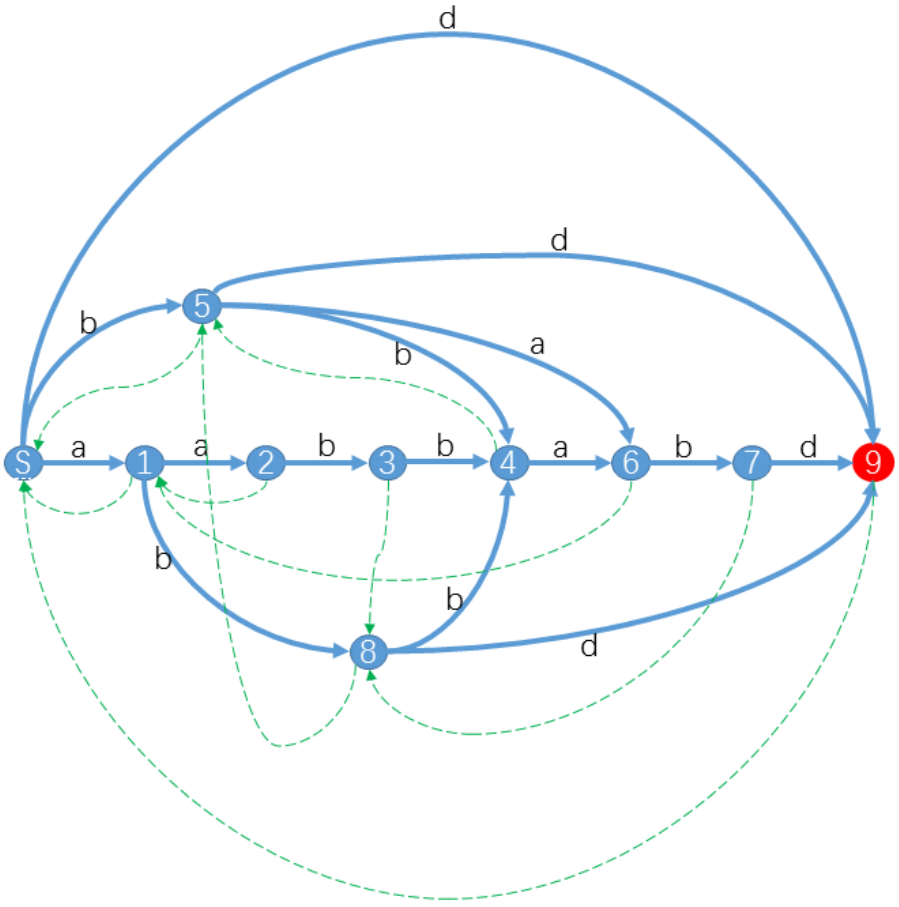
字符串

后缀自动机

aabbabd:




```
const int maxn = 2e5 + 5;
const int maxm = 26;

struct SAM {
    // 要开2倍字符串长度
    // 每次插入的是以i结点结尾的前缀
    // val 以i结尾的子串个数,等价于到根结点距离，也是i表示的最长的字符串的长度
    // i与pr[i] 成父子关系，父亲是儿子的一个后缀
    //添加第i个字符时，自动机里新增了diff = val[np] - val[pre[np]] 个与原先不同的子串，
    它们分别是 s[0.....i]，s[1.....i]，……，s[diff-1.....i]
    // pr[i] 不一定比 i 小
    int tr[maxn][maxm],pr[maxn],val[maxn],tot,last;
    int len[maxn],sub[maxn];
    void init() {
        last = tot = 0;
```

```
        memset(tr[0],-1,sizeof(tr[0]));
        pr[0] = -1;
        val[0] = 0;
    }
    void ins(int c) {
        int np = ++tot,p = last;
        val[np] = val[p] + 1;
        memset(tr[np], -1, sizeof(tr[np]));
        sub[np] = 1;
        while(~p && tr[p][c] == -1) tr[p][c] = np, p = pr[p];
        if(p == -1) pr[np] = 0;
        else {
            int q = tr[p][c];
            if(val[q] != val[p] + 1) {
                int nq = ++tot;
                memcpy(tr[nq], tr[q], sizeof(tr[q]));
                val[nq] = val[p] + 1;
                pr[nq] = pr[q];
                pr[q] = pr[np] = nq;
                while(~p && tr[p][c] == q) tr[p][c] = nq, p = pr[p];
            } else pr[np] = q;
        }
        last = np;
        //return val[np] - val[pr[np]];
    }
    queue<int> qq;
    void toposort() {
        for(int i = 1; i <= tot; i++) len[pr[i]]++;
        for(int i = 0; i <= tot; i++) {
            if(!len[i]) qq.push(i);
        }
        while(!qq.empty()) {
            int u = qq.front();
            qq.pop();
            sub[pr[u]] += sub[u];
            len[pr[u]]--;
            if(!len[pr[u]]) qq.push(pr[u]);
        }
    }
    int count() {
        int sum = 0;
        for(int i = 0; i <= tot; i++) sum += val[i] - val[pr[i]]; //i结点新增子串数
        return sum;
