哈希与字符串

Hash 的核心思想在于，将输入映射到一个值域较小、可以方便比较的范围。

我们定义一个把字符串映射到整数的函数 f，这个 f 称为是 Hash 函数。

具体来说，我们希望在 Hash 函数值不一样的时候，两个字符串一定不一样。

通常我们采用的是多项式 Hash 的方法，即

$$f(s)=\sum s[i]\times b^i \pmod M$$

这里面的 b 和 M 需要选取得足够合适才行，以使得 Hash 函数的值分布尽量均匀。

如果 b 和 M 互质，在输入随机的情况下，这个 Hash 函数在 $[0,M)$ 上每个值概率相等。

令 $f_i(s)$ 表示 $f(s[1..i])$，那么 $f(s[l..r])=\frac{f_r(s)-f_{l-1}(s)}{b^{l-1}}$，其中 $\frac{1}{b^{l-1}}$ 也可以预处理出来。

使用 unsigned long long来表示$f(s)$， unsigned long long表示无符号数，也就是所表示的数全部为正数，这样在正数范围内表示的范围比long long更大。

1455：【例题1】Oulipo

```
#include<bits/stdc++.h>
using namespace std;
int T;
int n,m;
int ans;
char s1[10000+10],s2[1000000+10];
unsigned long long power[1000000+10];
unsigned long long H[1000000+10];
unsigned long long s,b=27,h=1<<31;
int main(){
    power[0]=1;
    for(int i=1;i<=1000000;i++)
        power[i]=power[i-1]*b;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%s%s",s1+1,s2+1);
        n=strlen(s1+1);m=strlen(s2+1);
        H[0]=0;
        for(int i=1;i<=m;i++)
            H[i]=(H[i-1]*b+(unsigned long long)(s2[i]-'A'+1))%h;
```

```
        s=0;
        for(int i=1;i<=n;i++)
            s=(s*b+(unsigned long long)(s1[i]-'A'+1))%h;
        ans=0;
        for(int i=0;i<=m-n;i++)
            if(s==H[i+n]-H[i]*power[n])
                ans++;
        printf("%d\n",ans);
    }
    return 0;
}
```

