第一章 字符串目录

1、a 字符串增删改变 b 字符串的最小操作数

```
const int N = 1e3 + 5;
int T, cas = 0;
int n, m;
int dp[N][N];
char s[N], t[N];
int main(){
    while (scanf("%s%s", s, t) != EOF){
        int n = (int)strlen(s), m = (int)strlen(t);
        for (int i = 0; i <= n; i++)    dp[i][0] = i;
        for (int i = 0; i <= m; i++)    dp[0][i] = i;
        for (int i = 1; i <= n; i++){
            for (int j = 1; j <= m; j++){
                dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + 1;
                dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + (s[i - 1] != t[j - 1]));
            }
        }
        printf("%d\n", dp[n][m]);
    }
    return 0;
}
```

2、KMP

```
/*字符串匹配。给你两个字符串，寻找其中一个字符串是否包含另一个字符串，
  如果包含，返回包含的起始位置。*/
 /* next[]的含义，x[i - next[i]...i - 1] = x[0...next[i] - 1]
 * next[i]为满足 x[i - z...i - 1] = x[0...z - 1]的最大 z 值（就是 x 的自身匹配）*/
void KMP_Pre(char x[], int m, int next[]){
    int i, j;
    j = next[0] = -1;
    i = 0;
    while (i < m){
        while (-1 != j && x[i] != x[j])    j = next[j];
        next[++i] = ++j;
    }
    return ;
}
/* kmpNext[]的意思：next'[i] = next[next[...[next[i]]]]
 *  （直到 next'[i] < 0 或者 x[next'[i]] != x[i]）
 *  这样的预处理可以快一些 */
void preKMP(char x[], int m, int kmpNext[]){
    int i, j;
    j = kmpNext[0] = -1;
    i = 0;
```

```
    while (i < m){
        while (-1 != j && x[i] != x[j])     j = kmpNext[j];
        if (x[++i] == x[++j])       kmpNext[i] = kmpNext[j];
        else        kmpNext[i] = j;
    }
    return ;
}
/*此函数与上述两个函数中的任意一个搭配使用（即调用上述两个函数中的任意一个）
 * 返回 x 在 y 中出现的次数，可以重叠 */
int next[10010];
int KMP_Count(char x[], int m, char y[], int n){
    //  x 是模式串，y 是主串
    int i, j;
    int ans = 0;
    //  preKMP(x, m, next);
    KMP_Pre(x, m, next);
    i = j = 0;
    while (i < n){
        while (-1 != j && y[i] != x[j])     j = next[j];
        i++, j++;
        if (j >= m){
            ans++;
            j = next[j];
        }
    }
    return ans;
}
int main(){
    int ans = 0;
    char y[] = "ccabcabcaa";
    char a[] = "abc";
    int leny = strlen(y);
    int lena = strlen(a);
    ans = KMP_Count(a,lena,y,leny);//统计 a 在 y 中出现的次数
    cout << ans << endl;
}
```

3、扩展 kmp

```
int IT_MAX = 1 << 19;
int MOD = 1000000007;
const int INF = 0x3f3f3f3f;
const ll LL_INF = 0x3f3f3f3f3f3f3f3f;
const db PI = acos(-1);
const db ERR = 1e-10;
```

```
const int MAX_N = 100005;
bool cmp (int a , int b){
/*扩展 KMP 解决的问题：
定义母串 S 和子串 T，S 的长度为 n，T 的长度为 m；
求字符串 T 与字符串 S 的每一个后缀的最长公共前缀；
也就是说，设有 extend 数组：
extend[i]表示 T 与 S[i,n-1]的最长公共前缀，要求出所有 extend[i](0<=i<n)。*/
/*扩展 KMP
 * next[i]:x[i...m-1]的最长公共前缀
 * extend[i]:y[i...n-1]与 x[0...m-1]的最长公共前缀 */
void preEKMP(char x[], int m, int next[]){
    next[0] = m;
    int j = 0;
    while (j + 1 < m && x[j] == x[j + 1])       j++;
    next[1] = j;
    int k = 1;
    for (int i = 2; i < m; i++){
        int p = next[k] + k - 1;
        int L = next[i - k];
        if (i + L < p + 1)      next[i] = L;
        else{
            j = std::max(0, p - i + 1);
            while (i + j < m && x[i + j] == x[j])       j++;
            next[i] = j;
            k = i;
        }
    }
    return ;
}
void EKMP(char x[], int m, char y[], int n, int next[], int extend[]){
    preEKMP(x, m, next);
    int j = 0;
    while (j < n && j < m && x[j] == y[j])       j++;
    extend[0] = j;
    int k = 0;
    for (int i = 1; i < n; i++){
        int p = extend[k] + k - 1;
        int L = next[i - k];
        if (i + L < p + 1)       extend[i] = L;
        else{
            j = std::max(0, p - i + 1);
            while (i + j < n && j < m && y[i + j] == x[j])       j++;
            extend[i] = j;
            k = i;
```

```
            }
        }
        return ;
}
int main(){
        char y[] = "aaaabcaaaaa";
        char a[] = "aaaaa";
        int next[1000],extend[1000];
        int leny = strlen(y);
        int lena = strlen(a);
        EKMP(a,lena,y,leny,next,extend) ;
        for(int i = 0 ; i < 10 ; i ++)    cout << next[i] << " ";
}
```

