字符串

Trie

Trie注意开足数组大小!

- $ch[i][j]$ 表示Trie中结点 i 的孩子 j 的编号
- $val[i]>0$ 表示结点 i 为一个单词的结尾结点

```
struct Trie
{
    int ch[400010][26],val[400010];
    int sz;
    void init()
    {
        sz=1;
        memset(ch[0],0,sizeof(ch[0]));
    }
    int idx(char c){return c-'a';}
    void insert(char *s,int v)
    {
        int u=0,n=strlen(s);
        for (int i=0;i<n;i++)
        {
            int c=idx(s[i]);
            if (!ch[u][c])
            {
                memset(ch[sz],0,sizeof(ch[sz]));
                val[sz]=0;
                ch[u][c]=sz++;
            }
            u=ch[u][c];
        }
        val[u]=v;
    }
}tree;
```

可持久化Trie

- insert(s,n,x) 插入的字符串为$s_0,\cdots,s_{n-1}$，长度为n，当前是x位（其中x和u是树上对应的位置）
- $val[x]$表示当前节点上经过的单词个数

```
struct Tries
{
    int ch[20000010][2],val[20000010];
    int sz,h[20000010];
    void init()
    {
        sz=1;
        memset(h,-1,sizeof(h));h[0]=0;
        memset(val,0,sizeof(val));
        memset(ch[0],0,sizeof(ch[0]));
    }
    int idx(char c){return c-'0';}
    void insert(char *s,int n,int x)//No.x trie   u/x
    {
        int u=h[x-1];h[x]=sz++;x=h[x];
        for (int i=0;i<n;i++)
        {
            int c=idx(s[i]);
            for (int j=0;j<=1;j++)
                ch[x][j]=ch[u][j];
            memset(ch[sz],0,sizeof(ch[sz]));
            ch[x][c]=sz++;
            u=ch[u][c];
            x=ch[x][c];
            val[x]=val[u]+1;
        }
    }
}tree;
```

KMP

主串s，长度为n

模式串t，长度为m

```
struct kmp
{
    int s[1000010],t[1000010],Next[1000010];
    void Pre_KMP()
    {
        for (int i=0;i<=m;i++)
            Next[i]=0;
        int j=0,k=-1;
        Next[0]=-1;
        while(j<m)
        {
            if (k==-1||t[j]==t[k]) Next[++j]=++k;
            else k=Next[k];
        }
    }
    int KMP()
    {
        int i=0,j=0;
        while(i<n&&j<m)
        {
            if (j==-1||s[i]==t[j]) i++,j++;
            else j=Next[j];
        }
        if (j==m) return i-m;
        else return -1;
    }
}K;
```

常见应用

以下均指开始下标为0的字符串

- 长度为len字符串的最短循环节长度为$len-Next[len]$字符串-xiejiadong
- $Next[i]$表示$i-1$为结尾的字符串的最长公共前后缀（不算本身）
- 前缀i的最长循环节长度为$i-f[i]$(不算本身)

```
for (int i=1;i<=K.m;i++)
    if (K.Next[i]==0) f[i]=i;
    else f[i]=f[K.Next[i]];
```

扩展KMP

- $next[i]$表示模式串$T[0\cdots(m-1)]$和$T[i\cdots(m-1)]$的最长公共前缀。
- $extand[i]$表示模式串$T[0\cdots(m-1)]$和主串$S[i\cdots(m-1)]$的最长公共前缀。

```
struct exkmp
{
    int next[1000010],extand[1000010];
    char S[1000010],T[1000010];
    void GetNext()
    {
        int len=strlen(T),a=0;
        next[0]=len;
        while(a<len-1&&T[a]==T[a+1]) a++;
        next[1]=a;a=1;
        for(int k=2;k<len;k++)
        {
            int p=a+next[a]-1,L=next[k-a];
            if((k-1)+L>=p)
            {
                int j=(p-k+1)>0?(p-k+1):0;
                while(k+j<len&&T[k+j]==T[j]) j++;
                next[k]=j;
                a=k;
            }
            else next[k]=L;
        }
    }
    void GetExtand()
    {
        int slen=strlen(S),tlen=strlen(T),a=0;
        int MinLen=slen<tlen?slen:tlen;
        while(a<MinLen&&S[a]==T[a]) a++;
        extand[0]=a;a=0;
        for(int k=1;k<slen;k++)
        {
            int p=a+extand[a]-1,L=next[k-a];
            if((k-1)+L>=p)
            {
                int j=(p-k+1)>0?(p-k+1):0;
                while(k+j<slen&&j<tlen&&S[k+j]==T[j]) j++;
                extand[k]=j;
                a=k;
            }
            else extand[k]=L;
        }
    }
}K;
```