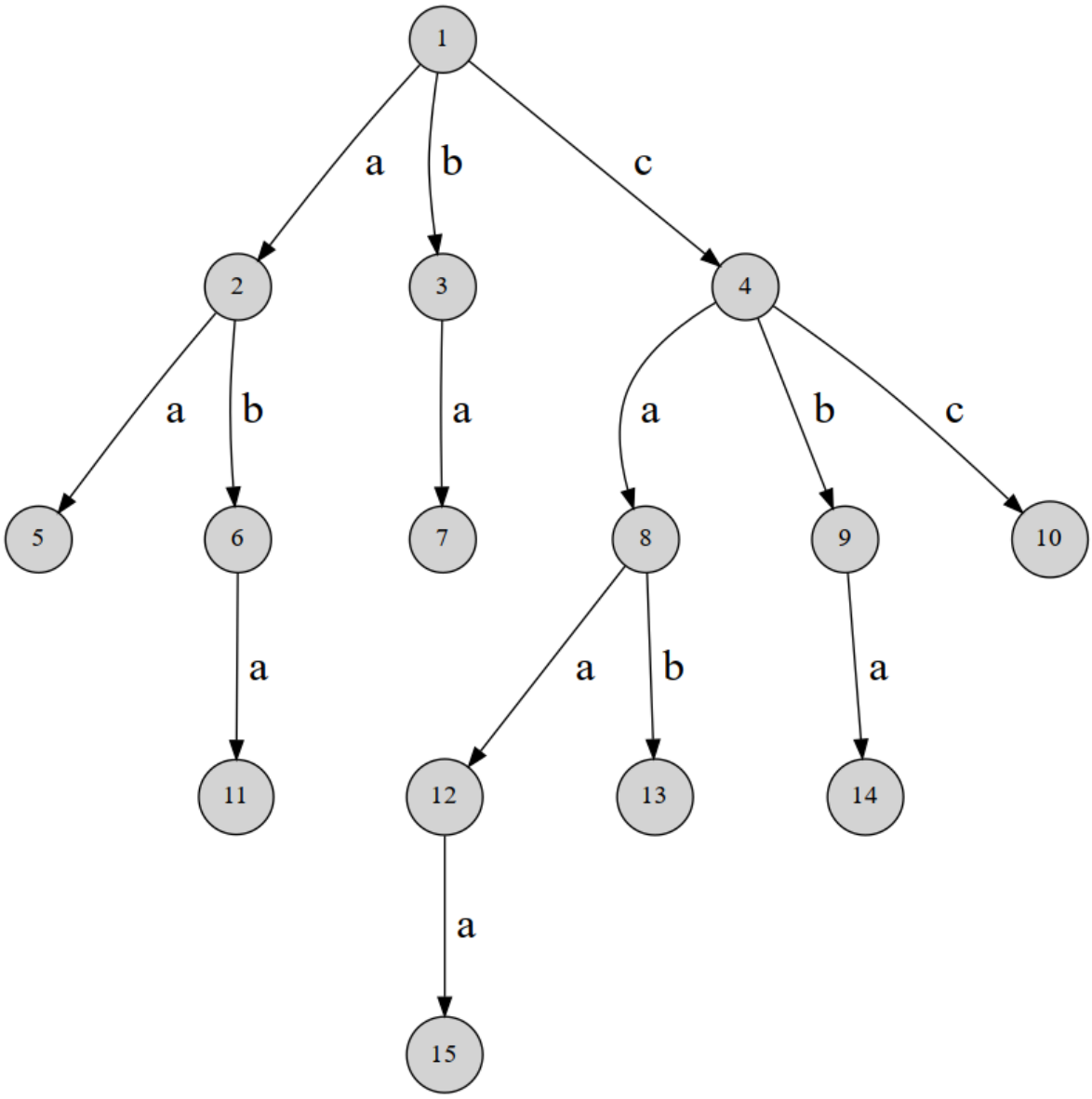
1456：【例题2】图书管理

```
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;
map<ull,int> mp;
ull b=27,mod=1<<31;
char op[205],str[205];
int main(){
    int t;
    cin>>t;
    getchar();
    while(t--){
        scanf("%s",op);
        cin.getline(str,205);
        //gets(str);//必须用这个，不然最后一个测试点会超时
        int len=strlen(str);
        ull s=0;
        for( int i=0; i<len; i++ ){
            s=(s*b+(ull)(str[i]-'\0'))%mod;
        }
        if(!strcmp(op,"find")){
            if(mp[s]==1)
                printf("yes\n");
            else
                printf("no\n");
        }
        else{
            mp[s]=1;
        }
    }
}
```

遇到的问题：输入字符串时使用C语言语法才能通过测试用例。

Trie

字典树，英文名 Trie。顾名思义，就是一个像字典一样的树。



Trie

可以发现，这棵字典树用边来代表字母，而从根结点到树上某一结点的路径就代表了一个字符串。

$1 \rightarrow 4 \rightarrow 8 \rightarrow 12$ 表示的就是字符串 `caa` 。

流程

1. 初始化
   一棵空Trie仅包含一个根节点，该点的字符指针均指向空。

```
int ch[N][Z],cnt;    //Z为字符集大小，N 是节点个数
bool val[N];
//若val=true则表示从根到该点经过的边上字母组成的字符串是实际字符串集合中的元素
```

