SPLAY:

```
struct NODE
{
    int lsum,rsum,rev,maxsum,sum,key,size,same;
    int c[2],father;
}tree[maxn];

void Initialize()
{
    for (int i = (int)5e5;i;i --) pool[++pool[0]] = i;
    scanf("%d%d",&N,&M);
    fo(i,1,N) scanf("%d",&A[i]);
}

void PushUp(int x)
{
    int l = tree[x].c[0], r = tree[x].c[1];
    if (l>0 && r>0)
    {
        int cl = max(0,tree[l].rsum), cr = max(0,tree[r].lsum);
        tree[x].size = tree[l].size + tree[r].size + 1;
        tree[x].lsum = max(tree[l].lsum,tree[l].sum+tree[x].key+cr);
        tree[x].rsum = max(tree[r].rsum,tree[r].sum+tree[x].key+cl);
        tree[x].maxsum                                                    =
max(max(tree[l].maxsum,tree[r].maxsum),cl+tree[x].key+cr);
        tree[x].sum = tree[l].sum + tree[r].sum + tree[x].key;
    } else
    if (l>0)
    {
        int cl = max(0,tree[l].rsum);
        tree[x].size = tree[l].size + 1;
        tree[x].lsum = max(tree[l].lsum,tree[l].sum+tree[x].key);
        tree[x].rsum = tree[x].key + cl;
```

```
            tree[x].maxsum = max(tree[l].maxsum,tree[x].key+cl);
            tree[x].sum = tree[l].sum + tree[x].key;
        } else
        if (r>0)
        {
            int cr = max(0,tree[r].lsum);
            tree[x].size = tree[r].size + 1;
            tree[x].rsum = max(tree[r].rsum,tree[r].sum+tree[x].key);
            tree[x].lsum = tree[x].key + cr;
            tree[x].maxsum = max(tree[r].maxsum,tree[x].key+cr);
            tree[x].sum = tree[r].sum + tree[x].key;
        } else
        {
            tree[x].size = 1;
            tree[x].lsum = tree[x].rsum = tree[x].maxsum = tree[x].sum = tree[x].key;
        }
}

void PutRev(int x)
{
    if (x<0) return;
    tree[x].rev ^= 1;
    swap(tree[x].lsum,tree[x].rsum);
}

void PutSame(int x,int v)
{
    if (x<0) return;
    tree[x].key = tree[x].same = v;
    tree[x].sum = v * tree[x].size;
    tree[x].lsum = tree[x].rsum = tree[x].maxsum =
max(tree[x].key,tree[x].sum);
}
```

```
void PushDown(int x)
{
    if (tree[x].rev)
    {
        PutRev(tree[x].c[0]);
        PutRev(tree[x].c[1]);
        swap(tree[x].c[0],tree[x].c[1]);
        tree[x].rev = 0;
    }
    if (tree[x].same != maxn)
    {
        PutSame(tree[x].c[0],tree[x].same);
        PutSame(tree[x].c[1],tree[x].same);
        tree[x].same = maxn;
    }
}

int MakeSplay(int l,int r,int father)
{
    int mid = (l + r) >> 1;
    int pos = pool[pool[0]--];
    tree[pos].size = 1;
    tree[pos].c[0] = tree[pos].c[1] = -maxn;
    tree[pos].same = maxn;
    tree[pos].rev = 0;
    tree[pos].key = A[mid];
    tree[pos].father = father;
    if (mid == 0) pos0 = pos;
    if (mid == N+1) posN1 = pos;
    if (l < mid) tree[pos].c[0] = MakeSplay(l,mid-1,pos);
    if (mid < r) tree[pos].c[1] = MakeSplay(mid+1,r,pos);
    PushUp(pos);
```

```
    return pos;
}

void rotate(int x,int d)
{
    int y = tree[x].c[d];
    int z = tree[y].c[!d];
    if (tree[x].father > 0)
    {
        int t = tree[x].father;
        int r = (tree[t].c[1] == x);
        tree[t].c[r] = y;
    }
    if (z > 0) tree[z].father = x;
    tree[x].c[d] = z;
    tree[y].father = tree[x].father;
    tree[x].father = y; tree[y].c[!d] = x;
    PushUp(x);
    PushUp(y);
}

int tank[maxn],dir[maxn];
void Splay(int x,int t)
{
    int cnt = 0;
    for (int i = x;i != tree[t].father;i = tree[i].father) tank[++cnt] = i;
    for (int i = cnt;i;i --) PushDown(tank[i]);
    fo(i,2,cnt) dir[i] = (tree[tank[i]].c[1] == tank[i-1]);
    for (int i = 2;i <= cnt;i += 2)
        if (i == cnt) rotate(tank[i],dir[i]);
        else
        {
            if (dir[i] == dir[i+1]) rotate(tank[i+1],dir[i+1]), rotate(tank[i],dir[i]);
```

```
                else rotate(tank[i],dir[i]), rotate(tank[i+1],dir[i+1]);
            }
        if (t == root) root = x;
}

int Select(int x,int v)
{
        int cur;
        PushDown(x);
        if (tree[x].c[0] > 0) cur = tree[tree[x].c[0]].size + 1; else cur = 1;
        if (cur == v) return x;
        if (v <= cur) return Select(tree[x].c[0],v);
        else return Select(tree[x].c[1],v-cur);
}

void INSERT()
{
        int posi,ctot;
        scanf("%d%d",&posi,&ctot);