AC 自动机

- fail树上，如果x是y的孩子，说明y是x的后缀

```
const int SIGMA_SIZE = 26;
const int MAXNODE = 5000010;
const int MAXS = 200 + 10;
struct AhoCorasickAutomata
{
    int ch[MAXNODE][SIGMA_SIZE];
    int f[MAXNODE];    // fail函数
    int val[MAXNODE];  // 每个字符串的结尾结点都有一个非0的val
    int last[MAXNODE]; // fail链上的下一个单词结尾结点
    int cnt[MAXS]; //用来统计模式串被找到了几次
    int sz;
    void init()
    {
        sz = 1;
        memset(ch[0], 0, sizeof(ch[0]));
        memset(cnt, 0, sizeof(cnt));
    }
    int idx(char c){return c-'A';}
    // 插入字符串。v必须非0
    void insert(char *s, int v)
    {
        int u = 0, n = strlen(s);
        for(int i = 0; i < n; i++)
        {
            int c = idx(s[i]);
            if(!ch[u][c])
            {
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;
                ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        val[u] = v;
    }
    // 递归打印以结点j结尾的所有字符串
    void print(int j)
    {
        if(j)
        {
            cnt[val[j]]++;
            print(last[j]);
        }
    }
    // 在T中找模板
    void find(char* T)
    {
        int n = strlen(T);
        int j = 0; // 当前结点编号，初始为根结点
        for(int i = 0; i < n; i++)
        {
            int c = idx(T[i]);
            j = ch[j][c];
            if(val[j]) print(j);
            else if(last[j]) print(last[j]); // 找到了！
        }
    }
    // 计算fail函数
    void getFail()
    {
        queue<int> q;
        f[0] = 0;
        // 初始化队列
        for(int c = 0; c < SIGMA_SIZE; c++)
        {
            int u = ch[0][c];
            if(u) { f[u] = 0; q.push(u); last[u] = 0; }
        }
        // 按BFS顺序计算fail
        while(!q.empty())
```

```
        {
            int r = q.front(); q.pop();
            for(int c = 0; c < SIGMA_SIZE; c++)
            {
                int u = ch[r][c];
                if(!u) {ch[r][c] = ch[f[r]][c];continue;}
                q.push(u);
                int v = f[r];
                while(v && !ch[v][c]) v = f[v];
                f[u] = ch[v][c];
                last[u] = val[f[u]] ? f[u] : last[f[u]];
            }
        }
    }
}ac;
```

