6.图

(xb) Kruskal算法

邓俊辉

deng@tsinghua.edu.cn

贪心策略

- 回顾Prim算法：

  代价$\boxed{最小}$的边，迟早会被采用

  $\boxed{次小}$的边，亦是如此

  $\boxed{再次小}$的，则未必 //回路！

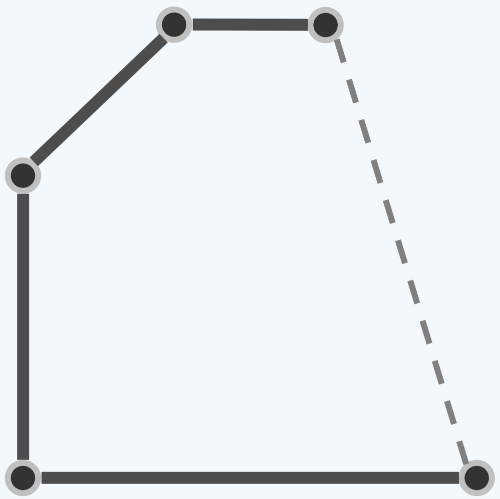
- Kruskal：贪心原则

  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...



算法框架

❖ 维护N的一个森林：$F = (V;\ E') \subseteq N = (V;\ E)$

❖ 初始化：$F = (V;\ \varnothing)$包含n棵树（各含 1 个顶点）和0条边

将所有边按照代价排序

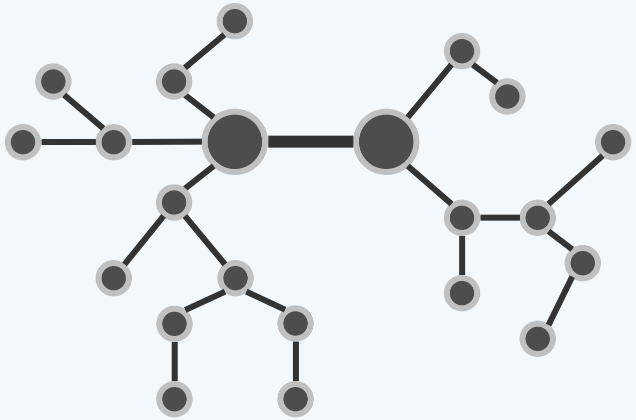
❖ 迭代：　找到当前最廉价的边e

若e的顶点来自F中不同的树，则

令$E' = E' \cup \{e\}$，然后

将e联接的2棵树合二为一

// 注意：引入e不致造成回路



❖ 重复上述过程，直到F成为1棵树

❖ 整个过程共迭代n-1次，选出n-1条边

正确性

❖ 定理：Kruskal引入的每条边都属于某棵MST

❖ 设边e = (u, v)的引入导致树T和S的合并

❖ 若：将(T; V\T)视作原网络N的割

则 ：e当属该割的一条跨边

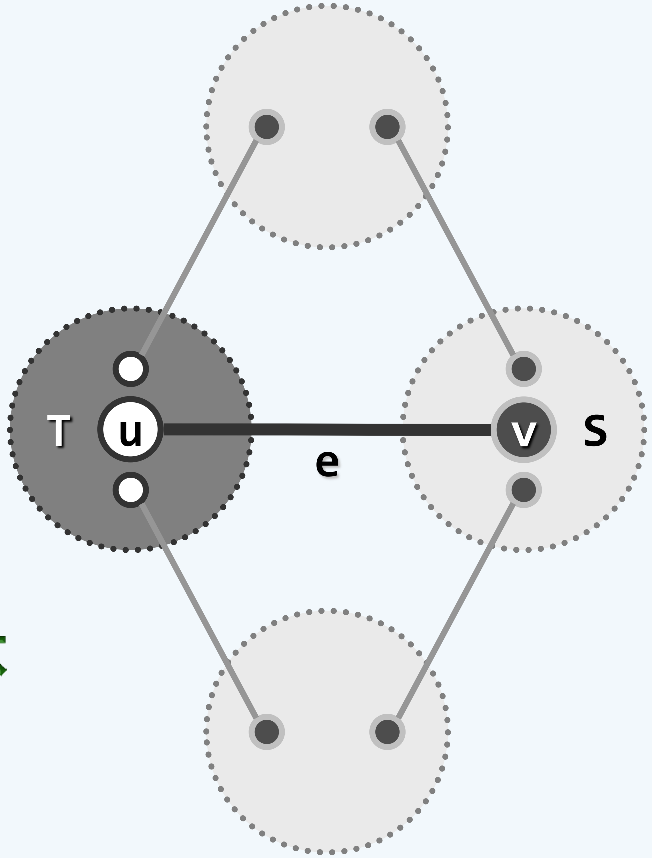
❖ 在确定应引入e之前

该割的所有跨边都经Kruskal考察，且只可能因不短于e而被淘汰

❖ 故：e当属该割的一条极短跨边