    }
    //公共串匹配
    int ff(char s[]) {
        int sz = strlen(s),u = 0,tmp = 0,c,i,res = 0;
        for(i = 0; i < sz; i++) {
            c = s[i] - 'a';
            if(~tr[u][c]) tmp++,u = tr[u][c];
            else {
                while(~u && tr[u][c] == -1) u = pr[u];
                if(~u) tmp = val[u] + 1,u = tr[u][c];
                else tmp = 0,u = 0;
            }
            res = max(res,tmp);
        }
        return res;
    }
} SS;
```

后缀数组

```
///后缀数组(Suffix Array)
/*
后缀数组是指将某个字符串的所有后缀按字典序排序后得到的数组
*/

///倍增法模板: $O(n\log{n})$采用基数排数
//n为字符个数 r[n - 1] 要比所有a[0, n - 2]要小
//r 字符串对应的数组
//m为最大字符值+1
// rk[i] 从下标i开始的后缀的排名
// sa[i] 第i小的后缀的起始下标位置 sa[0]为n
// height 是排名相邻的两个串的lcp  1~n
int sa[NUM];
int rk[NUM], height[NUM], sv[NUM], sn[NUM];
void da(char r[], int n, int m) {
  int i, j, k, *x = rk, *y = height;
  for (i = 0; i < m; i++) sn[i] = 0;
  for (i = 0; i < n; i++) sn[x[i] = r[i]]++;
  for (i = 1; i < m; i++) sn[i] += sn[i - 1];
  for (i = n - 1; i >= 0; i--) sa[--sn[x[i]]] = i;
  for (j = k = 1; k < n; j <<= 1, m = k) {
    for (k = 0, i = n - j; i < n; i++) y[k++] = i;
    for (i = 0; i < n; i++) if (sa[i] >= j) y[k++] = sa[i] - j;
    for (i = 0; i < n; i++) sv[i] = x[y[i]];
    for (i = 0; i < m; i++) sn[i] = 0;
    for (i = 0; i < n; i++) sn[sv[i]]++;
    for (i = 1; i < m; i++) sn[i] += sn[i - 1];
    for (i = n - 1; i >= 0; i--) sa[--sn[sv[i]]] = y[i];
    for (swap(x, y), x[sa[0]] = 0, i = k = 1; i < n; i++)
      x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] &&
                  y[sa[i] + j] == y[sa[i - 1] + j]) ? k - 1 : k++;
  }
  for (i = k = 0; i < n; i++) rk[sa[i]] = i;
  for (i = 0; i < n; height[rk[i++]] = k)
    for (k ? k-- : 0, j = sa[rk[i] - 1]; r[i + k] == r[j + k]; k++);
}

///DC3模板: $O(3n)$
int sa[NUM * 3], r[NUM * 3]; //sa数组和r数组要开三倍大小的空间
