off-line LCA

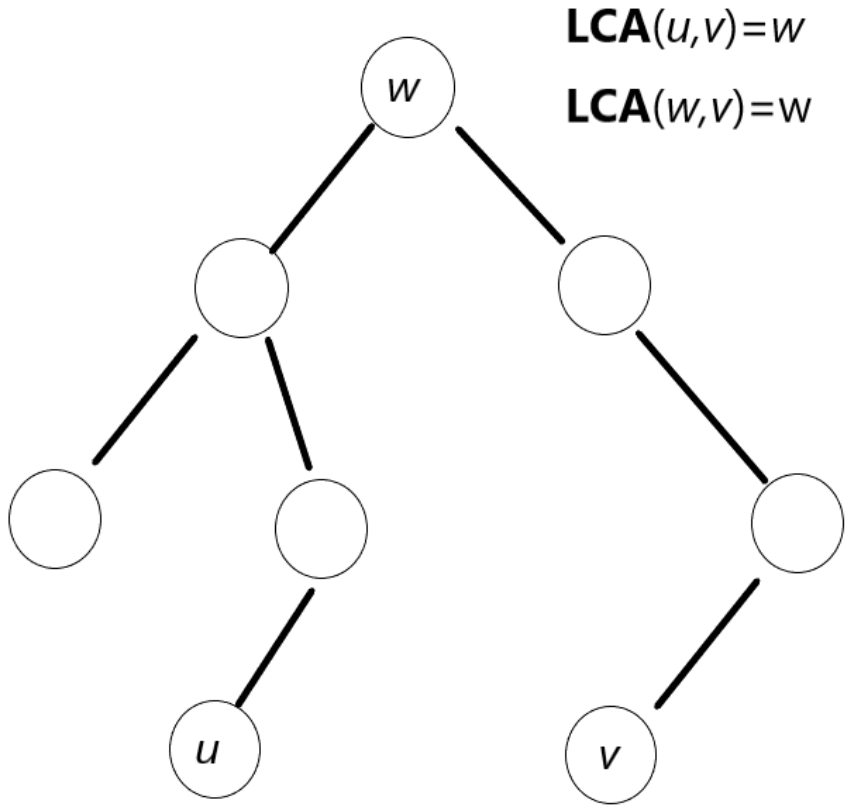
王康浩 2020.10.28

Introduction

- 离线算法和在线算法的不同
- 一个在线算法是指它可以以序列化的方式一个个的处理输入，也就是说在开始时并不需要已经知道所有的输入。
- 一个离线算法，在开始时就需要知道问题的所有输入数据，而且在解决一个问题后就要立即输出结果。

Introduction

- LCA（least common ancestor）
- The least common ancestor of two nodes u and v in a rooted tree T is the node w that is an ancestor of both u and v and that has the greatest depth in T. In the off-line least-common-ancestors problem, we are given a rooted tree T and an arbitrary set $P = \{\{u, v\}\}$ of unordered pairs of nodes in T, and we wish to determine the least common ancestor of each pair in P.



my first idea

Algorithm 1 least-common-ancestors algorithm

解答:

```
1:  procedure LCA(T, a, b)
2:      new array A
3:      i ← 1
4:      while a ≠ NILL do
5:          A[i] ← a
6:          a ← a.p
7:          i++
8:      end while
9:      while b ≠ NILL do
10:         if b ∈ A then
11:             return b
12:         else
13:             b ← b.p
14:         end if
15:     end while
16: end procedure
```

- 储存a和a 的祖先
- b自叶到根查找祖先
- 空间复杂度是O(h)
- h是树的高度
- 最坏时间复杂度:

$$p * (O(h) + h * O(h)) = O(ph^2)$$

- p是问题数量
- 在线算法

- 通过查询网络，得到LCA的常见解法有：
  - 基本解法
  - tarjan的解法
  - 倍增法
  - RMQ和LCA互相转化的解法

- 主要内容是tarjan's off-line LCA algorithm

基本解法

Algorithm 1 least-common-ancestors algorithm

解答:

```
1:  procedure LCA(T, a, b)
2:      while a ≠ b do
3:          if a.depth == b.depth then
4:              a = a.p
5:              b = b.p
6:          else
7:              if a.depth > b.depth then
8:                  a = a.p
9:              else
10:                 b = b.p
11:             end if
12:         end if
13:     end whilereturn b
14: end procedure
```

