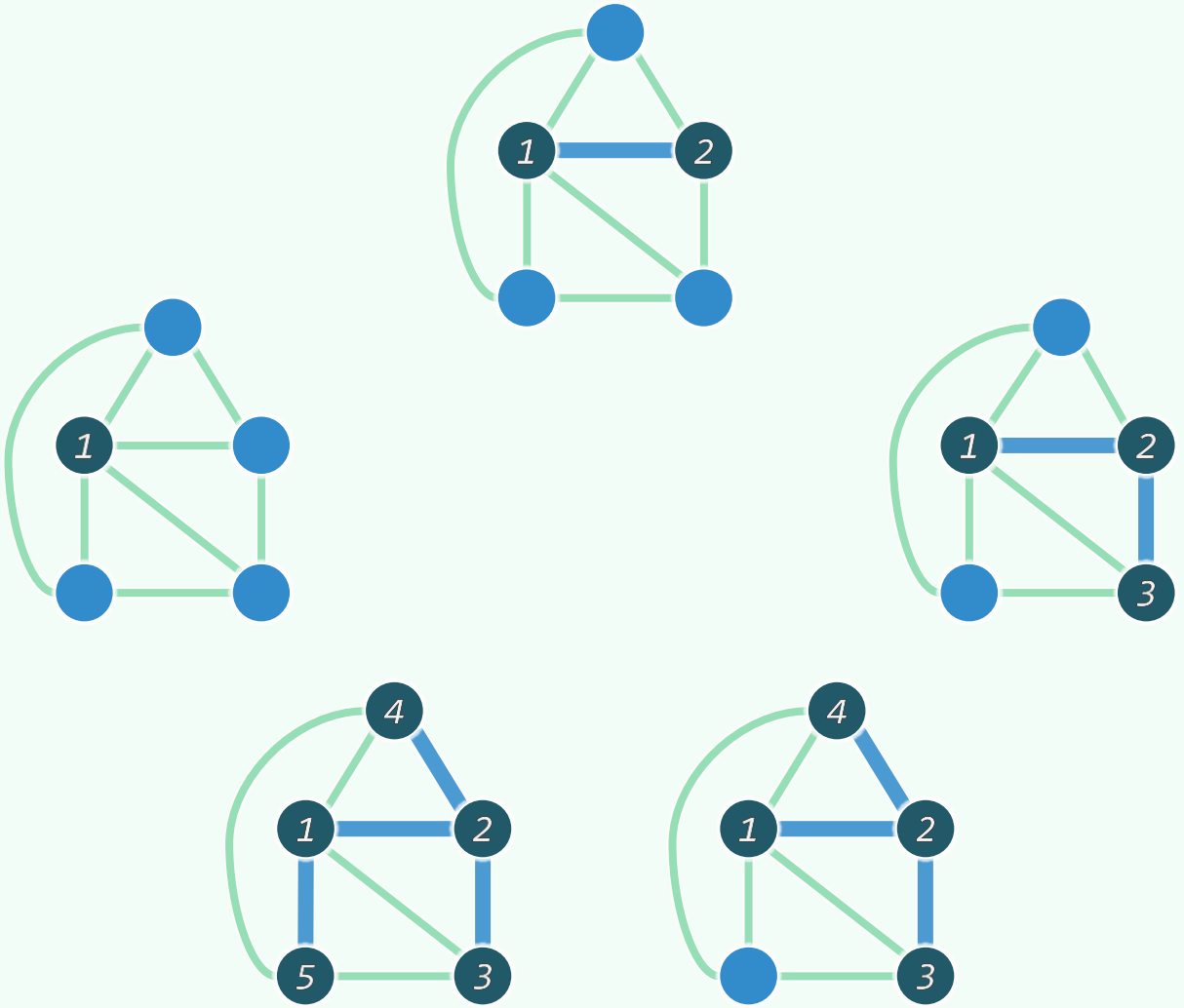
$$T_{k+1} = (V_{k+1}; E_{k+1})$$

$$= (V_k \cup \{\underline{v_{k+1}}\}; E_k \cup \{\underline{v_{k+1}u}\})$$

其中，$u \in V_k$

❖ 由此前的分析

- 只需将 $(V_k;\ V \backslash V_k)$ 视作原图的一个割
- 该割所有跨边中的极短者即是 $\underline{v_{k+1}u}$



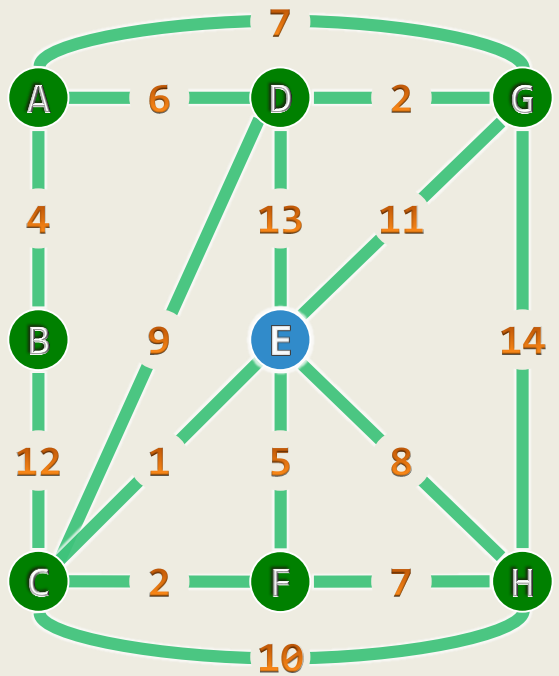
11-E3

图应用

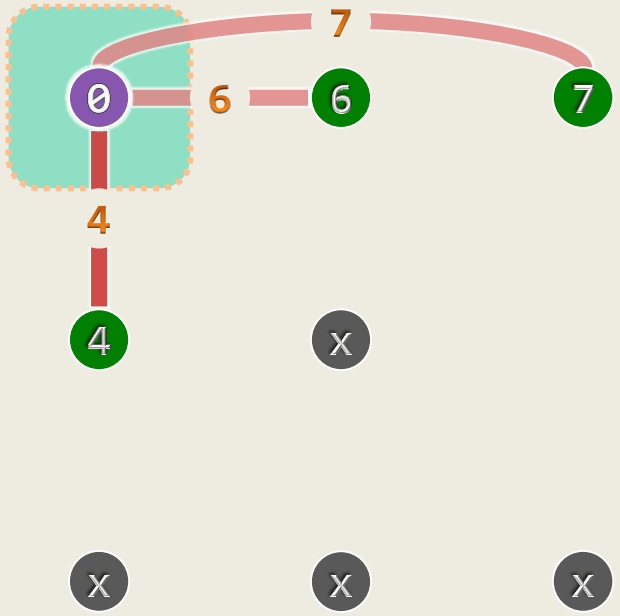
Prim算法：实例

邓俊辉

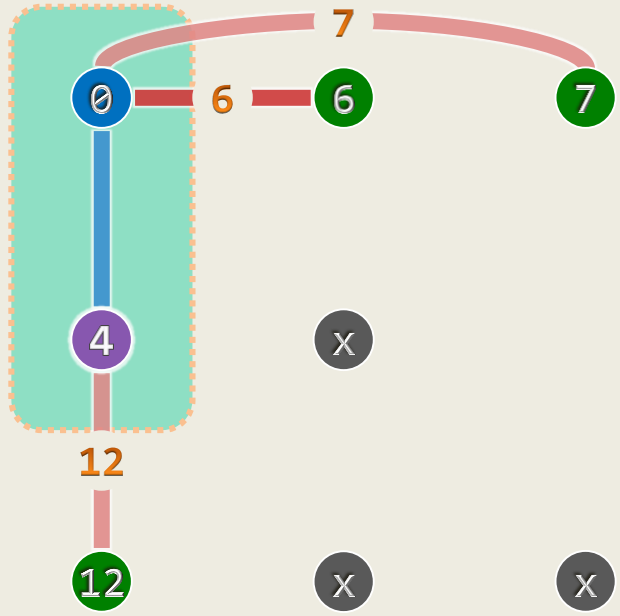
deng@tsinghua.edu.cn


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

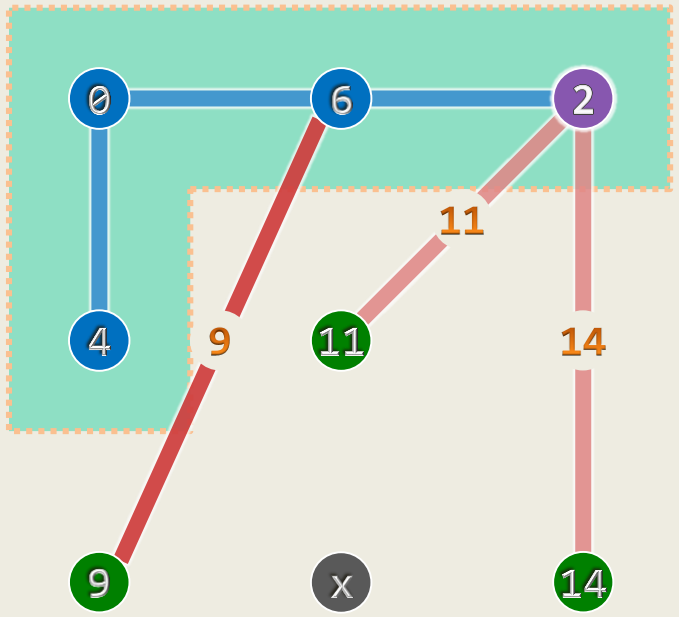
(a)


0
6
6
7
4
4
x
x
x
x

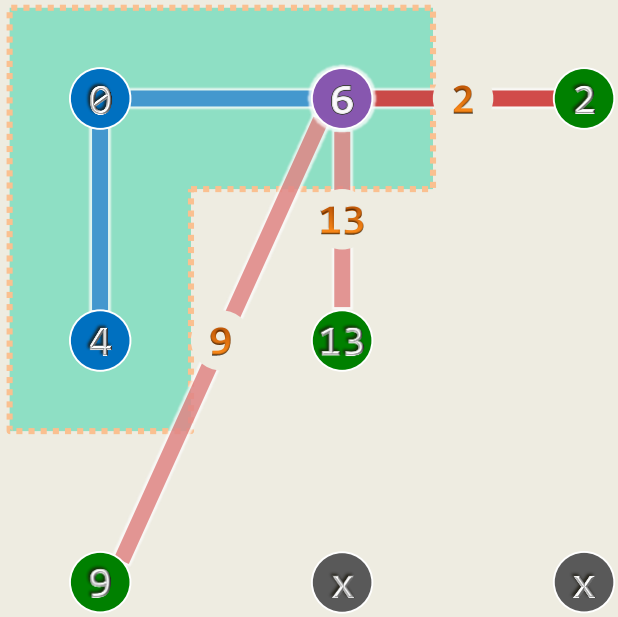
(b)