强制在线自动机

- 1 s 插入模式串 s
- 2 s 删除模式串 s
- 3 t 存在的模式串与主串 t 匹配，单个模式串的多个匹配均算

时间复杂度为均摊 $O(mlog|s|)$

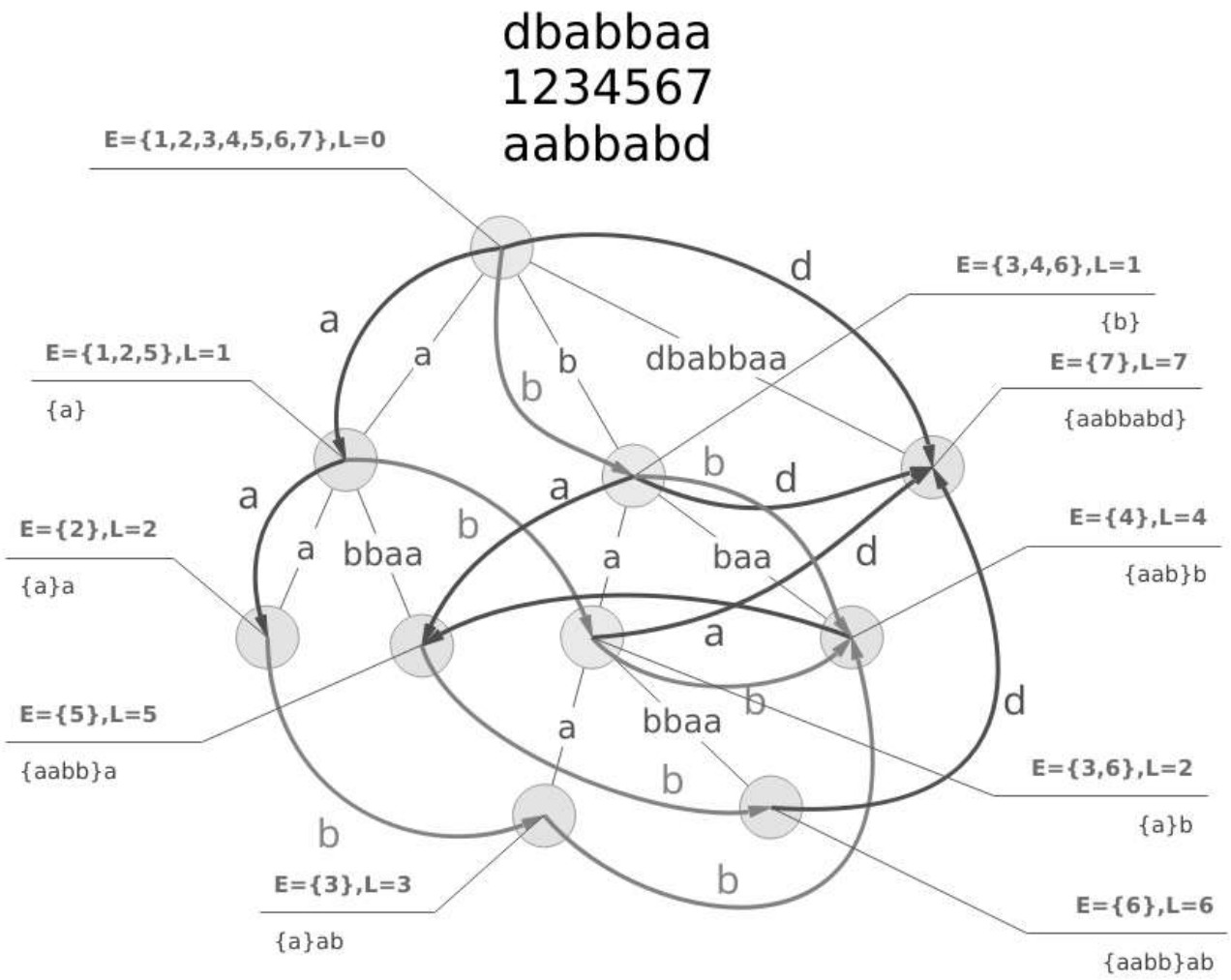
```
const int maxn=1e5+5,maxm=3e6+5,cut=300;    //maxn: 单个字符串最长长度 maxm: ac自动机最多节点数
struct node{
    node *fa,*ch[26],*fail;
    int end,cnt;    //cnt: 跑到这个节点上时，自动机内匹配成功的字符串数量
    node(){
        fa=fail=NULL; end=cnt=0;
        fill(ch,ch+26,fa);
    }
    inline void set_fail(node *u){
        fail=u;
        cnt=end+(u->cnt);
    }
};
node *que[maxm];
struct ACauto
{
    int siz;
    struct ac_auto{
        node *root;
        int size;
        ac_auto(){
            root=new node;
            size=0;
        }
        node *merge_node(node *a,node *b){
            if (!a) return b;
            if (!b) return a;
            node *v;
            for (int i=0;i<26;i++){
                v=a->ch[i]=merge_node(a->ch[i],b->ch[i]);
                if (v) v->fa=a;
            }
            a->end+=b->end;
            delete b;
            return a;
        }
        void clear(){
            int i,l,r;
            node *u,*v;
            l=r=0;
            que[r++]=root;
            while (l<r){
                u=que[l++];
                for (i=0;i<26;i++)
                    if (v=u->ch[i]) que[r++]=v;
                delete u;
            }
            root=new node;
            size=0;
        }
        void add_str(char s[])
        {
            node *u,*v;
            int i,id;
            u=root;
            for (i=0;s[i]!='\0';i++)
            {
                id=s[i]-'a';
                v=u->ch[id];
                if (!v){
                    v=new node;
                    u->ch[id]=v;
                    v->fa=u;
                }
                u=v;
            }
            v->end++;
            size+=i;
        }
        void merge_ac(ac_auto &b){
```

```
        size+=b.size;
        root=merge_node(root,b.root);
        b.root=new node;
        b.size=0;
    }
    void build_fail(){
        int i,l,r;
        node *u,*v,*w;
        l=0; r=0;
        for (i=0;i<26;i++)
            if (v=root->ch[i]){
                v->set_fail(root);
                que[r++]=v;
            }
        while (l<r){
            u=que[l++];
            for (i=0;i<26;i++)
                if (v=u->ch[i]){
                    que[r++]=v;
                    w=u->fail;
                    while (w!=root){
                        if (w->ch[i]){
                            v->set_fail(w->ch[i]);
                            break;
                        }
                        w=w->fail;
                    }
                    if (w==root){
                        if (w->ch[i]) v->set_fail(w->ch[i]);
                        else v->set_fail(root);
                    }
                }
        }
    }
    LL query(char s[]){
        node *u;
        int i,id;
        LL ans;
        u=root; ans=0;
        for (i=0;s[i]!='\0';i++){
            id=s[i]-'a';
            while (u!=root)
            {
                if (u->ch[id]){
                    u=u->ch[id];
                    break;
                }
                u=u->fail;
            }
            if (u==root){
                if (u->ch[id]) u=u->ch[id];else u=root;
            }
            ans+=u->cnt;
        }
        return ans;
    }
};
ac_auto ac1[maxn];
void clear()
{
    for (int i=0;i<siz;i++) ac1[i].clear();
    siz=0;
}
void insert(char s[])
{
    ac1[siz++].add_str(s);
    while (siz>1&&ac1[siz-1].size*2>ac1[siz-2].size){
        ac1[siz-2].merge_ac(ac1[siz-1]);
        siz--;
    }
    ac1[siz-1].build_fail();
```

```
    }
    LL query(char s[])
    {
        LL ans=0;
        for (int j=0;j<siz;j++)
            ans+=ac1[j].query(s);
        return ans;
    }
}ac1,ac2;
char s[maxn];
int main()
{
    int m,op;
    ac1.clear();ac2.clear();
    scanf("%d",&m);
    for (int i=0;i<m;i++)
    {
        scanf("%d%s",&op,s);
        if (op==1) ac1.insert(s);
        else if (op==3) printf("%lld\n",ac1.query(s)-ac2.query(s)),fflush(stdout);
        else ac2.insert(s);
    }
    return 0;
}
```

