

目录

785. 快速排序	3
787. 归并排序	4
788. 归并排序求逆序对	5
789. 数的范围(二分)	6
790. 数的三次方根(浮点数二分)	8
4177. 曲线(三分)	9
791. 高精度加法	11
792. 高精度减法	11
793. 高精度乘法	12
794. 高精度除法	13
802. 区间和(离散化)	13
803. 区间合并(多个区间合并, 贪心)	16
3302. 表达式求值	17
830. 单调栈(单调递增栈)	19
154. 滑动窗口(单调队列)	20
831. KMP 字符串	22
835. Trie 字符串统计	24
143. 最大异或对(trie 字典树存二进制)	25
836. 合并集合(并查集基本操作)	27
837. 连通块中点的数量(并查集基本操作+集合数量查询)	28
240. 食物链(带权并查集)	30
838. 堆排序(数组堆排序)	32
840. 模拟散列表(链地址法和开放地址法)	33
841. 字符串哈希	36
842. 排列数字(dfs 全排列)	37
843. n-皇后问题(dfs)	39
844. 走迷宫(基本 bfs, 距离到距离 bfs)	41
845. 八数码(状态到状态 bfs)	42
846. 树的重心(dfs, 树与图的深度优先遍历)	45
847. 图中点的层次(bfs, 树与图的广度优先遍历)	47
848. 有向图的拓扑序列	48
849. Dijkstra 求最短路 I(朴素版)[必正权边]	50
850. Dijkstra 求最短路 II(堆优化版)[必正权边]	51
853. 有边数限制的最短路(bellman-ford)[正负权边均可]	53
851. spfa 求最短路[正负权边均可]	55
852. spfa 判断负环[正负权边均可]	57
854. Floyd 求最短路(多源最短路)[正负权边均可]	59
858. Prim 算法求最小生成树	61
859. Kruskal 算法求最小生成树	62
860. 染色法判定二分图	64
861. 二分图的最大匹配(匈牙利算法)	66
866. 试除法判定质数	68
867. 分解质因数	69

868. 筛质数	70
869. 试除法求约数	72
870. 约数个数	73
871. 约数之和	75
872. 最大公约数	76
873. 欧拉函数	76
874. 筛法求欧拉函数	77
875. 快速幂	78
876. 快速幂求逆元	80
877. 扩展欧几里得算法	81
878. 线性同余方程(扩展欧几里得算法)	82
204. 表达整数的奇怪方式(扩展中国剩余定理 EXCRT)	84
883. 高斯消元解线性方程组	86
884. 高斯消元解异或线性方程组	88
885. 求组合数 I(递推公式初始化组合数答案数组)	91
886. 求组合数 II(初始化阶乘和其逆元数组+组合公式求解)	92
887. 求组合数 III(lucas 定理)	93
888. 求组合数 IV(高精度+计算质因子次数)	95
889. 满足条件的 01 序列(卡特兰数+求组合数)	97
890. 能被整除的数(容斥原理)	98
Nim 游戏相关理论	99
891. Nim 游戏(经典 Nim 游戏)	101
892. 台阶-Nim 游戏	102
893. 集合-Nim 游戏	103
894. 拆分-Nim 游戏	106
2. 01 背包问题	107
3. 完全背包问题	109
4. 多重背包问题 I	111
5. 多重背包问题 II(二进制优化)	112
9. 分组背包问题	114
241. 楼兰图腾(树状数组)	115
1057. 股票买卖 IV(dp-状态机模型- 2 状态)	117
1058. 股票买卖 V(dp-状态机模型- 3 状态)	119
91. 最短 Hamilton 路径(dp-状压 dp)	121
285. 没有上司的舞会(树形 dp,打家劫舍算法)	122
245. 你能回答这些问题吗(线段树,连续子区间和最大,单点修改)	125
3165. 第一类斯特林数--圆排序	127
3166. 第二类斯特林数--子集	130
1270. 数列区间最大值(st 表)	132
有向图 dfs 判环	133
1819. 序列中不同最大公约数的数目	134
365. 水壶问题(裴蜀定理)	135
P1452 求凸包+旋转卡壳	136
nlogn 求 1-n 范围的每个数的所有质因子	139

243. 一个简单的整数问题 2(线段树,懒标记).....	140
255. 第 K 小数(主席树).....	143
1169. 糖果(差分约束).....	147
取不相邻元素的方法总数(组合数).....	150
3164. 线性基(任意异或和最大).....	151
210. 异或运算(线性基,任意异或和第 k 小).....	152
线段树板子集合	155

785. 快速排序

给定你一个长度为 n 的整数数列。

请你使用快速排序对这个数列按照从小到大进行排序。

并将排好序的数列按顺序输出。

输入格式

输入共两行，第一行包含整数 n 。

第二行包含 n 个整数（所有整数均在 $1 \sim 10^9$ 范围内），表示整个数列。

输出格式

输出共一行，包含 n 个整数，表示排好序的数列。

数据范围

$1 \leq n \leq 100000$

输入样例：

```
5
3 1 2 4 5
```

输出样例：

```
1 2 3 4 5
```

```
#include<iostream>
using namespace std;
int n,a[100010];
void quick_sort(int a[],int l,int r){
    if(l>=r)return;
    int mid=l+r>>1,x=a[mid],i=l-1,j=r+1;
    while(i<j){
```

```

        do i++;while(a[i]<x);
        do j--;while(a[j]>x);
        if(i<j)swap(a[i],a[j]);
    }
    quick_sort(a,l,j),quick_sort(a,j+1,r);
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    quick_sort(a,1,n);
    for(int i=1;i<=n;i++)cout<<a[i]<<" ";
}

```

787. 归并排序

给定你一个长度为 n 的整数数列。

请你使用归并排序对这个数列按照从小到大进行排序。

并将排好序的数列按顺序输出。

输入格式

输入共两行，第一行包含整数 n 。

第二行包含 n 个整数（所有整数均在 $1 \sim 10^9$ 范围内），表示整个数列。

输出格式

输出共一行，包含 n 个整数，表示排好序的数列。

数据范围

$1 \leq n \leq 100000$

输入样例：

```

5
3 1 2 4 5

```

输出样例：

```

1 2 3 4 5

```

```

#include<iostream>
using namespace std;
int n,a[100010],temp[100010];

```

```

void merge_sort(int a[],int l,int r){
    if(l>=r)return;
    int mid=l+r>>1;
    merge_sort(a,l,mid),merge_sort(a,mid+1,r);
    int i=l,j=mid+1,k=0;
    while(i<=mid&&j<=r){
        if(a[i]<a[j])temp[k++]=a[i++];
        else temp[k++]=a[j++];
    }
    while(i<=mid)temp[k++]=a[i++];
    while(j<=r)temp[k++]=a[j++];
    for(int i=l,k=0;i<=r;i++)a[i]=temp[k++];
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    merge_sort(a,1,n);
    for(int i=1;i<=n;i++)cout<<a[i]<<" ";
    return 0;
}

```

788. 归并排序求逆序对

给定一个长度为 n 的整数数列，请你计算数列中的逆序对的数量。

逆序对的定义如下：对于数列的第 i 个和第 j 个元素，如果满足 $i < j$ 且 $a[i] > a[j]$ ，则其为一个逆序对；否则不是。

输入格式

第一行包含整数 n ，表示数列的长度。

第二行包含 n 个整数，表示整个数列。

输出格式

输出一个整数，表示逆序对的个数。

数据范围

$1 \leq n \leq 100000$,

数列中的元素的取值范围 $[1, 10^9]$ 。

输入样例：

```
6
```

```
2 3 4 5 6 1
```

输出样例:

```
5
```

```
#include<iostream>
using namespace std;
int a[100100],temp[100100],n;
long long cnt=0;
void merge_sort(int a[],int l,int r){
    if(l>=r)return;
    int i=l,mid=l+r>>1,j=mid+1,k=0;
    merge_sort(a,l,mid),merge_sort(a,j,r);
    while(i<=mid&&j<=r){
        if(a[j]<a[i]){
            temp[k++]=a[j++];
            cnt=cnt+(mid-i+1);
        }else temp[k++]=a[i++];
    }
    while(i<=mid)temp[k++]=a[i++];
    while(j<=r)temp[k++]=a[j++];
    for(int i=l,k=0;i<=r;i++)a[i]=temp[k++];
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    merge_sort(a,1,n);
    cout<<cnt<<endl;
    return 0;
}
```

789. 数的范围(二分)

给定一个按照升序排列的长度为 n 的整数数组，以及 q 个查询。

对于每个查询，返回一个元素 k 的起始位置和终止位置（位置从 0 开始计数）。

如果数组中不存在该元素，则返回 `-1 -1`。

输入格式

第一行包含整数 n 和 q ，表示数组长度和询问个数。

第二行包含 n 个整数（均在 $1 \sim 10000$ 范围内），表示完整数组。

接下来 q 行，每行包含一个整数 k ，表示一个询问元素。

输出格式

共 q 行，每行包含两个整数，表示所求元素的起始位置和终止位置。

如果数组中不存在该元素，则返回 `-1 -1`。

数据范围

$1 \leq n \leq 100000$

$1 \leq q \leq 10000$

$1 \leq k \leq 10000$

输入样例：

```
6 3
1 2 2 3 3 4
3
4
5
```

输出样例：

```
3 4
5 5
-1 -1
```

```
#include<iostream>
using namespace std;
int a[100010],n,q,ll,rr;
int main(){
    cin>>n>>q;
    for(int i=1;i<=n;i++)cin>>a[i];
    while(q--){
        int x;
        cin>>x;
        int l=1,r=n+1;
        while(l+1<r){
            int mid=l+r>>1;
            if(x<=a[mid])r=mid;
            else l=mid;
        }
        ll=r;
    }
}
```

```

        l=l-1,r=n+1;
        while(l+1<r){
            int mid=l+r>>1;
            if(x>=a[mid])l=mid;
            else r=mid;
        }
        rr=l;
        if(a[l1]!=x)cout<<-1<<" "<<-1<<endl;
        else cout<<l1-1<<" "<<rr-1<<endl;
    }
    return 0;
}

```

790. 数的三次方根(浮点数二分)

给定一个浮点数 n ，求它的三次方根。

输入格式

共一行，包含一个浮点数 n 。

输出格式

共一行，包含一个浮点数，表示问题的解。

注意，结果保留 6 位小数。

数据范围

$-10000 \leq n \leq 10000$

输入样例：

```
1000.00
```

输出样例：

```
10.000000
```

```

#include<iostream>
using namespace std;
double n;
int main(){
    cin>>n;
    double l=-10001,r=10001;
    while(l+1e-8<r){
        double mid=(l+r)/2;

```



```

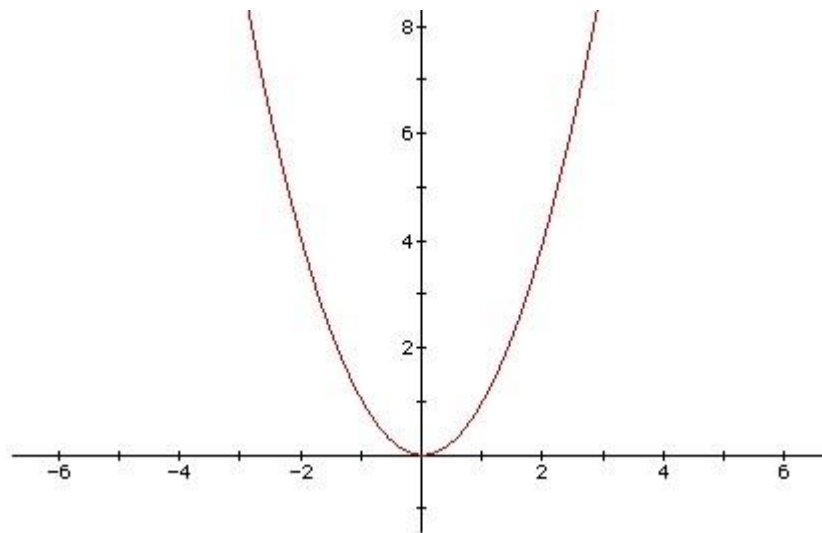
        if(mid*mid*mid>=n)r=mid;
        else l=mid;
    }
    printf("%f",r);
    return 0;
}

```

4177. 曲线(三分)

提示:C 语言%f,%lf 自动四舍五入。三分思想：分成三部分(l,ml,mr,r)，不断舍去极值点不在的一部分。

明明做作业的时候遇到了 n 个二次函数 $S_i(x)=ax^2+bx+c$ ，他突发奇想设计了一个新的函数 $F(x)=\max\{S_i(x)\}, i=1\dots n$ 。



明明现在想求这个函数在 $[0,1000]$ 的最小值，要求精确到小数点后四位，四舍五入。

输入格式

输入包含 T 组数据，每组第一行一个整数 n ；

接下来 n 行，每行 3 个整数 a,b,c ，用来表示每个二次函数的 3 个系数。注意：二次函数有可能退化成一个一次。

输出格式

每组数据输出一行，表示新函数 $F(x)$ 的在区间 $[0,1000]$ 上的最小值。精确到小数点后四位，四舍五入。

数据范围

$1 \leq T \leq 10$,

$1 \leq n \leq 105$,

$0 \leq a \leq 100$,

$0 \leq |b|, |c| \leq 5000$ 。

输入样例：

```
2
1
2 0 0
2
2 0 0
2 -4 2
```

输出样例：

```
0.0000
0.5000
```

```
#include<iostream>
#include<cmath>
using namespace std;
int t,n,a[100010],b[100010],c[100010];
double Fx(int a[],int b[],int c[],double x){
    double maxx=-1e9;
    for(int i=1;i<=n;i++)maxx=max(maxx,a[i]*x*x+b[i]*x+c[i]);
    return maxx;
}
double ans(int a[],int b[],int c[]){
    double l=0,r=1000;
    while(l+1e-9<r){
        double m=(r-l)/3;
        double ml=l+m,mr=r-m;
        if(Fx(a,b,c,ml)>Fx(a,b,c,mr))l=ml;
        else r=mr;
    }
    return Fx(a,b,c,l);
}
int main(){
    cin>>t;
    while(t--){
        cin>>n;
        for(int i=1;i<=n;i++){
```

```

        cin>>a[i]>>b[i]>>c[i];
    }
    printf("%.4lf\n",ans(a,b,c));
}
}

