# Hash

LL strhash (const char \*str) {

    LL c,i,h;

    for (i=h=0; str[i]; i++)    {

        c = str[i] - 'a';

        h = ( (h << 5) + h) ^ c;

    }

    return h;

}

17,37, 79, 163, 331,

673,1361,2729,5471,10949,

21911,43853, 87719, 175447, 350899,

701819,1403641,2807303,5614657, 11229331,

22458671, 44917381, 89834777, 179669557, 359339171,

718678369, 1437356741, 2147483647

# KMP

void GetNext(char\* T, int \* next)

{

    next[0]=-1; next[1] = 0;

    for(int i = 1,j = 0; T[i]; ){

        while(j != -1 && T[i] != T[j])j = next[j];

        i++; j++;

        next[i] = j;

    }

}

void KMP(char \* S, char \* T, int \* next) {

    int LT = strlen(T);

    ans = 0;

    for(int i = 0,j = 0; S[i]; i++){

        while(j != -1 && S[i] != T[j]) j = next[j];

        j++;

        if(j == LT){

            ans++;

            j = next[j];

        }

    }

}

# ExKMP

void preExKmp (char T[],int LT,int next[]) {

int ind = 0,k = 1;

next[0] = LT; //特殊值

while (ind + 1 < LT && T[ind + 1] == T[ind]) ++ind;

next[1] = ind;

for (int i = 2; i < LT; ++i) {

if (i <= k + next[k] - 1 && next[i - k] + i < k + next[k])

next[i] = next[i - k];

else {

ind = max(0, k + next[k] - i);

while (ind + i < LT && T[ind + i] == T[ind]) ++ind;

next[i] = ind, k = i;

}

}

}

void exKmp ( char S[], int LS, char T[], int LT ,int next[], int B[]) {

preExKmp (T, LT, next);

int ind = 0, k = 0;

while (ind < LS && ind < LT && T[ind] == S[ind]) ind++;

B[0] = ind;

for (int i = 1; i < LS; ++i) {

if (i < k + B[k] - 1 && next[i - k] + i < k + B[k])

B[i] = next[i - k];

else {

ind = max (0,k + B[k] - i);

while (ind + i < LS && ind < LT && S[ind + i] == T[ind]) ++ind;

B[i] = ind; //公共长度

k = i;

}

}

}

# Manacher

//HDU\_3068

const int N=1200000;

int len, p[N];

char s[N], str[N];

void Manacher(char str[], int p[])

{

    int i;

    int mx = 0;

    int id;

    for(i=len; i < N; i++)

        str[i] = 0; //没有这一句有问题。。就过不了ural1297，比如数据：ababa aba

    for(i=1; i<len; i++)

    {

        if( mx > i )

            p[i] = min( p[2\*id-i], p[id]+id-i );

        else

            p[i] = 1;

        for(; str[i+p[i]] == str[i-p[i]]; p[i]++)

            ;

        if( p[i] + i > mx )

        {

            mx = p[i] + i;

            id = i;

        }

    }

}

void init()

{

    int i, j, k;

    str[0] = '$';

    str[1] = '#';

    for(i=0; i<len; i++)

    {

        str[i\*2+2] = s[i];

        str[i\*2+3] = '#';

    }

    len = len\*2+2;

    s[len] = 0;

}

int main()

{

    int i, ans;

    while(scanf("%s", s)!=EOF)

    {

        len = strlen(s);

        init();

        Manacher(str, p);

        ans = 0;

        for(i=0; i<len; i++)

            if(p[i]>ans)

                ans = p[i];

        printf("%d\n", ans-1);

    }

    return 0;

}

# AC自动机

int n, ans;

int head, tail, tot;

struct TrieNode

{

int dp, visit;

TrieNode \*fail, \*next[KIND];

} trie[LEN], \*root, \*qu[LEN];

int start[N], end[N], value[N];

char str[LEN];

int hash[128], hashId;

TrieNode \*NewTrieNode()

{

for(int i = 0; i < 26; i++) trie[tot].next[i] = NULL;

trie[tot].dp = 0;

trie[tot].visit=-1;

return &trie[tot++];

}

int getId(char ch)

{

if(hash[ch] == -1)

{

hash[ch] = hashId++;

}

return hash[ch];