2. 插入当需要插入一个字符串S时，我们令一个指针P起初指向根节点。然后，依次扫描S中的每个字符 c：

   a.若P的c字符指针指向一个已经存在的节点Q，则令P＝Q。

   a.若P的c字符指针指向空，则新建一个节点Q，令P的 c 字符指针指向Q，然后令P＝Q。 当S中的字符扫描完毕时，在当前节点P上标记它是一个字符串的末尾。

```
void insert(char *s) {//char *s表示一个字符数组
    int len = strlen(s);
    int u = 1;//1为根节点
    for(int i = 0; i < len; ++i){
        int c = s[i] - 'a';
        if(!ch[u][c])
            ch[u][c] = ++tot;//若不存在这条边则要新建一个节点与转移边
        u = ch[u][c]; //tot为总点数
    }
    val[u] = true;
    //在串的结尾处将val赋值，表示它代表一个实际字符串集合中的元素
}
```

3. 查询当需要查询一个字符串S在Trie中是否存在时，我们令一个指针P起初指向根节点，然后依次扫描S中的每个字符c：

   a.若P的c字符指针指向空，则说明S没有被插入过Trie ，结束查询。

   b.若P的c字符指针指向一个已经存在的节点Q，则令P=Q 。

   当S中的字符扫描完毕时，若当前节点P被标记为一个字符串的末尾，则说明S 在 Trie 中存在，否则说明S没有被插入过Trie。

```
bool find(char s[]) {
    int len = strlen(s);
    int u = 1;
    for(int i = 0; i < len; ++i){
        int c = s[i] - 'a';
        if(!ch[u][c])
            return false;
        u = ch[u][c];
    }
    return true;
}
```

1471：【例题1】Phone List

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1e5+10;
int flag,t,n;
char str[15];
bool vis[maxn];//vis是为了防止出现911,91这种情况
int ch[maxn][15];
bool val[maxn];
int cnt;
void insert(char* s){
    int len=strlen(s),u=0;
    for( int i=0; i<len; i++ ){
        int v=s[i]-'0';
        if(!ch[u][v])
            ch[u][v]=cnt++;
        u=ch[u][v];
        if(vis[u]&&i==len-1){
            flag=1;
            return;
```

```
        }
        vis[u]=true;
        if(val[u]){
            flag=1;
            return;
        }
    }
    val[u]=1;
}
int main(){
    scanf("%d",&t);
    while(t--){
        flag=0;
        memset(ch,0,sizeof(ch));
        memset(vis,0,sizeof(vis));
        memset(val,0,sizeof(val));
        cnt=1;
        scanf("%d",&n);
        for( int i=0; i<n; i++ ){
            scanf("%s",str);
            insert(str);
        }
        if(flag)
            printf("NO\n");
        else
            printf("YES\n");
    }
    return 0;
}
```

1472：【例题2】The XOR Largest Pair

思路：对每一个数字 x 的二进制位进行查找，每一步都尽量找与当前位置相反的数字进行访问，如果有相反的数字，根据xor运算这一位就会留下一个 1 ，最后找出最大值就行。

```
#include <bits/stdc++.h>
using namespace std;
#define maxn 100005
int cnt;
int ch[maxn*32][2];
int n;
int a[maxn];
void insert(int x){
    int u=0;
    for(int i=31;i>=0;i--){
        int id=(x>>i)&1;
        if(!ch[u][id])
            ch[u][id]=++cnt;
        u=ch[u][id];
    }
}
int find(int x){
    int u=0;
    int ans=0;
    for(int i=31;i>=0;i--){
        int id=(x>>i)&1;
        if(ch[u][!id]){//如果存在与x这一位数字相反的边
```

```
            u=ch[u][!id];
            ans=(ans<<1)|1;//把这一位上的 1 给添加到ans上
        }
        else{//如果不存在
            u=ch[u][id];
            ans=ans<<1;//把0 给添加到ans上
        }
    }
    return ans;
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        insert(a[i]);
    }
    int maxx=-1;
    for(int i=1;i<=n;i++){
        maxx=max(maxx,find(a[i]));
    }
    printf("%d\n",maxx);
    return 0;
}
```

KMP算法

KMP求解什么问题?

字符串匹配。给你两个字符串，寻找其中一个字符串是否包含另一个字符串，如果包含，返回包含的起始位置。