        int pos = Select(root,posi+1), posnext = Select(root,posi+2);
        Splay(posnext,root);
        Splay(pos,tree[posnext].c[0]);
        fo(i,1,ctot) scanf("%d",&A[i]);
        tree[tree[root].c[0]].c[1] = MakeSplay(1,ctot,tree[root].c[0]);
        PushUp(tree[root].c[0]),PushUp(root);
}

void Dispose(int x)
{
        if (x<0) return;
        Dispose(tree[x].c[0]);
        Dispose(tree[x].c[1]);
        memset(&tree[x],0,sizeof tree[x]);
```

```
    pool[++pool[0]] = x;
}

int l,r,c,prel,sucr;
inline void Set(int l,int r)
{
    r = l + r - 1;
    int prel = Select(root,l), sucr = Select(root,r+2);
    Splay(sucr,root);
    Splay(prel,tree[sucr].c[0]);
}

void DELETE()
{
    scanf("%d%d",&l,&r);
    Set(l,r);
    int x = tree[root].c[0];
    Dispose(tree[x].c[1]);
    tree[x].c[1] = -maxn;
    PushUp(x),PushUp(root);
}

void MAKESAME()
{
    scanf("%d%d%d",&l,&r,&c);
    Set(l,r);
    int x = tree[root].c[0];
    PutSame(tree[x].c[1],c);
    PushUp(x),PushUp(root);
}

void REVERSE()
{
```

```
    scanf("%d%d",&l,&r);
    Set(l,r);
    int x = tree[root].c[0];
    PutRev(tree[x].c[1]);
    PushUp(x),PushUp(root);
}

void GETSUM(int cs)
{
    scanf("%d%d",&l,&r);
    if (!r)
    {
        printf("0\n");
        return;
    }
    Set(l,r);
    int x = tree[root].c[0];
    printf("%d\n",tree[tree[x].c[1]].sum);
}

void MAXSUM()
{
    Splay(posN1,root);
    Splay(pos0,tree[root].c[0]);
    printf("%d\n",tree[tree[tree[root].c[0]].c[1]].maxsum);
}

Treap_durable:
#define S(x) ((x)?((x)->size):0)
#define RAND(x) (double(rand())/RAND_MAX <= (x))

struct Node
{
```

```
    int size; Node *l,*r;
    char c; bool rev;
    Node(){}
    Node(char _c){c = _c,size = 1,l = r = 0,rev = 0;}
    Node(int _size,Node *_l,Node *_r,char _c,bool _rev){size = _size,c = _c,l = _l,r =
_r,rev = _rev;}
}pool[maxn];
Node* root;

inline void PutRev(Node *x)
{
    swap(x->l,x->r);
    x->rev = 0;
    if (x->l)
    {
        pool[++tail] = Node(x->l->size,x->l->l,x->l->r,x->l->c,(x->l->rev)^1);
        x->l = &pool[tail];
    }
    if (x->r)
    {
        pool[++tail] = Node(x->r->size,x->r->l,x->r->r,x->r->c,(x->r->rev)^1);
        x->r = &pool[tail];
    }
}

inline void PushUp(Node *x)
{
    x->size = S(x->l) + S(x->r) + 1;
}

void Split(Node *x,int p,Node *&L,Node *&R)
{
    if (!x)
```

```
    {
        L = R = 0;
        return;
    }
    if (x->rev) PutRev(x);
    pool[++tail] = Node(x->c);
    Node *cur = &pool[tail];
    if (S(x->l) >= p)
    {
        Split(x->l,p,L,R);
        cur->l = R, cur->r = x->r;
        PushUp(cur);
        R = cur;
    } else
    {
        Split(x->r,p-S(x->l)-1,L,R);
        cur->l = x->l, cur->r = L;
        PushUp(cur);
        L = cur;
    }
}

Node *Merge(Node *L,Node *R)
{
    if (!L) return R;
    if (!R) return L;
    if (L->rev) PutRev(L);
    if (R->rev) PutRev(R);
    Node *cur;
    if (RAND((double)L->size/(L->size+R->size)))
    {
        pool[++tail] = Node(L->c);
        cur = &pool[tail];
```

```
            cur->l = L->l, cur->r = Merge(L->r,R);
            PushUp(cur);
        } else
        {
            pool[++tail] = Node(R->c);
            cur = &pool[tail];
            cur->l = Merge(L,R->l), cur->r = R->r;
            PushUp(cur);
        }
        return cur;
}

char Select(Node *x,int k)
{
        if (x->rev) PutRev(x);
        int size = S(x->l);
        if (k == size+1) return x->c;
        return k<=size ? Select(x->l,k):Select(x->r,k-size-1);
}

Node *A,*B,*C,*D;
void Insert()
{
        int x;char c;
        scanf("%d %c",&x,&c);
        Split(root,x,A,B);
        pool[++tail] = Node(c);
        root = Merge(A,Merge(&pool[tail],B));
}

void Delete()
{
        int l,r;
```

```
    scanf("%d%d",&l,&r);
    Split(root,r,A,B);
    Split(A,l-1,C,D);
    root = Merge(C,B);
}

void Copy()
{
    int l,r,x;
    scanf("%d%d%d",&l,&r,&x);
    Split(root,r,D,C);
    Split(D,l-1,A,B);
    Split(root,x,A,C);
    root = Merge(A,Merge(B,C));
}

void Reverse()
{
    int l,r;
    scanf("%d%d",&l,&r);
    Split(root,r,D,C);
    Split(D,l-1,A,B);
    if (B) B->rev ^= 1;
    root = Merge(A,Merge(B,C));
}

void Query()
{
    int k;
    scanf("%d",&k);
    printf("%c",Select(root,k));
}
```

