字符串算法 part 1[1]

赖金霖[2]
清华大学　计算机科学与技术系

Aug 11, 2019

[1]本文件可在这里找到: https://github.com/lll6924/public_slides
[2]邮箱: laijl16@mails.tsinghua.edu.cn

赖金霖

- 高中就读于赣州市第三中学
- NOI 金牌 国家集训队
- 现就读于清华大学计算机科学与技术系
- NOIP2012~2015 一等奖
- NOIP2014 满分
- NOI2014 铜牌
- WC2015 二等奖
- APIO2015 金牌
- NOI2015 金牌
- 常用 id：lll6924

常用头像

常用函数 (C++98)

构建函数 (Constructor)

```
string();
string (const string& str);
string (const string& str, size_t pos, size_t len = npos);
string (const char* s);
string (const char* s, size_t n);
string (size_t n, char c);
template <class InputIterator>
   string (InputIterator first, InputIterator last);
```

迭代器 (Iterator)

```
iterator begin();                    iterator end();
reverse_iterator rbegin();           reverse_iterator rend();
```

常用函数 (C++98)

运算符 (Operator)

```
string& operator= (const string& str);
string& operator= (const char* s);
string& operator= (char c);
char& operator[] (size_t pos);
string& operator+= (const string& str);
string& operator+= (const char* s);
string& operator+= (char c);
string operator+ (const string& lhs, const string& rhs);
string operator+ (const string& lhs, const char* rhs);
string operator+ (const char* lhs, const string& rhs);
string operator+ (const string& lhs, char rhs);
string operator+ (char lhs, const string& rhs);
istream& operator>> (istream& is, string& str);
ostream& operator<< (ostream& os, const string& str);
```

常用函数 (C++98)

成员函数 (Function)

```
size_t size() const;                    bool empty() const;
size_t length() const;                  char& at (size_t pos);
void resize (size_t n);                 char& back();
void resize (size_t n, char c);         char& front();
void clear();

string& append (const string& str);
string& append (const string& str, size_t subpos, size_t sublen);
string& append (const char* s);
string& append (const char* s, size_t n);
string& append (size_t n, char c);
template <class InputIterator>
   string& append (InputIterator first, InputIterator last);
void push_back (char c);
```

常用函数 (C++98)

成员函数 (Function)

```
string& insert (size_t pos, const string& str);
string& insert (size_t pos, const string& str,
   size_t subpos, size_t sublen);
string& insert (size_t pos, const char* s);
string& insert (size_t pos, const char* s, size_t n);
string& insert (size_t pos, size_t n, char c);
string& erase (size_t pos = 0, size_t len = npos);
iterator erase (iterator p);
iterator erase (iterator first, iterator last);
void swap (string& str);
void pop_back();
const char* c_str() const;
size_t copy (char* s, size_t len, size_t pos = 0) const;
string substr (size_t pos = 0, size_t len = npos) const;
```

常用函数 (C++98)

成员函数 (Function)

```
size_t find (const string& str, size_t pos = 0) const;
size_t find (const char* s, size_t pos = 0) const;
size_t find (const char* s, size_t pos, size_t n) const;
size_t find (char c, size_t pos = 0) const;
int compare (const string& str) const;
int compare (size_t pos, size_t len, const string& str) const;
int compare (size_t pos, size_t len, const string& str,
   size_t subpos, size_t sublen) const;
int compare (const char* s) const;
int compare (size_t pos, size_t len, const char* s) const;
int compare (size_t pos, size_t len, const char* s, size_t n) const;
```

- 除此之外，还有 ==、!=、<、<=、>、>= 等运算符，assign、replace、rfind 等成员函数和 getline 的输入方法

问题

- 作为一个程序员，需要熟知上述接口函数，在合适的情况下组合合适的函数
- 作为一名算法学习者，需要了解各函数的具体实现，设计灵活而高效的算法完成任务
- 假设用 char 数组模拟 string 类型，你能实现上述函数吗?

单词统计问题

给定一个英语句子，统计每个单词的出现次数

样例输入

He will go with her, but he will not stay for long.

样例输出

he 2
will 2
...

这是一个定义完整的 (well-defined) 问题吗?

- 大小写问题 (He 和 he)
- 标点符号和空格的作用 (he 和 her)
- 输出的顺序

单词统计问题

在不同的定义下，问题有不同的输出，也有不同的算法

现规定

- 不区分大小写
- 仅以标点符号或空格为单词的间隔
- 按字典序输出所有单词的小写形式

单词统计问题

算法流程

- 把所有单词转成小写

单词统计问题

算法流程

- 把所有单词转成小写
- 用特殊符号将句子划分为单词数组

单词统计问题

算法流程

- 把所有单词转成小写
- 用特殊符号将句子划分为单词数组
- 使用“某种算法”找到不重复的单词列表，统计每个单词的数量，生成形如 [('he',2),('will',2),...] 的结构

单词统计问题

算法流程

- 把所有单词转成小写
- 用特殊符号将句子划分为单词数组
- 使用“某种算法”找到不重复的单词列表，统计每个单词的数量，生成形如 [('he',2),('will',2),...] 的结构
- 对单词进行排序，输出答案

“某种算法”的可能：遍历单词数组，若当前单词未被标记，则标记单词数组里所有与之相同的单词，同时统计数量

时间复杂度 $O(N^2L)$，N 为单词数量，L 为最长单词长度

排序统计算法

- 单词的位置与结果没有关系，与其在最后排序，不如先把单词排序试试

排序统计算法

- 单词的位置与结果没有关系，与其在最后排序，不如先把单词排序试试
- 按字典序将单词排序后，相同的单词会被自动分在一起
- 排序结果为 ['but','for','he','he',...]

