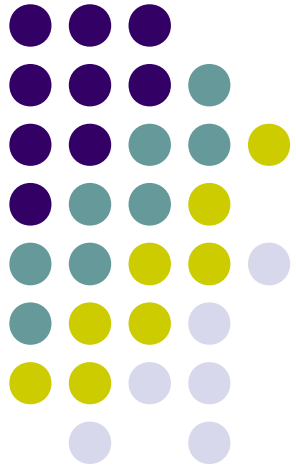
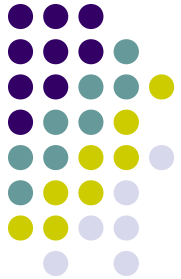
RMQ&LCA 问题

湖南省长郡中学 郭华阳



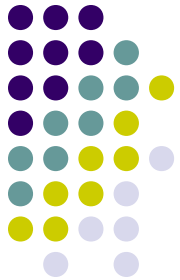
全文总揽

- 问题的提出
- 问题的解决
- 问题的应用



I. 问题的提出

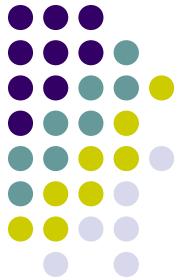
问题的提出



- LCA：基于有根树最近公共祖先问题

- LCA（T，u，v）：在有根树 T 中，询问一个距离根最远的结点 x，使得 x 同时为结点 u、v 的祖先

问题的提出

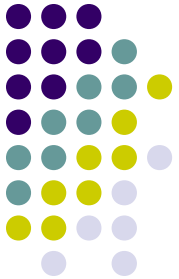


- RMQ：区间最小值询问问题
- RMQ（A，i，j）：对于线性序列 A 中，询问区间 [i，j] 上的最小值
- 特别的，若线性序列 A 任意两相邻元素相差为 ±1，那么建立在 A 上的 RMQ 称为 ±1RMQ

RMQ&LCA 在信息学竞赛中

- 各种竞赛试题中，如上海 2003 年省选的登山、 2002 年 POI 的商务旅行等，都是这类问题的应用及扩展

- 熟练的掌握及解决 RMQ&LCA 问题，对于研究和竞赛都是十分重要的



II. 问题的解决

RMQ&LCA 问题的解决

- 在研究人员的努力下， RMQ&LCA 问题已经获得了比较完善的解决方案，这里我们只列出一些常用算法的适用范围以及他们的时空复杂度：

算法名称	针对问题	时间消耗	空间消耗
ST 算法	一般 RMQ 问题		$O(N\log_2 N)$
Tarjan 算法	LCA 问题		O(N)
±1RMQ 算法	±1RMQ 问题		O(N)

注： N 表示问题规模， Q 表示询问次数

RMQ&LCA 问题的解决

- 我们以 O(f(N)) – O(g(N)) 描述一个在线算法，在 O(f(N)) 的时间内完成预处理，在 O(g(N)) 的时间内完成一次询问
- 以 O(f(N) + g(N)×Q) 描述一个离线算法的时间消耗

算法名称	针对问题	时间消耗	空间消耗
ST 算法	一般 RMQ 问题	$O(N\log_2 N)$-O(1)	$O(N\log_2 N)$
Tarjan 算法	LCA 问题	$O(N\alpha(N) + Q)$	O(N)
±1RMQ 算法	±1RMQ 问题	O(N)-O(1)	O(N)

注： N 表示问题规模， Q 表示询问次数

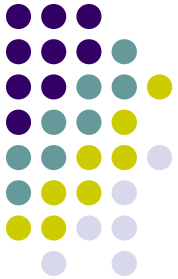
RMQ&LCA 问题的解决

- 这些算法各自具有特点，但是针对问题过于狭窄。下文中我们将会看到， RMQ 与 LCA 问题是可以互相转化的，这就大大扩宽了以下算法的应用面

算法名称	针对问题	时间消耗	空间消耗
ST 算法	一般 RMQ 问题	$O(N\log_2 N)$-$O(1)$	$O(N\log_2 N)$
Tarjan 算法	LCA 问题	$O(N\alpha(N) + Q)$	$O(N)$
±1RMQ 算法	±1RMQ 问题	$O(N)$-$O(1)$	$O(N)$

注： N 表示问题规模， Q 表示询问次数

RMQ 向 LCA 的转化

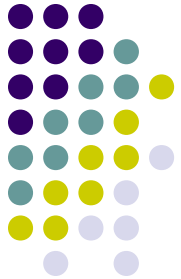


- 考察一个长度为 N 的序列 A，按照如下方法将其递归建立为一棵树：

1) 设序列中最小值为 A_k，建立优先级为 A_k 的根节点 T_k；
2) 将 A(1…k–1) 递归建树作为 T_k 的左子树；
3) 将 A(k+1…N) 递归建树作为 T_k 的右子树；

- 不难发现，这棵树是一棵优先级树

A = (7 5 8 1 10)

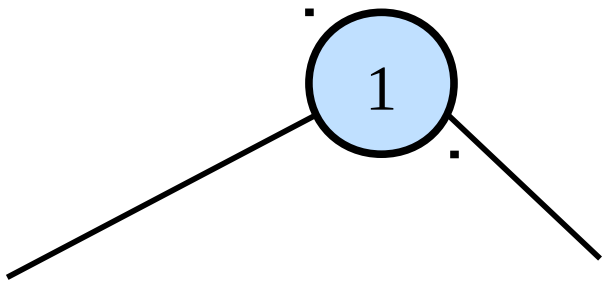


我们以 A 序列为例建立树 T

A = (7 5 8 1 10)

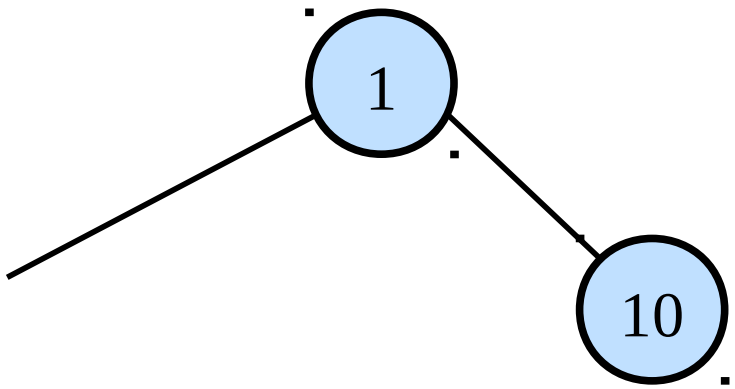
将最小元素 1 作为根节点

左右分别建树



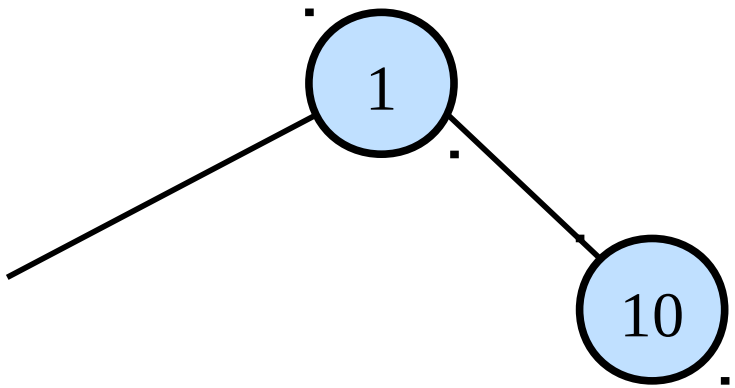
A = (7 5 8 1 10)

右子树只有一个结点，即 10



A = (7 5 8 1 10)

左子树有三个结点 7 、 5 、 8

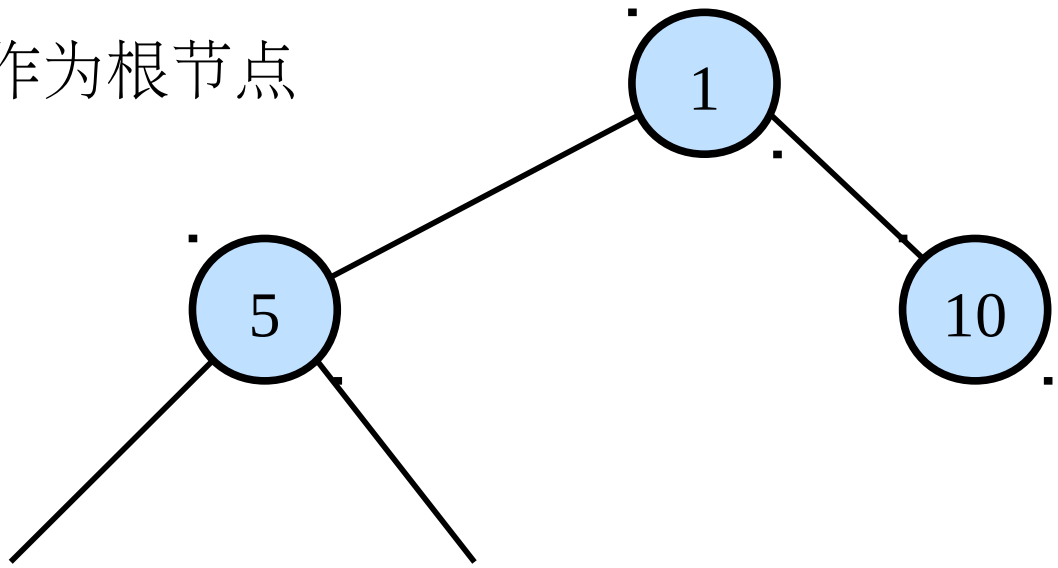


A = (7 5 8 1 10)

左子树有三个结点 7 、 5 、 8

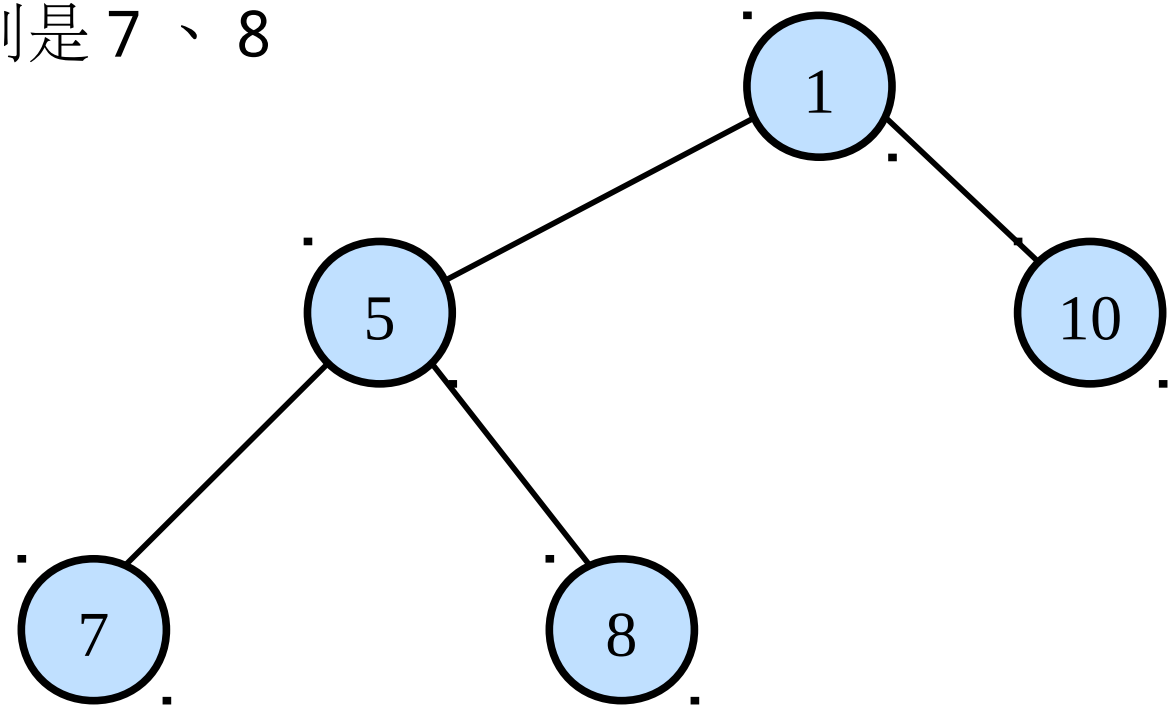
将最小元素 5 作为根节点

左右分别建树



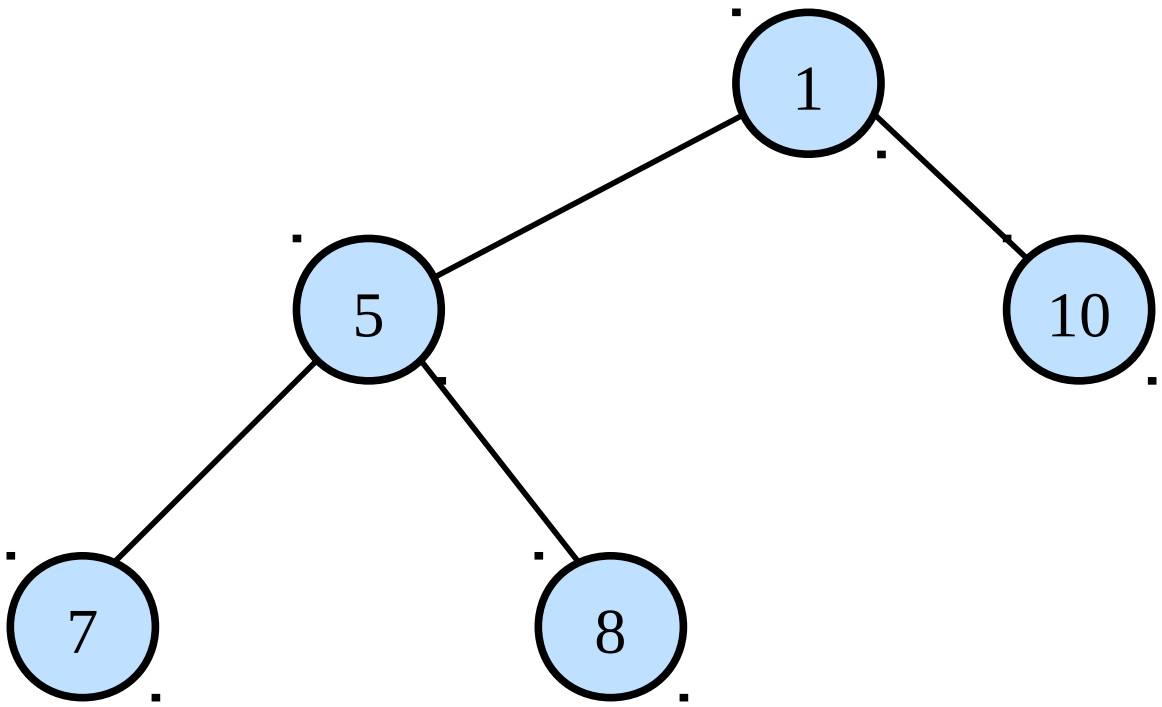
A = (7 5 8 1 10)

元素 5 的左右子树都只有一个结点，分别是 7 、 8

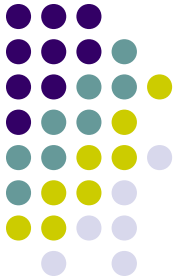


A = (7 5 8 1 10)