}

void insert(int L, int R)

{

TrieNode \*p = root;

for(int i = L; i < R; i++)

{

int key = getId(str[i]);

if(p->next[key] == NULL)

p->next[key] = NewTrieNode();

p = p->next[key];

}

}

void BuildAc()

{

head = tail = 0;

qu[tail++] = root;

root->fail = root;

while(head < tail)

{

//cout <<"go"<<head <<endl;

TrieNode \*now = qu[head++];

for(int i = 0; i < hashId; i++)

{

if(now->next[i] != NULL)

{

if(now == root) now->next[i]->fail = root;

else now->next[i]->fail = now->fail->next[i];

qu[tail++] = now->next[i];

}

else

{

if(now == root) now->next[i] = root;

else now->next[i] = now->fail->next[i];

}

}

}

}

void Query(int L, int R, int value, int num)

{

int temp = 0;

TrieNode \*p = root;

for(int i = L; i < R; i++)

{

int key = getId(str[i]);

p = p->next[key];

TrieNode \*q = p;

while(q != root&&q->visit != num)

{

temp = max(temp, q->dp);

q->visit = num;

q = q->fail;

}

}

p->dp = max(p->dp, temp + value);

ans = max(ans, p->dp);

}

int main()

{

//freopen("input.txt", "r", stdin);

scanf("%d", &Test);

Case = 0;

while(Test--)

{

tot = 0;

int len = 0;

for(int i = 0; i < 128; i++) hash[i] = -1;

hashId = 0;

root = NewTrieNode();

scanf("%d", &n);

//cout <<"go"<<endl;

for(int i = 0; i < n; i++)

{

scanf("%s%d", str+len, &value[i]);

start[i] = len; len = strlen(str); end[i] = len;

insert(start[i], end[i]);

}

//cout <<"go"<<endl;

//printf("%s\n", str);

//for(int i = 0; i < n; i++) printf("%d %d\n", Start[i], End[i]); //putchar('\n');

BuildAc();

//cout <<"go"<<endl;

ans = 0;

for(int i = 0; i < n; i++)

if(value[i] > 0 )

Query(start[i], end[i], value[i], i);

printf("Case #%d: %d\n", ++Case, ans);

}

return 0;

}

char str[LEN], words[25];

int ans, n, cnt = 0;

int head, tail, sum;

struct node

{

    int id, flag;

    struct node\* next[KIND];

    struct node\* fail;

}\*root, \*qu[1010], point[1010];

int f[1010][1010];

int getKey(char ch)

{

    switch(ch)

    {

        case 'A': return 0;

        case 'C': return 1;

        case 'G': return 2;

        default : return 3;

    }

}

node \* newNode()

{

    node \*p = &point[sum];

    p->id = sum++;

    p->fail = NULL;

    memset(p->next, NULL, sizeof(p->next));

    p->flag = 0;

    return p;

}

void insert(char s[])

{

    node \*p = root;

    int i = 0, len = strlen(s), key;

    while(i < len)

    {

        key = getKey(s[i]);

        if(p->next[key] == NULL)

            p->next[key] = newNode();

        p = p->next[key];

        i++;

    }

    p->flag = 1;

}

void build\_AC\_automation()

{

    node \*now;

    head = tail = 0;

    qu[tail++] = root;

    root->fail = NULL;

    while(head < tail)

    {

        now = qu[head++];

        for(int i = 0; i < KIND; i++)

        {

            if(now->next[i] != NULL)

            {

                if(now == root) now->next[i]->fail = root;

                else

                {

                    now->next[i]->fail = now->fail->next[i];

                    if(now->next[i]->fail->flag) now->next[i]->flag = 1;

                }

                qu[tail++] = now->next[i];

            }

            else

            {

                if(now == root) now->next[i] = root;

                else now->next[i] = now->fail->next[i];

            }

        }

    }

}

void dp()

{

    int len = strlen(str);

    for(int i = 0; i <= len; i++)

        for(int j = 0; j <= sum; j++)

            f[i][j] = INF;

    f[0][0] = 0;

    for(int i = 1; i <= len; i++)

        for(int j = 0; j <sum; j++)

        if(f[i-1][j] < INF)

        {

            for(int k = 0; k < KIND; k++)

                if(point[j].next[k]->flag != 1)