0
6
6
7
4
4
x
12
12
x
x

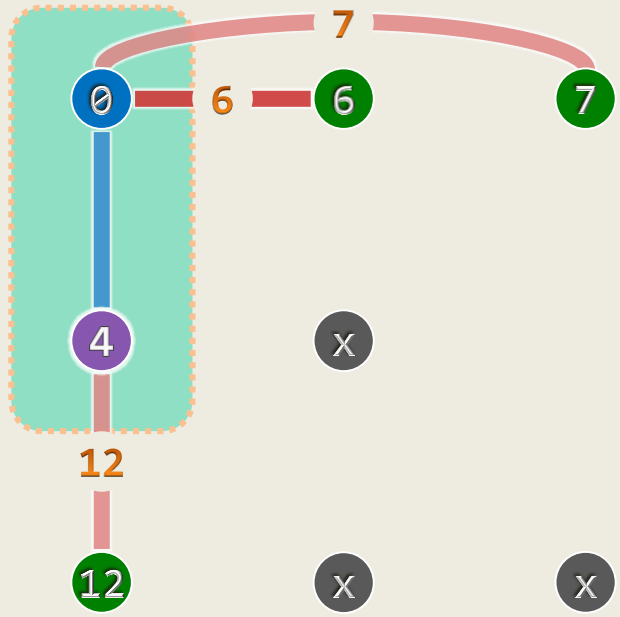
(c)



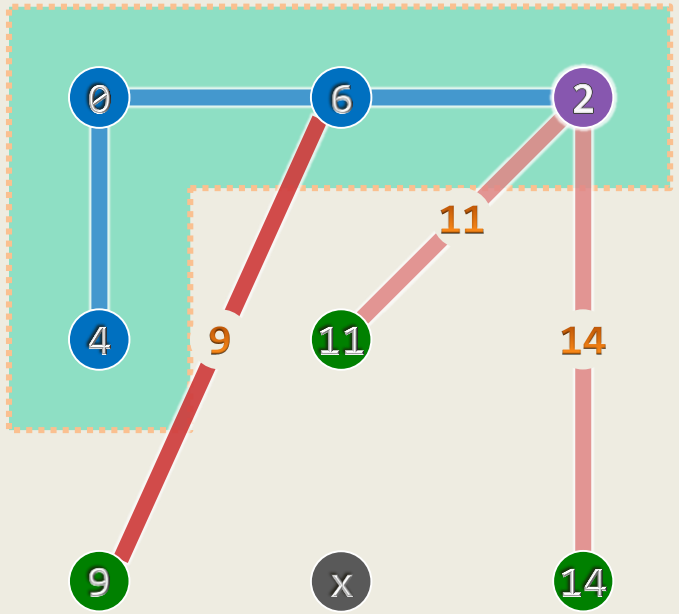
(e)



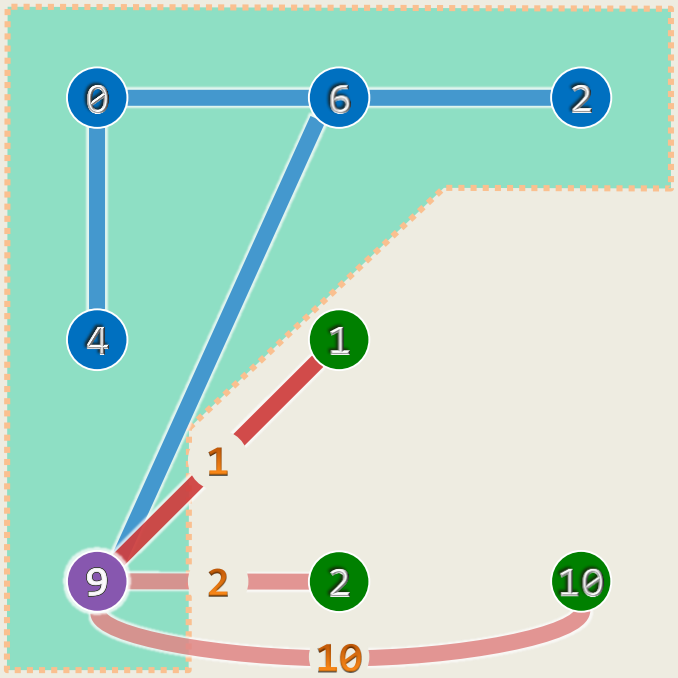
(d)



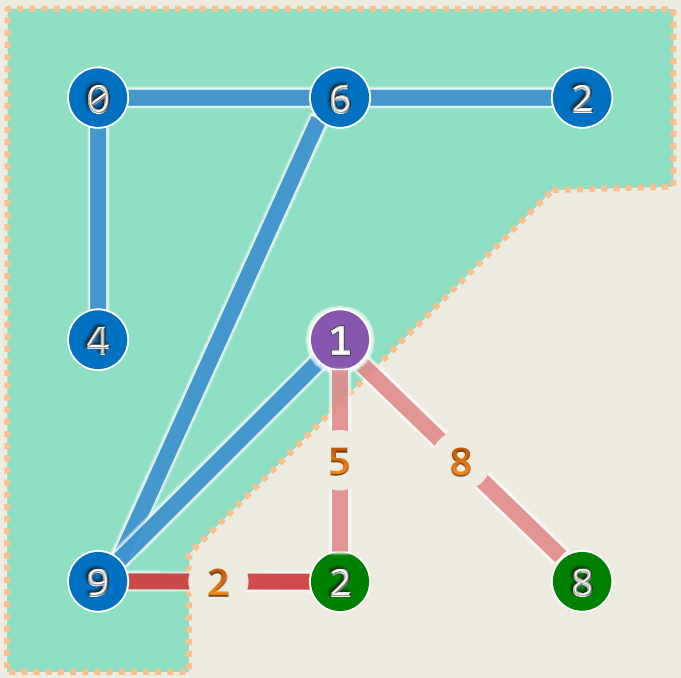
(c)



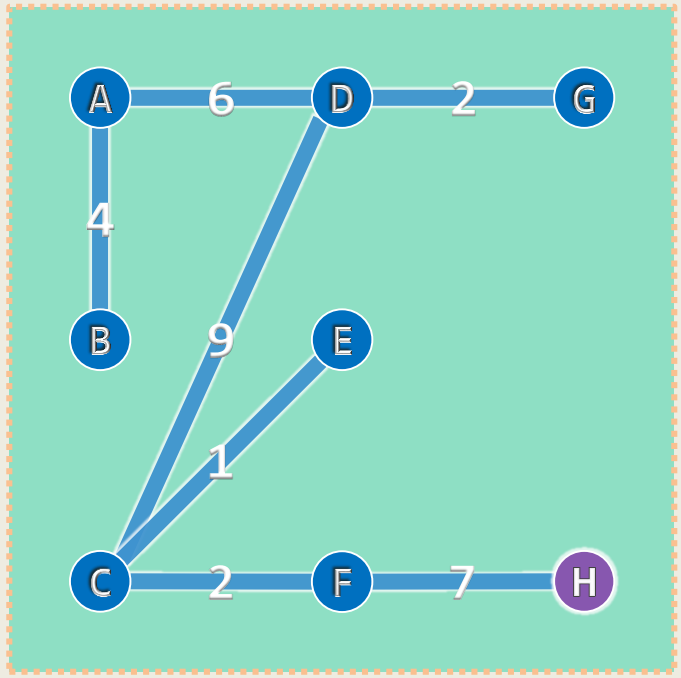
(e)



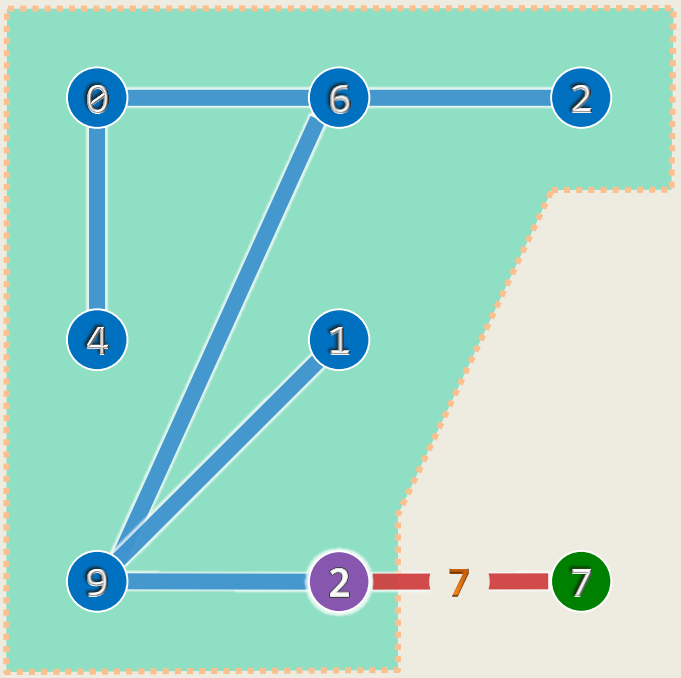
(f)



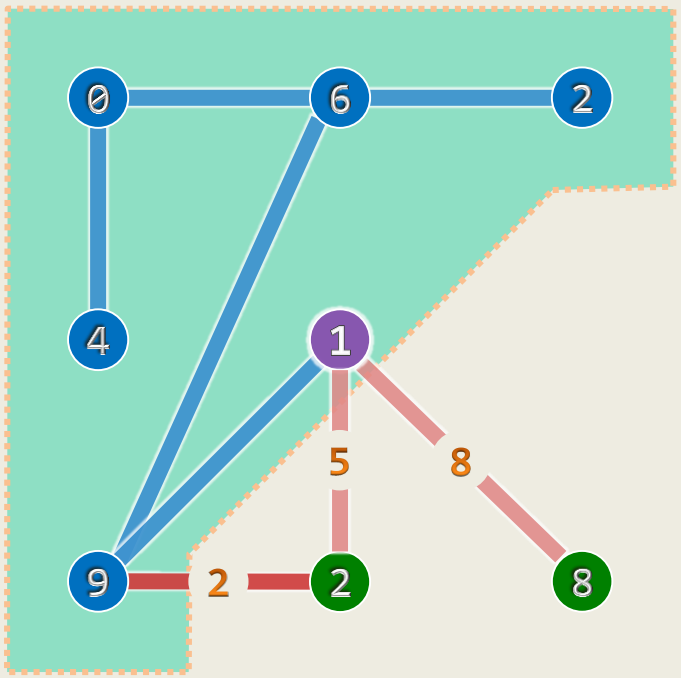
(g)