这样我们便得到了树 T



RMQ 向 LCA 的转化



- 对于 RMQ(A，i，j)：

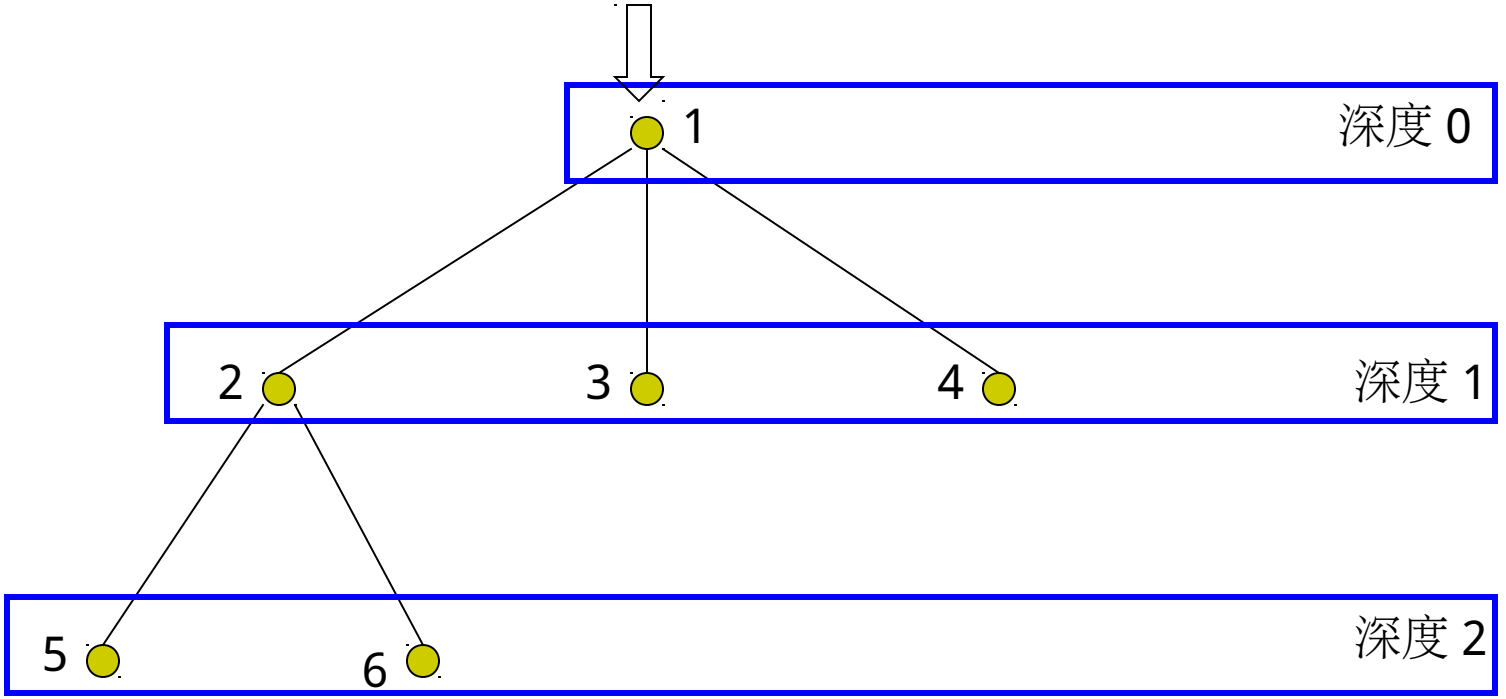
1) 设序列中最小值为 A_k，若 $k \in [i, j]$，那么答案为 k；
2) 若 $k > j$，那么答案为 $RMQ(A_{1..k-1}, i, j)$；
3) 若 $k < i$，那么答案为 $RMQ(A_{K+1..N}, i, j)$；

- 不难发现 RMQ(A，i，j) = LCA(T，i，j)！这就证明了 *RMQ* 问题可以转化为 *LCA* 问题

LCA 向 RMQ 的转化

- 对有根树 T 进行 DFS，将遍历到的结点按照顺序记下，我们将得到一个长度为 2N – 1 的序列，称之为 T 的欧拉序列 F

- 每个结点都在欧拉序列中出现，我们记录结点 u 在欧拉序列中第一次出现的位置为 pos(u)



欧拉序列 F： 1 2 5 2 6 2 1 3 1 4 1

深度序列 B： 0 1 2 1 2 1 0 1 0 1 0

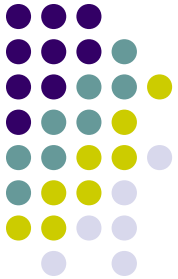
Pos (u)： 1 2 8 10 3 5

LCA 向 RMQ 的转化

- 根据 DFS 的性质，对于两结点 u 、 v ， 从 pos(u) 遍历到 pos(v) 的过程中经过 LCA(u, v) 有且仅有一次，且深度是深度序列 B[pos(u)…pos(v)] 中最小的

- 即 LCA(T, u, v) = RMQ(B, pos(u), pos(v)) ， 并且问题规模仍然是 O(N) 的

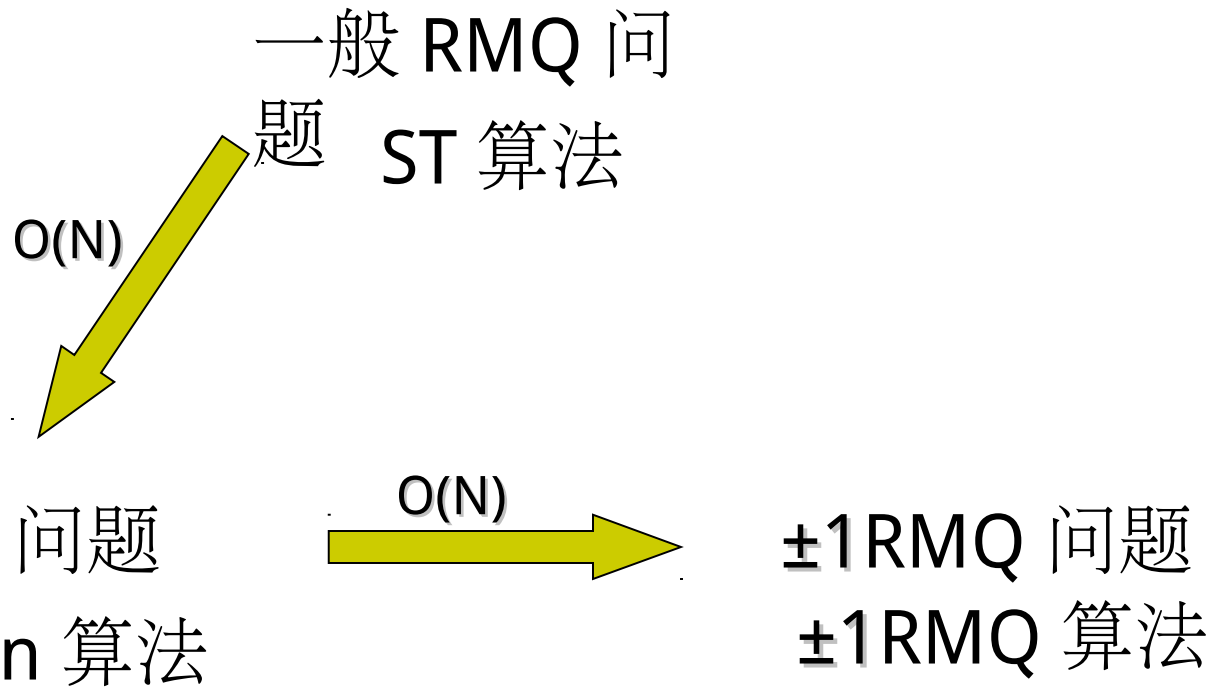
LCA 向 RMQ 的转化

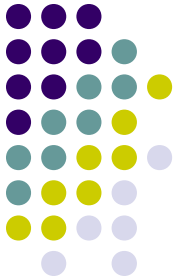


- 根据 DFS 的性质，对于两结点 u 、 v ，从 pos(u) 遍历到 pos(v) 的过程中经过 LCA(u, v) 有且仅有一次，且深度是深度序列 B[pos(u)…pos(v)] 中最小的

- 这就证明了 *LCA* 问题可以转化成 *RMQ* 问题
- *LCA* 与 *RMQ* 问题可以互相转化，并且可以在 *O(N)* 的时间内完成！

RMQ&LCA 算法关系图





III. 问题的应用

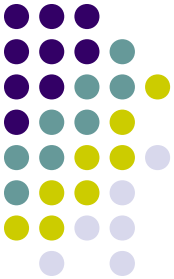
问题的应用

- RMQ&LCA 问题在算法研究及信息学竞赛中都发挥着十分重要的作用，它主要是以一种经典算法及解题思想的形式出现

- 本文主要以讨论一道例题：

  水管局长（2006 年冬令营试题）

水管局长（原题）



- SC 省 MY 市有着庞大的地下水管网络，嘟嘟是 MY 市的水管局长（就是管水管的啦），嘟嘟作为水管局长的工作就是：每天供水公司可能要将一定量的水从 x 处送往 y 处，嘟嘟需要为供水公司找到一条从 A 至 B 的水管的路径，接着通过信息化的控制中心通知路径上的水管进入准备送水状态，等到路径上每一条水管都准备好了，供水公司就可以开始送水了。嘟嘟一次只能处理一项送水任务，等到当前的送水任务完成了，才能处理下一项。
- 在 处理每项送水任务之前，路径上的水管都要进行一系列的准备操作，如清洗、消毒等等。嘟嘟在控制中心一声令下，这些水管的准备操作同时开始，但由于各条管道 的长度、内径不同，进行准备操作需要的时间可能不同。供水公司总是希望嘟嘟能找到这样一条送水路径，路径上的所有管道全都准备就绪所需要的时间尽量短。嘟 嘟希望你能帮助他完成这样的一个选择路径的系统，以满足供水公司的要求。另外，由于 MY 市的水管年代久远，一些水管会不时出现故障导致不能使用，你的程序必须考虑到这一点。
- 不妨将 MY 市的水管网络看作一幅简单无向图（即没有自环或重边）：水管是图中的边，水管的连接处为图中的结点。

题目模型简述

- 定义：
  - 一条路径的关键边为其中费用最大的边；
  - 一条路径的花费为其关键边的花费；
  - 两点之间的最短路为所有连接两点的路径中花费最小的一条；
- 给定带权无向图 G 及在 G 上的 Q 个操作形如：
  - 拆除无向边 (u, v) ；
  - 询问 u 、 v 之间的最短路花费；

水管局长

- 你需要写一个程序完成给出的操作

- 数据范围:

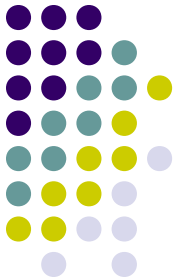
  结点数　　N ≤ 1000 ；

  边数　　M ≤ 100000 ；

  操作数　　Q ≤ 100000 ；

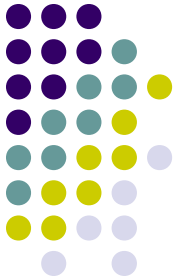
  删边操作　D ≤ 5000 ；

水管局长



- 周戈林同学在 2006 年冬令营中已经讨论过相关的问题，并提出了解决的类普里姆算法
- 类普里姆算法解决一次询问需要 $O(N^2)$ 的时间
- 注意到询问数可能达到 10^5，类普里姆算法不能解决本题，为什么时间复杂度如此之高呢?

水管局长



- 分析一下本题时间复杂度瓶颈所在：
  - 边数过多；
  - 完成询问的复杂度过高；

- 我们将在后文逐个把这些问题解决

引入最小生成森林

- 引理一:

  任意询问可以在 G 的最小生成森林中找到最优解

- 证明:

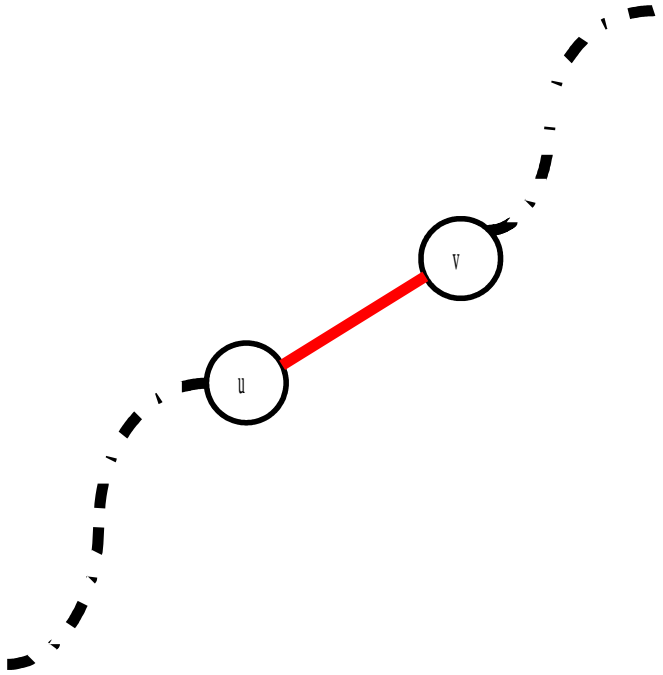
  记 Φ(P) 表示路径 P 中不在 T 中的边数

1) 若 Φ(P) = 0，则 P 在 T 上;
2) 若 Φ(P) > 0，则存在一条边 (u, v)∈P – T

引入最小生成森林

- 引理一:
  任意询问可以在 G 的最小生成森林中找到最优解

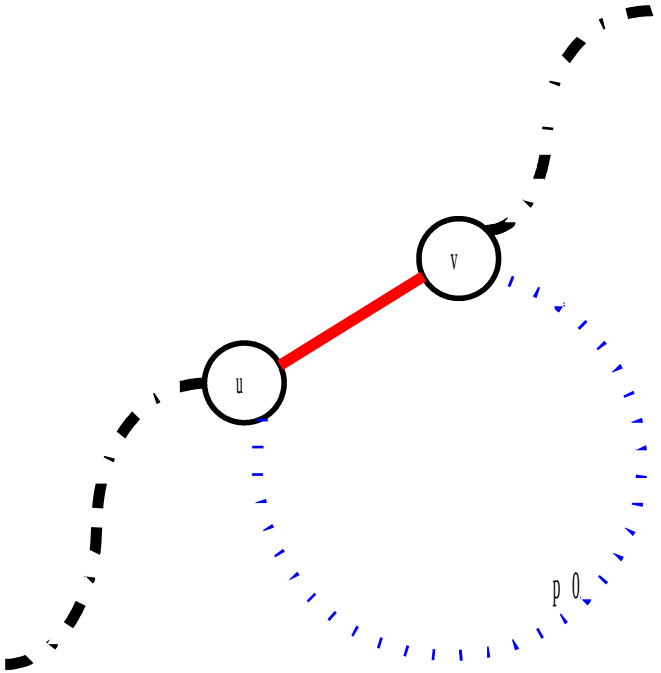
- 证明（续）:
  考察这条边 (u, v) :



引入最小生成森林

- 引理一：
  任意询问可以在 G 的最小生成森林中找到最优解

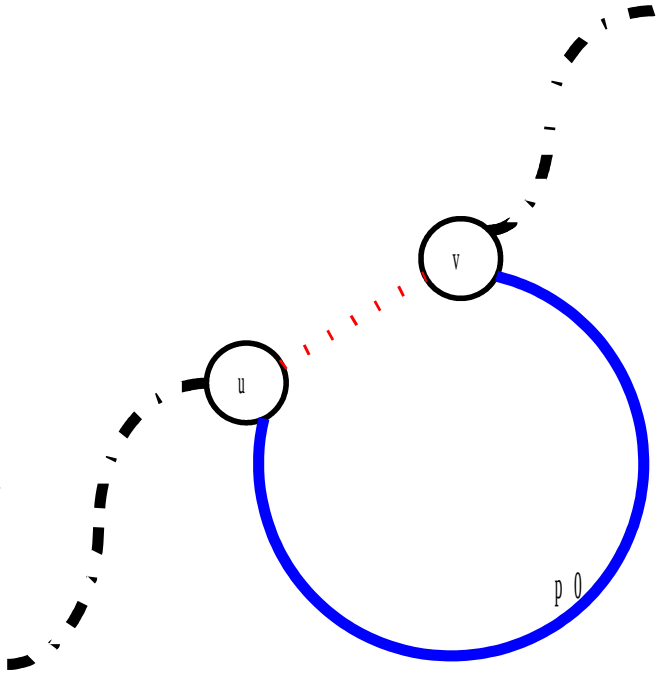
- 证明（续）：
  考察这条边 (u, v)：
  - 根据最小生成森林的性质，存在一条只有树边构成的路径 P_0 连接 u 、 v 两点，并且 P_0 的花费不大于 (u, v) 的花费



引入最小生成森林

- 引理一：
  任意询问可以在 G 的最小生成森林中找到最优解

- 证明（续）：
  考察这条边 (u, v)：
  - 根据最小生成森林的性质，存在一条只有树边构成的路径 P_0 连接 u 、 v 两点，并且 P_0 的花费不大于 (u, v) 的花费
  - 将 P 中 (u, v) 替换为路径 P0 ， P 的总花费不增且 Φ(P) 减小



引入最小生成森林

- 引理一:

  任意询问可以在 G 的最小生成森林中找到最优解

- 证明（续）:

  由于 Φ(P) ≥ 0，所以可以在有限次替换后将 P 变成一条 T 中的路径

  这就证明了该引理

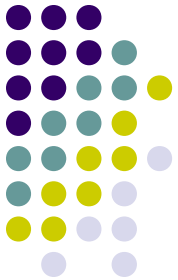
引入最小生成森林

- 引理一:

  任意询问可以在 G 的最小生成森林中找到最优解

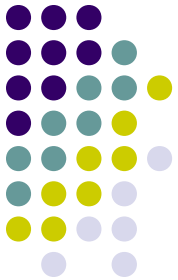
- 根据引理，我们只需要保存所有树边即可，这样边数下降到 O(N) 级别，第一个问题被解决。