4、最短公共祖先

```
/*求解类似于 包含 "alba"和"baccdasd" 最短多长的问题;
    答案为 len("albaccdasd") == 10; */
const int N = 1000010;
char a[2][N];
int fail[N];
inline int max(int a, int b){ return (a > b) ? a : b;}
int kmp(int &i, int &j, char* str, char* pat){
        int k;
        memset(fail, -1, sizeof(fail));
        for (i = 1; pat[i]; ++i){
                for (k = fail[i - 1]; k >= 0 && pat[i] != pat[k + 1]; k = fail[k]);
                if (pat[k + 1] == pat[i])      fail[i] = k + 1;
        }
        i = j = 0;
        while (str[i] && pat[j]){
                if (pat[j] == str[i]){ i++;
                else if (j == 0)        i++;        //   第一个字符匹配失败，从 str 下一个字符开始
                else        j = fail[j - 1] + 1;
        }
        if (pat[j])      return -1;
        else        return i - j;
}
int main(int argc, const char * argv[]){
        int T;
        scanf("%d", &T);
        while (T--){
                int i, j, l1 = 0, l2 = 0;
                cin >> a[0] >> a[1];
                int len1 = (int)strlen(a[0]), len2 = (int)strlen(a[1]), val;
```

```
            val = kmp(i, j, a[1], a[0]);                    //    a[1]在前
            if (val != -1)        l1 = len1;
            else{
//printf("i:%d, j:%d\n", i, j);
                if (i == len2 && j - 1 >= 0 && a[1][len2 - 1] == a[0][j - 1])      l1 = j;
            }
            val = kmp(i, j, a[0], a[1]);                    //    a[0]在前
            if (val != -1)      l2 = len2;
            else{
//printf("i:%d, j:%d\n", i, j);
                if (i == len1 && j - 1 >= 0 && a[0][len1 - 1] == a[1][j - 1]) l2 = j;
            }
//printf("l1:%d,l2:%d\n",l1,l2);
            printf("%d\n", len1 + len2 - max(l1, l2));
        }
        return 0;
}
```

