提高组400+试题 第五组-新

选数字

题解

本题解法预处理的复杂度为O(256*n)，单次查询的复杂度为O(256*m)。空间复杂度是O(256*n)。核心思想是使用容斥。

假如我们预处理出每个位置0-255出现了多少次的前缀，就可以O(1)统计出区间内每个数字出现的次数。但在这个基础上枚举3元组数量的话，单次查询的计算量为256*256。

如果考虑用预处理的数组，记录前缀有多少个数ai Or 0-255(ni)，等于ni，也就是如果ni在某一bit上为0，则ai在这个bit上也为0。如果ni在某一bit上为1，则ai在这一bit上可能为0也可能为1。

例如：ni = 180，180的二进制表示为10110100，对应的ai包括： 0(0) 4(100) 16(10000) 20(10100) 32(100000) 36(100100) 48(110000) 52(110100) 128(10000000) 132(10000100) 144(10010000) 148(10010100) 160(10100000) 164(10100100) 176(10110000) 180(10110100)

这样对于每一个查询l,r,x，我们可以通过预处理数组，O(1)知道(l,r)区间内，有多少个ai使得ai or x = x。我们设这个值为q。

还是以180为例，假如区间(l,r)中有q个数or 180 = 180。从这q个数中任取3个，他们or在一起的值，只可能是上面列表中的这16个值。我们在这个基础上，设计一个容斥，就可以求出具体有多少个三元组Or在一起的值恰好等于。

设F(l,r,n)表示区间(l,r)中有多少个数 Or n = n。G(l,r,n)表示区间有多少三元组Or在一起等于n。

则： $G(l,r,180) = \binom{F(l,r,180)}{3} - \binom{F(l,r,176)}{3} - \binom{F(l,r,164)}{3} \ldots + \binom{F(l,r,36)}{3} \ldots$

其中所有与180的二进制表示相差奇数个bit的系数为-1，偶数个bit的为+1。 每次计算最坏情况下，需要枚举256个数，所以单次查询的复杂度为O(256)。

标准代码

C++

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MaxN = 100000;
int a[MaxN + 5];
int sum[MaxN + 5][300 + 5];
int cont[300 + 5];

long long cal(long long x)
{
    return x * (x - 1) * (x - 2) / 6;
}

int main()
{
    cont[0] = 0, cont[1] = 1, cont[2] = 1;
    for (int i = 3; i <= 300; i++) cont[i] = cont[i / 2] + (i & 1);
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= 255; j++) {
            if((a[i] & j) == a[i]) {
                sum[i][j] = sum[i - 1][j] + 1;
            }
            else sum[i][j] = sum[i - 1][j];
        }
    }

    for (int cas = 1; cas <= m; cas++) {
        int l, r, x; scanf("%d%d%d", &l, &r, &x);
        long long res = 0;
        for (int j = 0; j <= 255; j++) {
            if((j & x) == j) {
                int op = cont[x] - cont[j];
                if(op & 1) res -= cal(sum[r][j] - sum[l - 1][j]);
                else res += cal(sum[r][j] - sum[l - 1][j]);
            }
        }
        printf("%lld\n", res);
    }
    return 0;
}
```

堆箱子

题解

将仓库俯视图的这个矩形长宽分别拓宽L并放上雕像，

那么每行每列都是若干个正方形+一条线组成的东西（线长x）

也就是(L+x)的若干倍

那么很明了了 就是求出最小的(L+x)，使得(M+L)和(N+L)除以它都能得到整数 那么只要g=(M+L,N+L)除以它是整数

假设g/(L+x)=y，那么就是求满足g/y>=L的最大y

显然是g/L取下整

那么答案显然是g/y-L

标准代码

C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstring>
#include <cstdio>
#include <cmath>
using namespace std;
typedef long long lol;
int gi(){
    int res=0,fh=1;char ch=getchar();
    while((ch>'9'||ch<'0')&&ch!='-')ch=getchar();
    if(ch=='-')fh=-1,ch=getchar();
    while(ch>='0'&&ch<='9')res=res*10+ch-'0',ch=getchar();
    return fh*res;
}
const int MAXN=100001;
const int INF=1e9;
int main(){
    int a=gi(),h=gi(),w=gi(),g=__gcd(h+a,w+a);
    if(h<a||w<a||g<a){printf("-1");return 0;}
    printf("%.5lf",1.0*g/(g/a)-a);
    return 0;
}


