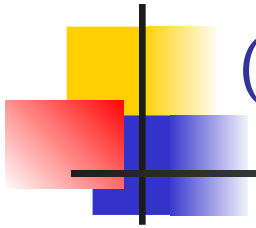
Dynamic Trees Problem, and its applications

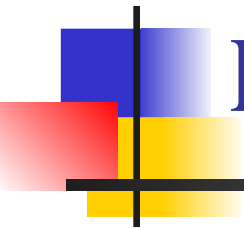
湖南省长郡中学 袁昕颢

xinhaoyuan[at]gmail[dot]com

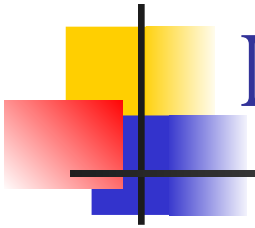


Overview

1. 动态树问题
   - 给出动态树问题的基本形式．
2. 解决动态树问题
   - 提出新的 Rake & Compress 方法．
3. 动态树问题的应用
   - 用最大流算法来说明动态树问题的应用．



Part I. Dynamic Trees Problem

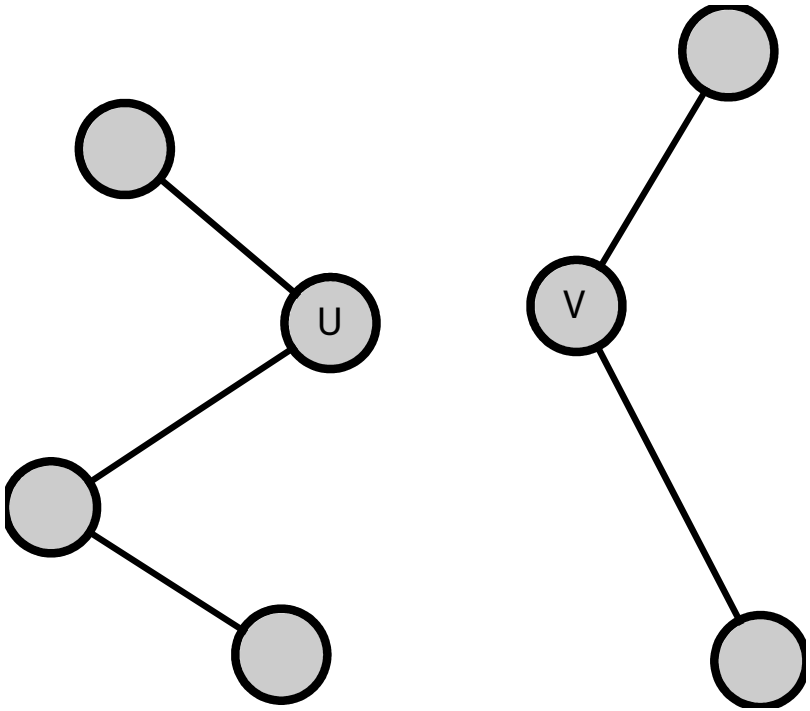


Dynamic Trees Problem

- 动态树问题 (Dynamic Trees Problem) 是图论中一类非常重要的经典问题． 许多图论算法， 尤其是在线动态算法都将其作为瓶颈问题．
- 研究和解决该问题具有很高的理论价值和实际价值．
- 什么是动态树问题呢？

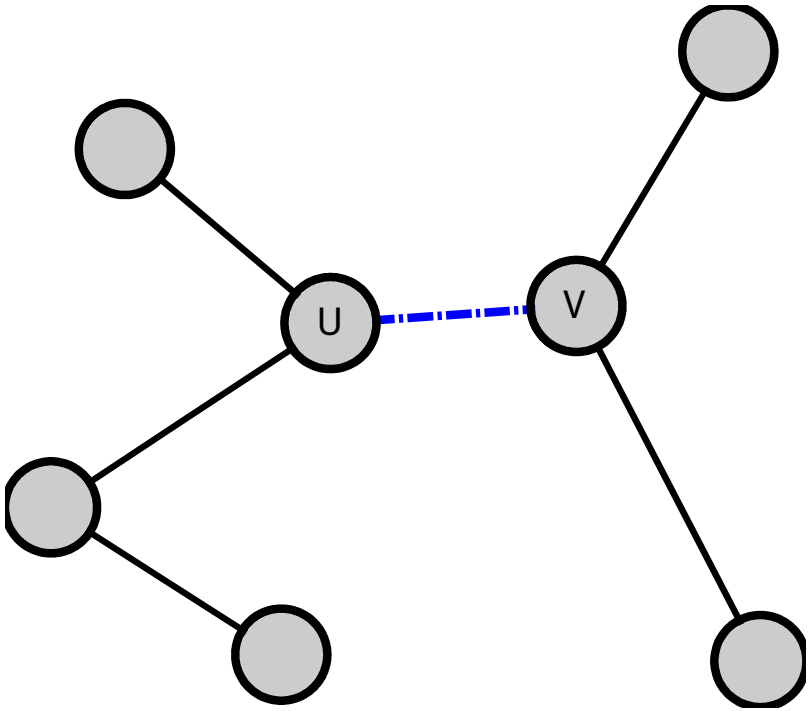
Dynamic Trees Problem

- 维护一个包含 N 个点的森林， 并且支持形态和权值信息的操作 .
- 形态信息



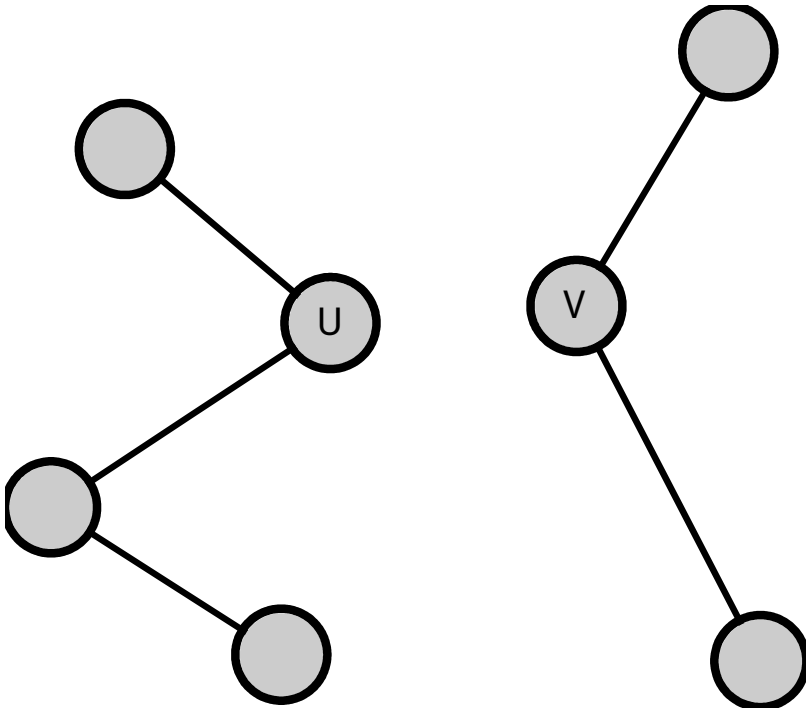
Dynamic Trees Problem

- 维护一个包含 N 个点的森林， 并且支持形态和权值信息的操作 .
- 形态信息
  - Link(u, v) — 添加边 (u, v)



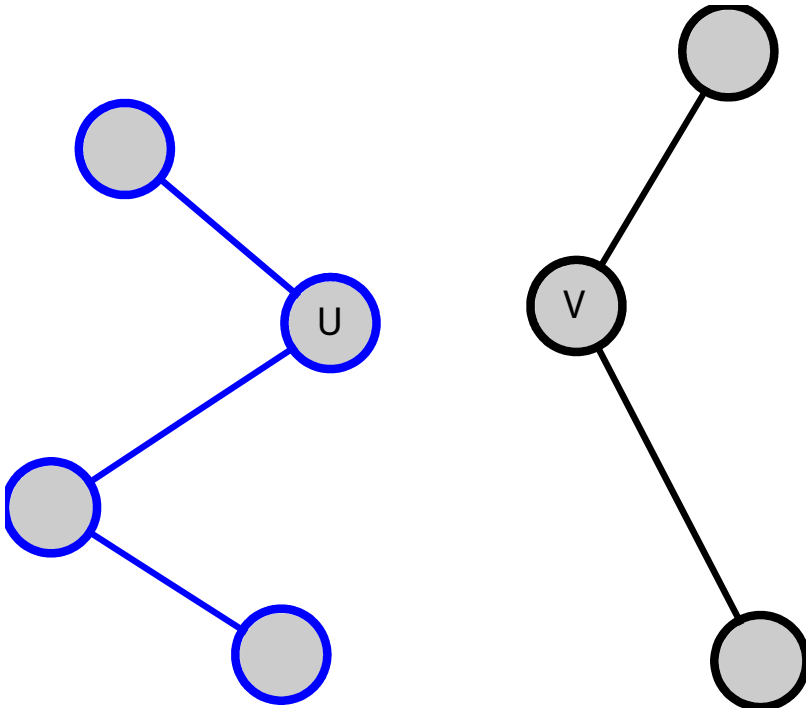
Dynamic Trees Problem

- 维护一个包含 N 个点的森林， 并且支持形态和权值信息的操作 .
- 形态信息
  - Link(u, v) — 添加边 (u, v)
  - Cut(u, v) — 删除边 (u, v)



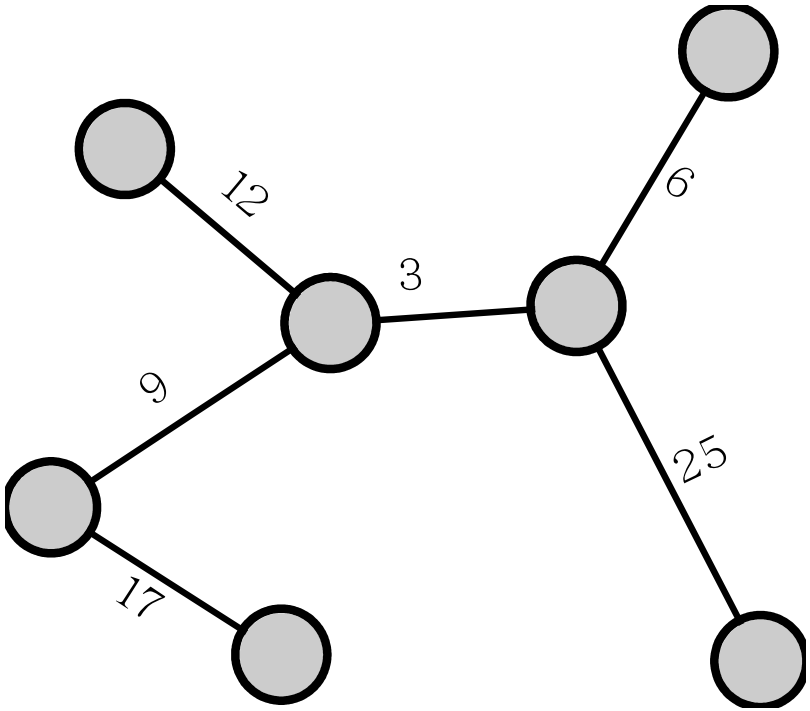
Dynamic Trees Problem

- 维护一个包含 N 个点的森林， 并且支持形态和权值信息的操作 .
- 形态信息
  - Link(u, v) — 添加边 (u, v)
  - Cut(u, v) — 删除边 (u, v)
  - Find(u) — 找到 u 所在的树 .