- 同样是对父节点查询
- 使用深度而不是祖先集合
- 时间复杂度 $O(n + ph)$
- 处理depth需要一次先序遍历

Tarjan's off-line least-common-ancestors algorithm

- To solve the off-line least-common-ancestors problem, the following procedure performs a tree walk of T with the initial call LCA(T.root). We assume that each node is colored WHITE prior to the walk.

```
LCA(u)
 1  MAKE-SET(u)
 2  FIND-SET(u).ancestor = u
 3  for each child v of u in T
 4       LCA(v)
 5       UNION(u, v)
 6       FIND-SET(u).ancestor = u
 7  u.color = BLACK
 8  for each node v such that {u, v} ∈ P
 9       if v.color == BLACK
10            print "The least common ancestor of"
                  u "and" v "is" FIND-SET(v).ancestor
```

- here are four questions.

a. Argue that line 10 executes exactly once for each pair $\{u, v\} \in P$

- 对于LCA(u) u可以遍历每个节点 则每对{u,v}都可遍历两边
- u黑v白：① v是u的祖先
  ② v和u在不同分支，v在右支
- u黑v黑：③u是v的祖先
  ④v和u在不同分支，v在左支

```
LCA(u)
1   MAKE-SET(u)
2   FIND-SET(u).ancestor = u
3   for each child v of u in T
4       LCA(v)
5       UNION(u, v)
6       FIND-SET(u).ancestor = u
7   u.color = BLACK
8   for each node v such that {u, v} ∈ P
9       if v.color == BLACK
10          print "The least common ancestor of"
                u "and" v "is" FIND-SET(v).ancestor
```

b. Argue that at the time of the call LCA(u), the number of sets in the disjoint-sedata structure equals the depth of u in T .

- 第一调用LCA(T.root) 每个调用是发生在循环line3-4,5,6中
- 以下：1：每次调用LCA(u)会产生一个新的集合
- 2：每次调用LCA(u)发生在LCA(u.p)中的line 4
- 3：LCA(u)中每次line5,6结束后，line4中产生的新集合会合并到 u 所在集合
- 设line3的循环是先左支后右支，
- 1.2.说明了， 集合数F(u.left)=F(u)+1 注：+1是make-set(u).
- 结合3， F(u.right)=F(u)+1 (左支产生的新集合被LCA(u)中line5 合并到u代表集合中)
- 得：递归条件： F(u)=F(u.p)+1, no matter u==u.p.left or right
- 初始条件：F(T.root)=0，T.root.depth=0 （第一调用）
- 可以得到： F(u)=u.depth

c. Prove that LCA correctly prints the least common ancestor of u and v for each pair $\{u, v\} \in P$

- 由(a)知道每一对$\{u, v\} \in P$ 都会打印
- 针对被打印的那次遍历{u,v}
- u黑v黑：③u是v的祖先

  ④v和u在不同分支，v在左支
- 对于情况④进行研究
- u,v 属于两个分支，则两支在点w分歧
- 显然，w及w的祖先节点是u,v的公共祖先节点，w是其中深度最大的。

```
LCA(u)
1   MAKE-SET(u)
2   FIND-SET(u).ancestor = u
3   for each child v of u in T
4       LCA(v)
5       UNION(u, v)
6       FIND-SET(u).ancestor = u
7   u.color = BLACK
8   for each node v such that {u, v} ∈ P
9       if v.color == BLACK
10          print "The least common ancestor of"
                u "and" v "is" FIND-SET(v).ancestor
```

d. Analyze the running time of LCA, assuming that we use the implementation othe disjoint-set data structure in Section 21.3.

- 由a得，每个节点都遍历一次
- MAKE-SET n 次,FIND-SET n-1次 ,UNION n-1次
- 即总执行操作 m=3n-2 次，其中 n次 MAKE-SET
- 带入section 21.3 末尾结论 这部分时间复杂度 $O(m\alpha(n))$
- 在可以预见的范围内时间复杂度为O(m)=O(n)
- 问题部分是q个问题 每个判断2次黑白，打印1次，复杂度O(q)，
- 总和O(n)+O(q)

倍增法 预处理

- 设f[x][k]表示x的2^k辈祖先,即从x向根节点走2^k步到达的节点。特别地,若该节点不存在,则令f[x,k]=0。f[x,0]就是x的父节点。可以得出f[x][k]=f[f[x][k-1]][k-1]。
- 我们可以对树进行遍历,由此得到f[x,0]，再计算f数组所有值。
- 以上部分是预处理,时间复杂度为O(nlog n)。之后可以多次对不同的x,y计算LCA,每次询问的时间复杂度为O(log n)。

