DFS

https://www.luogu.com.cn/problem/P2036

n种材料，每种有酸度s和苦度b。

总酸度 = 酸度乘积，总苦度 = 苦度之和

选择至少一种材料，使得酸度和苦度绝对差值最小

```
#include <bits/stdc++.h>
using namespace std;
int n,s[12],b[12],ans=0x7fffffff; // 初始化极大值

// 搜到第i种，目前已经选的材料的总酸度为x，总苦度为y
void dfs(int i, int x,int y)
{
    if(i>n) // 终止条件
    {
        if(x==1 && y==0) return;
        ans = min(ans,abs(x-y)); return;
    }
    // 选择第i种的情况
    dfs(i+1,x*s[i],y+b[i]);
    // 不选择第i种的情况
    dfs(i+1,x,y);
}

int main(void)
{
    // 输出
    cin >> n;
    for(int i=1;i<=n;i++)
        cin >> s[i] >> b[i];
    dfs(1,1,0);
    cout << ans << endl;
    return 0;
}
```

六、字符串

1. 字符串哈希

模板题

https://www.luogu.com.cn/problem/P3370

给定 N 个字符串，第 i 个串的长度为 M_i，字符串内包含数字、大小写字母。

求 N 个串中有多少个不同的字符串。

N ≤ 10000，M_i ≈ 1000，M_max ≤ 1500

sum <= 1.5e7

STL string

cal_hash(string) --> long long

哈希：把一个串映射到一个值，通过比较值，达成比较原始串的目的

需要关注的点：如何使同一个串映射成的值是固定的，而同一个值尽可能映射到固定的串？

映射函数

（举例直接ASCII表求和的例子）

根据问题的需要抽象出合适的哈希函数

模板题思路比较清晰：设计hash函数，把每个串映射到一个值，再看有多少个不同的值

最常见的思路是**进制哈希**

0xAB A-10 B-11 --> 11* 16^0 + 10*16^1

AB a-z 0-25 aaa aa

可以看作是一个>16进制的数，进制数 > 不同符号的种类数。

注意不要把任何字符映射到0

通常我们采用的是多项式 Hash 的方法，对于一个长度为 l 的字符串 s 来说，我们可以这样定义多项式 Hash 函数：$f(s)=\sum_{i=1}^{l} s[i]\times b^{l-i} \pmod M$。例如，对于字符串 xyz，其哈希函数值为 xb^2+yb+z。

x + yb + z* b^2

单哈希

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
typedef unsigned long long ull;
ull base=131; // 选用一个质数
ull a[10010]; // a[i]为第i个串映射得到的值
char s[10010];
int n,ans=1;
ull mod=19260817; // 选用一个大质数 1e9
// 串映射到值的函数
ull hashs(char s[])
{
    int len=strlen(s);
    ull ans=0;
    for (int i=0;i<len;i++)
        ans=(ans*base+(ull)s[i])%mod; // 类似之前的数字进制转换
        // 可以看成转换成base进制
        // 1 x x^2
    return ans;
```

```
}
set<ull> st;
main()
{
    scanf("%d",&n);
    for (int i=1;i<=n;i++) // sort mycmp()
    {
        scanf("%s",s); // 输入串
        a[i]=hashs(s); // 计算该串对应的哈希
        st.insert(a[i]); // logn

        st.find(1); --> st.end()
        st.erase(1); logn
        st.erase(st.begin());
    }
    map<int,int> mp;

    set<int> st; // mulitset
    set<int>::iterator it;
    for(it = st.begin();it!=st.end();it++){

