字典树

原理

- Int ch[sz][26];

- 每个节点有字符集个儿子，相同前缀的字符串会共用公共的前缀内存

- 我们可以利用字典树来对字符串进行高效的哈希

插入

```
char s[N];
int ch[N][26];
int sz;

void init () {
    memset(ch[0], 0, sizeof(ch[0]));
    sz = 1;
}

void insert(char *s) {
    int p = 0;
    for (int i = 0; s[i]; i++) {
        int c = s[i] - 'a';
        if (!ch[p][c]) {
            memset(ch[sz], 0, sizeof(ch[sz]));
            ch[p][c] = sz++;
        }
        p = ch[p][c];
    }
    // p 是字符串s的尾节点，通常可以在此处做一些标记
}
```

查找

- 与插入同理，沿着查询的字符串在字典树上往下走，如果不能走了就表示没有要查找的字符串

应用一

- 求多少个字符串包含特定前缀

应用二

- 给你10w个数a[ ]，然后给你10w个询问S，求a数组里面与S异或起来最大的数K

- 每个整数都是一个01串，从高位到低位依次插入字典树，查询的时候贪心选择即可

应用三

- 一棵树，每个点有点权， 每次询问x y z，求z与x到y路径上的权值的最大异或和

- 可持久化字典树，提取路径上的信息
- 然后跟应用二一样

什么是可持久化

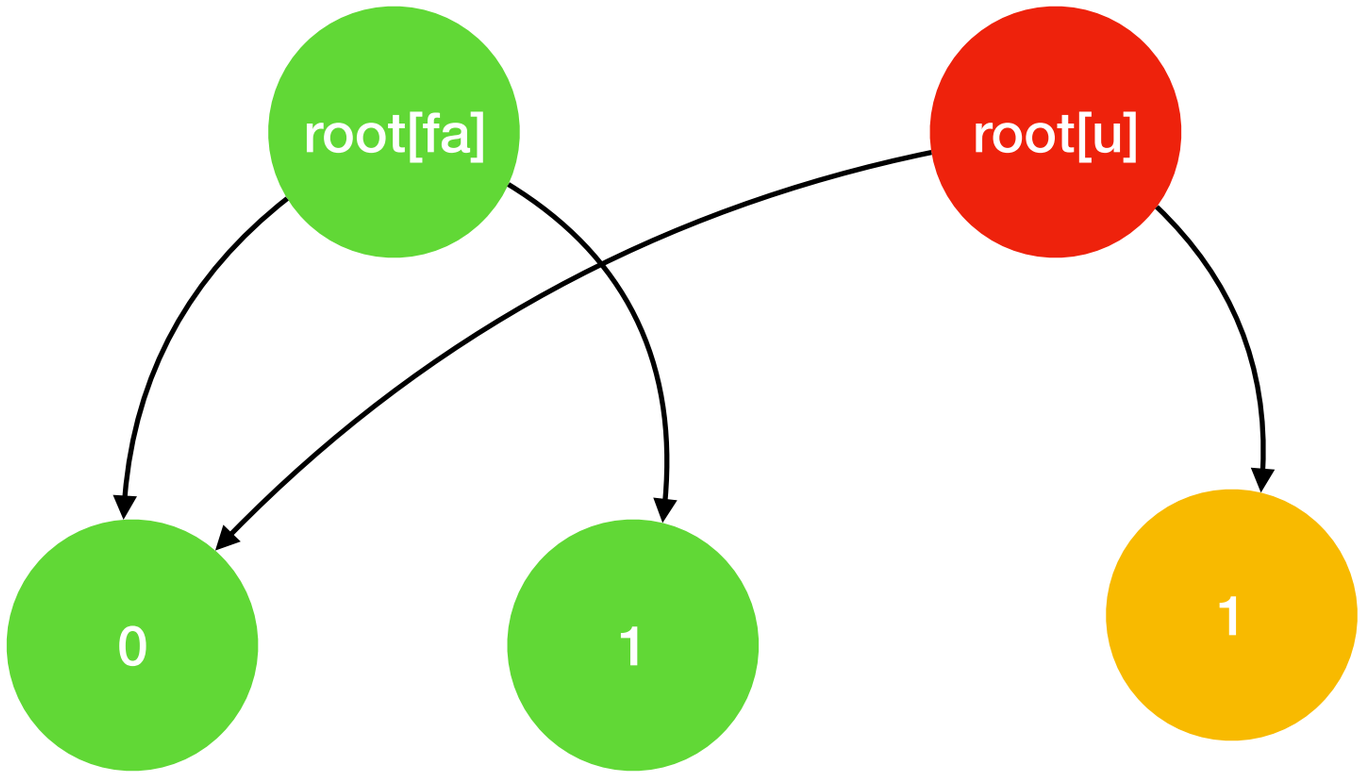
- 不修改历史信息
- 只在之前的信息上增加信息
- 利用空间换时间
- 每个节点维护到根的字典树信息，路径上的字典树信息可以通过x 与 y的信息减去lca的信息即可

具体写法

- 假设父亲的字典树根为root[fa],那么当前点u的字典树可以通过root[u] = insert(root[fa])得到，每插入进一个子树，都新建一个节点，每插入一个字符串就要新建字符串长度个节点。

```
int newnode(int value) {
    memset(ch[sz], 0, sizeof(ch[sz]) );
    sum[sz] = value;
    return sz++;
}

int insert(int x, int val, int dep)
{
    int rt = newnode(sum[x]+1);
    if(dep == -1) return rt;
    int d = (val>>dep) & 1;
    ch[rt][d^1] = ch[x][d^1];
    ch[rt][d] = insert(ch[x][d], val, dep-1);
    return rt;
}
```



插入一个字符串1
新建一个root[u],因为要插入1儿子里面去，所以递归新建一个1儿子，然后root[u]的0儿子复用root[fa]的0儿子，root[u]的1儿子指向新建好的1儿子