- 初始化 for all f[x][k]=0
- x=T.root
- function(x){
- k=log depth[x]
- for i from 1 to k
- 　　f[x][i]=f[f[x][k-1]][k-1]
- for each child y of x
- 　function(y)　}

基于f数组计算LCA(x,y)

- 1.设dep[x]表示x的深度。不妨设dep[x]≥dep[y](否则可交换x,y)。
- 2.用二进制拆分思想,把x向上调整到与y同一深度。具体来说,就是依次尝试从x向上走k= 2^{logn},…,2^1,2^0步,若到达的节点比y深,则令x=f[x,k]。重复至dep[x]=dep[y]。
- 3.若此时x=y,说明已经找到了LCA,LCA就等于y。
- 4.若此时x≠y，依次尝试把x,y同时向上走k=2^{logn},…,2^1,2^0步,若f[x,k]≠f[y,k](即仍未相会),则令x=f[x,k],y=f[y,k]。（重复）
- 5.此时x,y必定只差一步就相会了,它们的父节点f[x,0]就是LCA。

- 步骤2 while 循环条件
  - depth[x]≤depth[y]
- 步骤4
- while (x≠y && k≥0)
  - if(f[x][k]==f[y][k])
    - k--;
  - else
    - x=f[x][k];
    - y=f[y][k];
    - k=log depth[x];
- 循环结束时：x≠y，f[x][0]==f[y][0]
- 即 x.p==y.p

LCA和RMQ（区间最小值）转换

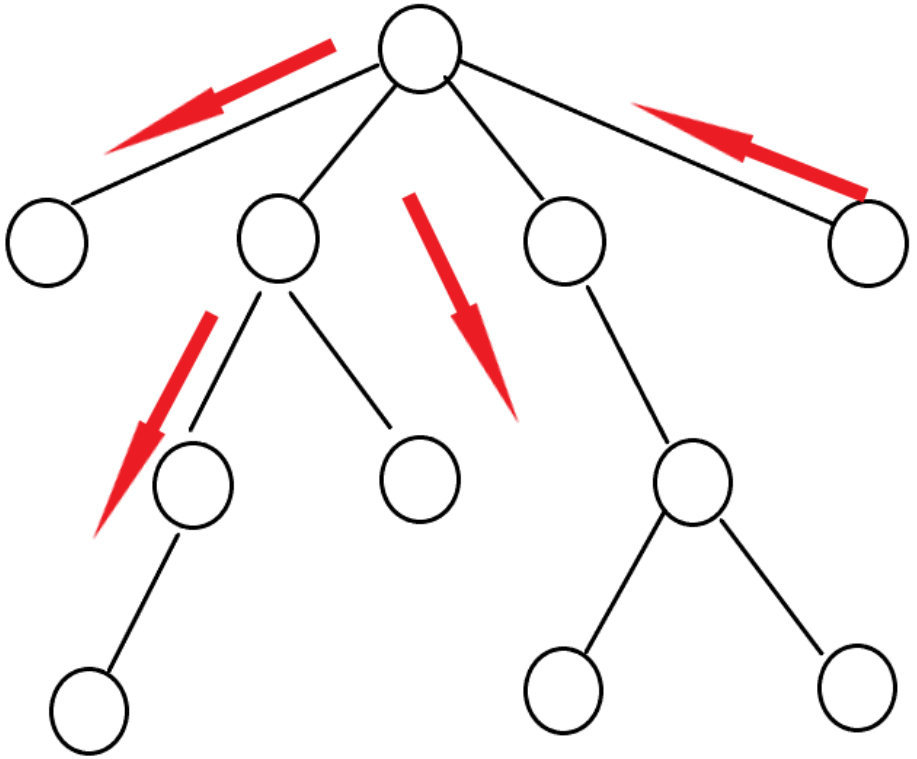
- 从根节点开始深度遍历一棵树产生表1

i(index)						
n[i](name)						
depth[i]						

- 对表处理得表2

i(name)					
g[i](index)					

- 任意节点对(u,v) 其lca
- 在表2中得 g[u],g[v],在表1中index[g[u],g[v]]区间找到depth[i]最小值对应index，输出name[index]



- 感谢马骏老师的解惑
  资料来源于TC和网络
- https://blog.csdn.net/jeryjeryjery/article/details/52853017
- https://www.cnblogs.com/ljy-endl/p/11349087.html