字符串算法第1节

“最小表示法”与“字符串hash”

重庆南开信竞基础课程

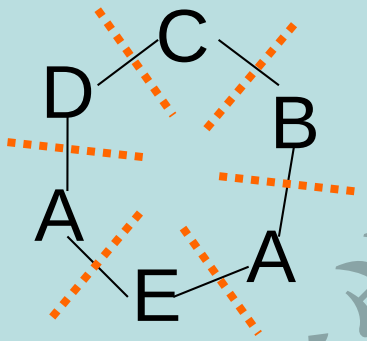
by helang

一.字符串的最小表示

什么是**字符串的最小表示**?

1.把一个长为len的字符串首尾相连围成一个圈；
2.然后以任意一个字符作为起点，顺时针数len个字符，都会产生一个新的长为len的字符串。总共有len个新字符串；
3.字符串的最小表示就是所有新字符串中字典序最小的那个；

C B A E A D



C B A E A D

B A E A D C

A E A D C B

E A D C B A

A D C B A E

D C B A E A

字符串A D C B A E 就是 字符串C B A E A D 的最小表示

求字符串的最小表示的方法

字符串 s= " C C C D C A E B "　　len=8

s=s+s=

C C C D C A E B C C C D C A E B

i　j　i　　ji ji j　j　j

用整型变量k来记录已经讨论的字符串长度

k =					
0	s[i+k]==s[j+k]	k++		CCC	i
1	s[i+k]==s[j+k]	k++			
2	s[i+k] < s[j+k]	j=j+k+1		CCD	j
0	s[i+k]==s[j+k]	k++			
1	s[i+k] > s[j+k]	i=i+k+1		CC	i
0	s[i+k]==s[j+k]	k++			
1	s[i+k] > s[j+k]	i=i+k+1	i==j	CA	j
0	s[i+k] > s[j+k]	i=i+k+1	i==j	j++	
0	s[i+k] < s[j+k]	j=j+k+1			
0	s[i+k] < s[j+k]	j=j+k+1			

判断从i开始的长度为len的字符串与从j开始的长度为len的字符串的字典序大小

从j开始的字符串字典序一定比从i开始的大。要找比i开始的字符串更小的，j就必须从新找一个位置为起点。

显然，从j到j+k之间任意位置为起点的字符串都要比从i开始的字符串大，所以j的值要该为j+k+1才可能找到更小的。

然后重新判断分别以i和j为起点的两个字符串的大小

j>len 程序结束，以min(i,j)为起点的长度为len的字符串就是所求最小表示

最小表示法的代码

```
cin>>s;
L=s.length();
s=s+s;
i=0;     j=1;
while(i<L && j<L)
{
                for(k=0;k<L;k++)
                if(s[i+k]!=s[j+k])break;
                if(k==L)break;
                if(s[i+k]>s[j+k])i=i+k+1;
                else if(s[i+k]<s[j+k])j=j+k+1;
                if(i==j)j++;
}
cout<<min(i,j)<<endl;
```

时间复杂度O(n)

习题：1226，2973

二.字符串的Hash

工商注册 NKOJ5494

你在工商局上班。

已经有n家公司注册了名字。你的任务是处理接下来的m个注册申请。每个申请操作会告知你一个由字母构成的公司名，你的工作是查询该名字是否已经被注册，若没有，则将其注册，并告诉客户"YES"。若该名字已经被注册了，直接告诉客户"NO"。

公司名字都有小写字母构成，且长度不超过30

0<=n<=50000

1<=m<=30000

我们考虑将每个单词（字符串）映射成一个“唯一”的数字。

我们申明一个数组`bool mark[100000]`，来记录每个出现过的单词。

假设，对于单词“nkoi”，我们将其转换成数字x，我们将`mark[x]=true`即可。

我们构造的转换函数称为**Hash函数**。

通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数(hsah函数)，存放记录的表叫做hash表(散列表)。——百度百科