排序统计算法

- 单词的位置与结果没有关系，与其在最后排序，不如先把单词排序试试
- 按字典序将单词排序后，相同的单词会被自动分在一起
- 排序结果为 ['but','for','he','he',...]
- 当第 i 个单词和第 i-1 个单词不同（或者 i=1）时，这是一个不重复的单词

排序统计算法

- 单词的位置与结果没有关系，与其在最后排序，不如先把单词排序试试
- 按字典序将单词排序后，相同的单词会被自动分在一起
- 排序结果为 ['but','for','he','he',...]
- 当第 i 个单词和第 i-1 个单词不同（或者 i=1）时，这是一个不重复的单词
- 统计每个单词最早出现的位置 l 和最晚出现的位置 r，此单词的出现次数为 r-l+1

排序统计算法

- 单词的位置与结果没有关系，与其在最后排序，不如先把单词排序试试
- 按字典序将单词排序后，相同的单词会被自动分在一起
- 排序结果为 ['but','for','he','he',...]
- 当第 i 个单词和第 i-1 个单词不同（或者 i=1）时，这是一个不重复的单词
- 统计每个单词最早出现的位置 l 和最晚出现的位置 r，此单词的出现次数为 r-l+1
- 时间复杂度 $O(NlogNL)$

排序统计算法的讨论

- 传统的排序问题中之所以使用快速排序等基于比较的排序算法，是因为单次比较的时间开销很小

排序统计算法的讨论

- 传统的排序问题中之所以使用快速排序等基于比较的排序算法，是因为单次比较的时间开销很小
- 基于比较的排序的比较次数下界为 $O(NlogN)$，每次比较的复杂度为 $O(L)$, 所以在这个思路下算法已经做到最优

排序统计算法的讨论

- 传统的排序问题中之所以使用快速排序等基于比较的排序算法，是因为单次比较的时间开销很小
- 基于比较的排序的比较次数下界为 $O(NlogN)$，每次比较的复杂度为 $O(L)$, 所以在这个思路下算法已经做到最优
- 不基于比较的排序会更适合这类问题吗?

排序统计算法的讨论

- 传统的排序问题中之所以使用快速排序等基于比较的排序算法，是因为单次比较的时间开销很小
- 基于比较的排序的比较次数下界为 $O(NlogN)$，每次比较的复杂度为 $O(L)$, 所以在这个思路下算法已经做到最优
- 不基于比较的排序会更适合这类问题吗?
- 比如：先根据第一个字母排序分类，时间复杂度 $O(N)$; 再分别原地按第二个字母排序，时间复杂度 $O(N)$;...
- 总时间复杂度为 $O(NL)$

排序统计算法的讨论

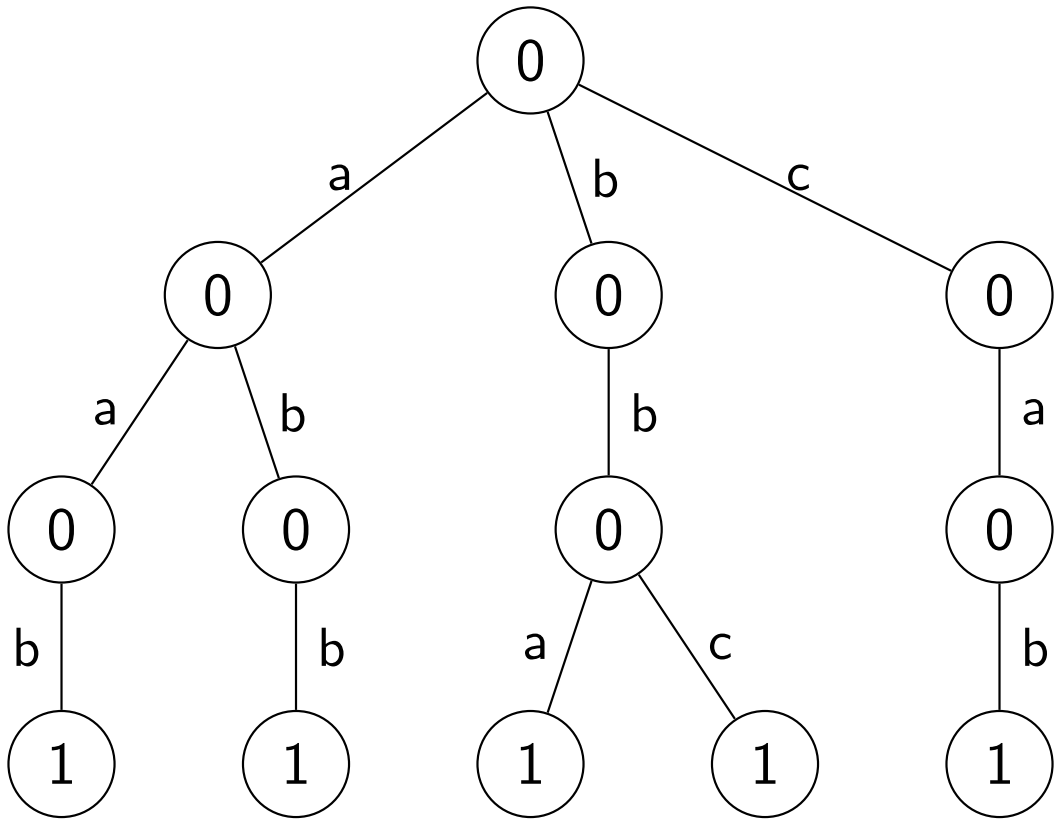
表: 上述算法的执行过程

a	1	2	3	4	5
初始	bbc	cab	abb	aab	bba
排序第一个字母	abb	aab	bbc	bba	cab
排序第二个字母	aab	abb	bbc	bba	cab
排序第三个字母	aab	abb	bba	bbc	cab

原地排序的实现可以参照基数排序算法，也可以使用大小为 26 的链表数组，在这里不作详细介绍，因为我们有更好的做法!

排序统计算法的讨论

更优雅的做法:Trie 树 (字典树)!