    }

    // n vis[]
    // set<int> stl
    cout << st.size() << '\n';
    // 排序 统计去重后还有多少个值
    sort(a+1,a+n+1); // nlogn mycmp() if(x<y)
    for (int i=2;i<=n;i++)
        if (a[i]!=a[i-1])
            ans++;
    printf("%d\n",ans);
}
```

相对单哈希，**双哈希**会更不容易发生碰撞

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
typedef unsigned long long ull;
ull base=131;
struct data
{
    ull x,y;
}a[10010];
char s[10010];
int n,ans=1;
ull mod1=19260817;
ull mod2=19660813;
// 两个hash的区别仅仅是选择的base数和mod数
// mod数都选用大质数 不要相同
// base数可以相同
```

```
ull hash1(char s[])
{
    int len=strlen(s);
    ull ans=0;
    for (int i=0;i<len;i++)
        ans=(ans*base+(ull)s[i])%mod1;
    return ans;
}
ull hash2(char s[])
{
    int len=strlen(s);
    ull ans=0;
    for (int i=0;i<len;i++)
        ans=(ans*base+(ull)s[i])%mod2;
    return ans;
}
bool comp(data a,data b)
{
    return a.x<b.x;
}
main()
{
    scanf("%d",&n);
    for (int i=1;i<=n;i++)
    {
        scanf("%s",s);
        // 采用两个不同的hash函数映射到两个值
        a[i].x=hash1(s); // mod1 base1
        a[i].y=hash2(s); // mod2 base2
    }
    // 排序去重 两个值都相同才认为两个串相同
    sort(a+1,a+n+1,comp);
    for (int i=2;i<=n;i++)
        if (a[i].x!=a[i-1].x || a[i-1].y!=a[i].y)
            ans++;
    printf("%d\n",ans);
}
```

base数和mod数选择的注意事项：

base mod

base数相当于base进制，大于所有字符对应的数字的最大值，不要含有模数的质因子(那还模什么)，比如一个字符集是a到z的题目，选择27、233、19260817 都是可以的。

比较常见的是取131和13331。

mod数选择一个较大的质数。哈希碰撞的理论发生概率为根号级别

绝大多数情况下，不要选择一个 10^9 级别的数，因为这样随机数据都会有Hash冲突，根据生日悖论，随便找上 10^9个串就有大概率出现至少一对Hash 值相等的串（参见BZOJ 3098 Hash Killer II）。

最稳妥的办法是选择两个 10^9级别的质数，只有模这两个数都相等才判断相等，但常数略大，目前暂时没有办法通过哈希碰撞卡掉这种写法（除了卡时间让它超时）（参见BZOJ 3099 Hash Killer III）。// codeforces hack

另一种选择mod数的方式是**自然溢出**。直接使用unsigned long long，不手动进行取模，它溢出时会自动对 2^{64}进行取模，如果出题人比较良心，这种做法也不会被卡，但这个是完全可以卡的，卡的方法参见BZOJ 3097 Hash Killer I。

自然溢出代码

```
ull base=131;
ull a[10010];
char s[10010];
int n,ans=1;
ull hashs(char s[])
{
    int len=strlen(s);
    ull ans=0;
    for (int i=0;i<len;i++)
        ans=ans*base+(ull)s[i];  // 不手动%mod 其他都一样
    return ans&0x7fffffff;
}
```

哈希在字符串的应用

https://www.luogu.com.cn/problem/P8630

给定一个长串 s。n个短串，每个长度为 8。

nlogn

cnt[i] i --> cnt[i] --> cnt[i] --> long long

问每个短串的所有排列在长串 s 中作为子串出现次数的总和。

模板题中，认为两个完全一致的串是匹配的

考虑这个问题下什么样的串是匹配的

由于考虑每个短串的所有排列，其实短串与长串s中任何一个长度为8的子串，只要每个字母出现的次数一致，就是匹配的

因此设计的哈希函数，需要把“每个字母出现次数”的信息映射到整数

```
#include <bits/stdc++.h>
using namespace std;
#define int unsigned long long
int base=114; // base进制数
int t[128],a[200001];
int change() // hash函数
{
    int cnt=1;
    for(int i='a';i<='z';i++) //
    cnt=cnt*base+t[i]; // 使用自然溢出
    return cnt;
```

```
}
signed main()
{
    int len,n,ans=0;
    string s,s1;
    cin>>s>>n;
    len=s.size();
    // 对于长串中所有长度为8的子串 映射成值放入a[i]
    for(int i=0;i<=len-8;i++)
    {
        memset(t,0,sizeof t); // temp
        for(int j=i;j<i+8;j++)
            t[s[j]]++; // cnt 1-26
        a[i]=change(); //
    }
    memset(t,0,sizeof t);
    while(n--)
    {
        memset(t, 0, sizeof(t));
        cin>>s1;
        // 对于每个短串 s1
        for(int j=0;j<8;j++)
        t[s1[j]]++; // 每个字母出现次数信息 cnt
        int b=change(); // b
        for(int i=0;i<=len-8;i++)
        if(b==a[i]) ans++;
    }
    cout<<ans;
}
```