完成询问



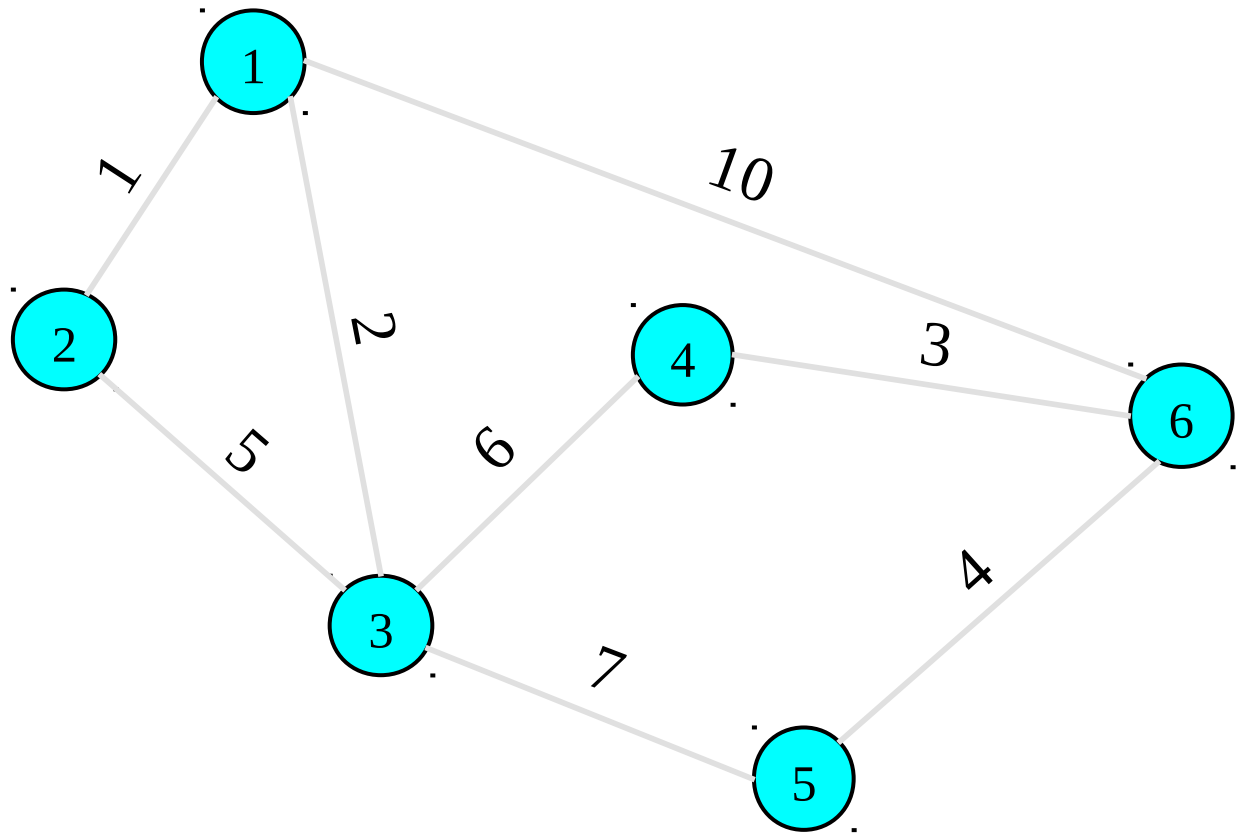
- 我们来考虑第二个问题
- 注意到最小生成森林中任意两点之间的路径是唯一的。我们可以采用 DFS 找出该路径中的关键边，时间消耗为 O(N)
- 考虑到 N <= 1000 、 Q <= 10^5，这样的时间复杂度仍然不能很好解决本题

深入思考

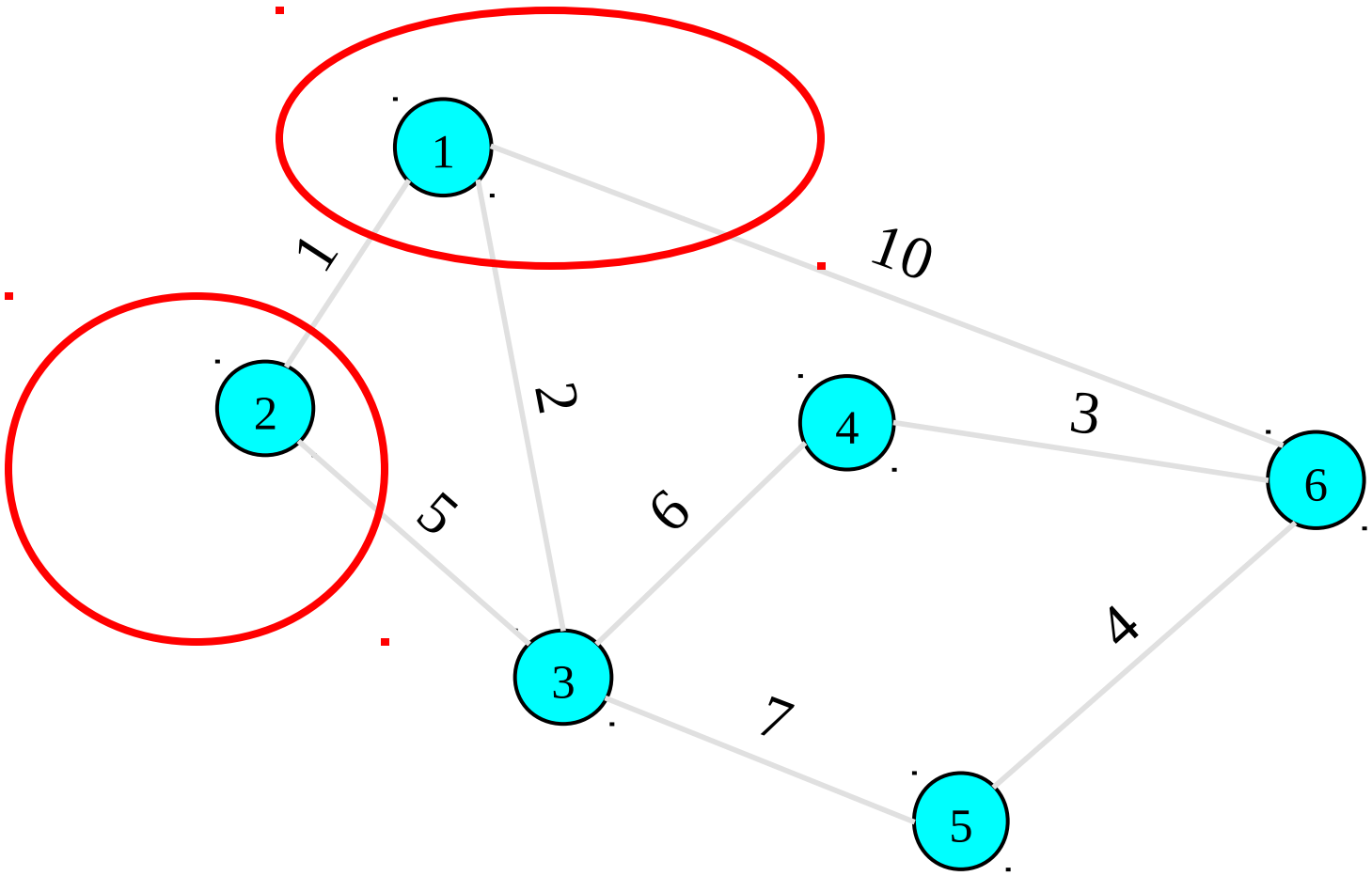


- 询问树上两点之间路径上的最大边，这种问题可以使用动态树等工具实现，但是都过于复杂。

- 为了找出一个简单、实用的算法，我们需要仔细的研究最小生成森林的性质
  - Kruskal 算法是建立最小生成森林中为我们所熟知的算法，以下我们将模拟一次 Kruskal 算法的执行

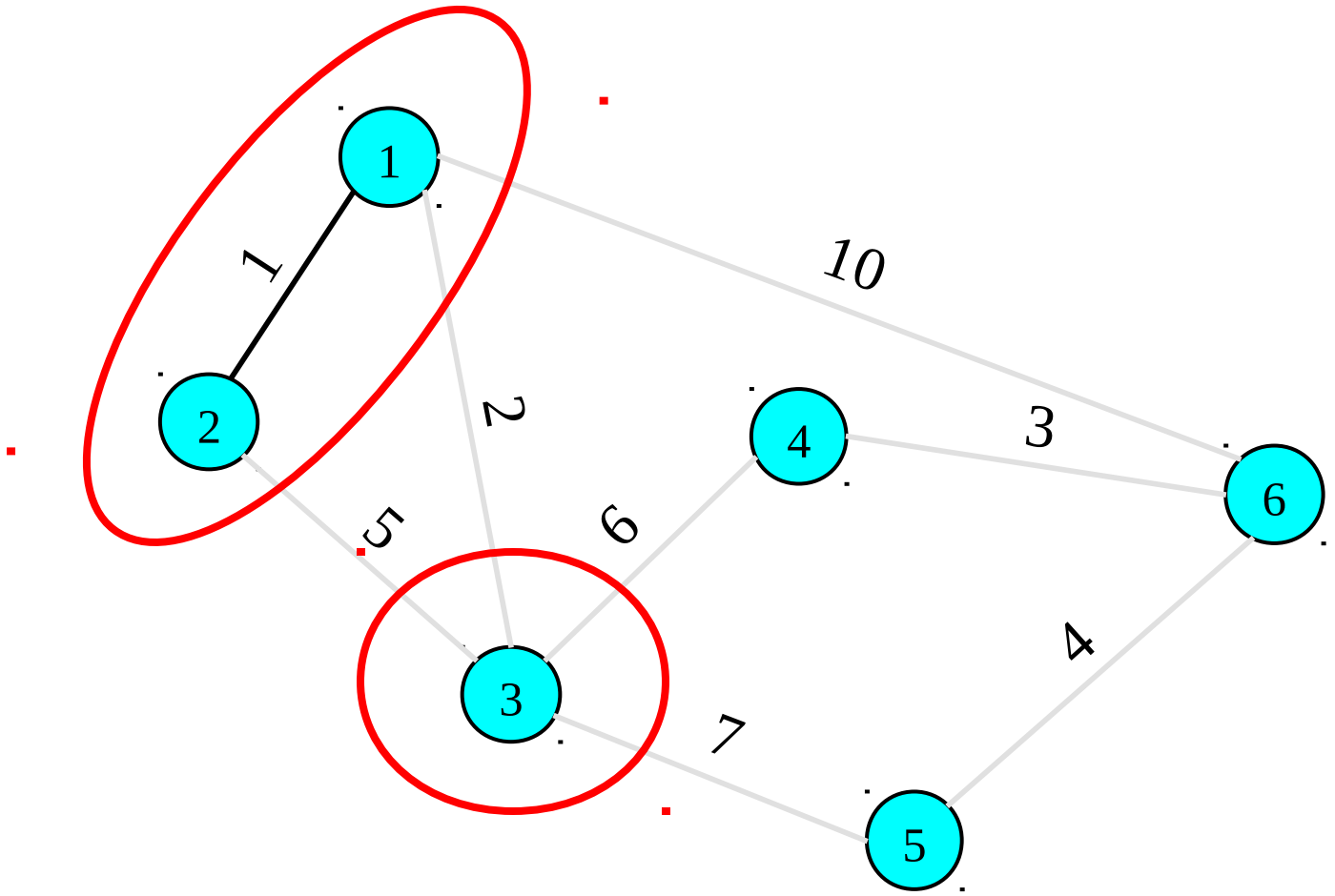


我们使用 Kruskal 算法生成上图的最小生成森林，将所有边按照权值从小到大加入



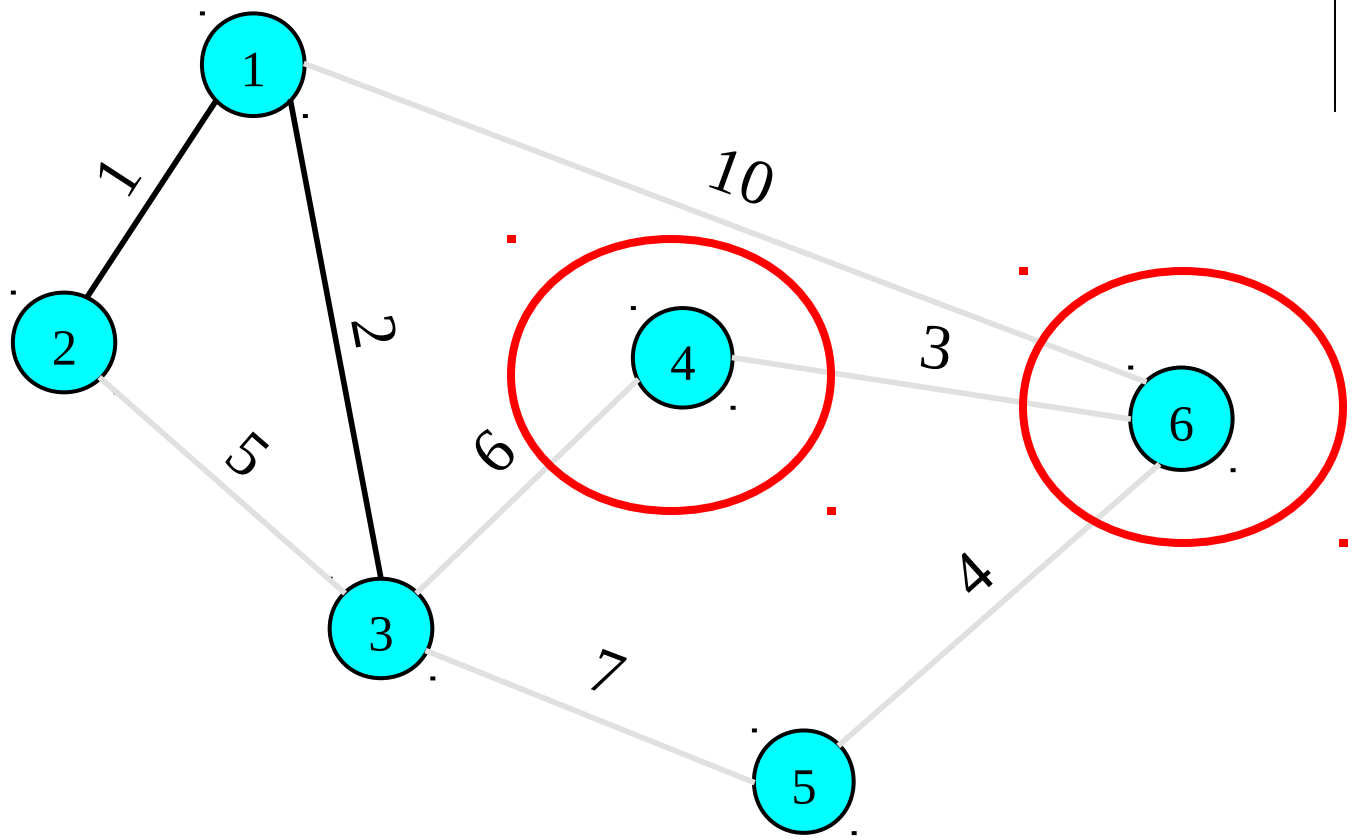
加入边 (1, 2) 为树边

注意到 Query(1, 2) 的关键边为 (1, 2)