int rk[NUM], height[NUM], sn[NUM], sv[NUM];
#define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
#define G(x) ((x) < tb ? (x) * 3 + 1 : ((x) - tb) * 3 + 2)
int cmp0(int r[], int a, int b)
{return r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2];}
int cmp12(int r[], int a, int b, int k) {
  if (k == 2) return r[a] < r[b] || (r[a] == r[b] && cmp12(r, a + 1, b + 1, 1));
  else return r[a] < r[b] || (r[a] == r[b] && sv[a + 1] < sv[b + 1]);
}
void sort(int r[], int a[], int b[], int n, int m) { //基数排序
  int i;
  for (i = 0; i < m; i++) sn[i] = 0;
  for (i = 0; i < n; i++) sn[sv[i] = r[a[i]]]++;
  for (i = 1; i < m; i++) sn[i] += sn[i - 1];
  for (i = n - 1; i >= 0; i--) b[--sn[sv[i]]] = a[i];
}
void dc3(int r[], int sa[], int n, int m) {
  int *rn = r + n, *san = sa + n, *wa = height, *wb = rk;
  int i, j, p, ta = 0, tb = (n + 1) / 3, tbc = 0;
  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])] = cmp0(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++) sv[wb[i] = G(san[i])] = i;
  for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++)
    sa[p] = cmp12(r, wa[i], wb[j], wb[j] % 3) ? wa[i++] : wb[j++];
  for (; i < ta; p++) sa[p] = wa[i++];
  for (; j < tbc; p++) sa[p] = wb[j++];
}

///高度数组longest commest prefix
//height[i] = suffix(sa[i])和suffix(sa[i - 1])的最长公共前缀lcp(sa[i],sa[i-1])
//rk[0..n-1]:rk[i]保存的是原串中suffix[i]的名次
//height数组性质:
//任意两个suffix(j)和suffix(k)(rank[j] < rank[k])的最长公共前缀: $\displaystyle \min_{i = j + 1 \to k}{\{height[rank[i]]\}}$
//$height[rank[i]] \geq height[rank[i - 1]] - 1$
int rk[maxn], height[maxn];
void cal_height(char *r, int *sa, int n) {
  int i, j, k = 0;
  for (i = 0; i < n; i++)rk[sa[i]] = i;
  for (i = 0; i < n; height[rk[i++]] = k)
    for (k ? k-- : 0, j = sa[rk[i] - 1]; r[i + k] == r[j + k]; k++);
}
```

