1 字符串

1.1 KMP 算法

```
void getnex(char *s, int *nex){
  int n = strlen(s + 1);
  for(int j = 0, i = 2; i <= n; i++){
    while(j && s[j + 1] != s[i])j = nex[j];
    if(s[i] == s[j + 1]) j++;
    nex[i] = j;
  }
}
```

1.2 扩展 KMP 算法

```
//nex[i] 表示 s 和其后缀 s[i, n] 的 lcp 的长度
void getnext(char s[], int n, int nex[])
{
  nex[1] = n;
  int &t = nex[2] = 0;
  for(; t + 2 <= n && s[1 + t] == s[2 + t]; t++);
  int pos = 2;
  for(int i = 3; i <= n; i++){
    if(i + nex[i - pos + 1] < pos + nex[pos])
      nex[i] = nex[i - pos + 1];
    else{
      int j = max(0, nex[pos] + pos - i);
      for(;i + j <= n && s[i + j] == s[j + 1]; j++);
      nex[i] = j; pos = i;
    }
  }
}
//extend[i] 表示 s2 和 s1 后缀 s1[i, n] 的 lcp 的长度
void getextend(char s1[], char s2[], int extend[])
{
  int n = strlen(s1 + 1), m = strlen(s2 + 1);
  getnext(s2, m, next);
  int &t = extend[1] = 0, pos = 1;
  for(; t < n && t < m && s1[1 + t] == s2[1 + t]; t++);
  for(int i = 2; i <= n; i++){
    if(i + nex[i - pos + 1] < pos + extend[pos])
      extend[i] = nex[i - pos + 1];
    else{
      int j = max(0, extend[pos] + pos - i);
      for(; i + j <= n && j < m && s1[i + j] == s2[j + 1]; j++);
      extend[i] = j; pos = i;
    }
  }
}
```

1.3 AC 自动机

```
const int C = 26, L = 1e5 + 5, N = 5e5+10;
int n, root, cnt, fail[N], son[N][26], num[N];
char s[L];
inline int newNode(){
  cnt++; fail[cnt] = num[cnt] = 0;
  memset(son[cnt], 0, sizeof(son[cnt]));
  return cnt;
}
void insert(char *s){
  int n = strlen(s + 1), now = 1;
  for(int i = 1; i <= n; i++){
    int c = s[i] - 'a';
    if(!son[now][c]) son[now][c] = newNode();
    now = son[now][c];
  }
  num[now]++;
}
void getfail(){
  static queue<int> Q;
  fail[root] = 0;
  Q.push(root);
  while(!Q.empty()){
    int now = Q.front();
    Q.pop();
    for(int i = 0; i < C; i++)
      if(son[now][i]){
        Q.push(son[now][i]);
        int p = fail[now];
        while(!son[p][i]) p = fail[p];
        fail[son[now][i]] = son[p][i];
      }
      else son[now][i] = son[fail[now]][i];
  }
}
int main(){
  cnt = 0; root = newNode();
  scanf("%d", &n);
  for(int i = 0; i < C; i++) son[0][i] = 1;
  for(int i = 1; i <= n; i++){
    scanf("%s", s + 1);
    insert(s);
  }
  getfail();
  return 0;
}
```

1.4 后缀自动机

1.4.1 广义后缀自动机（多串）

注意事项：空间是插入字符串总长度的 2 倍并请注意字符集大小。

```
const int N = 251010, C = 26;
int tot, las, root;
struct Node
{
  int son[C], len, par;
  void clear(){
    memset(son, 0, sizeof(son));
    par = len = 0;
  }
}node[N << 1];
inline int newNode(){return node[++tot].clear(), tot;}
void extend(int c)
{
  int p = las;
  if (node[p].son[c]) {
    int q = node[p].son[c];
    if (node[p].len + 1 == node[q].len)  las = q;
    else{
      int nq = newNode();
      las = nq; node[nq] = node[q];
      node[nq].len = node[p].len + 1;  node[q].par = nq;
      for (; p && node[p].son[c] == q; p = node[p].par)
        node[p].son[c] = nq;
    }
  }
  else{ // Naive Suffix Automaton
    int np = newNode();
    las = np; node[np].len = node[p].len + 1;
    for (; p && !node[p].son[c]; p = node[p].par)
      node[p].son[c] = np;
    if (!p) node[np].par = root;
    else{
      int q = node[p].son[c];
      if (node[p].len + 1 == node[q].len)
        node[np].par = q;
      else{
        int nq = newNode();
        node[nq] = node[q];
        node[nq].len = node[p].len + 1;
        node[q].par = node[np].par = nq;
        for (; p && node[p].son[c] == q; p = node[p].par)
          node[p].son[c] = nq;
      }
    }
  }
}
void add(char *s)
{
  int len = strlen(s + 1); las = root;
  for(int i = 1; i <= len; i++) extend(s[i] - 'a');
}
```