SA:

```
#define fo(i,a,b) for (int i = a;i <= b;i++)
#define fd(i,a,b) for (int i = a;i >= b;i--)

int main(void)
{
	cin >> (s+1);
	n = strlen(s+1);
	memset(tank,0,sizeof tank);
	fo(i,1,n) tank[s[i]] ++;
	fo(i,1,255) tank[i] += tank[i-1];
	fo(i,1,n) sa[tank[s[i]]--] = i;
	int j = 0;
	fo(i,1,n)
	{
		if (i == 1 || s[sa[i]] != s[sa[i-1]]) j ++;
		rank[sa[i]] = j;
	}
	for (int k = 1;k <= n;k <<= 1)
	{
		j = 0;
		fo(i,n+1,n+k) tmp[++j] = i - k;
		fo(i,1,n) if (sa[i] > k) tmp[++j] = sa[i] - k;
		memset(tank,0,sizeof tank);
		fo(i,1,n) tank[rank[tmp[i]]] ++;
		fo(i,1,n) tank[i] += tank[i-1];
		fd(i,n,1) sa[tank[rank[tmp[i]]]--] = tmp[i];
		copy(rank+1,rank+1+n,rk+1);
		j = 0;
		fo(i,1,n)
		{
			if ((i == 1) || (rk[sa[i]] != rk[sa[i-1]]) || (rk[sa[i]] == rk[sa[i-1]] &&
```

```
rk[sa[i]+k] != rk[sa[i-1]+k]))
                        j ++;
                rank[sa[i]] = j;
            }
        }
        fo(i,1,n)
        {
            height[rank[i]] = max(0,height[rank[i-1]]-1);
            while (s[i+height[rank[i]]] == s[sa[rank[i]-1]+height[rank[i]]])
height[rank[i]] ++;
        }
        int ans = 0;
        fo(i,1,n)
            if (height[i] > height[i-1]) ans += height[i] - height[i-1];
        cout<<ans<<endl;

        return 0;
}
```

