字符串

目录

后缀数组

倍增

```
void sortAndRank(int *a1, int *a2, int n, int &m, int j) {
    int i;
    memset(C, 0, sizeof(C));
    for(i = 0; i < n; i ++)     C[a1[i]] ++;
    for(i = 1; i <=m; i ++)     C[i] += C[i-1];
    for(i = n-1; i >= 0; i --)      sa[--C[a1[a2[i]]]] = a2[i];
    a2[sa[0]] = m = 0;
    for(i = 1; i < n; i ++)
a2[sa[i]] = a1[sa[i-1]]==a1[sa[i]] && a1[sa[i-1]+j]==a1[sa[i]+j] ?
m : ++ m;
}
void da(char*str, int n, int m) {
    int *a1 = A, *a2 = B, *tmp;
    int i, j, p;
    for(i = 0; i < n; i ++) {
        a1[i] = i;
        a2[i] = str[i];
    }
    a1[n] = a2[n] = -1;
    sortAndRank(a2, a1, n, m, 0);
    for(j = 1; m < n-1; j <<= 1) {
        p = 0;
        for(i = n-j; i < n; i ++)   a2[p ++] = i;
        for(i= 0; i < n; i ++)      if(sa[i]>=j)  a2[p ++] = sa[i]-j;
        sortAndRank(a1, a2, n, m, j);
        tmp = a1;  a1 = a2;   a2 = tmp;
    }
    rank = a1; height = a2;
}
void calHeight(char *str, int n) {
    int i, j, k;
    sa[-1] = n;
    for(height[0] = k = i = 0; i < n; i ++) {
        for(k ? k-- : 0, j = sa[rank[i]-1]; str[i+k]==str[j+k]; k++);
        height[rank[i]] = k;
    }
}
```

DC3

```
//DC3, dc3(str,sa,Len + 1,);calHeight(str,Len);
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
//3*maxn!
int wa[maxn],wb[maxn],wv[maxn],ws[maxn],wt[maxn*3];
int *sa=wt+1,*rank,*height;
inline int c0(int *r,int a,int b){
    return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];
}
inline int c12(int k,int *r,int a,int b){
    if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
    else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];
}

void sort(int *r,int *a,int *b,int n,int m){
    int i;
    for(i=0;i<n;i++) wv[i]=r[a[i]];
    for(i=0;i<m;i++) ws[i]=0;
    for(i=0;i<n;i++) ws[wv[i]]++;
```

```
    for(i=1;i<m;i++) ws[i]+=ws[i-1];
    for(i=n-1;i>=0;i--) b[--ws[wv[i]]]=a[i];
}
void dc3(int *r,int *sa,int n,int m){
    int i,j,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p,*rn=r+n;
    r[n]=r[n+1]=0;
    for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;
    sort(r+2,wa,wb,tbc,m);
    sort(r+1,wb,wa,tbc,m);
    sort(r,wa,wb,tbc,m);
    for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
    rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
    if(p<tbc) dc3(rn,san,tbc,p);
    else for(i=0;i<tbc;i++) san[rn[i]]=i;
    for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
    if(n%3==1) wb[ta++]=n-1;
    sort(r,wb,wa,ta,m);
    for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
    for(i=0,j=0,p=0;i<ta && j<tbc;p++)
        sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
    for(;i<ta;p++) sa[p]=wa[i++];
    for(;j<tbc;p++) sa[p]=wb[j++];
    rank=wa;height=wb;
}
void calHeight(int *str, int n){
    int i,j,k;
    for (i=0;i<n;i++) sa[i]=sa[i+1];
    for (i=0;i<n;i++) rank[sa[i]]=i;
    sa[-1]=n;
    for(height[0]=k=i=0;i<n;i++){
        for(k ? k-- : 0, j = sa[rank[i]-1]; str[i+k]==str[j+k]; k++);
        height[rank[i]] = k;
    }
}
```