5、AC

```
/*给你一个字典树,再给你一个字符串 ，查询树上的东西出现了几次 */
struct Trie{
    int next[500010][26], fail[500010], end[500010];
    int root, L;
    int newnode(){
        for (int i = 0; i < 26; i++)      next[L][i] = -1;
        end[L++] = 0;
        return L - 1;
    }
    void init(){
        L = 0;
        root = newnode();
    }
    void insert(char buf[]){
        int len = (int)strlen(buf);
        int now = root;
        for (int i = 0; i < len; i++){
            if (next[now][buf[i] - 'a'] == -1)    next[now][buf[i] - 'a'] = newnode();
            now = next[now][buf[i] - 'a'];
        }
        end[now]++;
    }
    void build(){
        queue<int>Q;
        fail[root] = root;
```

```
        for (int i = 0; i < 26; i++){
            if (next[root][i] == -1)      next[root][i] = root;
            else{
                fail[next[root][i]] = root;
                Q.push(next[root][i]);
            }
        }
        while (!Q.empty()){
            int now = Q.front();
            Q.pop();
            for (int i = 0;i < 26;i++){
                if (next[now][i] == -1)        next[now][i] = next[fail[now]][i];
                else{
                    fail[next[now][i]]=next[fail[now]][i];
                    Q.push(next[now][i]);
                }
            }
        }
    }
    int query(char buf[]){
        int len = (int)strlen(buf);
        int now = root;
        int res = 0;
        for (int i = 0; i < len; i++){
            now = next[now][buf[i] - 'a'];
            int temp = now;
            while (temp != root){
                res += end[temp];
                end[temp] = 0;
                temp = fail[temp];
            }
        }
        return res;
    }
    void debug(){
        for (int i = 0; i < L; i++){
            printf("id = %3d,fail = %3d,end = %3d,chi = [", i, fail[i], end[i]);
            for (int j = 0; j < 26; j++)      printf("%2d", next[i][j]);
            printf("]\n");
        }
    }
};
char buf[1000010];
Trie ac;
```

```
int main(){
	int T;
	int n;
	scanf("%d", &T);
	while(T--){
		scanf("%d", &n);
		ac.init();
		for (int i = 0; i < n; i++){
			scanf("%s", buf);
			ac.insert(buf);
		}
		ac.build();
		scanf("%s", buf);
		printf("%d\n", ac.query(buf));
	}
	return 0;
}
```

6、kmp 简易版 c++函数

```
/*    strstr 函数
 *   功能：在串中查找指定字符串的第一次出现
 *   用法：char *strstr(char *strOne, char *strTwo);
 *   据说 strstr 函数和 KMP 的算法效率差不多*/
int main(int argc, const char * argv[]){
	char strOne[] = "Borland International";
	char strTwo[] = "nation";
	char *ptr;
	ptr = strstr(strOne, strTwo);
	std::cout << ptr << '\n';
	return 0;
}
```

7、KR 字符串匹配(预处理,kmp 失败可用)

```
// Rabin Karp Algorithm
/*未调试 */
void Rabin_Karp_search(const string &T, const string &P, int d, int q){
	int m = P.length();
	int n = T.length();
	int i, j;
	int p = 0;    // hash value for pattern
	int t = 0; // hash value for txt
	int h = 1;
	// The value of h would be "pow(d, M-1)%q"
	for (i = 0; i < m-1; i++)
```

```
        h = (h*d)%q;
    // Calculate the hash value of pattern and first window of text
    for (i = 0; i < m; i++){
        p = (d*p + P[i])%q;
        t = (d*t + T[i])%q;
    }
    // Slide the pattern over text one by one
    for (i = 0; i <= n - m; i++){
        // Chaeck the hash values of current window of text and pattern
        // If the hash values match then only check for characters on by one
        if ( p == t ){
            /* Check for characters one by one */
            for (j = 0; j < m; j++) if (T[i+j] != P[j])    break;
            if (j == m)    // if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]
                cout<<"Pattern found at index :"<< i<<endl;
        }
        // Calulate hash value for next window of text: Remove leading digit,
        // add trailing digit
        if ( i < n-m ){
            t = (d*(t - T[i]*h) + T[i+m])%q;
            // We might get negative value of t, converting it to positive
            if(t < 0)        t = (t + q);
        }
    }
}
int main(){
    string T = "Rabin – Karp string search algorithm: Rabin-Karp";
    string P = "Rabin";
    int q = 101;    // A prime number
    int d=16;
    Rabin_Karp_search(T,P,d,q);
    system("pause");
    return 0;
}
```