Manacher:

```
void Manacher()
{
    int id = 0;
    fo(i,1,N)
    {
        if (id+r[id] > i) r[i] = min(id+r[id]-i,r[2*id-i]);
        else r[i] = 0;
        while (i+r[i]+1 <= N && i-r[i]-1 >= 1 && s[i+r[i]+1] == s[i-r[i]-1]) r[i] ++;
        if (i+r[i] > id+r[id]) id = i;
    }
}
```

SBT:

```
int N,T,tot;
int left[maxn],right[maxn],size[maxn],key[maxn];

inline void L_rotate(int &t)
{
    int k = right[t];
    right[t] = left[k];
    left[k] = t;
    size[k] = size[t];
    size[t] = size[left[t]] + size[right[t]] + 1;
    t = k;
}

inline void R_rotate(int &t)
{
    int k = left[t];
    left[t] = right[k];
    right[k] = t;
    size[k] = size[t];
    size[t] = size[left[t]] + size[right[t]] + 1;
    t = k;
}

inline void maintain(int &t,bool flag)
{
    if (!flag)
    {
        if (size[left[left[t]]] > size[right[t]])
            R_rotate(t);
        else if (size[right[left[t]]] > size[right[t]])
            L_rotate(left[t]), R_rotate(t);
        else return;
    } else
```

```
    {
        if (size[right[right[t]]] > size[left[t]])
            L_rotate(t);
        else if (size[left[right[t]]] > size[left[t]])
            R_rotate(right[t]), L_rotate(t);
        else return;
    }
    maintain(left[t],0);
    maintain(right[t],1);
    maintain(t,0);
    maintain(t,1);
}

inline void Insert(int &t,int v)
{
    if (!t)
    {
        t = ++ tot;
        size[t] = 1;
        left[t] = right[t] = 0;
        key[t] = v;
    } else
    {
        size[t] ++;
        if (v < key[t]) Insert(left[t],v);
        else Insert(right[t],v);
        maintain(t,v >= key[t]);
    }
}

inline int Delete(int &t,int v)
{
    size[t] --;
```

```
    if ((key[t] == v) || (v < key[t] && !left[t]) || (v >= key[t] && !right[t]))
    {
        int ret = key[t];
        if (!left[t] || !right[t]) t = left[t] + right[t];
        else key[t] = Delete(left[t],key[t]+1);
        return ret;
    }
    if (v < key[t]) return Delete(left[t],v);
    else return Delete(right[t],v);
}

inline int Rank(int t,int v)
{
    if (!t) return 0;
    if (v <= key[t]) return Rank(left[t],v);
    else return size[left[t]] + 1 + Rank(right[t],v);
}

inline int Select(int t,int v)
{
    int temp = size[left[t]] + 1;
    if (temp == v) return key[t];
    if (v <= temp) return Select(left[t],v);
    else return Select(right[t],v-temp);
}

inline int Pred(int t,int v)
{
    if (!t) return INF;
    if (v <= key[t]) return Pred(left[t],v);
    else
    {
        int temp = Pred(right[t],v);
```

```
        if (temp == INF) return key[t];
        else return temp;
    }
}

inline int Succ(int t,int v)
{
    if (!t) return INF;
    if (v >= key[t]) return Succ(right[t],v);
    else
    {
        int temp = Succ(left[t],v);
        if (temp == INF) return key[t];
        else return temp;
    }
}
```