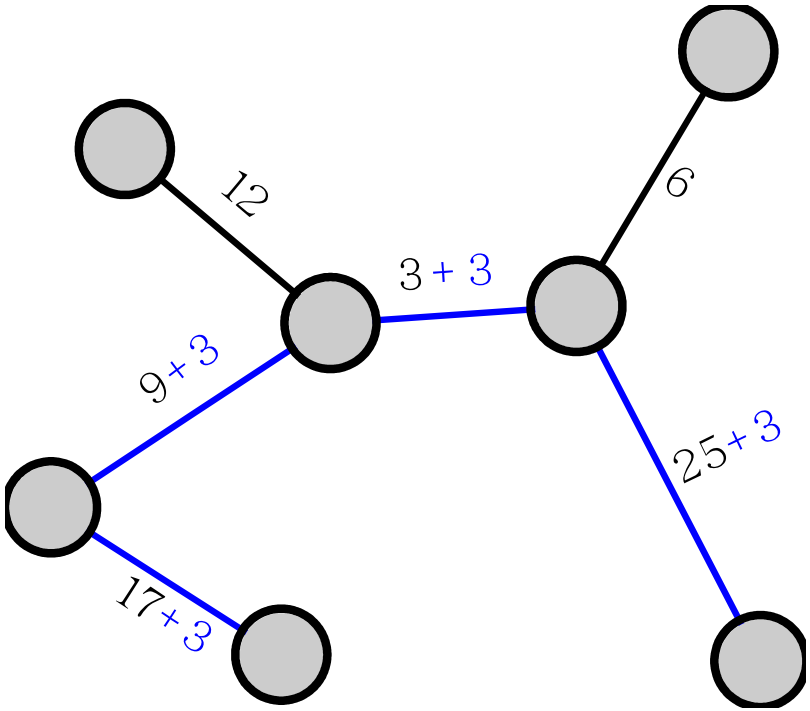
Dynamic Trees Problem

- 维护一个包含 N 个点的森林， 并且支持形态和权值信息的操作 .
- 权值信息



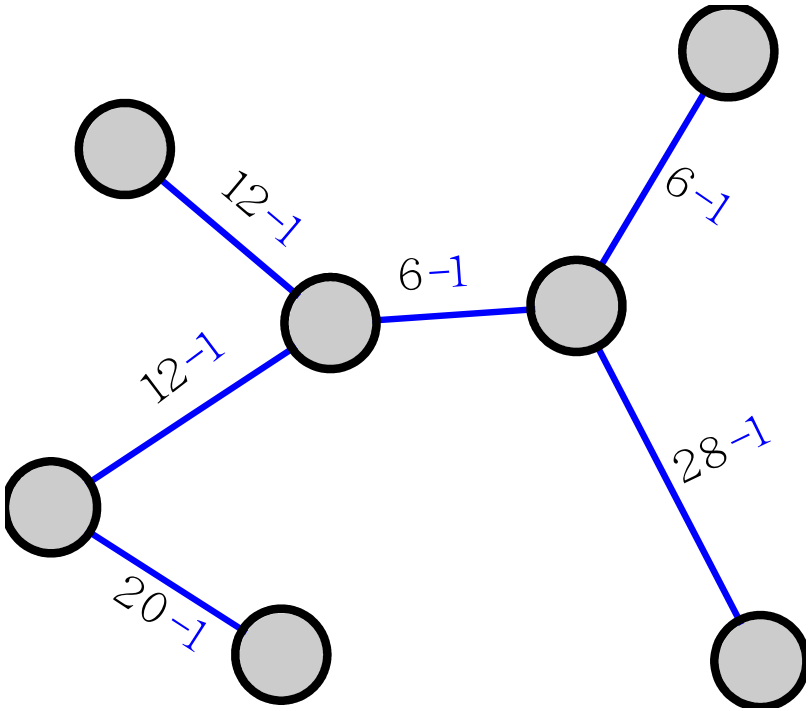
Dynamic Trees Problem

- 维护一个包含 N 个点的森林， 并且支持形态和权值信息的操作 .
- 权值信息
  - 路径操作： 对一条简单路径上的所有对象进行操作



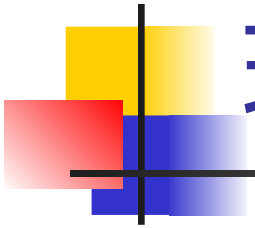
Dynamic Trees Problem

- 维护一个包含 N 个点的森林， 并且支持形态和权值信息的操作 .
- 权值信息
  - 路径操作： 对一条简单路径上的所有对象进行操作
  - 树操作： 对一棵树内的所有对象进行操作



现有结果

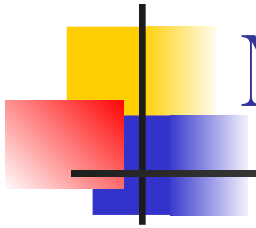
基本原理	Euler Tour	Path Decomposing	Divide and Conquer
相关实现	Euler Tour Trees	ST-Trees[1,2]	(Self Adjust) Top-Trees[3,5]
局限性	不支持路径操作	不支持树权操作	常数过大



理论补充

- 对于一个完整的动态树问题， 目前公认的下界是 $O(\log_2 N)$ per operation， 并已经被上述方法达到 .
- 但是由于巨大的常数因子， 动态树在实践中并没有发挥应有的作用 .
- 动态树问题仍然没有完美解决， 并且仍然处在热烈讨论中 .

Part II. Solving Dynamic Trees Problem

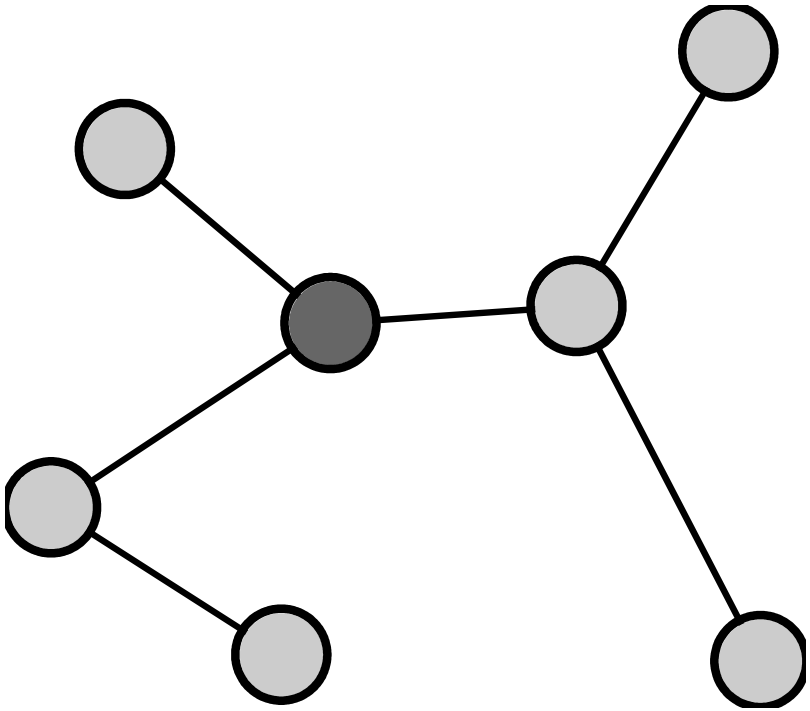


New Idea

- 在这里， 我向大家介绍一种新的解决动态树问题的思路． 这种思路简单， 而且， 可以得到效率非常高的具体实现．

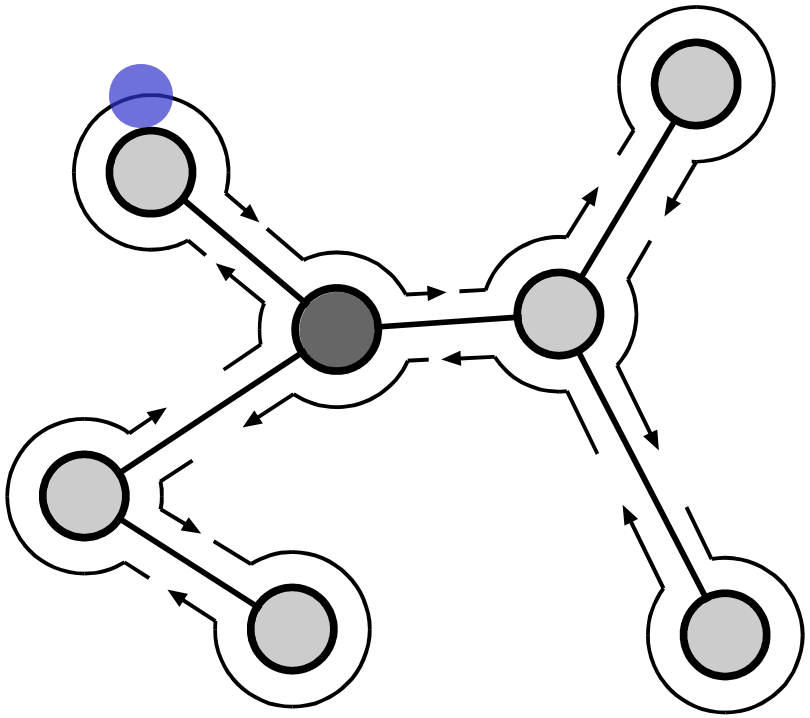
Ⅰ. 树， 与其平面刻画 .

- 一棵树的平面刻画，直观地说就是将一棵树的点和边画在平面上． 边与边仅在顶点处相交 .



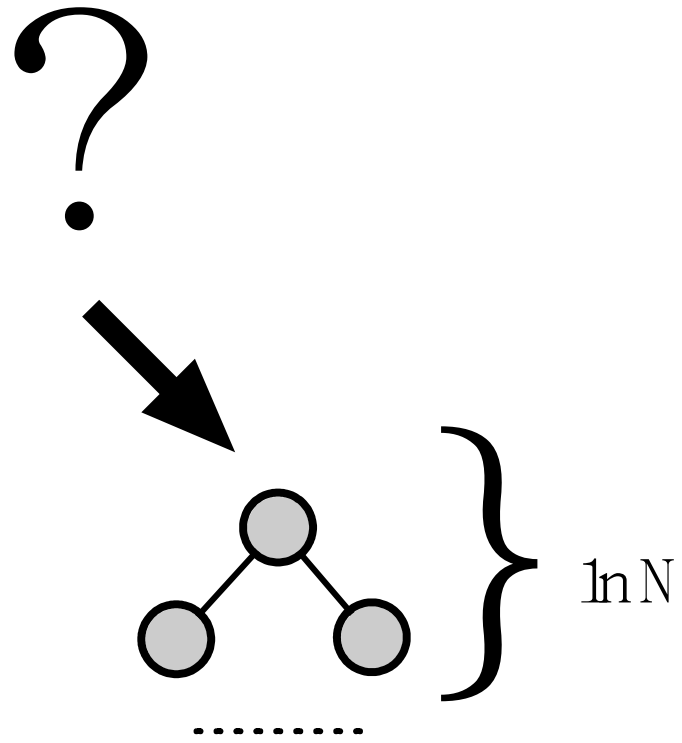
Ⅰ. 树， 与其平面刻画 .

- 一棵树的平面刻画，直观地说就是将一棵树的点和边画在平面上． 边与边仅在顶点处相交 .
- 确定一棵树的平面刻画， 等价于确定这棵树的 Euler Tour.



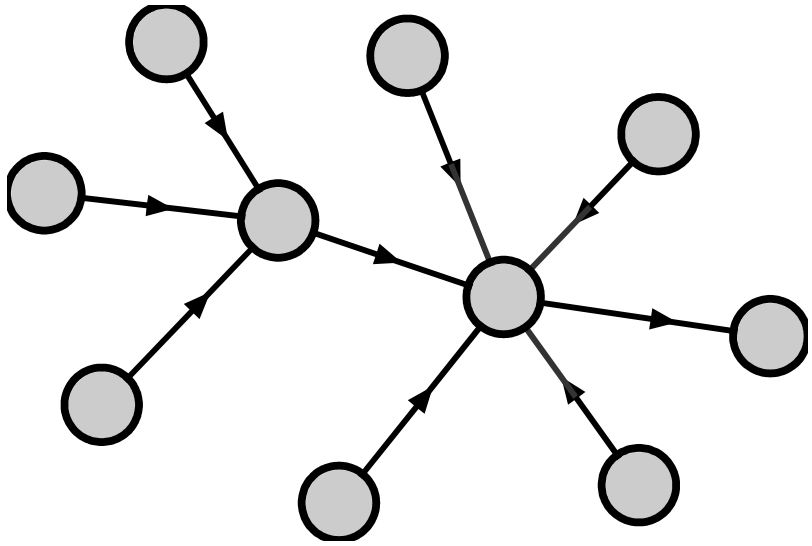
II. 等价映射

- 事实上， 所有解决动态树问题的方法，归根结底都使用等价映射的基本思想．
- 即， 将任意形态的树（原树）映射到度限制， 深度平均的新树（像树）.



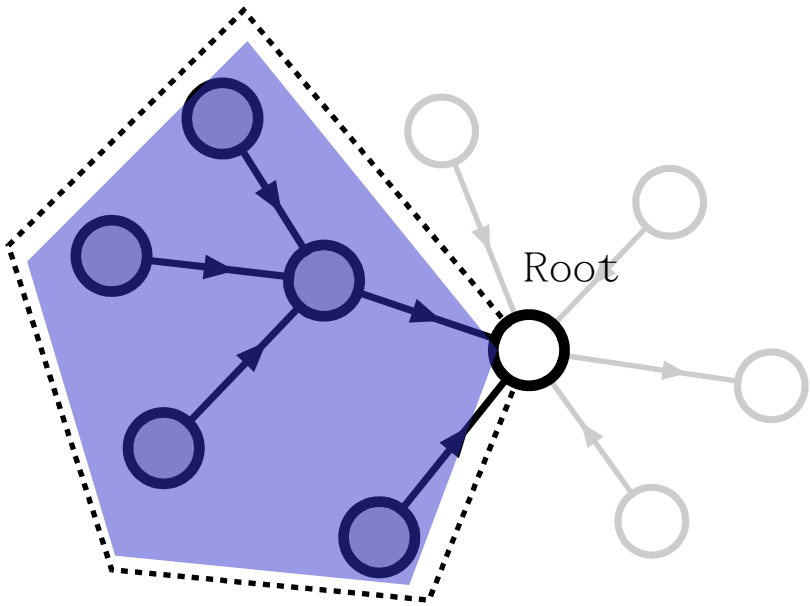
III. Rake & Compress

- 这里介绍一种 Rake & Compress[5,6] 方法 .
- 即将原树映射到一棵 Rake & Compress Trees (**Abbr.** R&C Trees).
- R&C Trees 由 Rake 节点和 Compress 节点组成 .



