字符串

```
/* KMP */
int k = 0; nxt[1] = 0, nxt[0] = -1;
for(int i = 2; i <= m; i++){
    while(k ^ -1 && t[k + 1] != t[i]) k = nxt[k];
    nxt[i] = (k += 1);
}
k = 0;
for(int i = 1; i <= n; i++){
    while(k ^ -1 && t[k + 1] != s[i]) k = nxt[k];
    if((k += 1) == m) printf("%d", i - m + 1), puts("");
}

/* Tire and ACAM */
const int CN = 1e6 + 6;
class ACAM {
  public: int son[CN][26], fail[CN], e[CN], idx; queue<int> Q;
    void ins(char s[]){ // 建立 tire 结构
        int u = 0;
        for(int i = 0; s[i]; i++){
            if(!son[u][s[i] - 'a']) son[u][s[i] - 'a'] = ++idx;
            u = son[u][s[i] - 'a'];
        }
        e[u]++;
    }
    void bd(){ // 建立 fail 指针
        for(int i = 0; i < 26; i++) if(son[0][i]) Q.push(son[0][i]);
        while(!Q.empty()){
            int u = Q.front(); Q.pop();
            for(int i = 0; i < 26; i++){
                if(son[u][i]) nxt[son[u][i]] = son[nxt[u]][i], Q.push(son[u]
[i]);
                else son[u][i] = son[nxt[u]][i];
            }
        }
    }
} D;

/* 后缀数组 SA */
/* 后缀数组通过将后缀按字典序排序来获得一些优美的性质 */
int rk[CN << 1], sa[CN], ht[CN];
/* rk[] 将 s[i:] 映射到排序后的排名 rk[i]，有类似于离散化的作用 */
namespace SA{
    int prk[CN << 1], id[CN], px[CN], cnt[CN];
    void sort(char a[], int n){
        int m = max(n, 300);
        for(int i = 1; i <= n; i++) rk[i] = a[i];
        for(int i = 1; i <= n; i++) cnt[ rk[i] ]++;
        for(int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for(int i = n; i; i--) sa[ cnt[ rk[i] ]-- ] = i; // id[i] = i
        for(int w = 1; w < n; w <<= 1){
            memset(cnt, 0, sizeof(cnt));
```

```
            for(int i = 1; i <= n; i++) id[i] = sa[i];
            for(int i = 1; i <= n; i++) cnt[ px[i] = rk[id[i] + w] ]++;
            for(int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
            for(int i = n; i; i--) sa[ cnt[ px[i] ]-- ] = id[i];
            memset(cnt, 0, sizeof(cnt));
            for(int i = 1; i <= n; i++) id[i] = sa[i];
            for(int i = 1; i <= n; i++) cnt[ px[i] = rk[id[i]] ]++;
            for(int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
            for(int i = n; i; i--) sa[ cnt[ px[i] ]-- ] = id[i];
            memcpy(prk, rk, sizeof(rk)), m = 0;
            for(int i = 1; i <= n; i++)
                if(prk[sa[i]] == prk[sa[i - 1]] && prk[sa[i] + w] == prk[sa[i -
1] + w])
                    rk[ sa[i] ] = m;
                else rk[ sa[i] ] = ++m;
            if(m == n) break;
        }
        for(int p = 0, i = 1; i <= n; i++){
            if(p) p--;
            while(a[i + p] == a[sa[rk[i] - 1] + p]) p++;
            ht[ rk[i] ] = p;
        }
    }
}

/* SAM */
/* 后缀自动机可以将字符串的每一个子串双射在有向单词无环图（DAWG）上，从而获得一系列优美的性质
*/
const int CN = 1e6 + 6;
class SAM{
  public: int len[CN << 1], nxt[CN << 1], last, sz; int son[CN << 1][26];
    SAM() {memset(son, 0, sizeof(son)), len[0] = 0, nxt[0] = -1, sz = 1, last =