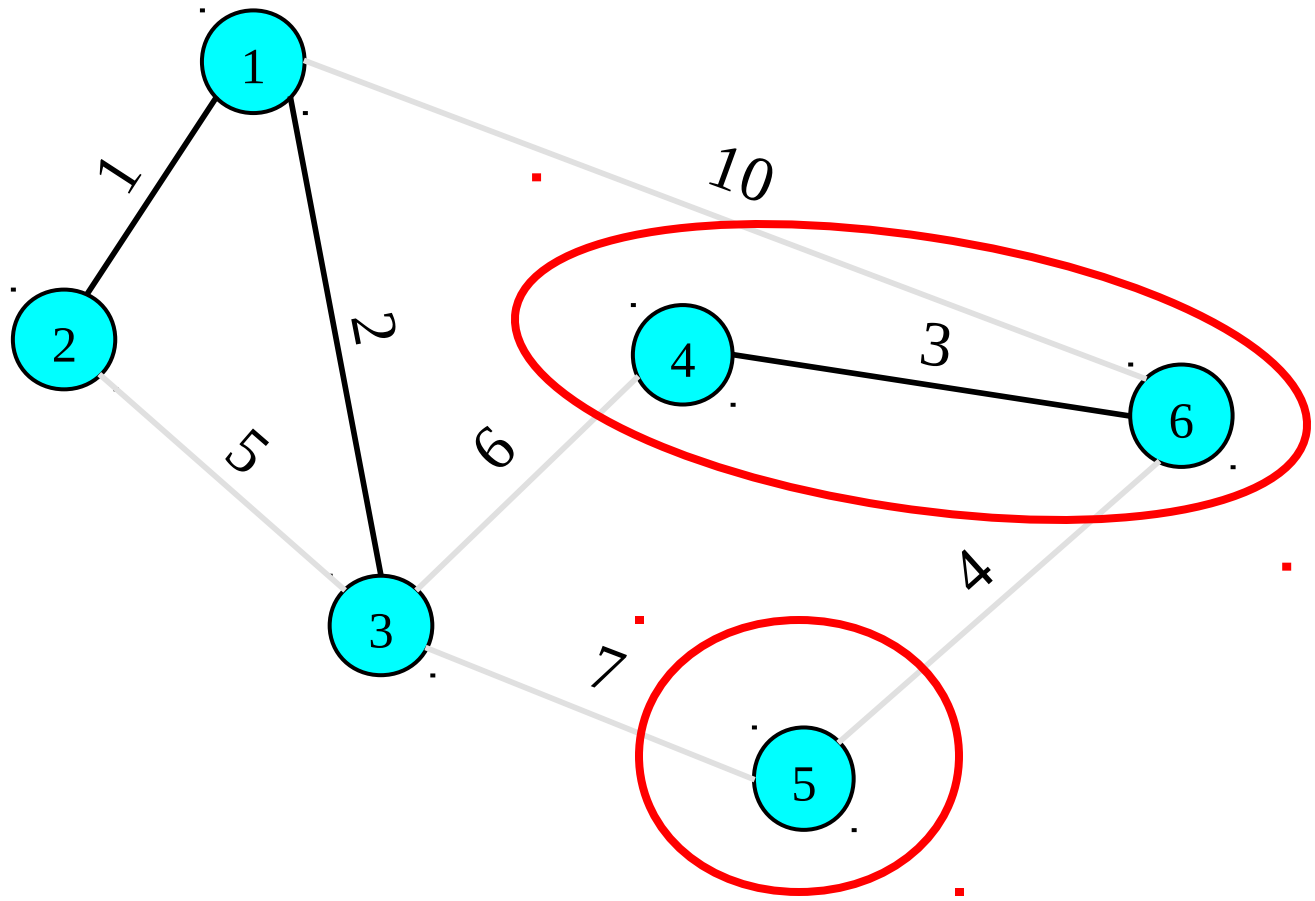
加入边 (1, 3) 为树边

注意到 Query(1|2, 3) 的关键边为 (1, 3)



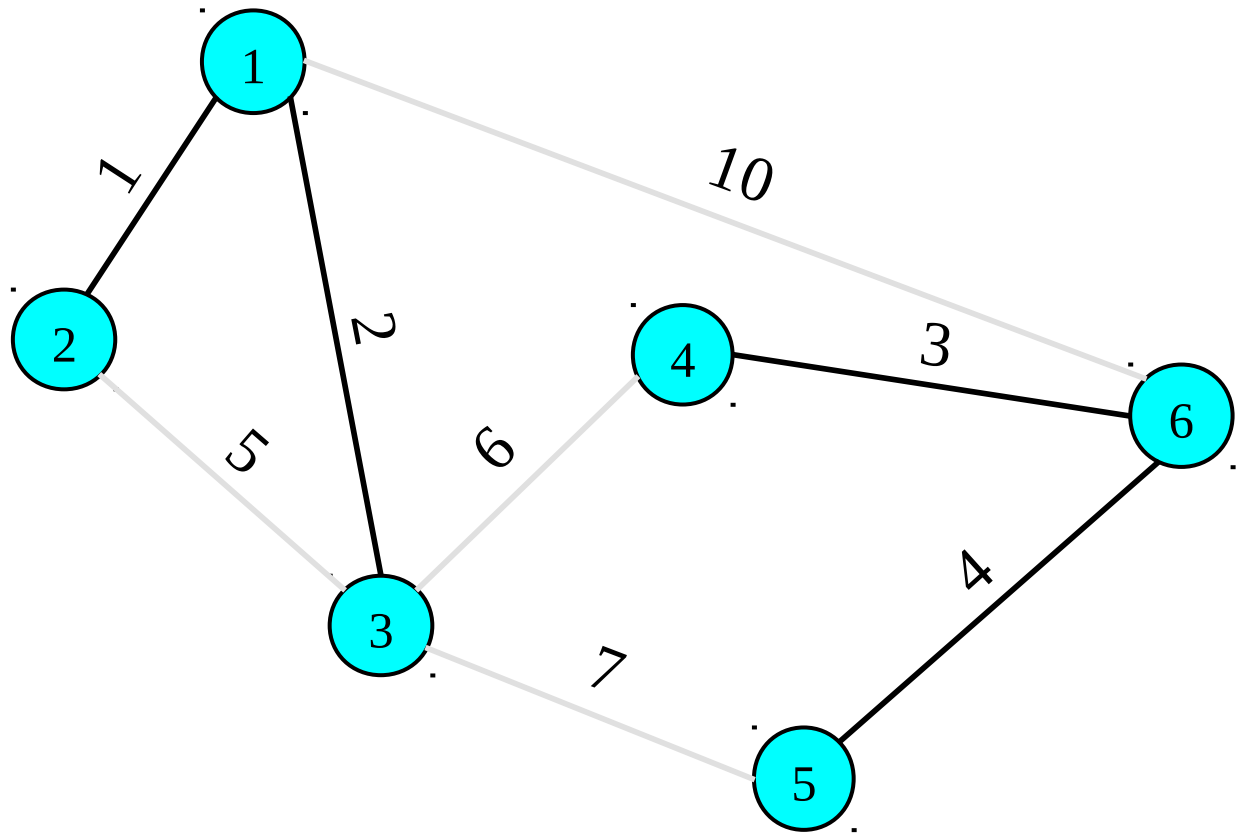
加入边 (4, 6) 为树边

注意到 Query(4, 6) 的关键边为 (4, 6)



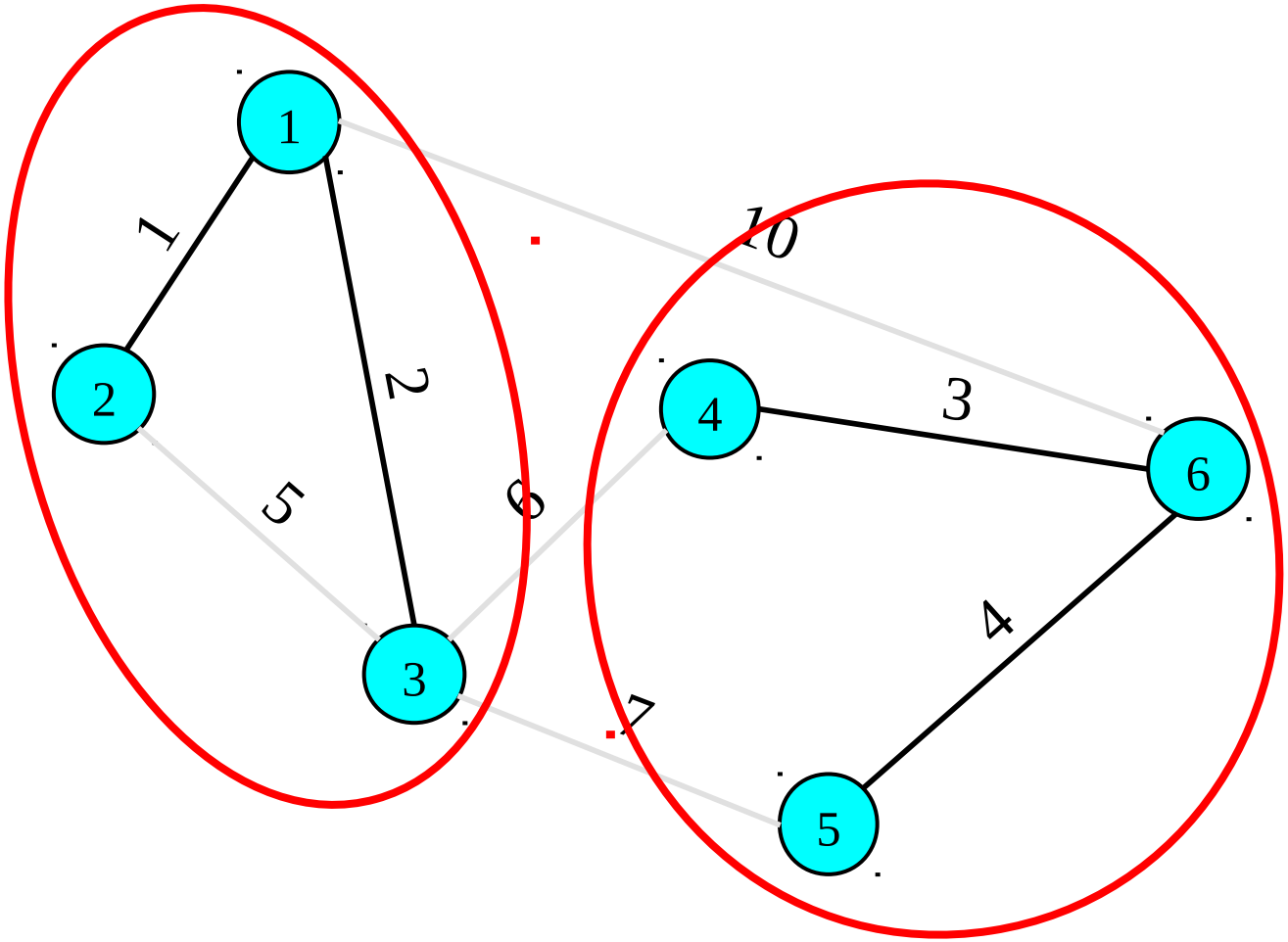
加入边 (5, 6) 为树边

注意到 Query(4|6, 5) 的关键边为 (5, 6)



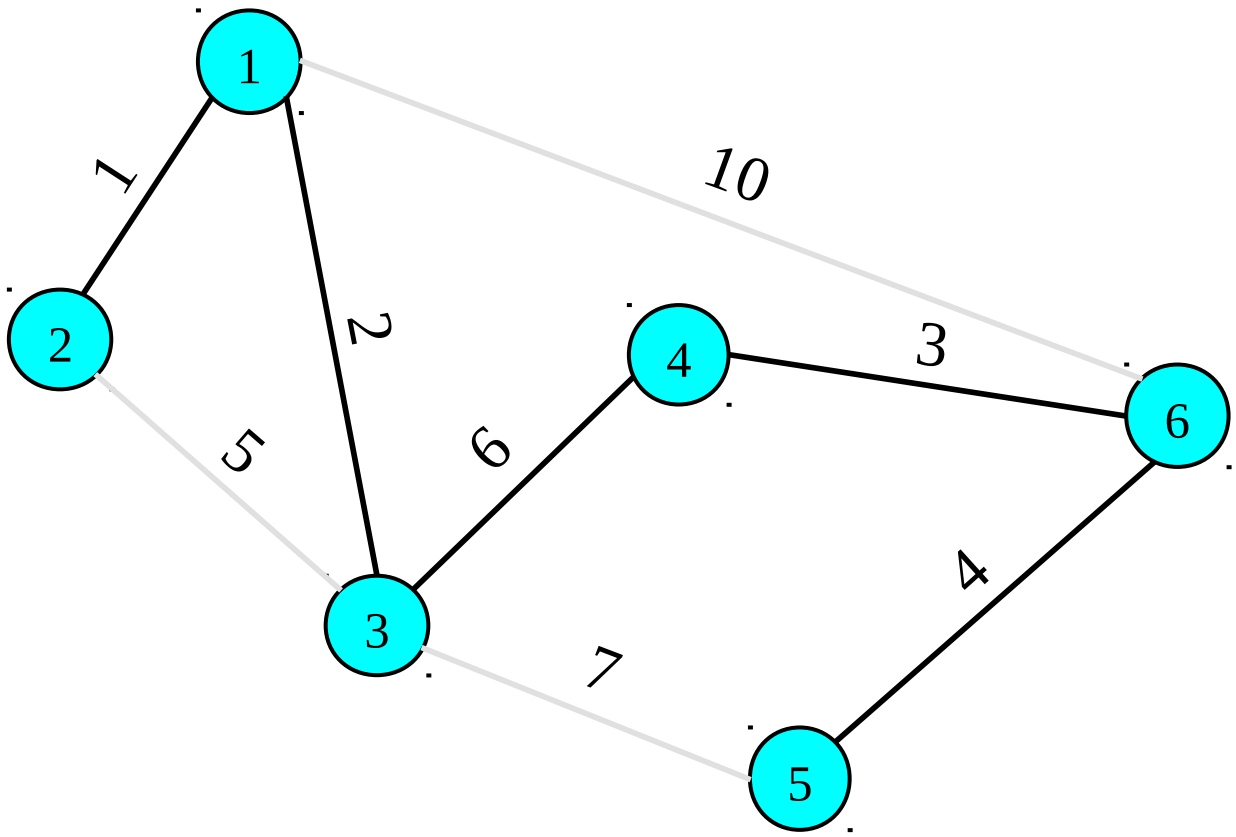
加入边 (2, 3)

注意到已存在路径 (2, 1) – (1, 3)，所以 (2, 3) 非树边



加入边 (3, 4) 为树边

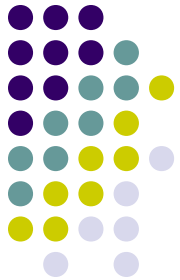
注意到 Query(1|2|3,4|5|6) 的关键边为 (3, 4)