后缀数组应用

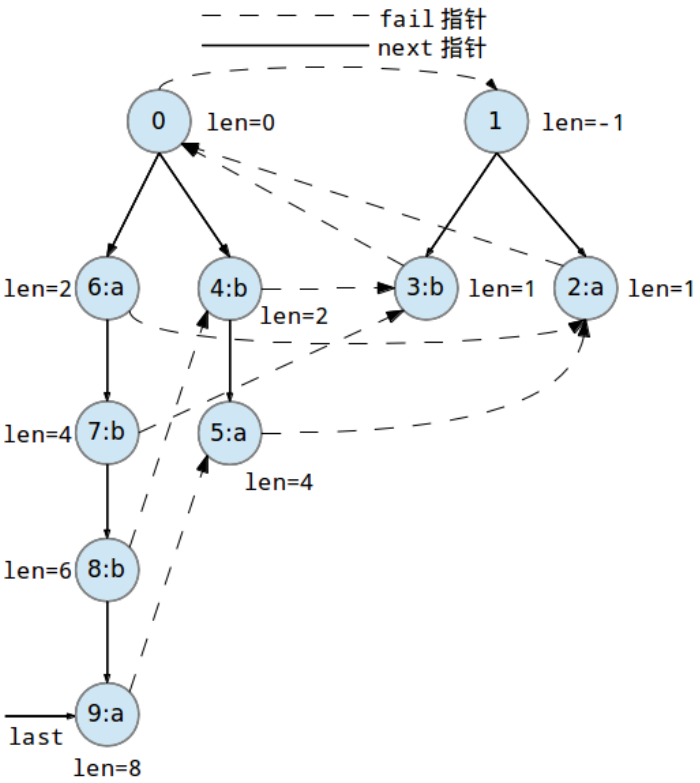
询问任意两个后缀的最长公共前缀: RMQ问题, min(i=j+1-->k){height[rk[i]]}
重复子串: 字符串R在字符串L中至少出现2次, 称R是L的重复子串
可重叠最长重复子串: $O(n)$ height数组中的最大值
不可重叠最长重复子串: $O(n \log n)$变为二分答案, 判断是否存在两个长度为k的子串是相同且不重叠的. 将排序后后缀分为若干组, 其中每组的后缀的height值都不小于k, 然后有希望成为最长公共前缀不小于k的两个后缀一定在同一组, 然后对于每组后缀, 判断sa的最大值和最小值之差不小于k, 如果一组满足, 则存在, 否则不存在.
可重叠的k次最长重复子串: $O(n \log n)$ 二分答案, 将后缀分为若干组, 判断有没有一个组的后缀个数不小于k.
不相同的子串个数: 等价于所有后缀之间不相同的前缀的个数O(n): 后缀按suffix(sa[1]), suffix(sa[2]), · · ·, suffix(n)的顺序计算, 新进一个后缀suffix(sa[k]), 将产生n - sa[k] + 1的新的前缀, 其中height[k]的和前面是相同的, 所以suffix(sa[k])贡献n - sa[k] + 1 - height[k]个不同的子串. 故答案是 $\sum_{k=1}^{n} n - sa[k] - 1 - height[k]$.
最长回文子串: 字符串S(长度n)变为字符串+特殊字符+反写的字符串, 求以某字符(位置k)为中心的最长回文子串(长度为奇数或偶数), 长度为: 奇数lcp(suffix(k), suffix(2*n + 2 - k)); 偶数lcp(suffix(k), suffix(2*n + 3 - k)) $O(n \log n)$ RMQ:O(n)
连续重复子串: 字符串L是有字符串S重复R次得到的.
给定L, 求R的最大值: O(n), 枚举S的长度k, 先判断L的长度是否能被k整除, 在看lcp(suffix(1), suffix(k+1))是否等于n - k. 求解时只需预处理height数组中的每一个数到height[rk[1]]的最小值即可
给定字符串, 求重复次数最多的连续重复子串$O(n \log n)$: 先穷举长度L, 然后求长度为L的子串最多能连续出现几次. 首先连续出现1次是肯定可以的, 所以这里只考虑至少2次的情况. 假设在原字符串中连续出现2次, 记这个子字符串为S, 那么S肯定包括了字符r[0], r[L], r[L*2], r[L*3], · · ·中的某相邻的两个. 所以只须看字符r[L*i*和r[*L*(i+1)]往前和往后各能匹配到多远, 记这个总长度为K, 那么这里连续出现了K/L+1次. 最后看最大值是多少.
字符串A和B最长公共前缀O(|A|+|B|): 新串: A+特殊字符#+B, 答案为排名相邻且属于不同的字符串的height的最大值
长度不小于k的公共子串的个数: 连接两串A+#+B, 对后缀数组分组(每组height值都不小于k), 每组中扫描到B时, 统计与前面的A的后缀能产生多少个长度不小于k的公共子串, 统计得结果.
给定n个字符串, 求出现在不小于k个字符串中的最长子串$O(n \log n)$: 连接所有字符串, 二分答案, 然后分组, 判断每组后缀是否出现在至少k个不同的原串中.
给定n个字符串, 求在每个字符串中至少出现两次且不重叠的最长子串$O(n \log n)$: 做法同上, 也是先将n个字符串连起来, 中间用不相同的且没有出现在字符串中的字符隔开, 求后缀数组. 然后二分答案, 再将后缀分组. 判断的时候, 要看是否有一组后缀在每个原来的字符串中至少出现两次, 并且在每个原来的字符串中, 后缀的起始位置的最大值与最小值之差是否不小于当前答案(判断能否做到不重叠, 如果题目中没有不重叠的要求, 那么不用做此判断).
给定n个字符串, 求出现或反转后出现在每个字符串中的最长子串: 只需要先将每个字符串都反过来写一遍, 中间用一个互不相同的且没有出现在字符串中的字符隔开, 再将n个字符串全部连起来, 中间也是用一个互不相同的且没有出现在字符串中的字符隔开, 求后缀数组. 然后二分答案, 再将后缀分组. 判断的时候, 要看是否有一组后缀在每个原来的字符串或反转后的字符串中出现. 这个做法的时间复杂度为$O(n \log n)$.

