2023.8.2 数据结构下

训练

1. Inversion Graph

【问题描述】

给定一个排列p1, p2, ..., pn，根据以下规则构建一个无向图：当i < j且pi > pj时，在顶点i和顶点j之间添加一条无向边。现在任务是计算该图中的连通分量数量。

两个顶点u和v属于同一个连通分量，当且仅当它们之间至少存在一条通过边相连的路径。

【链接】

https://www.luogu.com.cn/problem/CF1638C

【输入格式】

每个测试包含多个测试用例。第一行包含一个整数t（1 ⩽ t ⩽ 10^5），表示测试用例的数量。接下来是每个测试用例的描述。

每个测试用例的第一行包含一个整数n（1 ⩽ n ⩽ 10^5），表示排列的长度。

每个测试用例的第二行包含n个整数p1, p2, ..., pn（1 ⩽ pi ⩽ n），表示排列的元素。

所有测试用例中n的总和不超过2 × 10^5。

【输出格式】

对于每个测试用例，输出一个整数k，表示连通分量的数量。

【输入样例】

6

3

1 2 3

5

2 1 4 3 5

6

6 1 4 2 5 3

1

1

6

3 2 1 6 5 4

5

3 1 5 2 4

【输出样例】

3

3

1

1

2

1

【数据范围】

约束条件：

1 ⩽ t ⩽ 10^5

1 ⩽ n ⩽ 10^5

1 ⩽ pi ⩽ n

所有测试用例中n的总和不超过2 × 10^5。

题解

我们可以考虑独立的连通块是如何构成的。

我们发现每个连通块在原序列中是连续的。如果 p[i] 后有小于 p[i] 的元素，那么两数之间一定会构成一个逆序对，那么一个连通块就不可能以 p[i] 结尾。

反之，如果所有比 p[i] 小的都在 p[i] 之前，也就是说 1→p[i] （含 p[i] ）构成了 p[i] 的全排列，那么一个连通块一定以 p[i] 结尾。

在题目中，只要预处理 1 到当前位置的前缀和，如果这个前缀和等于 (i+1)×i/2，那么答案就加 1。

或者可以用单调栈来处理 思路类似。

代码

```
#include <bits/stdc++.h>using namespace std;
int T,n,a[100005],ans=0;
long long sum=0;
int main(){
        cin>>T;
        for(int i=1;i<=T;i++)
        {
                cin>>n;
                for(int i=1;i<=n;i++)
                {
                        cin>>a[i];sum+=a[i];//sum前缀和 if(sum==1LL*(1+i)*i/2)ans
                }
                cout<<ans<<endl;sum=0;ans=0;
        }
        return 0;
}
```

单调栈版本：

```
#include<bits/stdc++.h>

using namespace std;

constexpr int N = 1e6 + 5;

void solve() {
        int n;
        cin >> n;

        vector<int> p(n);
        for(int i = 0; i < n; ++i) {
                cin >> p[i];
        }
        reverse(p.begin(), p.end());

        stack<int> s;
        for(int i = 0; i < n; ++i) {
                if(s.empty()) {
                        s.push(p[i]);
                        continue;
                }
                if(s.top() < p[i]) {
```

```
                        int mn = 2e9;
                        while(!s.empty() && s.top() < p[i]) {
                                mn = min(mn, s.top());
                                s.pop();
                        }
                        s.push(mn);
                }
                else {
                        s.push(p[i]);
                }
        }

        cout << s.size() << "\n";
}

int main() {
        ios::sync_with_stdio(false);
        cin.tie(nullptr);

        int t;
        cin >> t;

        while(t--) {
                solve();
        }

        return 0;
}
```

2. Working routine

【问题描述】
Vasiliy终于开始工作，这里有一大堆等待他完成的任务。给定一个由n行m列组成的矩阵和q个任务。每个任务是交换给定矩阵中两个子矩阵。

对于每个任务，Vasiliy会知道六个整数ai, bi, ci, di, hi, wi，其中ai是第一个矩阵左上角的行索引，bi是第一个矩阵左上角的列索引，ci是第二个矩阵左上角的行索引，di是第二个矩阵左上角的列索引，hi是两个矩阵的高度，wi是两个矩阵的宽度。

保证在一个任务中，两个矩阵不会相交或相邻，即没有任何一个元素同时属于这两个矩阵，且不存在某两个元素分别属于两个矩阵且相邻（共边）。

Vasiliy想要知道在完成所有任务后，矩阵会是什么样子。

【链接】

https://www.luogu.com.cn/problem/CF706E

【输入格式】
第一行包含三个整数n, m, q，表示矩阵的行数、列数和任务数。
接下来n行，每行包含m个整数vi,j (1 <= vi,j <= 10^9) ，表示矩阵的初始值。
接下来q行，每行包含六个整数ai, bi, ci, di, hi, wi (1 <= ai, ci <= n，1 <= bi, di <= m，1 <= hi, wi <= m) ，表示交换的两个子矩阵的左上角坐标及其高度和宽度。