BKDRhash

字符串hash模板，BKDR_Hash

```
bool mark[524288];
unsigned int BKDRHash(string str)          //处理字符串
{
    unsigned int seed = 131;                // hash种子，一般取质数：31,131,1313,13131,131313 等.
    unsigned int hash = 0, i=0, Len=str.length();
    while (i<Len) hash = hash * seed + str[i++];   //计算hash值
    return (hash & 0x7FFFF);                //0x7FFFF为十六进制，==524287=2^19-1
                                            //保证最后的hash值在0到524287之间
   //如果想要最后的hash值在1000000之间，就return(hash % 1000000)
}

int main()       //输入10000个名字，统计其中不同的名字的个数
{
      int i,x,cnt=0;
      string a;
      for(i=1;i<=10000;i++)
      {
           cin>>a;
           x=BKDRHash(a);
           if(mark[x]==false){cnt++; mark[x]=true;}
      }
      cout<<cnt;
}
```

字符串hash模板，BKDR_Hash

```
bool mark[524288];
unsigned int BKDRHash(string str)          //处理字符串
{
    unsigned int seed = 131;               // hash种子，一般取质数：31,131,1313,13131,131313 等.
    unsigned int hash = 0, i=0, Len=str.length();
    while (i<Len) hash = hash * seed + str[i++];   //计算hash值
    return (hash & 0x7FFFF);               //0x7FFFF为十六进制，==524287=2^19-1
                                           //保证最后的hash值在0到524287之间

}
```

可能出现”冲突”,两个字符串算出的hash值相同，怎么办?

可以考虑双hash，例如：

将BKDRHash1()种子设为1313，将BKDRHash2()种子设为233。

对于某个字符串s，假设x=BKDRHahs1(s)，y=BKDRHash2(s)

若mark1[x]==true 并且 mark2[y]==true，我们才认为s已出现过。

前缀和Hash
(区间Hash)

对一个字符串进行哈希操作：

```
char S[100000];
unsigned long long Hash[100005];     //Hash[i]记录S的[1,i]这段子串的哈希值
Scanf("%s",S+1);

int P = 233;                         //哈希种子P，取质数
void getHash()
{
      int len = strlen(S+1);
      for(int i = 1; i <= len; i++) Hash[i] = Hash[i - 1] * P + S[i] - 'a';
}
```

上面哈希的好处是，我们可以快速获得任意一个子串的哈希值。
比如，我们想要得到S的[L,R]这段子串的哈希值：

```
long long getSubHash(int L, int R)
{
      return Hash[R] - Hash[L - 1] * P^(R - L + 1);
}
```

注意一点：
Hash值储存在unsigned long long里面， 那样溢出时，会自动取余2的64次方，but这样可能会使2个不同串的哈希值相同，但这样的概率极低（不排除你的运气不好）

举个例子：
设整数A[i]为字符S[i]转换成的数字,即S[i]-'a'
Hash[i]表示S[1,i]的哈希值，这里我们把它写成Hash[1,i]

Hash[1]= **A[1]**
Hash[2]= **A[1]*P + A[2]**
Hash[3]= (A[1]*P + A[2])*P + A[3] = **$A[1]*P^2 + A[2]*P + A[3]$**
Hash[4]= (A[1]*P + A[2])*P + A[3])*P + A[4] = **$A[1]*P^3 + A[2]*P^2 + A[3]*P + A[4]$**
Hash[5]= **$A[1]*P^4 + A[2]*P^3 + A[3]*P^2 + A[4]*p^1 + A[5]$**
……
Hash[K]= **$A[1]*P^{K-1} + A[2]*P^{K-2} + A[3]*P^{K-2} + … + A[K]*P^{K-K}$**

我们要求S的[2,4]这段区间的哈希值：
即Hash[3,5]= $A[3]*P^{5-3} + A[4]*P^{5-4} + A[5]*P^{5-5}$ = **$A[3]*P^2 + A[4]*P^1 + A[5]$**
Hash[5]= Hash[3,5] + $A[1]*P^4 + A[2]*P^3$
= Hash[3,5] + (**A[1]*P + A[2]**)$*P^3$
= Hash[3,5] + Hash[2]$*P^3$
于是可以得到：
Hash[3,5] = Hash[1,5] – Hash[1,2]$*P^3$
即Hash[5] – Hash[2]$*P^3$