```

快速排序

题解

每次中枢选择时选中最大值或者最小值可以使得只排除掉一个数字，从而得到最大递归深度n。 要使得字典序最小，a数组给出后可以知道每次中枢选择的位置，如果该位置左边的全部数字已经填满，则该位置放置当前处理的最小值，否则放置最大值。

读懂题目中的快排代码，正确理解选中最小值和最大值时候得交换逻辑，并正确维护原始下标即可。

例如样例中的

4 2

1 2

这个数据，可以先维护下标数组

[0] [1] [2] [3]

第一次选择1这里做为中枢，此处填入最大值或者最小值均可满足最大递归深度的要求。

而要满足最小字典序，则最小值应该尽可能放更左侧。而[1]左侧还有[0]没有填入数据，所以最小值应该留着放在[0]

所以[1]放入最大值4，数据为[0] 1 [2] [3]

按照前面代码的交换逻辑，与最前面交换，数组变为

1 [0] [2] [3]

再根据循环逻辑，交换得到

[3] [0] [2] 1

然后开始下一轮递归，此时选择index为2的数字作为中枢

由于[0]还为空[2]处仍然应该填入最大值3

[3] [0] 2 1

2与数组头[3]交换作为中枢，然后遍历后再把中枢放置到应该在的数组末尾，整个数组仍然为

[3] [0] 2 1

继续下一轮递归，此时选择index为1的数字作为中枢 [0]应该填入最小值1，数组变为

[3] 0 2 1

交换之后为

0 [3] 2 1

然后[3]填入最后元素2

0 3 2 1

数组排序完成

按照原始下标来看，题目要求的解就是

1 4 3 2

标准代码

C++

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 50000+55;
int n,k,a[N],h[N],r[N];

int main()
{
    while (scanf("%d%d",&n,&k)==2){
        for (int i=0;i<k;++i) scanf("%d",&r[i]);
        memset(a,0,sizeof a);
        for (int i=1;i<=n;++i) h[i] = i;
        int L=1,R=n,wL = 1;
        for (int i=0;i<n;++i){
            int x = r[i%k]%(n-i);
            x = x + L;
            if (h[x]==wL) {
                a[ h[x] ] = L;
                swap(h[x],h[L]);
                L++;
                while (a[wL]) wL++;
            }
            else{
                a[ h[x] ] = R;
                swap(h[x],h[L]);
                swap(h[L],h[R]);
                R--;
            }
        }

        for (int i=1;i<=n;++i) printf("%d\n",a[i]);
    }
    return 0;
}


```

统计学带师

题解

考虑对 $s_{1\dots n}$ 建出广义后缀自动机（把 S 一起建进去也是可以的），记 $\mathrm{endpos}_i(p)$ 为状态 p 在 s_i 中的终止位置集合。

对于一次询问，可以考虑通过预处理 + 倍增来找到 $S_{l\dots r}$ 在后缀自动机上对应的状态，并将询问离线到 Parent Tree 上。

则问题变为对于后缀自动机上的一个状态 p ，求 $\left|\mathrm{endpos}_i(p)\right|$ 的第 k 小值。

如果只是要求得 $\left|\mathrm{endpos}_i(p)\right|$ 的值，只需要在 Parent Tree 上对子树线段树合并即可解决。

求第 k 小值让我们联想到对 $\left|\mathrm{endpos}_i(p)\right|$ 建**权值线段树**。

那么，一个解决方案是，在线段树合并的**同时维护一棵权值线段树**。首先合并这棵权值线段树，再在另一棵线段树合并到叶子结点时，若双方都有值，在权值线段树上同时修改贡献。

复杂度可以使用类似启发式合并的方式来证明。

也有另一个和这个方法本质相同的解决方案：使用静态链分治（即「树上启发式合并」），每次遍历所有儿子的子树，保留重儿子子树内维护的权值线段树，再次遍历轻儿子子树来计算轻儿子子树的贡献。

复杂度的证明非常经典，留作读者思考 / 自行查找资料解决。

以上第一种方案的复杂度是 $O((n+q)\log n)$ 的，但常数较大；而第二种方案的复杂度是 $O(n\log^2 n+q\log n)$ 的（视 $n,|S|,\sum|s_i|$ 同阶）。

当然，只需要将权值线段树可持久化即可在线地解决这道题目。

标准代码

C++