后缀

后缀自动机



- len[x] 状态 x 的最长子串长度
- fa[x] 状态 x 的后缀链接

- t 转移边
- a[] 为拓扑序
- f[x] 状态 x 是否为原始结点（没有被拆点）

```
#define N 2000010
const int M=N<<1;

struct sam
{
    int t[M][26],len[M]={-1},fa[M],sz=2,last=1,f[M];
    void init(){memset(t,0,(sz+10)*sizeof t[0]);sz=2;last=1;}
    void ins(int ch)
    {
        int p=last,np=last=sz++;f[np]=1;
        len[np]=len[p]+1;
        for (;p&&!t[p][ch];p=fa[p]) t[p][ch]=np;
        if (!p) {fa[np]=1;return;}
        int q=t[p][ch];
        if (len[p]+1==len[q]) fa[np]=q;
        else
        {
            int nq=sz++;len[nq]=len[p]+1;
            memcpy(t[nq],t[q],sizeof t[0]);
            fa[nq]=fa[q];
            fa[np]=fa[q]=nq;
            for (;t[p][ch]==q;p=fa[p]) t[p][ch]=nq;
        }
    }
    int c[M]={1},a[M];
    void rsort()
    {
        for (int i=1;i<sz;i++) c[i]=0;
        for (int i=1;i<sz;i++) c[len[i]]++;
        for (int i=1;i<sz;i++) c[i]+=c[i-1];
        for (int i=1;i<sz;i++) a[--c[len[i]]]=i;
    }
}
```