上面是描述排序算法分类过程的多叉树，也恰好是表示这些单词的 Trie 树

Trie 树的性质

- 每条边上有一个字母
- 每个结点代表一个字符串，由根节点到它路径上的字母组成
- 每个结点表示某些单词的前缀
- 结点上的数字表示这个字符串的出现次数!
- 从任意结点出发向下走若干步到另一个结点，沿途的字母相接为某个单词的某个区间，如果终点数字大于 0，则是某个单词的后缀

Trie 树的构建

Trie 树的构建不需要太高深的做法，我们以一个例子来说明 Trie 树的构建过程

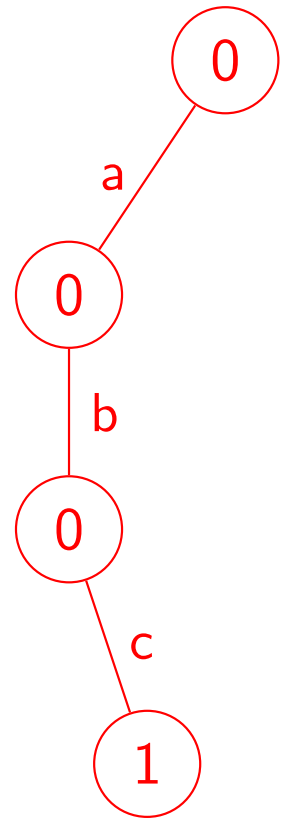
算法流程 (伪代码)

```
for w in 单词数组:
    node = root // root 为 Trie 树的根节点
    for i in 1..L: // L 为 w 的长度
        node = node.son[w[i]] //node.son 为 node 的子结点
    end for
    count[node]++ //count 是每个结点上的数字
end for
```

假设我们的词语数组为 ['abc','bac','aba','ab','bac']

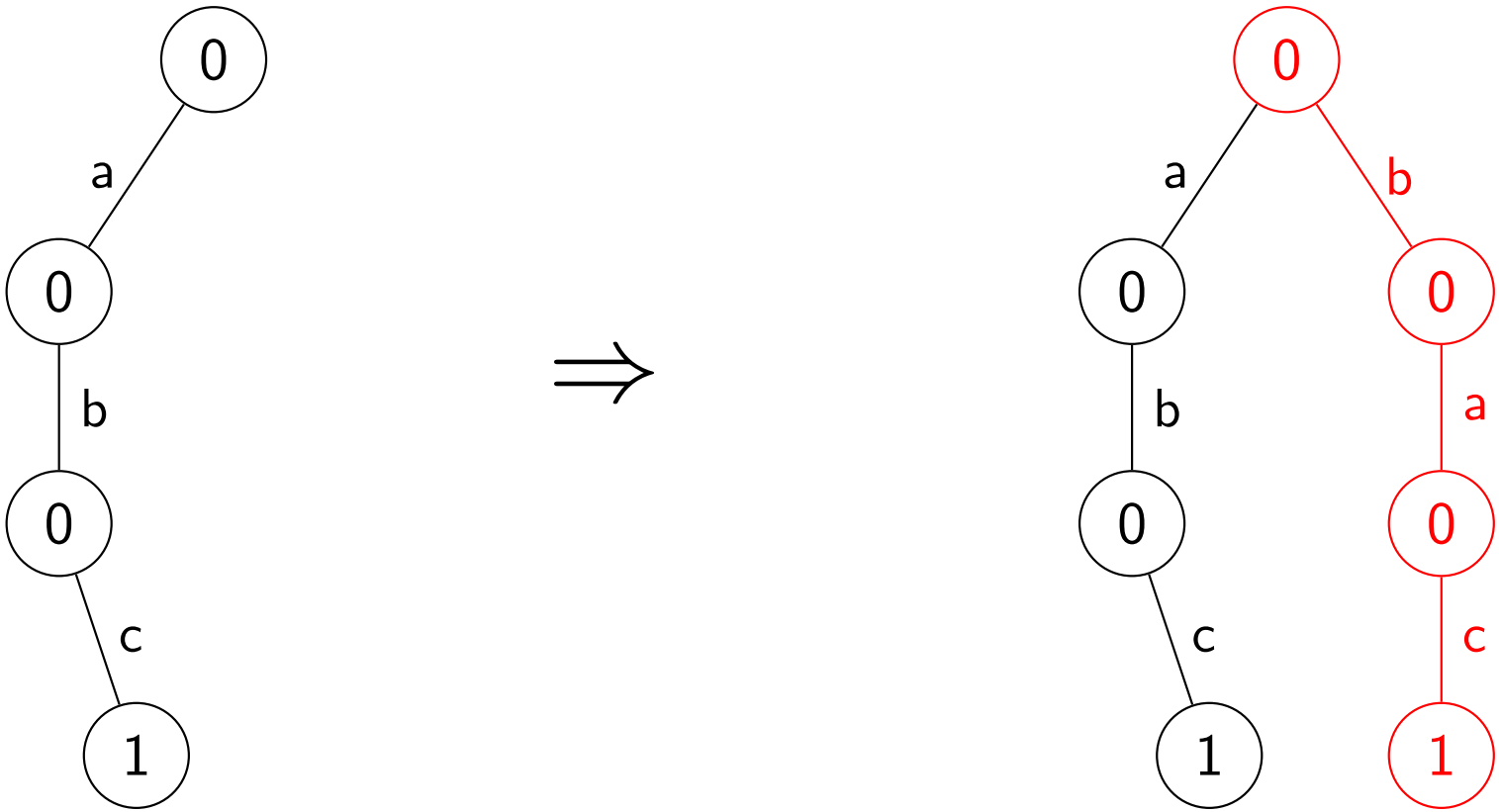
Trie 树的构建

词语数组为 ['abc','bac','aba','ab','bac']