1.4.2 sam-ypm

sam-nsubstr

```
//SAM 利用后缀树进行计算，由儿子向 parert 更新
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef pair<int, int> pii;
const int inf = 1e9;
const int N = 251010, C = 26;
int tot, las, root;
struct Node
{
  int son[C], len, par, count;
  void clear(){
    memset(son, 0, sizeof(son));
    par = count = len = 0;
  }
}node[N << 1];
inline int newNode(){return node[++tot].clear(), tot;}
void extend(int c)//传入转化为数字之后的字符，从 0 开始
{
  int p = las, np = newNode(); las = np;
  node[np].len = node[p].len + 1;
  for(;p && !node[p].son[c]; p = node[p].par)
    node[p].son[c] = np;
  if(p == 0) node[np].par = root;
  else{
    int q = node[p].son[c];
    if(node[p].len + 1 == node[q].len)
      node[np].par = q;
    else{
      int nq = newNode();
      node[nq] = node[q];
      node[nq].len = node[p].len + 1;
      node[q].par = node[np].par = nq;
      for(;p && node[p].son[c] == q; p = node[p].par)
        node[p].son[c] = nq;
    }
  }
}
int main(){
  static char s[N];
  while(scanf("%s", s + 1) == 1){
    tot = 0;
    root = las = newNode();
    int n = strlen(s + 1);
    for(int i = 1;i <= n; i++) extend(s[i] - 'a');
    static int cnt[N], order[N << 1];
    memset(cnt, 0, sizeof(*cnt) * (n + 5));
    for(int  i = 1; i <= tot; i++) cnt[node[i].len]++;
    for(int i = 1; i <= n; i++) cnt[i] += cnt[i - 1];
    for(int i = tot; i; i--) order[ cnt[node[i].len]-- ] = i;
    static int dp[N]; memset(dp, 0, sizeof(dp));
    //dp[i] 为长度为 i 的子串中出现次数最多的串的出现次数
    for(int now = root, i = 1;  i <= n; i++){
      now = node[now].son[s[i] - 'a'];
      node[now].count++;
    }
    for(int i = tot; i; i--){
      Node &now = node[order[i]];
      dp[now.len] = max(dp[now.len], now.count);
      node[now.par].count += now.count;
    }
    for(int i = n - 1; i; i--) dp[i] = max(dp[i], dp[i + 1]);
    for(int i = 1; i <= n; i++) printf("%d\n", dp[i]);
  }
}
```

sam-lcs

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef pair<int, int> pii;
const int inf = 1e9;
const int N = 101010, C = 26;
```

```
int tot, las, root;
struct Node{
  int son[C], len, par, count;
  void clear(){
    memset(son, 0, sizeof(son));
    par = count = len = 0;
  }
}node[N << 1];
inline int newNode(){return node[++tot].clear(), tot;}
void extend(int c)//传入转化为数字之后的字符，从 0 开始
{
  int p = las, np = newNode(); las = np;
  node[np].len = node[p].len + 1;
  for(;p && !node[p].son[c]; p = node[p].par)
    node[p].son[c] = np;
  if(p == 0) node[np].par = root;
  else{
    int q = node[p].son[c];
    if(node[p].len + 1 == node[q].len)
      node[np].par = q;
    else{
      int nq = newNode(); node[nq] = node[q];
      node[nq].len = node[p].len + 1;
      node[q].par = node[np].par = nq;
      for(;p && node[p].son[c] == q; p = node[p].par)
        node[p].son[c] = nq;
    }
  }
}
int main(){
  static char s[N];
  scanf("%s", s + 1);
  tot = 0; root = las = newNode();
  int n = strlen(s + 1);
  for(int i = 1;i <= n; i++)
    extend(s[i] - 'a');
  static int cnt[N], order[N << 1];
  memset(cnt, 0, sizeof(*cnt) * (n + 5));
  for(int i = 1; i <= tot; i++) cnt[node[i].len]++;
  for(int i = 1; i <= n; i++) cnt[i] += cnt[i - 1];
  for(int i = tot; i; i--) order[ cnt[node[i].len]-- ] = i;
  static int ANS[N << 1], dp[N << 1];
  memset(dp, 0, sizeof(*dp) * (tot + 5));
  for(int i = 1; i <= tot; i++) ANS[i] = node[i].len;
  while(scanf("%s", s + 1) == 1){
    n = strlen(s + 1);
    for(int now = root, len = 0, i = 1; i <= n; i++){
      int c = s[i] - 'a';
      while(now != root && !node[now].son[c])
        now = node[now].par;
      if(node[now].son[c]){
        len = min(len, node[now].len) + 1;
        now = node[now].son[c];
      }
      else len = 0;
      dp[now] = max(dp[now], len);
    }
    for(int i = tot; i; i--){
      int now = order[i];
      dp[node[now].par] = max(dp[node[now].par], dp[now]);
      ANS[now] = min(ANS[now], dp[now]);
      dp[now] = 0;
    }
  }
  int ans = 0;
  for(int i = 1; i<= tot; i++) ans = max(ans, ANS[i]);
  printf("%d\n", ans);
}
```

1.5 后缀数组

注意事项： $\mathcal{O}(n \log n)$ 倍增构造。