Dinic:

```
inline bool bfs()
{
    static int list[maxn*maxn];
    int st,en;
    list[st=en=1] = source;
    memset(lab,255,sizeof lab);
    lab[source] = 0;
    while (st <= en)
    {
        int x = list[st];
        for(int i = a[x];i;i = c[i])
            if (d[i] && lab[b[i]] == -1)
            {
                lab[b[i]] = lab[x] + 1;
                list[++en] = b[i];
```

```
            }
        st ++;
    }
    return lab[sink] > -1;
}

inline int dfs(int x,int flow)
{
    if (x == sink) return flow;
    int ret = 0;
    for (int i = a[x];i;i = c[i])
        if (d[i] && lab[b[i]] == lab[x] + 1)
        {
            int now = dfs(b[i],min(flow,d[i]));
            d[i] -= now;
            d[i^1] += now;
            flow -= now;
            ret += now;
            if (!flow) return ret;
        }
    lab[x] = -1;
    return ret;
}
```

点分治：

```
void Get_siz(int u,int fa = 0)
{
    siz[u] = 1;
    for (Edge *e = H[u];e;e = e->next)
    {
        int v = e->v;
        if (v != fa && vis[v] != Time)
        {
```

```
            Get_siz(v,u);
            siz[u] += siz[v];
        }
    }
}

void Get_root(int u,int fa = 0)
{
    num[u] = 0;
    for (Edge *e = H[u];e;e = e->next)
    {
        int v = e->v;
        if (v != fa && vis[v] != Time)
        {
            Get_root(v,u);
            num[u] = max(num[u],siz[v]);
        }
    }
    num[u] = max(num[u],size-siz[u]);
    if (num[u] < num[root]) root = u;
}

void Get_dis(int u,int val,int fa = 0)
{
    dis[u] = val;
    S[++cnt] = node[u] - val;
    for (Edge *e = H[u];e;e = e->next)
    {
        int v = e->v;
        if (v != fa && vis[v] != Time) Get_dis(v,dis[u]+e->c,u);
    }
}
```

```
int Search(const Node &x)
{
	int l = 0, r = cnt;
	while (l < r)
	{
		int m = (l + r + 1) >> 1;
		if (S[m] <= x) l = m;
		else r = m - 1;
	}
	return r;
}

void Get_ans(int u,int sign,int fa = 0)
{
	if (!mart[u])
	{
		int tmp = Search(Node(dis[u],u));
		ans[u] += (cnt-tmp) * sign;
	}
	for (Edge *e = H[u];e;e = e->next)
	{
		int v = e->v;
		if (v != fa && vis[v] != Time) Get_ans(v,sign,u);
	}
}

void Deal(int u,int val,int sign)
{
	cnt = 0;
	Get_dis(u,val);
	sort(S+1,S+cnt+1);
	Get_ans(u,sign);
}
```

```
void Divide(int u)
{
    Get_siz(u);
    size = siz[u];
    root = 0;
    Get_root(u);
    vis[root] = Time;
    Deal(root,0,1);
    for (Edge *e = H[root];e;e = e->next) if (vis[e->v] != Time) Deal(e->v,e->c,-1);
    for (Edge *e = H[root];e;e = e->next) if (vis[e->v] != Time) Divide(e->v);
}
```

cdq 分治：

```
/*
```

数据　　by 80 中集训

【题目大意】

为了写论文，Alex 经常要整理大量的数据。这一次，Alex 面临一个严峻的考验：他需要实现一个数据结构来维护一个点集。

现在，二维平面上有 N 个点。Alex 需要实现以下三种操作：

1. 在点集里添加一个点；

2. 给出一个点，查询它到点集里所有点的曼哈顿距离的最小值；

3. 给出一个点，查询它到点集里所有点的曼哈顿距离的最大值。

两个点的曼哈顿距离定义为它们的横坐标差的绝对值与纵坐标差的绝对值的和。这么困难的问题，Alex 当然不会做，只好再次请你帮忙了。

【题解】

这题的最大值是比较好做的，维护已经插入过的点的四种状态的最小值。

即 x+y,x-y,-x+y,-x-y，求答案时用询问的点对应的状态减去出现过的最小值，再

取个 max 即可。
（参见最大曼哈顿距离 http://blog.csdn.net/hedongnike/article/details/20649231）

最小值的维护：
这就用到 CDQ 分治了。
我们把所有操作和询问读进来，原本的点也当作插入操作。
然后我们得到了一个按照操作时间顺序的序列，有插入操作和询问操作。
注意到所有的询问操作只会被它前面的插入操作影响，我们从这里入手。
假设当前要处理的操作序列为[l,r]。
我们先二分出中间点 mid。