最长回文子串

https://www.acwing.com/file_system/file/content/whole/index/content/3690/

给定一个字符串，求它的最长回文子串长度。

数据范围只能做 O(n) 或者 O(nlogn)

考虑一个更小的问题：如何判断一个串是不是回文串？s s'

朴素做法O(n)。有没有O(n)预处理，O(1)查询的方法？

```
typedef unsigned long long ULL;
ULL h[N], p[N]; // h[k]存储字符串前k个字母的哈希值, p[k]存储 P^k mod 2^64

// 初始化
p[0] = 1; // 1 1*10  O(n)
for (int i = 1; i <= n; i ++ )
{
    h[i] = h[i - 1] * P + str[i]; // 123 h[i]123 h[i]12
    p[i] = p[i - 1] * P;
}
```

```
// 12345
// 10000 1000 100 10 1

// 计算子串 str[l ~ r] 的哈希值 O(1)
ULL get(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}
abc
cba
```

对于一个串s和它反过来的串s'，通过以上思路可以预处理出s的每个子串的哈希值，和s'的每个子串的哈希值。

如果在原串s上和反串s'上，区间[L,R]的哈希值相等，则[L,R]部分子串为回文。

预处理复杂度O(n)，每次查询/判断子串是否回文的复杂度为O(1)。

基于以上，考虑怎么求最长回文子串长度？ 二分[1-5] [1,3] [3,5]

[1-2] [4-5]

abcba

abba

观察回文串，我们可以发现回文串分为两类：

1. 奇回文串 $a[1 \sim n]$，长度 n 为奇数，且 $a[1 \sim n/2+1] = reverse(a[n/2+1 \sim m])$，它的中心点是一个字符。其中 $reverse(a)$ 表示把字符串 a 倒过来；
2. 偶回文串 $b[1 \sim n]$，长度 n 为偶数，且 $b[1 \sim n/2] = reverse(b[m/2+1 \sim m])$，它的中心点事两个字符之间的夹缝。

我们需要一个正的字符串，还需要一个反的字符串，如果正字符串等于反的字符串，那么奇数回文串就长度 $\times 2+1$,偶数回文串就直接长度 $\times 2$ 即可。

我们这么做是因为要找回文串，也就是前缀与后缀相等，拆分为一个正的字符串和一个反的字符串会更好处理。

所以，我们可以预处理出前缀 $hash$ 值，类似地，我们倒着预处理，求出后缀 $hash$ 值，就可以 $O(1)$ 计算任意子串正着或倒着读的 $hash$ 值。

这道题目中，我们枚举回文子串的中心位置 i。看从这个位置出发向左右两侧最长可以扩展出多长的回文串。也就是说：

1. 求出一个最大的整数 p 使得 $s[i-p \sim i] = reverse(s[i \sim i+p])$；
2. 求出一个最大的整数 q 使得 $s[i-q \sim i-1] = reverse(s[i \sim i+q-1])$。