【输出格式】
输出n行，每行包含m个整数，表示完成q次交换后的矩阵。

【输入样例】
4 4 2
1 1 2 2
1 1 2 2
3 3 4 4
3 3 4 4
1 1 3 3 2 2
3 1 1 3 2 2

【输出样例】
4 4 3 3
4 4 3 3
2 2 1 1
2 2 1 1

【数据范围】
约束条件：
2 <= n, m <= 1000
1 <= q <= 10000
1 <= vi,j <= 10^9
1 <= ai, ci <= n
1 <= bi, di <= m
1 <= hi, wi <= m

题解

如果我们记录每个点右边和下边的是哪个点，就可以通过遍历得到整个矩形

如图，红色是修改的矩形，我们只需要修改绿色部分和蓝色部分的链表就好，思路很简单，就是细节很多。

代码

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<set>
#include<vector>
#include<ctime>
#define ll long long
#define pr(x) cerr<<#x<<"="<<x<<endl
#define id(a,b) ((a)*(m+1)+b)
int a[1100][1100],d[3010000],r[3010000],n,m,i,j,u1,u2,v1,v2,h,q,w,next,now,o,p,o
using namespace std;
int main()
{
 freopen("a.in","r",stdin);
 freopen("a.out","w",stdout);
 scanf("%d %d %d",&n,&m,&q);
 for (i=1;i<=n;i++)
        {
  for (j=1;j<=m;j++)
                {
   scanf("%d",&a[i][j]);
                }
        }
 for (i=0;i<=n;i++)
        {
  for (j=0;j<=m;j++)
                {
                        r[id(i,j)]=id(i,j+1);
                        d[id(i,j)]=id(i+1,j);
                }
        }
 for (i=1;i<=q;i++)
        {
  scanf("%d %d %d %d %d %d",&u1,&v1,&u2,&v2,&h,&w);
                o=p=0;
  for (j=1;j<u1;j++) o=d[o];
  for (j=1;j<v1;j++) o=r[o];
  for (j=1;j<u2;j++) p=d[p];
  for (j=1;j<v2;j++) p=r[p];
                oo=o,pp=p;
  for (j=1;j<=w;j++)        o=r[o],p=r[p],swap(d[o],d[p]);
  for (j=1;j<=h;j++)        o=d[o],p=d[p],swap(r[o],r[p]);
                o=oo,p=pp;
  for (j=1;j<=h;j++)        o=d[o],p=d[p],swap(r[o],r[p]);
  for (j=1;j<=w;j++)        o=r[o],p=r[p],swap(d[o],d[p]);
```

```
48          }
49  //for (i=0;i<=14;i++)        printf("r[%d]=%d d[%d]=%d\n",i,r[i],i,d[i]);
50  for (i=1;i<=n;i++)
51          {
52   for (j=0;j<=m;j++)
53                  {
54    if (j==0) now=id(i,0);
55    else
56                          {
57                                  now=r[now];
58     printf("%d ",a[now/(m+1)][now%(m+1)]);
59                          }
60                  }
61                  cout<<endl;
62          }
63 }
```

3. 最大数

【问题描述】

现在请求你维护一个数列，要求提供以下两种操作：

1、 查询操作。

语法：Q L

功能：查询当前数列中末尾L个数中的最大的数，并输出这个数的值。

限制：L不超过当前数列的长度。（L > 0）

2、 插入操作。

语法：A n

功能：将n加上t，其中t是最近一次查询操作的答案（如果还未执行过查询操作，则t = 0），并将所得结果对一个固定的常数D取模，将所得答案插入到数列的末尾。

限制：n是整数（可能为负数）并且在长整范围内。

注意：初始时数列是空的，没有一个数。

【输入格式】

第一行两个整数，M 和 D，其中 M 表示操作的个数，D 如上文中所述。

接下来的 M 行，每行一个字符串，描述一个具体的操作。语法如上文所述。

【输出格式】

对于每一个查询操作，你应该按照顺序依次输出结果，每个结果占一行。

【输入样例】

5 100

A 96

Q 1

A 97

Q 1

Q 2

【输出样例】

96

93

96

【说明/提示】

数据规模与约定：

对于全部的测试点，保证 1 ⩽ M ⩽ 2 × 10^5，1 ⩽ D ⩽ 2 × 10^9。

题解

我们发现，在需要维护区间最值的时候经常使用单调栈，尤其是这道题没有删除，并且操作只在序列末端。

维护一个单调递减栈，最后一个数最后加入，所以一定在栈内。而如果查询后两个的话，如果末尾最大显然好说，如果倒数第二个最大的话就可以查到这个最大的元素。

考虑具体开头/末尾 三个数/两个数的情况是一种很好的考虑单调栈和单调队列的做法，可以假设之间元素的大小关系然后确定做法。

查询使用二分，在单调栈的元素中记录编号，对答案进行二分，由于栈递减，所以查到的第一个编号在后x个的元素就是答案。

代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#define ll long long
using namespace std;
struct node
{
 ll id,v;
}st[200010];
ll t,md,m,o,ans,l,r,mid,n,lastans;
int main ()
{
 scanf("%lld %lld",&m,&md);
 while (m--)
 {
  char ch[10];
  scanf("%s",ch);
  if (ch[0]=='A')
  {
   n++;
   scanf("%lld",&o);
   o=(o%md+lastans)%md;
   while (o>st[t].v&&t) t--;
   st[++t].v=o;
   st[t].id=n;
  }
  else
  {
   l=1,r=t,ans=1;
   scanf("%lld",&o);
   while (l<=r)
   {
    mid=(l+r)>>1;
```

```
36      if (st[mid].id>=n-o+1)
37      {
38       r=mid-1,ans=mid;
39      }
40      else l=mid+1;
41     }
42     printf("%lld\n",st[ans].v);
43     lastans=st[ans].v;
44    }
45   }
46 }
```