                {

                    node \*p = point[j].next[k];

                    f[i][p->id] = min(f[i][p->id], f[i-1][j] + (getKey(str[i-1]) != k));

                }

        }

    ans = INF;

    for(int j = 0; j < sum; j++)

        ans = min(ans, f[len][j]);

    if(ans >= INF) ans = -1;

}

int main()

{

    //freopen("input.txt", "r", stdin);

    while(scanf("%d", &n) != EOF && n)

    {

        sum = 0;

        root = newNode();

        for(int i = 0; i < n; i++)

        {

            scanf("%s", words);

            insert(words);

        }

        //cout <<"build"<<endl;

        build\_AC\_automation();

        //cout <<"build"<<endl;

        scanf("%s", str);

        dp();

        //cout <<"build"<<endl;

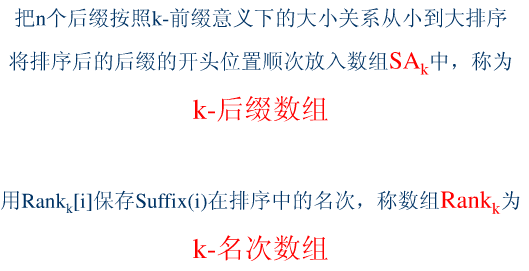
        printf("Case %d: %d\n", ++cnt, ans);

    }

    return 0;

}

# 后缀数组



1、suffix(i) 字符串 s的从 第 i个字符开始的后缀表示为 suffix(i)

2、后缀数组 SA：它保存 1..n的某个排列 SA[1 ] ，SA[2] ， …… ， SA[n] ，并且保证suffix(SA[i])

3、名次数组 Rank[i]：保存的是 Suffix(i)在所有后缀中从小到大排列的 “ 名次 ” 。就是说 在字符串s中下标为i的字符， 以该字符开始的后缀Suffix(i)是第Rank[i]大的后缀。若 sa[i]=j，则 rank[j]=i。后缀数组sa与名次数组rank的关系为互逆关系。

4、height数组：定义height[i]=suffix(sa[i-1])和suffix(sa[i]) 的最长公共前缀，也就是排名相邻的两个后缀的最长公共前缀。

int n;

int num[Max];

int sa[Max], rank[Max], height[Max]; //sa[1~n]value(0~n-1); rank[0..n-1]value(1..n); height[2..n]

int wa[Max], wb[Max], wv[Max], wd[Max];

int cmp( int \*r, int a, int b, int l ) {

return r[a] == r[b] && r[a+l] == r[b+l];

}

void da( int \*r, int n, int m ) { // 倍增算法 r为待匹配数组 n为总长度+1 m为字符范围

int i, j, k=0, p, \*x = wa, \*y = wb, \*t;

for ( i = 0; i < m; i ++ ) wd[i] = 0;

for ( i = 0; i < n; i ++ ) wd[x[i]=r[i]] ++;

for ( i = 1; i < m; i ++ ) wd[i] += wd[i-1];

for ( i = n-1; i >= 0; i -- ) sa[-- wd[x[i]]] = i;

for ( j = 1, p = 1; p < n; j \*= 2, m = p ) {

for ( p = 0, i = n-j; i < n; i ++ ) y[p ++] = i;

for ( i = 0; i < n; i ++ ) if ( sa[i] >= j ) y[p ++] = sa[i] - j;

for ( i = 0; i < n; i ++ ) wv[i] = x[y[i]];

for ( i = 0; i < m; i ++ ) wd[i] = 0;

for ( i = 0; i < n; i ++ ) wd[wv[i]] ++;

for ( i = 1; i < m; i ++ ) wd[i] += wd[i-1];

for ( i = n-1; i >= 0; i -- ) sa[-- wd[wv[i]]] = y[i];

for ( t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i ++ ) {

x[sa[i]] = cmp( y, sa[i-1], sa[i], j ) ? p - 1: p ++;

}

}

for ( i = 0; i < n; i ++ ) rank[sa[i]] = i;

for ( i = 0; i < n-1; height[rank[i ++]] = k ) {

for ( k ? k -- : 0, j = sa[rank[i]-1]; r[i+k] == r[j+k]; k ++ );

}

}

# 后缀树

# 后缀自动机

# 字符串最小表示法