此时，我们已经的到了最小生成森林 T

更重要的是，我们也已经得到了所有询问的回答！

重要引理的发现

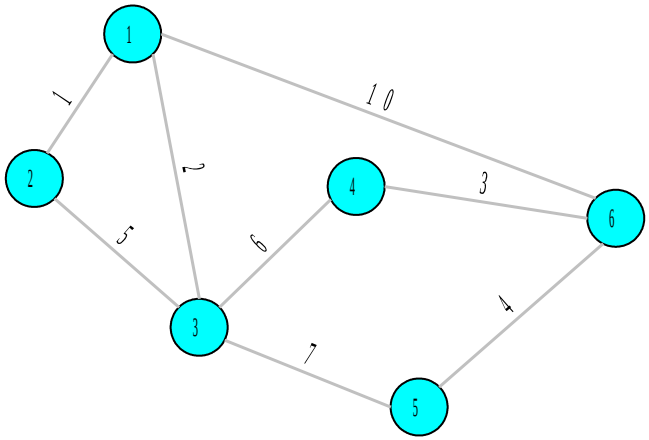


- 对 Kruskal 过程仔细思考，我们得出关键：

- 引理二：
- 任意两点 u 、 v 间最短路径的关键边，为执行 Kruskal 算法中第一次将此两点连通的树边！

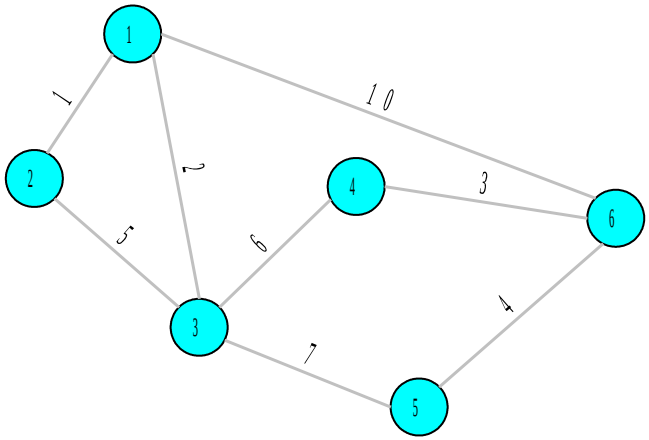
Kruskal 生成顺序森林

- 如何适当的应用引理二呢？所有的树边和结点需要被有机的结合起来，这里我们使用 Kruskal 生成顺序森林（简称顺序森林）
- 仍然考虑下图：



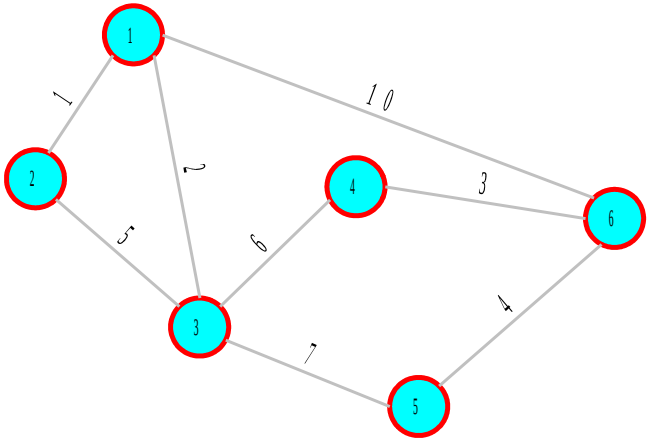
Kruskal 生成顺序森林

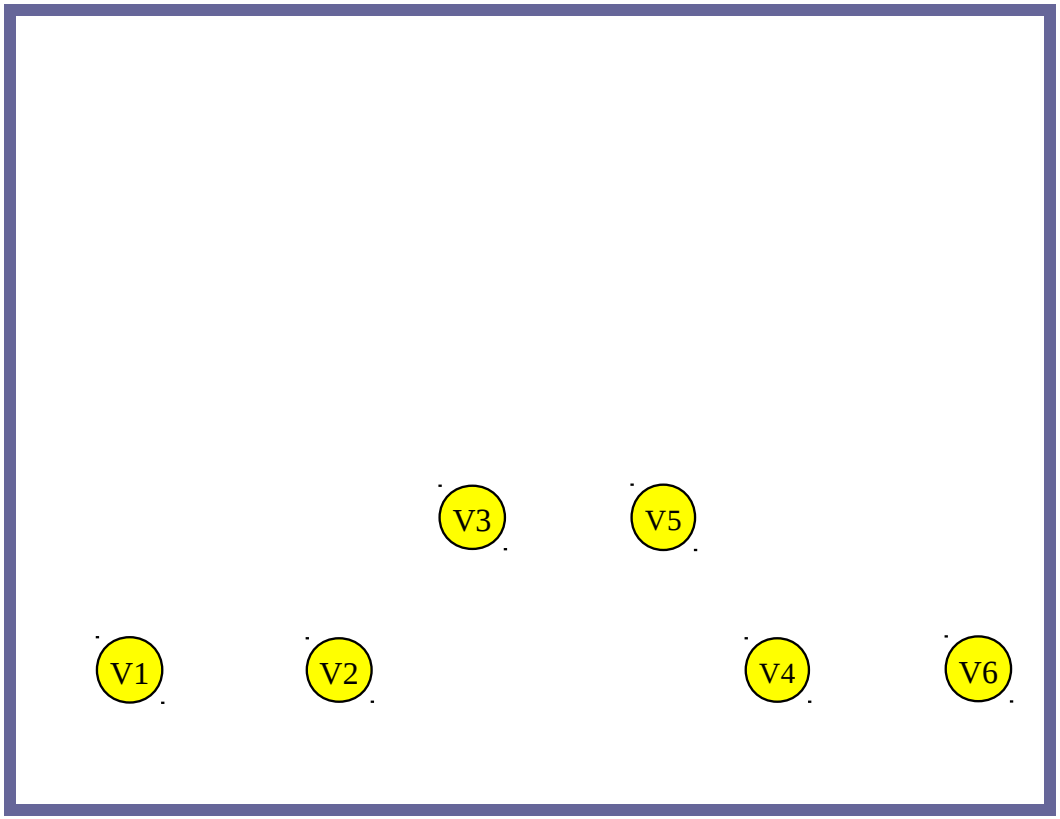
- 顺序森林的每个叶结点对应原图的一个结点，如下图所示，叶结点 V_i 对应原图的结点 i



Kruskal 生成顺序森林

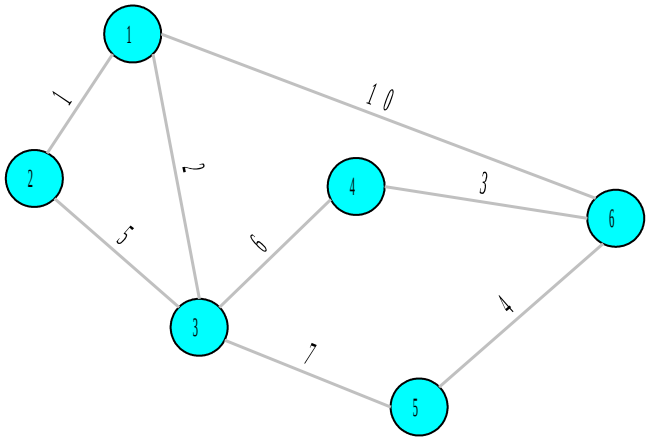
- 顺序森林的每个叶结点对应原图的一个结点，如下图所示，叶结点 V_i 对应原图的结点 i

