// ========= Trie 树 =========

```
const int maxnode = 4000010;
const int sigma_size = 26;
struct Trie {
  int ch[maxnode][sigma_size];
  int sz, val[maxnode];
  void clear()
{sz=0;memset(ch[0],0,sizeof(ch[0]));}
  // 一开始只有一个根节点 0
  // 点的编号: 0 ~ sz
  int idx(char c){return c-'a';}
  void insert(char *s, int v) {
    int u = 0, n = strlen(s);
    for(int i=0;i<n;i++) {
      int id = idx(s[i]);
      if(!ch[u][id]) {
        ++sz;
        memset(ch[sz],0,sizeof(ch[sz]));
        val[sz]=0;
        ch[u][id]=sz;
      }
      u = ch[u][id];
    }
    val[u] += v;
  }
  int search(char *s) {
    int u = 0, n = strlen(s);
    for(int i=0;i<n;i++) {
      int id = idx(s[i]);
      if(ch[u][id]==0)
        return 0;
      u = ch[u][id];
    }
    return val[u];
  }
};
Trie trie;
```

// ========= AC 自动机 =========

```
inline void insert(char* word, int value) {
  int len = strlen(word), j = 0;
  for (int i=0; i<len; ++i) {
    int c = word[i] - 'a';
    if(!ch[j][c]) ch[j][c] = ++size;
    j = ch[j][c];
  }
  val[j]+=value;
}
inline void GetFail() {
  queue<int> q;
  fail[0] = 0;
  for (int c = 0; c < Sigma_Size; ++c) {
    int p = ch[0][c];
    if(p) {
      fail[p] = last[p] = 0;
      q.push(p);
    }
  }
  while(!q.empty()) {
    int head = q.front();
    q.pop();
    for (int c = 0; c < Sigma_Size; ++c) {
      int u = ch[head][c];
      if(!u) continue;
      q.push(u);
      int v = fail[head];
      while(v && !ch[v][c]) v = fail[v];
      fail[u] = ch[v][c];
      last[u] = val[fail[u]] ? fail[u] :
last[fail[u]];
//这样保证了沿 last 数组经过的节点(除了 u
与 root) 都会是单词节点(val>0)
//val[u]有可能大于 0
    }
  } }
inline void Founded(int x) {
        for(; x; x=last[x]) cnt[x]++;}
// last[i]=j 表 j 节点表示的单词是 i 节点单词
的后缀，且 j 节点是单词节点
// 递归打印与结点 i 后缀相同的前缀节点编
号
// 进入此函数前需保证 val[x]>0
// cnt[] 记录某个点代表的单词 在文章中出
现的次数
inline void Find(char* text) {
  int j = 0, len = strlen(text);
  memset(cnt, 0, sizeof(cnt));
  for (int i=0; i<len; ++i) {
    int c = text[i] - 'a';
    while(j && !ch[j][c]) j = fail[j];
    j = ch[j][c];
    if(val[j]) Founded(j);
    else if(last[j]) Founded(last[j]);
  }
}
// main(): insert(P, 1); GetFail(); Find(T);
```

// ========= KMP 匹配=========

```
char P[maxn]; // Pattern 短串
char T[maxn]; // Text 长串
int f[maxn];
```

```
void getFail(char* P,int* f) {
//字符串 p 自我匹配
   int m = strlen(P);
   f[0] = f[1] = 0;
   for(int i = 1; i < m; i++) {
      int j = f[i];
      while(j && P[i]!=P[j])
         j = f[j];
      f[i+1] = P[i]==P[j] ? j+1 : 0;
   }
}
void Find(char* T, char* P, int* f) {
//p 去匹配字符串 T
   int n = strlen(T), m = strlen(P);
   getFail(P, f);  //得出部分匹配表
   int j = 0;
   //j:短串的下标  i: 长串下标
   for (int i = 0; i < n; i++) {
      while (j && P[j] != T[i])
         j = f[j];
      if (P[j] == T[i]) j++;
      if(j == m)
         printf("%d ", i-m+1);
   }
   puts("");
}
int main() {
   // c++ getline(cin, P)
   // c   gets(P)
   while (gets(P))
   {gets(T); Find(T, P, f);}
}
```

// ========== 后缀数组 ==========

```
const int CHARSET_SIZE = 257;
string s, s2;
int sa[maxn], rk[maxn], ht[maxn];
int cnt[maxn], rk1[maxn], rk2[maxn];
bool cmpSA(int *y,int a,int b,int k, int n) {
   int a1=y[a];
   int b1=y[b];
   int a2=a+k >= n ? -1: y[a+k];
   int b2=b+k >= n ? -1: y[b+k];
   return a1==b1 && a2==b2;
}
void buildSA(const string& str,int n,int m){
// or “const char* s”
   for(int i = 0; i < m; i++) cnt[i] = 0;
   for(int i = 0; i < n; i++)
      ++cnt[rk1[i]=(int)str[i]];
   for(int i = 1; i < m; i++) cnt[i] += cnt[i-1];
   for(int i = n-1; i >= 0; i--)
         sa[--cnt[rk1[i]]]=i;
   for(int len=1; len<=n; len<<=1) {
      int p=0;
      for(int i = n-len; i < n; i++) rk2[p++]=i;
      for(int i=0; i<n; i++)
         if( sa[i]>=len )
            rk2[p++]=sa[i]-len;
      for(int i = 0; i < n; i++) cnt[i]=0;
      for(int i = 0; i < n; i++)
      ++cnt[rk1[rk2[i]]];
      for(int i = 1; i < n; i++)
         cnt[i] += cnt[i-1];
      for(int i = n-1; i >= 0; i--)
         sa[--cnt[rk1[rk2[i]]]]=rk2[i];
      for(int i = 0; i < n; i++)
         swap(rk1[i], rk2[i]);
      int tot_rk = 1;
      rk1[sa[0]] = 0;
      for (int i = 1; i < n; i++)
      rk1[sa[i]] =
         cmpSA(rk2,sa[i],sa[i-1],len,n)
         ? tot_rk-1 : tot_rk++;
      if (tot_rk >= n) break;
   }
}
void getHeight(const string& str, int n) {
   for (int i = 0; i<n; i++) rk[sa[i]] = i;
   ht[0] = 0;
   for (int i = 0, k = 0; i < n; i++) {
      if (rk[i] == 0) continue;
      int j = sa[rk[i] - 1];
      if (k) k--;
      while (str[i + k] == str[j + k]) k++;
      ht[rk[i]] = k;
   }
}
int main() {
   getline(cin, s);
   getline(cin, s2);
   s = s + '$' + s2;
   int N = s.size();
   buildSA(s, N, CHARSET_SIZE);
   getHeight(s, N);
}
```