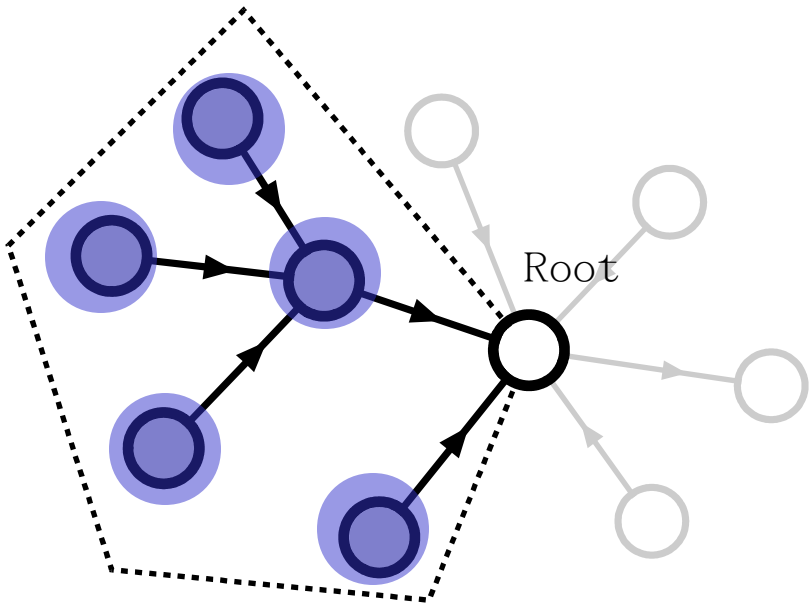
1. Rake Nodes

- Rake 节点 i 是原树中以某节点为根的有根子树的映射 .



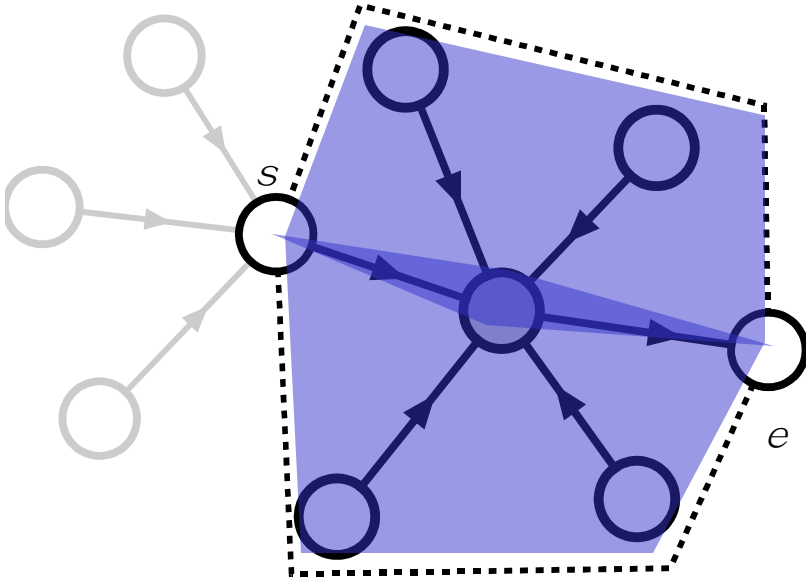
1. Rake Nodes

- Rake 节点 i 是原树中以某节点为根的有根子树的映射 .
- 特别地， 如果该节点仅包含根本身， 那么该 Rake 节点没有后继（叶子节点）. 否则令 Next(i) 表示 i 所代表的除了根以外的其它点组成的集合 .



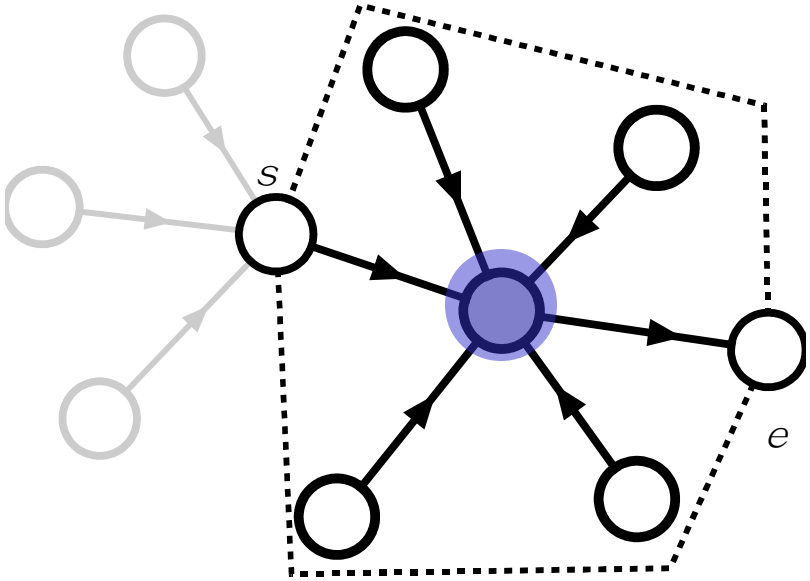
2. Compress Nodes

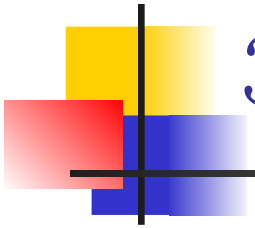
- Compress 节点 j，是原树中以某条路径为根的有根子树的映射.



2. Compress Nodes

- Compress 节点 j，是原树中以某条路径为根的有根子树的映射．
- 特别地，如果路径长度为 1．那么该 Compress 节点没有后继．否则令 Next(j) 表示 j 代表的路径上的非端点集合．



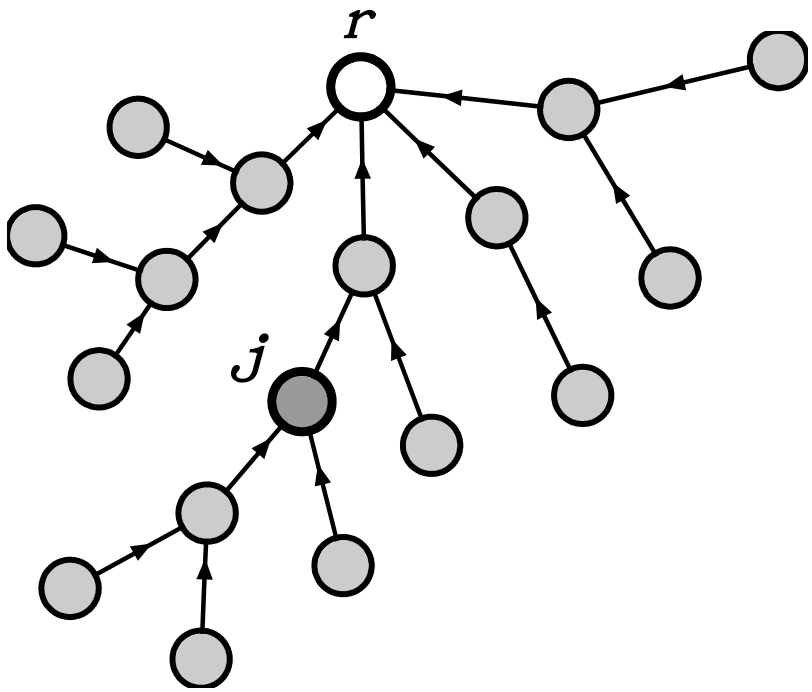


3. R&C Trees

- 对于一个非叶子 Rake/Compress 节点 i, Next(i) 非空 .
- 对于每个 Next(i) 中的元素 j. 我们采用如下方法划分节点 i:

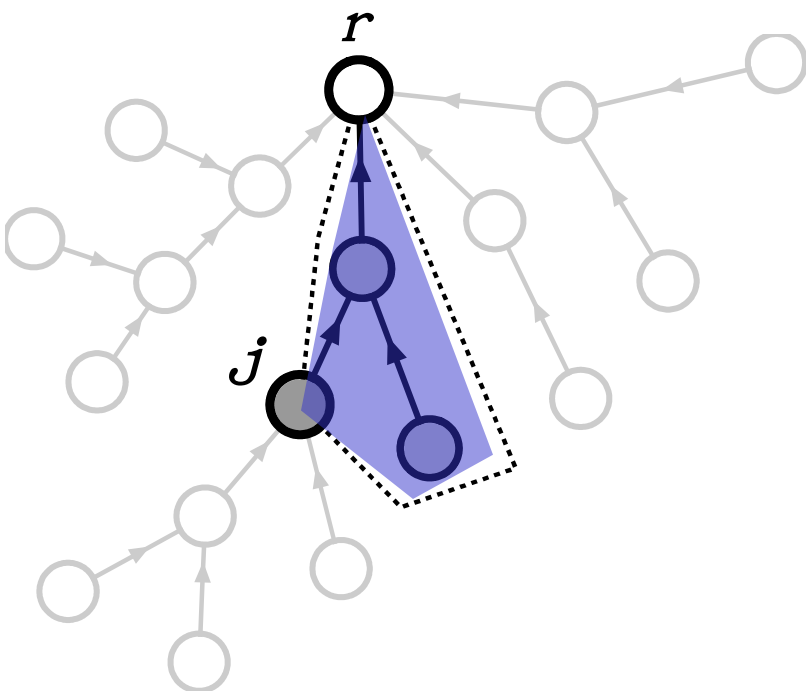
1’ Rake 节点的划分

- 令 r 表示 i 的根 .



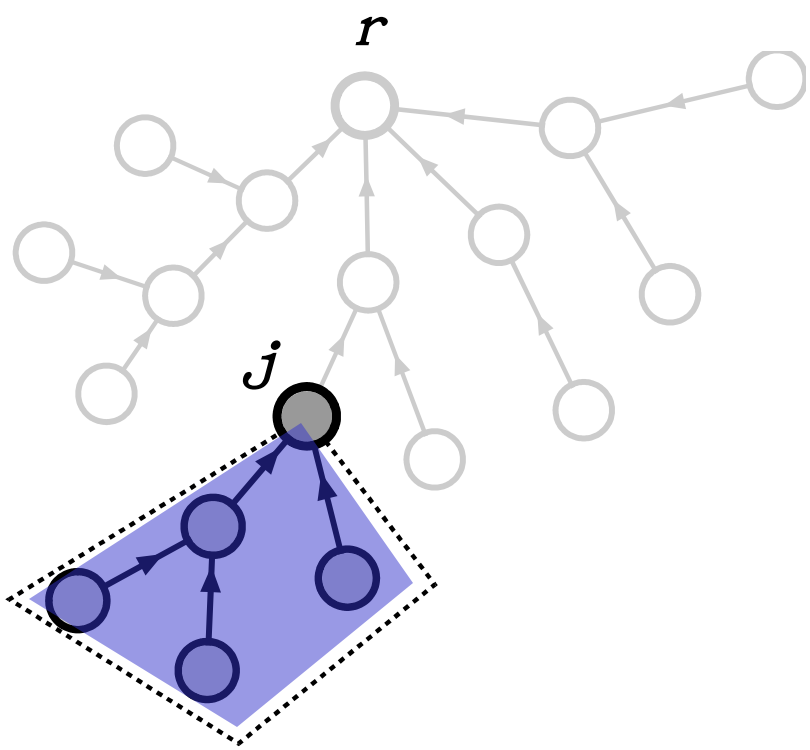
1’ Rake 节点的划分

- 令 r 表示 i 的根 .
- 将路径 $j \rightarrow r$ 作为新的 Compress 节点 .



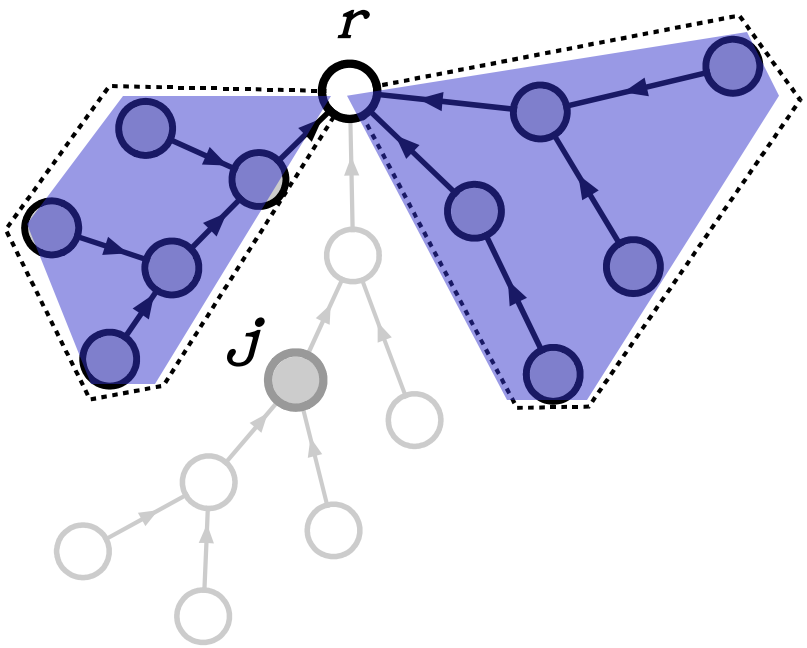
1’ Rake 节点的划分

- 令 r 表示 i 的根 .
- 将路径 $j \rightarrow r$ 作为新的 Compress 节点 .
- 将 j 和 j 的子孙作为新的 Rake 节点 .