(i)



(h)



(g)

11-E4

图应用

Prim算法：正确性

邓俊辉
deng@tsinghua.edu.cn

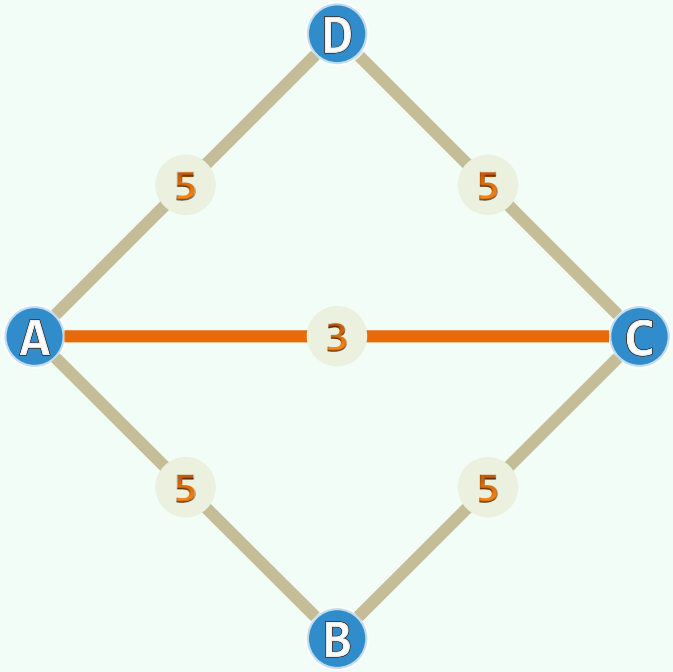
他唱着：“田野里的道路不止一条，”于是我们大家觉得甘美而恐怖

似是而非

- 设Prim算法依次选取了边{ e_2, e_3, ..., e_n }
- 其中每一条边e_k，的确都属于某棵MST
- 但在MST不唯一时...

  由此并不能确认，最终的T必是（一棵）MST
- 由极短跨边构成的支撑树，未必就是一棵MST

  反例...



可行的证明

- 在不增加总权重的前提下

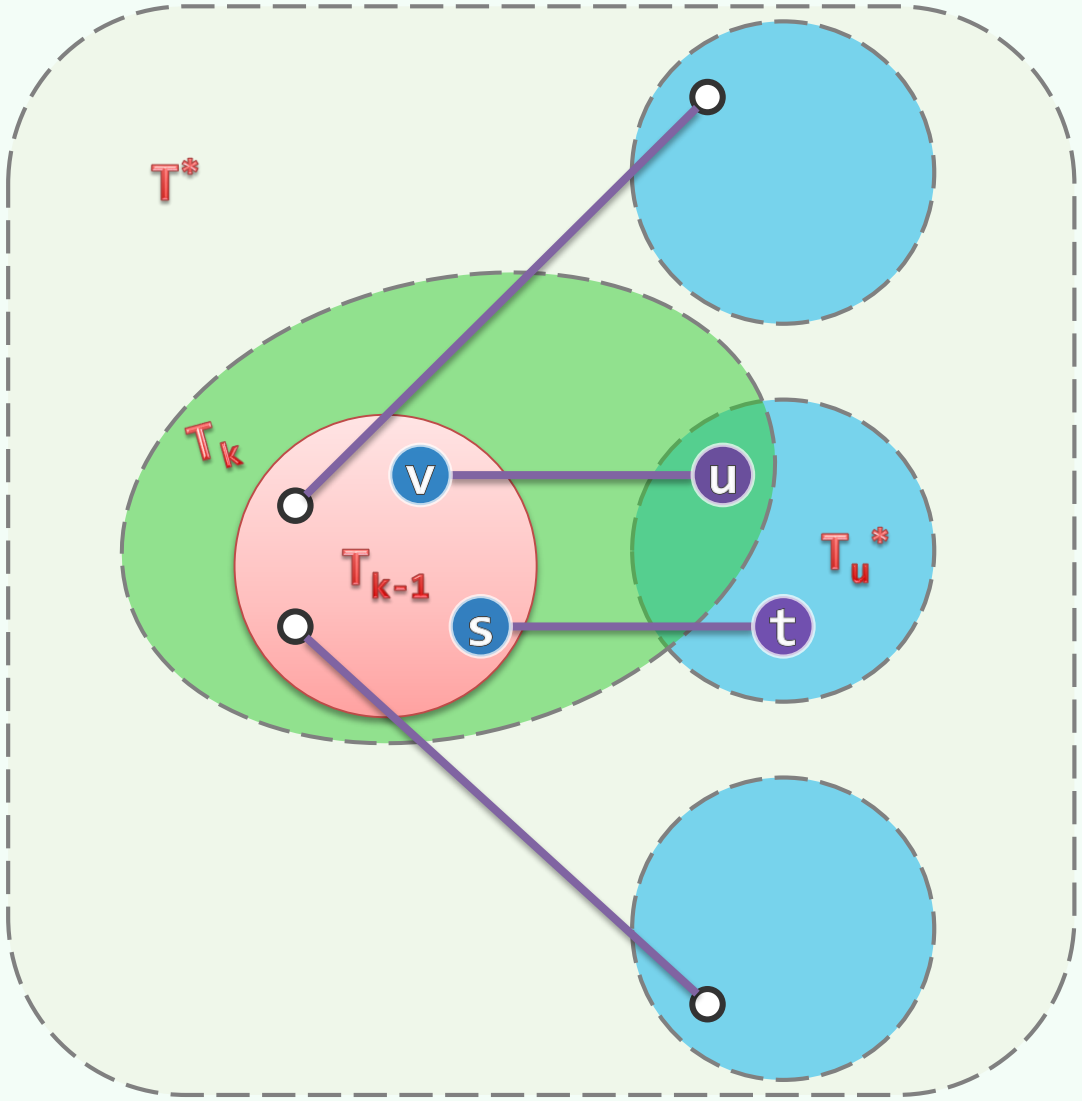
  可以将任一MST转换为T

  每一T_k都是某棵MST的子树，$1 \le k \le n$

- 《习题解析》，6-28题

  数学归纳

  ...



11-E5

图应用

Prim算法：实现

傍边一将，圆睁环眼，倒竖虎须，挺丈八蛇矛，飞马大叫：“三姓家奴休走！燕人张飞在此！”吕布见了，弃了公孙瓒，便战张飞

邓俊辉
deng@tsinghua.edu.cn

PFS

❖ $\forall v \notin V_k$, let $\text{priority}(v) = \|V_k, v\| \leq \infty$

❖ 于是套用PFS框架，为将 T_k 扩充至 T_{k+1}，只需

- 选出优先级最高的跨边 e_k 及其对应顶点 v_k，并将其加入 T_k
- 随后，更新 $V \backslash V_{k+1}$ 中所有顶点的优先级（数）

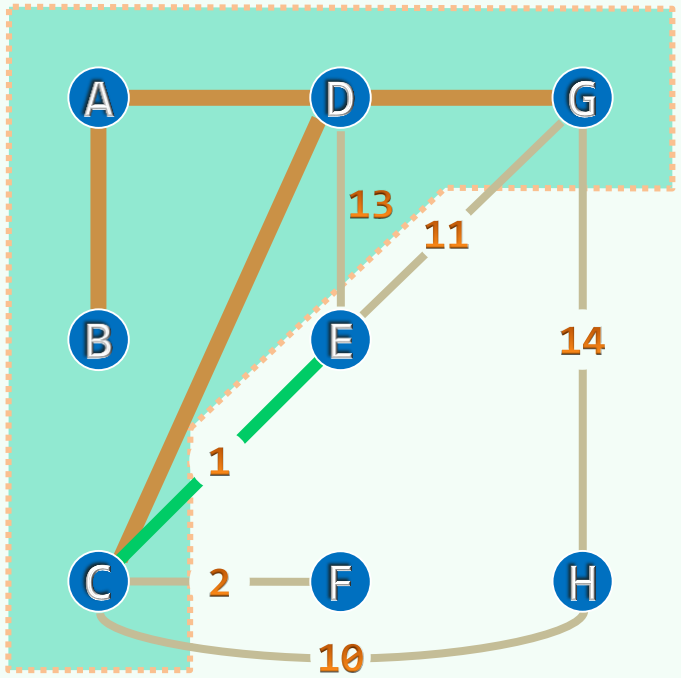
❖ 注意：优先级数随后可能改变（降低）的顶点，必与 v_k 邻接

❖ 因此，只需枚举 v_k 的每一邻接顶点 u，并取

$$\text{priority}(u) = \min(\text{priority}(u), \|v_k, u\|)$$

❖ 以上完全符合PFS的框架，唯一要做的工作无非是

按照prioUpdater()规范，编写一个优先级（数）更新器...