```

791. 高精度加法

```

#include<iostream>
#include<vector>
#include<cstring>
using namespace std;
vector<int> add(vector<int>&A,vector<int>&B){
    vector<int>C;
    int t=0;
    for(int i=0;i<A.size()||i<B.size();i++){
        if(i<A.size())t+=A[i];
        if(i<B.size())t+=B[i];
        C.push_back(t%10);
        t/=10;
    }
    if(t)C.push_back(t);
    return C;
}
int main(){
    vector<int>A,B,C;
    string a,b;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    for(int i=b.size()-1;i>=0;i--)B.push_back(b[i]-'0');
    C=add(A,B);
    for(int i=C.size()-1;i>=0;i--)cout<<C[i];
    return 0;
}

```

792. 高精度减法

```

#include<iostream>
#include<vector>
#include<cstring>
using namespace std;
vector<int> reduce(vector<int>&A,vector<int>&B){
    vector<int>C;

```

```

int t=0;
for(int i=0;i<A.size();i++){
    t+=A[i];
    if(i<B.size())t=t-B[i];
    C.push_back((t+10)%10);
    if(t<0)t=-1;
    else t=0;
}
while(C.size()>1&&C.back()==0)C.pop_back();
return C;
}
int main(){
    vector<int>A,B,C;
    bool reverse=false;
    string a,b;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    for(int i=b.size()-1;i>=0;i--)B.push_back(b[i]-'0');
    if(a.size()>b.size()){
        C=reduce(A,B);
    }else if(a.size()<b.size()){
        C=reduce(B,A),reverse=true;
    }else if(a>=b){
        C=reduce(A,B);
    }else{
        C=reduce(B,A),reverse=true;
    }
    if(reverse==true)cout<<"-";
    for(int i=C.size()-1;i>=0;i--)cout<<C[i];
    return 0;
}

```

793. 高精度乘法

```

#include<bits/stdc++.h>
using namespace std;
vector<int>mul(vector<int>&A,int b){
    vector<int>C;
    for(int i=0,t=0;i<A.size()||t!=0;i++){
        if(i<A.size())t+=A[i]*b;
        C.push_back(t%10);
        t/=10;
    }
    while(C.size()>1&&C.back()==0)C.pop_back();
}

```

```

        return C;
    }
    int main(){
        int b;
        string a;
        cin>>a>>b;
        vector<int>A,C;
        for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
        C=mul(A,b);
        for(int i=C.size()-1;i>=0;i--)cout<<C[i];
    }

```

794. 高精度除法

```

#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
vector<int>div(vector<int>&A,int b,int &r){
    vector<int>C;
    for(int i=A.size()-1;i>=0;i--){
        r=r*10+A[i];
        C.push_back(r/b);
        r%=b;
    }
    reverse(C.begin(),C.end());
    while(C.size()>1&&C.back()==0)C.pop_back();
    return C;
}
int main(){
    int b,r=0;
    vector<int>A;
    string a;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    auto C=div(A,b,r);
    for(int i=C.size()-1;i>=0;i--)cout<<C[i];
    cout<<endl<<r;
}

```

802. 区间和(离散化)

假定有一个无限长的数轴，数轴上每个坐标上的数都是 0。

现在，我们首先进行 n 次操作，每次操作将某一位置 x 上的数加 c 。

接下来，进行 m 次询问，每个询问包含两个整数 l 和 r ，你需要求出在区间 $[l,r]$ 之间的所有数的和。

输入格式

第一行包含两个整数 n 和 m 。

接下来 n 行，每行包含两个整数 x 和 c 。

再接下来 m 行，每行包含两个整数 l 和 r 。

输出格式

共 m 行，每行输出一个询问中所求的区间内数字和。

数据范围

$-10^9 \leq x \leq 10^9$,

$1 \leq n, m \leq 10^5$,

$-10^9 \leq l \leq r \leq 10^9$,

$-10000 \leq c \leq 10000$

输入样例：

```
3 3
1 2
3 6
7 5
1 3
4 6
7 8
```

输出样例：

```
8
0
5
```

```
#include<iostream>
#include<vector>
#include<algorithm>
```

```

using namespace std;
typedef pair<int,int> PII;
int n,m,s[300010],a[300010];
vector<int>alls;
vector<PII>add,query;
int find(int x){
    int l=-1,r=alls.size();
    while(l+1!=r){
        int mid=l+r>>1;
        if(alls[mid]<=x)l=mid;
        else r=mid;
    }
    return l+1;
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        int x,y;
        cin>>x>>y;
        add.push_back({x,y});
        alls.push_back(x);
    }
    for(int i=1;i<=m;i++){
        int x,y;
        cin>>x>>y;
        query.push_back({x,y});
        alls.push_back(x);
        alls.push_back(y);
    }
    sort(alls.begin(),alls.end());
    alls.erase(unique(alls.begin(),alls.end()),alls.end());
    for(auto i:add){
        a[find(i.first)]+=i.second;
    }
    for(int i=1;i<=alls.size();i++){
        s[i]=s[i-1]+a[i];
    }
    for(auto i:query){
        cout<<s[find(i.second)]-s[find(i.first)-1]<<endl;
    }
}

```

803. 区间合并(多个区间合并，贪心)

给定 n 个区间 $[l_i, r_i]$ ，要求合并所有有交集的区间。

注意如果在端点处相交，也算有交集。

输出合并完成后的区间个数。

例如： $[1, 3]$ 和 $[2, 6]$ 可以合并为一个区间 $[1, 6]$ 。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含两个整数 l 和 r 。

输出格式

共一行，包含一个整数，表示合并区间完成后的区间个数。

数据范围

$1 \leq n \leq 100000$,

$-10^9 \leq l_i \leq r_i \leq 10^9$

输入样例：

```
5
1 2
2 4
5 6
7 8
7 9
```

输出样例：

```
3
```

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
typedef pair<int,int>PII;
vector<PII>interval,res;
int main(){
```



```

int n;
cin>>n;
for(int i=1;i<=n;i++){
    int a,b;
    cin>>a>>b;
    interval.push_back({a,b});
}
sort(interval.begin(),interval.end());
int l=interval[0].first,r=interval[0].second;
for(int i=1;i<interval.size();i++){
    int l1=interval[i].first,r1=interval[i].second;
    if(l1<=r&&r1>r){
        r=r1;
    }else if(l1>r){
        res.push_back({l,r});
        l=l1,r=r1;
    }
}
res.push_back({l,r});
cout<<res.size()<<endl;
return 0;
}

```

3302. 表达式求值

给定一个表达式，其中运算符仅包含 `+, -, *, /`（加 减 乘 整除），可能包含括号，请你求出表达式的最终值。

注意：

- 数据保证给定的表达式合法。
- 题目保证符号 `-` 只作为减号出现，不作为负号出现，例如，`-1+2`, `(2+2)*(-(1+1)+2)` 之类表达式均不会出现。
- 题目保证表达式中所有数字均为正整数。
- 题目保证表达式在中间计算过程以及结果中，均不超过 $2^{31}-1$ 。
- 题目中的整除是指向 0 取整，也就是说对于大于 0 的结果向下取整，例如 $5/3=1$ ，对于小于 0 的结果向上取整，例如 $5/(1-4)=-1$ 。
- C++ 和 Java 中的整除默认是向零取整；Python 中的整除 `//` 默认向下取整，因此 Python 的 `eval()` 函数中的整除也是向下取整，在本题中不能直接使用。

输入格式

共一行，为给定表达式。

输出格式

共一行，为表达式的结果。

数据范围

表达式的长度不超过 10^5 。

输入样例：

```
(2+2)*(1+1)
```

输出样例：

```
8
```

```
#include<iostream>
#include<stack>
#include<string>
#include<unordered_map>
using namespace std;
stack<int>num;
stack<char>op;
string s;
unordered_map<char,int>h{{'+',1},{ '-',1},{ '*',2},{ '/',2}};
void eval(){
    int b=num.top();
    num.pop();
    int a=num.top();
    num.pop();
    char c=op.top();
    op.pop();
    if(c=='+')num.push(a+b);
    if(c=='-')num.push(a-b);
    if(c=='*')num.push(a*b);
    if(c=='/')num.push(a/b);
}
int main(){
    cin>>s;
    for(int i=0;i<s.size();i++){
        char c=s[i];
        if(isdigit(c)){
            int j=i,res=0;
            while(isdigit(s[j])){
                res=res*10+s[j]-'0';
                j++;
            }
            i=j-1;
        }
    }
}
```

```

        num.push(res);
    }else if(c=='('){
        op.push(c);
    }else if(c==')'){
        while(op.top()!='(')eval();
        op.pop();
    }else{
        while(!op.empty() && h[op.top()]>=h[c])eval();
        op.push(c);
    }
}
while(!op.empty())eval();
cout<<num.top()<<endl;
}

```

830. 单调栈(单调递增栈)

给定一个长度为 N 的整数数列，输出每个数左边第一个比它小的数，如果不存在则输出 -1 。

输入格式

第一行包含整数 N ，表示数列长度。

第二行包含 N 个整数，表示整数数列。

输出格式

共一行，包含 N 个整数，其中第 i 个数表示第 i 个数的左边第一个比它小的数，如果不存在则输出 -1 。

数据范围

$1 \leq N \leq 10^5$

$1 \leq \text{数列中元素} \leq 10^9$

输入样例：

```

5
3 4 2 7 5

```

输出样例：

```

-1 3 -1 2 2

```

```

#include<iostream>
using namespace std;
const int N=100010;

```

```

int stack[N],tt,n,x;
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    cin>>n;
    while(n--){
        cin>>x;
        while(tt&&stack[tt]>=x)tt--;
        stack[++tt]=x;
        if(tt-1>=1)cout<<stack[tt-1]<<" ";
        else cout<<-1<<" ";
    }
    return 0;
}

```

154. 滑动窗口(单调队列)

给定一个大小为 $n \leq 10^6$ 的数组。

有一个大小为 k 的滑动窗口，它从数组的最左边移动到最右边。

你只能在窗口中看到 k 个数字。

每次滑动窗口向右移动一个位置。

以下是一个例子：

该数组为 `[1 3 -1 -3 5 3 6 7]`， k 为 3。

窗口位置	最小值	最大值
[1 3 -1] -3 5 3 6 7	-1	3
1 [3 -1 -3] 5 3 6 7	-3	3
1 3 [-1 -3 5] 3 6 7	-3	5
1 3 -1 [-3 5 3] 6 7	-3	5
1 3 -1 -3 [5 3 6] 7	3	6
1 3 -1 -3 5 [3 6 7]	3	7

你的任务是确定滑动窗口位于每个位置时，窗口中的最大值和最小值。

输入格式

输入包含两行。

第一行包含两个整数 n 和 k ，分别代表数组长度和滑动窗口的长度。

第二行有 n 个整数，代表数组的具体数值。

同行数据之间用空格隔开。

输出格式

输出包含两个。

第一行输出，从左至右，每个位置滑动窗口中的最小值。

第二行输出，从左至右，每个位置滑动窗口中的最大值。

输入样例：

```
8 3
1 3 -1 -3 5 3 6 7
```

输出样例：

```
-1 -3 -3 -3 3 3
3 3 5 5 6 7
```

```
#include<iostream>
using namespace std;
const int N=1000010;
int a[N],hh,tt=-1,n,k,q[N];
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    cin>>n>>k;
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<=n;i++){
        while(tt>=hh&& a[q[tt]]>=a[i])tt--;
        q[++tt]=i;
        if(i>=k)cout<<a[q[hh]]<<" ";
        if(i-k+1==q[hh]){
            hh++;
        }
    }
    cout<<endl;
    hh=0,tt=-1;
    for(int i=1;i<=n;i++){
        while(tt>=hh&& a[q[tt]]<=a[i])tt--;
        q[++tt]=i;
        if(i>=k)cout<<a[q[hh]]<<" ";
        if(i-k+1==q[hh]){
```

```

        hh++;
    }
}
return 0;
}

```

831. KMP 字符串

给定一个字符串 S ，以及一个模式串 P ，所有字符串中只包含大小写英文字母以及阿拉伯数字。

模式串 P 在字符串 S 中多次作为子串出现。

求出模式串 P 在字符串 S 中所有出现的位置的起始下标。

输入格式

第一行输入整数 N ，表示字符串 P 的长度。

第二行输入字符串 P 。

第三行输入整数 M ，表示字符串 S 的长度。

第四行输入字符串 S 。

输出格式

共一行，输出所有出现位置的起始下标（下标从 0 开始计数），整数之间用空格隔开。

数据范围

$1 \leq N \leq 10^5$

$1 \leq M \leq 10^6$

输入样例：

```

3
aba
5
ababa

```

输出样例：

```

0 2

```

第一种，匹配下标从 0 开始， $ne[0]=-1$ 与第二种思路一样，都减一，因为比较是从 0 开始的，类如数组从 0 开始存储和从 1 开始存储的区别，都向前挫一位。

```
#include<iostream>
```

```

using namespace std;
char str[100010],str2[1000010];
int n,m,ne[100010];
int main(){
    cin>>n>>str>>m>>str2;
    ne[0]=-1;
    for(int i=1,j=-1;i<n;i++){
        while(j!=-1&&str[j+1]!=str[i])j=ne[j];
        if(str[j+1]==str[i])j++;
        ne[i]=j;
    }
    for(int i=0,j=-1;i<m;i++){
        while(j!=-1&&str[j+1]!=str2[i])j=ne[j];
        if(str[j+1]==str2[i])j++;
        if(j==n-1){
            cout<<i-j<<" ";
            j=ne[j];
        }
    }
}

```

第二种，匹配下标从 1 开始， $ne[1]=0$ $ne[i]=j$ 表示 $1\sim j$ 和 $i-j+1\sim i$ 的字符串相等， $j+1$ 位是比
较位， j 位用于 kmp 移动位置后比较位与主串比较