4. Stamp Rally

【问题描述】

我们有一个无向图，有N个顶点和M条边。顶点编号为1到N，边编号为1到M。第i条边连接着顶点ai和bi。该图是连通的。

在这个图上，有Q对兄弟参加了名为“Stamp Rally”的活动。第i对兄弟的Stamp Rally规则如下：

一个兄弟从顶点xi开始，另一个从顶点yi开始。

两个兄弟沿着边探索图，总共访问zi个顶点，包括起始顶点。在这里，即使一个顶点被多次访问，或者被两个兄弟访问，也只计数一次。

得分定义为两者访问的边的最大编号。他们的目标是最小化这个值。

找出每一对兄弟的最小可能得分。

【链接】

https://www.luogu.com.cn/problem/AT_agc002_d

【输入格式】

输入格式如下：

N M

a1 b1

a2 b2

...

aM bM

Q

x1 y1 z1

x2 y2 z2

...

xQ yQ zQ

【输出格式】

输出Q行。第i行应包含第i对兄弟的最小可能得分。

【输入样例】

5 6

2 3

4 5

1 2

1 3

1 4

1 5

6

2 4 3

2 4 4

2 4 5

1 3 3

1 3 4

1 3 5

【输出样例】

1

2

3

1

5

5

【数据范围】

约束条件：

3≤N≤10^5

N−1≤M≤10^5

1≤ai<bi≤N

给定的图是连通的。

1≤Q≤10^5

1≤xj<yj≤N

3≤zj≤N

输入为整数。

题解

这个题我们可以考虑二分。

对于询问(x,y,z)，二分k为答案，暴力做法是加入编号为1-k的边，判断x，y是否在同一个联通块内，如果是，则可覆盖点数为sz[x所在联通块点数]（记为sz[x]），如果不是，则可覆盖点数为sz[x]+sz[y]。

然后看到询问特别多。em我们就可以考虑整体二分。(二分答案k，判断有多少个询问的答案小于等于它）。至于判断x,y是否处于同一联通块和求联通块大小，可以采用并查集。并查集可以按秩合并，这样撤销比较方便。

代码

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
using namespace std;
int n,m,Q,x[200010],y[200010],xx,yy,z;
struct ASK{int x,y,v,id;
}q[200010],re[200010];
int ans[200010],t[2][200010];ASK a[200010],b[200010];
int sz[200010],fa[200010];
int get_fa(int x){return fa[x]==x?x:get_fa(fa[x]);}
void solve(int l,int r,int ansl,int ansr)
{
    int _x,_y;
    if (ansl==ansr)
    {
```