ST查询LCP

```
//Sparse Table RMQ For LCP with SA
int mm[maxn],*rmq;//让rmq=height
int best[14][maxn];
void Init(int n) {
    int i,j,a,b;
    rmq[0] = -999999999;//让rmq[0]取最反向值
    mm[0]=-1;
    for(i=1; i<=n; i++) {
        mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
        best[0][i]=i;
    }
    for(i=1; i<=mm[n]; i++) {
        for(j=1; j<=n+1-(1<<i); j++) {
            a=best[i-1][j];
            b=best[i-1][j+(1<<(i-1))];
            best[i][j]=rmq[a] < rmq[b]?a:b;
        }
    }
}
int query(int a, int b) {
    if(a > b)  return 0;
    int t;
    t=mm[b-a+1];
    a=best[t][a];
    b=best[t][b-(1<<t)+1];
```

```
    return rmq[a] < rmq[b] ? a : b;
}
int getLCP(int p,int q) {
    if (p>q) return getLCP(q,p);
    return rmq[query(p,q)];
}
```

最长公共子串

```
void init() {
    a[N]='\001';
    a[N+1]=0;
    strcat(a,temp);
    Len=strlen(a);
    da(a,Len,'Z'+1);
    calHeight(a,Len);
}
void work() {
    int res=-1,begin,end;
    for (int i=0;i<Len-1;i++) {
        int t=min(sa[i],sa[i+1]);
        int s=max(sa[i],sa[i+1]);
        if (t>=0 && t<N && s>=N+1 && s<Len) {
            if (res<height[i+1]) {
                res=height[i+1];
                begin=sa[i];
                end=sa[i]+res-1;
            }
        }
    }
    a[end+1]=0;
    printf("%s\n",a+begin);
}
```

最长回文子串

```
void init() {
    scanf("%s", a);
    N = strlen(a);
    a[N] = 1;
    int i, j;
    for (i = N - 1, j = N + 1; i >= 0; i--, j++) {
        hash[i] = j;
        a[j] = a[i];
    }
    a[j] = 0;
    Len = strlen(a);
    da(a, Len, 'z' + 1);
    calHeight(a, Len);
    rmq = height;
    Init( Len);
}
void work() {
    int index = 0, max = 1, type = 1;
    for (int i = 1; i < N; i++) {
        int t = getLCA(rank[i + 1], rank[hash[i - 1]]);
        int len = 2 * t + 1;
        if (max < len) {
            max = len;
            index = i;
            type = 1;
        }
```

```
    }
    for (int i = 0; i < N; i++) {
        int t = getLCA(rank[i + 1], rank[hash[i]]);
        int len = 2 * t;
        if (max < len) {
            max = len;
            index = i;
            type = 2;
        }
    }
    if (type == 1) {
        int len = (max - 1) / 2;
        for (int i = index - len; i <= index + len; i++)
            printf("%c", a[i]);
    } else {
        int len = max / 2;
        for (int i = index - len + 1; i <= index + 1 + len - 1; i++)
            printf("%c", a[i]);
    }
    printf("\n");
}
```

子串计数

```
void init() {
    scanf("%s", a);
    N = strlen(a);
    da(a, N, 'z' + 1);
    calHeight(a, N);
}
void work() {
    int len = 0;
    for (int i = 0; i < N; i++) {
        len += N - i - height[i];
    }
    printf("%d\n", len);
}
```