```

#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
char p[100010],s[1000010];
int n,m;
int ne[100010];
int main(){
    cin>>n>>p+1>>m>>s+1;
    for(int i=2,j=0;i<=n;i++){
        while(j&&p[i]!=p[j+1])j=ne[j];
        if(p[i]==p[j+1])j++;
        ne[i]=j;
    }
    for(int i=1,j=0;i<=m;i++){
        while(j&&s[i]!=p[j+1])j=ne[j];
        if(s[i]==p[j+1])j++;
        if(j==n){
            cout<<i-j<<" ";
            j=ne[j];
        }
    }
}

```

```
    return 0;  
}
```

835. Trie 字符串统计

维护一个字符串集合，支持两种操作：

1. **I x** 向集合中插入一个字符串 x ；
2. **Q x** 询问一个字符串在集合中出现了多少次。

共有 N 个操作，所有输入的字符串总长度不超过 10^5 ，字符串仅包含小写英文字母。

输入格式

第一行包含整数 N ，表示操作数。

接下来 N 行，每行包含一个操作指令，指令为 **I x** 或 **Q x** 中的一种。

输出格式

对于每个询问指令 **Q x**，都要输出一个整数作为结果，表示 x 在集合中出现的次数。

每个结果占一行。

数据范围

$1 \leq N \leq 2 \times 10^4$

输入样例：

```
5  
  
I abc  
  
Q abc  
  
Q ab  
  
I ab  
  
Q ab
```

输出样例：

```
1  
  
0  
  
1
```

```
#include<iostream>
```



```

#include<cstring>
const int N=100010;
int trie[N][27],idx,n,cnt[N];
using namespace std;
void insert(string a){
    int p=0,t;
    for(int i=0;i<a.size();i++){
        t=a[i]-'a';
        if(!trie[p][t])trie[p][t]=++idx;
        p=trie[p][t];
    }
    cnt[p]++;
}
int query(string a){
    int p=0,t;
    for(int i=0;i<a.size();i++){
        t=a[i]-'a';
        if(!trie[p][t])return 0;
        p=trie[p][t];
    }
    return cnt[p];
}
int main(){
    cin>>n;
    string str,a;
    while(n--){
        cin>>str;
        if(str=="I"){
            cin>>a;
            insert(a);
        }else{
            cin>>a;
            cout<<query(a)<<endl;
        }
    }
    return 0;
}

```

143. 最大异或对(trie 字典树存二进制)

在给定的 N 个整数 A_1, A_2, \dots, A_N 中选出两个进行 xor（异或）运算，得到的结果最大是多少？

输入格式

第一行输入一个整数 N 。

第二行输入 N 个整数 $A_1 \sim A_N$ 。

输出格式

输出一个整数表示答案。

数据范围

$1 \leq N \leq 10^5$,

$0 \leq A_i < 2^{31}$

输入样例：

```
3
1 2 3
```

输出样例：

```
3
```

```
#include<iostream>
using namespace std;
int n,trie[4000000][2],idx,arr[100010],big=0;
void insert(int x){
    int p=0;
    for(int i=30;i>=0;i--){
        int u=x>>i&1;
        if(!trie[p][u])trie[p][u]=++idx;
        p=trie[p][u];
    }
}
int query(int x){
    int p=0,res=0;
    for(int i=30;i>=0;i--){
        int u=x>>i&1;
        if(trie[p][!u]){
            res+=1<<i;
            p=trie[p][!u];
        }else p=trie[p][u];
    }
    return res;
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    cin>>n;
```

```

for(int i=1;i<=n;i++)cin>>arr[i],insert(arr[i]);
for(int i=1;i<=n;i++)big=max(big,query(arr[i]));
cout<<big<<endl;
return 0;
}

```

836. 合并集合(并查集基本操作)

一共有 n 个数，编号是 $1 \sim n$ ，最开始每个数各自在一个集合中。

现在要进行 m 个操作，操作共有两种：

1. `M a b`，将编号为 a 和 b 的两个数所在的集合合并，如果两个数已经在同一个集合中，则忽略这个操作；
2. `Q a b`，询问编号为 a 和 b 的两个数是否在同一个集合中；

输入格式

第一行输入整数 n 和 m 。

接下来 m 行，每行包含一个操作指令，指令为 `M a b` 或 `Q a b` 中的一种。

输出格式

对于每个询问指令 `Q a b`，都要输出一个结果，如果 a 和 b 在同一集合内，则输出 `Yes`，否则输出 `No`。

每个结果占一行。

数据范围

$1 \leq n, m \leq 105$

输入样例：

```

4 5
M 1 2
M 3 4
Q 1 2
Q 1 3
Q 3 4

```

输出样例：

```

Yes

```

No

Yes

```
#include<iostream>
using namespace std;
int p[100010];
int find(int x){
    if(x!=p[x])p[x]=find(p[x]);
    return p[x];
}
int main(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)p[i]=i;
    for(int i=1;i<=m;i++){
        char c[2];
        int a,b;
        cin>>c>>a>>b;
        if(c[0]=='M'){
            p[find(a)]=find(b);
        }else{
            if(find(a)==find(b))cout<<"Yes"<<endl;
            else cout<<"No"<<endl;
        }
    }
    return 0;
}
```

837. 连通块中点的数量(并查集基本操作+集合数量查询)

给定一个包含 n 个点（编号为 $1 \sim n$ ）的无向图，初始时图中没有边。

现在要进行 m 个操作，操作共有三种：

1. `C a b`，在点 a 和点 b 之间连一条边， a 和 b 可能相等；
2. `Q1 a b`，询问点 a 和点 b 是否在同一个连通块中， a 和 b 可能相等；
3. `Q2 a`，询问点 a 所在连通块中点的数量；

输入格式

第一行输入整数 n 和 m 。

接下来 m 行，每行包含一个操作指令，指令为 `C a b`，`Q1 a b` 或 `Q2 a` 中的一种。

输出格式

对于每个询问指令 **Q1 a b**，如果 **a** 和 **b** 在同一个连通块中，则输出 **Yes**，否则输出 **No**。

对于每个询问指令 **Q2 a**，输出一个整数表示点 **a** 所在连通块中点的数量

每个结果占一行。

数据范围

$1 \leq n, m \leq 10^5$

输入样例：

```
5 5
C 1 2
Q1 1 2
Q2 1
C 2 5
Q2 5
```

输出样例：

```
Yes
2
3
```

```
#include<iostream>
#include<cstring>
using namespace std;
int n,m,cnt[100010],p[100010];
int find(int x){
    if(x!=p[x])p[x]=find(p[x]);
    return p[x];
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)p[i]=i,cnt[i]=1;
    for(int i=1;i<=m;i++){
        char ch[3];
        int a,b;
        cin>>ch;
        if(!strcmp(ch,"C")){
            cin>>a>>b;
            if(find(a)!=find(b)){
```

```

        cnt[find(b)]+=cnt[find(a)];
        p[find(a)]=find(b);
    }
}
else if(!strcmp(ch,"Q1")){
    cin>>a>>b;
    if(find(a)==find(b))cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
}
else{
    cin>>a;
    cout<<cnt[find(a)]<<endl;
}
}
return 0;
}

```

240. 食物链(带权并查集)

动物王国中有三类动物 A,B,C，这三类动物的食物链构成了有趣的环形。

A 吃 B，B 吃 C，C 吃 A。

现有 N 个动物，以 1~ N 编号。

每个动物都是 A,B,C 中的一种，但是我们并不知道它到底是哪一种。

有人用两种说法对这 N 个动物所构成的食物链关系进行描述：

第一种说法是 1 X Y，表示 X 和 Y 是同类。

第二种说法是 2 X Y，表示 X 吃 Y。

此人对 N 个动物，用上述两种说法，一句接一句地说出 K 句话，这 K 句话有的是真的，有的是假的。

当一句话满足下列三条之一时，这句话就是假话，否则就是真话。

1. 当前的话与前面的某些真的话冲突，就是假话；
2. 当前的话中 X 或 Y 比 N 大，就是假话；
3. 当前的话表示 X 吃 X，就是假话。

你的任务是根据给定的 N 和 K 句话，输出假话的总数。

输入格式

第一行是两个整数 N 和 K，以一个空格分隔。

以下 KK 行每行是三个正整数 D，X，Y，两数之间用一个空格隔开，其中 D 表示说法的种类。

若 $D=1$ ，则表示 X 和 Y 是同类。

若 $D=2$ ，则表示 X 吃 Y 。

输出格式

只有一个整数，表示假话的数目。

数据范围

$1 \leq N \leq 50000$,

$0 \leq K \leq 100000$

输入样例：

```
100 7
1 101 1
2 1 2
2 2 3
2 3 3
1 1 3
2 3 1
1 5 5
```

输出样例：

```
3
```

```
#include<iostream>
using namespace std;
//带权并查集，权mod0和根节点同类，mod1吃根节点，mod2被根节点吃
int p[50010],d[50010],n,k,res;
int find(int x){
    if(x!=p[x]){
        int u=find(p[x]);
        d[x]+=d[p[x]];
        p[x]=u;
    }
    return p[x];
}
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++)p[i]=i;
```

```

for(int i=1;i<=k;i++){
    int t,x,y;
    cin>>t>>x>>y;
    if(x>n||y>n)res++;
    else{
        int px=find(x),py=find(y);
        if(t==1){//x 和 y 是同类
            if(px==py){//x 和 y 在一个集合中，判断是不是真话
                if((d[x]-d[y])%3!=0)res++; //发现不是同类，说假话
            }else{//不在一个集合中，根据题目意思无法判断就默认是真话
                p[px]=py; //因为是真话要完整关系方便之后继续判断是不是真话
                d[px]=d[y]-d[x];
            }
        }else{//x 和 y 不是同类
            if(px==py){//x 和 y 在一个集合中，判断是不是真话 x 吃 y
                if((d[x]-d[y]-1)%3!=0)res++;
            }else{//不在一个集合中，根据题目意思无法判断就默认是真话
                p[px]=py;
                d[px]=d[y]-d[x]+1;
            }
        }
    }
}

cout<<res<<endl;
return 0;
}

```

838. 堆排序(数组堆排序)

注意：若要排序完后整个数组小到大有序,可以先建大跟堆,然后和最后一个调换,n--,固定最后一个,最后一个就是最大的,然后重新 down,交换第一个和倒数第二个,反复进行,最后数组就是从小到大的。这题只是针 m 个对输出有序。

输入一个长度为 n 的整数数列，从小到大输出前 m 小的数。

输入格式

第一行包含整数 n 和 m。

第二行包含 n 个整数，表示整数数列。

输出格式

共一行，包含 m 个整数，表示整数数列中前 m 小的数。

数据范围

$1 \leq m \leq n \leq 10^5$,

$1 \leq \text{数列中元素} \leq 10^9$

输入样例:

```
5 3
4 5 1 3 2
```


输出样例:

```
1 2 3
```

```
#include<iostream>
using namespace std;
int heap[100010],n,m;
// 提供该题不需要的up 函数
void up(int x){
    if(x<=1)return;
    if(heap[x]<heap[x/2])swap(heap[x],heap[x/2]),up(x/2);
}
void down(int x){
    int t=x;
    if(2*x<=n&&heap[2*x]<heap[t])t=2*x;
    if(2*x+1<=n&&heap[2*x+1]<heap[t])t=2*x+1;
    if(t!=x)swap(heap[t],heap[x]),down(t);
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>heap[i];
    for(int i=n/2;i-->0)down(i);
    for(int i=1;i<=m;i++){
        cout<<heap[1]<<" ";
        heap[1]=heap[n--];
        down(1);
    }
    return 0;
}
```

840. 模拟散列表(链地址法和开放地址法)

维护一个集合，支持如下几种操作：

1.  `x`，插入一个整数 `x`；

2. `Q x`, 询问整数 x 是否在集合中出现过;

现在要进行 N 次操作, 对于每个询问操作输出对应的结果。

输入格式

第一行包含整数 N , 表示操作数量。

接下来 N 行, 每行包含一个操作指令, 操作指令为 `I x`, `Q x` 中的一种。

输出格式

对于每个询问指令 `Q x`, 输出一个询问结果, 如果 x 在集合中出现过, 则输出 `Yes`, 否则输出 `No`。

每个结果占一行。

数据范围

$1 \leq N \leq 10^5$

$-10^9 \leq x \leq 10^9$

输入样例:

```
5
I 1
I 2
I 3
Q 2
Q 5
```

输出样例:

```
Yes
No
```

链地址法

```
#include<iostream>
#include<cstring>
using namespace std;
int h[100003],n,idx,ne[100003],e[100003];
void insert(int x){
    int k=((x%100003)+100003)%100003;
    e[idx]=x,ne[idx]=h[k],h[k]=idx++;
}
int find(int x){
```

```

    int k=((x%100003)+100003)%100003;
    for(int i=h[k];i!=-1;i=ne[i]){
        if(e[i]==x)return true;
    }
    return false;
}
int main(){
    cin>>n;
    memset(h,-1,sizeof h);
    while(n--){
        char ch[2];
        int a;
        cin>>ch>>a;
        if(ch[0]=='I'){
            insert(a);
        }else{
            if(find(a))cout<<"Yes"<<endl;
            else cout<<"No"<<endl;
        }
    }
    return 0;
}

```

开放地址法

```

#include<bits/stdc++.h>
using namespace std;
int h[200003],null=0x3f3f3f3f;
int find(int x){
    int k=fabs(x%200003);
    while(h[k]!=null&&h[k]!=x){
        k++;
        if(k==200003)k=0;
    }
    return k;
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    memset(h,0x3f,sizeof h);
    int n;
    cin>>n;
    for(int i=0;i<n;i++){
        char ch[2];
        int a;
    }
}

```

```

        cin>>ch>>a;
        int k=find(a);
        if(*ch=='I'){
            h[k]=a;
        }else{
            if(h[k]==null)cout<<"No"<<endl;
            else cout<<"Yes"<<endl;
        }
    }
    return 0;
}

```

841. 字符串哈希

给定一个长度为 n 的字符串,再给定 m 个询问,每个询问包含四个整数 $l1,r1,l2,r2$,请你判断 $[l1,r1]$ 和 $[l2,r2]$ 这两个区间所包含的字符串子串是否完全相同。

字符串中只包含大小写英文字母和数字。

输入格式

第一行包含整数 n 和 m ,表示字符串长度和询问次数。

第二行包含一个长度为 n 的字符串,字符串中只包含大小写英文字母和数字。

接下来 m 行,每行包含四个整数 $l1,r1,l2,r2$,表示一次询问所涉及的两个区间。

注意,字符串的位置从 1 开始编号。

输出格式

对于每个询问输出一个结果,如果两个字符串子串完全相同则输出 **Yes**,否则输出 **No**。

每个结果占一行。

数据范围

$1 \leq n, m \leq 105$

输入样例:

```

8 3
aabbbaabb
1 3 5 7
1 3 6 8
1 2 1 2

```

输出样例:

Yes

No

Yes

```
#include<iostream>
using namespace std;
typedef unsigned long long ULL;
ULL h[100010],p[100010],n,m,M=131;
char a[100010];
ULL get(int l,int r){
    return h[r]-h[l-1]*p[r-l+1];
}
int main(){
    cin>>n>>m>>a+1;
    p[0]=1;
    for(int i=1;i<=n;i++){
        p[i]=p[i-1]*M;
        h[i]=h[i-1]*M+a[i];
    }
    while(m--){
        int l1,r1,l2,r2;
        cin>>l1>>r1>>l2>>r2;
        if(get(l1,r1)==get(l2,r2))cout<<"Yes"<<endl;
        else cout<<"No"<<endl;
    }
    return 0;
}
```

842. 排列数字(dfs 全排列)

给定一个整数 n ，将数字 $1 \sim n$ 排成一行，将会有很多种排列方法。

现在，请你按照字典序将所有的排列方法输出。

输入格式

共一行，包含一个整数 n 。

输出格式

按字典序输出所有排列方案，每个方案占一行。

数据范围

$1 \leq n \leq 7$

输入样例:

```
3
```

输出样例:

```
1 2 3
```

```
1 3 2
```

```
2 1 3
```

```
2 3 1
```

```
3 1 2
```

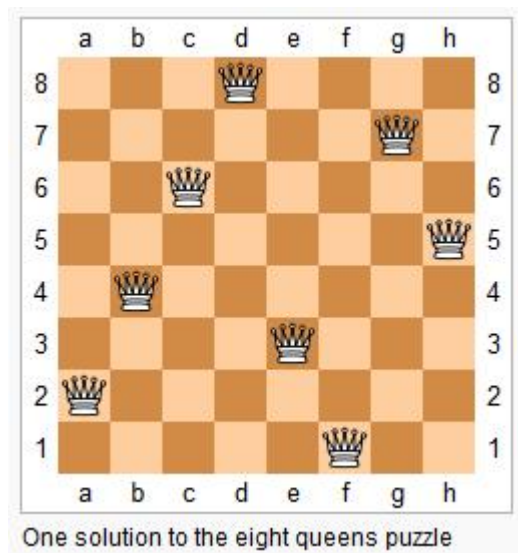
```
3 2 1
```

```
#include<iostream>
using namespace std;
int n,a[8],st[8];
void dfs(int x){
    if(x==n+1){
        for(int i=1;i<=n;i++)cout<<a[i]<<" ";
        cout<<endl;
        return;
    }
    for(int i=1;i<=n;i++){
        if(!st[i]){
            st[i]=1;
            a[x]=i;
            dfs(x+1);
            st[i]=0;
        }
    }
}
int main(){
    cin>>n;
    dfs(1);
    return 0;
}
```

843. n-皇后问题(DFS)

注意:斜线判断可以利用 $y = -x + b$ 和 $y = x + b$ 映射关系进行判断。即 $st[b] = 1$, 表示该斜线上已经存在皇后

n-皇后问题是指将 n 个皇后放在 $n \times n$ 的国际象棋棋盘上, 使得皇后不能相互攻击到, 即任意两个皇后都不能处于同一行、同一列或同一斜线上。



现在给定整数 n, 请你输出所有的满足条件的棋子摆法。

输入格式

共一行, 包含整数 n。

输出格式

每个解决方案占 n 行, 每行输出一个长度为 n 的字符串, 用来表示完整的棋盘状态。

其中 \cdot 表示某一个位置的方格状态为空, Q 表示某一个位置的方格上摆着皇后。

每个方案输出完成后, 输出一个空行。

注意: 行末不能有多余空格。

输出方案的顺序任意, 只要不重复且没有遗漏即可。

数据范围

$1 \leq n \leq 9$

输入样例:

```
4
```

输出样例:

```
.Q..
```

```

...Q
Q...
..Q.

..Q.
Q...
...Q
.Q..

```

```

#include<iostream>
using namespace std;
int g[10][10],l[10],zxie[20],yxie[20];
int n;
void dfs(int x){
    if(x>n){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                if(g[i][j])cout<<"Q";
                else cout<<".";
            }
            cout<<endl;
        }
        cout<<endl;
        return ;
    }
    for(int i=1;i<=n;i++){
        if(l[i])continue;
        if(zxie[i-x+n])continue; //y=x+b  b=y-x  因为数组下标不为负因此加n
        if(yxie[i+x])continue; //y=-x+b  b=y+x
        g[x][i]=1;
        l[i]=1;
        zxie[i-x+n]=1;
        yxie[i+x]=1;
        dfs(x+1);
        l[i]=0;
        zxie[i-x+n]=0;
        yxie[i+x]=0;
        g[x][i]=0;
    }
}

```



```

}
int main(){
    cin>>n;
    dfs(1);
}

```

844. 走迷宫(基本 bfs, 距离到距离 bfs)

给定一个 $n \times m$ 的二维整数数组，用来表示一个迷宫，数组中只包含 0 或 1，其中 0 表示可以走的路，1 表示不可通过的墙壁。

最初，有一个人位于左上角 (1,1) 处，已知该人每次可以向上、下、左、右任意一个方向移动一个位置。

请问，该人从左上角移动至右下角 (n,m) 处，至少需要移动多少次。

数据保证 (1,1) 处和 (n,m) 处的数字为 0，且一定至少存在一条通路。

输入格式

第一行包含两个整数 n 和 m 。

接下来 n 行，每行包含 m 个整数（0 或 1），表示完整的二维数组迷宫。

输出格式

输出一个整数，表示从左上角移动至右下角的最少移动次数。

数据范围

$1 \leq n, m \leq 100$

输入样例：

```

5 5
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0

```

输出样例：

```

8

```

```

#include<iostream>

```

```

#include<queue>
#include<cstring>
using namespace std;
int dx[]={0,0,-1,1};
int dy[]={1,-1,0,0};
int n,m,g[110][110],dis[110][110];
typedef pair<int,int> PII;
void bfs(int x,int y){
    memset(dis,0x3f,sizeof dis);
    queue<PII>q;
    q.push({x,y});
    dis[x][y]=0;
    while(q.size()){
        auto t=q.front();
        q.pop();
        for(int i=0;i<=3;i++){
            int nx=t.first+dx[i],ny=t.second+dy[i];
            if(nx>n||nx<1||ny>m||ny<1)continue;//边界检查
            if(g[nx][ny])continue;//障碍物检查
            if(dis[nx][ny]!=0x3f3f3f3f)continue;//可走点只走最短路径的一次
            dis[nx][ny]=dis[t.first][t.second]+1;
            q.push({nx,ny});
        }
    }
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            cin>>g[i][j];
    bfs(1,1);
    if(dis[n][m]!=0x3f3f3f3f)cout<<dis[n][m]<<endl;
    return 0;
}

```

845. 八数码(状态到状态 bfs)

注意:巧妙的用 string 存储每一个状态,巧妙的将 string 横向存储坐标转换成矩阵存储坐标
 在一个 3×3 的网格中, 1~ 8 这 8 个数字和一个 x 恰好不重不漏地分布在这 3×3 的网格中。

例如:

```

1 2 3
x 4 6

```

```
7 5 8
```

在游戏过程中，可以把 `x` 与其上、下、左、右四个方向之一的数字交换（如果存在）。

我们的目的是通过交换，使得网格变为如下排列（称为正确排列）：

```
1 2 3
4 5 6
7 8 x
```

例如，示例中图形就可以通过让 `x` 先后与右、下、右三个方向的数字交换成功得到正确排列。

交换过程如下：

```
1 2 3   1 2 3   1 2 3   1 2 3
x 4 6   4 x 6   4 5 6   4 5 6
7 5 8   7 5 8   7 x 8   7 8 x
```

现在，给你一个初始网格，请你求出得到正确排列至少需要进行多少次交换。

输入格式

输入占一行，将 3×3 的初始网格描绘出来。

例如，如果初始网格如下所示：

```
1 2 3
x 4 6
7 5 8
```

则输入为： `1 2 3 x 4 6 7 5 8`

输出格式

输出占一行，包含一个整数，表示最少交换次数。

如果不存在解决方案，则输出 `-1`。

输入样例：

```
2 3 4 1 5 x 7 6 8
```

输出样例

```
19
```

```
#include <iostream>
#include <algorithm>
```

```

#include <unordered_map>
#include <queue>
using namespace std;
int bfs(string state)//state-暂时状态 bfs
{
    queue<string> q;
    unordered_map<string, int> d;
    q.push(state);
    d[state] = 0;
    int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
    string end = "12345678x";
    while (q.size())
    {
        auto t = q.front();
        q.pop();

        if (t == end) return d[t];

        int distance = d[t];
        int k = t.find('x');
        int x = k / 3, y = k % 3; //从字符串下标映射到3,3 矩阵坐标
        for (int i = 0; i < 4; i++)
        {
            int a = x + dx[i], b = y + dy[i];
            if (a >= 0 && a < 3 && b >= 0 && b < 3)
            {
                swap(t[a * 3 + b], t[k]); //从3,3 矩阵坐标映射到字符串下标
                if (!d.count(t))
                {
                    d[t] = distance + 1;
                    q.push(t);
                }
                swap(t[a * 3 + b], t[k]); //恢复现场,不破坏for 的下一初始状态
            }
        }
    }
    return -1;
}

int main()
{
    char s;
    string state;
    for (int i = 0; i < 9; i++)
    {

```

```

        cin >> s;
        state += s;
    }
    cout << bfs(state) << endl;
    return 0;
}

```

846. 树的重心(DFS, 树与图的深度优先遍历)

注意: 为什么需要 `res = max(res, n - sum)` (`big = max(big, n - res)`); 因为 `st` 数组标记是否被遍历, 又因为 `main` 函数里 `dfs(1)` 这个 `1` 是随意的, 所以就拿样例来说, 如果 `1` 开始, 我们知道 `4` 是数的中心, 而这时候遍历 `4` 的时候, 因为 `1` 已经被遍历过了, 所以 `res` 在 `for` 循环里不能往上走 (不能往 `1` 那边走), 那 `res = max(res, n - sum)` 就有用了

给定一颗树, 树中包含 n 个结点 (编号 $1 \sim n$) 和 $n-1$ 条无向边。

请你找到树的重心, 并输出将重心删除后, 剩余各个连通块中点数的最大值。

重心定义: 重心是指树中的一个结点, 如果将这个点删除后, 剩余各个连通块中点数的最大值最小, 那么这个节点被称为树的重心。

输入格式

第一行包含整数 n , 表示树的结点数。

接下来 $n-1$ 行, 每行包含两个整数 a 和 b , 表示点 a 和点 b 之间存在一条边。

输出格式

输出一个整数 m , 表示将重心删除后, 剩余各个连通块中点数的最大值。

数据范围

$1 \leq n \leq 10^5$

输入样例

```

9
1 2
1 7
1 4
2 8
2 5
4 3
3 9

```

输出样例:

4

```

#include<iostream>
#include<cstring>
using namespace std;
int h[100010],e[200010],ne[200010],st[100010],idx,n,ans=1e9;
void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
//以x为根节点的树的结点个数，注意访问过的点不算
int dfs(int x){
    st[x]=1;
    int res=1,big=0;
    for(int i=h[x];i!=-1;i=ne[i]){
        int t=e[i];
        if(st[t])continue;
        int u=dfs(t);
        res+=u;
        big=max(big,u);
    }
    big=max(big,n-res);
    ans=min(ans,big);
    return res;
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    memset(h,-1,sizeof h);
    cin>>n;
    int a,b;
    for(int i=0;i<n-1;i++){
        cin>>a>>b;
        add(a,b),add(b,a);
    }
    dfs(a);
    cout<<ans<<endl;
}

```

847. 图中点的层次(bfs,树与图的广度优先遍历)

给定一个 n 个点 m 条边的有向图，图中可能存在重边和自环。

所有边的长度都是 1，点的编号为 $1 \sim n$ 。

请你求出 1 号点到 n 号点的最短距离，如果从 1 号点无法走到 n 号点，输出 -1。

输入格式

第一行包含两个整数 n 和 m 。

接下来 m 行，每行包含两个整数 a 和 b ，表示存在一条从 a 走到 b 的长度为 1 的边。

输出格式

输出一个整数，表示 1 号点到 n 号点的最短距离。

数据范围

$1 \leq n, m \leq 10^5$

输入样例：

```
4 5
1 2
2 3
3 4
1 3
1 4
```

输出样例：

```
1
```

```
#include<iostream>
#include<queue>
#include<cstring>
using namespace std;
int h[100010],e[100010],ne[100010],idx,n,m,dis[100010];
void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
queue<int>q;
void bfs(int x){
    q.push(x);
```

```

while(q.size()){
    int t=q.front();
    q.pop();
    for(int i=h[t];i!=-1;i=ne[i]){
        int s=e[i];
        if(dis[s])continue;
        q.push(s);
        dis[s]=dis[t]+1;
    }
}
if(dis[n])cout<<dis[n]<<endl;
else cout<<-1<<endl;
}
int main(){
    cin>>n>>m;
    memset(h,-1,sizeof h);
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        add(a,b);
    }
    if(1==n){
        cout<<0<<endl;
        return 0;
    }
    bfs(1);
    return 0;
}

```

848. 有向图的拓扑序列

给定一个 n 个点 m 条边的有向图，点的编号是 1 到 n ，图中可能存在重边和自环。

请输出任意一个该有向图的拓扑序列，如果拓扑序列不存在，则输出 -1 。

若一个由图中所有点构成的序列 A 满足：对于图中的每条边 (x,y) ， x 在 A 中都出现在 y 之前，则称 A 是该图的一个拓扑序列。

输入格式

第一行包含两个整数 n 和 m 。

接下来 m 行，每行包含两个整数 x 和 y ，表示存在一条从点 x 到点 y 的有向边 (x,y) 。

输出格式

共一行，如果存在拓扑序列，则输出任意一个合法的拓扑序列即可。

否则输出 -1。

数据范围

$1 \leq n, m \leq 10^5$

输入样例：