❖ 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



排序

- 需做全排序吗？

  若做，将耗时$\mathcal{O}(e\log e) = \mathcal{O}(n^2\log n)$ //稠密图可达上界

- 实际上，大多数情况下只需考虑前$\Theta(n)$条边

- 将所有边组织成优先队列 //比如，以堆实现

  建堆，$\mathcal{O}(e)$

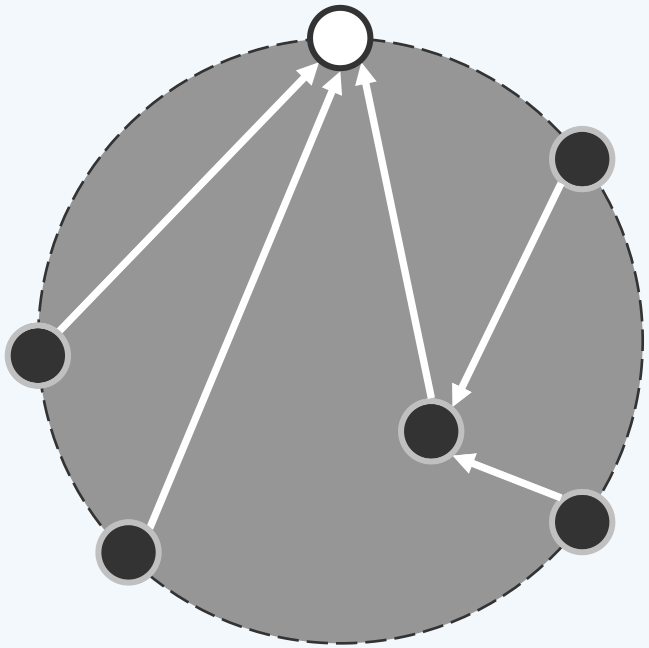
  删除并复原，$\mathcal{O}(\log e)$

  共迭代$\mathcal{O}(e)$次 //实际中往往远小于e，尤其是对于稠密图

  总共 $= \mathcal{O}(e) + \mathcal{O}(\log e) \times \mathcal{O}(e) = \mathcal{O}(e\log n)$

回路检测

- 如何$\boxed{\text{高效}}$地检测回路，并且合并树？



- 首领节点 $\boxed{\texttt{leader node}}$
  - 每棵树各选出一个首领
  - 各顶点设指针parent
  - 沿parent指针可找到对应的首领
  - leader.parent = NO_PARENT
- 尝试引入新边e = (u, v)时
  - 由parent指针，找到并比对 leader(u) 和 leader(v)
  - e的引入造成回路 iff leader(u) = leader(v)
- 如经检查确定可以合并，又该如何$\boxed{\text{高效}}$地合并u和v所属的树？

树合并

❖ 合并树T(u)和T(v)后，令leader(u).parent = leader(v)