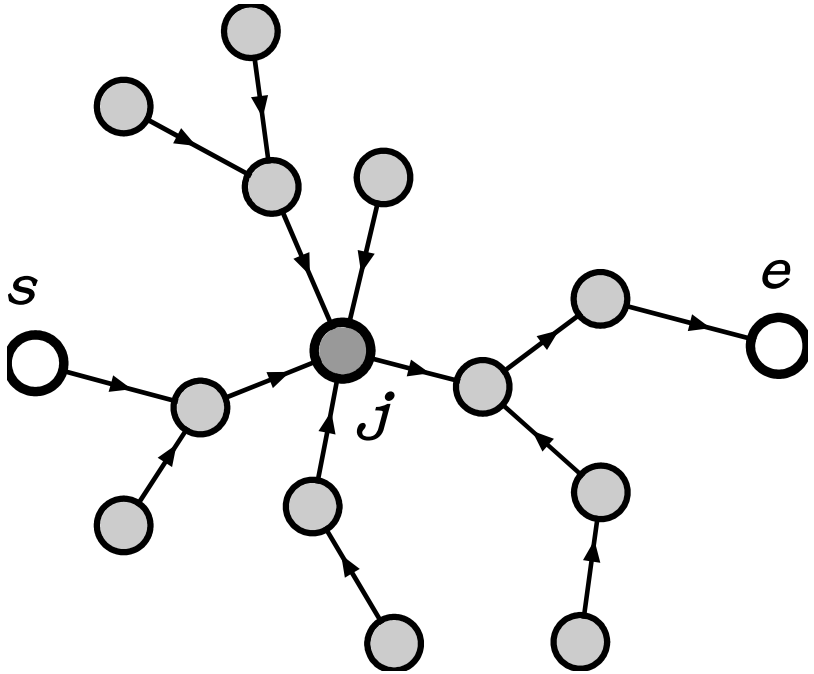
1’ Rake 节点的划分

- 令 r 表示 i 的根．
- 将路径 $j \rightarrow r$ 作为新的 Compress 节点．
- 将 j 和 j 的子孙作为新的 Rake 节点．
- i 中路径 $j \rightarrow r$ 的左手方向和右手方向各为一个新的 Rake 节点．



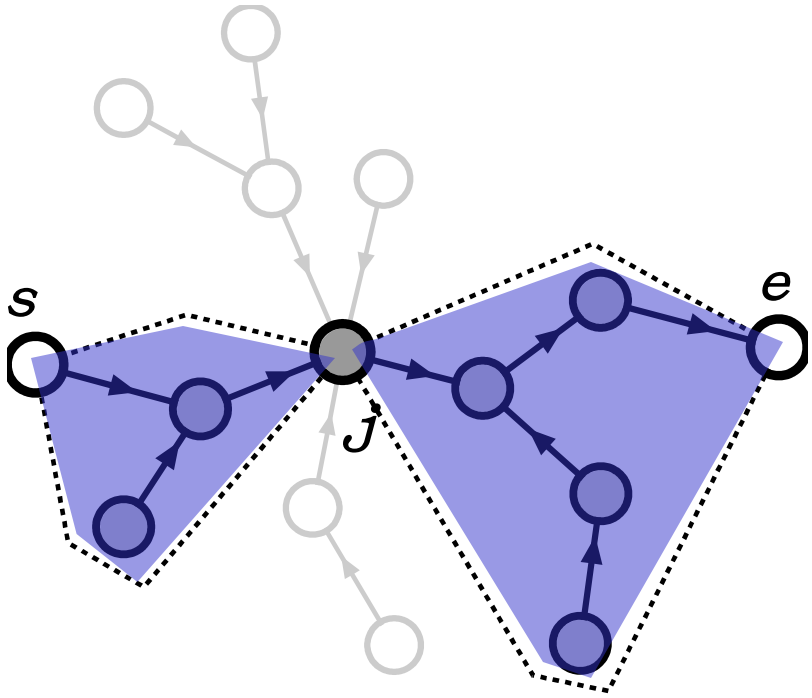
2’ Compress 节点的划分

- 令 s,e 分别表示 i 中路径的头和尾 .



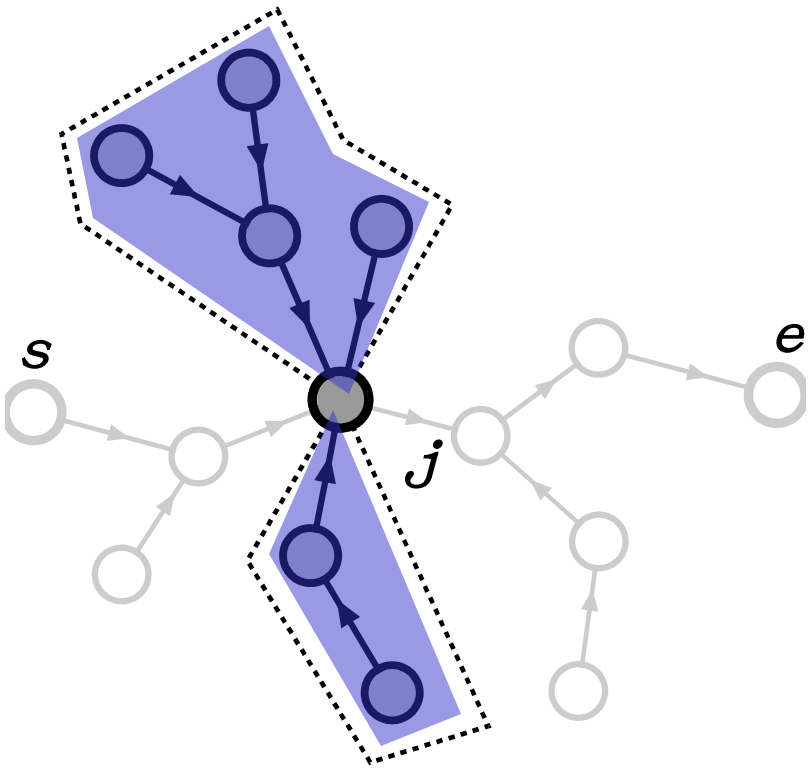
2’ Compress 节点的划分

- 令 s,e 分别表示 i 中路径的头和尾 .
- $s \rightarrow j, j \rightarrow e$ 分别成为新的 Compress 节点



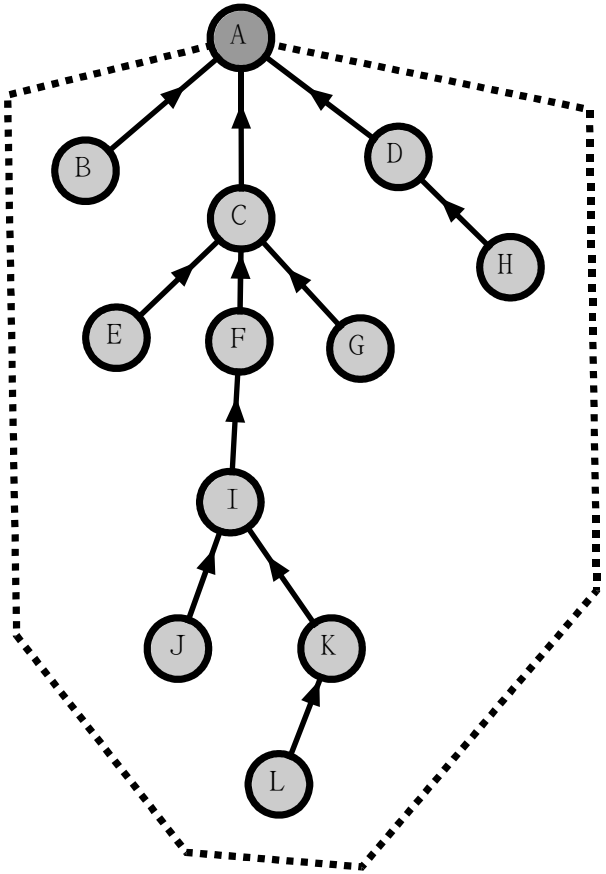
2’ Compress 节点的划分

- 令 s,e 分别表示 i 中路径的头和尾．
- $s \rightarrow j, j \rightarrow e$ 分别成为新的 Compress 节点
- j 的其他子孙被划分成两部分， 分别作为新的 Rake 节点．



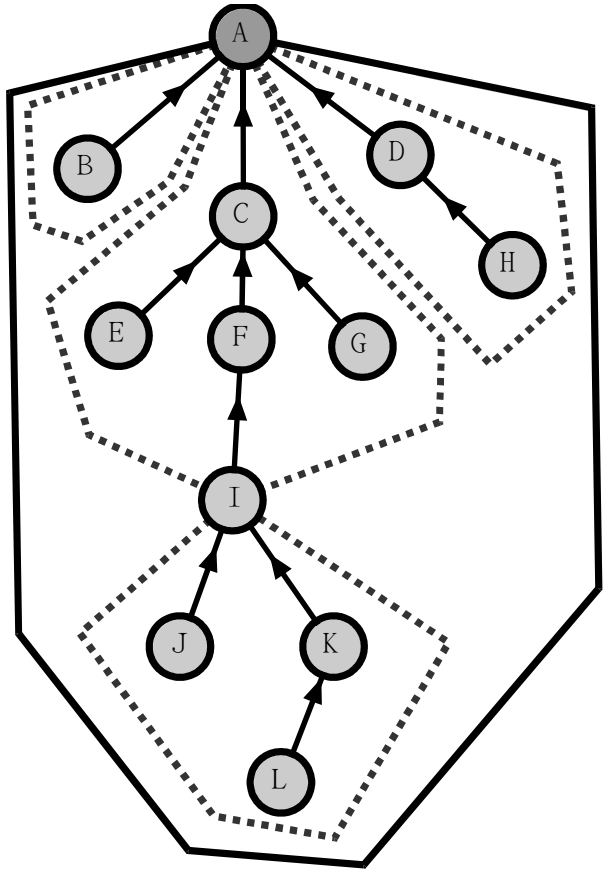
R&C Trees

- 约定， 一个有根树对应 R&C Trees 就是整个树的 Rake Node.
- 这样的分解方式本身就保证度的限制（一个节点最多被剖分出 4 个子节点）
- 我们以右图为例，展示一棵原树如何被映射到像树。