```
3 3
1 2
2 3
1 3
```

输出样例：

```
1 2 3
```

```
#include<iostream>
#include<cstring>
#include<queue>
using namespace std;
const int N=100010;
int h[N],e[N],ne[N],idx,n,m,d[N],cnt,ans[N];
queue<int>q;
void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
void tpx(){
    for(int i=1;i<=n;i++){
        if(!d[i])q.push(i),ans[cnt++]=i;
    }
    while(q.size()){
        int t=q.front();
        q.pop();
        for(int i=h[t];i!=-1;i=ne[i]){
            d[e[i]]--;
            if(!d[e[i]])q.push(e[i]),ans[cnt++]=e[i];
        }
    }
    if(cnt==n){
        for(int i=0;i<cnt;i++)cout<<ans[i]<<" ";
    }else{
        cout<<-1<<endl;
    }
}
```

```

int main(){
    memset(h,-1,sizeof h);
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        add(a,b);
        d[b]++;
    }
    tpx();
    return 0;
}

```

849. Dijkstra 求最短路 I(朴素版)[必正权边]

复杂度：一般 $O(n^2)$

给定一个 n 个点 m 条边的有向图，图中可能存在重边和自环，所有边权均为正值。

请你求出 1 号点到 n 号点的最短距离，如果无法从 1 号点走到 n 号点，则输出 -1 。

输入格式

第一行包含整数 n 和 m 。

接下来 m 行每行包含三个整数 x,y,z ，表示存在一条从点 x 到点 y 的有向边，边长为 z 。

输出格式

输出一个整数，表示 1 号点到 n 号点的最短距离。

如果路径不存在，则输出 -1 。

数据范围

$1 \leq n \leq 500$,

$1 \leq m \leq 105$,

图中涉及边长均不超过 10000。

输入样例：

```

3 3
1 2 2
2 3 1
1 3 4

```

输出样例:

3

```
#include<iostream>
#include<cstring>
using namespace std;
int e[100010],h[510],ne[100010],w[100010],n,m,idx,dis[510],st[510];
void add(int a,int b,int c){
    e[idx]=b,ne[idx]=h[a],w[idx]=c,h[a]=idx++;
}
int dijkstra(){
    dis[1]=0;
    for(int i=0;i<n;i++){
        int t=-1;
        for(int j=1;j<=n;j++){
            if(!st[j]&&(t==-1||dis[t]>dis[j]))t=j;
        }
        st[t]=1;
        for(int j=h[t];j!=-1;j=ne[j]){
            if(st[e[j]])continue;
            int s=e[j],wi=w[j];
            dis[s]=min(dis[s],dis[t]+wi);
        }
    }
    if(dis[n]==0x3f3f3f3f)return -1;
    else return dis[n];
}
int main(){
    memset(h,-1,sizeof h);
    memset(dis,0x3f,sizeof dis);
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    cout<<dijkstra()<<endl;
    return 0;
}
```

850. Dijkstra 求最短路 II(堆优化版)[必正权边]

复杂度: 一般 $O(m\log n)$

给定一个 n 个点 m 条边的有向图，图中可能存在重边和自环，所有边权均为非负值。

请你求出 1 号点到 n 号点的最短距离，如果无法从 1 号点走到 n 号点，则输出 -1 。

输入格式

第一行包含整数 n 和 m 。

接下来 m 行每行包含三个整数 x,y,z ，表示存在一条从点 x 到点 y 的有向边，边长为 z 。

输出格式

输出一个整数，表示 1 号点到 n 号点的最短距离。

如果路径不存在，则输出 -1 。

数据范围

$1 \leq n, m \leq 1.5 \times 10^5$,

图中涉及边长均不小于 0，且不超过 10000。

数据保证：如果最短路存在，则最短路的长度不超过 10^9 。

输入样例：

```
3 3
1 2 2
2 3 1
1 3 4
```

输出样例：

```
3
```

```
#include<iostream>
#include<cstring>
#include<queue>
using namespace std;
const int N=150100;
int h[N],e[N],ne[N],w[N],idx,n,m,st[N],dis[N];
typedef pair<int,int>PII;
priority_queue<PII,vector<PII>,greater<PII>>heap;
void add(int a,int b,int c){
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
int dijkstra(){
    dis[1]=0;
```

```

    heap.push({0,1});
    while(heap.size()){
        auto t=heap.top();
        heap.pop();
        if(st[t.second])continue;//若该点被其他点松弛几次加入 heap 几次，该点最多松弛
        其他的点一次。
        st[t.second]=1;
        for(int i=h[t.second];i!=-1;i=ne[i]){
            int s=e[i],wi=w[i];

            if(dis[s]>dis[t.second]+wi)dis[s]=dis[t.second]+wi,heap.push({dis[s],s});
        }
    }
    if(dis[n]==0x3f3f3f3f)return -1;
    else return dis[n];
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    memset(h,-1,sizeof h);
    memset(dis,0x3f,sizeof dis);
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    cout<<dijkstra();
    return 0;
}

```

853. 有边数限制的最短路(bellman-ford)[正负权边均可]

复杂度: $O(nm)$

思想:循环 k 次(边数限制次数)，每次循环每条边进行松弛(每次循环松弛边的时候要使用每次循环最开始的备份数据 dis 而不是实时变化的 dis 防止串联)。

注意:题目边权可能为负,因此 $0x3f3f3f3f$ 可能会被负权减小,因此 $dis[i]!=0x3f3f3f3f$ 不一定能到,有可能是无法到的,需要优化。可在松弛边的时候加上判断 $back[edges[j].a]!=0x3f3f3f3f$ 如果该点本来就不可到达,就不用松弛边了,这样 $dis[i]!=0x3f3f3f3f$ 就一定能到。

给定一个 n 个点 m 条边的有向图，图中可能存在重边和自环，边权可能为负数。

请你求出从 1 号点到 n 号点的最多经过 k 条边的最短距离，如果无法从 1 号点走到 n 号点，输出 `impossible`。

注意：图中可能 存在负权回路 。

输入格式

第一行包含三个整数 n,m,k 。

接下来 m 行，每行包含三个整数 x,y,z ，表示存在一条从点 x 到点 y 的有向边，边长为 z 。

点的编号为 $1\sim n$ 。

输出格式

输出一个整数，表示从 1 号点到 n 号点的最多经过 k 条边的最短距离。

如果不存在满足条件的路径，则输出 `impossible`。

数据范围

$1\leq n,k\leq 500$,

$1\leq m\leq 10000$,

$1\leq x,y\leq n$,

任意边长的绝对值不超过 10000 。

输入样例：

```
3 3 1
1 2 1
2 3 1
1 3 3
```

输出样例：

```
3
```

```
#include<bits/stdc++.h>
using namespace std;
int n,m,k,dis[510],back[510];
struct {
    int a,b,c;
}e[10010];
void bellman_ford(){
    dis[1]=0;
    for(int i=0;i<k;i++){
        memcpy(back,dis,sizeof dis);
        for(int j=0;j<m;j++){
            if(dis[e[j].b]>back[e[j].a]+e[j].c)dis[e[j].b]=back[e[j].a]+e[j].c;
        }
    }
}
```

```

        if(dis[n]>0x3f3f3f3f/2)cout<<"impossible";
        else cout<<dis[n];
    }
int main(){
    memset(dis,0x3f,sizeof dis);
    cin>>n>>m>>k;
    int a,b,c;
    for(int i=0;i<m;i++){
        cin>>a>>b>>c;
        e[i]={a,b,c};
    }
    bellman_ford();
}

```

851. spfa 求最短路[正负权边均可]

复杂度：一般 $O(m)$, 最坏 $O(nm)$

注意:对 bellman-ford 算法的优化, bellman-ford 算法每次循环松弛所有边, 但是有的边不需要松弛, 比如第一次循环, 即限制边数 1 次到达各点的最短距离, 那么离源点有两条边的点限制边数 1 次到达不了, 松弛之后 dis 是无穷大, 没松弛之前本来初始化是无穷大, 没必要松弛, 因此 spfa 优化, 只有 dis 有改变的边的端点, 对其他边的松弛才会有效, 才会加入队列, 相比于 dijkstra 每次是取 dis 最小的点作为最终答案的一部分并对其他边进行松弛, spfa 则是对每个可能能松弛边的点都进行进行松弛, 那么 dis 最小的点一定也会松弛。

给定一个 n 个点 m 条边的有向图, 图中可能存在重边和自环, 边权可能为负数。

请你求出 1 号点到 n 号点的最短距离, 如果无法从 1 号点走到 n 号点, 则输出 `impossible`。

数据保证不存在负权回路。

输入格式

第一行包含整数 n 和 m 。

接下来 m 行每行包含三个整数 x,y,z , 表示存在一条从点 x 到点 y 的有向边, 边长为 z 。

输出格式

输出一个整数, 表示 1 号点到 n 号点的最短距离。

如果路径不存在, 则输出 `impossible`。

数据范围

$1 \leq n, m \leq 105$,

图中涉及边长绝对值均不超过 10000。

输入样例:

```
3 3
1 2 5
2 3 -3
1 3 4
```

输出样例:

```
2
```

```
#include<iostream>
#include<cstring>
#include<queue>
using namespace std;
const int N=100010;
queue<int>q;
int n,m,dis[N],st[N],h[N],e[N],ne[N],w[N],idx;
void add(int a,int b,int c){
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
void spfa(){
    dis[1]=0;
    st[1]=1;
    q.push(1);
    while(q.size()){
        int t=q.front();
        q.pop();
        st[t]=0;
        for(int i=h[t];i!=-1;i=ne[i]){
            int s=e[i];
            if(dis[s]>dis[t]+w[i]){
                dis[s]=dis[t]+w[i];
                if(!st[s])q.push(s),st[s]=1;
            }
        }
    }
    if(dis[n]==0x3f3f3f3f)cout<<"impossible"<<endl;
    else cout<<dis[n]<<endl;
}
int main(){
    memset(dis,0x3f,sizeof dis);
    memset(h,-1,sizeof h);
    cin>>n>>m;
    for(int i=0;i<m;i++){
```



```

    int a,b,c;
    cin>>a>>b>>c;
    add(a,b,c);
}
spfa();
return 0;
}

```

852. spfa 判断负环[正负权边均可]

注意:判断负环是判断整个图,因此最开始 push 所有点进去,dis 都为 0,这样才能搜索到所有点路径上面是否有负环,如果只 push 一个点进去,负环可能不在这个点连通的路径上。相较于 spfa 算法,还需要增加 cnt 数组,判断该点是由几条边走过来的,如果 $\geq n$ 的话,必存在负环。

给定一个 n 个点 m 条边的有向图,图中可能存在重边和自环,边权可能为负数。

请你判断图中是否存在负权回路。

输入格式

第一行包含整数 n 和 m 。

接下来 m 行每行包含三个整数 x,y,z ,表示存在一条从点 x 到点 y 的有向边,边长为 z 。

输出格式

如果图中存在负权回路,则输出 Yes,否则输出 No。

数据范围

$1 \leq n \leq 2000$,

$1 \leq m \leq 10000$,

图中涉及边长绝对值均不超过 10000。

输入样例:

```

3 3
1 2 -1
2 3 4
3 1 -4

```

输出样例:

```

Yes

```

```

#include<iostream>
#include<cstring>
#include<queue>
using namespace std;
const int N=100010;
queue<int>q;
int n,m,dis[N],st[N],h[N],e[N],ne[N],w[N],idx,cnt[N];
void add(int a,int b,int c){
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
bool spfa(){
    for(int i=1;i<=n;i++){
        dis[i]=0;
        st[i]=1;
        q.push(i);
    }
    while(q.size()){
        int t=q.front();
        q.pop();
        st[t]=0;
        for(int i=h[t];i!=-1;i=ne[i]){
            int s=e[i];
            if(dis[s]>dis[t]+w[i]){
                dis[s]=dis[t]+w[i];
                cnt[s]=cnt[t]+1;
                if(cnt[s]>=n)return true;
                if(!st[s])q.push(s),st[s]=1;
            }
        }
    }
    return false;
}
int main(){
    memset(dis,0x3f,sizeof dis);
    memset(h,-1,sizeof h);
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    if(spfa())cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
    return 0;
}

```

}

854. Floyd 求最短路(多源最短路)[正负权边均可]

复杂度: $O(n^3)$

注意:类似 853. 有边数限制的最短路(bellman-ford),题目边权可能为负,因此 `0x3f3f3f3f` 可能会被负权减小,因此 `dis[i]!=0x3f3f3f3f` 不一定能到,有可能是无法到的,需要优化。可在松弛边的时候加上判断 `if(g[i][k]==INF||g[k][j]==INF)continue;`,如果该点本来就不可到达,就不用松弛边了,这样 `dis[i]!=0x3f3f3f3f` 就一定能到。

理解:对于 floyd 的理解,该代码中可以理解 $i \rightarrow k \rightarrow j$, k 视为点最先固定,然后固定 i 即 $i \rightarrow k$ 边,然后遍历 j 对其他所有边进行松弛,看 $i \rightarrow j$ 是否比 $i \rightarrow k \rightarrow j$ 大,大就松弛, j 遍历完遍历 i ,即换其他边(k 点没变)对所有边松弛, i 遍历完再换 k 点重新来一遍 i, j 分别遍历松弛。

给定一个 n 个点 m 条边的有向图,图中可能存在重边和自环,边权可能为负数。

再给定 k 个询问,每个询问包含两个整数 x 和 y ,表示查询从点 x 到点 y 的最短距离,如果路径不存在,则输出 `impossible`。

数据保证图中不存在负权回路。

输入格式

第一行包含三个整数 n, m, k 。

接下来 m 行,每行包含三个整数 x, y, z ,表示存在一条从点 x 到点 y 的有向边,边长为 z 。

接下来 k 行,每行包含两个整数 x, y ,表示询问点 x 到点 y 的最短距离。

输出格式

共 k 行,每行输出一个整数,表示询问的结果,若询问两点间不存在路径,则输出 `impossible`。

数据范围

$1 \leq n \leq 200$,

$1 \leq k \leq n^2$

$1 \leq m \leq 20000$,

图中涉及边长绝对值均不超过 10000。

输入样例:

```
3 3 2
1 2 1
2 3 2
1 3 1
2 1
```

1 3

输出样例:

Impossible

1

```
#include<iostream>
#include<cstring>
using namespace std;
const int N=210,INF=0x3f3f3f3f;
int n,m,g[N][N],k;
void floyd(){
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                g[i][j]=min(g[i][j],g[i][k]+g[k][j]);
            }
        }
    }
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    cin>>n>>m>>k;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(i==j)g[i][j]=0;
    else g[i][j]=INF;
    for(int i=0;i<m;i++){
        int a,b,c;
        cin>>a>>b>>c;
        g[a][b]=min(g[a][b],c);
    }
    floyd();
    for(int i=0;i<k;i++){
        int a,b;
        cin>>a>>b;
        if(g[a][b]>=INF/2)cout<<"impossible"<<endl;
        else cout<<g[a][b]<<endl;
    }
    return 0;
}
```

858. Prim 算法求最小生成树

注意:代码实现利用 `dijkstra` 算法思想,每次取到集合的最小距离(而不是到源点的最小距离)的点加入集合并松弛其他边,若松弛后找最小边发现没有的话那么就有孤立点不是最小生成树。

给定一个 n 个点 m 条边的无向图,图中可能存在重边和自环,边权可能为负数。

求最小生成树的树边权重之和,如果最小生成树不存在则输出 `impossible`。

给定一张边带权的无向图 $G=(V,E)$,其中 V 表示图中点的集合, E 表示图中边的集合, $n=|V|$, $m=|E|$ 。

由 V 中的全部 n 个顶点和 E 中 $n-1$ 条边构成的无向连通子图被称为 G 的一棵生成树,其中边的权值之和最小的生成树被称为无向图 G 的最小生成树。

输入格式

第一行包含两个整数 n 和 m 。

接下来 m 行,每行包含三个整数 u,v,w ,表示点 u 和点 v 之间存在一条权值为 w 的边。

输出格式

共一行,若存在最小生成树,则输出一个整数,表示最小生成树的树边权重之和,如果最小生成树不存在则输出 `impossible`。

数据范围

$1 \leq n \leq 500$,

$1 \leq m \leq 10^5$,

图中涉及边的边权的绝对值均不超过 10000。

输入样例:

```
4 5
1 2 1
1 3 2
1 4 3
2 3 2
3 4 4
```

输出样例:

```
6
```

```
#include<iostream>
#include<cstring>
```

```

using namespace std;
const int N=510,INF=0x3f3f3f3f;
int g[N][N],n,m,st[N],res,cnt,dis[N];
bool prim(){
    dis[1]=0;
    for(int i=0;i<n;i++){
        int t=-1;
        for(int j=1;j<=n;j++){
            if(!st[j]&&(t==-1||dis[t]>dis[j]))t=j;
        }
        st[t]=1;
        if(dis[t]==INF)return false;
        res+=dis[t];
        for(int j=1;j<=n;j++){
            if(!st[j])dis[j]=min(dis[j],g[t][j]);
        }
    }
    return true;
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    memset(dis,0x3f,sizeof dis);
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(i==j)g[i][j]=0;
    else g[i][j]=INF;
    for(int i=0;i<m;i++){
        int a,b,c;
        cin>>a>>b>>c;
        g[b][a]=g[a][b]=min(g[a][b],c);
    }
    if(prim())cout<<res<<endl;
    else cout<<"impossible"<<endl;
    return 0;
}

```

859. Kruskal 算法求最小生成树

算法复杂度: $O(m\log m)$

给定一个 n 个点 m 条边的无向图，图中可能存在重边和自环，边权可能为负数。

求最小生成树的树边权重之和，如果最小生成树不存在则输出 `impossible`。

给定一张边带权的无向图 $G=(V,E)$, 其中 V 表示图中点的集合, E 表示图中边的集合, $n=|V|$, $m=|E|$ 。

由 V 中的全部 n 个顶点和 E 中 $n-1$ 条边构成的无向连通子图被称为 G 的一棵生成树, 其中边的权值之和最小的生成树被称为无向图 G 的最小生成树。

输入格式

第一行包含两个整数 n 和 m 。

接下来 m 行, 每行包含三个整数 u,v,w , 表示点 u 和点 v 之间存在一条权值为 w 的边。

输出格式

共一行, 若存在最小生成树, 则输出一个整数, 表示最小生成树的树边权重之和, 如果最小生成树不存在则输出 `impossible`。

数据范围

$1 \leq n \leq 10^5$,

$1 \leq m \leq 2 \cdot 10^5$,

图中涉及边的边权的绝对值均不超过 1000。

输入样例:

```
4 5
1 2 1
1 3 2
1 4 3
2 3 2
3 4 4
```

输出样例:

```
6
```

```
#include<iostream>
#include<algorithm>
using namespace std;
const int N=100010,M=200010;
int p[N],n,m,cnt,res;
struct edge{
    int a,b,c;
}edges[M];
bool cmp(struct edge &a,struct edge &b){
    return a.c<b.c;
```

```

}
int find(int x){
    if(x!=p[x])p[x]=find(p[x]);
    return p[x];
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)p[i]=i;
    for(int i=0;i<m;i++){
        int a,b,c;
        cin>>a>>b>>c;
        edges[i]={a,b,c};
    }
    sort(edges,edges+m,cmp);
    for(int i=0;i<m;i++){
        if(find(edges[i].a)!=find(edges[i].b)){
            res+=edges[i].c;
            cnt++;
            p[find(edges[i].a)]=find(edges[i].b);
        }
    }
    if(cnt==n-1)cout<<res<<endl;
    else cout<<"impossible"<<endl;
}

```

860. 染色法判定二分图

算法复杂度: $O(n+m)$

二分图定义:不存在奇数环

注意:染色的时候二分图可能有多个连通分支，因此染色最开始要遍历每个点以每个点为起始点染色，同时判断如果该点染过色就不用染了，这样即使有多个连通分支都可以染色。

给定一个 n 个点 m 条边的无向图，图中可能存在重边和自环。

请你判断这个图是否是二分图。

输入格式

第一行包含两个整数 n 和 m 。

接下来 m 行，每行包含两个整数 u 和 v ，表示点 u 和点 v 之间存在一条边。

输出格式

如果给定图是二分图，则输出 **Yes**，否则输出 **No**。

数据范围

$1 \leq n, m \leq 10^5$

输入样例:

```
4 4
1 3
1 4
2 3
2 4
```

输出样例:

Yes

```
#include<iostream>
#include<cstring>
using namespace std;
const int N=100010,M=200010;
int h[N],e[M],ne[M],idx,color[N],n,m;
void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
bool dfs(int x,int c){
    color[x]=c;
    for(int i=h[x];i!=-1;i=ne[i]){
        int t=e[i];
        if(color[t]){
            if(color[t]==color[x])return false;
        }else{
            if(!dfs(t,3-c))return false;
        }
    }
    return true;
}
int main(){
    memset(h,-1,sizeof h);
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        add(a,b);
        add(b,a);
    }
```

```

bool flag=true;
for(int i=1;i<=n;i++){
    if(color[i])continue;
    if(!dfs(i,1)){
        flag=false;
        break;
    }
}
if(flag)cout<<"Yes"<<endl;
else cout<<"No"<<endl;
return 0;
}

```

861. 二分图的最大匹配(匈牙利算法)

思想:为左边的点找能否匹配右边的点,若可以就匹配,若不可以,创造机会,看和不可以点有关系的右边的点原先匹配的左边的点能不能换个右边的点,然后这个右边的点给不可以的左边的点,若创造不了机会就放弃。

注意:思想中只用到了左边找右边,虽然是无向图,可以只存左->右有向就行。

注意:每一轮 find 必须要初始化 st 数组,保证每次 find 递归考虑右边的点一次,否则可能会出现左 1 点到右 1 点右 1 点到左 1 点死循环。

算法复杂度: $O(nm)$

给定一个二分图,其中左半部包含 n_1 个点(编号 $1 \sim n_1$),右半部包含 n_2 个点(编号 $1 \sim n_2$),二分图共包含 m 条边。

数据保证任意一条边的两个端点都不可能在同一部分中。

请你求出二分图的最大匹配数。

二分图的匹配:给定一个二分图 G ,在 G 的一个子图 M 中, M 的边集 $\{E\}$ 中的任意两条边都不依附于同一个顶点,则称 M 是一个匹配。

二分图的最大匹配:所有匹配中包含边数最多的一组匹配被称为二分图的最大匹配,其边数即为最大匹配数。

输入格式

第一行包含三个整数 n_1 、 n_2 和 m 。

接下来 m 行,每行包含两个整数 u 和 v ,表示左半部点集中的点 u 和右半部点集中的点 v 之间存在一条边。

输出格式

输出一个整数,表示二分图的最大匹配数。

数据范围

$1 \leq n_1, n_2 \leq 500,$

$1 \leq u \leq n_1,$

$1 \leq v \leq n_2,$

$1 \leq m \leq 10^5$

输入样例:

```
2 2 4
1 1
1 2
2 1
2 2
```

输出样例:

```
2
```

```
#include<iostream>
#include<cstring>
using namespace std;
const int N=510,M=100010;
int st[N],belong[N],n1,n2,m,idx,h[N],e[M],ne[M],cnt;
void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
bool find(int x){
    for(int i=h[x];i!=-1;i=ne[i]){
        int t=e[i];
        if(st[t])continue;
        st[t]=1;
        if(!belong[t]||find(belong[t])){
            belong[t]=x;
            return true;
        }
    }
    return false;
}
int main(){
    memset(h,-1,sizeof h);
    cin>>n1>>n2>>m;
    for(int i=0;i<m;i++){
```

```

    int a,b;
    cin>>a>>b;
    add(a,b);
}
for(int i=1;i<=n1;i++){
    memset(st,0,sizeof st);
    if(find(i))cnt++;
}
cout<<cnt<<endl;
return 0;
}

```

866. 试除法判定质数

证明为什么只用遍历到跟号 x : 假设 d 是 x 的因数则 $d|x$ 则 $(x/d)|x$ 假设 $d \leq (x/d)$ 解得 $d \leq \sqrt{x}$

给定 n 个正整数 a_i ，判定每个数是否是质数。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含一个正整数 a_i 。

输出格式

共 n 行，其中第 i 行输出第 i 个正整数 a_i 是否为质数，是则输出 **Yes**，否则输出 **No**。

数据范围

$1 \leq n \leq 100$,

$1 \leq a_i \leq 2^{31}-1$

输入样例：

```

2
2
6

```

输出样例：

```

Yes
No

```

```
#include<iostream>
```

```

using namespace std;
bool is_prime(int x){
    if(x<=1)return false;
    for(int i=2;i<=x/i;i++){
        if(x%i==0)return false;
    }
    return true;
}
int main(){
    int n,x;
    cin>>n;
    while(n--){
        cin>>x;
        if(is_prime(x))cout<<"Yes"<<endl;
        else cout<<"No"<<endl;
    }
    return 0;
}

```

867. 分解质因数

性质：n 中最多只含有一个大于 \sqrt{n} 的因子。证明通过反证法：如果有两个大于 \sqrt{n} 的因子，那么相乘会大于 n，矛盾。证毕

给定 n 个正整数 a_i ，将每个数分解质因数，并按照质因数从小到大的顺序输出每个质因数的底数和指数。

输入格式

第一行包含整数 n。

接下来 n 行，每行包含一个正整数 a_i 。

输出格式

对于每个正整数 a_i ，按照从小到大的顺序输出其分解质因数后，每个质因数的底数和指数，每个底数和指数占一行。

每个正整数的质因数全部输出完毕后，输出一个空行。

数据范围

$1 \leq n \leq 100$,

$2 \leq a_i \leq 2 \times 10^9$

输入样例：

```
2
```

6
8

输出样例:

2 1
3 1

2 3

```
#include<iostream>
using namespace std;
int n,x;
void fj(int x){
    for(int i=2;i<=x/i;i++){
        if(x%i==0){
            int cnt=0;
            while(x%i==0)x/=i,cnt++;
            cout<<i<<" "<<cnt<<endl;
        }
    }
    if(x>1)cout<<x<<" "<<1<<endl;
}
int main(){
    cin>>n;
    while(n--){
        cin>>x;
        fj(x);
        cout<<endl;
    }
    return 0;
}
```

868. 筛质数

线性筛法思想:筛掉的数都是用该数的最小质因子筛的,所以被筛掉的数都只会被筛一次实现线性筛。

$i \times \text{prime}[j] == 0$ $\text{prime}[j]$ 是质数,且被 i 整除,且是从小到大遍历的,因此必是 i 的最小质因子,也必是 $i \times \text{prime}[j]$ 的最小质因子。因此满足这个条件要 `break`,否则可能会被比较大的质因子筛掉。
 $i \times \text{prime}[j] \neq 0$ $\text{prime}[j]$ 是质数,且被 i 不能整除,且是从小到大遍历的,因此必比 i 的最小质因子小,所以说是 $i \times \text{prime}[j]$ 的最小质因子(因为对于单独 $\text{prime}[j]$,它是自己的最小质因子)。

综上所述只要筛掉 $i * \text{prime}[j]$ 筛掉的数都是最小质因子筛的, 只会筛一次。

给定一个正整数 n , 请你求出 $1 \sim n$ 中质数的个数。

输入格式

共一行, 包含整数 n 。

输出格式

共一行, 包含一个整数, 表示 $1 \sim n$ 中质数的个数。

数据范围

$1 \leq n \leq 10^6$

输入样例:

8

输出样例:

4

1. 线性筛法 $O(n)$ 是质数就往后筛, 一个数只会被筛一次