先递归进去子问题[l,mid],[mid+1,r]

然后我们处理前半部分插入对后半部分询问的影响。
我们先标记前半部分的询问和后半部分的询问，然后把它们按 x 坐标排序。
然后便按 x 坐标顺序小到大做。
按照离散化后的 y 坐标建立两个树状数组，维护做到当前询问之前，插入的某个状态的最小值。
[设当前询问的点的坐标为(x0,y0)，我们分别考虑之前插入的点的 y 坐标大于 y0 的和小于 y0 的（计算曼哈顿距离的方式不同）]
（此处较难表达，请看代码或画图）
遇到插入的操作则插入相应的状态值到相应的树状数组，遇到询问则更新该询问的答案。

最后还要反过来按 x 坐标从大到小做，同样要用树状数组维护。
状态为（x+y,x-y,-x+y,-x-y），即为询问点的四个方向上计算曼哈顿距离所需状态。
*/
```
void Solve(int l,int r)
{
    if (l == r) return;
    int mid = (l+r) >> 1;
    Solve(l,mid);
```

```
Solve(mid+1,r);
// Prepare
fo(i,l,r) A[i].flag = -1;
fo(i,l,mid) if (A[i].op == 0) A[i].flag = 0;
fo(i,mid+1,r) if (A[i].op == 1) A[i].flag = 1;
sort(A+l,A+r+1,cmpx);
int tot = 0;
fo(i,l,r)
    if (A[i].flag == 0 || A[i].flag == 1)
        B[++tot] = OPER(A[i].y,i);
// Work
int lim = Discretize(1,tot);
fill(up+1,up+lim+1,-INF);
fill(down+1,down+lim+1,-INF);
fo(i,l,r)
    if (A[i].flag == 0)
    {
        Ins_up(ord[i],A[i].x-A[i].y);
        Ins_down(ord[i],A[i].x+A[i].y,lim);
    } else
    if (A[i].flag == 1)
    {
        ans[A[i].id] = min(ans[A[i].id],A[i].x-A[i].y-Query_up(ord[i],lim));
        ans[A[i].id] = min(ans[A[i].id],A[i].x+A[i].y-Query_down(ord[i]));
    }
fill(up+1,up+lim+1,-INF);
fill(down+1,down+lim+1,-INF);
fd(i,r,l)
    if (A[i].flag == 0)
    {
        Ins_up(ord[i],-A[i].x-A[i].y);
        Ins_down(ord[i],-A[i].x+A[i].y,lim);
    } else
```

```
        if (A[i].flag == 1)
        {
            ans[A[i].id] = min(ans[A[i].id],-A[i].x-A[i].y-Query_up(ord[i],lim));
            ans[A[i].id] = min(ans[A[i].id],-A[i].x+A[i].y-Query_down(ord[i]));
        }
}
```

Euler 筛：

```
void EulerSelect()
{
    static int prime[maxn],p[maxn],s[maxn];
    static bool notprime[maxn];
    int tot = 0;
    miu[1] = 1;
    fo(i,2,mx)
    {
        if (!notprime[i]) prime[++tot] = i, s[i] = i, p[i] = i, miu[i] = -1;
        fo(j,1,tot)
        {
            if (i * prime[j] > mx) break;
            notprime[i*prime[j]] = 1;
            s[i*prime[j]] = prime[j];
            if (i % prime[j] == 0)
            {
                p[i*prime[j]] = p[i] * prime[j];
                miu[i*prime[j]] = 0;
                break;
            }
            p[i*prime[j]] = prime[j];
            miu[i*prime[j]] = miu[i] * miu[prime[j]];
        }
    }
    a[1] = PI(f[1]=1,1);
```

```
    fo(i,2,mx)
    {
        if (i == p[i]) f[i] = ((LL)p[i]*s[i]-1)/(s[i]-1);
        else f[i] = f[i/p[i]] * f[p[i]];
        a[i]= PI(f[i],i);
    }
}
```