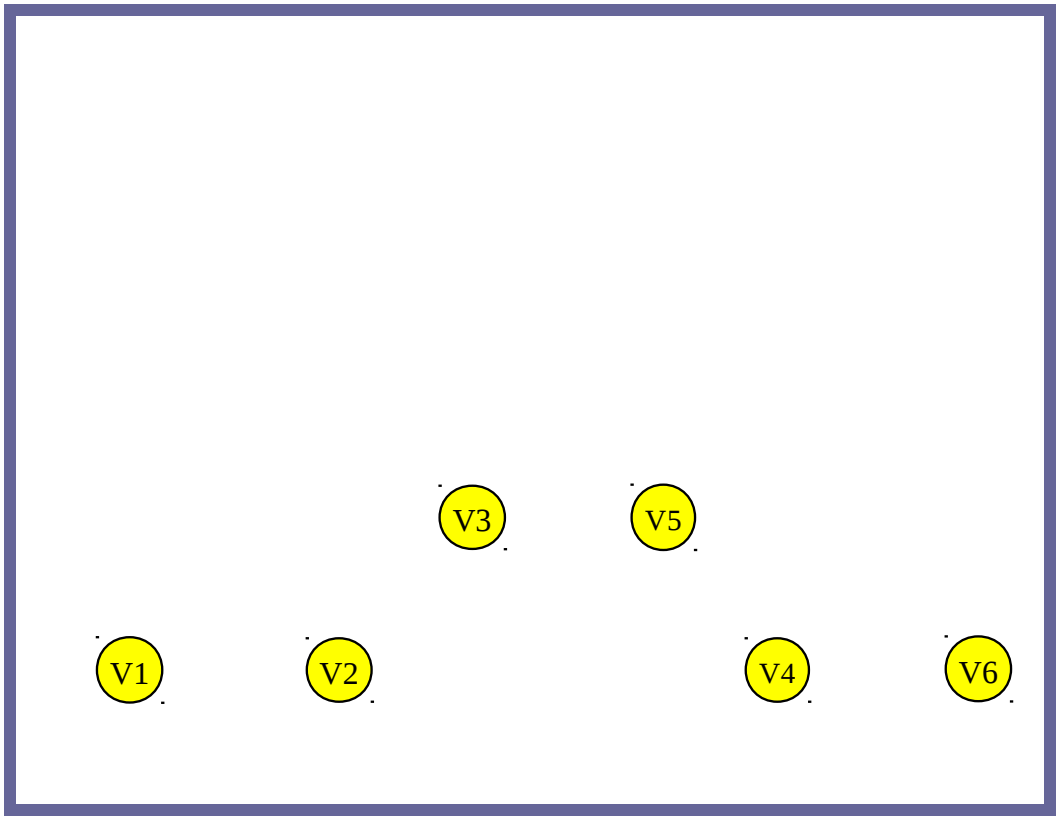




Kruskal 生成顺序森林

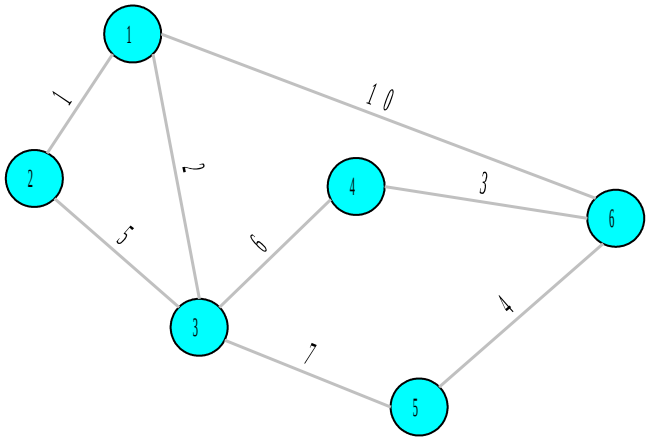
- 树边 E_i 被加入时，该边所连接的两个连通块在顺序森林中必然是两棵树

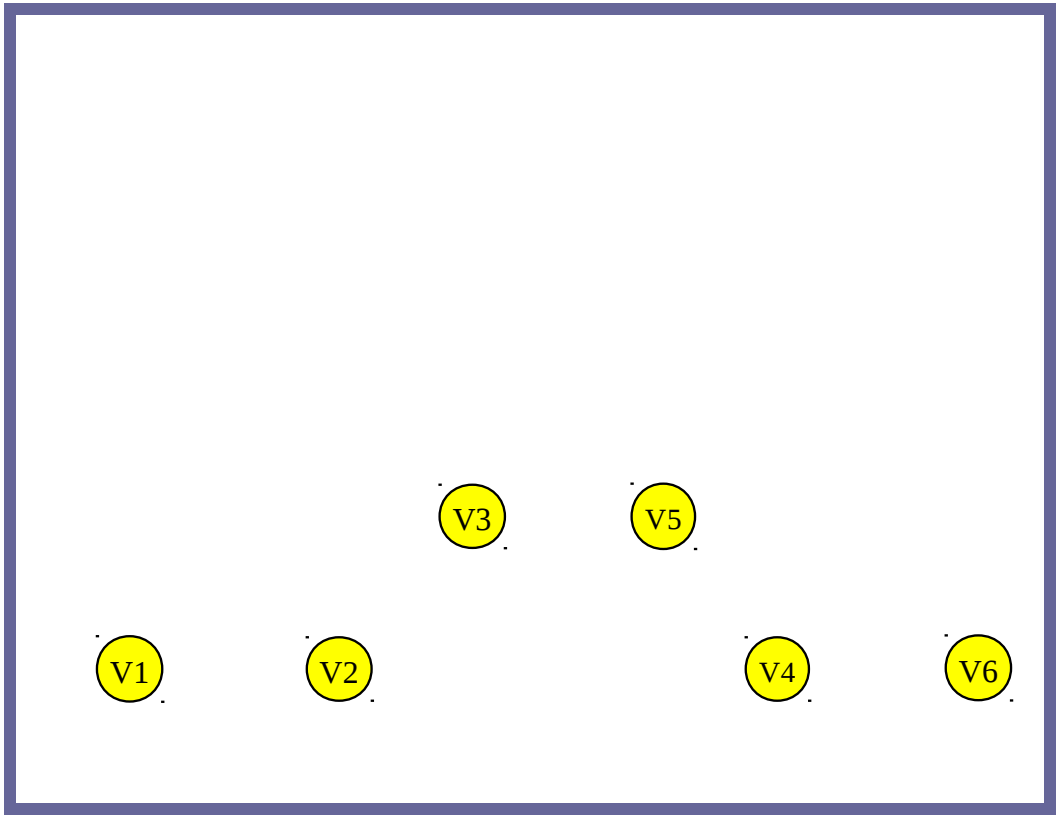




Kruskal 生成顺序森林

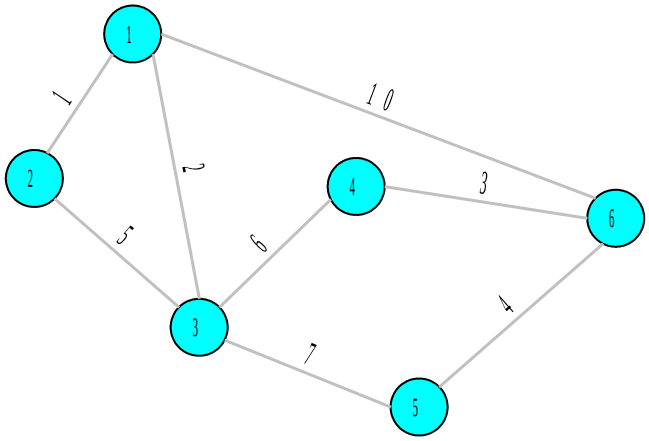
- 建立顺序森林结点与 Ei 相对应，其左右孩子分别为两个连通块对应树的根结点

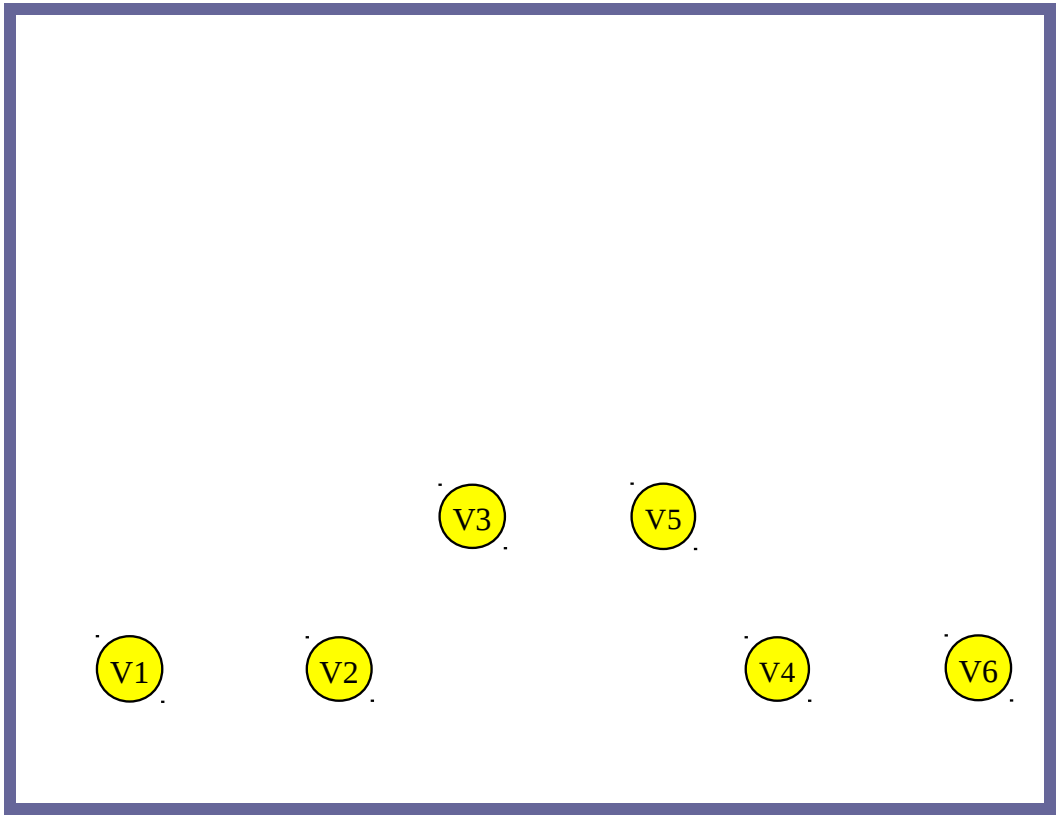




Kruskal 生成顺序森林

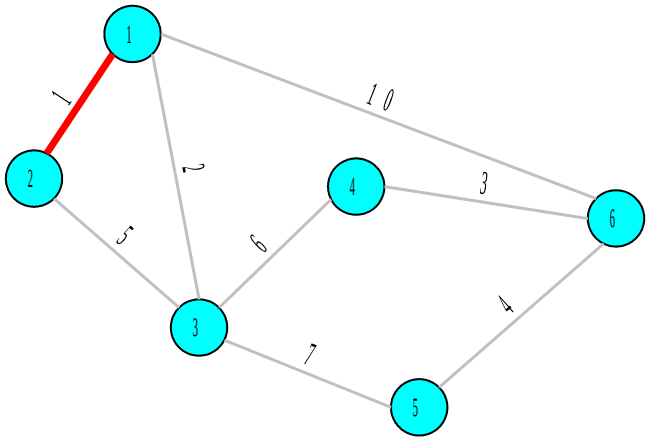
- 我们模拟将所有的树边依次加入：

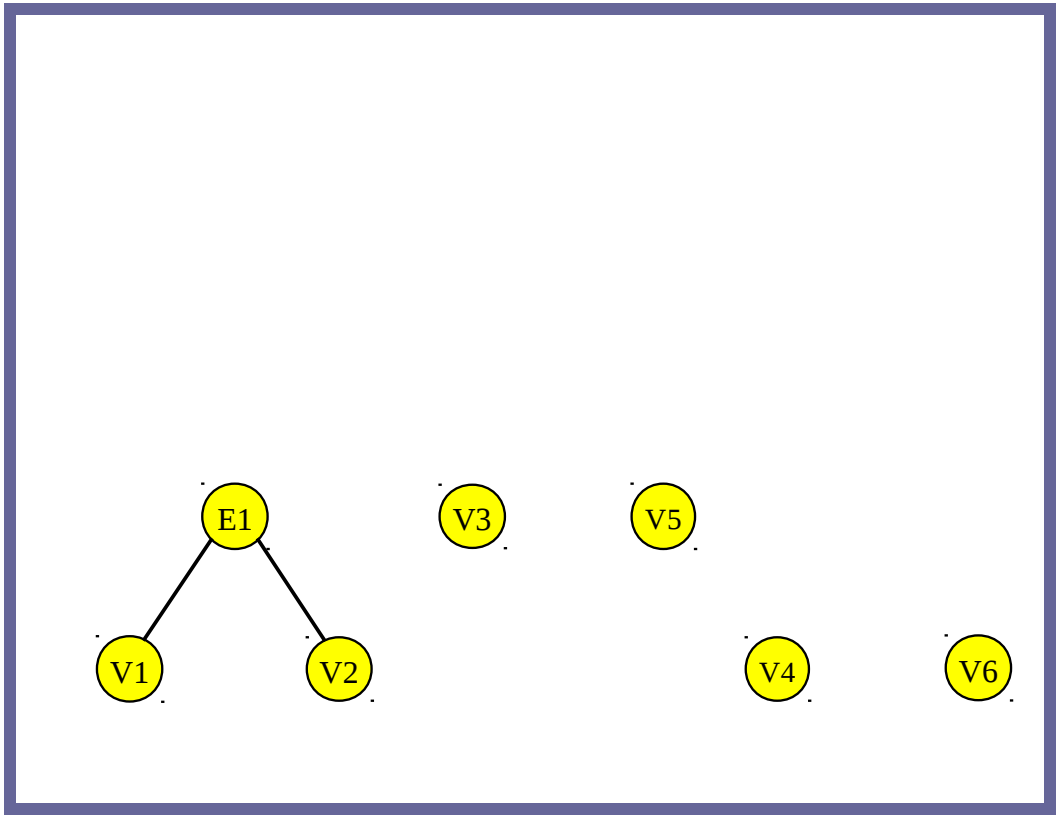




Kruskal 生成顺序森林

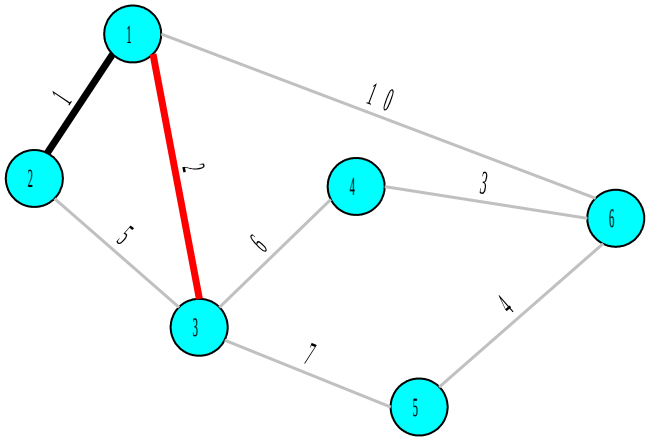
- 添加树边 (1, 2)，将原图结点 1 、 2 合并为一个连通块；我们建立顺序森林结点 E_1，两个儿子为 V_1 、 V_2

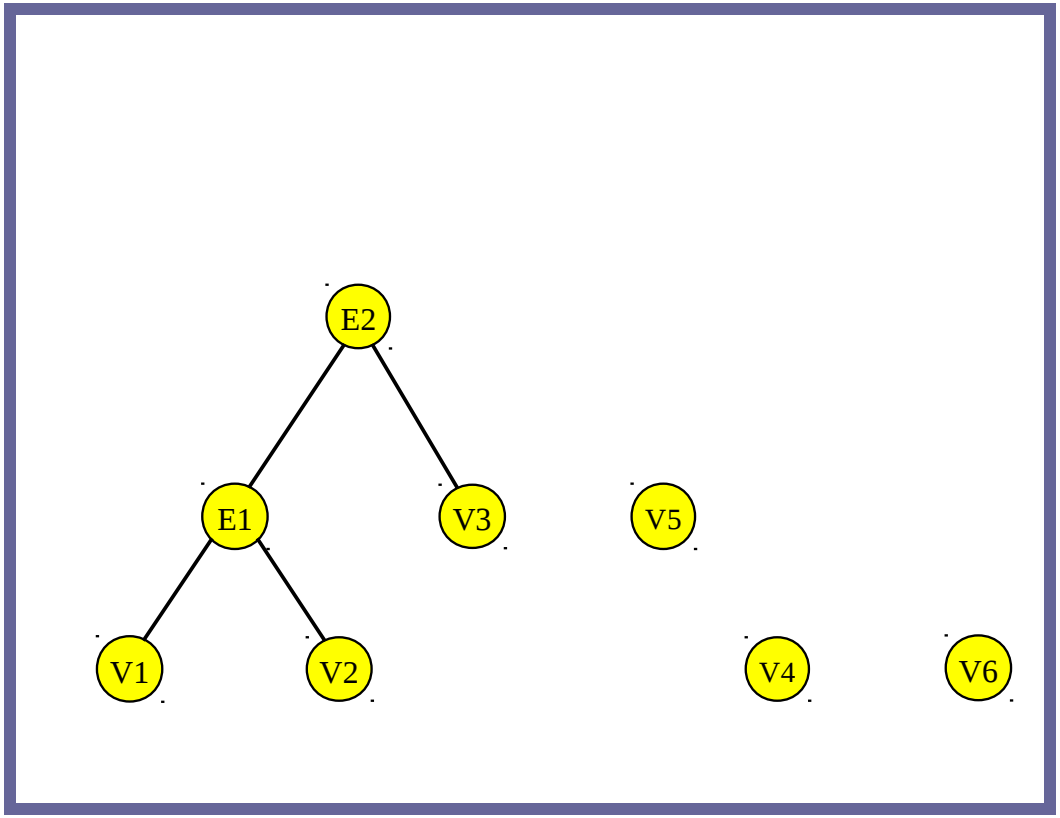




Kruskal 生成顺序森林

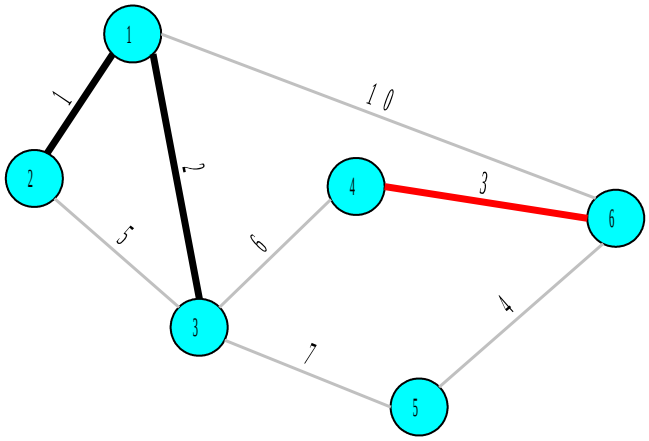
- 添加树边 (1, 3)，将原图结点 1 、 2 、 3 合并为一个连通块；我们建立顺序森林结点 E_2，两个儿子为 E_1 、 V_3

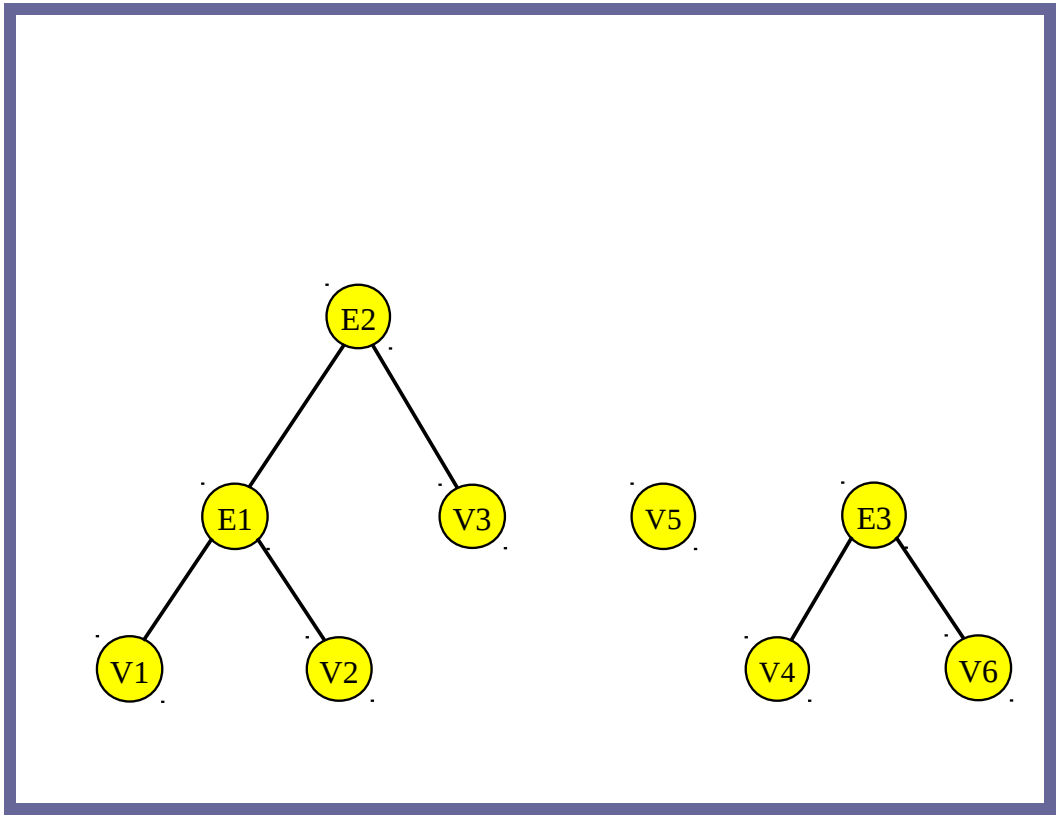




Kruskal 生成顺序森林

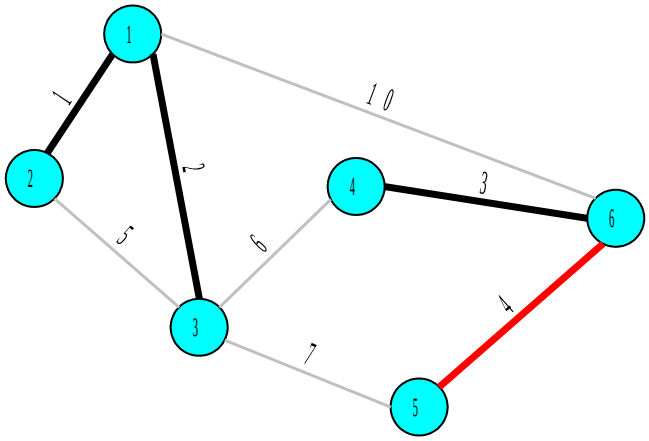
- 类似的添加树边 (4, 6) 时，建立顺序森林结点 E_3，两儿子为 V_4、V_6

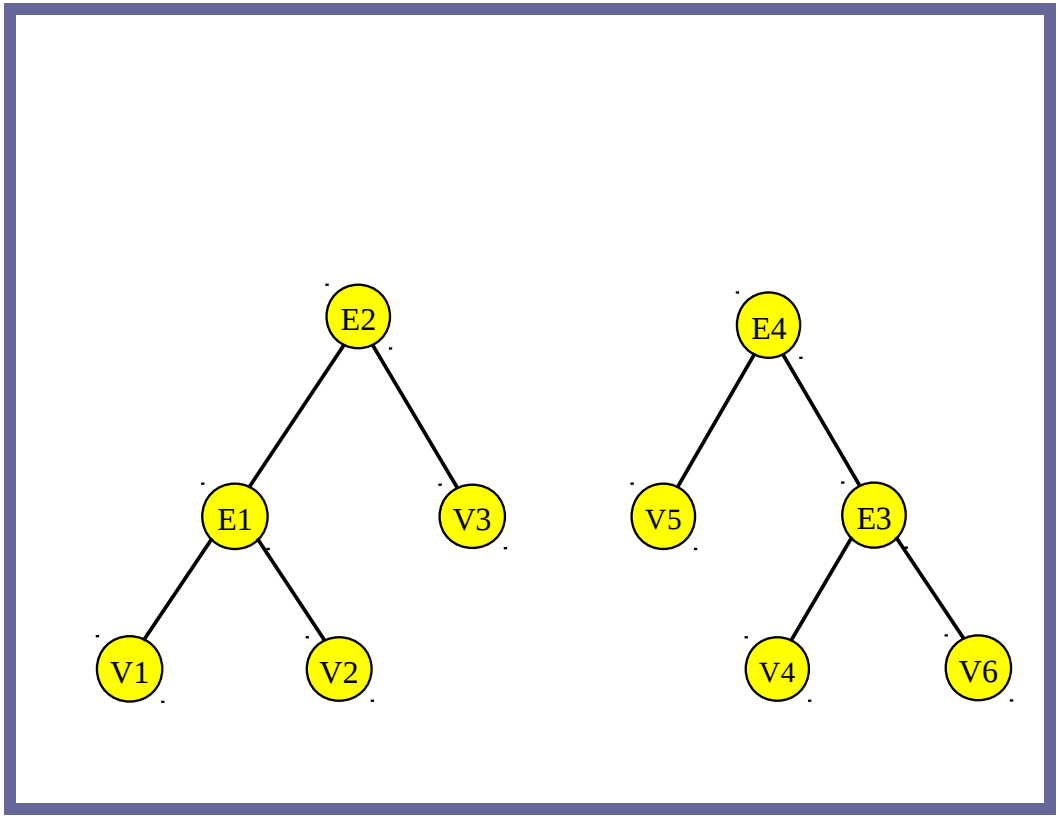




Kruskal 生成顺序森林

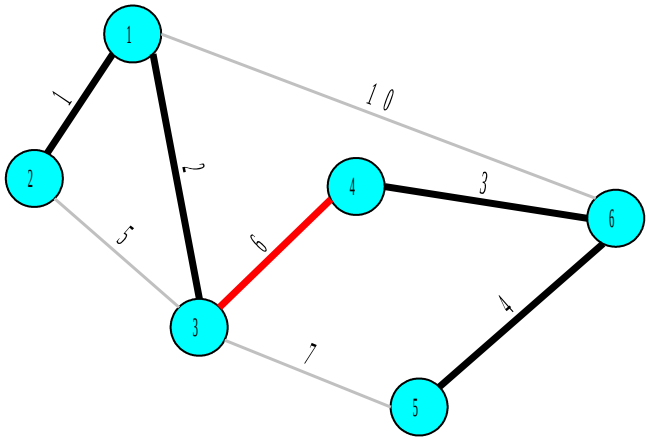
- 添加树边 (5, 6) 时，建立顺序森林结点 E_4，两儿子为 V_5 、 E_3