(f)

Priority Updater ~ PrimPU

```
g->pfs( 0, PrimPU<char, Rank>() ); //从顶点0出发，启动Prim算法

template <typename Tv, typename Te> struct PrimPU { //Prim算法的顶点优先级更新器
   virtual void operator()( Graph<Tv, Te>* g, Rank v, Rank u ) { //对v的每个
      if ( UNDISCOVERED != g->status(u) ) return; //尚未被发现的邻居u，按
      if ( g->priority(u) > g->weight(v, u) ) { //Prim
         g->priority(u) = g->weight(v, u); //策略
         g->parent(u)   = v; //做松弛
      }
   }
};
```

图应用

Kruskal算法：算法

11-F1

今天是过往的最后一天，也是未来的第一天

煮豆持作羹，漉菽以为汁；萁在釜下燃，豆在釜中泣；本是同根生，相煎何太急

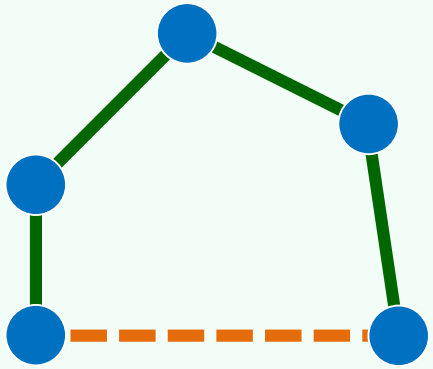
两个人在一起，人家就要造谣言；正如两根树枝接近，蜘蛛就要挂网

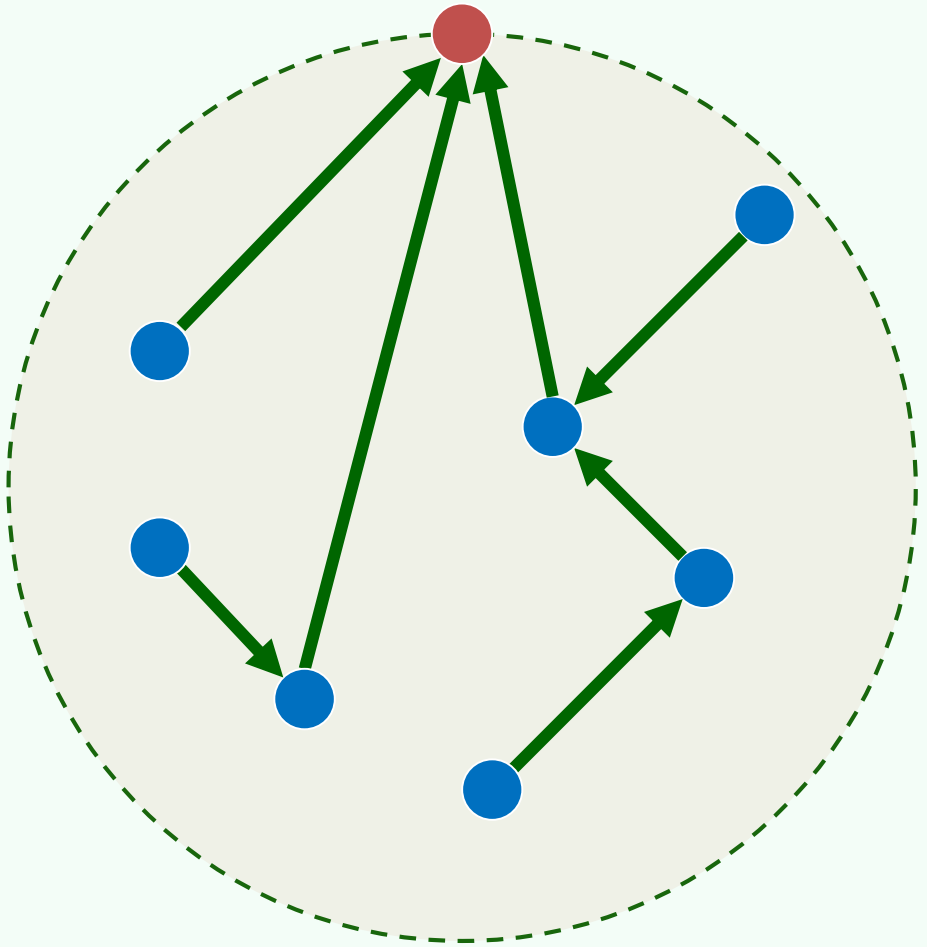
邓俊辉

deng@tsinghua.edu.cn

贪心策略

- ❖ 回顾Prim算法
  - 最短边，迟早会被采用
  - 次短边，亦是如此
  - 再次短者，则未必 //回路!
- ❖ Kruskal：贪心原则
  - 根据代价，从小到大依次尝试各边
  - 只要“安全”，就加入该边
- ❖ 但是，每步局部最优 = 全局最优?
- ❖ 确实，Kruskal很幸运...

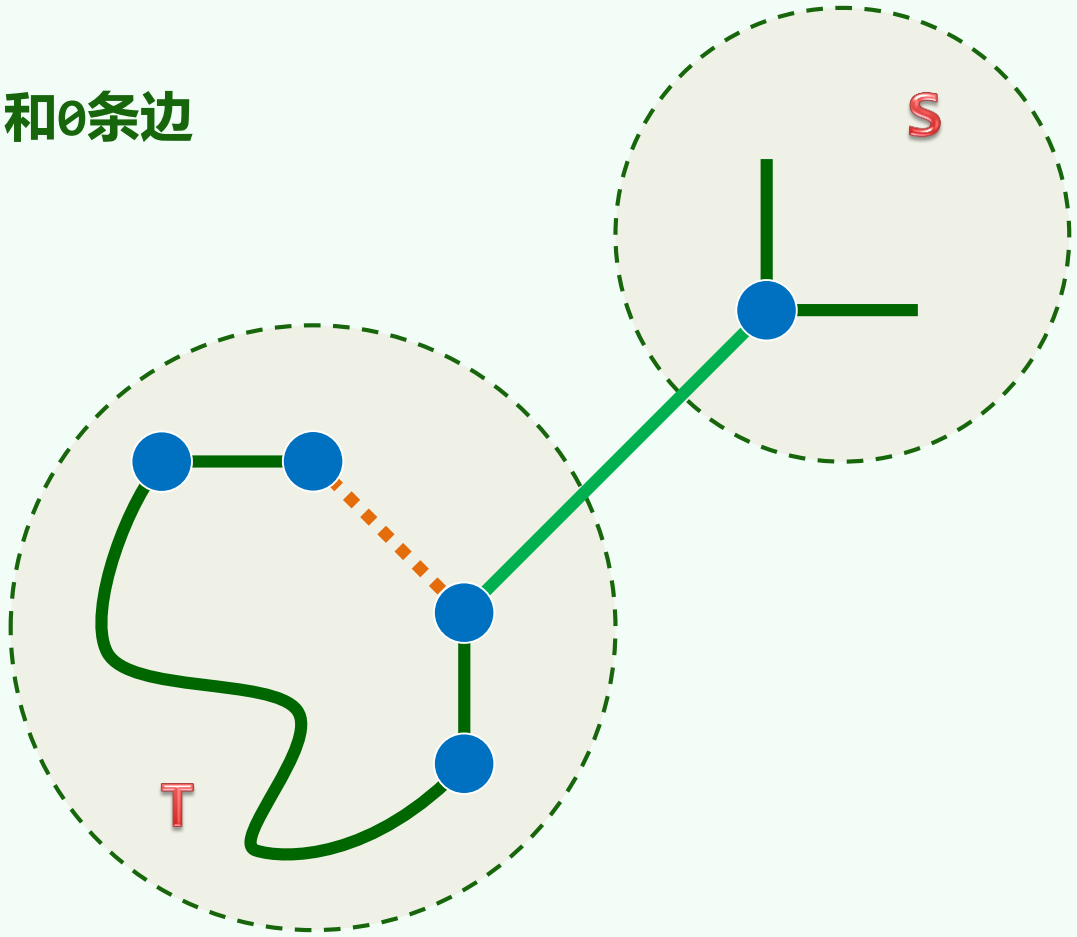




算法

- 维护N的一个森林：$R = (V; F) \subseteq N = (V; E)$
- 初始化：$R = (V; \varnothing)$包含 n 棵树（各含1个顶点）和0条边

  所有边按代价**非降序**排列
- 迭代，直到 R 成为一棵树
  - 找到当前**最廉价**的边 $e = (v, u)$
  - 若 v 和 u 来自 R 中**不同**的树，则
    - 令 $F = F \cup \{e\}$，然后
    - 合并由 e 联接的2棵树
- 整个过程共迭代 $n - 1$ 次，选出 $n - 1$ 条边



正确性：Kruskal引入的每条边都属于某棵MST

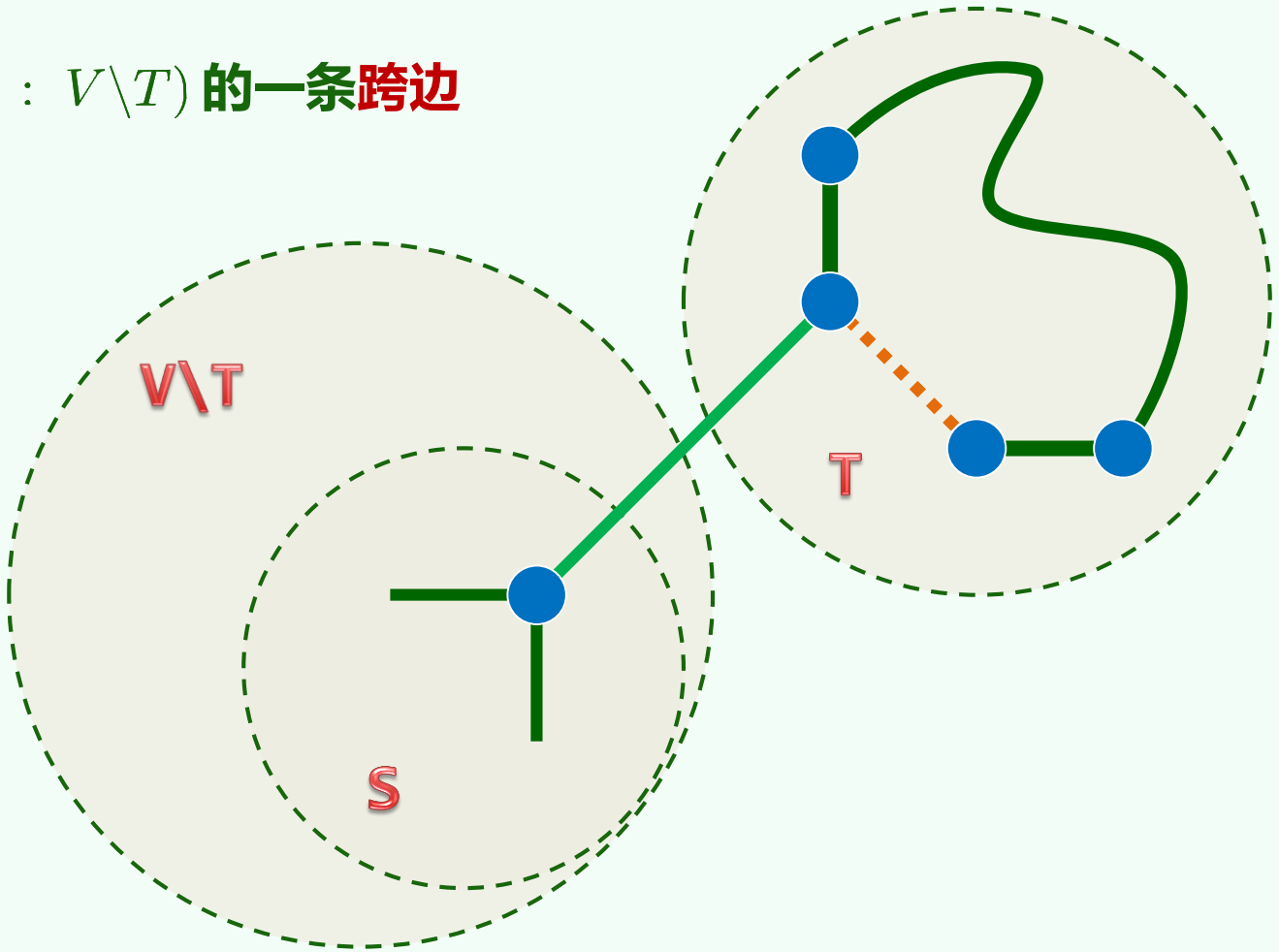
- 若 v 和 u 分属两棵树 S 和 T，则 e 是割 $(T\ :\ V\backslash T)$ 的一条跨边
- 实际上，该割所有不长于e的跨边

  均已被Kruskal考查过，并全部淘汰
- 故Kruskal应该采用 e
- 反之，若 v 和 u 同属一棵树

  则 e 的引入必然导致一个环路

  且沿此环路，e 为最长边，故应淘汰
- 与Prim同理，以上论述也不充分

  为严格起见，仍需归纳证明：Kruskal算法过程中不断生长的森林，总是某棵MST的子图



图应用

Kruskal算法：并查集

11-F2

Following the leader, the leader, the leader,
We're following the leader wherever he may go.