广义后缀自动机

```
#define N 2000010
const int M=N<<1;

struct sam{
    int t[M][10],len[M]={-1},fa[M],sz=2,last=1;
    LL cnt[M][2];
    void init(){memset(t,0,(sz+10)*sizeof t[0]);sz=2;last=1;}
    void ins(int ch,int id)//id 表示属于哪一个字符串
    {
        int p=last,np=0,nq=0,q=-1;
        if (!t[p][ch])
        {
            np=sz++;
            len[np]=len[p]+1;
            for (;p&&!t[p][ch];p=fa[p]) t[p][ch]=np;
        }
        if (!p) fa[np]=1;
        else
        {
            q=t[p][ch];
            if (len[p]+1==len[q]) fa[np]=q;
            else
            {
                nq=sz++;len[nq]=len[p]+1;
                memcpy(t[nq],t[q],sizeof t[0]);
                fa[nq]=fa[q];
                fa[np]=fa[q]=nq;
                for (;t[p][ch]==q;p=fa[p]) t[p][ch]=nq;
            }
        }
        last=np?np:nq?nq:q;
        cnt[last][id]=1;
    }
    int c[M]={1},a[M];
    void rsort()
    {
        for (int i=1;i<sz;i++) c[i]=0;
        for (int i=1;i<sz;i++) c[len[i]]++;
        for (int i=1;i<sz;i++) c[i]+=c[i-1];
        for (int i=1;i<sz;i++) a[--c[len[i]]]=i;
    }
};
```

按字典序建立后缀树

- 注意逆序插入
- rsort2 里的 a 不是拓扑序，需要拓扑序就去树上做
- one[] 是结点表示状态所有子串的结束位置（因为是反串插入，所以实际是原串的开始位置）
- ed[] 表示当前结点的结束位置 endpos 是子树 ed[] 集合

```
struct SAM
{
    int t[M][26],len[M]={-1},fa[M],sz=2,last=1,f[M],ed[M],one[M],pos[M];
    vector<int> G[M];
    void init()
    {
        memset(ed,0,sizeof(ed));
        memset(t,0,(sz+10)*sizeof t[0]);sz=2;last=1;
    }
    void ins(int ch, int pp)//pp是字符串位置
    {
        int p=last,np=last=sz++;ed[np]=pp+1;
        len[np]=len[p]+1;one[np]=pos[np]=pp;
        for (;p&&!t[p][ch];p=fa[p]) t[p][ch]=np;
        if (!p){fa[np]=1;return;}
        int q=t[p][ch];
        if (len[q]==len[p]+1) fa[np]=q;
        else
        {
            int nq=sz++;len[nq]=len[p]+1;one[nq]=one[q];
            memcpy(t[nq],t[q],sizeof t[0]);
            fa[nq]=fa[q];
            fa[q]=fa[np]=nq;
            for (;p&&t[p][ch]==q;p=fa[p]) t[p][ch]=nq;
        }
    assert(sz<=200000);
    }
    int up[M],c[256]={2},a[M];
    void rsort2()
    {
        for (int i=1;i<256;i++) c[i]=0;
        for (int i=2;i<sz;i++) up[i] = s[one[i] + len[fa[i]]];
        for (int i=2;i<sz;i++) c[up[i]]++;
        for (int i=1;i<256;i++) c[i] += c[i - 1];
        for (int i=2;i<sz;i++) a[--c[up[i]]] = i;
        for (int i=1;i<sz;i++) G[i].clear();
        for (int i=2;i<sz;i++) G[fa[a[i]]].push_back(a[i]);
    }
}sam;
```

经典应用

- 本质不同的子串数量

```
LL ans=0;
for (int i=2;i<a.sz;i++)
    ans+=(LL)(a.len[i]-a.len[a.fa[i]]);
```

- 状态 x 的出现次数

```
for (int i=a.sz-1;i>=1;i--)
    a.f[a.fa[a.a[i]]]+=a.f[a.a[i]];
```

