字符串1

哈希Hash

哈希算法可以把一个任意长度的字符串映射为一个非负整数，且冲突概率很小。**一个字符出串的哈希值可以看作是一个 P 进制数对 M 的余数。**

- 一般取 p = 131 或 p = 13331， M = 2^{64} 。

使用 unsigned long long 存hash值，产生溢出时相当于自动对 2^{64} 取模，可以避免低效的取模操作。

可以用递推的方式 O(n) 求出字符串所有前缀的Hash值：

$$H(i) = H(i-1) * p \ + \ val(s[i])$$

O(1)计算字符串任意子串的Hash值：以从 l 到 r 位置的子串s为例

$$H(s) = H(r) - H(l-1) * p^{r-l+1}$$

例题：

给一个仅包含26个小写字母的字符串，每次询问输入4个数字l1,r1,l2,r2。问l1到r1的子串与l2到r2的子串是否相同。

```
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;
const int N = 1e5 + 5, base = 131; //13331

char s[N];
ull h[N], p[N];

ull get_hash(int l, int r){
    return h[r] - h[l - 1] * p[r - l + 1];
}

int main()
{
    scanf("%s",s + 1);
    int n = strlen(s + 1);

    p[0] = 1;
    for(int i = 1; i <= n; ++ i){
        h[i] = h[i - 1] * base + (s[i] - 'a' + 1);
        p[i] = p[i - 1] * base;
    }

    int q; scanf("%d",&q);
    while(q --){
        int l1, r1, l2, r2; scanf("%d%d%d%d",&l1, &r1, &l2, &r2);
        if(get_hash(l1, r1) == get_hash(l2, r2)) printf("Yes\n");
        else printf("No\n");
    }
    return 0;
```

```
}
```

KMP算法

- 字符串匹配算法, $O(n+m)$
- 应用：找字符串最小循环节
  - $len - next[i]$为字符串的最小循环节，如果 $len\%(len-next[i]) == 0 $,此字符串最小循周期为 $len/(len - next[i])$

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e5 + 5;

char t[N], p[N];
int n, m;

int nxt[N];

void get_nxt(){
    int i = 0, j = -1;
    nxt[0] = -1;
    while(i < m){
        if(j == -1 || p[i] == p[j]) {
            i ++, j ++, nxt[i] = j;
        }
        else j = nxt[j];
    }
}

void kmp(){
    get_nxt();
    int i = 0, j = 0;
    while(i < n && j < m){
        if(j == -1 || t[i] == p[j]) i ++, j ++;
        else j = nxt[j];
    }
    if(j == m) printf("%d\n",i - m + 1);
    else printf("-1\n");
}

int main()
{
    int T; scanf("%d", &T);
    while(T --){
        scanf("%d%d",&n,&m);
        scanf("%s", t); scanf("%s", p);
        kmp();
    }
    return 0;
}
```

Manacher算法

- 计算字符串最长回文串长度

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e5 + 5;
char ss[N], s[(N << 1) + 10];
int len[(N << 1) + 10];

int Manacher(){
    int right, mid, maxlen, i;
    memset(s, 0, sizeof(s));
    memset(len, 0, sizeof(len));
    s[0] = '$';
    for(i = 0; ss[i]; ++ i){
        s[(i << 1) + 1] = '#';
        s[(i << 1) + 2] = ss[i];
    }
    s[(i << 1) + 1] = '#';

    maxlen = right = mid = 0;
    for(i = 1; s[i]; ++ i){
        len[i] = (i < right ? min(len[(mid << 1) - i], right - i) : 1);
        while(s[i + len[i]] == s[i - len[i]]) ++ len[i];
        maxlen = max(maxlen, len[i]);
        if(right < i + len[i]) mid = i, right = i + len[i];
    }
    return maxlen - 1;
}

int main()
{
    scanf("%s",ss);
    printf("%d\n",Manacher());
    return 0;
}
```

字典树Trie

- 给出n个单词和m个询问，每次询问一个子串是多少个单词的前缀?
-  1596007382133

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
```

```
int trie[N][30], num[N];
int cnt = 1;
bool vis[N];
char s[N];

void trie_Insert(char *str){
    int root = 0, len = strlen(str);
    for(int i = 0; i < len; ++ i){
        int id = s[i] - 'a';
        if(!trie[root][id]) trie[root][id] = ++ cnt;
        num[trie[root][id]] ++;
        root = trie[root][id];
    }
    vis[root] = true;
}
int trie_Find(char *str){
    int root = 0, len = strlen(str);
    for(int i = 0; i < len; ++ i){
        int id = str[i] - 'a';
        if(!trie[root][id]) return 0;
        root = trie[root][id];
    }
    return num[root];
}
int main()
{
    cnt = 0;
    while(gets(s)){
        if(s[0] == '\0') break;
        trie_Insert(s);
    }
    while(~scanf("%s",&s)){
        printf("%d\n",trie_Find(s));
    }
    return 0;
}
```