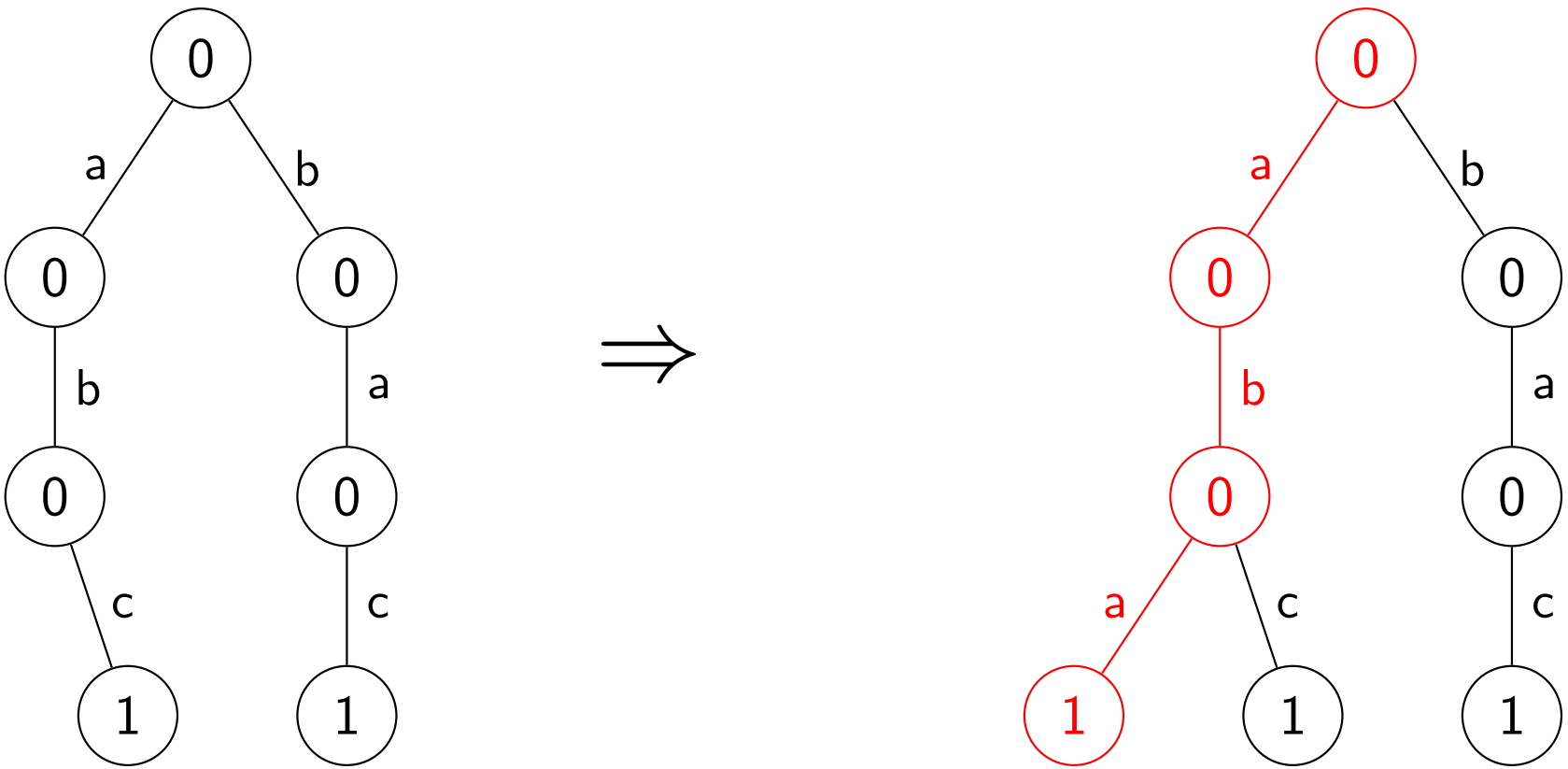
Trie 树的构建

词语数组为 ['abc','bac','aba','ab','bac']



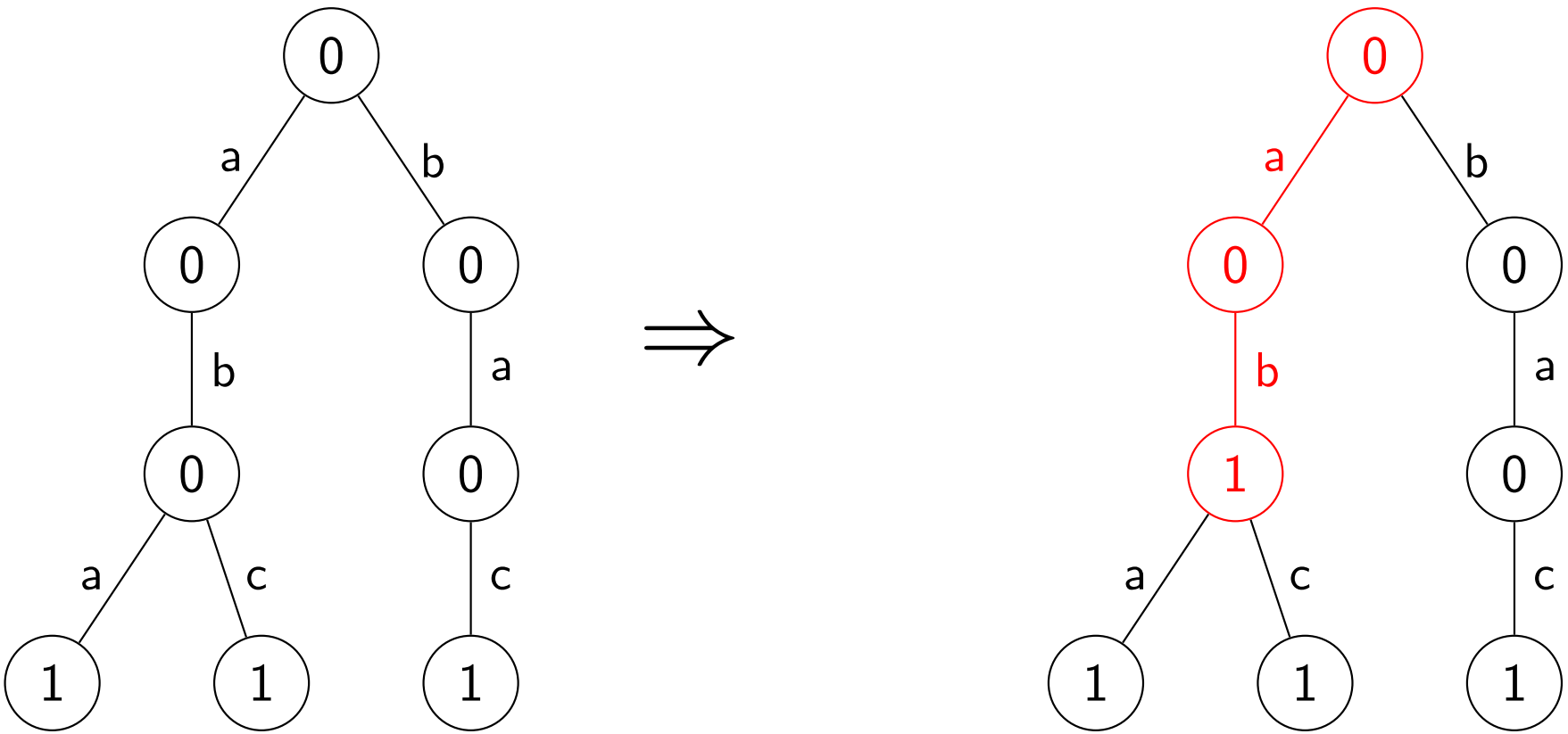
Trie 树的构建

词语数组为 ['abc','bac','aba','ab','bac']