这里的人事关系是由一个个“单位”组成的.....白天里“单位”是魂，
人活在一个一个的单位里.....我很庆幸，我是个有单位的人

邓俊辉
deng@tsinghua.edu.cn

Union-Find

- 给定一组互不相交的等价类

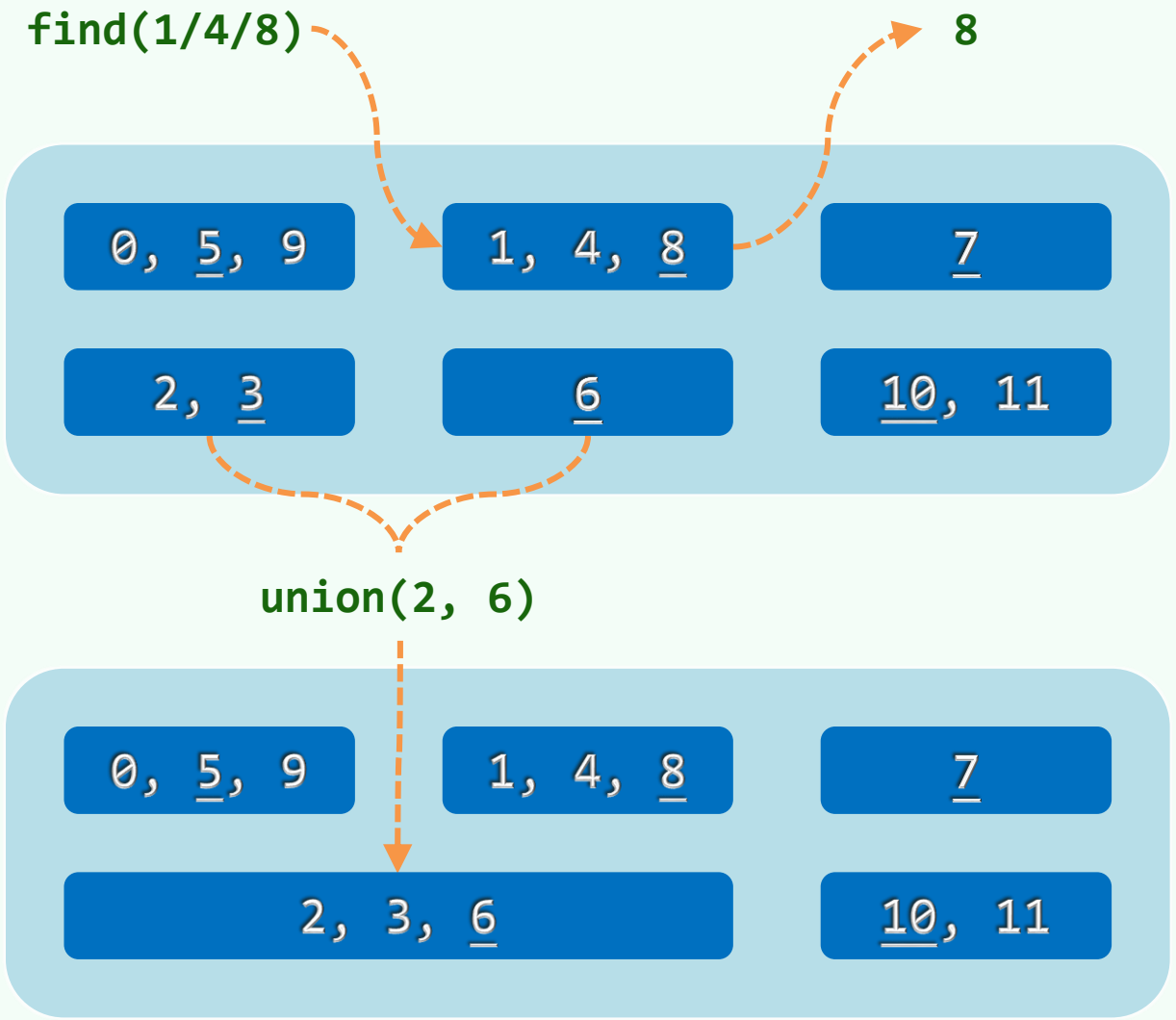
  各由其中一个成员作为代表

- Find(x)：找到元素x所属等价类

- Union(x, y)：合并x和y所属等价类

- Singleton：初始时各包含一个元素

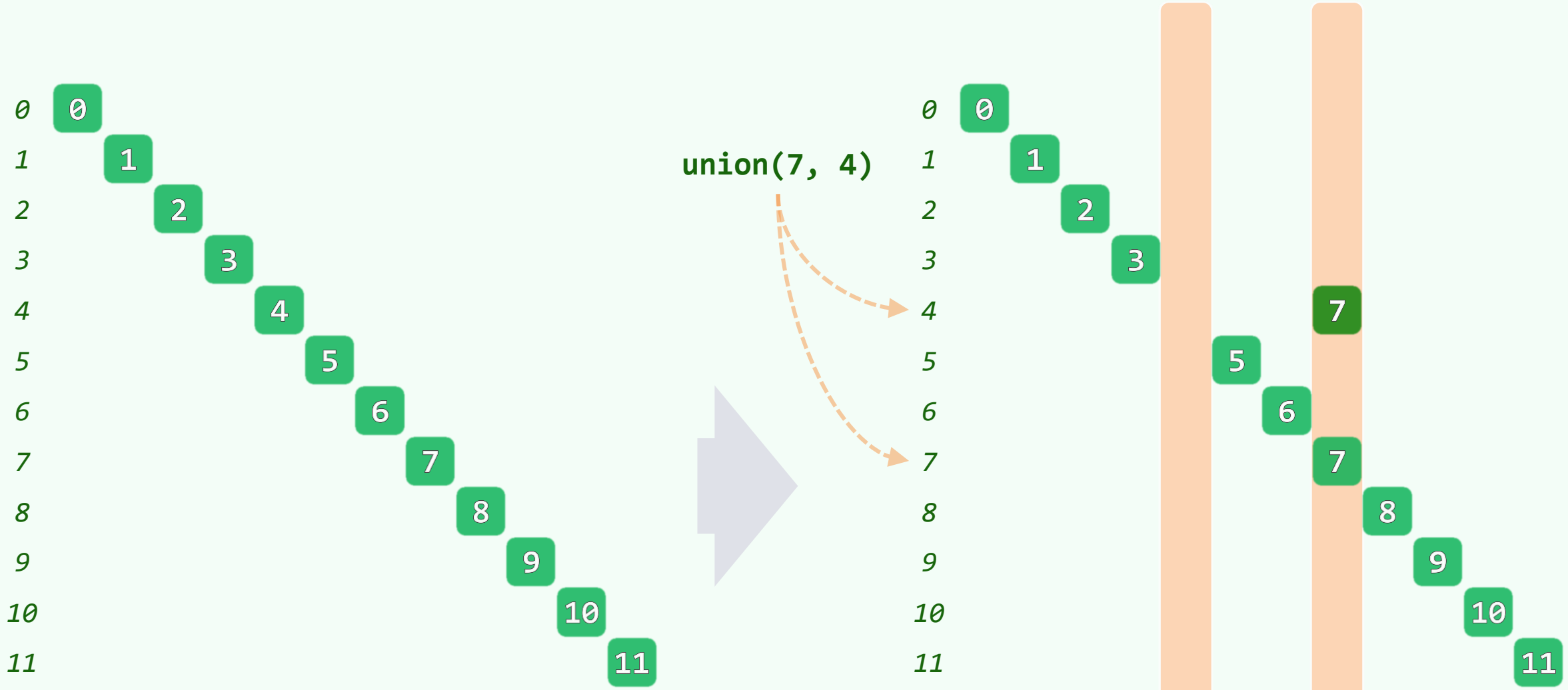
- Kruskal = Union-Find



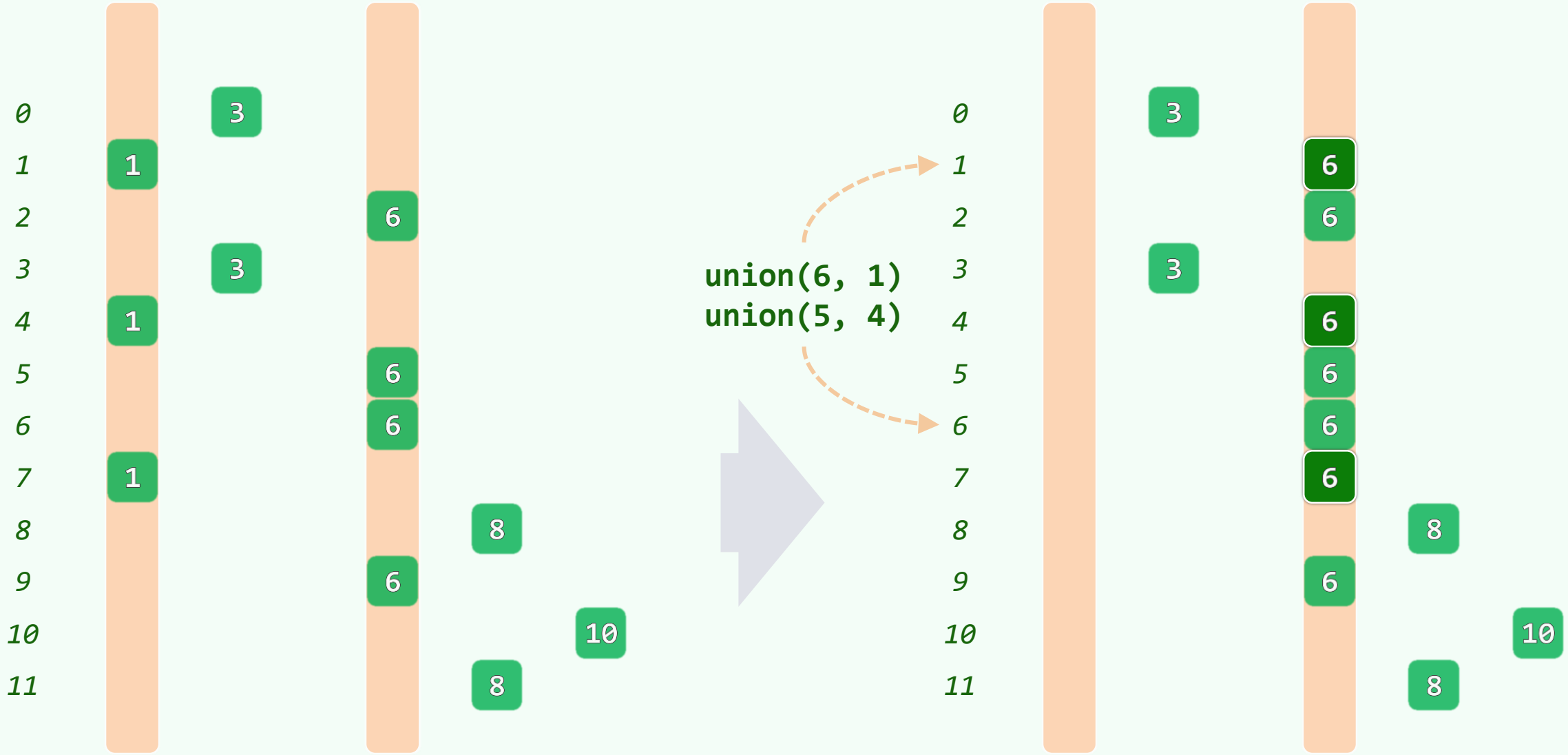
Quick-Find

```
class UnionFind:

    def __init__(self, n): #group[]记录各元素所属子集；初始各成一类，以[0,n)间整数标识
