IL 思路及遇到的问题、解决方法

-----陈健琦

前序：

XML：可扩展标记语言，其标签是可自定义的。可以将 XML 用树模型表示，从而解决问题。

HTML：是超文本标记语言，其标签是不可自定义的。

XML 的例子：

<lib = “YDDB”>

<au = “WY”>

<age> 18 </age>

</au>

<au = “ZCM”>

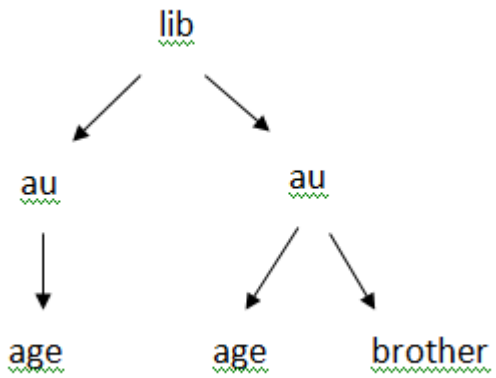
<age> 17 </age>

<brother = “WY”> </brother>

</au>

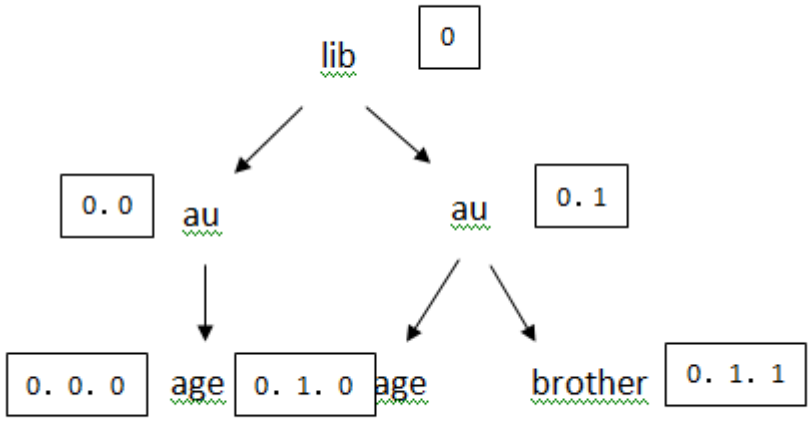
</lib>

将上述 XML 文档表示成树的形式为：

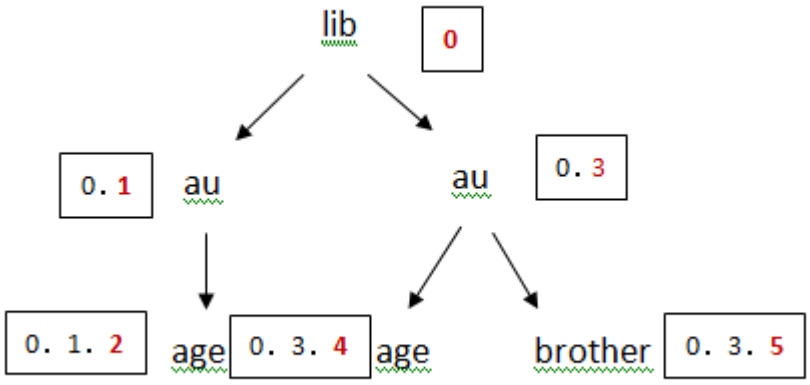


我们将其进行编码：