找每个串最短子串使得不出现在其他串中, Ural 1713

```
void init() {
    scanf("%d", &N);
    char temp[110];
    for (int i = 1; i <= N; i++) {
        scanf("%s", temp);
        start[i] = Len;
        for (int j = 0; temp[j]; j++) {
            a[Len++] = temp[j];
            len[i]++;
        }
        a[Len++] = i + 1000;
    }
    dc3(a, sa, Len, 1000 + N + 1);
    calHeight(a, Len);
    memcpy(rmq + 1, height + 1, maxn * sizeof(int));
    Init( Len);
    first[0].max = first[0].min = rank[0];
    for (int i = 1; i < Len; i++) {
        if (rank[i] > first[i - 1].max)
            first[i].max = rank[i];
        else
            first[i].max = first[i - 1].max;
```

```
        if (rank[i] < first[i - 1].min)
            first[i].min = rank[i];
        else
            first[i].min = first[i - 1].min;
    }
    last[Len - 1].max = last[Len - 1].min = rank[Len - 1];
    for (int i = Len - 2; i >= 0; i--) {
        if (rank[i] > last[i + 1].max)
            last[i].max = rank[i];
        else
            last[i].max = last[i + 1].max;
        if (rank[i] < last[i + 1].min)
            last[i].min = rank[i];
        else
            last[i].min = last[i + 1].min;
    }
}
void work() {
    for (int i = 1; i <= N; i++) {
        int alen = 1000, astart;
        for (int j = start[i]; j < start[i] + len[i]; j++) {
            int res = -1;
            if (start[i] >= 1) {
                int p = first[start[i] - 1].min;
                int q = first[start[i] - 1].max;
                if (rank[j] >= p && rank[j] <= q) {
                    int t;
                    for (t = rank[j]; t <= q && sa[t] >= start[i] &&
sa[t]
                            < start[i] + len[i]; t++)
                        ;
                    if (res < getLCA(rank[j], t))
                        res = getLCA(rank[j], t);
                    for (t = rank[j]; t >= p && sa[t] >= start[i] &&
sa[t]
                            < start[i] + len[i]; t--)
                        ;
                    if (res < getLCA(rank[j], t))
                        res = getLCA(rank[j], t);
                } else {
                    if (rank[j] < p) {
                        if (res < getLCA(rank[j], p))
                            res = getLCA(rank[j], p);
                    }
                    if (rank[j] > q) {
                        if (res < getLCA(rank[j], q))
                            res = getLCA(rank[j], q);
                    }
                }
            }
            int p = last[start[i] + len[i]].min;
            int q = last[start[i] + len[i]].max;
            if (rank[j] >= p && rank[j] <= q) {
                int t;
                for (t = rank[j]; t <= q && sa[t] >= start[i] && sa[t]
                        < start[i] + len[i]; t++)
                    ;
                if (res < getLCA(rank[j], t))
                    res = getLCA(rank[j], t);
                for (t = rank[j]; t >= p && sa[t] >= start[i] && sa[t]
                        < start[i] + len[i]; t--)
                    ;
                if (res < getLCA(rank[j], t))
```

```
                res = getLCA(rank[j], t);
            } else {
                if (rank[j] < p) {
                    if (res < getLCA(rank[j], p))
                        res = getLCA(rank[j], p);
                }
                if (rank[j] > q) {
                    if (res < getLCA(rank[j], q))
                        res = getLCA(rank[j], q);
                }
            }
            if (alen > res + 1 && res < start[i] + len[i] - j) {
                alen = res + 1;
                astart = j;
            }
        }
        for (int j = astart; j < astart + alen; j++)
            printf("%c", (char) a[j]);
        printf("\n");
    }
}
```