我们可以以此理解**Hash[L,R]= Hash[R] – Hash[L-1]$*P^{R-L+1}$**

字符串hash习题：1809, 3708

附：树的最小表示法

NKOJ 3458

题目大意：给了两个字符串作为有根树的遍历，向下为0，向上为1，然后判断两棵树是否同构。

分析：

从根节点出发，最终还要回到根节点。对树的每条边，有去必有回，去为0，回为1。显然，遍历整个树会得到一个串，而这个串里面的0和1个数相等。

由递归性可知，遍历子树得到的串，0和1也相等。每棵子树中0和1的数量相同。也就是说，遍历的整个过程就是得到许多01相等的串。

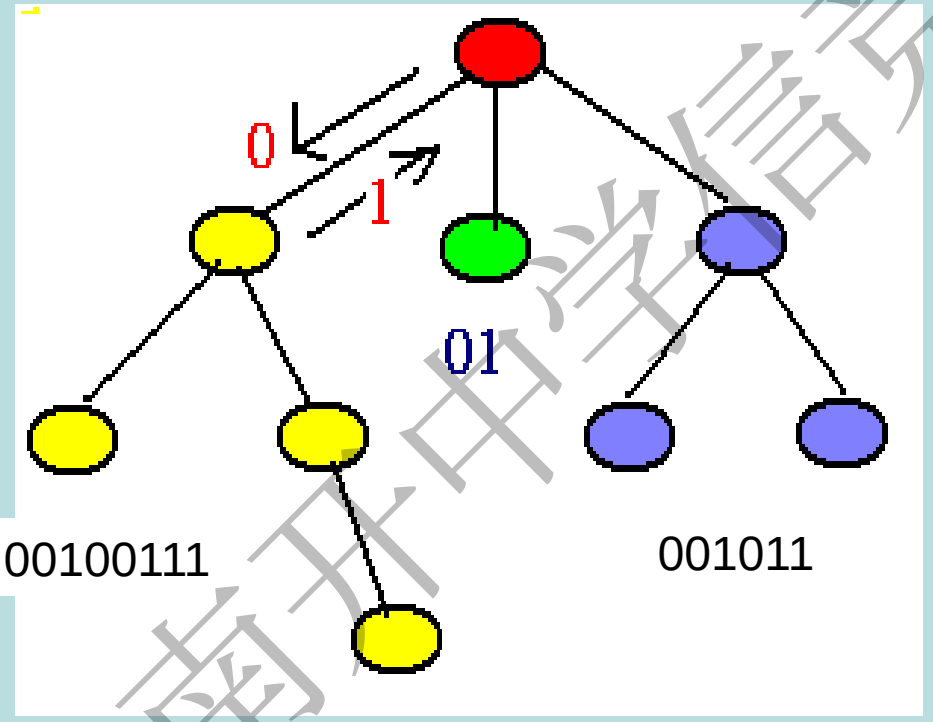
题目中所给的串恰由这些0和1数量相同的子串构成。这样，如果对这些子串按字典顺序排序，原串就可以变为另外一种形式。

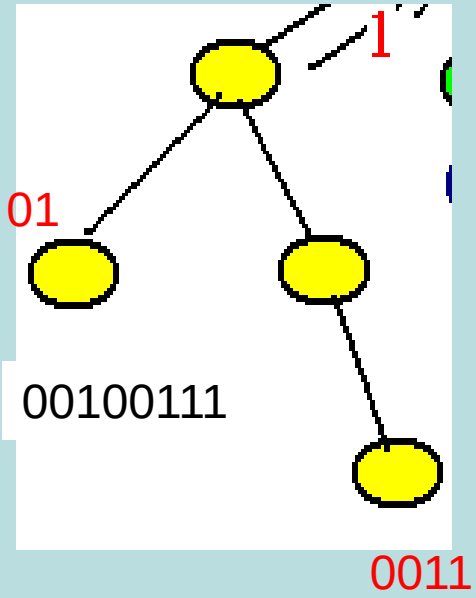
原串：0010011101001011

子串：00100111　　01　　001011

该树的最小表示：

0010011　　001011　　01





```
#include<cstdio>
#include<string>
#include<vector>
#include<algorithm>
using namespace std;
string a;
string dfs(int i){
    vector<string>v;
    int n=a.size();
    while(i<n && a[i]!='1'){
        v.push_back('0'+dfs(i+1));
        i+=v.back().size();
    }
    string r;
    sort(v.begin(),v.end());
    for(int k=0;k<v.size();k++)   r+=v[k];
    return r+'1';
}
int main(){
        cin>>a;
        string aa=dfs(0);
        scanf("%s",s);
        cin>>a;
        string bb=dfs(0);
        if(aa==bb)  printf("same\n");
        else  printf("different\n");
}
```