AC 自动机：

```
struct Node
{
    int son[10],fail;
    bool flag;
}t[maxn];

void Insert(int *s)
{
    int x = 0;
    fo(i,1,s[0])
    {
        if (!t[x].son[s[i]]) t[x].son[s[i]] = ++ tot;
        x = t[x].son[s[i]];
    }
    t[x].flag = 1;
}

void Construct_fail()
{
    static int d[maxn];
    int l = 0,r = 1;
    while (l < r)
    {
        int x = d[++l];
```

```
            fo(i,0,9)
                if (t[x].son[i])
                {
                    int v = t[x].son[i];
                    if (x == 0) t[v].fail = 0;
                    else t[v].fail = t[t[x].fail].son[i];
                    t[v].flag |= t[t[v].fail].flag;
                    d[++r] = v;
                } else t[x].son[i] = t[t[x].fail].son[i];
        }
}

Extended_kmp:
void Calc_next(char *s,int *next)
{
        next[0] = N;
        int pos,prev,j=-1;
        for (int i = 1;i < N;i ++,j --)
            if (j < 0 || i+next[i-prev] >= pos)
            {
                if (j < 0) j = 0, pos = i;
                while (pos < N && s[pos] == s[j]) pos ++,j ++;
                next[i] = j, prev = i;
            } else next[i] = next[i-prev];
}

void Extended_KMP(char *T,char *s,int *next,int *ext)
{
        Calc_next(T,next);
        int pos,prev,j=-1;
        for (int i = 0;i < N;i ++,j --)
            if (j < 0 || i+next[i-prev] >= pos)
            {
```

```
            if (j < 0) j = 0, pos = i;
            while (pos < N && j < N && s[pos] == T[j]) pos ++,j ++;
            ext[i] = j, prev = i;
        } else ext[i] = next[i-prev];
}
```

左偏树：

```
int Merge(int a,int b)
{
    if (!a) return b;
    if (!b) return a;
    if (t[a].key < t[b].key) swap(a,b);
    t[a].r = Merge(t[a].r,b);
    if (t[t[a].l].dis < t[t[a].r].dis) swap(t[a].l,t[a].r);
    t[a].dis = t[t[a].r].dis + 1;
    t[a].size = t[t[a].l].size + t[t[a].r].size + 1;
    t[a].sum = t[t[a].l].sum + t[t[a].r].sum + t[a].key;
    return a;
}

void Work()
{
    static int q[maxn];
    int l = 0, r = 1;
    q[1] = 1;
    while (l < r)
    {
        int x = q[++l];
        for (int i = a[x];i;i = c[i]) q[++r] = b[i];
    }
    LL ans = 0;
    for (;r;r --)
    {
```

```
            int x = q[r];
            t[++n] = Node(C[x]);
            f[x] = n;
            for (int i = a[x];i;i = c[i]) f[x] = Merge(f[x],f[b[i]]);
            while (t[f[x]].sum > M) f[x] = Merge(t[f[x]].l,t[f[x]].r);
            ans = max(ans,(LL)t[f[x]].size*L[x]);
        }
        printf("%lld\n",ans);
}
```