```
char *str = "bacbababadababacambabacaddababacasdsd";
char *ptr = "ababaca";
```

算法说明

朴素的字符串匹配过程:

从目标字符串str（假设长度为n）的第一个下标选取和ptr长度（长度为m）一样的子字符串进行比较，如果一样，就返回开始处的下标值，不一样，选取str下一个下标，同样选取长度为n的字符串进行比较，直到str的末尾。这样的时间复杂度是$O(n*m)$。

复杂度太高！！！

KMP：时间复杂度$O(m+n)$

充分利用了目标字符串ptr的性质（比如里面部分字符串的重复性，即使不存在重复字段，在比较时，实现最大的移动量）。

next转移函数（pre前缀函数）

len表示最长前缀长度，i,以字符串ababcabaa为例

1)ptr[len]==ptr[i]　　len++;next[i]=len;i++;

2)ptr[len]!=ptr[i]　　len=next[len-1];

```
void getNext(){
```

```
    nextt[0]=0;
    int len=0;
    int i=1;
    while(i<m){
        if(s2[i]==s2[len]){
            len++;
            nextt[i]=len;
            i++;
        }
        else{
            if(len>0){
                len=nextt[len-1];
            }
            else{//避免死循环
                nextt[i]=len;
                i++;
            }
        }
    }
}
```

KMP搜索

str n i

ptr m j

str[i]==ptr[j] i++;j++;

str[i]!=ptr[j] j=next[j];

```
int kmp(){
    int i=0,j=0;
    while(i<n){
        if(s1[i]==s2[j]){
            i++;j++;
        }
        else{
            j=nextt[j];
            if(j==-1){//j到了-1
                i++;
                j++;
            }
        }
        if(j==m){
            printf("找到了\n");
            break;
        }
    }
    return ans;
}
```

1465：【例题1】剪花布条

```
#include<bits/stdc++.h>
using namespace std;
string s1,s2;
int nextt[1004],n,m;
```

```
void getNext(){
    nextt[0]=0;
    int len=0;
    int i=1;
    while(i<m){
        if(s2[i]==s2[len]){
            len++;
            nextt[i]=len;
            i++;
        }
        else{
            if(len>0){
                len=nextt[len-1];
            }
            else{//避免死循环
                nextt[i]=len;
                i++;
            }
        }
    }
}
void move_next(){
    for(int i=m-1;i>0;i--){
        nextt[i]=nextt[i-1];
    }
    nextt[0]=-1;
}
int kmp(){
    int ans=0,j=0;
    int i=0;
    while(i<n){
        if(s1[i]==s2[j]){
            i++;j++;
        }
        else{
            j=nextt[j];
            if(j==-1){//j到了-1
                i++;
                j++;
            }
        }
        if(j==m){
            ans++;
            j=0;
        }
    }
    return ans;
}
int main(){
    while(1){
        cin>>s1;
        n=s1.size();
        if(n==1&&s1[0]=='#')
            break;
        cin>>s2;
        m=s2.size();
        getNext();
        move_next();
```

```
        printf("%d\n",kmp());
    }
}
```

1466：【例题2】Power Strings

m字符串长度，如果存在循环节，则m-next[m-1]表示每个循环节的长度，则

若m%(m-next[m-1])==0，ans=m/(m-next[m-1])

若m%(m-next[m-1])!=0，ans=1