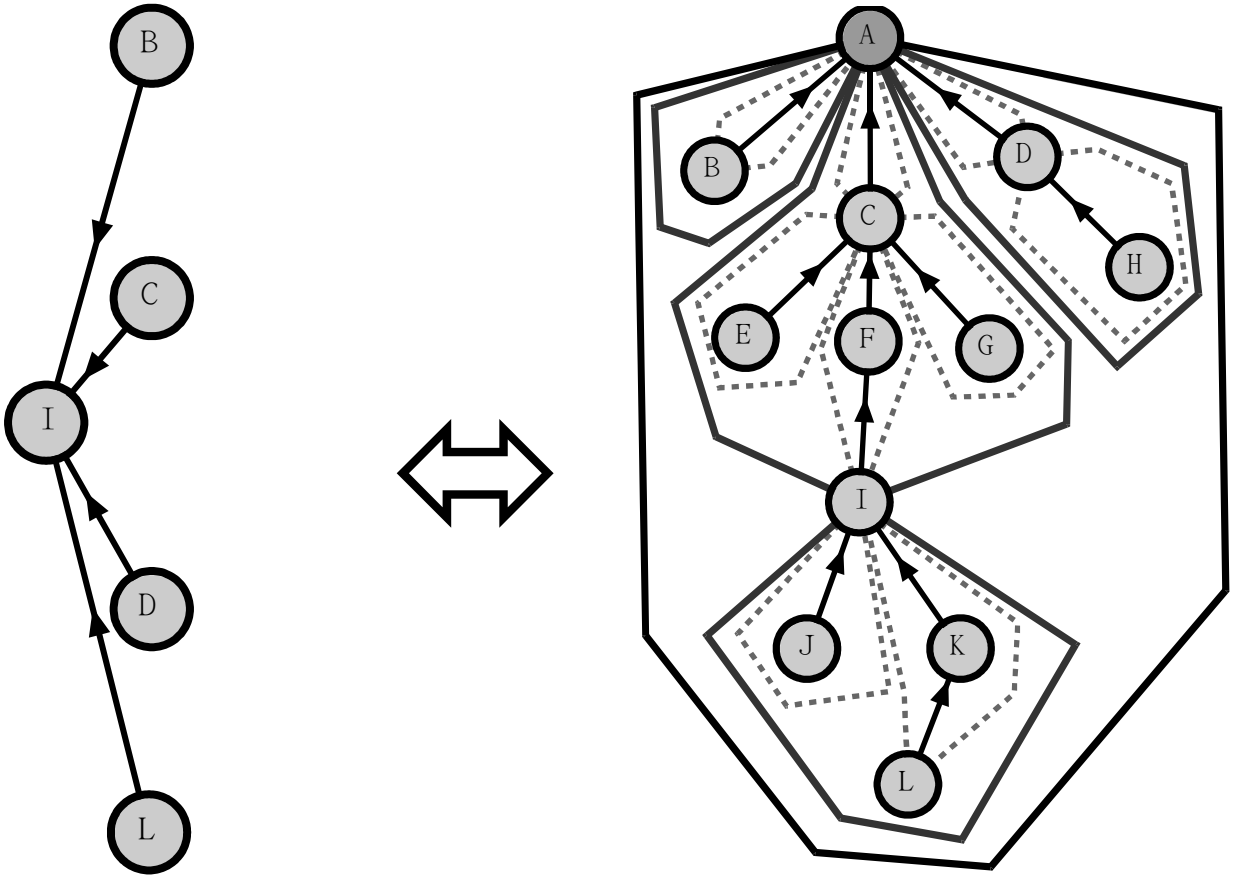
Level 1

选取I作为第1层剖分点，原Rake节点被剖分成4个部分，4个部分分别剖分。



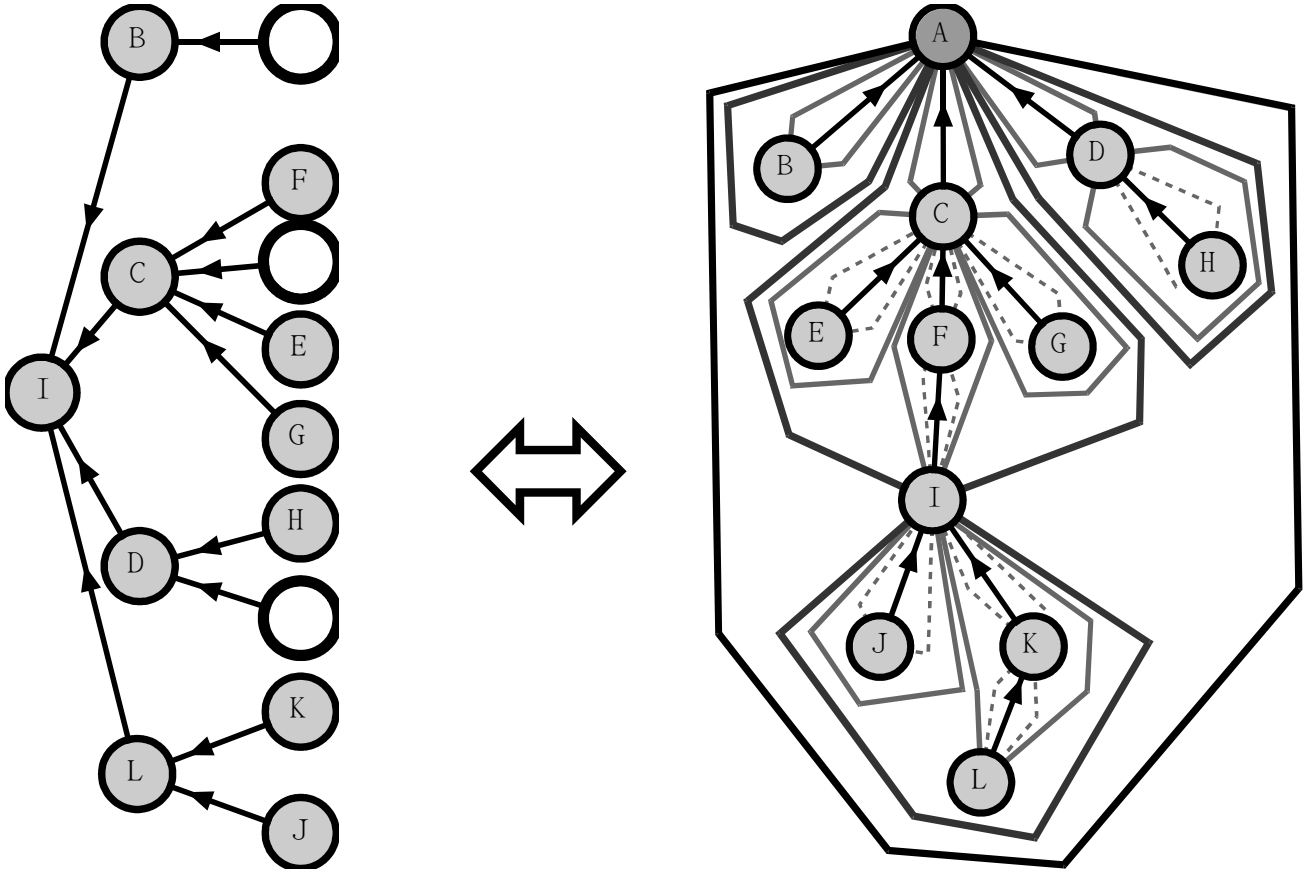
Level 2

选取B,C,D,L 作为第2 层剖分点。



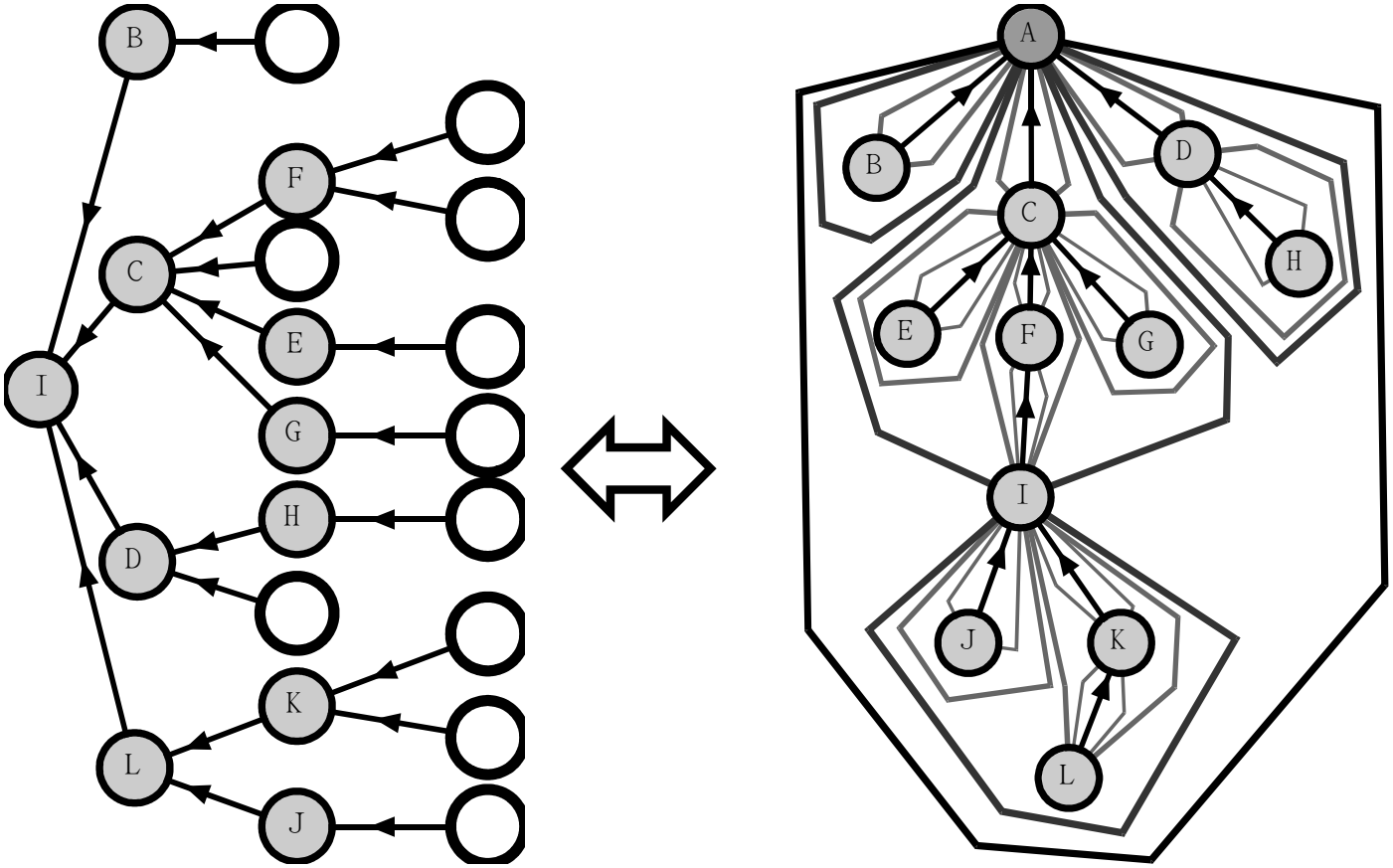
Level 3

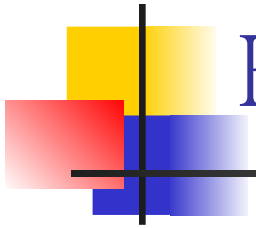
选取F,E,G,H,K,J 作为第3 层剖分点。



Final

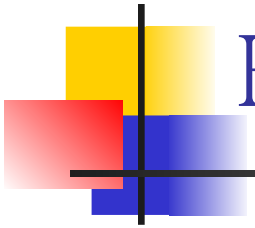
这样，我们就构建出了整个树的R&C Trees。





Randomized

- 决定 R&C Trees 深度的关键因素在于选择 Next(i) 中元素的方法．
- 一个比较好的方法是， 随机选择！
- 如果等概率的选择 Next(i) 中的元素，可以证明这样的深度下界是 $\Omega(\ln^2 N)$．问题并没有完全解决．
- 必须采取更为合理的随机策略．

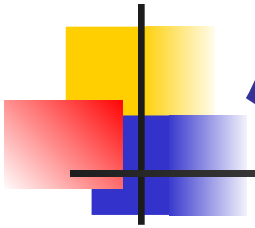


Randomized

- 对于 Rake 节点， 仍然采取等概率的方法选择 .
- 对于 Compress 节点 i, j 表示 Next(i) 中的元素， 令 Weight(j) 表示 j 剩余的子孙个数 (包括 j) .
- 现在以 Weight(j) 作为加权， 令 S 表示所有 Weight(k) 的和， 则 j 有 Weight(j)/S 的概率被选择 .

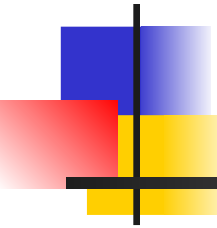
Randomized : Master Theorem

- 可以证明， 通过这样合理的改造， 一个 N 的点的树可以被映射到一个期望深度为 $O(\ln N)$ 的 R&C Trees.
- 基于这种思想， RP-Trees 被提出． 事实上， 这就是 Treap[7] 通过 R&C 思想在任意树的拓展．
- 通过这种思想， 我们可以得到均摊，甚至是严格深度 $O(\ln N)$ 的算法．



小结

- 相比较于 Path Decomposing 和 Divide & Conquer，Rake & Compress 具有思想简单，常数小，实现复杂度低等特点．
- R&C 思想最大的特点是，利用这种思想可以很方便地将各种平衡二叉树技巧拓展到任意树形态上去．
- 为 Dynamic Trees Problem 注入了新的血液．



Part III. Applications

Applications …

- 网络优化
  - 最大流 [4] √
  - 最小费用流
  - …
- 动态算法
  - 动态连通性 [3,8]
  - 动态最小生成森林 [3]
  - …
- …

最大流问题

- 最大流问题是非常经典的图论问题． 经典的解决算法有最短增广路， 预流推进．
- 通过改造最短增广路算法并应用动态树，可以得到 $O(NM\ln N)$ 的算法． N 为点数， M 为边数．
- 经典的最短增广路算法是通过 BFS， 每次在残余网络中找到一条最短 S-T 增广路并进行增广．

最大流问题

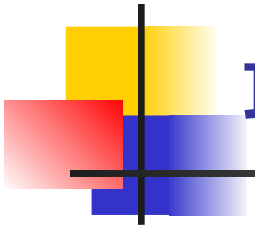
- 引理： 只需要 $O(NM)$ 次增广 .
- 证明： 考察当前最短路的必要边集 A
  - 所有 S→T 的最短路全部由 A 中元素组成
  - $|A| \leqslant M$
- 每次都找到最短增广路， 增广后 A 中元素必有一条边被删除（残余量为 0）.

最大流问题

- 在 S→T 最短路长度被提高之前不可能有边从 A 外加入到 A 内 .
- $O(M)$ 次增广后 S→T 增广路长度必被提高 . 因此最多执行 $O(NM)$ 次增广 .
- 综上所述， 引理得证 .

最大流问题

- 原始算法的时间复杂度为 $O(NM^2)$.
- 令 $D(x)$ 表示 x 到 T 的最短增广路下界.
  - 对于所有残余量 >0 的边 $u \rightarrow v$, 满足 $D(u) \leqslant D(v)+1$.
  - 如果 $D(u)=D(v)+1$, 则称 $u \rightarrow v$ 为有效边.
  - 引理: 全部由有效边组成的到 T 的路径一定是最短路径!