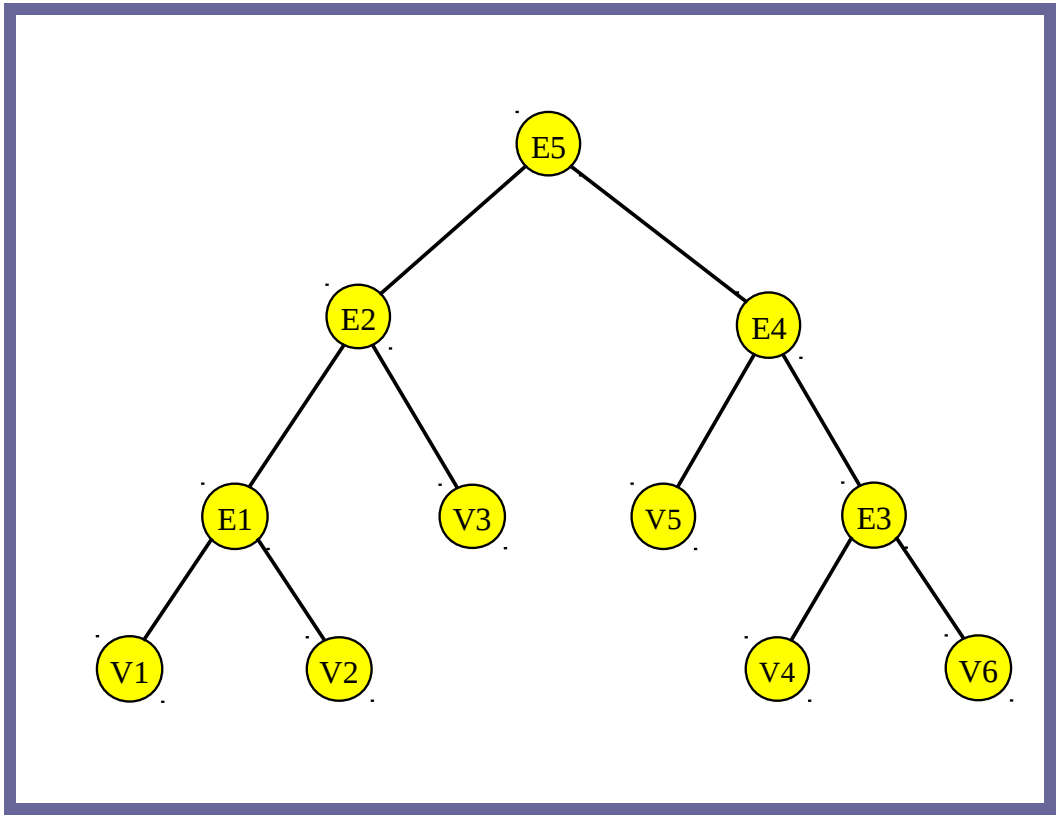




Kruskal 生成顺序森林

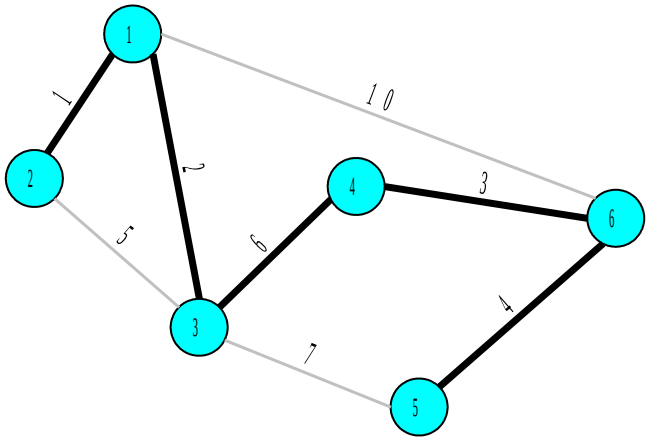
- 添加树边 (3, 4) 时，建立顺序森林结点 E_5，两儿子为 E_2、E_4

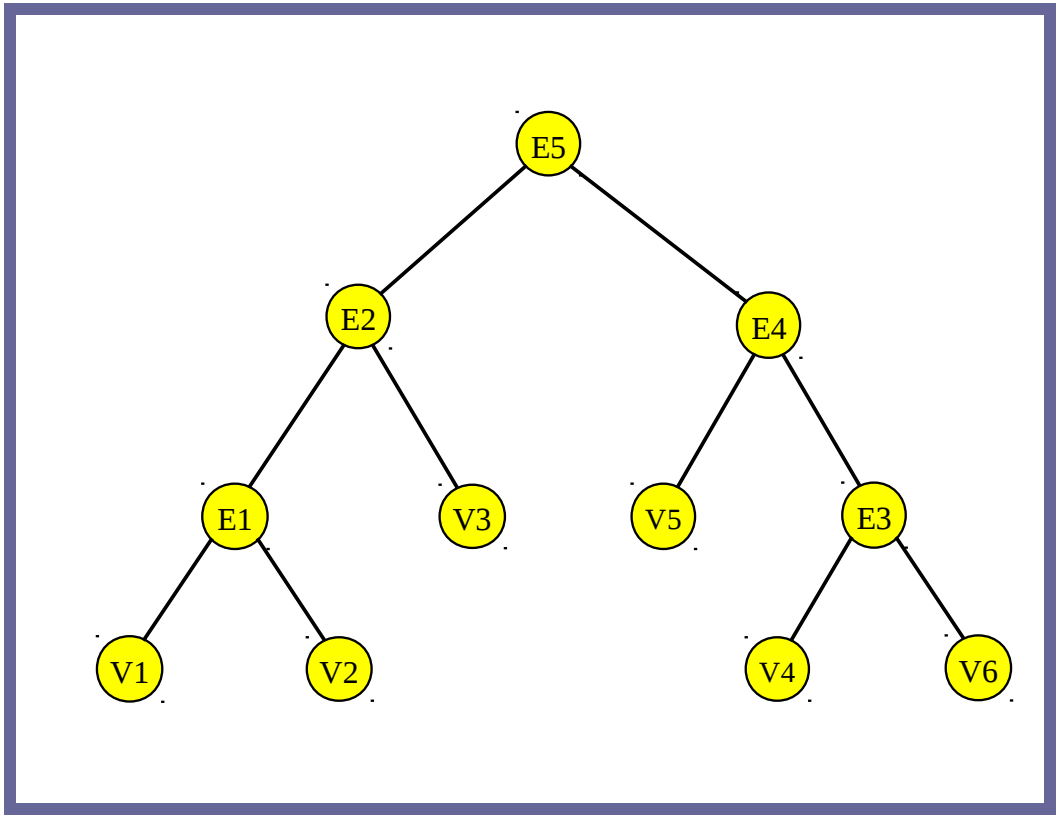




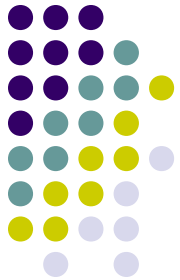
Kruskal 生成顺序森林

- 我们得到了图 G 的最小生成顺序森林





引入 **LCA** 解决问题！



- 不难发现， Query(V_i, V_j) 即为顺序森林中他们的最近公共祖先！
- 根据已有的成果，我们可以在 O(N + Q) 的时间内完成两次删边操作之间的所有询问
- 可以采用 Tarjan 算法解决

水管局长

- 让我们总结以上的算法流程：

1) 生成结束时的最小生成森林和顺序森林；
2) 从后往前完成操作：对于删边操作，重新生成最小生成森林和顺序森林；对于连续的询问操作，将其作为离线 LCA 询问在顺序森林上处理；
3) 输出答案；

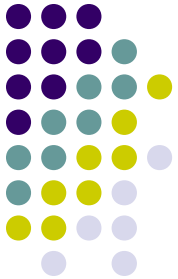
时间复杂度

- 在我的论文中以下结论被给出：
  1) 生成、维护最小生成森林和顺序树总时间花费为 $O(M\log_2 M + DN\alpha(N))$
  2) 询问可以在 $O(Q + ND)$ 的时间内解决

- 总的时间复杂度为：

$$O(M\log_2 M + DNa(N) + Q)$$

小结



- 我们先发现了解题的关键所在：
  - 边数过多（数据规模大）
  - 求解时间复杂度过高
- 针对第一点，我们引入最小生成森林
- 针对第二点，我们充分利用了经典算法
- 在已有的问题模型和算法上，合理的转化，最终得到了本题的解决方案

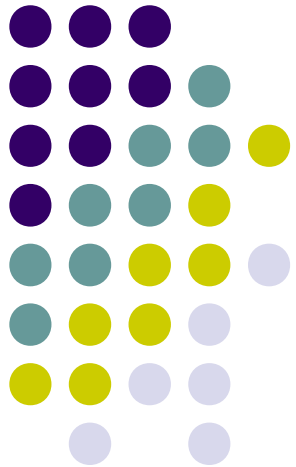
总结

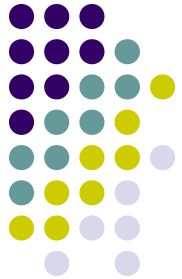
- RMQ&LCA 问题作为经典问题无论在算法学习上还是实际应用中都发挥了巨大的作用

- 解决问题研究算法时，恰当的引入经典问题和经典算法，无疑会大大加大我们的解题速度和精度！
- 站在巨人的肩膀上，我们才能看得更高、更远！

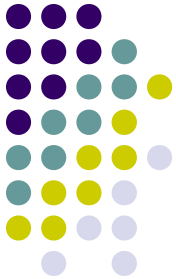
谢谢！

敬请批评指正！！





引入最小生成森林的时间花费



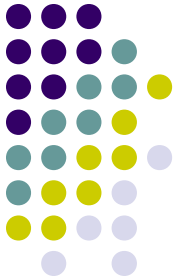
- 时间复杂度分为两个部分：
  - 生成最小生成森林的时间消耗
  - 维护最小生成森林的时间消耗

- 生成的复杂度由经典的 Kruskal 算法给出：

$$O(M\log_2 M)$$

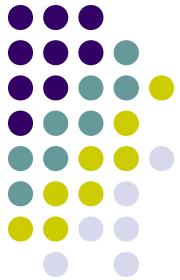
- 以下主要讨论维护的时间消耗

引入最小生成森林的时间花费



- 考察题目给出的删边操作
  - 若需删除的边非树边，那么不需要进行维护
  - 否则，我们需要枚举另一条边来进行补充，枚举量高达 O(M)！
- 正难则反，若是向图中添加一条新的边呢?
  - 注意到一个重要的性质，原非树边在添加新边后仍然为非树边！
  - 算法只需在原树边与新边中找出新的最小生成森林

引入最小生成森林的时间花费



- 将删边操作转化为添边操作，从后往前执行所有操作即可
- 综上，算法只需要在不超过 N 条边中建立最小生成森林，复杂度为 $O(N\alpha(N))$
- 注意到，最小顺序森林的生成与最小生成森林是同步的，所以亦可以在$O(N\alpha(N))$ 的时间内完成

完成询问的时间花费

- 按照添边操作，可以把操作序列分为很多段：
- 对于连续两个添边操作之间的询问操作，我们可以采取经典算法解决，时间花费为 O(N + Q')。总时间复杂度为：

  O(ND + Q)

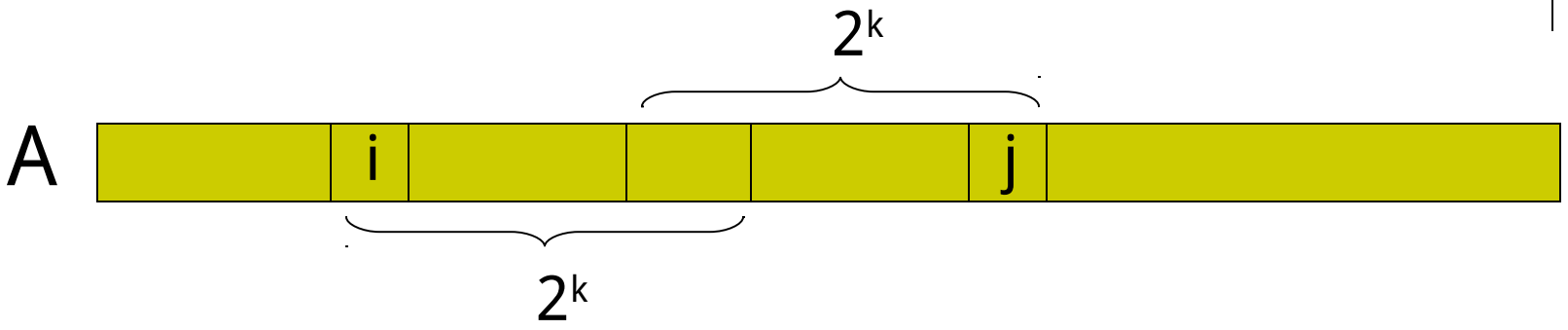
- 添边操作
  - 询问操作
  - ……
  - 询问操作
- 添边操作
  - 询问操作
  - ……
  - 询问操作
- ……

Sparse Table 算法

- 一般 RMQ 的 Sparse Table （ST）算法是基于倍增思想设计的 $O(N\log_2 N) – O(1)$ 在线算法

- 算法记录从每个元素开始的连续的长度为 2^k 的区间中元素的最小值，并以在常数时间内解决询问

Sparse Table 算法

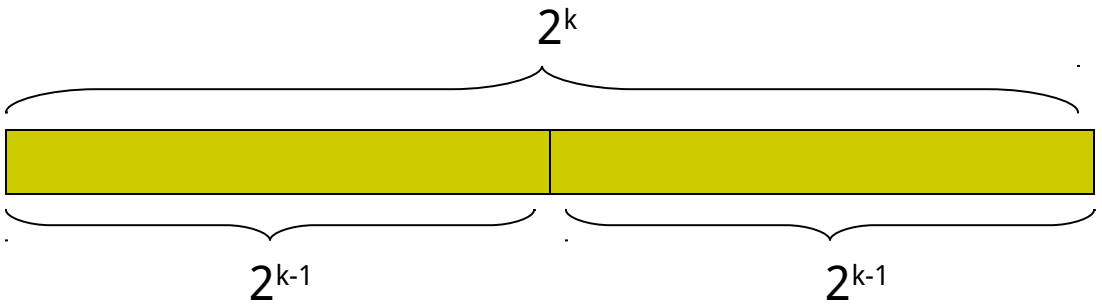


- 对于 RMQ （A， i， j），我们可以找到两段极大的长度为 2 的幂的区间（如图）覆盖 [i， j]

- 由已经计算的结果在 O(1) 的时间内解决询问

Sparse Table 算法

- 我们可以用 O(N) 的时间处理长度为 1 的区间
- 对于长度为 2^k 的区间的最小值，可以由两个长度为 2^{k-1} 的区间的最小值得到（如图）



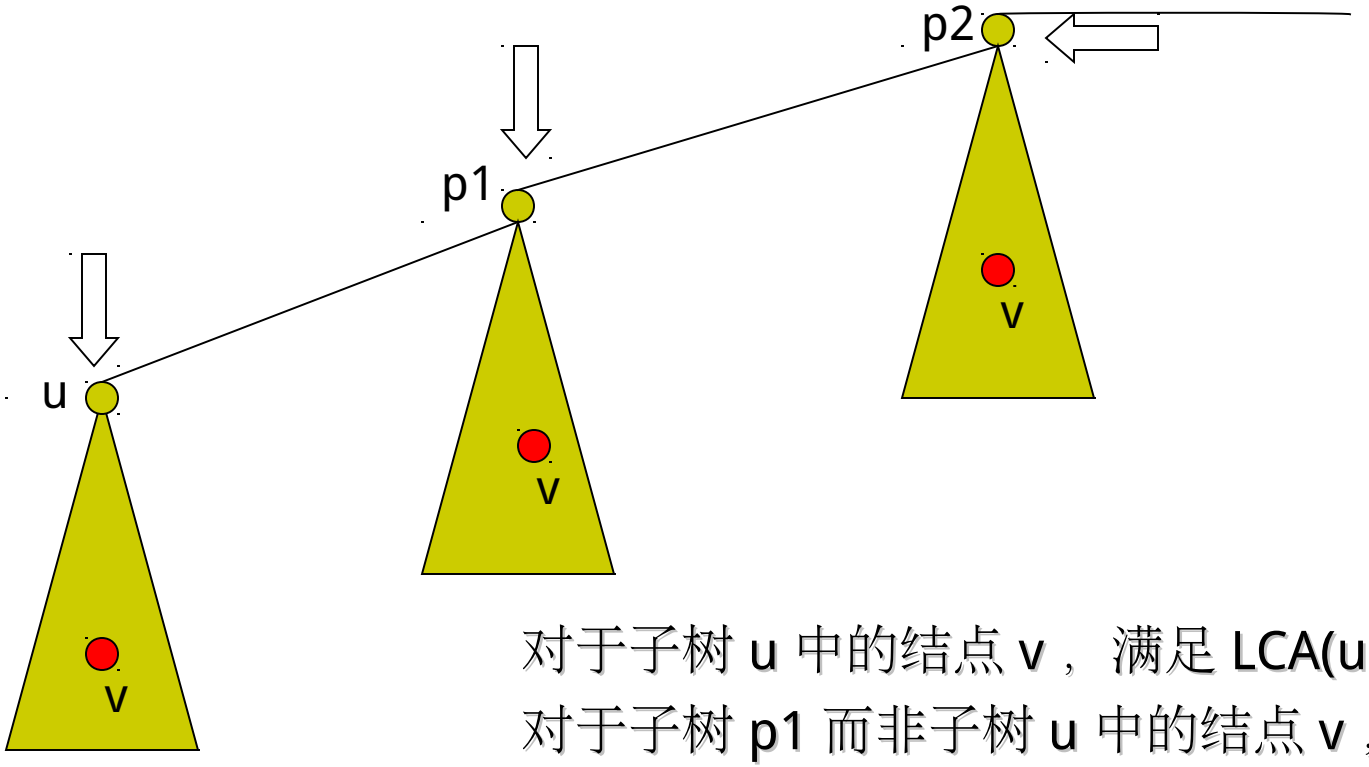
- 利用倍增思想，在 $O(N\log_2 N)$ 的时间内，我们可以预处理所有长度为 2 的幂的区间的最小值

Tarjan 算法

- 解决 LCA 问题的 Tarjan 算法利用并查集在一次 DFS （深度优先遍历）中完成所有询问。它是时间复杂度为 $O(Na_{(N)} + Q)$ 的离线算法