AC自动机可重合,不可重合匹配

```
struct TreeNode {
    TreeNode* next[26];
    TreeNode *jump;
    bool accept;
    int id;
};
struct Query {
    char content[8];
    bool overlap;
    int result;
    int last;
    int len;
};
Query a[100010];
TreeNode nodes[100010], *root = nodes;
TreeNode* queue[600010];
char source[100010];
int parent[100010];
int N, C;
void build(bool overlap) {
    C = 1;
    memset(nodes, 0, sizeof(nodes));
    for (int i = 1; i <= N; i++) {
        if (a[i].overlap != overlap)
            continue;
        TreeNode *current = root;
        for (int j = 0; a[i].content[j]; j++) {
            int k = a[i].content[j] - 'a';
            if (!current->next[k]) {
                nodes[C].jump = root;
                current->next[k] = &nodes[C++];
            }
            current = current->next[k];
            if (!a[i].content[j + 1]) {
                current->accept = true;
                if (!current->id)
                    current->id = i;
                parent[i] = current->id;
            }
```

```
        }
    }
    int head = 0, rear = 0;
    queue[rear++] = root;
    while (head != rear) {
        TreeNode *current = queue[head++];
        for (int i = 0; i < 26; i++) {
            if (!current->next[i])
                continue;
            if (current != root) {
                int mark = 0;
                for (TreeNode *t = current->jump; t != root; t = t-
>jump) {
                    if (t->next[i]) {
                        current->next[i]->jump = t->next[i];
                        if (t->next[i]->accept)
                            current->next[i]->accept = true;
                        mark = 1;
                        break;
                    }
                }
                if (!mark && root->next[i]) {
                    current->next[i]->jump = root->next[i];
                    if (root->next[i]->accept)
                        current->next[i]->accept = true;
                }
            }
            queue[rear++] = current->next[i];
        }
    }
}
void match(bool overlap) {
    TreeNode *current = root;
    for (int i = 0; source[i]; i++) {
        int j = source[i] - 'a';
        int flag = 0;
        for (TreeNode *t = current; t; t = t->jump) {
            if (t->next[j]) {
                if (!flag) {
                    flag = 1;
                    current = t->next[j];
                }
                if (t->next[j]->accept) {
                    int index = t->next[j]->id;
                    if (!overlap) {
                        if (i + 1 - a[index].last >= a[index].len) {
                            a[index].result++;
                            a[index].last = i + 1;
                        }
                    } else {
                        a[index].result++;
                    }
                }
            }
        }
        if (!flag)
            current = root;
    }
}
void init() {
    scanf("%d", &N);
    memset(a, 0, sizeof(a));
    for (int i = 1; i <= N; i++) {
```

```
        int p;
        scanf("%d%s", &p, a[i].content);
        if (!p)
            a[i].overlap = true;
        a[i].len = strlen(a[i].content);
    }
}
void work(int num) {
    build(false);
    match(false);
    build(true);
    match(true);
    printf("Case %d\n", num);
    for (int i = 1; i <= N; i++) {
        printf("%d\n", a[parent[i]].result);
    }
    printf("\n");
}
```

AC自动机状压DP

```
struct TreeNode {
    TreeNode* next[26];
    TreeNode *jump;
    bool accept;
    int id, index;
};
const int Mod = 20090717;
TreeNode *root = NULL;
TreeNode* nodes[200];
TreeNode* queue[200];
int N, M, K, C;
char temp[100];
int matchState[200][26];
TreeNode* match[200][26];
int ones(int x) {
    x = (x & 0x55555555) + ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x & 0x0F0F0F0F) + ((x >> 4) & 0x0F0F0F0F);
    x = (x & 0x00FF00FF) + ((x >> 8) & 0x00FF00FF);
    x = (x & 0x0000FFFF) + ((x >> 16) & 0x0000FFFF);
    return x;
}
int one[1 << 11];
void init() {
    C = 0;
    root = new TreeNode;
    root->id = C;
    nodes[C++] = root;
    root->accept = false;
    root->jump = NULL;
    root->index = 0;
    memset(root->next, NULL, sizeof(root->next));
    for (int i = 0; i < M; i++) {
        scanf("%s", temp);
        TreeNode *current = root;
        for (int j = 0; temp[j]; j++) {
            int k = temp[j] - 'a';
            if (!current->next[k]) {
                TreeNode *t = new TreeNode;
                t->jump = root;
                t->accept = false;
                memset(t->next, NULL, sizeof(t->next));
```

```
                t->id = C;
                t->index = 0;
                nodes[C++] = t;
                current->next[k] = t;
            }
            current = current->next[k];
            if (!temp[j + 1]) {
                current->accept = true;
                current->index |= 1 << i;
            }
        }
    }
    int head = 0, rear = 0;
    queue[rear++] = root;
    while (head != rear) {
        TreeNode *current = queue[head++];
        for (int i = 0; i < 26; i++) {
            if (!current->next[i])
                continue;
            if (current != root) {
                int mark = 0;
                for (TreeNode *t = current->jump; t != root; t = t-
>jump) {
                    if (t->next[i]) {
                        current->next[i]->jump = t->next[i];
                        if (t->next[i]->accept)
                            current->next[i]->accept = true;
                        mark = 1;
                        break;
                    }
                }
                if (!mark && root->next[i]) {
                    current->next[i]->jump = root->next[i];
                    if (root->next[i]->accept)
                        current->next[i]->accept = true;
                }
            }
            queue[rear++] = current->next[i];
        }
    }
    memset(matchState, 0, sizeof(matchState));
    memset(match, NULL, sizeof(match));
    for (int i = 0; i < C; i++) {
        for (int j = 0; j < 26; j++) {
            for (TreeNode *current = nodes[i]; current; current =
current->jump) {
                if (current->next[j]) {
                    if (!match[i][j])
                        match[i][j] = current->next[j];
                    if (current->next[j]->accept) {
                        matchState[i][j] |= current->next[j]->index;
                    }
                }
            }
        }
    }
}
int dp[(1 << 10) + 1][110][26];
void DP() {
    memset(dp, 0, sizeof(dp));
    dp[0][0][0] = 1;
    for (int i = 0; i <= N - 1; i++) {
        for (int j = 0; j < C; j++) {
```

```
            for (int k = 0; k < 1 << M; k++) {
                if (!dp[k][j][i])
                    continue;
                if (one[k] >= K)
                    continue;
                for (int l = 0; l < 26; l++) {
                    int index = (match[j][l] == NULL) ? 0 : match[j]
[l]->id;
                    dp[k | matchState[j][l]][index][i + 1] += dp[k][j]
[i];
                    if (dp[k | matchState[j][l]][index][i + 1] >= Mod)
                        dp[k | matchState[j][l]][index][i + 1] %= Mod;
                }
            }
        }
    }
    int result = 0;
    for (int i = 0; i < 1 << M; i++) {
        if (one[i] < K) {
            for (int j = 0; j < C; j++) {
                result += dp[i][j][N];
                result %= Mod;
            }
        }
    }
    int p = 1;
    for (int i = 1; i <= N; i++) {
        p *= 26;
        p %= Mod;
    }
    result = p - result;
    result %= Mod;
    result += Mod;
    result %= Mod;
    printf("%d\n", result);
}
```