回文

Manacher

- 注意 N 需要开两倍

```
struct Manacher
{
    int f[N];//f[i]代表以i为中心的最长回文串长度
    void solve(char *a,int n)//a="#a#b#a#"
    {
        int r=0,p=0;
        for(int i=0;i<n;i++)
        {
            if (i<r) f[i]=min(f[2*p-i],r-i);
            else f[i]=1;
            while (i-f[i]>=0&&i+f[i]<n&&a[i-f[i]]==a[i+f[i]]) f[i]++;
            if (f[i]+i-1>r) r=f[i]+i-1,p=i;
        }
        for (int i=0;i<n;i++) f[i]--;
    }
}man;
```

回文自动机

```
const int maxn = 300010*2;
const int ALP = 26;

struct PAM{
    int next[maxn][ALP];
    int fail[maxn] ;//fail指针，失配后跳转到fail指针指向的节点
    int cnt[maxn] ; //表示节点i表示的本质不同的串的个数（建树时求出的不是完全的，最后count()函数跑一遍以后才是正确的）
    int num[maxn] ; //表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
    int len[maxn] ;//len[i]表示节点i表示的回文串的长度（一个节点表示一个回文串）
    int s[maxn] ;//存放添加的字符
    int last ;//指向新添加一个字母后所形成的最长回文串表示的节点。
    int n ;//表示添加的字符个数。
    int p ;//表示添加的节点个数。

    int newnode(int l){
        for(int i=0;i<ALP;i++)
            next[p][i]=0;
        cnt[p]=num[p]=0;
        len[p]=l;
        return p++;
    }
    void init(){
        p = 0;
        newnode(0);
        newnode(-1);
        last = 0;
        n = 0;
        s[n] = -1;
        fail[0] = 1;
    }
    int get_fail(int x){
        while(s[n-len[x]-1] != s[n]) x = fail[x];
        return x;
    }
    void add(int c){
        c = c-'a';
        s[++n] = c;
        int cur = get_fail(last);
        if(!next[cur][c]){
            int now = newnode(len[cur]+2);    //注意字符串长度为1的是由-1拓展过来的，要特判
            fail[now] = next[get_fail(fail[cur])][c];
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        last = next[cur][c];
        cnt[last]++;
    }
    void count(){
        for(int i=p-1;i>=0;i--)
            cnt[fail[i]] += cnt[i];
    }
}pam;

char s[maxn];
int main(){
    scanf("%s",s);
    int len = strlen(s);
    pam.init();
    for(int i=0;i<len;i++)
    {
        pam.add(s[i]);
    }
    pam.count();
    long long ret = 0;
    for(int i=2;i<pam.p;i++)     //遍历所有的回文串
    {
        ret = max((long long)pam.len[i]*pam.cnt[i],ret);
    }
```

```
    cout<<ret<<endl;
    return 0;
}
```

遍历

对于 A 中的每个回文子串，B 中和该子串相同的子串个数的总和

```
LL dfs(int an,int bn){
    LL ret = 0;
    for(int i=0;i<ALP;i++) if(pam1.next[an][i]!=0 && pam2.next[bn][i]!=0)
        ret += (LL)pam1.cnt[pam1.next[an][i]] * pam2.cnt[pam2.next[bn][i]]
            + dfs(pam1.next[an][i],pam2.next[bn][i]);
    return ret;
}

LL ret = dfs(0,0) + dfs(1,1);
```