```
#define ws wws
const int MAXN = 201010;
int wa[MAXN], wb[MAXN], wv[MAXN], ws[MAXN];
int sa[MAXN], rk[MAXN], height[MAXN];
char s[MAXN];
inline bool cmp(int *r, int a, int b, int l)
```

```
{return r[a] == r[b] && r[a + l] == r[b + l];}
void SA(char *r, int *sa, int n, int m){
  int *x = wa, *y = wb;
  for(int i = 1; i <= m; i++)ws[i] = 0;
  for(int i = 1; i <= n; i++)ws[x[i] = r[i]]++;
  for(int i = 1; i <= m; i++)ws[i] += ws[i - 1];
  for(int i = n; i > 0; i--)sa[ ws[x[i]]-- ] = i;
  for(int j = 1, p = 0; p < n; j <<= 1, m = p){
    p = 0;
    for(int i = n - j + 1; i <= n; i++)y[++p] = i;
    for(int i = 1; i <= n; i++)if(sa[i] > j) y[++p] = sa[i] - j;
    for(int i = 1; i <= n; i++)wv[i] = x[y[i]];
    for(int i = 1; i <= m; i++)ws[i] = 0;
    for(int i = 1; i <= n; i++)ws[wv[i]]++;
    for(int i = 1; i <= m; i++)ws[i] += ws[i - 1];
    for(int i = n; i > 0; i--)sa[ ws[wv[i]]-- ] = y[i];
    swap(x, y); x[sa[1]] = p = 1;
    for(int i = 2; i <= n; i++)
      x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p : ++p;
  }
}
void getheight(char *r, int *sa, int *rk, int *h, int n){
  for(int i = 1; i <= n; i++) rk[sa[i]] = i;
  for(int i = 1, p = 0; i <= n; i++, p ? p-- : 0){
    int j = sa[rk[i] - 1];
    while(r[i + p] == r[j + p]) p++;
    h[rk[i]] = p;
  }
}
```

1.6 Manacher

注意事项： 1-based 算法，请注意下标。

```
void manacher(char *st){
  static char s[N << 1];
  static int p[N << 1];
  int n = strlen(st + 1);
  s[0] = '$'; s[1] = '#';
  for(int i = 1; i <= n; i++)
    s[i << 1] = st[i], s[(i << 1) + 1] = '#';
  s[(n = n * 2 + 1) + 1] = 0;
  int pos, mx = 0, res = 0;
  for(int i = 1; i <= n; i++){
    p[i] = (mx > i) ? min(p[pos * 2 - i], mx - i) : 1;
    while(s[i + p[i]] == s[i - p[i]]) p[i]++;
    if(p[i] + i - 1 > mx) mx = p[i] + i - 1, pos = i;
  }
}
```

1.7 循环串的最小表示

注意事项： 0-Based 算法，请注意下标。

```
int getmin(char *s, int n){// 0-base
  int i = 0, j = 1, k = 0;
  while(i < n && j < n && k < n){
    int x = i + k; if(x >= n) x -= n;
    int y = j + k; if(y >= n) y -= n;
    if(s[x] == s[y]) k++;
    else{
      if(s[x] > s[y]) i += k + 1;
      else j += k + 1;
      if(i == j) j++;
      k = 0;
    }
  }
  return min(i ,j);
}
```