编码方式一：第一个孩子为 0，第二个孩子为 1



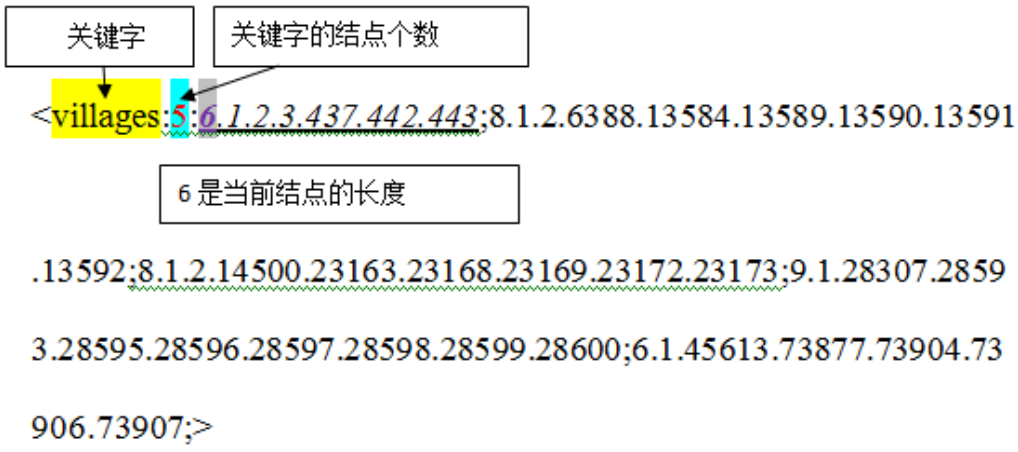
编码方式二：将其进行先序遍历，遍历的顺序就是其编码的最后一位。红色代表的是先序遍历的顺序。



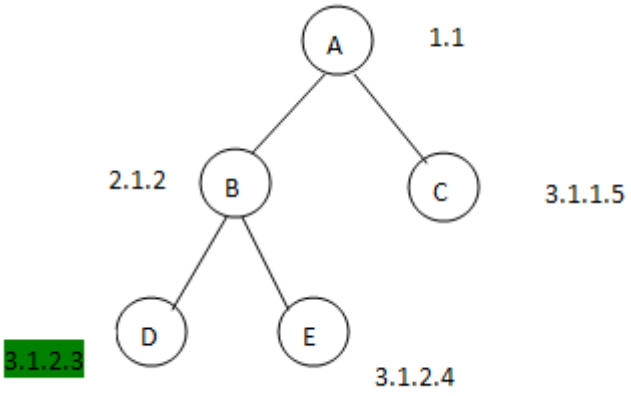
我们可以在这个编码的基础上添加其编码的长度。这种编码与前一种相比，此编码要优于前一种方法，因为我们可以根据某一个编号定位到具体的一个结点，例如：1 我们可以定位到结尾为 1 的结点上。而前一种编码不能做到这一点。

下面是 IL 算法的思路以及遇到的问题、解决方法

1. 首先将论文看懂，看论文时并没有太大的困难，倒是感觉论文中的内容很简单，后来做完 IL 后才领悟到，IL 本身的算法并不难，难的是如何将思想转化为程序。
2. 论文看懂后，拿到了倒排表，看不懂的是：这是什么意思？