        self.g = self.n = n; self.group = [ k for k in range(n) ]

    def find(self, k):
        return self.group[k]

    def union(self, i, j):
        iGroup , jGroup = self.group[i] , self.group[j]
        if iGroup == jGroup: return
        for k in range(self.n):
            if (self.group[k] == jGroup): self.group[k] = iGroup
        self.g -= 1
```

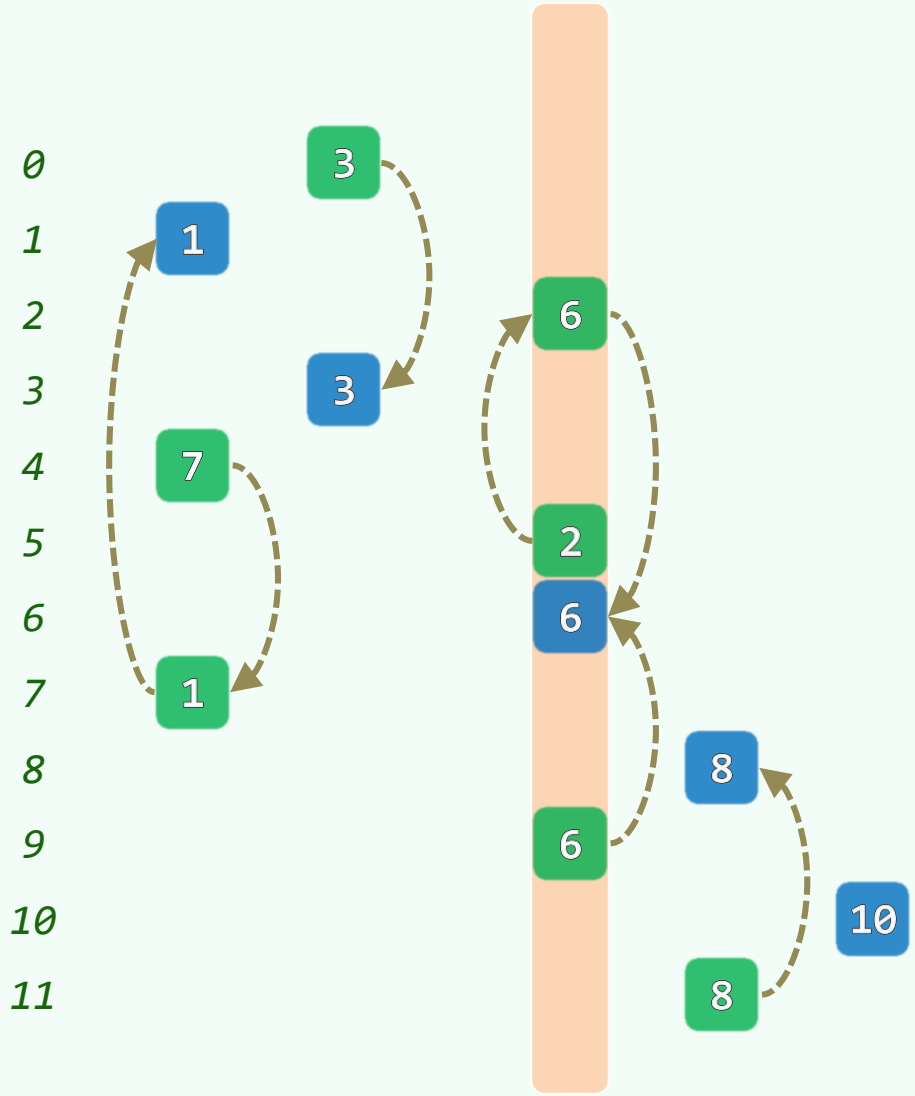
Slow-Union (1/2)



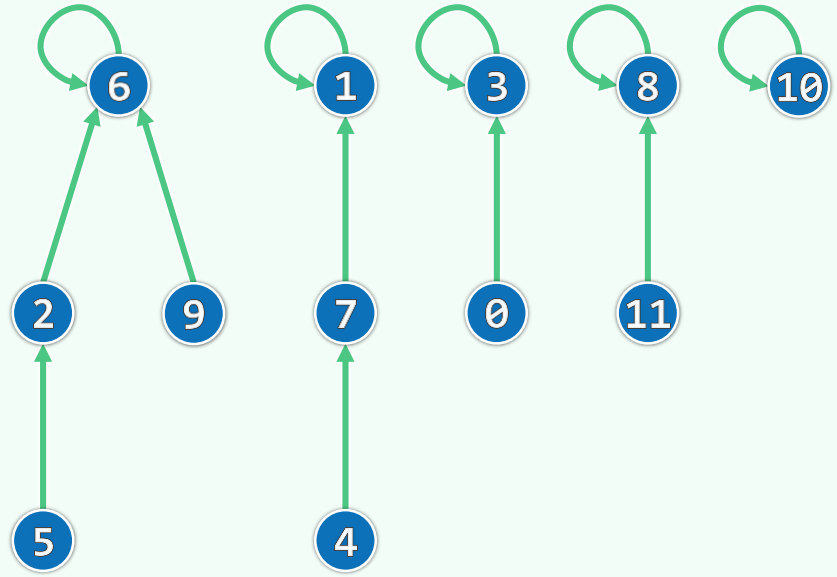
Slow-Union (2/2)