AC自动机矩阵递推

```
int hash[128];
const int Mod = 100000;
struct TreeNode {
    TreeNode* next[4];
    TreeNode *jump;
    bool accept;
    int id;
};
TreeNode *root = NULL;
TreeNode nodes[10001];
int M, N, C = 0;
char temp[12];
TreeNode* queue[10010];
void init() {
    memset(nodes, 0, sizeof(nodes));
    hash['A'] = 0;
    hash['G'] = 1;
    hash['C'] = 2;
    hash['T'] = 3;
    root = &nodes[C];
    root->id = C++;
    scanf("%d%d", &M, &N);
    for (int i = 1; i <= M; i++) {
        scanf("%s", temp);
```

```
        TreeNode *current = root;
        for (int j = 0; temp[j]; j++) {
            int k = hash[temp[j]];
            if (!current->next[k]) {
                TreeNode *t = &nodes[C];
                t->jump = root;
                t->id = C++;
                current->next[k] = t;
            }
            current = current->next[k];
            if (!temp[j + 1])
                current->accept = true;
        }
    }
    int head = 0, rear = 0;
    queue[rear++] = root;
    while (head != rear) {
        TreeNode *current = queue[head++];
        for (int i = 0; i < 4; i++) {
            if (!current->next[i])
                continue;
            if (current != root) {
                int mark = 0;
                TreeNode *t = current->jump;
                while (t != root) {
                    if (t->next[i]) {
                        current->next[i]->jump = t->next[i];
                        if (t->next[i]->accept)
                            current->next[i]->accept = true;
                        mark = 1;
                        break;
                    }
                    t = t->jump;
                }
                if (!mark && root->next[i]) {
                    current->next[i]->jump = root->next[i];
                    if (root->next[i]->accept)
                        current->next[i]->accept = true;
                }
            }
            queue[rear++] = current->next[i];
        }
    }
}
long long numMat[100][100];
void matMult(long long res[][100], long long a[][100], long long b[]
[100]) {
    for (int i = 0; i < C; i++) {
        for (int j = 0; j < C; j++) {
            res[i][j] = 0;
            for (int k = 0; k < C; k++) {
                res[i][j] += a[i][k] * b[k][j];
            }
            res[i][j] %= Mod;
        }
    }
}
void fastMult(long long res[100][100], long long a[100][100],
unsigned int n) {
    long long temp[100][100];
    memset(temp, 0, sizeof(temp));
    for (int i = 0; i < 100; i++) {
        temp[i][i] = 1;
```

```
    }
    int flag = 0;
    for (int i = 31; i >= 0; i--) {
        if (flag) {
            matMult(res, temp, temp);
            memcpy(temp, res, sizeof(numMat));
        }
        if (n & (1 << i)) {
            matMult(res, temp, a);
            memcpy(temp, res, sizeof(numMat));
            flag = 1;
        }
    }
}
void work() {
    for (int i = 0; i < C; i++) {
        for (int j = 0; j < 4; j++) {
            int flag = 0;
            for (TreeNode *current = nodes + i; current; current
                    = current->jump) {
                if (current->accept)
                    break;
                if (current->next[j]) {
                    flag = 1;
                    if (!current->next[j]->accept) {
                        numMat[current->next[j]->id][i]++;
                    }
                    break;
                }
            }
            if (!flag) {
                numMat[0][i]++;
            }
        }
    }
    long long temp[100][100];
    fastMult(temp, numMat, N);
    long long result = 0;
    for (int i = 0; i < C; i++) {
        result += temp[i][0];
    }
    result %= Mod;
    result += Mod;
    result %= Mod;
    printf("%lld\n", result);
}
```