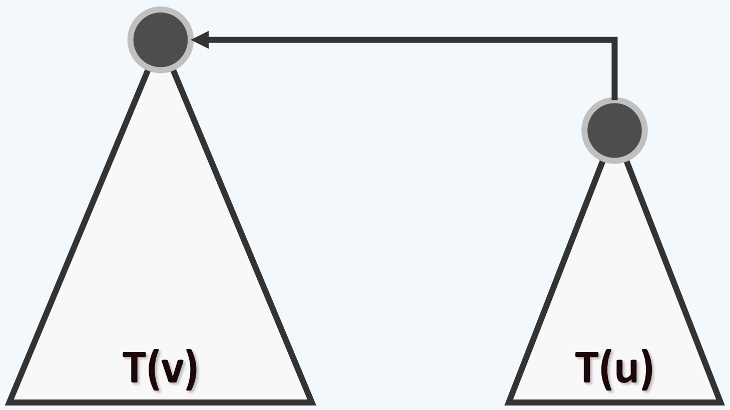
❖ 注意：合并后，树的深度 = $\mathcal{O}(n)$

总会不幸达到上界吗？很有可能！于是 ...

❖ 对leader的$\mathcal{O}(n)$次查询，共需$\mathcal{O}(n^2)$时间

❖ 改进：leader.parent = -(#nodes)

```
if  (leader(u).parent > leader(v).parent)
        leader(u).parent = leader(v);
else leader(v).parent = leader(u);
```



❖ 使用改进后算法，树合并后深度 = $\mathcal{O}(\log n)$ //YAN, p.142

❖ 因此，查询leader的总复杂度 = $\mathcal{O}(n\log n)$

Union-Find

❖ Union-Find问题

给定一组互不相交的等价类

由各自的一个成员作为代表

❖ Singleton

初始时各包含一个元素

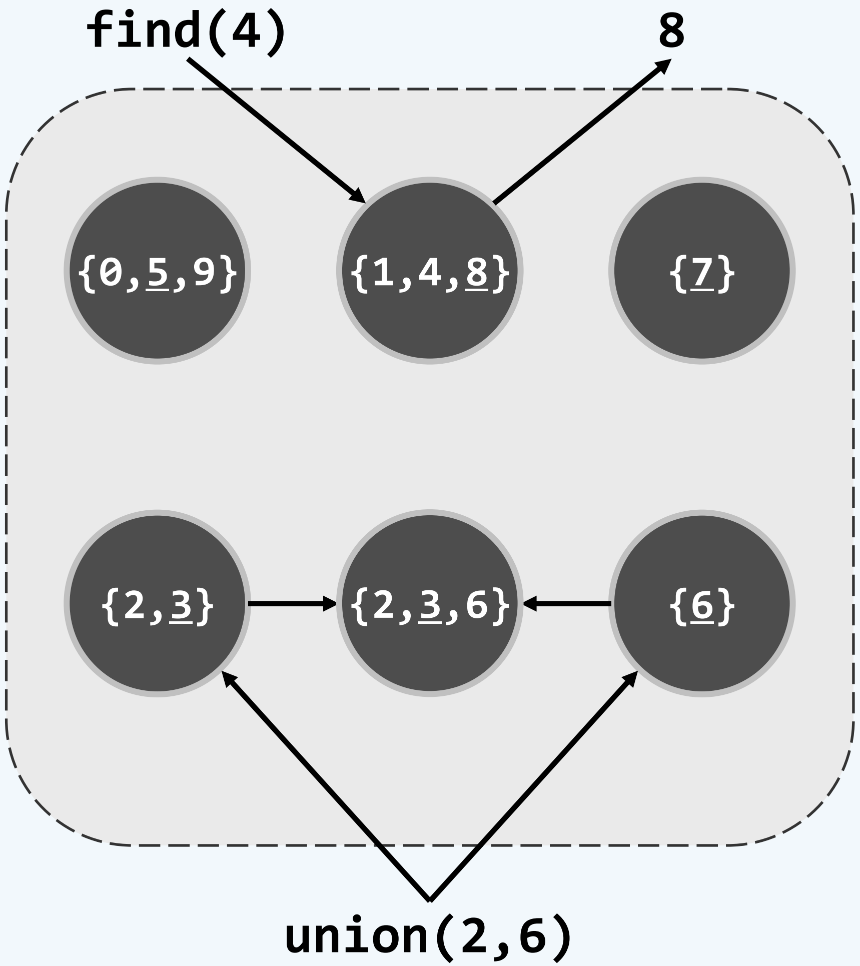
❖ Find(x)