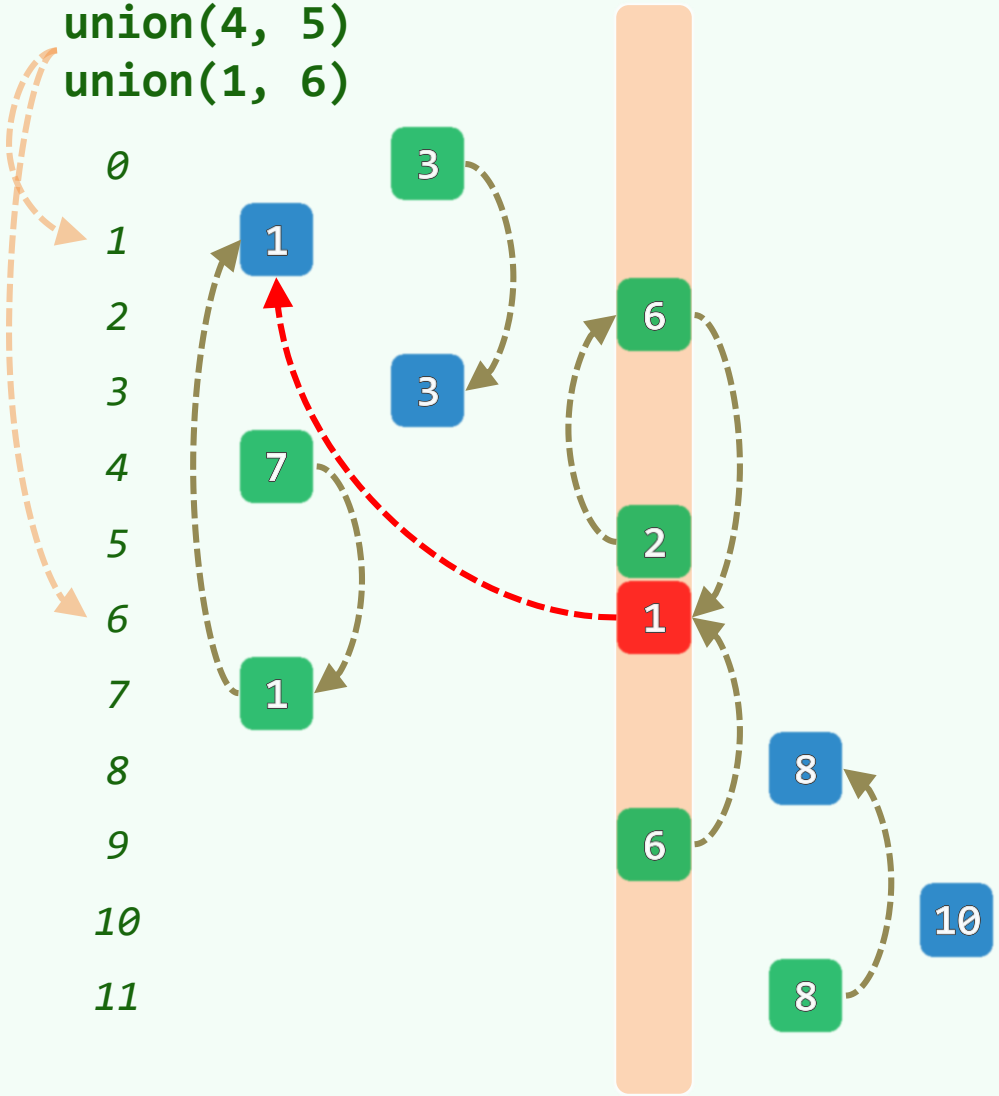
Group ~ Parent ~ Root



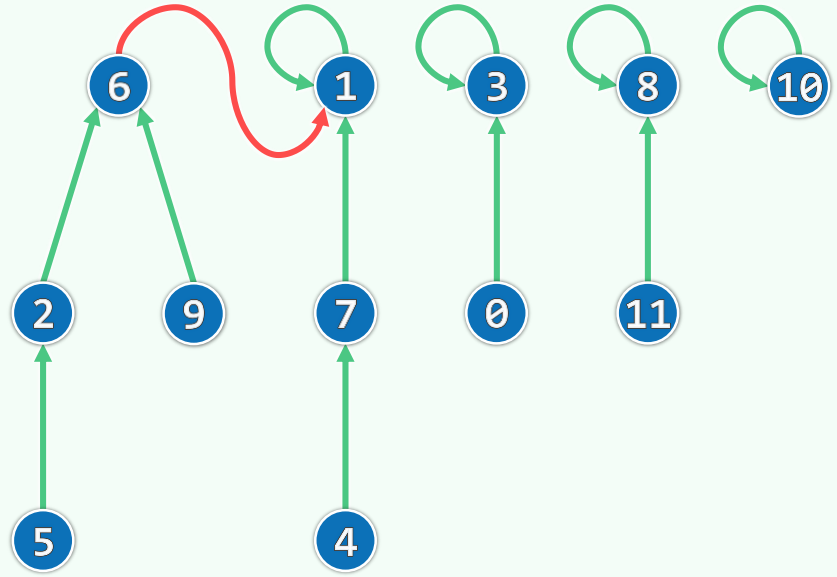
	rank	parent
0	3	3
1	1	1
2	6	6
3	3	3
4	7	7
5	2	2
6	6	6
7	1	1
8	8	8
9	6	6
10	10	10
11	8	8



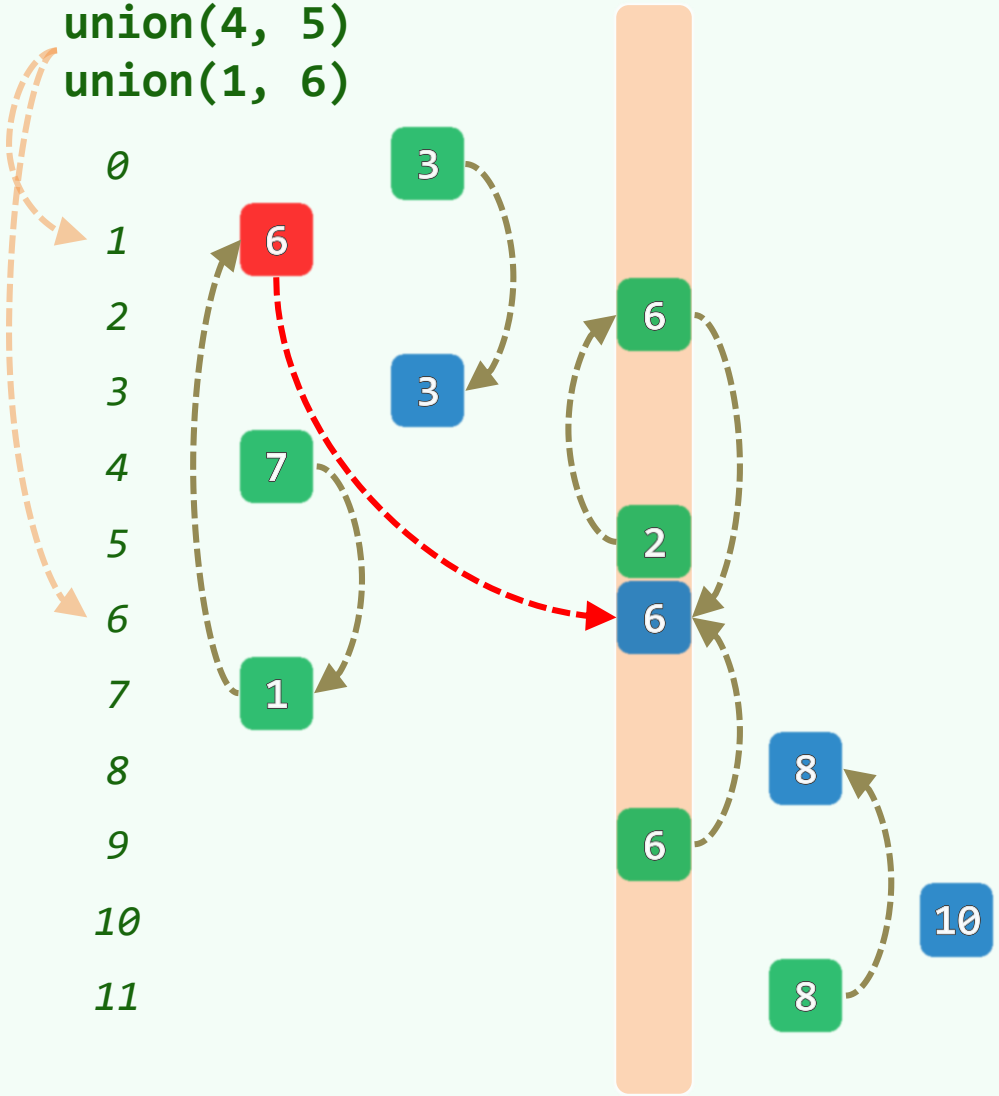
Quick-Union



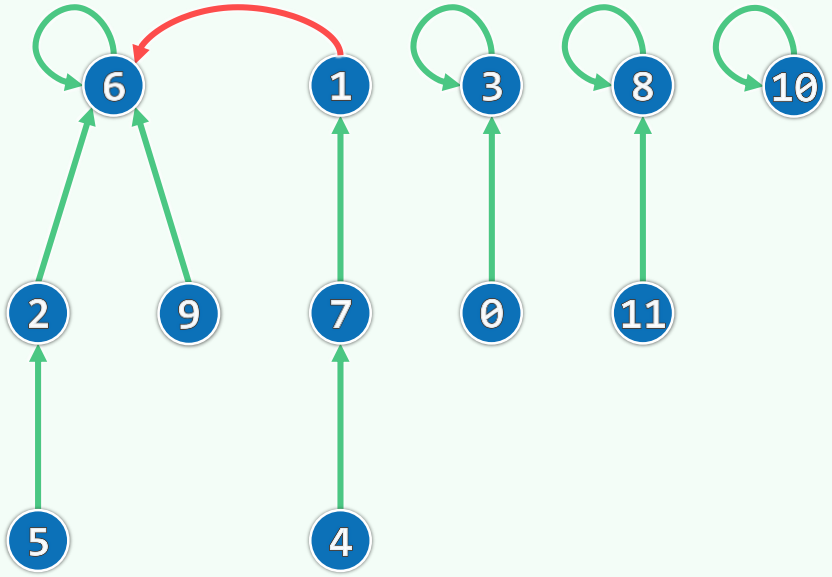
	rank	parent
0	0	3
1	1	1
2	2	6
3	3	3
4	4	7
5	5	2
6	6	1
7	7	1
8	8	8
9	9	6
10	10	10
11	11	8