最大流问题

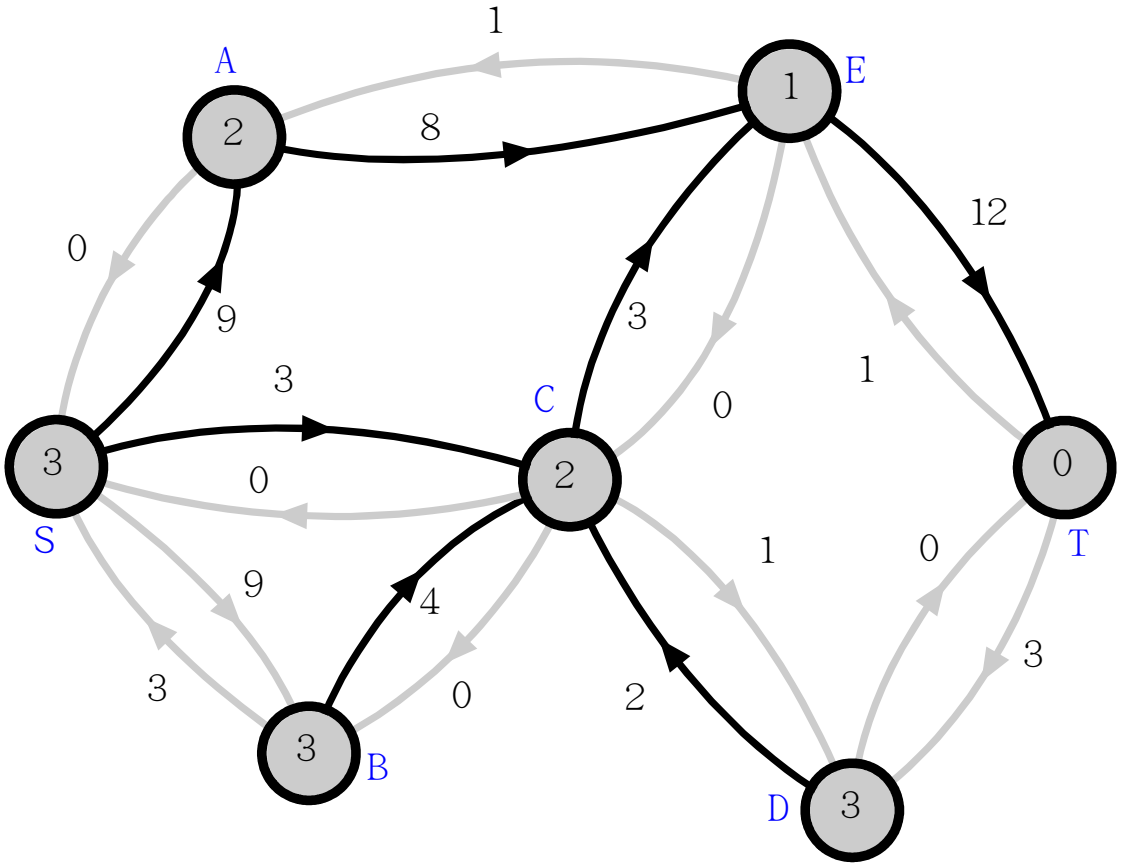
- 每次在残余网络中沿着让 $D(x)$ 递减的有效边前进． 并不断修正（抬高）$D(x)$．可以证明， 该优化方法将寻找最短增广路的时间降为均摊 $O(N)$， 所以需要的总时间降为 $O(N^2M)$．
- 那么， 能不能再次改进呢？

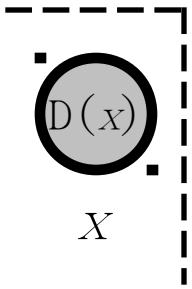
最大流问题

- 一个想法是， 维护有效边子集的生成森林．
- 如果 S，T 不在同一棵树内． 令 R 为 S 所在树内 D 值最小的点（最低点）， 如果存在有效边 $R \rightarrow R'$， 则执行 Link(R,R')．
- 如果此时 R 没有连出的有效边， 显然 D(R) 是可以改进的． 即这个下界是松的， 此时我们抬高 D(R)． 并更新有效边集．
- 当 S 和 T 都在同一棵树内时（即最低点为 T)，对树中的 S-T 路径进行增广．
  - 每次所增广的路都是最短增广路．

Residual network

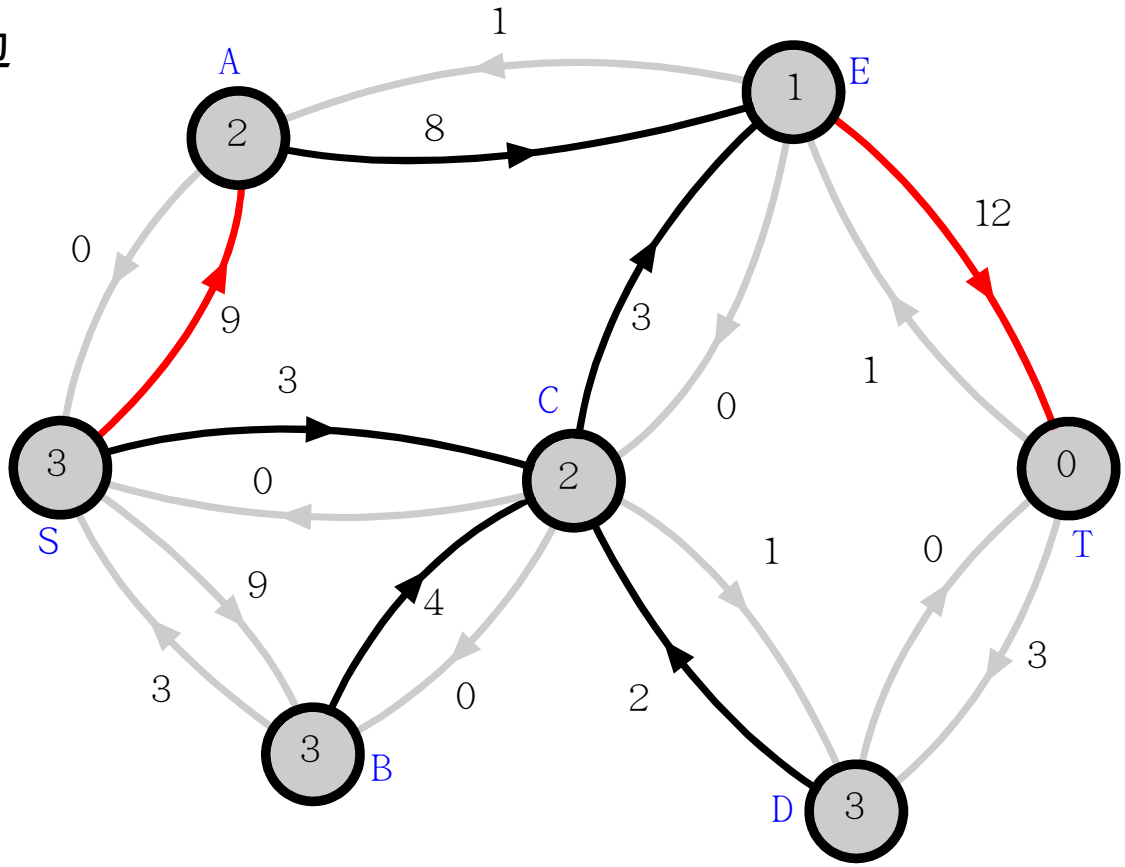
黑色边为有效边

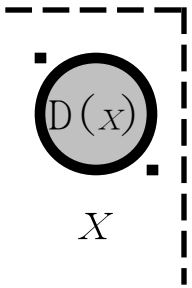




Residual network

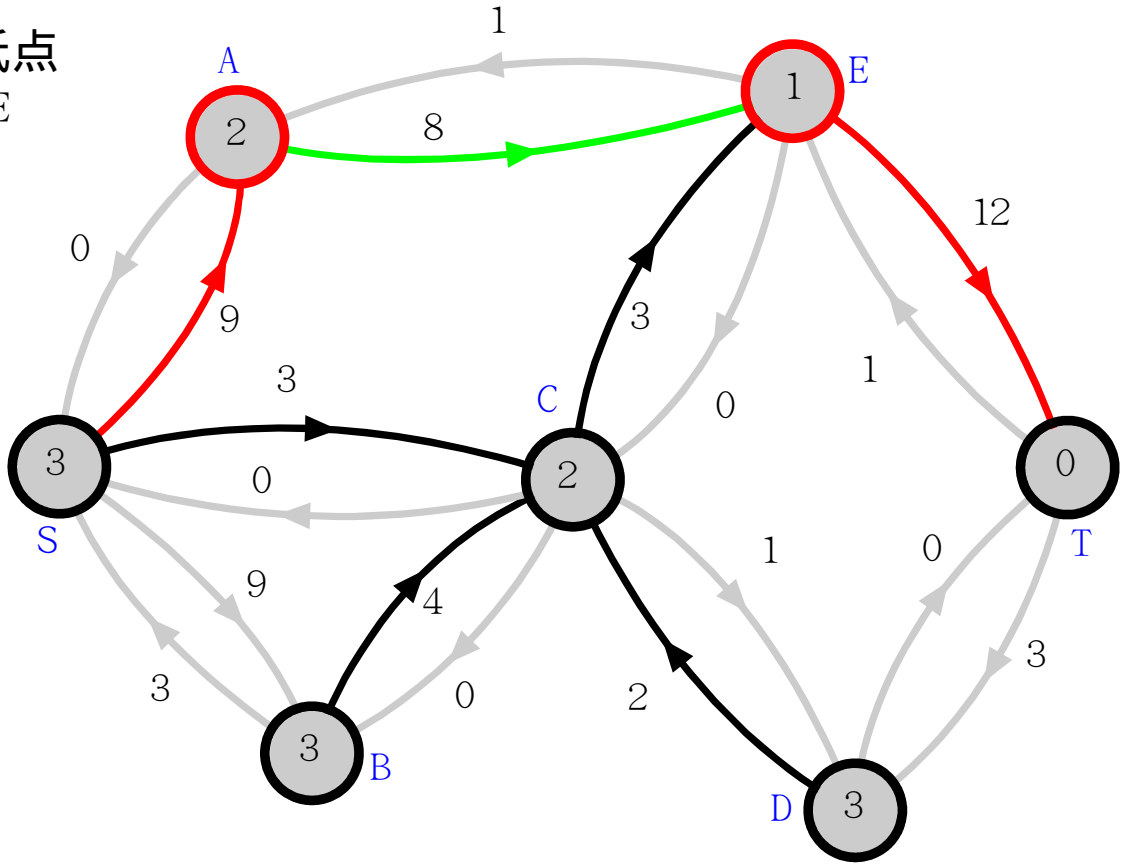
红色边为生成森林边

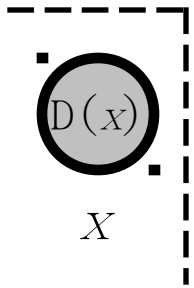




Residual network

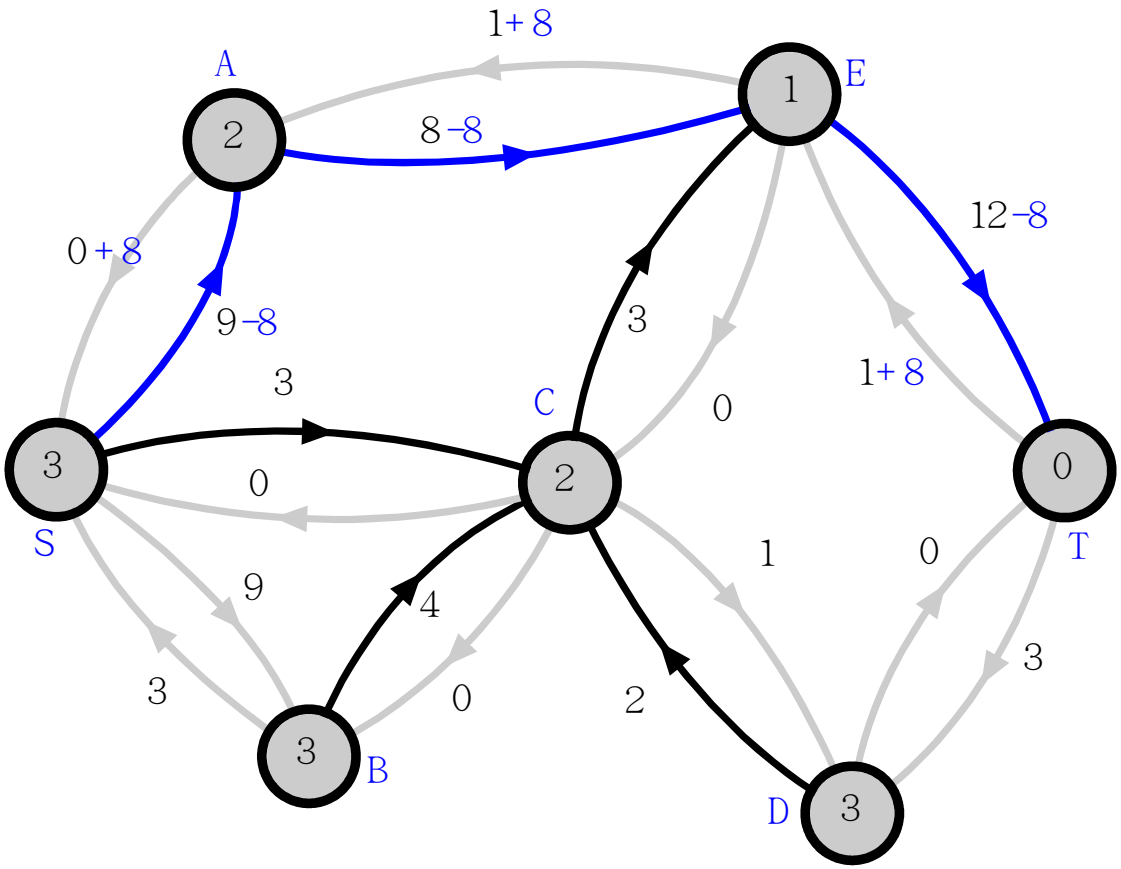
找到一条由当前最低点
A 连出的有效边 A→E
执行 Link(A,E)