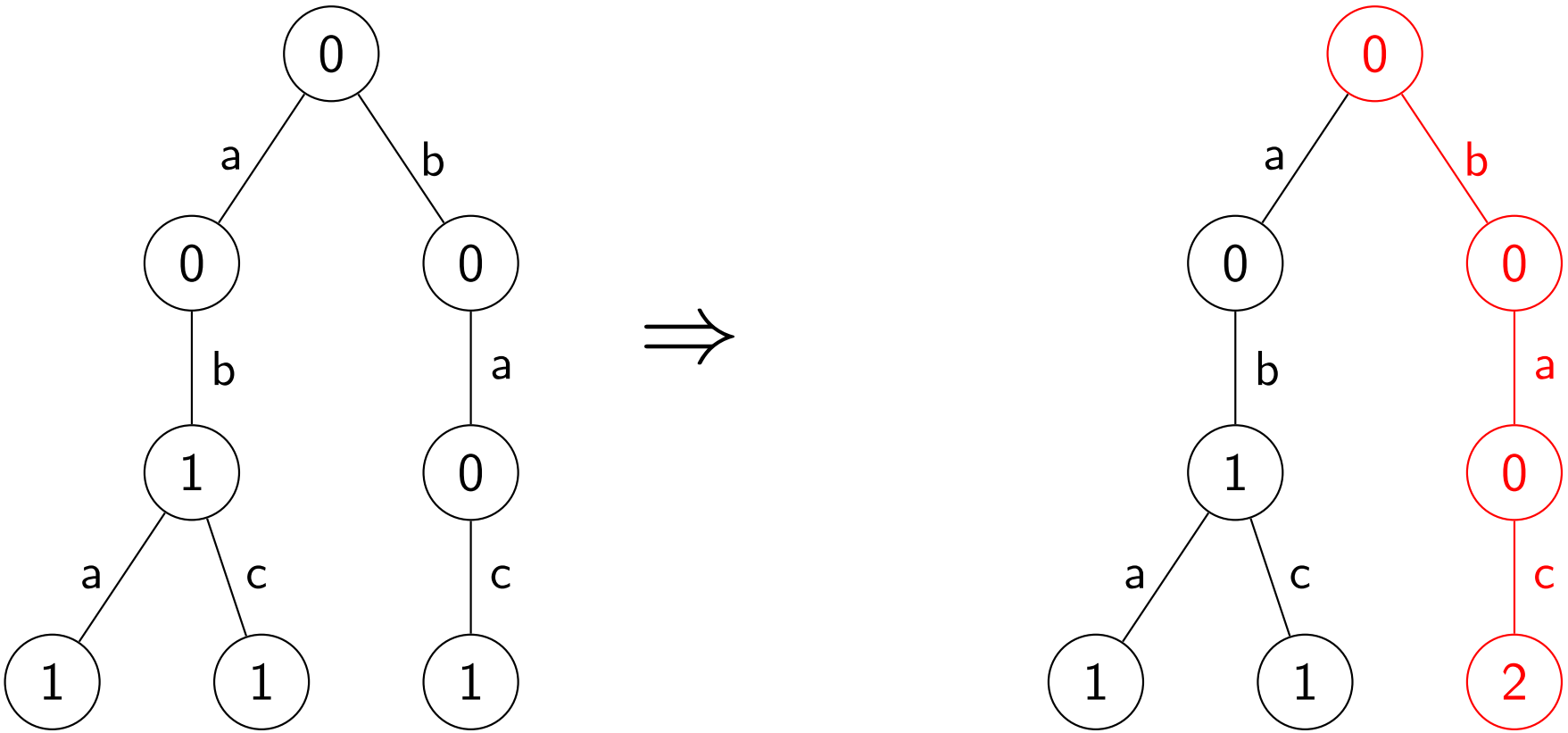
Trie 树的构建

词语数组为 ['abc','bac','aba','ab','bac']

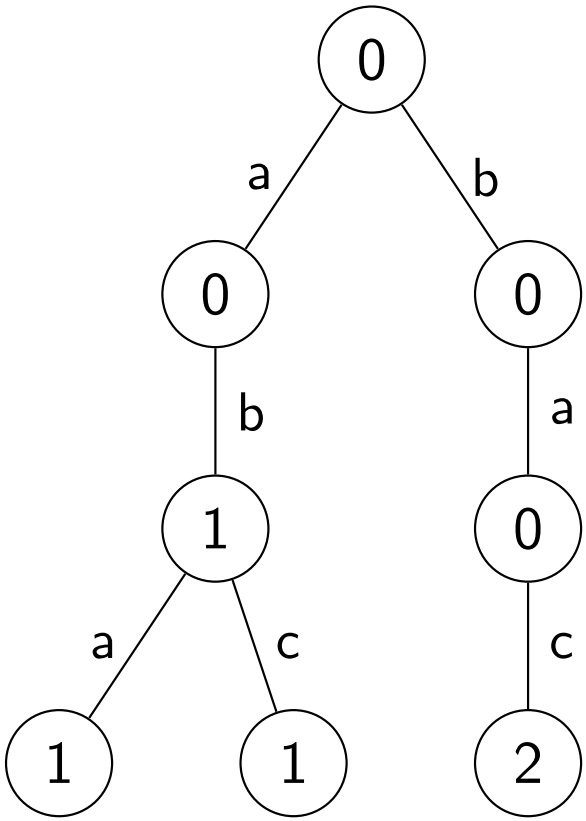


Trie 树的构建

词语数组为 ['abc','bac','aba','ab','bac']