```cpp
#include <cstdio>
#include <vector>
#include <utility>
#include <cstring>
using namespace std;
const int N = 1e5;
const int Q = 2e5;
const int LG = 18;
int n,m,q;
int ed[N + 5],len[N + 5];
char s[N + 5],t[N + 5];
int ans[Q + 5];
namespace SEG
{
    #define ls (p << 1)
    #define rs (ls | 1)
    int seg[(N << 3) + 10];
    void insert(int x,int k,int p,int tl,int tr)
    {
        seg[p] += k;
        if(tl == tr)
            return ;
        int mid = tl + tr >> 1;
        x <= mid ? insert(x,k,ls,tl,mid) : insert(x,k,rs,mid + 1,tr);
    }
    int query(int k,int p,int tl,int tr)
    {
        if(tl == tr)
            return tl;
        int mid = tl + tr >> 1;
        int sum = seg[ls];
        return k <= sum ? query(k,ls,tl,mid) : query(k - sum,rs,mid + 1,tr);
    }
}
namespace SAM
{
    struct node
    {
        int ch[26];
        int fa,len;
    } sam[(N << 1) + 5];
    int las = 1,tot = 1;
    int c[N + 5],a[(N << 1) + 5];
    int sz[(N << 1) + 5],son[(N << 1) + 5];
```

```
45        vector<int> edp[(N << 1) + 5];
46        inline void insert(int x,int pos)
47        {
48            if(sam[las].ch[x])
49            {
50                int cur = las,q = sam[cur].ch[x];
51                if(sam[cur].len + 1 == sam[q].len)
52                    las = q,++sz[las],edp[las].push_back(pos);
53                else
54                {
55                    int nxt = ++tot;
56                    sam[nxt] = sam[q],sam[nxt].len = sam[cur].len + 1,sam[q].fa = nxt;
57                    for(;cur && sam[cur].ch[x] == q;cur = sam[cur].fa)
58                        sam[cur].ch[x] = nxt;
59                    las = nxt,++sz[las],edp[las].push_back(pos);
60                }
61                return ;
62            }
63            int cur = las,p = ++tot;
64            sam[p].len = sam[cur].len + 1;
65            for(;cur && !sam[cur].ch[x];cur = sam[cur].fa)
66                sam[cur].ch[x] = p;
67            if(!cur)
68                sam[p].fa = 1;
69            else
70            {
71                int q = sam[cur].ch[x];
72                if(sam[cur].len + 1 == sam[q].len)
73                    sam[p].fa = q;
74                else
75                {
76                    int nxt = ++tot;
77
78                    sam[nxt] = sam[q],sam[nxt].len = sam[cur].len + 1,sam[p].fa = sam[q].fa = nxt;    for(;cur && sam[cur].ch[x] == q;cur = sam[cur].fa)
79                        sam[cur].ch[x] = nxt;
80                }
81            }
82            las = p,++sz[las],edp[las].push_back(pos);
83        }
84        int to[(N << 1) + 5],pre[(N << 1) + 5],first[(N << 1) + 5];
85        inline void add(int u,int v)
86        {
87            static int tot = 0;
```