利用后缀数组和ST表得到与第pos个后缀有长度为len的公共前缀的后缀范围[getL(pos, val), getR(pos, val)]

```
struct ST {
  int st[NUM][22], Lg[NUM];
  void init(const int a[], int n) {
```

```
    Lg[1] = 0;
    for (int i = 2; i <= n; ++i) Lg[i] = Lg[i >> 1] + 1;
    for (int i = n - 1; i >= 0; --i) {
      st[i][0] = a[i];
      for (int j = 1; i + (1 << j) <= n; ++j)
        st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
  }
  int Min(int l, int r) {
    int k = Lg[r - l + 1];
    return min(st[l][k], st[r - (1 << k) + 1][k]);
  }
  int getL(int pos, int val) {
    int l = 0, r = pos - 1, mid, ans = pos;
    while (l <= r) {
      mid = (l + r) >> 1;
      if (Min(mid + 1, pos) >= val) {
        ans = mid;
        r = mid - 1;
      }
      else l = mid + 1;
    }
    return ans;
  }
  int getR(int pos, int val) {
    int l = pos + 1, r = N, mid, ans = pos;
    while (l <= r) {
      mid = (l + r) >> 1;
      if (Min(pos + 1, mid) >= val) {
        ans = mid;
        l = mid + 1;
      }
      else r = mid - 1;
    }
    return ans;
  }
} st;
```

回文自动机




```
const int MAXN = 100005;
const int N = 26;

struct Palindromic_Tree
{
    int next[MAXN][N];//next指针，next指针和字典树类似，指向的串为当前串两端加上同一个字符构成
    int fail[MAXN];//fail指针，失配后跳转到fail指针指向的节点
    int cnt[MAXN];//节点i表示的本质不同的串的个数,要跑一遍count函数才行
    int num[MAXN];//以节点i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
    int S[MAXN];//存放添加的字符
    int last;//指向上一个字符所在的节点，方便下一次add
    // 把以当前点结尾的相邻两个回文串的长度作差,把相邻的相同差值的归为一段,可以证明最多只有log段
    // diff表示差值,slink表示下一个不同差值的点 fp表示该点dp取最优值的前驱位置
    int diff[maxn], slink[maxn], fp[maxn];
    int n;//字符数组指针
    int p;//节点指针

    int newnode(int l) //新建节点
    {
        for(int i = 0; i < N; ++i) next[p][i] = 0;
        cnt[p] = 0;
        num[p] = 0;
        len[p] = l;
        return p++;
    }
    void init() //初始化
    {
        p = 0;
        newnode(0);
        newnode(-1);
        last = 0;
        n = 0;
        S[n] = -1;//开头放一个字符集中没有的字符，减少特判
        fail[0] = 1;
    }
    int get_fail(int x) //和KMP一样，失配后找一个尽量最长的
    {
```

```
        while(S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }
    void add(int c)
    {
        c -= 'a';
        S[++n] = c;
        int cur = get_fail(last);//通过上一个回文串找这个回文串的匹配位置
        if ( !next[cur][c]) //如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串
        {
            int now = newnode(len[cur] + 2) ;//新建节点
            fail[now] = next[get_fail(fail[cur])][c];//和AC自动机一样建立fail指针，以便失配后跳转
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
            diff[now] = len[now] - len[fail[now]];
            slink[now] = (diff[now] == diff[fail[now]] ? slink[fail[now]] : fail[now]);
        }
        last = next[cur][c];
        return last;
    }
    // 回文分解,记得初始化dp
    void solve(int dp[], int pre[], char s[], int n) {
        int i, j;
        init();
        dp[0] = 0;
        fp[0] = 1;
        // 字符串范围在1-n
        for(i = 1; i <= n; i++) {
            // 以差值不同分为不同的部分
            for(j = add(s[i]); j; j = slink[j]) {
                // 这个部分的第一个下标
                fp[j] = i - (len[slink[j]] + diff[j]);
                // 差值相同表示属于同一个部分
                // 因为前面把fail[j]之后的点的最优值处理到了fail[j],所以只需要处理j和fail[j]即可得到这个部分的最优值
                // 并把最优值放到fp[j]里
                if(diff[fail[j]] == diff[j] && dp[fp[j]] > dp[fp[fail[j]]]) fp[j] = fp[fail[j]];
                if(dp[i] > dp[fp[j]] + 1) {
                    dp[i] = dp[fp[j]] + 1;
                    pre[i] = fp[j]; // 方案记录
                }
            }
        }
    }
    void count()
    {
        for(int i = p - 1;i >= 0; --i) cnt[fail[i]] += cnt[i];
        //父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串！
    }
};
```