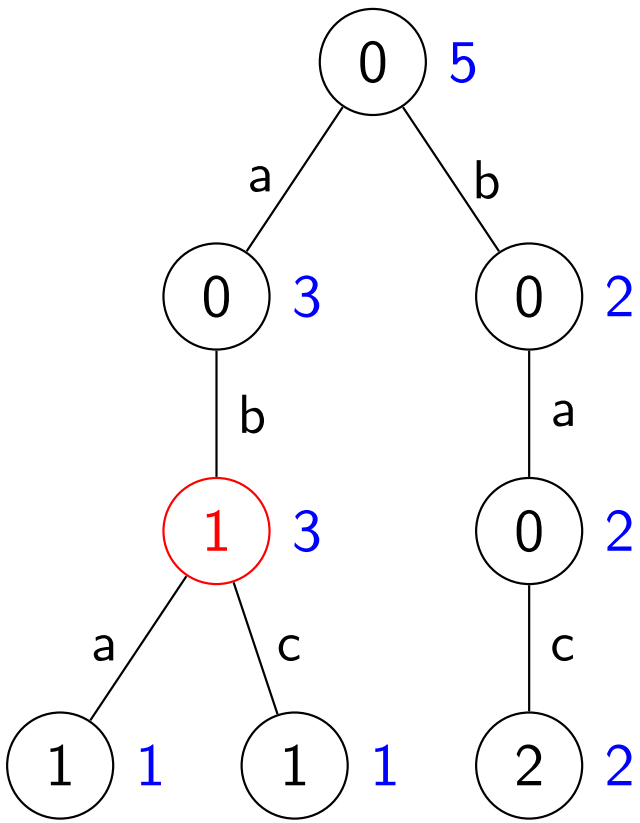


Trie 树的用途



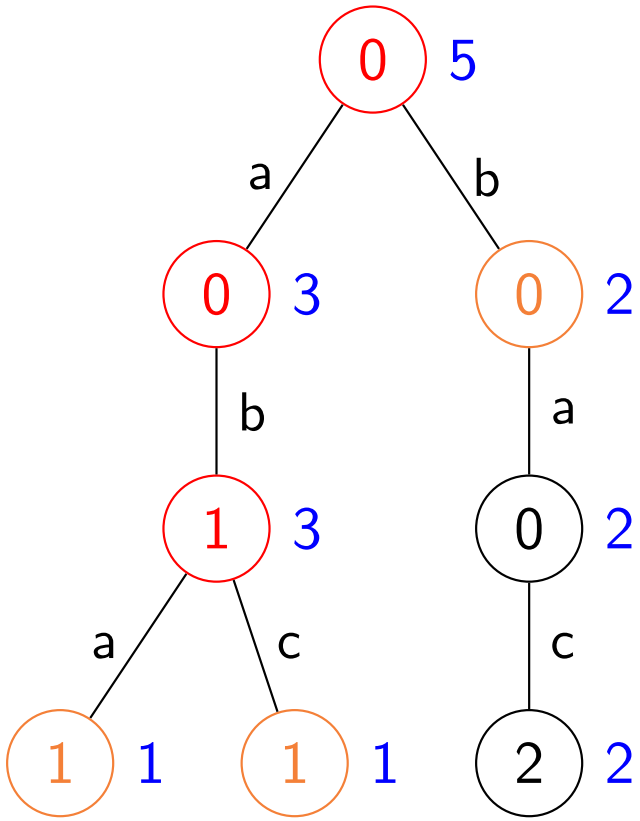
- 在一组单词里查找一个单词
- 对一组单词进行排序
- 统计每个单词的出现次数，时间复杂度 O(NL)
- 统计以一个字符串为前缀的单词的数量
- 统计字典序大于（小于）给定字符串的单词数量

Trie 树的用途



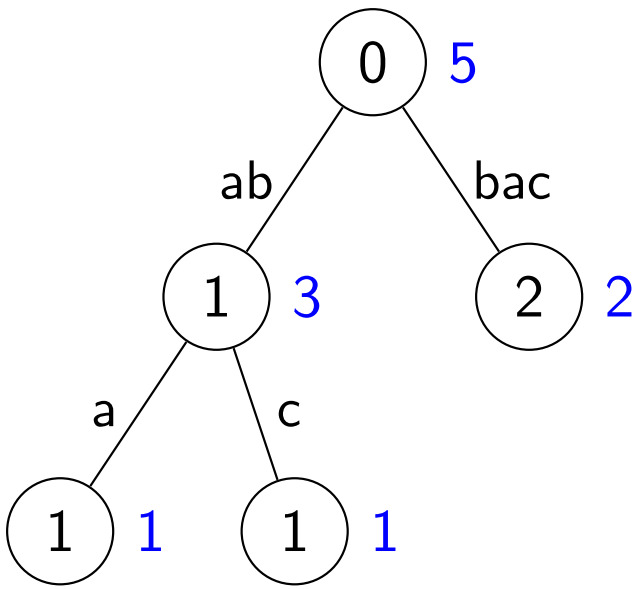
- 假如我们要计算以'ab' 为前缀的字符串数量
- 我们可以通过对树 DFS，以 O(N) 的时间复杂度计算所有结点的后代的词语数量（如蓝色所示）
- 其实在建树的过程中就可以算出所有结点的后代的单词数量，要怎么做?
- 表示'ab' 的结点如红色所示，它后代的词语数量 3 就表示了以'ab' 为前缀的字符串数量

Trie 树的用途



- 假如我们要计算字典序大于'ab' 的字符串数量
- 字典序大于'ab' 的字符串一定在标红结点的“右侧”或“下侧”，将橙色结点后代的单词数量相加即可
- 字典序大于'aba' 的字符串有几个?
- 字典序大于'abb' 的字符串有几个?