我们对 p 和 q 进行二分答案，在所有枚举过的回文子串长度中取 $\max$ 即是本题答案，时间复杂度 $O(n \log n)$。

abcdc

cdcba

abcba

abba

abba

i-p i

i i+p

i

参考代码：

对于X2+1还是X2的问题，代码实现有以下技巧：

如果真实回文串长度为奇数，扩充后最左边是 #。否则最左边是字母。

> 这里有一个小技巧：在每个字符前面添加 `#`，比如 `aba` 就被扩充为 `#a#b#a`，遍历每个字符，尝试以所有字符为对称中心的回文串最大的长度可能是多少。**注意：这样做，长度的 $\times 2$ 或 $\times 2+1$ 要省略。**
>
> 比如第一个 `a`，最大回文串是 `#a#`，第二个 `b`，最大回文串是 `a#b#a`。可以发现，当回文串最左边字符是 `#` 时，实际回文串的长度就是填充后对称中心左边子串的长度，例如 `#a#` 中对称中心 `a` 左边只有一个字符，回文串的长度就是 1；而当回文串最左边字符是字母时，实际回文串的长度就是填充后对称中心左边子串的长度加上 1，例如 `a#b#a` 中对称中心 `b` 左边有 2 个字符，回文串的长度就是 $2+1=3$。

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cstdio>
#include <cmath>
using namespace std;

#define ull unsigned long long
const int N = 2000010, P = 131;

char s[N];
ull h1[N], h2[N], p[N];

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

int main()
{
    int cnt = 0;
    while (scanf("%s", s + 1) && strcmp(s + 1, "END")) // 读入字符串
    {
        int n = strlen(s + 1) * 2;
        for (int i = n; i; i -= 2)
        {
            s[i] = s[i / 2];
            s[i - 1] = 'z' + 1;
        }
        // 哈希
```

```
        p[0] = 1;
        for (int i = 1, j = n; i <= n; i ++ , j -- )
        {
            h1[i] = h1[i - 1] * P + s[i] - 'a' + 1;
            h2[i] = h2[i - 1] * P + s[j] - 'a' + 1;
            p[i] = p[i - 1] * P;
        }
        ull ans = 0;
        // 枚举回文串中心位置
        for (int i = 1; i <= n; i ++ )
        {// p --> [i-p]*2+1
         // q -- [i-q]*
            ull l = 0, r = min(i - 1, n - i);
            while (l < r)
            {
                ull mid = (l + r + 1) / 2;
                if (get(h1, i - mid, i - 1) != get(h2, n - (i + mid) + 1, n - i))
r = mid - 1;
                else l = mid;
            }
            if (s[i - l] <= 'z') ans = max(ans, l + 1);
            else ans = max(ans, l);
        }
        printf("Case %d: %d\n", ++ cnt, ans);
    }
    return 0;
}
```

2. 字符串匹配和KMP

通常是给定一个长度为 10^5 或 10^6 的长子串（母串）和 n 个长度比较短的子串（模式串）。所有模式串的长度之和也在 10^5 或 10^6 。

最经典的字符串匹配是问每个模式串是否在母串中出现。

设母串长 n，模式串长 m，暴力做法复杂度 O(nm)。

现在需要复杂度O(n+m)

模板KMP

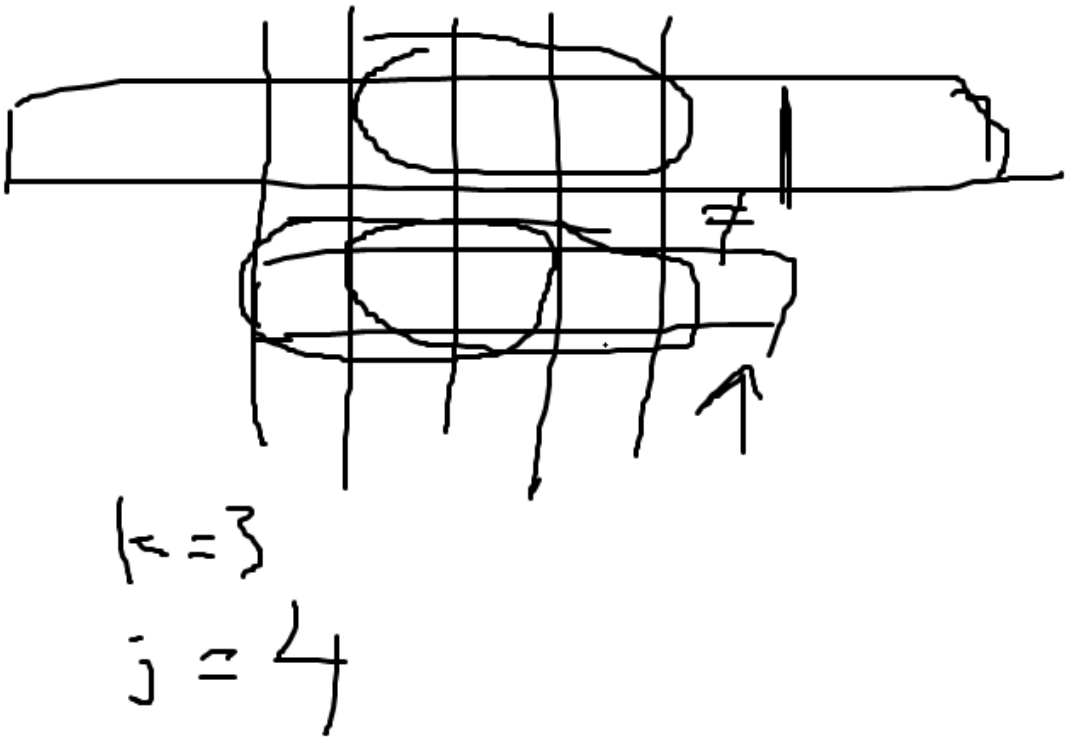
https://www.luogu.com.cn/problem/P3375

给定长串 s1，模式串 s2。求s2在s1中所有出现的位置。

定义一个字符串 *s* 的 border 为 *s* 的一个非 s 本身的子串 t，满足 t 既是 *s* 的前缀，又是 *s* 的后缀（画个图）

对于 *s*2，你还需要求出对于其每个前缀 *s*' （在s2自身上）的最长 border t' 的长度。（看下样例）

（先画图解释KMP为什么能优化）