3. 编号方法：



以 D 为例：

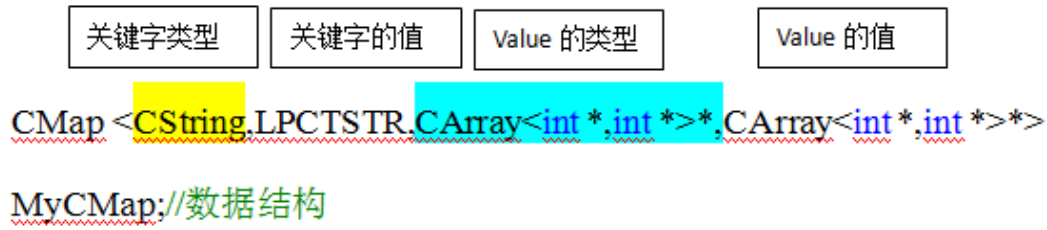
第一个 3：是编号的长度

最后一个 3：是前序优先遍历的顺序

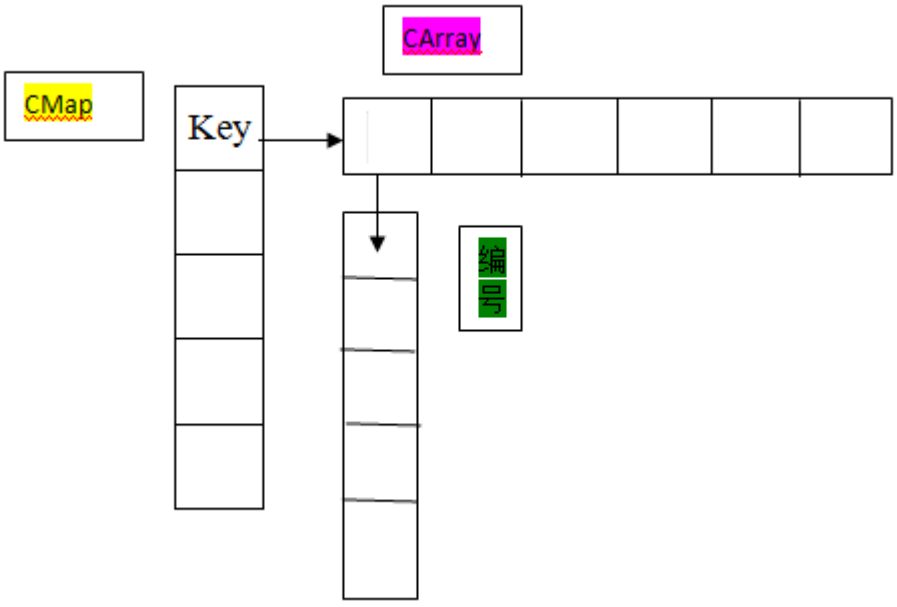
中间是其父节点的编号。

4．看懂了倒排表后，接下来就是考虑怎么将关键字取出，怎么将该关键字的个数取出，怎么将结点的编号取出。选择怎样的数据结构的问题。

我采用的是 CMap 数据结构，具体是



5.接下来，就开始研究这个数据结构的意义了，CMap使用方法相当于JAVA中的HashMap,即是一个KEY-Value的存储结构。



※C++中CMap的底层实现：

CMap是一种Hash Map，Hash Map要求每个元素都要有一个Hash值——一个关于KEY的函数，Hash Map用这个值作为Hash表的索引。如果有多个KEY的Hash值相同，它们将以链表的方式存储。

※JAVA中的HashMap与TreeMap的区别：在java 2集合框架中的

Map接口有两个通用实现:HashMap和TreeMap. HashMap是采用哈希表实现,是Map接口的最好的全面实现.TreeMap实现了Map的子接口SortedMap,采用红黑树作为底层存储结构,提供了按照键排序的Map存储.

◆LPCTSTR 知识：

http://baike.baidu.com/link?url=jOrAe1-jtJFg18K13VZklU4IIT8rwXeDYSO90mHfeHiIarU7i2nEpNMMAQhowiS6CTMypKW5XC3fSywWOz4fuK