8、最长回文子串

```
/*求最长回文子串*/
const int MAXN = 110010;
char A[MAXN * 2];
int B[MAXN * 2];
void Manacher(char s[], int len){
    int l = 0;
    A[l++] = '$';     //0 下标存储为其他字符
    A[l++] = '#';
```

```
    for (int i = 0; i < len; i++){
        A[l++] = s[i];
        A[l++] = '#';
    }
    A[l] = 0;            //空字符
    int mx = 0;
    int id = 0;
    for (int i = 0; i < l; i++){
        B[i] = mx > i ? std::min(B[2 * id - i], mx - i) : 1;
        while (A[i + B[i]] == A[i - B[i]])     B[i]++;
        if (i + B[i] > mx){
            mx = i + B[i];
            id = i;
        }
    }
    return ;
}
 /* abaaba
 * i:    0   1   2   3   4   5   6   7   8   9 10 11 12 13 14
 * A:    $   #   a   #   b   #   a   #   a   #   b   #   a   # '\0'
 * B:    1   1   2   1   4   1   2   7   2   1   4   1   2   1      //以第 i 个为中心的回文半径（包括第 i 个）*/
char s[MAXN];
int main(int argc, const char * argv[]){
    while (std::cin >> s){
        int len = (int)strlen(s);
        Manacher(s, len);
        int ans = 0;
        for (int i = 0; i < 2 * len + 2; i++) {   //两倍长度并且首位插有字符，所以 i < 2 * len + 2
            ans = std::max(ans, B[i] - 1);
        }
        std::cout << ans << std::endl;
    }
    return 0;
}
```

9、后缀自动机

```
const int CHAR = 26;
const int MAXN = 250010;
struct SAM_Node{
    SAM_Node *fa, *next[CHAR];
    int len;
    int id, pos;
    SAM_Node(){}
```

```
    SAM_Node(int _len){
        fa = 0;
        len = _len;
        memset(next, 0, sizeof(next));
    }
};
SAM_Node SAM_node[MAXN * 2], *SAM_root, *SAM_last;
int SAM_size;
SAM_Node *newSAM_Node(int len){
    SAM_node[SAM_size] = SAM_Node(len);
    SAM_node[SAM_size].id = SAM_size;
    return &SAM_node[SAM_size++];
}
SAM_Node *newSAM_Node(SAM_Node *p){
    SAM_node[SAM_size] = *p; SAM_node[SAM_size].id = SAM_size;
    return &SAM_node[SAM_size++];
}
void SAM_init(){
    SAM_size = 0;
    SAM_root = SAM_last = newSAM_Node(0);
    SAM_node[0].pos = 0;
}
void SAM_add(int x, int len){
    SAM_Node *p = SAM_last, *np = newSAM_Node(p->len+1);
    np->pos = len;
    SAM_last = np;
    for (; p && !p->next[x]; p = p->fa)    p->next[x] = np;
    if (!p){np->fa = SAM_root; return ;}
    SAM_Node *q = p->next[x];
    if (q->len == p->len + 1){np->fa = q;    return ;}
    SAM_Node *nq = newSAM_Node(q);
    nq->len = p->len + 1;
    q->fa = nq;
    np->fa = nq;
    for(;p && p->next[x] == q; p = p->fa)
        p->next[x] = nq;
}
void SAM_build(char *s){
    SAM_init();
    int len = (int)strlen(s);
    for (int i = 0; i < len; i++)      SAM_add(s[i] - 'a', i + 1);
}
/*
 //  加入串后进行拓扑排序
```

```
  char str[MAXN];
  int topocnt[MAXN];
  SAM_Node *topsam[MAXN * 2];
  int n = (int)strlen(str);
  SAM_build(str);
  memset(topocnt, 0, sizeof(topocnt));
  for (int i = 0; i < SAM_size; i++)    topocnt[SAM_node[i].len]++;
  for (int i = 1; i <= n; i++)      topocnt[i] += topocnt[i-1];
  for (int i = 0; i < SAM_size; i++)    topsam[--topocnt[SAM_node[i].len]] = &SAM_node[i];
  */
//   多串的建立:
//   多串的建立,注意 SAM_init()的调用
//void SAM_build(char *s){
//      int len = (int)strlen(s);
//      SAM_last = SAM_root;
//      for (int i = 0; i < len; i++){
//            if (!SAM_last->next[s[i] - '0'] || !(SAM_last->next[s[i] - '0']->len == i+1))
SAM_add(s[i] - '0',i+1);
//            else        SAM_last = SAM_last->next[s[i] - '0'];
//      }
//}
int main(){
      char c[] = "abacacad";
      SAM_build(c);
      int i = 1;
      while(i != SAM_size){
            cout << SAM_node[i].id << " " << SAM_node[i].len << " " << SAM_node[i].pos << endl;
            i ++;
      }
}
```