```
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
int next[1000005];
int al,bl;
char s,a[1000005],b[1000005];
// 计算next数组
// next[i]的定义是：对于模式串的前i个字符，找到它最长的前缀 = 后缀
void getnext(){
    // 初始化
    int p=0;next[1]=0;
    // 遍历s2字符串
    for(int i=2;i<=bl;i++){
        // 如果b[i]!=b[p+1] 说明没有长度为p+1的border
        // 把p往前跳（这里还是画图解释）
        while(p&&b[i]!=b[p+1]) p=next[p];
        // 如果有匹配 border = p+1
        if(b[p+1]==b[i]) p++;
        // 填进next数组
        next[i]=p;
    }
    return;
}
// 如何根据next数组求模式串每次出现的位置？
// 考虑暴力求解法，如果s1[i]!=s2[j+1]，则下一次把s2往右边移动一格 重新从s2的第一个字符开始匹配
// 但如果已知s2的前k个字符 = s2[1:j]的后k个字符。在s1[i]!=s2[j]时，可以把s2往右边移动j-k格 重新从s2的第k+1个字符开始匹配
void KMP(){
    // 字符串匹配算法
```

```
    int p=0;
    for(int i=1;i<=al;i++){
        // 如果出现失配（s1[i]!=s2[p+1]），把p往前跳
        // （还是跟着上面的图演示吧）
        while(p&&b[p+1]!=a[i]) p=next[p];
        if(b[p+1]==a[i]) p++; // 匹配上了 继续往后看
        if(p==bl){ // 完全匹配上了
            // 如果目前是s1[i] == s2[bl]
            // 寻找下一个匹配时 s2的[1:p]部分都匹配好了
            cout<<i-bl+1<<endl;p=next[p];
        }
    }
    for(int i=1;i<=bl;i++) cout<<next[i]<<" ";
    // next[i]其实就是s2长度为i的前缀的最长border
    // 即s2长度为i的前缀上，最长的前缀 = 后缀的长度
    return;
}
int main(){
    cin>>a+1; // 读入母串
    al=strlen(a+1);
    cin>>b+1; // 读入子串（模式串）
    bl=strlen(b+1);
    getnext(); // 计算next数组 fail
    KMP();     // 输出答案
    return 0;
}
```