找到元素x所属等价类

❖ Union(x, y)

合并x和y所属等价类

❖ Kruskal = Union-Find



Union-Find

❖ [Tarjan-83] : $\mathcal{O}(\alpha(n))$ amortized time per Union/Find

❖ $\alpha(n)$: inverse Ackermann function

$\log^* n$ = #logs s.t. $\log(\log(\dots(\log n)\dots)) < 2$

$\log^{**} n$ = #$\log^*$s s.t. $\log^*(\log^*(\dots(\log^* n)\dots)) < 2$

... ...

$\log^{**\cdots*} n$ = ...

... ...

❖ $\alpha(n)$ = #stars s.t. $\log^{**\cdots*} n < 3$

❖ α(目前可观察到宇宙范围内的粒子总数) < 4

复杂度

- ❖ 初始化森林，$\mathcal{O}(n)$
- ❖ 建立PQ，$\mathcal{O}(e)$
- ❖ 迭代共$\mathcal{O}(e)$次： 取出队首并调整PQ，$\mathcal{O}(\log e)$
  回路检测 + 边输出，$\mathcal{O}(\log n)$
  树合并，$\mathcal{O}(1)$
- ❖ 总共 = $\mathcal{O}(e\log e)$ = $\mathcal{O}(e\log n)$
- ❖ 对稀疏图，迭代次数 = e = $\Theta(n)$，共$\mathcal{O}(n\log n) \ll \mathcal{O}(n^2)$
  能更快吗？
- ❖ [Fredman & Tarjan-87]
  采用Fibonacci堆，对稀疏图可做到$\mathcal{O}(e\log\log n)$

其它算法

- BORUVKA（1920s）

  每个$\boxed{\text{点}}$与自己的$\boxed{\text{最近邻居}}$相连（构造出一个森林）

  每棵$\boxed{\text{树}}$与自己的$\boxed{\text{最近邻居}}$相连

  迭代上述过程，直到...

- VYSSOTSKY（1960s）

  每次增加一条边

  如果出现回路，将回路上$\boxed{\text{最长}}$的边删去

更新结果

❖ Gabow, Galil, Spencer, & Tarjan：$\mathcal{O}(m \times \log(\beta(m,n)))$

Efficient algorithms for finding minimum spanning trees

in undirected and directed graphs

Combinatorica, vol. 6, 1986, pp. 109–122

$\beta(m,n)$ = smallest i s.t. $\log(\log(\log(\ldots\log(n)\ldots))) < m/n$

where the logs are nested i times

❖ Fredman & Willard：权值均为不大的整数时，最坏情况仅需线性时间

Trans-dichotomous algorithms for

minimum spanning trees and shortest paths

31st IEEE Symp. Foundations of Comp. Sci., 1990, pp.719–725

更新结果

- YAO (1995): $\mathcal{O}(e\times\log\log n)$
- Karger, Klein, & Tarjan：随机算法，期望的线性时间

  A randomized linear-time algorithm

  to find minimum spanning trees

  J. ACM, vol. 42, 1995, pp.321-328
- CHAZELLE (2000): MST与union-find问题的复杂度相同

相关话题

- Euclidean MST

  Delaunay Triangulation

  Gabriel Graph

  Relative Neighborhood Graph

  $\Omega(n\log n)$

- Steiner MST

  NP-hard

  Approximation