主要是用于：1）用来表示你的字符是否使用UNICODE

2）用来进行类型转换，例：

CString 转 LPCTSTR:

```
CString cStr;
const char *lpctStr=(LPCTSTR)cStr;
```

LPCTSTR 转 CString:

```
LPCTSTR lpctStr;
CString cStr=lpctStr;
```

6.弄懂了数据结构就开始学习CMap,CArray，CString的知识，见附件。主要弄懂CMap的存取，CArray的存取，CString的截取等操作。

主要思路：

给定一系列查询关键字，我们需要求的是这些关键字的SLCA即最小最低共同祖先。

1） 对这些给定的关键字按照长度，即关键字中编码个数由小到大进行排序，排序的目的是拿编码个数较小的关键字与

关键字个数多的中去比较。这样比较的意义是计算速度较快。

2） 将到其后的倒排表中，即次短的倒排表中去计算左右匹配。

左匹配：得到比该编码小的中最大的

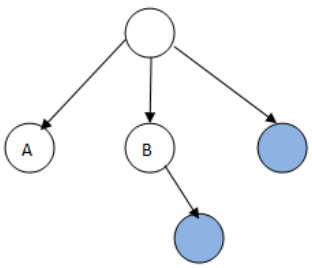
右匹配：得到比该编码大的中最小的

进行左右匹配的原因是距离越近其公共祖先就越低。

3） 求出左右匹配后，拿该编码和左右匹配求LCA,即最小公共祖先。LCA实际是编码中最长的那个。

4） 我们要求的是 SLCA（是一个结点与一个集合比较所得出的最小的公共祖先），只需要比较编码长度与编码的最后一位即可，编码长且编码最后一位大者的是祖先。然后处理最短倒排表中的第二个编码，得到 SLCA，判断是否是祖先关系，若不是祖先关系将其加入到最后的结果集中。

注：倒排表中的结点序号是有序的，如果计算当前的结点 B 与前一个结果集中的结点 A 没有祖先关系，那么后面新计算出的结点 C 也不会与结点 A 有祖先关系。因为集合中的结点编号越来越大，所以后面计算的结果的编号要么是 B 的孩子，要么与 B 没有祖先关系。如下图所示：



//////////////////////////以下是函数的总体介绍////////////////////////////////////

■IL_main()

{

//对关键字的多少进行排序

//求slca(),调用的函数get_slca()

//释放空间

}

■get_slca()

{//左右匹配

//调用lca()，求出最低共同祖先

//求子孙结点，调用descendant()

//条件1：判断结点大小

//条件2：判断是否是祖先结点。若满足以上两个条件，那么将结点加入到结果集中

//释放空间

}

■descedant()//返回子孙结点

{}

■lca()//求最低共同祖先

{}

■Lm()//左右匹配

{}

7.学习完数据结构后，考虑怎么有效的进行取出的问题（问题4中提到的）。

以下是截取方式（可能不是最佳的）：

1） 查找到'<'位置，'：'位置，然后用CString类型截取出关键字。

关键字取出后有冒号，需要过滤掉。注意Mid()函数的第二个参数是截取的个数。

2） 找到第二个'：'，用Mid()截取出结点的个数。截断字符串。

3） 考虑使用二重循环来遍历所有。

第一层循环是使用getline()读取每行；第二层循环是依次读取读入的这行的数据。那么就是找'.',这样可以依次取出编号的长度，以及编号。取出编号长度，以及编号后将其存储到int型的指针中。

知识点：

★String转化成int型函数 _ttoi()。

○最后一个数字即分号前的数，单独处理。

○当当前的结点编号结束后要加入到CArray中

★将string转换为CString,用 c_str()

注意：

写程序时遇到的粗心的问题：

◎使用while()自变量忘记自加

◎变量命名不规范（没有见名知意等），使得测试时花费了很多时间

◎什么时候该加入到CArray中，什么时候该加入到CMap中

◎小于和小于等于问题

◎由于逻辑问题导致每次取的个数都不一样，这个需要细心观察

◎存储的长度问题，考虑是否是需要加一

全部结束后，将CArray加入到CMap中去。

刚刚开始写读函数可能会遇到各种问题，当时在读函数上用了几天的时间。由于逻辑问题，易把问题想的太复杂了。

8.左右匹配问题（**重点**，时间主要花费在左右匹配上了）

计算它的左右匹配的原因是距离越近，其LCA越低。

注意1:在做左右匹配时一定要保证数据保存的正确性。因此需要测试一下。

左匹配：无论是左匹配还是右匹配都使用**折半查找**。因为折半查找的时间复杂度是$O(\log n)$。

注意2：由于数据集中不同的关键字的结点的编号可能会相同。例如：text 和 tissue 同时含有 7.1.2.3505.5383.5394.5407.5411。

这是因为XML中的标签和及其此标签中的文本会被抽象成为不同的关键字。例如：

```
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
```

```
<body>Don't forget the meeting!</body>
</note>
```