Trie 树的变种



- 处理的对象不一定是字符串：以 0∼9 代替 a∼ z，可以得到一个不基于比较的对正整数的排序算法，时间复杂度 O(NA)，A 为正整数位数，通常 A>logN
- 边上的标识不一定是字符：若允许边上的标识为字符串，我们可以得到一个更加“压缩”的 Trie 树，如左图所示

Trie 树的实现

```
struct node{
  int son[26],count;//son 数组初始化为 0
} nodes[N*26];
int root=0,nodecount=1;
void insert(const char* s,int len){
  int now=root;
  for(int i=0;i<len;i++){
    v=(int)(s[i]-'a');
    if(nodes[now].son[v]==0)
      nodes[now].son[v]=nodecount++;
    now=nodes[now].son[v];
  }
  nodes[now].count++;
}
```

例题讲解-[TJOI2010] 阅读理解

- 有 N(N<=100) 篇短文，每篇短文有若干个单词，短文长度不超过 S(S<=5000)，单词长度不超过 L(L<=20)
- 有 M(M<=10000) 个单词，求每个单词在哪些短文中出现过

例题讲解-[TJOI2010] 阅读理解

- 有 N(N<=100) 篇短文，每篇短文有若干个单词，短文长度不超过 S(S<=5000)，单词长度不超过 L(L<=20)
- 有 M(M<=10000) 个单词，求每个单词在哪些短文中出现过
- 做法一：对每篇短文分别建立一棵 Trie 树，然后对每个单词，进入这 N 个 Trie 树种查找，时间复杂度 O(NS+NML)

例题讲解-[TJOI2010] 阅读理解

- 有 N(N<=100) 篇短文，每篇短文有若干个单词，短文长度不超过 S(S<=5000)，单词长度不超过 L(L<=20)
- 有 M(M<=10000) 个单词，求每个单词在哪些短文中出现过
- 做法一：对每篇短文分别建立一棵 Trie 树，然后对每个单词，进入这 N 个 Trie 树种查找，时间复杂度 O(NS+NML)
- 做法二：对所有短文建立同一棵 Trie 树，每个结点代表了一个单词，记录提及它的短文编号（可用链表），在查询时找到对应结点，将编号取出即可，时间复杂度 O(NS+M(L+N))

例题讲解-[USACO08DEC] 秘密消息 Secret Message

- 有 M(M<=50000) 条二进制信息正被加密传输，第 i 条信息的前 $b_i(b_i<=10000)$ 位被拦截
- 对这些信息，有 N(N<=50000) 条猜测，第 j 条猜测的前 $c_j(c_j<=10000)$ 位已知
- 对每条猜测，求它有可能是哪些信息，输出可能的数量
- $\sum b_i + \sum c_j <= 500000$

样例描述

已知信息的前缀为 [010，1，100，110]
猜测的前缀为 [0，1，01，01001，11]
每个猜测的答案分别为 [1，3，1，1，2]

例题讲解-[USACO08DEC] 秘密消息 Secret Message

- 猜测的前缀 C 可能是信息的前缀 B 的充分必要条件是：C 是 B 的前缀或者 B 是 C 的前缀
- 问题转化为对每条猜测，求有多少条信息满足上述条件

例题讲解-[USACO08DEC] 秘密消息 Secret Message

- 猜测的前缀 C 可能是信息的前缀 B 的充分必要条件是：C 是 B 的前缀或者 B 是 C 的前缀
- 问题转化为对每条猜测，求有多少条信息满足上述条件
- 可以将所有信息建一棵二进制 Trie 树
- 对每条猜测，找到它在 Trie 树上对应的结点 v，v 后代的单词数量 + 根结点到 v 路径上的单词数量即为答案

例题讲解-[USACO12DEC] 第一!First!

- 有 N(N<=30000) 个由小写字母组成的字符串
- 现在可以任意排列字母的大小顺序（如原本规定 a<b<c<...<z，现在可以规定比如 b<a<f<z<t<...），这样的排列方案数为 26!
- 对每种方案，我们有字典序最小的字符串，现在求有哪些字符串在排列字母大小顺序后可能成为字典序最小的字符串
- 字符总数不超过 300000

例题讲解-[USACO12DEC] 第一!First!

- 此问题涉及字符串的比较，考虑用 Trie 求解
- 在 a<b<c<..<z 的情况下，如何找到 Trie 树上字典序最小的字符串?

例题讲解-[USACO12DEC] 第一!First!

- 此问题涉及字符串的比较，考虑用 Trie 求解
- 在 a<b<c<..<z 的情况下，如何找到 Trie 树上字典序最小的字符串?
- 从根结点出发，贪心地找后代边上里最小的字母作为下一步，遇到有字符串的结点就停止

例题讲解-[USACO12DEC] 第一!First!

- 此问题涉及字符串的比较，考虑用 Trie 求解
- 在 a<b<c<..<z 的情况下，如何找到 Trie 树上字典序最小的字符串?
- 从根结点出发，贪心地找后代边上里最小的字母作为下一步，遇到有字符串的结点就停止
- 使一个字符串成为字典序最小的字符串，需要什么条件?

例题讲解-[USACO12DEC] 第一!First!

- 此问题涉及字符串的比较，考虑用 Trie 求解
- 在 a<b<c<..<z 的情况下，如何找到 Trie 树上字典序最小的字符串?
- 从根结点出发，贪心地找后代边上里最小的字母作为下一步，遇到有字符串的结点就停止
- 使一个字符串成为字典序最小的字符串，需要什么条件?
- 从根结点出发，到表示这个字符串的路径上，每条边都是贪心地选择的

例题讲解-[USACO12DEC] 第一!First!