Tarjan 算法

- 考察树 T 中所有与结点 u 有关的询问 (u, v)

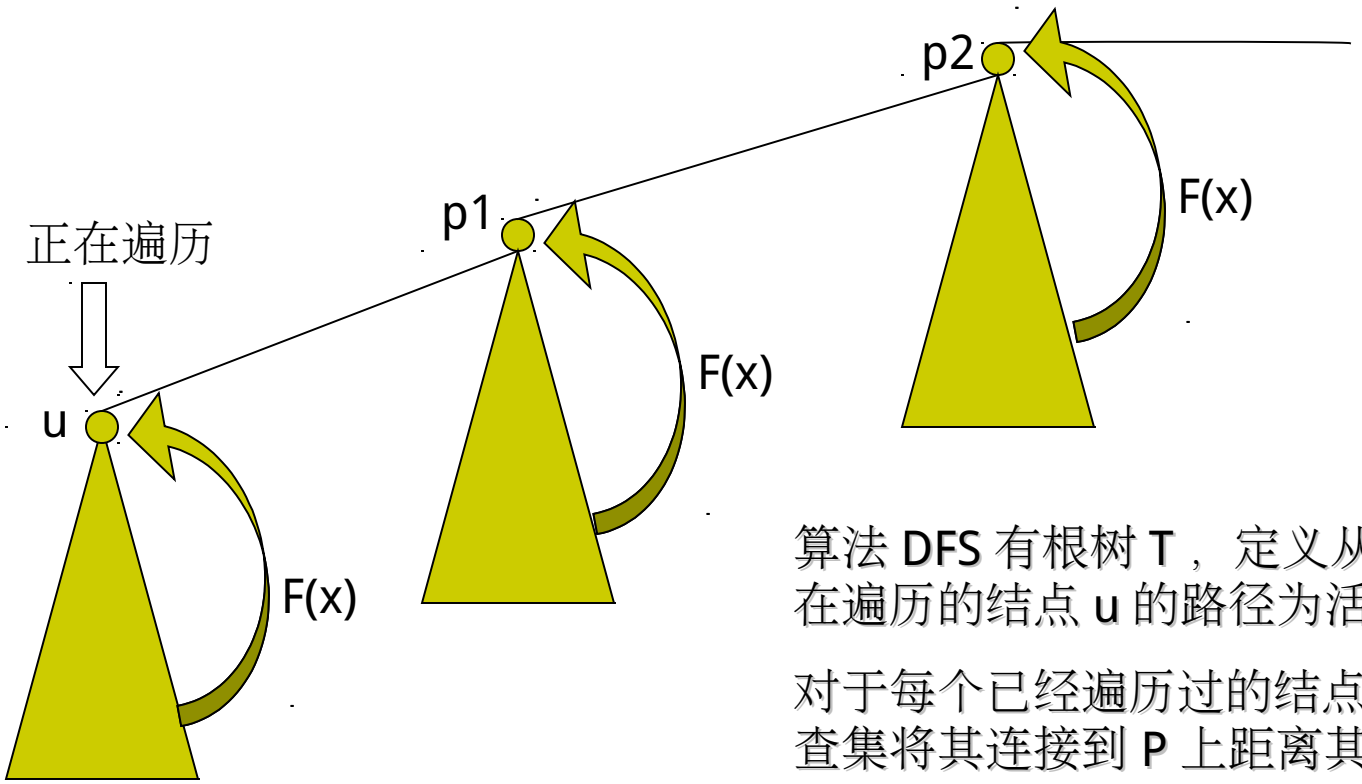


对于子树 u 中的结点 v，满足 LCA(u, v) = v

对于子树 p1 而非子树 u 中的结点 v，满足 LCA(u, v) = p1

对于子树 p2 而非子树 p1 中的结点 v，满足 LCA(u, v) = p2

Tarjan 算法

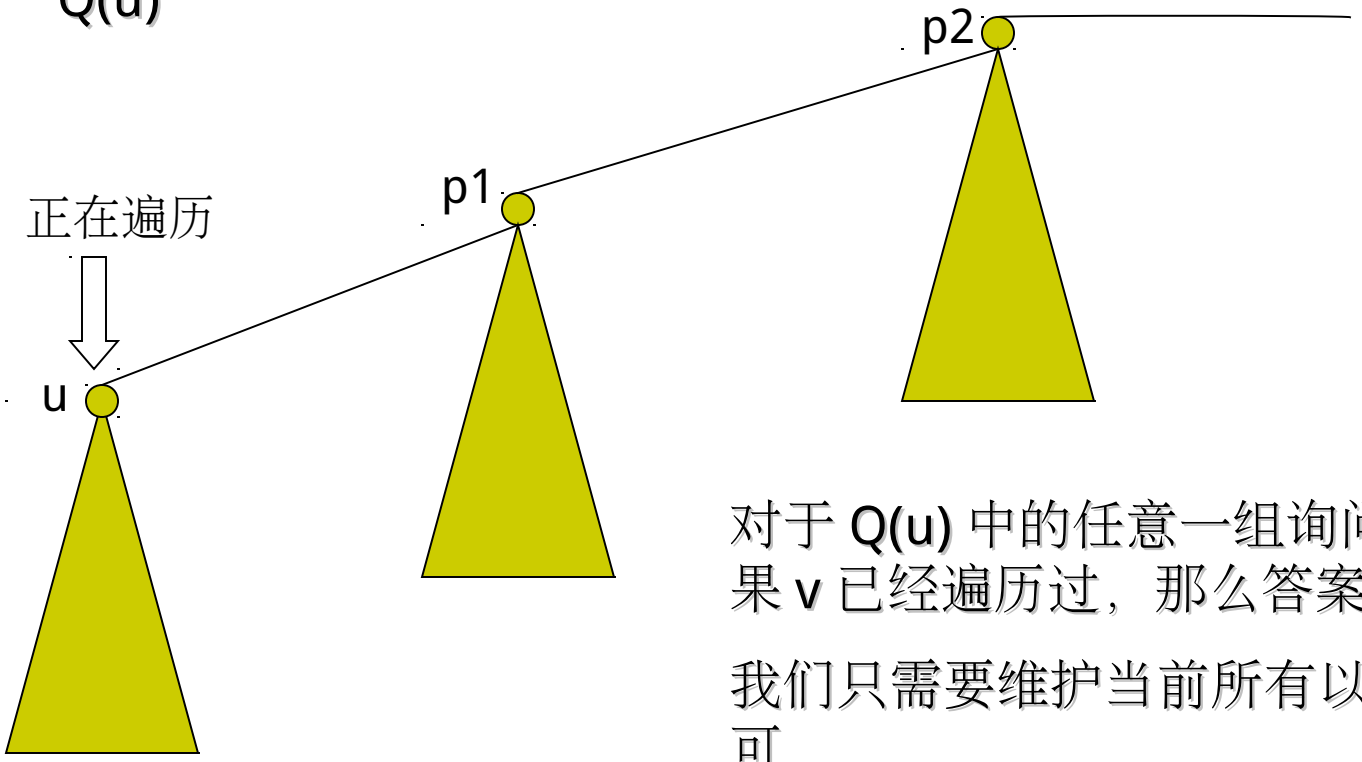


算法 DFS 有根树 T，定义从根节点到当前正在遍历的结点 u 的路径为活跃路径 P

对于每个已经遍历过的结点 x，我们使用并查集将其连接到 P 上距离其最近的结点 F(x)

Tarjan 算法

记录与 u 有关的询问集合为 Q(u)

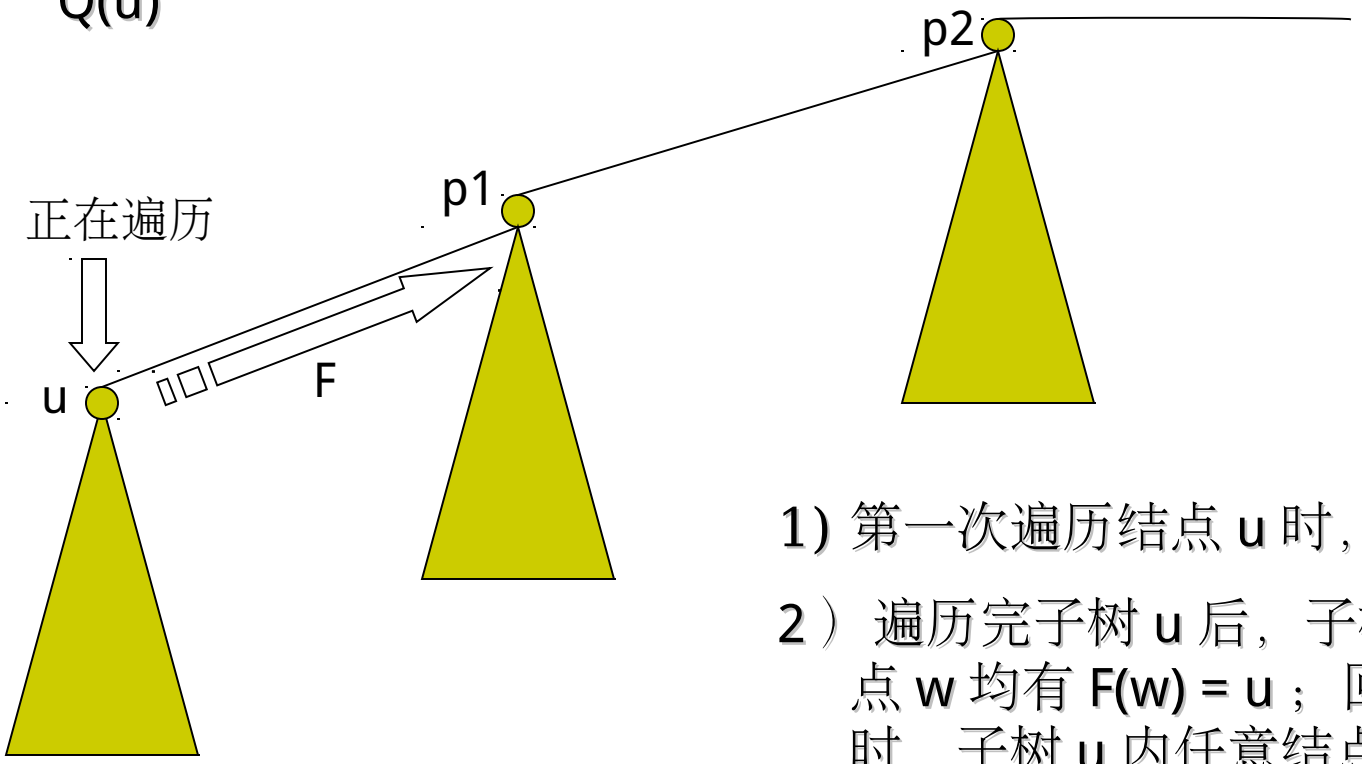


对于 Q(u) 中的任意一组询问 LCA(u, v)，如果 v 已经遍历过，那么答案即为 F(v)

我们只需要维护当前所有以遍历结点的 F 即可

Tarjan 算法

记录与 u 有关的询问集合为 Q(u)



1) 第一次遍历结点 u 时，有 F(u) = u ；

2）遍历完子树 u 后，子树 u 内任意结点 w 均有 F(w) = u ；回溯回结点 p1 时，子树 u 内任意结点 w 均有 F(w) = p1 ，使用并查集完成即可；

Tarjan 算法

- 算法流程:

Tarjan_DFS(u)

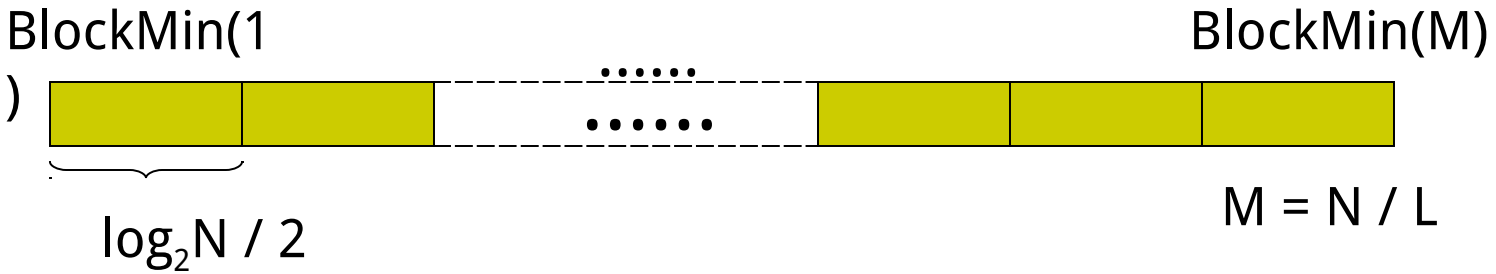
1) F(u) ← u ;
2) For (u, v) ∈ Q(u) do Answer(u, v) ← F(v)
3) For v ∈ son(u)
   a) Tarjan_DFS(v) ;
   b) F(v) ← u ;

注：此处 F 采用并查集实现

±1RMQ 算法

- 算法的核心思想在于分块：

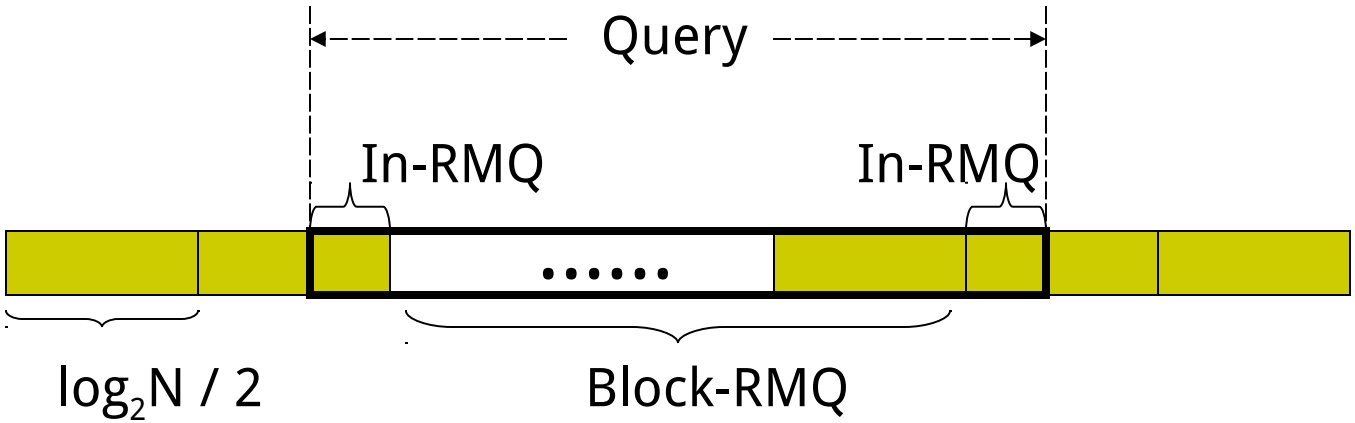
- 以 $L = \log_2 N / 2$ 块长把 B 划分为 $M = N / L$ 段，记录第 k 块的最小元素为 BlockMin(k)，把 M 块的最小值组成序列 Blocks，利用分块思想，我们可以把询问分为两个部分询问：



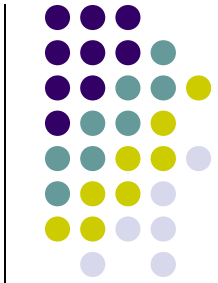
±1RMQ 算法

1) 连续的 BlockMin 取最小值，即 Block-RMQ；
2) 两端块中某一部分取最小值，即 In-RMQ；

这两个问题都可以 O(N) – O(1) 内实现



Block-RMQ



- Block-RMQ 采用 ST 算法来实现，根据前文所讨论的，时间复杂度为 $O(M\log_2 M) – O(1)$

- 因为 $M\log_2 M < 2N / \log_2 N * \log_2 N = O(N)$，所以 Block-RMQ 的复杂度不大于 $O(N) – O(1)$

In-RMQ

a) B 中任意两个相邻数为 ±1，所以本质不同的块至多有 2^{L-1} 种；

b) 对于每一种块上的询问至多只有 $O(L^2)$ 种；

- 所以本质不同的询问数至多有

  $O(2^{L-1}L^2) = O(N^{0.5}\log^2_2 N) < O(N)$

- 完成 In-RMQ 只需预处理所有本质不同的询问对应作答即可，时间复杂度为 O(N) - O(1)

±1RMQ 算法

- 因为 LCA 问题可以转化为生成序列的 RMQ 问题且问题规模不变，所以 LCA 问题可以在 O(N)-O(1) 的时间内解决；

- 因为 RMQ 问题可以转化为 LCA 问题且问题规模不变，所以 RMQ 问题可以在 O(N) - O(1) 的时间内解决；