0;}
    void extend(int c){
        int u = sz++, p = last;
        last = u, len[u] = len[p] + 1;
        while(p != -1 && !son[p][c]) son[p][c] = u, p = nxt[p];
        if(p == -1) return (void)(nxt[u] = 0);
        int d = son[p][c];
        if(len[d] == len[p] + 1) return (void)(nxt[u] = d);
        int v = sz++;
        len[v] = len[p] + 1, nxt[v] = nxt[d], nxt[u] = nxt[d] = v;
        memcpy(son[v], son[d], sizeof(son[d]));
        while(p != -1 && son[p][c] == d) son[p][c] = v, p = nxt[p];
    }
};

/* 广义SAM 多串 */
/* 先建立 Tire 树的结构，再在 Tire 树上建立 Parent 树和 DAWG 即可。注意在 SAM 的结构建立完
成后，原有的 Tire 结构会被破坏 */
class PAIR {public: int x, y;};
PAIR mp(int x, int y) {PAIR o; o.x = x, o.y = y; return o;}
class SAM {
  public: int nxt[CN << 1], son[CN << 1][26], len[CN << 1], idx; queue<PAIR> Q;
    SAM() {nxt[0] = -1;}
```

```
    int et(int p, int c){ // 在 SAM 结构上的 p 节点后扩展出字符 c
        int u = son[p][c]; if(len[u]) return u;
        len[u] = len[p] + 1, p = nxt[p];
        while(p ^ -1 && !son[p][c]) son[p][c] = u, p = nxt[p];
        if(p == -1) return u;
        int d = son[p][c];
        if(len[d] == len[p] + 1) return nxt[u] = d, u;
        int v = ++idx;
        len[v] = len[p] + 1, nxt[v] = nxt[d], nxt[d] = nxt[u] = v;
        for(int i = 0; i < 26; i++) if(len[son[d][i]]) son[v][i] = son[d][i];
        while(p ^ -1 && son[p][c] == d) son[p][c] = v, p = nxt[p];
        return u;
    }
    void ins(char ch[]){
        int u = 0;
        for(int i = 1; ch[i]; i++){
            if(!son[u][ch[i] - 'a']) son[u][ch[i] - 'a'] = ++idx;
            u = son[u][ch[i] - 'a'];
        }
    }
    void bd(){
        for(int i = 0; i < 26; i++) if(son[0][i]) Q.push(mp(0, i));
        while(!Q.empty()){
            int u, x = Q.front().x, y = Q.front().y; Q.pop();
            u = et(x, y);
            for(int i = 0; i < 26; i++) if(son[u][i]) Q.push(mp(u, i));
        }
    }
} ;

/* Manacher 求回文半径 */
cin >> (s + 1); n = strlen(s + 1);
c[0] = '$', c[1] = '#';
for(int i = 1; i <= n; i++) c[i << 1] = s[i], c[i << 1 | 1] = '#';
n = n << 1 | 1, c[n + 1] = '.';
int k, i0 = 1; r[1] = 1;
for(int i = 2; i <= n; i++){
    k = min(i0 + r[i0] - i, r[(i0 << 1) - i]);
    while(c[i + k] == c[i - k]) k++;
    r[i] = k; if(i + r[i] > i0 + r[i0]) i0 = i;
}

/* PAM */
int n; char ch[CN];
class PAM{
  public: int len[CN], nxt[CN], son[CN][26], idx, lst;
    PAM() {len[0] = nxt[0] = -1, idx = lst = 1;}
    void et(int r){
        int p = lst;
        while(ch[r - len[p] - 1] ^ ch[r]) p = nxt[p];
        if(!son[p][ch[r] - 'a']){
            son[p][ch[r] - 'a'] = ++idx, len[idx] = len[p] + 2;
            int pp = nxt[p];
            while(pp ^ -1 && ch[r - len[pp] - 1] ^ ch[r]) pp = nxt[pp];
            nxt[idx] = pp ^ -1 ? son[pp][ch[r] - 'a'] : 1;
```

```
        }
        lst = son[p][ch[r] - 'a'];
    }
} ;
```