KMP自动机DP

```
int next[10010];
char a[10010];
int N, M, Len;
char source[10010];
void init() {
    memset(next, 0, sizeof(next));
    memset(a, 0, sizeof(a));
    scanf("%s", source + 1);
    scanf("%s", a + 1);
    N = strlen(a + 1);
    M = strlen(source + 1);
    next[1] = 0;
    int k = 0;
    for (int i = 2; i <= N; i++) {
```

```
        while (k > 0 && a[k + 1] != a[i])
            k = next[k];
        if (a[k + 1] == a[i])
            k++;
        next[i] = k;
    }
}
int dp[2][10010];
int trans[1010][30];
void build() {
    for (int i = 0; i < N; i++) {
        for (char k = 'a'; k <= 'z'; k++) {
            int q = i;
            while (q > 0 && a[q + 1] != k)
                q = next[q];
            if (q + 1 == N) {
                trans[i][k - 'a'] = -1;
                continue;
            }
            if (a[q + 1] == k)
                trans[i][k - 'a'] = q + 1;
            else
                trans[i][k - 'a'] = 0;
        }
    }
}
void work() {
    build();
    memset(dp, 0, sizeof(dp));
    //dp[0][0]=0;
    for (int i = 0; i < M; i++) {
        int current = i & 1;
        int nex = (i + 1) & 1;
        memcpy(dp[nex], dp[current], sizeof(dp[0]));
        for (int k = 0; k < N; k++) {
            int t = trans[k][source[i + 1] - 'a'];
            if (t == -1)
                continue;
            dp[nex][t] = max(dp[nex][t], dp[current][k] + 1);
        }
    }
    int res = 0;
    for (int i = 0; i <= N - 1; i++) {
        res = max(res, dp[M & 1][i]);
    }
    printf("%d\n", M - res);
}
```

扩展KMP

```
int A[maxn], B[maxn];
void build(char *T, char *S) {
    int lenT = strlen(T);
    int lenS = strlen(S);
    int j = 0;
    while (j + 1 < lenT && T[j] == T[j + 1])
        j++;
    A[1] = j;
    int k = 1;
    for (int i = 2; i < lenT; i++) {
        int Len = k + A[k] - 1, L = A[i - k];
        if (L < Len - i + 1)
            A[i] = L;
```

```
        else {
            j = max(0, Len - i + 1);
            while (i + j < lenT && T[i + j] == T[j])
                j++;
            A[i] = j;
            k = i;
        }
    }
    j = 0;
    while (j < lenS && j < lenT && T[j] == S[j])
        j++;
    B[0] = j;
    k = 0;
    for (int i = 1; i < lenS; i++) {
        int Len = k + B[k] - 1, L = A[i - k];
        if (L < Len - i + 1)
            B[i] = L;
        else {
            j = max(0, Len - i + 1);
            while (i + j < lenS && j < lenT && S[i + j] == T[j])
                j++;
            B[i] = j;
            k = i;
        }
    }
}
```