```
#include <bits/stdc++.h>
using namespace std;
int m,nextt[1000005];
char s[1000005];
void getNext(){
    nextt[0]=0;
    int len=0;
    int i=1;
    while(i<m){
        if(s[i]==s[len]){
            len++;
            nextt[i]=len;
            i++;
        }
        else{
            if(len>0){
                len=nextt[len-1];
            }
            else{//避免死循环
                nextt[i]=len;
                i++;
            }
        }
    }
}
int main(){
    while(scanf("%s",s)){
        if (s[0]=='.')
            break;
        m=strlen(s);
        getNext();
        printf("%d\n",m % (m-nextt[m-1]) == 0 ?m / (m-nextt[m-1]) : 1 );
    }
}
```

AC 自动机

AC 自动机是 **以 TRIE 的结构为基础**，结合 **KMP 的思想** 建立的。

失配指针

AC 自动机利用一个 fail 指针来辅助多模式串的匹配。

状态 u 的 fail 指针指向另一个状态 v，其中 $v \in Q$，且 v 是 u 的最长后缀（即在若干个后缀状态中取最长的一个作为 fail 指针）。

fail 指针与 KMP 中的 next 指针的异同

- 共同点：两者同样是在失配的时候用于跳转的指针。
- 不同点：next 指针求的是最长 Border（即最长的相同前后缀），而 fail 指针指向所有模式串的前缀中匹配当前状态的最长后缀。

因为 KMP 只对一个模式串做匹配，而 AC 自动机要对多个模式串做匹配。有可能 fail 指针指向的结点对应着另一个模式串，两者前缀不同。

构建fail指针

考虑字典树中当前的结点 u，u 的父结点是 p，p 通过字符 `c` 的边指向 u，即 $trie[p,c]=u$。假设深度小于 u 的所有结点的 $fail$ 指针都已求得。

- 如果 $trie[fail[p],c]$ 存在：则让 u 的 fail 指针指向 $trie[fail[p],c]$。相当于在 p 和 $fail[p]$ 后面加一个字符 `c`，分别对应 u 和 $fail[u]$。
- 如果 $trie[fail[p],c]$ 不存在：那么我们继续找到 $trie[fail[fail[p]],c]$。重复 1 的判断过程，一直跳 fail 指针直到根结点。
- 如果真的没有，就让 fail 指针指向根结点。

如此即完成了 $fail[u]$ 的构建。

以字符串i he his she hers举例

build()函数

构建 fail 指针，构建自动机

- `tr[u,c]` 可以理解为从状态（结点）u 后加一个字符 `c` 到达的状态（结点），即一个状态转移函数 $trans(u,c)$。
- `q` 队列，用于 BFS 遍历字典树。
- `fail[u]` 结点 u 的 fail 指针。

```
void build() {
  for (int i = 0; i < 26; i++)
    if (tr[0][i]) //将根结点的子结点一一入队
        q.push(tr[0][i]);
  while (q.size()) {
    int u = q.front();//取出队首的结点 u
    q.pop();
    for (int i = 0; i < 26; i++) {//遍历字符集,即寻找u的子节点
      if (tr[u][i]){
        fail[tr[u][i]] = tr[fail[u]][i];
        q.push(tr[u][i]);
      }
      else
        tr[u][i] = tr[fail[u]][i];//为什么有这个else，以his的s为例
    }
  }
}
```

将结点按 BFS 顺序入队，依次求 fail 指针。这里的字典树根结点为 0，我们将根结点的子结点一一入队。

然后开始 BFS：每次取出队首的结点 u。fail[u]指针已经求得，我们要求 u 的子结点们的 fail 指针。然后遍历字符集（这里是 0-25，对应 a-z）：

- 如果 $tr(u,i)$ 存在，我们就将 $tr(u,i)$ 的 fail 指针赋值为 $tr(fail[u],i)$ 。这里似乎有一个问题。根据之前的讲解，我们应该用 while 循环，不停的跳 fail 指针，判断是否存在字符 `i` 对应的结点，然后赋值。不过在代码中我们一句话就做完这件事了。
- 否则， $tr(u,i)$ 不存在，就让 $tr(u,i)$ 指向 $tr(fail[u],i)$ 的状态。

query()匹配函数