```
#include<iostream>
using namespace std;
int prime[1000010], cnt, st[1000010];
int main(){
    int n;
    cin >> n;
    for(int i=2; i<=n; i++){
        if(!st[i]){
            prime[cnt++] = i;
        }
        for(int j=0; prime[j]<=n/i; j++){
            st[prime[j]*i] = 1;
            if(i%prime[j]==0) break;
        }
    }
    cout << cnt << endl;
    return 0;
}
```

2. 埃氏筛法 $O(n \log \log n)$ 是质数就向后筛, 一个数可能会被多个质数筛, 如 $7 * 13$ 会被 7 和 13 筛

```
#include<iostream>
using namespace std;
int prime[1000010], cnt, st[1000010];
int main(){
    int n;
```

```

    cin>>n;
    for(int i=2;i<=n;i++){
        if(!st[i]){
            prime[cnt++]=i;
            for(int j=2*i;j<=n;j+=i){
                st[j]=1;
            }
        }
    }
    cout<<cnt<<endl;
    return 0;
}

```

3. 朴素筛法 $O(n\log n)$ 不管合数还是质数都向后筛

```

void get_primes(){
    for(int i=2;i<=n;i++){
        if(!st[i]) primes[cnt++]=i;
        for(int j=2*i;j<=n;j+=i){
            st[j]=1;
        }
    }
}

```

869. 试除法求约数

给定 n 个正整数 a_i ，对于每个整数 a_i ，请你按照从小到大的顺序输出它的所有约数。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含一个整数 a_i 。

输出格式

输出共 n 行，其中第 i 行输出第 i 个整数 a_i 的所有约数。

数据范围

$1 \leq n \leq 100$,

$1 \leq a_i \leq 2 \times 10^9$

输入样例：

```

2
6

```


8

输出样例:

1 2 3 6

1 2 4 8

```
#include<iostream>
#include<algorithm>
using namespace std;
vector<int>qys(int x){
    vector<int>ans;
    for(int i=1;i<=x/i;i++){
        if(x%i==0){
            ans.push_back(i);
            if(i!=x/i)ans.push_back(x/i);
        }
    }
    sort(ans.begin(),ans.end());
    return ans;
}
int main(){
    int n,x;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>x;
        auto t=qys(x);
        for(auto j: t){
            cout<<j<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

870. 约数个数

$N = (p_1^{x_1})(p_2^{x_2})(p_3^{x_3}) \cdots (p_k^{x_k})$ 约数个数 $= (x_1+1)(x_2+1)(x_3+1) \cdots (x_k+1)$

其中 N 为任意约数, p 为质因子, x 为质因子的次数, 取值范围 $(0-x)$

给定 n 个正整数 a_i , 请你输出这些数的乘积的约数个数, 答案对 10^9+7 取模。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含一个整数 a_i 。

输出格式

输出一个整数，表示所给正整数的乘积的约数个数，答案需对 10^9+7 取模。

数据范围

$1 \leq n \leq 100$,

$1 \leq a_i \leq 2 \times 10^9$

输入样例：

```
3
2
6
8
```

输出样例：

```
12
```

```
#include<iostream>
#include<unordered_map>
using namespace std;
unordered_map<int,int>prime;
const int mod=1e9+7;
int n,x,ans=1;
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>x;
        for(int j=2;j<=x/j;j++){
            if(x%j==0){
                while(x%j==0)x/=j,prime[j]++;
            }
        }
        if(x>1)prime[x]++;
    }
    for(auto i : prime){
        ans=1ll*ans*(i.second+1)%mod;
    }
    cout<<ans<<endl;
    return 0;
}
```

871. 约数之和

如果 $N = p_1^{c_1} * p_2^{c_2} * \dots * p_k^{c_k}$

约数个数: $(c_1+1)*(c_2+1)*\dots*(c_k+1)$

约数之和: $(p_1^0+p_1^1+\dots+p_1^{c_1})*\dots*(p_k^0+p_k^1+\dots+p_k^{c_k})$

给定 n 个正整数 a_i , 请你输出这些数的乘积的约数之和, 答案对 10^9+7 取模。

输入格式

第一行包含整数 n 。

接下来 n 行, 每行包含一个整数 a_i 。

输出格式

输出一个整数, 表示所给正整数的乘积的约数之和, 答案需对 10^9+7 取模。

数据范围

$1 \leq n \leq 100$,

$1 \leq a_i \leq 2 \times 10^9$

输入样例:

```
3
2
6
8
```

输出样例:

```
252
```

```
#include<iostream>
#include<unordered_map>
using namespace std;
const int mod=1e9+7;
int n,x,ans=1;
unordered_map<int,int>primes;
int main(){
    cin>>n,x;
    while(n--){
        cin>>x;
        for(int i=2;i<=x/i;i++){
            while(x%i==0)x/=i,primes[i]++;
        }
    }
    for(int i=2;i<=x;i++){
        int t=primes[i];
        int s=1;
        while(t--){
            s=(s*i+1)%mod;
        }
        ans=(ans*s)%mod;
    }
    cout<<ans<<endl;
}
```

```

    }
    if(x>1)primes[x]++;
}
for(auto i : primes){
    int p=i.first,c=i.second,t=1;
    while(c--)t=(111*t*p+1)%mod;
    ans=111*ans*t%mod;
}
cout<<ans<<endl;
}

```

872. 最大公约数

```

int gcd(int a,int b){
    return b?gcd(b,a%b):a;
}

```

873. 欧拉函数

给定 n 个正整数 a_i ，请你求出每个数的欧拉函数。

欧拉函数的定义

$1 \sim N$ 中与 N 互质的数的个数被称为欧拉函数，记为 $\phi(N)$ 。

若在算数基本定理中， $N=p_1^{a_1}p_2^{a_2}\dots p_m^{a_m}$ ，则：

$$\phi(N) = N \times (p_1-1)/p_1 \times (p_2-1)/p_2 \times \dots \times (p_m-1)/p_m$$

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含一个正整数 a_i 。

输出格式

输出共 n 行，每行输出一个正整数 a_i 的欧拉函数。

数据范围

$1 \leq n \leq 100$,

$1 \leq a_i \leq 2 \times 10^9$

输入样例：

```

3
3

```

6
8

输出样例:

2
2
4

```
#include<iostream>
using namespace std;
int n,x;
int main(){
    cin>>n;
    while(n--){
        cin>>x;
        int res=x;
        for(int i=2;i<=x/i;i++){
            if(x%i==0){
                while(x%i==0)x/=i;
                res=res/i*(i-1);
            }
        }
        if(x>1){
            res=res/x*(x-1);
        }
        cout<<res<<endl;
    }
    return 0;
}
```

874. 筛法求欧拉函数

给定一个正整数 n ，求 $1 \sim n$ 中每个数的欧拉函数之和。

输入格式

共一行，包含一个整数 n 。

输出格式

共一行，包含一个整数，表示 $1 \sim n$ 中每个数的欧拉函数之和。

数据范围

$1 \leq n \leq 10^6$

输入样例:

6

输出样例:

12

```
#include<iostream>
using namespace std;
const int N=1000010;
int prime[N],cnt,n,st[N],phi[N];
int main(){
    cin>>n;
    phi[1]=1;
    st[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i])prime[cnt++]=i,phi[i]=i-1;
        for(int j=0;prime[j]<=n/i;j++){
            st[prime[j]*i]=1;
            if(i%prime[j]==0){//prime[j]是i中的最小质因子
                phi[i*prime[j]]=phi[i]*prime[j];
                break;
            }else{//prime[j]不是i中的一个质因子,且比最小质因子还小
                phi[i*prime[j]]=phi[i]*(prime[j]-1);
            }
        }
    }
    long long res=0;
    for(int i=1;i<=n;i++)res+=phi[i];
    cout<<res<<endl;
    return 0;
}
```

875. 快速幂

给定 n 组 a_i, b_i, p_i , 对于每组数据, 求出 $a_i^{b_i} \bmod p_i$ 的值。

输入格式

第一行包含整数 n 。

接下来 n 行, 每行包含三个整数 a_i, b_i, p_i 。

输出格式

对于每组数据，输出一个结果，表示 $a_i^{b_i} \bmod p_i$ 的值。

每个结果占一行。

数据范围

$1 \leq n \leq 100000$,

$1 \leq a_i, b_i, p_i \leq 2 \times 10^9$

输入样例：

```
2
3 2 5
4 3 9
```

输出样例：

```
4
1
```

```
#include<iostream>
using namespace std;
int qi_mi(int a,int b,int p){
    int ans=1;
    while(b){
        if(b&1)ans=1ll*ans*a%p;
        a=1ll*a*a%p;
        b>>=1;
    }
    return ans;
}
int main(){
    int n;
    cin>>n;
    while(n--){
        int a,b,p;
        cin>>a>>b>>p;
        cout<<qi_mi(a,b,p)<<endl;
    }
    return 0;
}
```

876. 快速幂求逆元

$a/b \equiv a * x \pmod{p}$ a 和 p 必须互质，否则无逆元

1. 当 p 为质数, 用快速幂求解

2. 当 p 不是质数, 用扩展欧几里得算法求解

给定 n 组 a_i, p_i , 其中 p_i 是质数, 求 a_i 模 p_i 的乘法逆元, 若逆元不存在则输出 `impossible`。

注意: 请返回在 $0 \sim p-1$ 之间的逆元。

乘法逆元的定义

若整数 b, m 互质, 并且对于任意的整数 a , 如果满足 $b|a$, 则存在一个整数 x , 使得 $a/b \equiv a * x \pmod{m}$, 则称 x 为 b 的模 m 乘法逆元, 记为 $b^{-1} \pmod{m}$ 。

b 存在乘法逆元的充要条件是 b 与模数 m 互质。当模数 m 为质数时, $b^{(m-2)}$ 即为 b 的乘法逆元。

输入格式

第一行包含整数 n 。

接下来 n 行, 每行包含一个数组 a_i, p_i , 数据保证 p_i 是质数。

输出格式

输出共 n 行, 每组数据输出一个结果, 每个结果占一行。

若 a_i 模 p_i 的乘法逆元存在, 则输出一个整数, 表示逆元, 否则输出 `impossible`。

数据范围

$1 \leq n \leq 10^5$,

$1 \leq a_i, p_i \leq 2 * 10^9$

输入样例:

```
3
4 3
8 5
6 3
```

输出样例:

```
1
2
impossible
```



```

#include<iostream>
using namespace std;
int n;
int qi_mi(int a,int b,int p){
    int ans=1;
    while(b){
        if(b&1)ans=1ll*ans*a%p;
        b>>=1;
        a=1ll*a*a%p;
    }
    return ans;
}
int main(){
    cin>>n;
    while(n--){
        int a,b;
        cin>>a>>b;
        if(a%b==0)cout<<"impossible"<<endl;
        else cout<<qi_mi(a,b-2,b)<<endl;
    }
    return 0;
}

```

877. 扩展欧几里得算法

$a/b \equiv a \cdot x \pmod{p}$ a 和 p 必须互质，否则无逆元

1.当 p 为质数,用快速幂求解

2.当 p 不是质数,用扩展欧几里得算法求解

该题只是扩展欧几里得算法的普遍用法，不是求逆元。

给定 n 对正整数 a_i, b_i ，对于每对数，求出一组 x_i, y_i ，使其满足 $a_i \cdot x_i + b_i \cdot y_i = \gcd(a_i, b_i)$ 。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含两个整数 a_i, b_i 。

输出格式

输出共 n 行，对于每组 a_i, b_i ，求出一组满足条件的 x_i, y_i ，每组结果占一行。

本题答案不唯一，输出任意满足条件的 x_i, y_i 均可。

数据范围

$1 \leq n \leq 10^5$,

$1 \leq a_i, b_i \leq 2 \times 10^9$

输入样例:

```
2
4 6
8 18
```

输出样例:

```
-1 1 -2 1
```

```
#include<iostream>
using namespace std;
int n;
int exgcd(int a,int b,int &x,int &y){
    if(!b){
        x=1,y=0;
        return a;
    }else{
        int d=exgcd(b,a%b,y,x);
        //a*x1+b*y1=b*y+ax-a/b*b*x
        //a*x1+b*y1=ax+b(y-a/b*x)
        //x1=x y1=y-a/b*x
        y=y-(a/b)*x;
        return d;
    }
}
int main(){
    cin>>n;
    int x,y,a,b;
    while(n--){
        cin>>a>>b;
        exgcd(a,b,x,y);
        cout<<x<<" "<<y<<endl;
    }
    return 0;
}
```

878. 线性同余方程(扩展欧几里得算法)

给定 n 组数据 a_i, b_i, m_i , 对于每组数求出一个 x_i , 使其满足 $a_i x_i \equiv b_i \pmod{m_i}$, 如果无解则输出 `impossible`。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含一组数据 a_i, b_i, m_i 。

输出格式

输出共 n 行，每组数据输出一个整数表示一个满足条件的 x_i ，如果无解则输出 `impossible`。

每组数据结果占一行，结果可能不唯一，输出任意一个满足条件的结果均可。

输出答案必须在 `int` 范围之内。

数据范围

$1 \leq n \leq 10^5$,

$1 \leq a_i, b_i, m_i \leq 2 \times 10^9$

输入样例：

```
2
2 3 6
4 3 5
```

输出样例：

```
Impossible
-3
```

```
#include<iostream>
using namespace std;
int n;
int exgcd(int a,int b,int &x,int &y){
    if(!b){
        x=1,y=0;
        return a;
    }
    int d=exgcd(b,a%b,x,y);
    // ax1+by1=bx1+ay1-a/b*by1
    // ax1+by1=ay+b(x-a/b*y)
    //x1=y y1=x-a/b*y
    int temp=x;
    x=y;
    y=temp-(a/b)*y;
    return d;
}
}
```

```

int main(){
    cin>>n;
    while(n--){
        int a,b,m,x,y;
        cin>>a>>b>>m;
        //ax+my=b
        int d=exgcd(a,m,x,y);
        if(b%d==0)cout<<1ll*x*b/d%m<<endl;
        else cout<<"impossible"<<endl;
    }
    return 0;
}

```

204. 表达整数的奇怪方式(扩展中国剩余定理 EXCRT)