AC自动机

```
const int csize = 4;
const int maxn = 1e5 + 5;
int f[maxn][csize + 1];//csize为当前结点fail指针
int m;
int val[maxn];
char ss[maxn];

void insert(char *s) {
    int l = strlen(s);
    int i,c,t = 1;
```

```
    for(i = 0; i < l; i++) {
        c = mp[s[i]];
        if(!f[t][c]) {
            memset(f[m],0,sizeof(f[m]));
            f[t][c] = m++;
        }
        t = f[t][c];
    }
    val[t] = 1;
}

queue<int>q;
void build() {
    int u,v,i;
    q.push(1);
    while(!q.empty()) {
        u = q.front();
        q.pop();
        for(i = 0; i < csize; i++) {
            if(f[u][i]) {
                v = f[u][csize];
                while(v && !f[v][i]) v = f[v][csize];
                f[f[u][i]][csize] = v ? f[v][i] : 1 ;
                q.push(f[u][i]);
            } else f[u][i] = (u != 1) ? f[f[u][csize]][i] : 1 ;
        }
    }
}
```

KMP

```
int f[100000];//next
// 范围 1~n
// 特征匹配(满足相同大小关系)下的kmp  p[j] =  p[i]变为大于该值的数的个数和等于该值的数的个数分别相等
// 即重定义相等关系
int kmp(string p,string s) {
    int n = p.size();
    int m = s.size();
    int i,j;
    int ans = 0;
    memset(f,0,sizeof(f));
    for(i = 1; i < n; i++) {
        j = i;
        while(j > 0) {
            j = f[j];
            if(p[j] == p[i]) {
                f[i+1] = j + 1;
                break;
            }
        }
    }
    for(i = 0,j = 0; i < m; i++) {
        if(j < n && s[i] == p[j]) j++;
        else {
            while(j > 0) {
                j = f[j];
                if(s[i] == p[j]) {
                    j++;
                    break;
                }
            }
        }
```

```
        if(j == n) ans++;
    }
    return ans;
}
```

EXKMP

```
const int nn = 100005;
int pa[nn],pb[nn];//a,b的匹配结果
char a[nn],b[nn];

void exkmp() {
    int i,j,k,m,n;
    int len,l;
    m = n = strlen(b);
    for(j = 0; j + 1 < m && a[j] == a[1 + j]; j++);
    pa[1] = j;
    for(i = 2,k = 1; i < m; i++) {
        len = k + pa[k];
        l = pa[i - k];
        if(l + i < len) pa[i] = l;
        else {
            for(j = max(0,len - i); i + j < m && a[j] == a[i + j]; j++);
            pa[i] = j;
            k = i;
        }
    }
    for(j = 0; j < n && i < m && a[j] == b[j]; j++);
    pb[0] = j;
    for(i = 1,k = 0; i < n; i++) {
        len = k + pb[k];
        l = pa[i - k];
        if(l + i < len) pb[i] = l;
        else {
            for(j = max(0,len - i); i + j < n && j < m && a[j] == b[i + j]; j++);
            pb[i] = j;
            k = i;
        }
    }
}
```