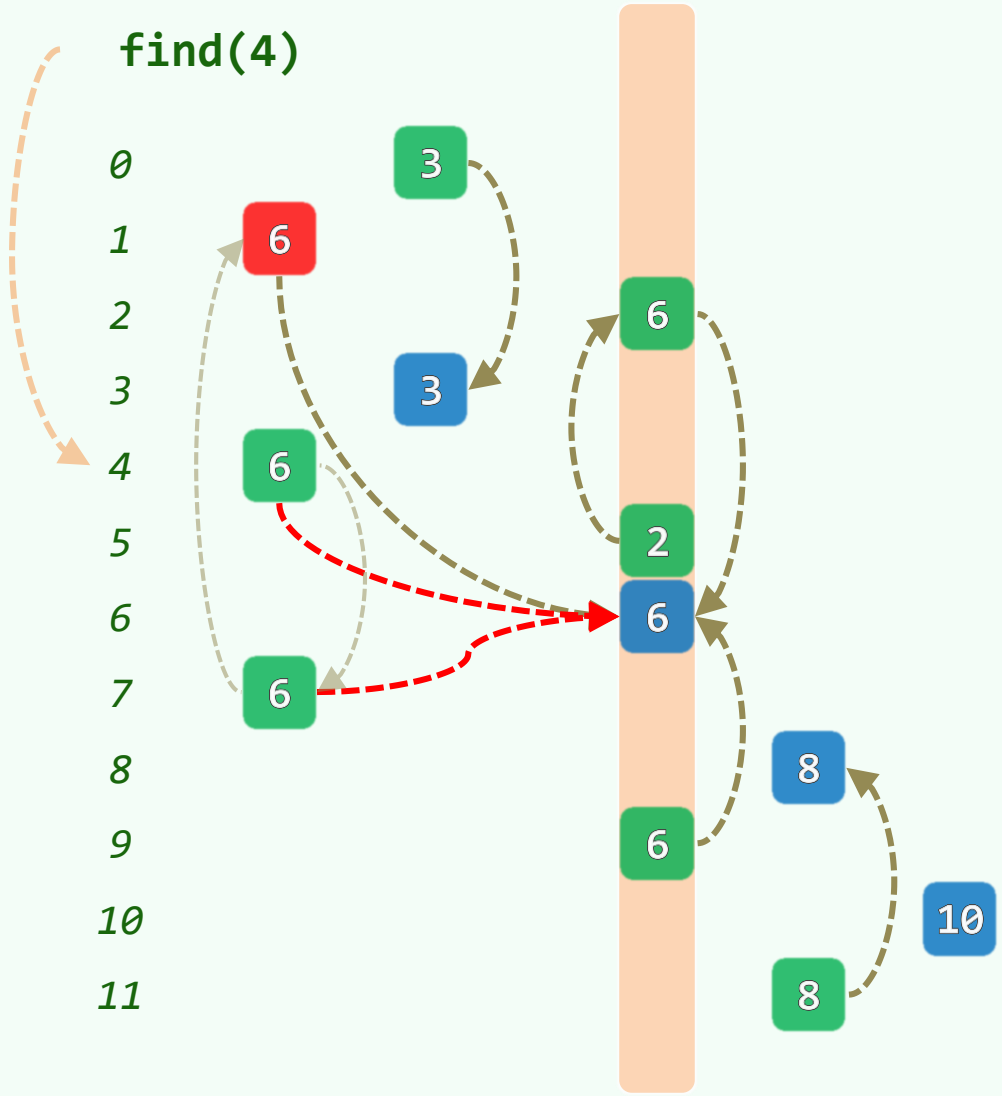
Weighting By Size Or Height



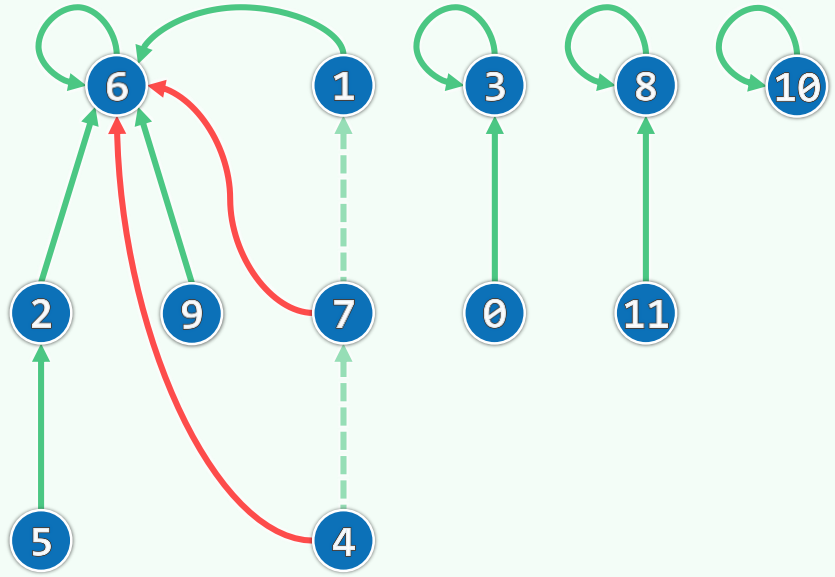
	rank	parent	size
0		3	
1		6	~~3~~
2		6	
3		3	2
4		7	
5		2	
6		6	~~4~~ 7
7		1	
8		8	2
9		6	
10		10	1
11		8	



Path Compression/Folding



rank	parent	size
0	3	
1	6	
2	6	
3	3	2
4	6	
5	2	
6	6	7
7	6	
8	8	2
9	6	
10	10	1
11	8	



图应用

Floyd-Warshall算法

11-G

邓俊辉

deng@tsinghua.edu.cn

让我们测量一下自己的活动半径，并待在那个中心吧，就像蜘蛛待在网的中心一样

从Dijkstra到Floyd-Warshall

- 给定带权网络G，计算其中**所有点对**之间的最短距离

- 应用：确定G的中心点（center） = 半径最小的顶点

  s的半径 = radius(G,s) = 所有顶点到s的最大距离

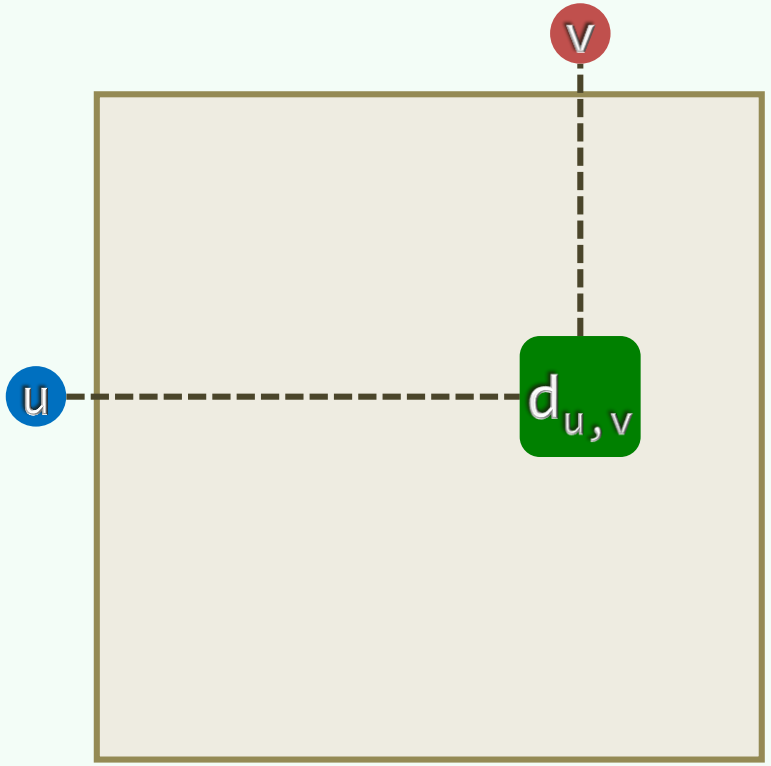
- 直觉：依次将各顶点作为源点，调用Dijkstra算法

  时间 $= n \times \mathcal{O}(n^2) = \mathcal{O}(n^3)$ —— 可否更快?

- 思路：图矩阵 --> 最短路径矩阵

- 效率：$\mathcal{O}(n^3)$，与执行n次Dijkstra相同 —— 既如此，F.W.之价值何在?

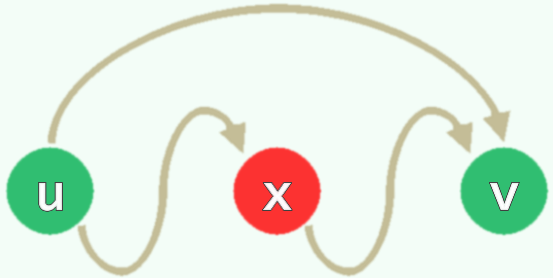
- 优点：形式简单、算法紧凑、便于实现；允许**负权边**（尽管仍不能有**负权环路**）

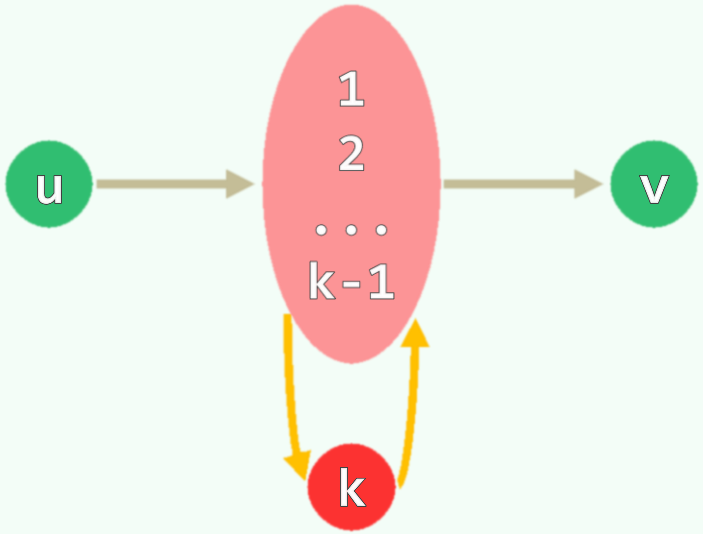


问题 + 特点

❖ u和v之间的最短路径可能是

- 不存在通路，或者
- 直接连接，或者
- 最短路径(u，x) + 最短路径(x，v)





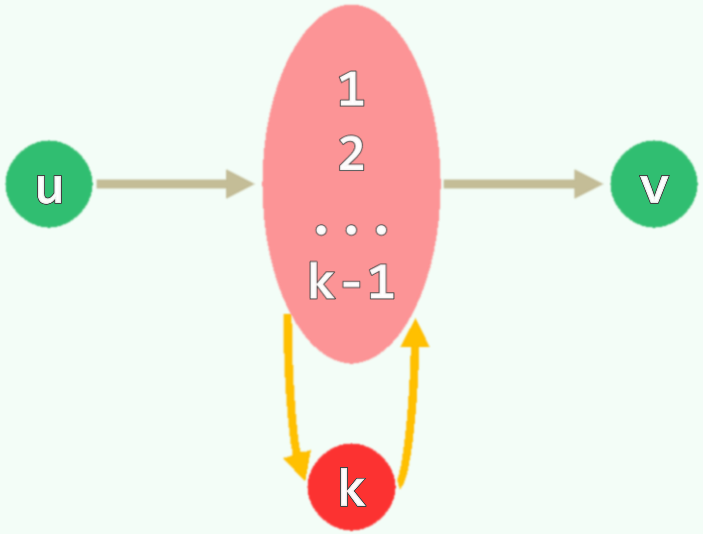
❖ 将所有顶点随意编号：1，2，...，n

定义：$d^k(u,v)$ = 中途只经过前k个顶点中转，从u通往v的最短路径长度

- $d^k(u,v) = w(u,v) \qquad (\text{if k} = 0)$
- $d^k(u,v) = \min\{\ d^{k-1}(u,v),\ d^{k-1}(u,k) + d^{k-1}(k,v)\ \} \qquad (\text{if k} \geq 1)$

蛮力递归

```
weight dist( node * u, node * v, int k )
   if ( k < 1 ) return w( u, v );
   u2v  =  dist( u, v, k-1 ); //经前k-1个点中转
   for each node x ∉ { u, v } //x作为第k个可中转点
      u2x2v = dist( u, x, k-1 )
            + dist( x, v, k-1 ); //递归
      u2v   = min( u2v, u2x2v ); //择优
   return u2v;
```



- 存在大量重复的递归调用，如何避免?
- 动态规划之记忆化：

  维护一张表，记录需要反复计算的数值

动态规划

```
for k in range(0, n)
  for u in range(0, n)
    for v in range(0, n)
      A[u][v] =
        min( A[u][v], A[u][k] + A[k][v] )
```