```
int query(char *t) {
  int u = 0, res = 0;
  for (int i = 1; t[i]; i++) {
    u = tr[u][t[i] - 'a'];  // 转移
    for (int j = u; j && e[j] != -1; j = fail[j]) {
      res += e[j], e[j] = -1;
    }
  }
  return res;
}
```

洛谷P3808

https://www.luogu.org/problem/P3808

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6+6;
int tr[N][26], tot;
int e[N], fail[N];
void insert(char *s) {
    int u = 0;
    for (int i = 1; s[i]; i++) {
        if (!tr[u][s[i] - 'a']) tr[u][s[i] - 'a'] = ++tot;
        u = tr[u][s[i] - 'a'];
    }
    e[u]++;
}
queue<int> q;
void build() {
    for (int i = 0; i < 26; i++)
        if (tr[0][i]) q.push(tr[0][i]);
    while (q.size()) {
        int u = q.front();
        q.pop();
        for (int i = 0; i < 26; i++) {
            if (tr[u][i])
                fail[tr[u][i]] = tr[fail[u]][i], q.push(tr[u][i]);
            else
                tr[u][i] = tr[fail[u]][i];
        }
    }
}
int query(char *t) {
    int u = 0, res = 0;
    for (int i = 1; t[i]; i++) {
        u = tr[u][t[i] - 'a'];  // 转移
        for (int j = u; j && e[j] != -1; j = fail[j]) {
            res += e[j], e[j] = -1;
        }
```

```
    }
    return res;
}
int main() {
    int n;
    scanf("%d", &n);
    char s1[N];
    for (int i = 1; i <= n; i++) {
        scanf("%s", s1 + 1);
        insert(s1);
    }
    char s2[N];
    scanf("%s", s2 + 1);
    build();
    printf("%d\n", query(s2));
    return 0;
}
```

洛谷3796

https://www.luogu.org/problem/P3796

```
#include <bits/stdc++.h>
using namespace std;
const int N = 156, L = 1e6 + 6;
const int SZ = N * 80;
int tot, tr[SZ][26];
int fail[SZ], idx[SZ], val[SZ];
int cnt[N];  // 记录第 i 个字符串的出现次数
void init() {
    memset(fail, 0, sizeof(fail));
    memset(tr, 0, sizeof(tr));
    memset(val, 0, sizeof(val));
    memset(cnt, 0, sizeof(cnt));
    memset(idx, 0, sizeof(idx));
    tot = 0;
}
void insert(char *s, int id) {  // id 表示原始字符串的编号
    int u = 0;
    for (int i = 1; s[i]; i++) {
        if (!tr[u][s[i] - 'a']) tr[u][s[i] - 'a'] = ++tot;
        u = tr[u][s[i] - 'a'];
    }
    idx[u] = id;
}
queue<int> q;
void build() {
    for (int i = 0; i < 26; i++)
        if (tr[0][i]) q.push(tr[0][i]);
    while (q.size()) {
        int u = q.front();
        q.pop();
        for (int i = 0; i < 26; i++) {
            if (tr[u][i])
                fail[tr[u][i]] = tr[fail[u]][i], q.push(tr[u][i]);
            else
                tr[u][i] = tr[fail[u]][i];
```

```
        }
    }
}
int query(char *t) {  // 返回最大的出现次数
    int u = 0, res = 0;
    for (int i = 1; t[i]; i++) {
        u = tr[u][t[i] - 'a'];
        for (int j = u; j; j = fail[j]) val[j]++;
    }
    for (int i = 0; i <= tot; i++)
        if (idx[i]) res = max(res, val[i]), cnt[idx[i]] = val[i];
    return res;
}
int n;
char s[N][100], t[L];
int main() {
    while (~scanf("%d", &n)) {
        if (n == 0) break;
        init();
        for (int i = 1; i <= n; i++) scanf("%s", s[i] + 1), insert(s[i], i);
        build();
        scanf("%s", t + 1);
        int x = query(t);
        printf("%d\n", x);
        for (int i = 1; i <= n; i++)
            if (cnt[i] == x) printf("%s\n", s[i] + 1);
    }
    return 0;
}
```