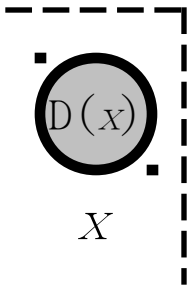




Residual network

找到最短增广路
S→A→E→T
增广量为 8

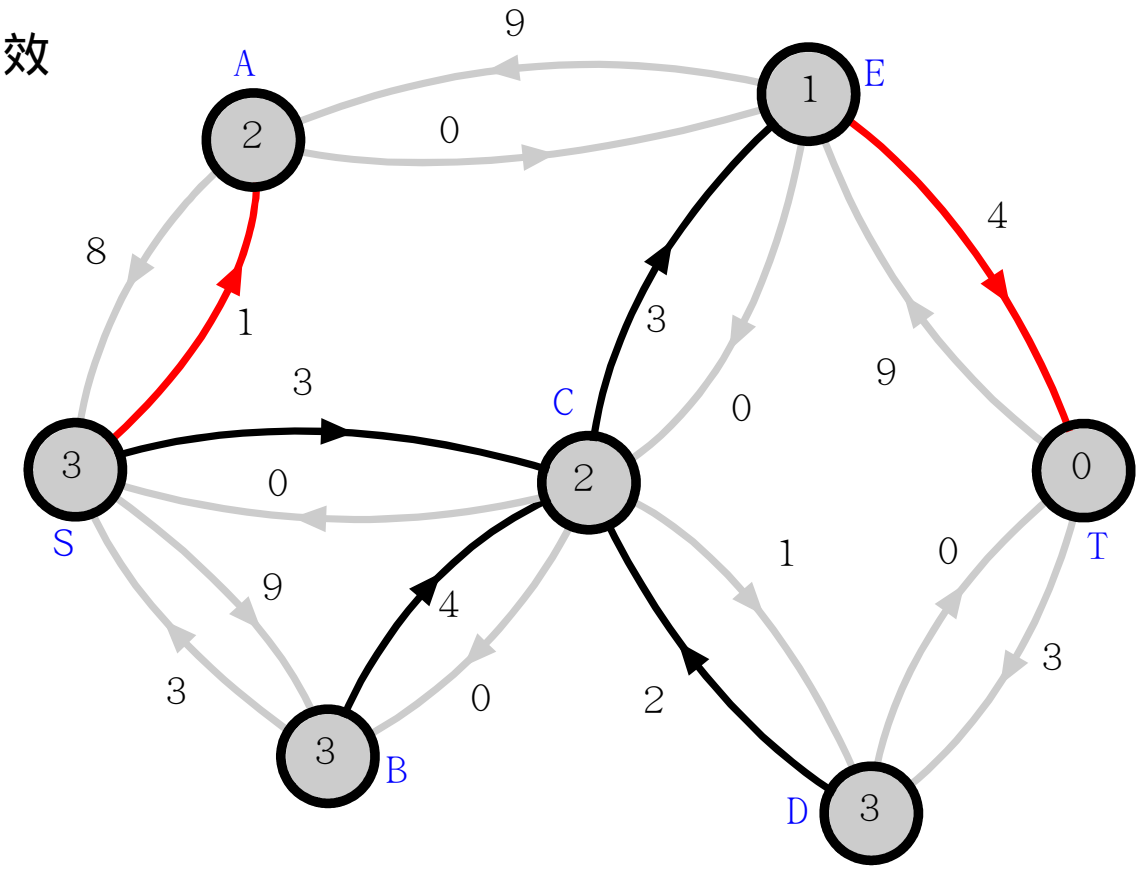


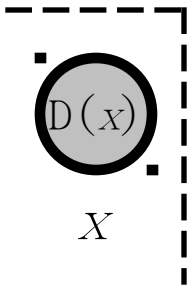


Residual network

增广后 A→E 不再有效

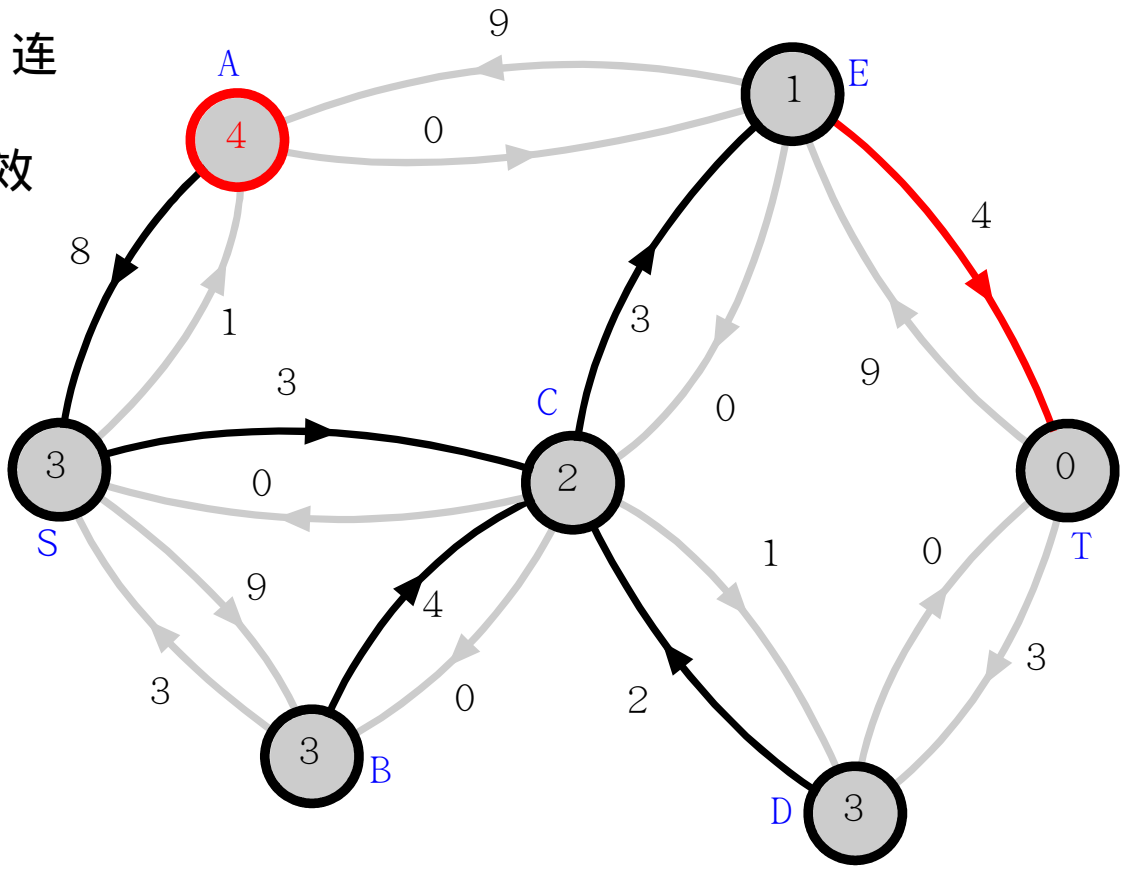
Cut(A,E)

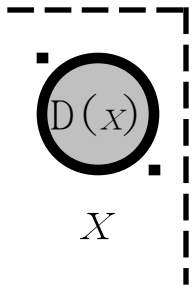




Residual network

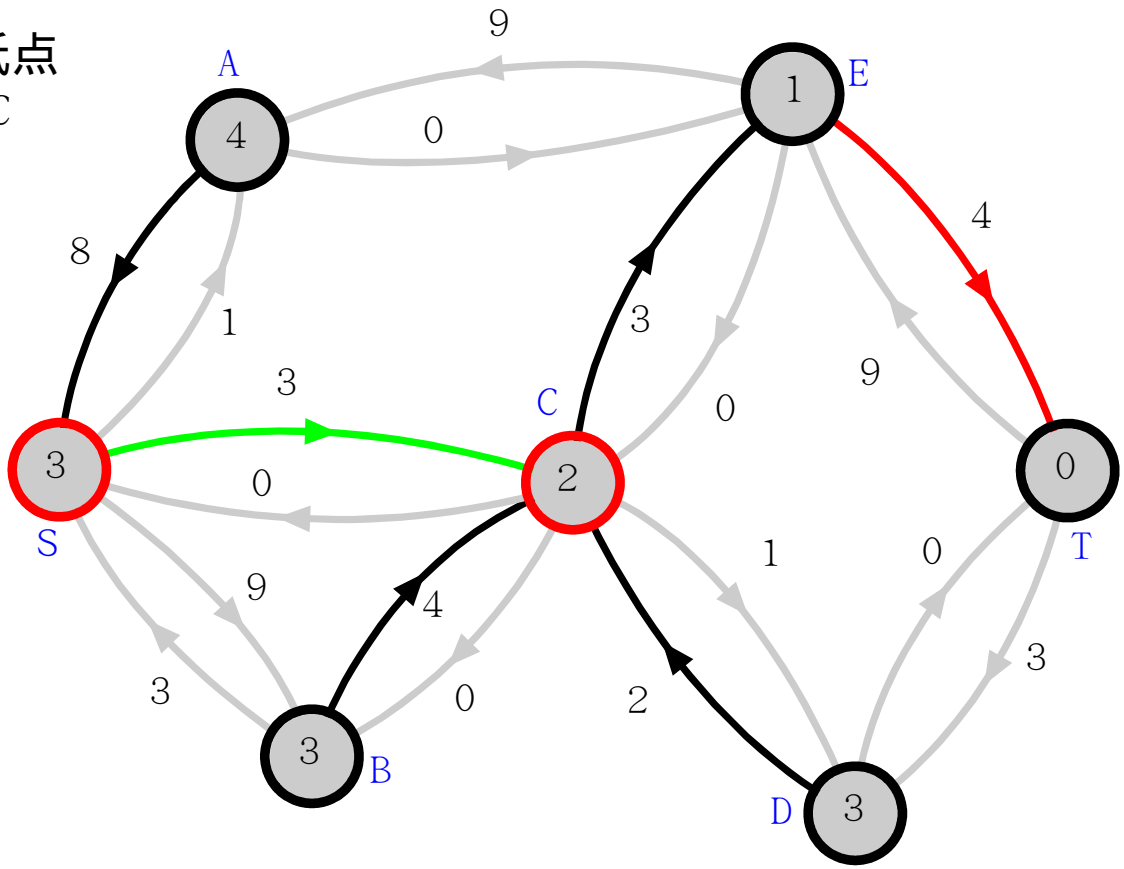
尝试找到从最低点 A 连出的边，失败
抬高 D(A) 并更新有效边

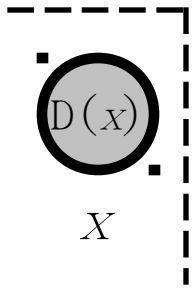




Residual network

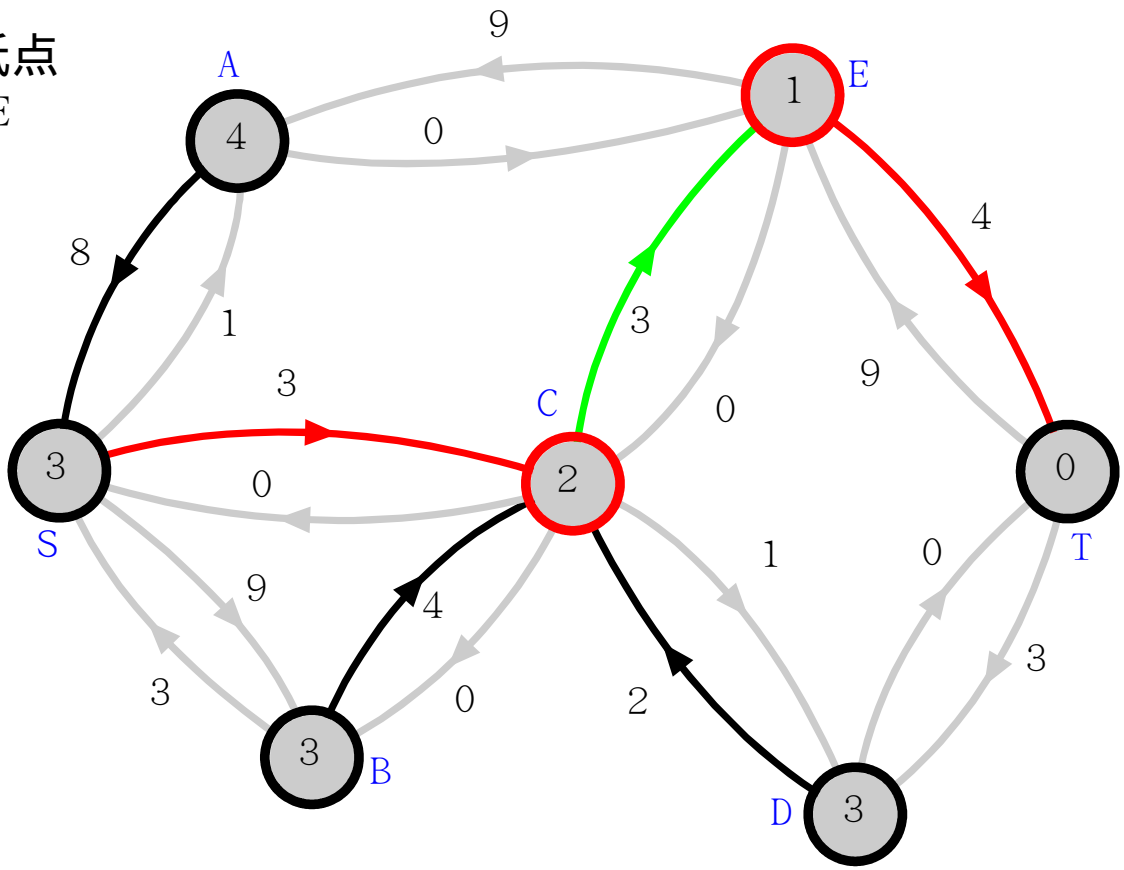
找到一条由当前最低点S连出的有效边 S→C
执行 Link(S,C)