```
        for (int i=l;i<=r;i++) ans[q[i].id]=ansl;
        _x=get_fa(x[ansl]);_y=get_fa(y[ansl]);
        if (_x!=_y)
        {
            if (sz[_x]>sz[_y]) swap(_x,_y);
            fa[_x]=_y;sz[_y]+=sz[_x];
        }
        return;
    }
    int ansmid=ansl+ansr>>1,js1=0,js2=0,top=0;
    for (int i=ansl;i<=ansmid;i++)
    {
        _x=get_fa(x[i]);_y=get_fa(y[i]);
        if (_x!=_y)
        {
            if (sz[_x]>sz[_y]) swap(_x,_y);
            fa[_x]=_y;sz[_y]+=sz[_x];
            t[0][++top]=_x;t[1][top]=_y;
        }
    }
    for (int i=l;i<=r;i++)
    {
        _x=get_fa(q[i].x);_y=get_fa(q[i].y);
        if ((_y!=_x&&sz[_x]+sz[_y]>=q[i].v)||(_y==_x&&sz[_x]>=q[i].v)) a[++js1]=
        else b[++js2]=q[i];
    }
    for (;top;top--)
    {
        fa[t[0][top]]=t[0][top];sz[t[1][top]]-=sz[t[0][top]];
    }
    for (int i=1;i<=js1;i++) q[i+l-1]=a[i];
    for (int i=1;i<=js2;i++) q[i+l+js1-1]=b[i];
    solve(l,l+js1-1,ansl,ansmid);
    solve(l+js1,r,ansmid+1,ansr);
}
int main()
{
    scanf("%d%d",&n,&m);
    for (int i=1;i<=m;i++)    scanf("%d%d",&x[i],&y[i]);
    scanf("%d",&Q);
    for (int i=1;i<=Q;i++)
    {
        scanf("%d%d%d",&xx,&yy,&z);
        q[i]=ASK{xx,yy,z,i};
    }
    for (int i=1;i<=n;i++) fa[i]=i,sz[i]=1;
    solve(1,Q,1,m);
```

```
64      for (int i=1;i<=Q;i++) printf("%d\n",ans[i]);
65  }
```

5. Nikitosh and xor

【问题描述】

给出一个序列，求出两个不相交的子区间a[l1-r1],a[l2-r2]，使得两个区间异或和之和最大。

【链接】https://vjudge.csgrandeur.cn/problem/CodeChef-REBXOR

【输入格式】

第一行输入序列长度n

接下来一行输入序列

【输出格式】

输出最大的和。

【样例输入】

5

1 2 3 1 2

【样例输出】

6

题解

考虑简化问题，如何求出异或和最大的区间?

区间异或和可以表示成前缀异或和异或的形式。所以对于每个右端点，我们要找到一个左端点使得异或和最大，就是对于一个数，找到一个数和它异或最大。

可以用trie树维护。

先查询，再把新的数字插入。

这样就可以求出每一部分的最大的区间异或和。

考虑合并答案，只要枚举中间点，用左边的最大值和右边的最大值合并更新答案即可。

代码

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<cstdlib>
#include<vector>
#include<ctime>
#define ll long long
#define pr(x) cerr<<#x<<"="<<x<<endl
using namespace std;
#define N 100010
int tr[4000010][2],cnt,bin[40],a[N],n,i,ans,ansl[N],ansr[N],suml[N],sumr[N];
void insert(int x)
{
 int now=1;
 for (int i=30;i>=1;i--)
        {
  int b=((x>>(i-1))&1);
  if (tr[now][b]) now=tr[now][b];
  else tr[now][b]=++cnt,now=cnt;
        }
}
int query(int x)
{
 int now=1,ret=0;
 for (int i=30;i>=1;i--)
        {
  int b=((x>>(i-1))&1);
  if (tr[now][b^1]) now=tr[now][b^1],ret+=bin[i];
  else now=tr[now][b];
        }
 return ret;
}
int main()
{
 freopen("a.in","r",stdin);
 freopen("a.out","w",stdout);
 scanf("%d",&n);
        cnt=1;
        bin[1]=1;for (i=2;i<=33;i++) bin[i]=bin[i-1]<<1;
 for (i=1;i<=n;i++)
        {
  scanf("%d",&a[i]);
        }
 for (i=1;i<=n;i++) suml[i]=suml[i-1]^a[i];
 for (i=n;i>=1;i--) sumr[i]=sumr[i+1]^a[i];
```

```
 insert(a[1]);
        ansl[1]=a[1];
 for (i=2;i<=n;i++)
        {
                ansl[i]=max(ansl[i-1],max(suml[i],query(suml[i])));
  insert(suml[i]);
        }
 memset(tr,0,sizeof(tr));
        cnt=1;
 insert(a[n]);
        ansr[n]=a[n];
 for (i=n-1;i>=1;i--)
        {
                ansr[i]=max(ansr[i+1],max(sumr[i],query(sumr[i])));
  insert(sumr[i]);
        }
 for (i=n;i>=1;i--) ansl[i]=ansl[i-1];
 for (i=1;i<=n;i++) ans=max(ans,ansl[i]+ansr[i]);
 printf("%d\n",ans);
 return 0;
}
```