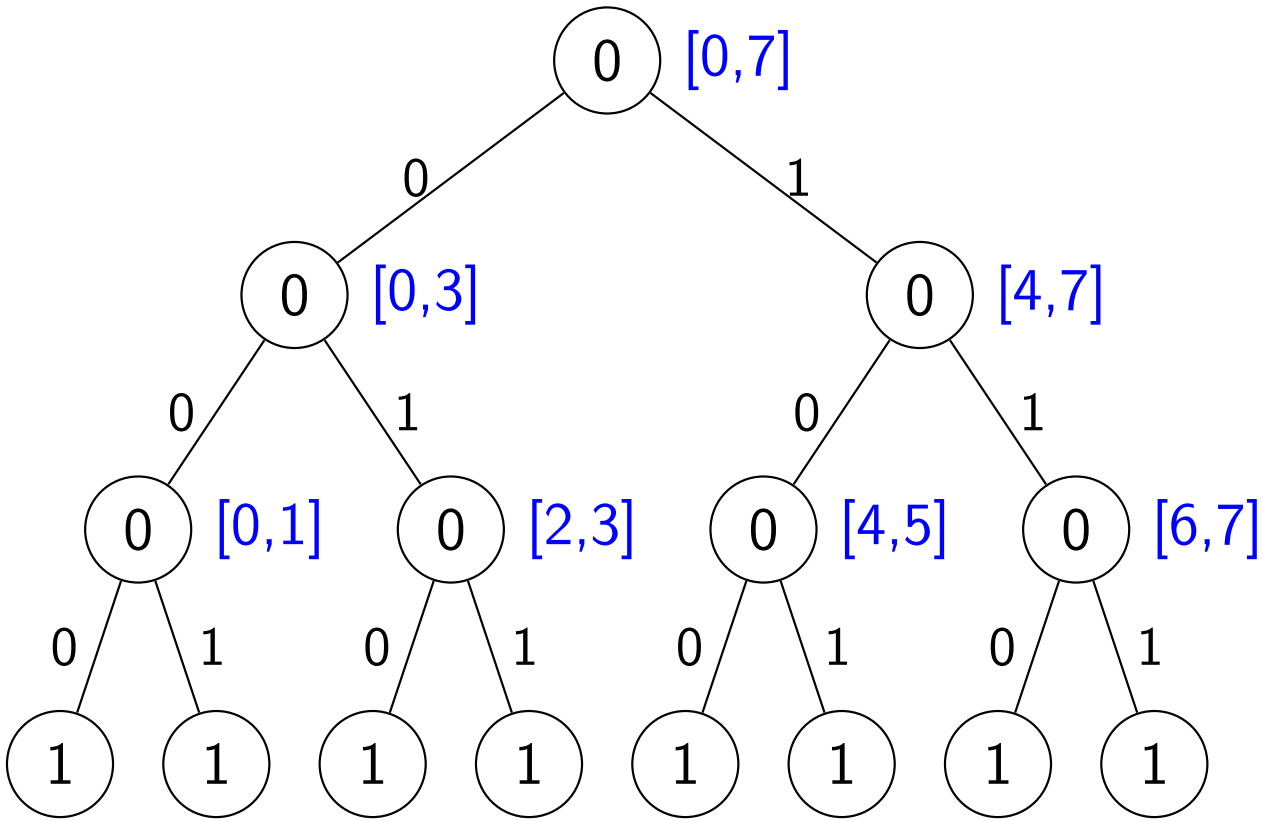
- 此问题涉及字符串的比较，考虑用 Trie 求解
- 在 a<b<c<..<z 的情况下，如何找到 Trie 树上字典序最小的字符串?
- 从根结点出发，贪心地找后代边上里最小的字母作为下一步，遇到有字符串的结点就停止
- 使一个字符串成为字典序最小的字符串，需要什么条件?
- 从根结点出发，到表示这个字符串的路径上，每条边都是贪心地选择的
- 假设当前结点有 a,b,c 三条后代边，选择了 b，那么一定要满足两个不等式 b<a 和 b<c
- DFS 遍历整个 Trie 树，以贪心策略到达每个结点需要满足一系列不等式，不等式建立的有向图可以在 DFS 过程中维护。到达字符串结点时，如果不成环，就一定能找到一个合适的顺序，使它字典序最小；反之若成环，则此字符串不可能字典序最小

例题讲解-[USACO12DEC] 第一!First!

- 此问题涉及字符串的比较，考虑用 Trie 求解
- 在 a<b<c<..<z 的情况下，如何找到 Trie 树上字典序最小的字符串?
- 从根结点出发，贪心地找后代边上里最小的字母作为下一步，遇到有字符串的结点就停止
- 使一个字符串成为字典序最小的字符串，需要什么条件?
- 从根结点出发，到表示这个字符串的路径上，每条边都是贪心地选择的
- 假设当前结点有 a,b,c 三条后代边，选择了 b，那么一定要满足两个不等式 b<a 和 b<c
- DFS 遍历整个 Trie 树，以贪心策略到达每个结点需要满足一系列不等式，不等式建立的有向图可以在 DFS 过程中维护。到达字符串结点时，如果不成环，就一定能找到一个合适的顺序，使它字典序最小；反之若成环，则此字符串不可能字典序最小
- 不等式判环采用拓扑排序，时间复杂度为 $O(26^2 * N)$

Trie 树和线段树的关系

考虑一个存储 [000,001,010,011,100,101,110,111] 的 Trie 树



Trie 树和线段树的关系

- 每个 01 串的前缀都可以由它的祖先表示，反过来看，每个结点表示它后代字符串的前缀

Trie 树和线段树的关系

- 每个 01 串的前缀都可以由它的祖先表示，反过来看，每个结点表示它后代字符串的前缀
- 用十进制而不是二进制看这些 01 串，它们恰好分别表示 [0,7]！
- Trie 树中每个结点的后代的数字恰为一个区间！

Trie 树和线段树的关系

- 每个 01 串的前缀都可以由它的祖先表示，反过来看，每个结点表示它后代字符串的前缀
- 用十进制而不是二进制看这些 01 串，它们恰好分别表示 [0,7]！
- Trie 树中每个结点的后代的数字恰为一个区间！
- 把一个区间（区间长度为 2 的幂）的数字转为二进制，再建立 Trie 树，我们就建立了这个区间的线段树！

Trie 树和线段树的关系

- 每个 01 串的前缀都可以由它的祖先表示，反过来看，每个结点表示它后代字符串的前缀
- 用十进制而不是二进制看这些 01 串，它们恰好分别表示 [0,7]！
- Trie 树中每个结点的后代的数字恰为一个区间！
- 把一个区间（区间长度为 2 的幂）的数字转为二进制，再建立 Trie 树，我们就建立了这个区间的线段树！
- 事实上，即使区间长度不为 2 的幂，我们建立的 Trie 树也是一棵平衡的树，线段树的任何操作仍为正确的复杂度，只是需要考虑特殊情况
- 如果感兴趣的话，可以了解 zkw 线段树

Thanks for listening

Any Questions?