```
88          to[++tot] = v,pre[tot] = first[u],first[u] = tot;
89      }
90      int f[LG + 5][(N << 1) + 5];
91      inline void build()
92      {
93          for(register int i = 1;i <= tot;++i)
94              ++c[sam[i].len],i > 1 && (add(f[0][i] = sam[i].fa,i),1);
95          for(register int i = 1;i <= LG;++i)
96              for(register int j = 2;j <= tot;++j)
97                  f[i][j] = f[i - 1][f[i - 1][j]];
98          for(register int i = 1;i <= N;++i)
99              c[i] += c[i - 1];
100         for(register int i = tot;i > 1;--i)
101             a[c[sam[i].len]--] = i;
102         for(register int i = tot;i > 1;--i)
103         {
104             sz[sam[a[i]].fa] += sz[a[i]];
105             if(!son[sam[a[i]].fa] || sz[son[sam[a[i]].fa]] < sz[a[i]])
106                 son[sam[a[i]].fa] = a[i];
107         }
108     }
109     inline int get(int p,int d)
110     {
111         if(sam[p].len < d)
112             return 0;
113         for(register int i = LG;~i;--i)
114             if(f[i][p] && sam[f[i][p]].len >= d)
115                 p = f[i][p];
116         return p;
117     }
118     vector< pair<int,int> > qry[(N << 1) + 5];
119     int cnt[N + 5];
120     void dfs(int p)
121     {
122         for(register int i = first[p];i;i = pre[i])
123             if(to[i] ^ son[p])
124             {
125                 dfs(to[i]);
126
                 for(register vector<int>::iterator it = edp[to[i]].begin();it !=
127 edp[to[i]].end();++it)
                     SEG::insert(cnt[*it],-1,1,0,N),--
128 cnt[*it],SEG::insert(cnt[*it],1,1,0,N);
129         if(son[p])
```

```
130                 dfs(son[p]);
131
132         for(register vector<int>::iterator it = edp[p].begin();it != edp[p].end();++it)
133             SEG::insert(cnt[*it],-1,1,0,N),++cnt[*it],SEG::insert(cnt[*it],1,1,0,N);
134             if(son[p])
135             {
136                 edp[p].swap(edp[son[p]]);
137
138                 for(register vector<int>::iterator it = edp[son[p]].begin();it != edp[son[p]].end();++it) edp[p].push_back(*it);
139                 vector<int>().swap(edp[son[p]]);
140             }
141             for(register int i = first[p];i;i = pre[i])
142                 if(to[i] ^ son[p])
143
144                     for(register vector<int>::iterator it = edp[to[i]].begin();it != edp[to[i]].end();++it)  edp[p].push_back(*it),
145
146
147 SEG::insert(cnt[*it],-1,1,0,N),++cnt[*it],SEG::insert(cnt[*it],1,1,0,N);
148         for(register vector< pair<int,int> >::iterator it = qry[p].begin();it != qry[p].end();++it) ans[it->second] = SEG::query(it->first,1,0,N);
149         }
150     }
151     int main()
152     {
153         scanf("%d%d%s",&n,&q,s + 1),m = strlen(s + 1);
154         for(register int i = 1;i <= n;++i)
155         {
156             scanf("%s",t),SAM::las = 1;
157             for(register char *c = t;*c;SAM::insert(*c++ - 'a',i));
158         }
159         SAM::build();
160         for(register int i = 1,x,p = 1,l = 0;i <= m;++i)
161         {
162             x = s[i] - 'a';
163             if(SAM::sam[p].ch[x])
164                 p = SAM::sam[p].ch[x],++l;
165             else
166             {
167                 for(;p && !SAM::sam[p].ch[x];p = SAM::sam[p].fa);
168                 !p ? (p = 1,l = 0) : (l = SAM::sam[p].len + 1,p = SAM::sam[p].ch[x]);
169             }
```

```
168             ed[i] = p,len[i] = l;
169         }
170         int l,r,k;
171         for(register int i = 1;i <= q;++i)
172             scanf("%d%d%d",&l,&r,&k),
173 
174          len[r] >= r - l + 1 && (SAM::qry[SAM::get(ed[r],r - l + 1)].push_back(make_pair(k,i)),1);SAM::dfs(1);
175         for(register int i = 1;i <= q;++i)
176             printf("%d\n",ans[i]);
177     }
178 
179 
```