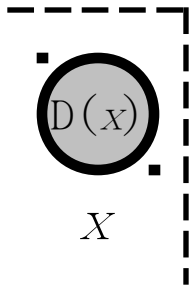




Residual network

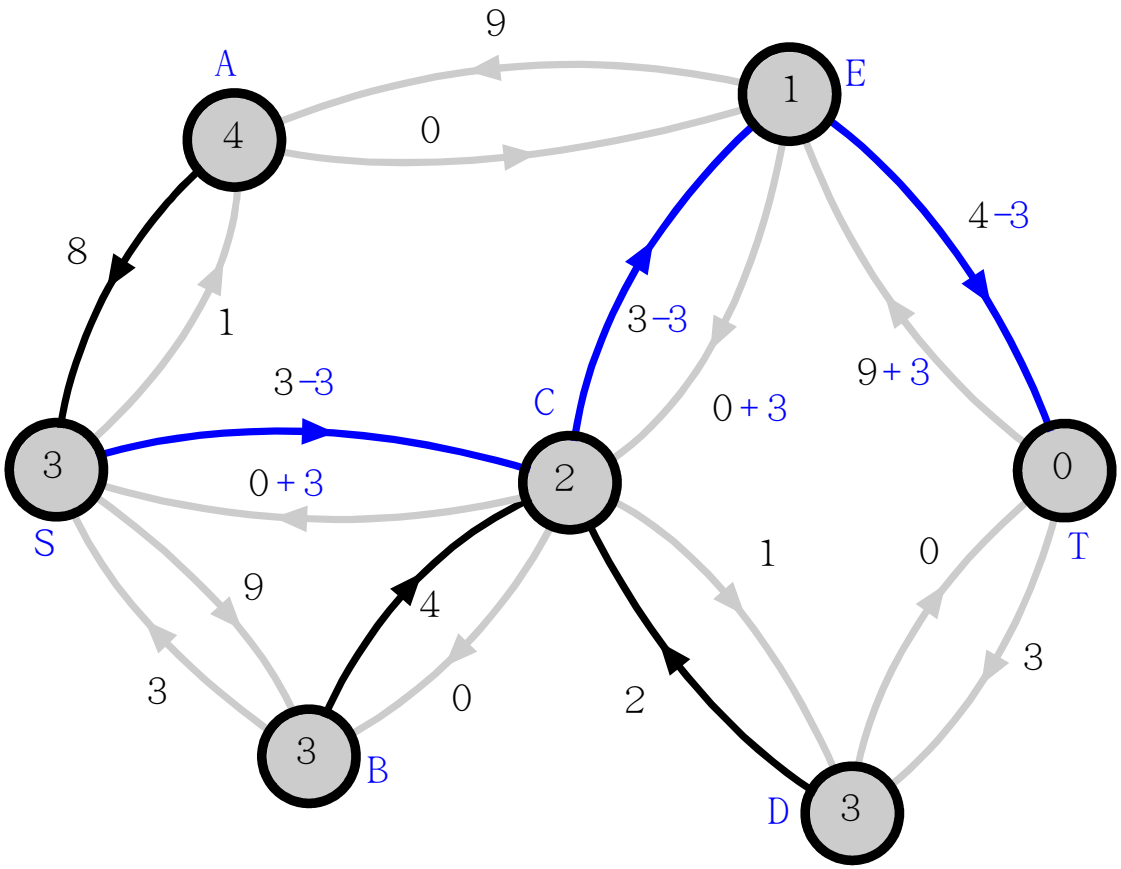
找到一条由当前最低点C连出的有效边 C→E
执行 Link(C,E)

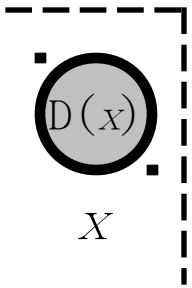




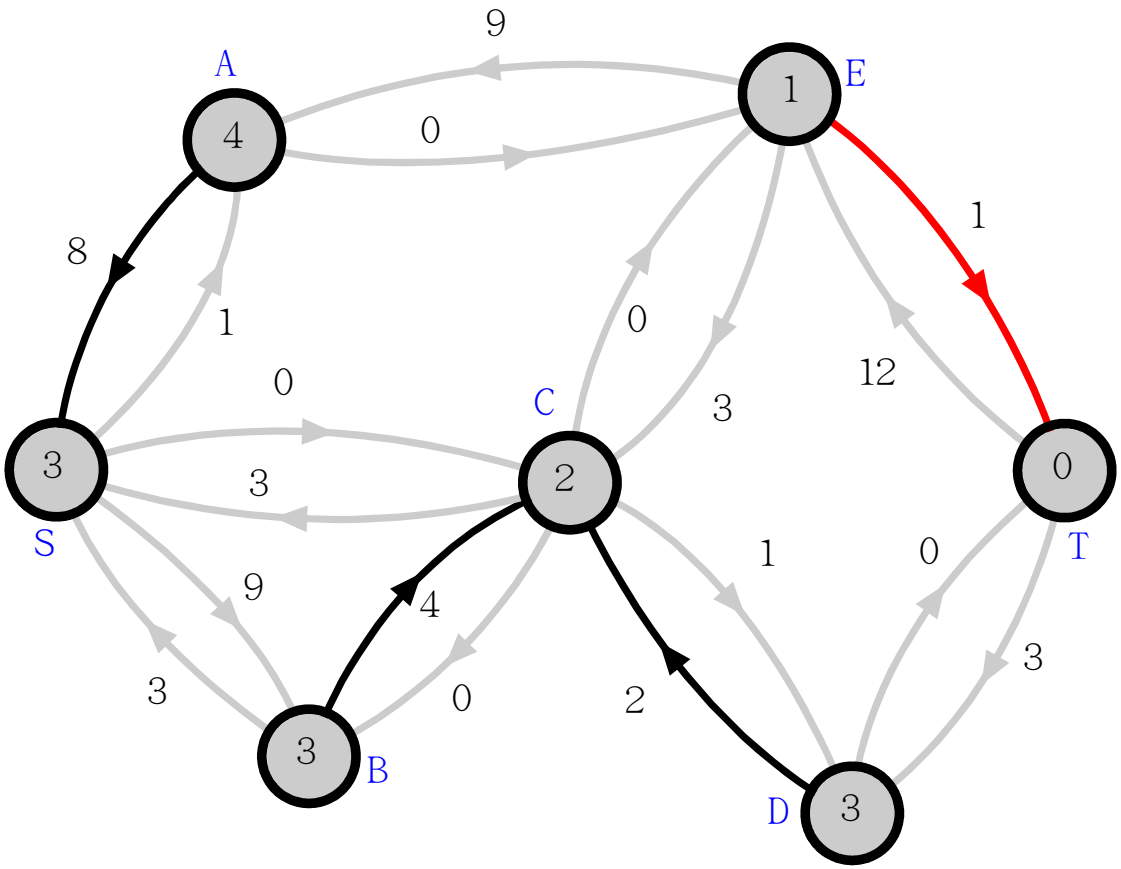
Residual network

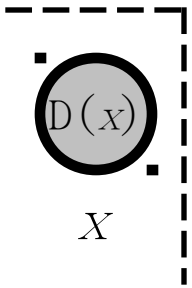
找到最短增广路
S→C→E→T
增广量为 3

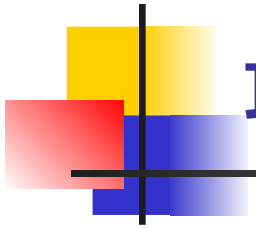




Residual network







最大流问题

- 由于流程和最短增广路一样， 故正确性显然．
- 现在考察其时间复杂度．

最大流问题

- 前面已经说过， 最多执行 O(*NM*) 次增广， 所以在增广上的总时间为 O(*NM*1n*N*)．
- 因为 0⩽D(*x*)⩽*N*， 所以最多执行 O(*NM*) 次 Cut 操作． 因此花费在 Cut 上的总时间为 O(*NM*1n*N*)．
- 又因为在一个点的 D 值被抬高之前， 每个点均摊被 Link 过 1 次． 综上所述， 总的时间复杂度为 O(*NM*1n*N*)

时间复杂度

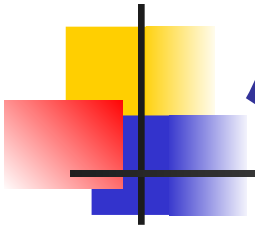
算法名称	**原始算法**	引进 $D(x)$	动态树
找最短路	$O(NM^2)$	$O(N^2M)$	$O(NM\ln N)$
进行增广	$O(N^2M)$	$O(N^2M)$	$O(NM\ln N)$
调整 $D(x)$	×	$O(NM)$	$O(NM\ln N)$
总计	$O(NM^2)$	$O(N^2M)$	$O(NM\ln N)$

时间复杂度

算法名称	原始算法	引进 D(x)	动态树
找最短路	$O(NM^2)$	$O(N^2M)$	$O(NM\ln N)$
进行增广	$O(N^2M)$	$O(N^2M)$	$O(NM\ln N)$
调整 D(x)	×	$O(NM)$	$O(NM\ln N)$
总计	$O(NM^2)$	$O(N^2M)$	$O(NM\ln N)$

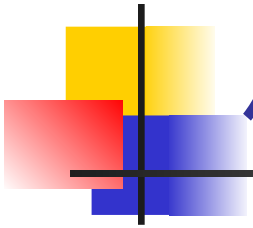
时间复杂度

算法名称	原始算法	引进 $D(x)$	**动态树**
找最短路	$O(NM^2)$	$O(N^2M)$	$O(NM\ln N)$
进行增广	$O(N^2M)$	$O(N^2M)$	$O(NM\ln N)$
调整 $D(x)$	×	$O(NM)$	$O(NM\ln N)$
总计	$O(NM^2)$	$O(N^2M)$	$O(NM\ln N)$



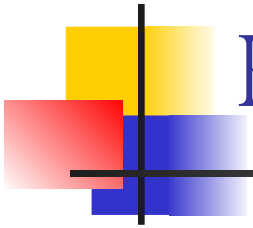
小结

- 使用动态树问题来优化算法的关键是解决瓶颈问题．
- 在本例中， 我们将复杂度摊平到了每一种操作中． 使得总的时间复杂度获得了质的飞跃．
- 其实， 不仅仅是最大流算法， 网络优化的很多算法都可以使用动态树来优化，并得到很好的理论复杂度．



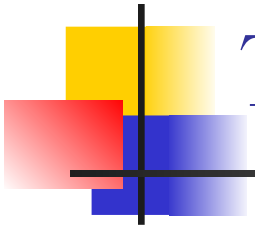
总结

- 研究动态树问题是非常有价值的， 研究它可以加深对图论的理解． 其解决方法使用的技巧亦非常具有启发性． 更何况其问题本身又是那么美妙！
- 对于一个复杂的经典问题， 我们不应仅仅满足于“知道” 或者“学会”， 更需要融会贯通． 这对我们无论是日常学习或者竞赛活动都是非常有好处的．



References

- [1] Daniel D. Sleator, Robert Endre Tarjan, A data structure for dynamic trees, *Journal of Computer and System Sciences*, v.26 n.3, p.362-391, June 1983.
- [2] Daniel Dominic Sleator , Robert Endre Tarjan, Self-adjusting binary search trees, *Journal of the ACM (JACM)*, v.32 n.3, p.652-686, July 1985
- [3] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Minimizing diameters of dynamic trees. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 270-280, 1997.
- [4] Ahujia, R. K. Dynamic Trees Implementations. *Network Flows: Theory, Algorithms, and Applications*, p.265-273.
- [5] R. Tarjan and R. Werneck. Self-adjusting top trees. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.
- [6] Umut A. Acar, Guy E. Blelloch, Jorge L.Vittes, Separating Structure from Data in Dynamic Trees.
- [7] C. R. Aragon, R. G. Seidel, Randomized Search Trees, Proc. 30th Ann. *IEEE Symposium on Foundations of Computer Science*, pp. 540-545, October 1989.
- [8] **周源，《Dynamic Connectivity》研究报告，*CTSC2005作业-研究报告*.**



Thank you!!!

Questions are welcome!!!