网络流：

先来看这么一个题。

给出一个无向图 G=(V,E)，并有定义在 V 上的权 a 和 b，以及定义在 E 上的权 w。（这里提及的所有权都默认为非负整数。）

选出一个点集 S，最大化∑(x 属于 S)a[x]+∑(x 不属于 S)b[x]+∑(x 和 y 同时属于或不同时属于 S)w(x,y)。

这是一个可以直接用最小割解决的问题。

添加源 S 和汇 T，对于每个 x，连边(S,x,a[x])和(x,T,b[x])，而 w(x,y)直接变为对应边的容量。

考察这个图的割的意义不难得出最终的答案=∑a[x]+∑b[x]+∑w(x,y)-最小割。

好..我们来改一改。

现在我们需要最大的化的式子变成了这样。

∑(x 属于 S)a[x]+∑(x 不属于 S)b[x]+∑(x 属于 S 且 y 不属于 S)w(x,y)。

....很可惜似乎直接做是不可行的，如果有靠谱的做法请不吝赐教。

我们需要一个重要的条件：图 G 是二分图。

对于 X 中的每个点 x，连边(S,x,a[x])和(x,T,b[x])；

对于 Y 中的每个点 y，连边(S,y,b[y])和(y,T,a[y])。

而边权 w(x,y)仍然直接变为对应边的容量。

下面来看另外一种改法。

新增一个定义在 E 上的权 r。

最大化∑(x 属于 S)a[x]+∑(x 不属于 S)b[x]+∑(x 和 y 都属于 S)w(x,y)+∑(x 和 y

都不属于 S)r(x,y)。
（请留意，二分图的条件已经吃粑粑去了。）
回顾前面的解题过程，在考虑使用割的意义建图的时候，我们的割边实际上是“损失”。
在这一题中点权似乎仍然是老样子，但是边权有些麻烦了。
1. 当 x 和 y 同时属于 S 的时候，我们损失了 r(x,y)。
2. 当 x 和 y 同时不属于 S 的时候，我们损失了 w(x,y)。
3. 当 x 属于 S 和 y 属于 S 的真假性不同的时候，我们损失了 w(x,y)以及 r(x,y)。
正确的表达这些损失要用到的技巧是权值的重新分配。

首先，添加源 S 和汇 T，对于每个 x，连边(S,x,a[x])和(x,T,b[x])

考虑第 1 条，可以视为，当 x 属于 S 的时候，损失 r(x,y)/2，y 同理；
此时即要将 x，y 向 T 连各连一条 r(x,y)/2 的边，因为割的边代表损失的利益，而 S 连向 x，y 的边保留代表取此种方案。

第 2 条类似。

而在第 3 条描述的情形下，根据前面已经做出的调整（根据前两条，已经割了 S 向 x 的边，y 向 T 的边（或是 S 到 y，x 到 T）），我们就会计算出(r(x,y)+w(x,y))/2 的损失；
然而真正的损失是 w(x,y)+r(x,y)，因此我们对 x 到 y，y 到 x 分别连上一条容量(r(x,y)+w(x,y))/2 的边来补上这个损失。
这样一来所有的损失都被正确的计算了。

再来看一道有点像的题。
给定无重边无自环的无向图 G=(V,E)，定义在 V 上的权 a，E 上的权 w，求一个点集 S，最大化：
2*Σ(x,y 属于 S)w(x,y)-Σ(x 属于 S 且 y 不属于 S)w(x,y)-Σ(x 属于 S)a[x]。
（为什么要乘 2 呢...Σ枚举的 x,y 是无序的，但是根据题意 w(x,y)和 w(y,x)都要算进去）
令 b[x]=Σ(y)w(x,y)，则Σ(x 属于 S 且 y 不属于 S)w(x,y)=Σ(x 属于 S)b[x]-2*Σ(x,y 属于 S)w(x,y)，

进而要最大化的式子变为 Σ(x,y 属于 S)4*w(x,y) - Σ(x 属于 S)(a[x]+b[x])。
很经典的最大权闭合图，这里略去。
但是这样搞总觉得有点脱离主题，于是我们再来看一个做法。
据Σw(x,y)=Σ(x,y 属于 S)w(x,y)+Σ(x,y 不属于 S)w(x,y)+Σ(x 属于 S 且 y 不属于 S)w(x,y)，将目标式化为
2*Σw(x,y) - 3*Σ(x 属于 S 且 y 不属于 S)w(x,y) - Σ(x 属于 S)a[x]。
从而只需要最小化 3*Σ(x 属于 S 且 y 不属于 S)w(x,y) + 2*Σ(x,y 不属于 S)w(x,y) + Σ(x 属于 S)a[x]。
而这个东西等于 Σ(y 不属于 S)w(x,y) + 2*Σ(x 属于 S 且 y 不属于 S)w(x,y) + Σ(x 属于 S)a[x]。
（恩，2*Σ(x,y 不属于 S)w(x,y)+Σ(x 属于 S 且 y 不属于 S)w(x,y)=Σ(y 不属于 S)w(x,y)看起来挺纠结的。）
不管怎么说弄成这样了以后就可以利用割的意义来构造了：
Σw(x,y)就是把 y 割出 S 的损失，a[x]就是把 x 割进 S 的损失，将 x,y 分开的损失则是 2*w(x,y)。
于是直接最小割就完了。