Shift And

适用于模式串比较短,每个匹配位置可以有多个字符可以匹配
定义$bitset <> T[i]$为第i字符可以在哪些匹配位置出现
核心代码$D = (D << 1|1)\&T[S[i]]$

最小串表示

```
string smallest(string s) {
    int i,j,k,l;
    int n = s.size() / 2;
    for(i = 0,j = 1; j < n; ) {
        for(k = 0; k < n && s[i + k] == s[j + k]; k++);
        if(k >= n) break;
        if(s[i + k] < s[j + k]) j += k + 1;
        else {
            l = i + k;
            i = j;
            j = max(l,j) + 1;
        }
```

```
    }
    printf("%d",i + 1);
    return s.substr(i,n);
}
```

Manacher算法

```
const int N=200005;
char T[N];   //原字符串
char S[N];   //转换后的字符串
int  R[N];   //回文半径

void Init(char *T) {
    S[0] = 24;
    int len=strlen(T);
    for(int i=0; i<=len; i++) {
        S[2*i+1]='#';
        S[2*i+2]=T[i];
    }
}

int Manacher(char *S) {
    int k=0,mx=0;
    int len=strlen(S);
    for(int i=0; i<len; i++) {
        if(mx>i)
            R[i]=R[2*k-i]<mx-i? R[2*k-i] : mx-i;
        else
            R[i]=1;
        while(S[i+R[i]]==S[i-R[i]])
            R[i]++;
        if(R[i]+i>mx) {
            mx=R[i]+i;
            k=i;
        }
    }
    int ans=1;
    for(int i=0; i<len; i++)
        ans=R[i]>ans? R[i] : ans;
    return ans - 1;
}
```

区间不同子串个数

复杂度$nlog^2n$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef pair<int,int> P;
const int N = 100005;
int go[N << 1][26],fail[N << 1],len[N << 1],tot,last;
int n;
int cnt = 0;
namespace seg {
int tag[N << 2];
LL sum[N << 2];
inline void Tag(int me,int l,int r,int v) {
    tag[me] += v;
    sum[me] += (r - l + 1) * 1ll * v;
}
inline void down(int me,int l,int r) {
```

```
    if(tag[me] == 0)return;
    int mid = (l + r) >> 1;
    Tag(me << 1,l,mid,tag[me]);
    Tag(me << 1 | 1,mid + 1,r,tag[me]);
    tag[me] = 0;
}
void add(int me,int l,int r,int x,int y,int v) {
    if(l ^ r) down(me,l,r);
    if(x <= l && r <= y) {
        Tag(me,l,r,v);
        return;
    }
    int mid = (l + r) >> 1;
    if(x <= mid) add(me << 1,l,mid,x,y,v);
    if(y > mid) add(me << 1 | 1,mid + 1,r,x,y,v);
    sum[me] = sum[me << 1] + sum[me << 1 | 1];
}
LL ask(int me,int l,int r,int x,int y) {
    if(l ^ r) down(me,l,r);
    if(x <= l && r <= y)return sum[me];
    int mid = (l + r) >> 1;
    LL ret = 0;
    if(x <= mid) ret += ask(me << 1,l,mid,x,y);
    if(y > mid) ret += ask(me << 1 | 1,mid + 1,r,x,y);
    return ret;
}
void Do(int pre,int now,int L,int R) {
    if(L > R)return;
    ++cnt;
    if(pre)add(1,1,n,pre - R + 1,pre - L + 1,-1);
    add(1,1,n,now - R + 1,now - L + 1,1);
}
};
namespace lct {
int l[N << 2],r[N << 2],fa[N << 2];
int last[N << 2];
inline bool top(int x) {
    return (!fa[x]) || (l[fa[x]] != x && r[fa[x]] != x);
}
inline void left(int x) {
    int y = fa[x];
    int z = fa[y];
    r[y] = l[x];
    if(l[x]) fa[l[x]] = y;
    fa[x] = z;
    if(l[z] == y) l[z] = x;
    else if(r[z] == y) r[z] = x;
    l[x] = y;
    fa[y] = x;
}
inline void right(int x) {
    int y = fa[x];
    int z = fa[y];
    l[y] = r[x];
    if(r[x]) fa[r[x]] = y;
    fa[x] = z;
    if(l[z] == y) l[z] = x;
    else if(r[z] == y) r[z] = x;
    r[x] = y;
    fa[y] = x;
}
inline void down(int x) {
    if(l[x]) last[l[x]] = last[x];
    if(r[x]) last[r[x]] = last[x];
```