灰色区域的form 与John的编号相同。

在左右匹配时，必须考虑左右匹配不存在的情况。

提示：刚刚开始写左右匹配时可能会想到把左右匹配写成一个函数，但是若发现这个有点困难，何不放一放，暂且将左右匹配写为两个函数，待实现左右匹配时再将左右匹配合为一个函数即可。

9.lca(v1,v2)即找最低共同祖先

- 若任意一个为空，则为空。
- 若都不为空，则返回最长共同前缀（即最低共同祖先）。

最长共同前缀的求法：根据参数（指针类型），循环访问，其数据是否相等，直到不相等，退出循环。从而得到其最长共同前缀。

写这个函数并没有什么难度。

注：在此注意的是长度的边界问题。

10. descedant(v1,v2)//返回子孙结点

- 若其中有一个为空则返回的是非空的参数。
- 若都不为空，返回的是v1,v2中的子孙结点。

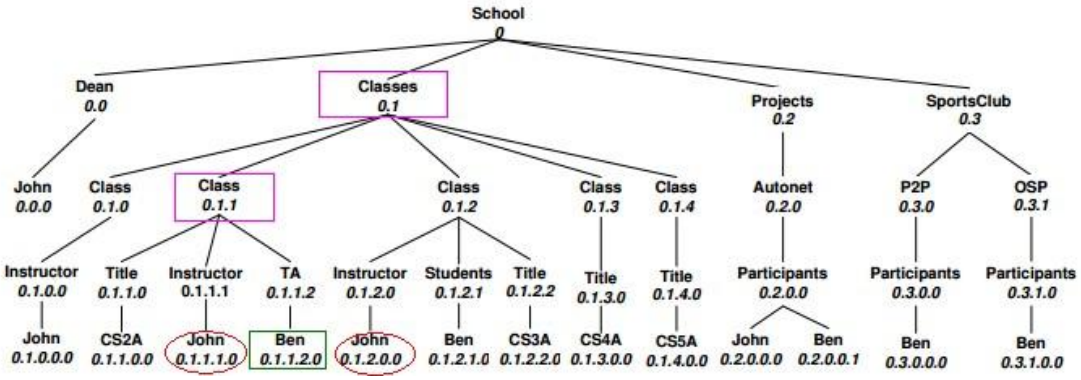
子孙结点的判断方法：

观察子孙结点与其祖先结点，祖先结点的编号会完全在子孙结点中出

现，这样以来，可以判断其长度即可，长度长的为子孙结点，长度短的是祖先结点。

可能会有此顾虑：参数v1,v2会不会出现不是祖先和子孙的关系？

这种情况是不会存在的，如图所示：



Ben 0.1.1.2.0(与自己写的编码不太一样，但道理一样)与John的左匹配是0.1.1.1.0，右匹配是0.1.2.0.0。在经过lca()后为0.1.1和0.1。结果肯定是祖孙关系。所以descedant(),传递的两个参数是确定的祖孙关系。

11.get_slca()找最小最低共同祖先

本函数就是调用前面的几个函数。

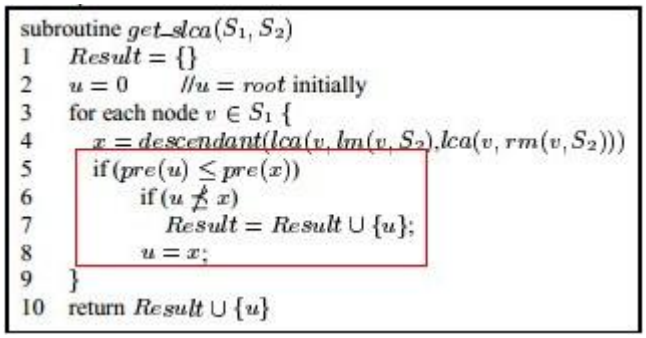
```
subroutine get_slca(S1, S2)
1   Result = {}
2   u = 0        //u = root initially
3   for each node v ∈ S1 {
4      x = descendant(lca(v, lm(v, S2)),lca(v, rm(v, S2)))
5      if (pre(u) ≤ pre(x))
6           if (u ⋠ x)
7              Result = Result ∪ {u};
8           u = x;
9   }
10  return Result ∪ {u}
```