Palindrome Series

- 以 n 为结尾的所有回文串按照长度排序以后，可以得到 $k(k \leq log_2 n)$ 个按照长度的等差数列 。

下面代码能够离线求得子串 $[l,r]$ 中本质不同的回文串个数

```
const int maxn = 300010*4;
const int ALP = 26;
struct PAM{
    int next[maxn][ALP];
    int fail[maxn] ;//fail指针，失配后跳转到fail指针指向的节点
    int cnt[maxn] ; //表示节点i表示的本质不同的串的个数（建树时求出的不是完全的，最后count()函数跑一遍以后才是正确的）
    int num[maxn] ; //表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
    int len[maxn] ;//len[i]表示节点i表示的回文串的长度（一个节点表示一个回文串）
    int s[maxn] ;//存放添加的字符
    int last ;//指向新添加一个字母后所形成的最长回文串表示的节点。
    int n ;//表示添加的字符个数。
    int p ;//表示添加的节点个数。
    int fa[maxn],d[maxn],size[maxn],pos[maxn];
    //fa是等差数列第一项，d是公差，size是回文子树大小，pos是回文树dfs序
    int newnode(int l){
        for(int i=0;i<ALP;i++)
            next[p][i]=0;
        cnt[p]=num[p]=0;
        len[p]=l;
        return p++;
    }
    void init(){
        p = 0;
        newnode(0);
        newnode(-1);
        last = 0;
        n = 0;
        s[n] = -1;
        fail[0] = 1;
    }
    int get_fail(int x){
        while(s[n-len[x]-1] != s[n]) x = fail[x];
        return x;
    }
    void add(int c){
        c = c-'a';
        s[++n] = c;
        int cur = get_fail(last);
        if(!next[cur][c]){
            int now = newnode(len[cur]+2);    //注意字符串长度为1的是由-1拓展过来的，要特判
            fail[now] = next[get_fail(fail[cur])][c];
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
            d[now]=len[now]-len[fail[now]];
            if (d[now]==d[fail[now]]) fa[now]=fa[fail[now]];else fa[now]=now;
        }
        last = next[cur][c];
        cnt[last]++;
    }
    void count(){
        for(int i=p-1;i>=0;i--)
            cnt[fail[i]] += cnt[i];
    }
    vector<int> e[maxn];int sum;
    void dfs(int x)
    {
        pos[x]=++sum;
        size[x]=1;
        for (int i=0;i<e[x].size();i++)
        {
            dfs(e[x][i]);
            size[x]+=size[e[x][i]];
        }
    }
    void getfail()
    {
        for (int i=2;i<p;i++)
            e[fail[i]].push_back(i);
        sum=0;
        dfs(0);
```

```
    }
}pam;

char s[maxn];
int n,m;
struct data
{
    int l,id;
    data(int a=0,int b=0):l(a),id(b){}
};
vector<data> q[maxn];

struct ST
{
    int Max[maxn*4];
    int query(int now,int tl,int tr,int l,int r)
    {
        if (tl>=l&&tr<=r) return Max[now];
        int mid=(tl+tr)>>1,qx=0,qy=0;
        if (l<=mid) qx=query(now<<1,tl,mid,l,r);
        if (mid+1<=r) qy=query(now<<1|1,mid+1,tr,l,r);
        return max(qx,qy);
    }

    void modify(int now,int tl,int tr,int x,int v)
    {
        if (tl==tr) return (void) (Max[now]=v);
        int mid=(tl+tr)>>1;
        if (x<=mid) modify(now<<1,tl,mid,x,v);
        else modify(now<<1|1,mid+1,tr,x,v);
        Max[now]=max(Max[now<<1],Max[now<<1|1]);
    }
}tree;

struct bit
{
    int c[maxn];
    void add(int x,int v)
    {
        for (int i=x;i<=n;i+=i&-i)
            c[i]+=v;
    }
    int sum(int x)
    {
        int ret=0;
        for (int i=x;i;i-=i&-i)
            ret+=c[i];
        return ret;
    }
}bittree;

int res[maxn];

int main()
{
    scanf("%d%d",&n,&m);
    scanf("%s",s+1);
    pam.init();
    for(int i=1;i<=n;i++)
        pam.add(s[i]);
    pam.count();
    pam.getfail();
    int x,y;
    for (int i=1;i<=m;i++)
        scanf("%d%d",&x,&y),q[y].push_back(data(x,i));
    int now=1;
    for (int i=1;i<=n;i++)
    {
        while (s[i]!=s[i-pam.len[now]-1]) now=pam.fail[now];
        now=pam.next[now][s[i]-'a'];
        for (int x=now;x;x=pam.fail[pam.fa[x]])
```

```
        {
            int l=max(1,tree.query(1,1,pam.sum,pam.pos[x],pam.pos[x]+pam.size[x]-1)-pam.len[x]+2);
            int r=i-pam.len[pam.fa[x]]+2;
            bittree.add(l,1);bittree.add(r,-1);
        }
        tree.modify(1,1,pam.sum,pam.pos[now],i);
        for (int j=0;j<q[i].size();j++)
            res[q[i][j].id]=bittree.sum(q[i][j].l);
    }
}
```