```
}
int q[N << 2];
inline void splay(int x) {
    q[q[0] = 1] = x;
    for(int k = x; !top(k); k = fa[k]) q[++q[0]] = fa[k];
    for(int i = q[0]; i >= 1; i--) down(q[i]);
    while(!top(x)) {
        int y = fa[x];
        int z = fa[y];
        if(top(y)) {
            if(l[y] == x) right(x);
            else left(x);
        } else {
            if(r[z] == y) {
                if(r[y] == x) left(y),left(x);
                else right(x),left(x);
            } else {
                if(l[y] == x) right(y),right(x);
                else left(x),right(x);
            }
        }
    }
}
void Access(int x,int cov) {
    int y = 0;
    for(; x; y = x,x = fa[x]) {
        splay(x);
        down(x);
        r[x] = 0;
        int L,R;
        int z = x;
        while(l[z]) z = l[z];
        L = len[fail[z]] + 1;
        splay(z);
        splay(x);
        z = x;
        while(r[z]) z = r[z];
        R = len[z];
        splay(z);
        splay(x);
        seg::Do(last[x],cov,L,R);
        r[x] = y;
        last[x] = cov;
    }
}
void SetFa(int x,int y,int po) {
    fa[x] = y;
    Access(x,po);
}
void split(int x,int y,int d) {
    splay(y);
    down(y);
    r[y] = 0;
    fa[d] = y;
    splay(x);
    fa[x] = d;
    last[d] = last[x];
}
};
namespace sam {
void init() {
    tot = last = 1;
}
void expended(int x,int po) {
```

```
    int gt = ++tot;
    len[gt] = len[last] + 1;
    int p = last;
    last = tot;
    for(; p && (!go[p][x]); p = fail[p]) go[p][x] = gt;
    if(!p) {
        fail[gt] = 1;
        lct::SetFa(gt,1,po);
        return;
    }
    int xx = go[p][x];
    if(len[xx] == len[p] + 1) {
        fail[gt] = xx;
        lct::SetFa(gt,xx,po);
        return;
    }
    int tt = ++tot;
    len[tt] = len[p] + 1;
    fail[tt] = fail[xx];
    int dt = fail[xx];
    fail[xx] = fail[gt] = tt;
    lct::split(xx,dt,tt);
    lct::SetFa(gt,tt,po);
    for(int i = 0; i <= 25; i++) go[tt][i] = go[xx][i];
    for(; p && (go[p][x] == xx); p = fail[p]) go[p][x] = tt;
}
};
int Q;
char str[N];
int qL[N];
vector<int> que[N];
LL ans[N];
void Main() {
    for(int i = 1; i <= n; i++) {
        sam::expended(str[i] - 'a',i);
        for(int j = 0; j < (int)que[i].size(); j++) {
            int id = que[i][j];
            ans[id] = seg::ask(1,1,n,qL[id],n);
        }
    }
}
void init() {
    int w,r;
    scanf("%s",str + 1);
    n = strlen(str + 1);
    scanf("%d%d",&Q,&w);
//询问区间[qL[i],r],下标从1开始
    for(int i = 1; i <= Q; i++) {
        scanf("%d",&qL[i]);
        r = qL[i] + w - 1;
        que[r].push_back(i);
    }
    sam::init();
}
void Output() {
    for(int i = 1; i <= Q; i++) printf("%lld\n",ans[i]);
}
int main() {
    init();
    Main();
    Output();
    return 0;
}
```