◎初始根结点u = 0，在实现时可以设为1.1。

◎判断u的编码是否是小于x的编码

◎判断u是否是x的祖先，若不是祖先则加入到Result中。

u = x是包含在if(pre(u) < pre(x))中的，伪代码有规定（可能是不成文）缩进对齐的是同一级别。即：



12.写一个调用get_slca()的函数即可。

//

优化阶段

首先优化的是左右匹配，可以考虑是否是用一个函数来完成左右匹配。

解决方法：使用了两个全局变量left,right，来记录的是左右匹配的位置，而不是使用指针；左匹配做完之后，右匹配的位置也就得到了。这样可以通过位置来获得左右匹配。

1. Lca(),descendant()

   优化的方法：

   将返回的最低共同祖先/子孙结点的编号改为使用结点类来存储。而结点类中包含了一个指针，指向的是子孙的编号；一个int型的变量存储的是编号的长度。

2. get_slca()

   优化方法：

   将u的初始化的1.1改为了，第一次的求子孙的结果。减少了一

步（将1.1做为祖先）。在此不做进一步的分析。当然u相应的改为了结点类型的，因为我们在加入Result中时使用的是指针型，因此在加入前设置一个指针，将结点存的值复制给指针。

将比较大小单独做一个函数进行判断，判断时所传递的参数是结点类型。且不允许修改其值。即：

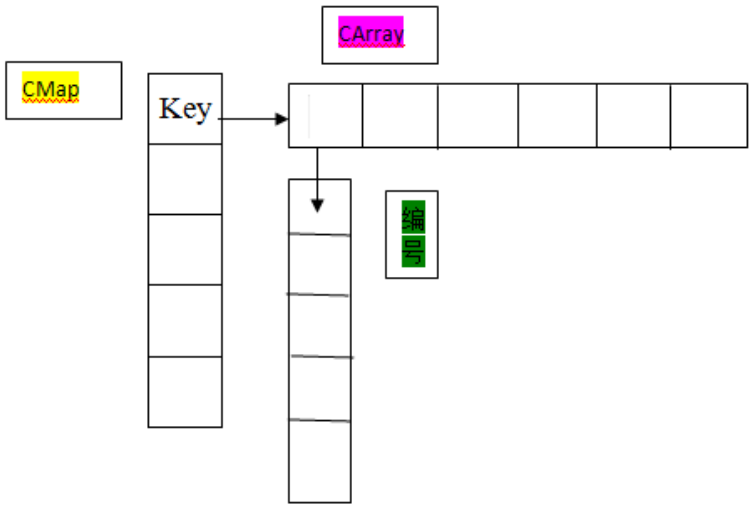
```
int InStream::Compare(const Node &v, const Node &node)//比较大小
```

3. 在调用get_slca时的优化

因为少量的结点中找量大的结点的左右匹配的时间<将量大的与量小的结点中找左右匹配时所用的时间。所以可以考虑将比较的关键字的个数进行按升序排序。然后再调用get_slca()函数。使用的是直接插入排序。

4. 释放空间

刚刚写完xmark500时仅仅只能跑一遍，这就需要进行释放空间。



Carray->RemoveAll()，仅仅是将Carray中的空间进行释放，而空间中

的指针没有被释放掉。因此将指针进行释放，这就需要使用双层循环将Carray进行释放干净。

相应的CMap也需要在析构函数中进行释放空间。

◆注意：释放空间时，需要将之前不再使用的空间进行释放，若下次还会使用，则需申请一个临时的空间暂存，使用完后，将临时空间释放即可。

5.此外还可以将结点类转换为结构体。如下：

```
struct Node
//class Node
{
//public:
//   Node(void);
//   ~Node(void);
//public:
int length;//长度
    int *p;//指针
};
```

将cpp文件注释掉即可。