EXCRT 扩展中国剩余定理[mi 不要求互质] 和 CRT 中国剩余定理[mi 要求两两互质]

EXCRT 两式合并法（推公式） CRT 题可以 EXCRT 做

CRT 公式构造法（记结论）

给定 $2n$ 个整数 a_1, a_2, \dots, a_n 和 m_1, m_2, \dots, m_n , 求一个最小的非负整数 x , 满足 $\forall i \in [1, n], x \equiv m_i \pmod{a_i}$ 。

输入格式

第 1 行包含整数 n 。

第 $2 \dots n+1$ 行：每 $i+1$ 行包含两个整数 a_i 和 m_i , 数之间用空格隔开。

输出格式

输出最小非负整数 x , 如果 x 不存在, 则输出 -1 。

数据范围

$1 \leq a_i \leq 2^{31}-1$,

$0 \leq m_i < a_i$

$1 \leq n \leq 25$

所有 m_i 的最小公倍数在 64 位有符号整数范围内。

输入样例：

```

2
8 7
11 9

```

输出样例:

31

$$x1=a1(\text{mod } m1) \quad x2=a2(\text{mod } m2)$$

$$x1 = a1 + m1*y1 \quad x2 = a2 + m2*y2 \quad (y1, y2 \text{ 为任意整数, 与下面 } k \text{ 一致})$$

$$x1=x2$$

$$a1 + m1*y1=a2 + m2*y2$$

$$m1*y1-m2*y2=a2-a1$$

设 $d=\text{gcd}(m1, m2, y10, y20)$ 若 $d \mid a2-a1$ 有解 反之无解

有解则 $y1$ 通解= $y10 + k*m2/d$ $y2$ 通解= $y20 + k*m1/d$ ($y10$ 和 $y20$ 表示通过 `exgcd` 求出来的特解)(两个通解带到方程中除了特解 $y10$ 和 $y20$, 后面的 k 啥啥都抵消了)

$$\text{求 } x1 \quad x1 = a1 + m1*y1 = a1 + m1*(y10 + k*m2/d) = a1 + m1*y10 + k*m1*m2/d$$

$$x1 = a1 + y1*m1$$

$$\text{对比知 } a1=a1+m1*y10 \quad m1=m1*m2/d$$

```
#include<iostream>
using namespace std;
typedef long long LL;
LL exgcd(LL a, LL b, LL &x, LL &y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }

    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

int main(){
    LL n, a1, m1;
    bool flag=true;
    cin>>n;
    cin>>m1>>a1;
    for(int i=1; i<n; i++){
        LL m2, a2, y10, y20;
        cin>>m2>>a2;
        LL d=exgcd(m1, m2, y10, y20);
        if((a2-a1)%d){
            flag=false;
            break;
        }else{
            y10*=(a2-a1)/d;
            LL mod=abs(m2/d);
```

883. 高斯消元解线性方程组

$$\left. \begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\dots\dots\dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \end{aligned} \right\}$$

所有输入系数以及常数均保留两位小数，绝对值均不超过 100。

输入样例：

```
3
1.00 2.00 -1.00 -6.00
2.00 1.00 -3.00 -9.00
-1.00 -1.00 2.00 7.00
```

输出样例：

```
1.00
-2.00
3.00
```

```
#include<iostream>
#include<cmath>
using namespace std;
const double eps=1e-6;
const int N=110;
double a[N][N];
int n;
int gauss(double a[N][N]){
    int r=1; // 线代行阶梯化简从上到下定下来的行
    for(int c=1;c<=n;c++){
        int t=r;
        for(int j=r;j<=n;j++){ // 找每列最大数
            if(fabs(a[j][c])>fabs(a[t][c]))t=j;
        }
        if(fabs(a[t][c])<eps)continue;
        for(int j=c;j<=n;j++)swap(a[t][j],a[r][j]); // 最大数这行交换到上面
        for(int j=n+1;j>=c;j--)a[r][j]/=a[r][c];
        for(int j=r+1;j<=n;j++){
            if(fabs(a[j][c])<eps)continue;
            for(int k=n+1;k>=c;k--)a[j][k]-=a[r][k]*a[j][c];
        }
        r++;
    }
    if(r>=n+1){
        for(int i=n-1;i>=0;i--){
            for(int j=i+1;j<=n;j++){
```

```

        a[i][n+1]-=a[j][n+1]*a[i][j];
    }
}
return 0;
}
else {
    for(int i=r;i<=n;i++){
        if(fabs(a[i][n+1])>eps)return 2;
    }
    return 1;
}
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n+1;j++){
            cin>>a[i][j];
        }
    }
    int t=gauss(a);
    if(t==0){//唯一解
        for(int i=1;i<=n;i++){
            printf("%.2f\n",a[i][n+1]);
        }
    }else if(t==1){//无数解
        cout<<"Infinite group solutions"<<endl;
    }else{//无解
        cout<<"No solution"<<endl;
    }
    return 0;
}

```

884. 高斯消元解异或线性方程组

输入一个包含 n 个方程 n 个未知数的异或线性方程组。

方程组中的系数和常数为 0 或 1，每个未知数的取值也为 0 或 1。

求解这个方程组。

异或线性方程组示例如下：

$$M[1][1]x[1] \oplus M[1][2]x[2] \oplus \dots \oplus M[1][n]x[n] = B[1]$$

$$M[2][1]x[1] \oplus M[2][2]x[2] \oplus \dots \oplus M[2][n]x[n] = B[2]$$

...

$$M[n][1]x[1] \oplus M[n][2]x[2] \oplus \dots \oplus M[n][n]x[n] = B[n]$$

其中 \oplus 表示异或(XOR), $M[i][j]$ 表示第 i 个式子中 $x[j]$ 的系数, $B[i]$ 是第 i 个方程右端的常数, 取值均为 0 或 1。

输入格式

第一行包含整数 n 。

接下来 n 行, 每行包含 $n+1$ 个整数 0 或 1, 表示一个方程的 n 个系数以及等号右侧的常数。

输出格式

如果给定线性方程组存在唯一解, 则输出共 n 行, 其中第 i 行输出第 i 个未知数的解。

如果给定线性方程组存在多组解, 则输出 `Multiple sets of solutions`。

如果给定线性方程组无解, 则输出 `No solution`。

数据范围

$$1 \leq n \leq 100$$

输入样例:

```
3
1 1 0 1
0 1 1 0
1 0 0 1
```

输出样例:

```
1
0
0
```

```
#include<iostream>
using namespace std;
const int N=110;
int n,a[N][N];
int gauss(){
    int r=1;//线代行阶梯化简从上到下定下来的行
    for(int c=1;c<=n;c++){
        int t=r;
        for(int i=r;i<=n;i++){
```

```

        if(a[i][c]){
            t=i;
            break;
        }
    }
    if(!a[t][c])continue;
    for(int i=c;i<=n+1;i++)swap(a[t][i],a[r][i]);
    for(int i=r+1;i<=n;i++){
        if(!a[i][c])continue;
        for(int j=n+1;j>=1;j--){
            a[i][j]^=a[r][j];
        }
    }
    r++;
}
if(r==n+1){
    for(int i=n-1;i>=1;i--){
        for(int j=i+1;j<=n;j++){
            a[i][n+1]^=a[i][j]&a[j][n+1];
        }
    }
    return 0;
}else if(r<n+1){
    for(int i=r;i<=n;i++){
        if(a[i][n+1]!=0)return 2;
    }
    return 1;
}
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n+1;j++){
            cin>>a[i][j];
        }
    }
    int t=gauss();
    if(t==0){//唯一解
        for(int i=1;i<=n;i++)cout<<a[i][n+1]<<endl;
    }else if(t==1){//无穷解
    {
        cout<<"Multiple sets of solutions"<<endl;
    }else{//无解
        cout<<"No solution"<<endl;
    }
}

```

```

    }
    return 0;
}

```

885. 求组合数 I(递推公式初始化组合数答案数组)

给定 n 组询问，每组询问给定两个整数 a, b ，请你输出 $C[a][b] \bmod (10^9+7)$ 的值。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含一组 a 和 b 。

输出格式

共 n 行，每行输出一个询问的解。

数据范围

$1 \leq n \leq 10000$,

$1 \leq b \leq a \leq 2000$

输入样例：

```

3
3 1
5 3
2 2

```

输出样例：

```

3
10
1

```

```

#include<iostream>
using namespace std;
const int N=2010,mod=1e9+7;
int n,C[N][N];
int main(){
    cin>>n;
    for(int i=0;i<N;i++)
        for(int j=0;j<=i;j++)

```

```

        if(!j)C[i][j]=1;
    else C[i][j]=(C[i-1][j]+C[i-1][j-1])%mod;
    for(int i=0;i<n;i++){
        int a,b;
        cin>>a>>b;
        cout<<C[a][b]<<endl;
    }
    return 0;
}

```

886. 求组合数 II(初始化阶乘和其逆元数组+组合公式求解)

给定 n 组询问，每组询问给定两个整数 a, b ，请你输出 $C[a][b] \bmod (10^9+7)$ 的值。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含一组 a 和 b 。

输出格式

共 n 行，每行输出一个询问的解。

数据范围

$1 \leq n \leq 10000$,

$1 \leq b \leq a \leq 10^5$

输入样例：

```

3
3 1
5 3
2 2

```

输出样例：

```

3
10
1

```

```

#include<iostream>
using namespace std;

```

```

const int N=100010,mod=1e9+7;
int fact[N],infact[N],n;
int qi_mi(int a,int b,int p){
    int res=1;
    while(b){
        if(b&1)res=1ll*res*a%p;
        b>>=1;
        a=1ll*a*a%p;
    }
    return res;
}
int main(){
    cin>>n;
    fact[0]=infact[0]=1;
    for(int i=1;i<N;i++){
        fact[i]=1ll*i*fact[i-1]%mod;
        infact[i]=1ll*infact[i-1]*qi_mi(i,mod-2,mod)%mod;
    }
    for(int i=0;i<n;i++){
        int a,b;
        cin>>a>>b;
        cout<<1ll*fact[a]*infact[a-b]%mod*infact[b]%mod<<endl;
    }
    return 0;
}

```

887. 求组合数 III(lucas 定理)

$$C_a^b \equiv C_{\lfloor \frac{a}{p} \rfloor}^{\lfloor \frac{b}{p} \rfloor} \cdot C_{a \bmod p}^{b \bmod p} \pmod{p}$$

lucas 定理:

给定 n 组询问，每组询问给定三个整数 a, b, p ，其中 p 是质数，请你输出 $C[a][b] \bmod p$ 的值。

输入格式

第一行包含整数 n 。

接下来 n 行，每行包含一组 a, b, p 。

输出格式

共 n 行，每行输出一个询问的解。

数据范围

$1 \leq n \leq 20$,

$1 \leq b \leq a \leq 10^{18}$,

$1 \leq p \leq 10^5$,

输入样例:

```
3
5 3 7
3 1 5
6 4 13
```

输出样例:

```
3
3
2
```

```
#include<iostream>
using namespace std;
const int N=100000;
typedef long long LL;
int n;
int qi_mi(int a,int b,int p){
    int res=1;
    while(b){
        if(b&1)res=1ll*res*a%p;
        b>>=1;
        a=1ll*a*a%p;
    }
    return res;
}
int C(int a,int b,int p){
    LL ans=1;
    for(int i=1,j=a;i<=b;i++,j--){
        ans=ans*j%p;
        ans=ans*qi_mi(i,p-2,p)%p;
    }
    return ans;
}
int lucas(LL a,LL b,int p){
    //a<p && b<p, 又因为p 是质数, 所以a,b 与p 互质, 所以(a-b) 也与p 互质, 因此可以直接快速
    幂求逆元
    if(a<p && b<p)return C(a,b,p);
```

```

        return 1ll*lucas(a/p,b/p,p)*C(a%p,b%p,p)%p;
    }
int main(){
    cin>>n;
    while(n--){
        LL a,b,c;
        cin>>a>>b>>c;
        cout<<lucas(a,b,c)<<endl;
    }
}

```

888. 求组合数 IV(高精度+计算质因子次数)

求 $n!$ 中某质因子 p 出现几次公式: $\text{cnt} = n/p + n/p^2 + n/p^3 + \dots$

思想:求解 $C[a][b]$, 其中 a, b 特别大, 需要使用高精度, 由于组合数公式中有乘除需要高精乘和高精除, 这里利用 $C[a][b]$ 必定是整数, 直接计算 $a!, b!, (a-b)!$ 的所有质因子的次数, 则 $C[a][b] = a! / (b! * (a-b)!)$, 即 $a!$ 中对应质因子次数减 $b!$ 和 $(a-b)!$ 对应的质因子次数, 最后直接利用高精乘将所有(质因子^次数)乘起来即可。

输入 a, b , 求 $C[a][b]$ 的值。

注意结果可能很大, 需要使用高精度计算。

输入格式

共一行, 包含两个整数 a 和 b 。

输出格式

共一行, 输出 $C[a][b]$ 的值。

数据范围

$1 \leq b \leq a \leq 5000$

输入样例:

```
5 3
```

输出样例:

```
10
```

```

#include<iostream>
#include<vector>
using namespace std;
const int N=5010;
int prime[N],cnt,st[N],sum[N];
void init(int n){

```

```

    st[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i])prime[cnt++]=i;
        for(int j=0;prime[j]<=n/i;j++){
            st[prime[j]*i]=1;
            if(i%prime[j]==0)break;
        }
    }
}

int getnum(int n,int p){
    int res=0;
    while(n){
        res+=n/p;
        n/=p;
    }
    return res;
}

vector<int>mul(vector<int>&A,int b){
    int t=0;
    vector<int>C;
    for(int i=0;i<A.size();i++){
        t+=A[i]*b;
        C.push_back(t%10);
        t/=10;
    }
    while(t){
        C.push_back(t%10);
        t/=10;
    }
    while(C.size()>1&&C.back()==0)C.pop_back();
    return C;
}

int main(){
    int a,b;
    cin>>a>>b;
    init(a);
    for(int i=0;i<cnt;i++){
        sum[i]=getnum(a,prime[i])-getnum(b,prime[i])-getnum(a-b,prime[i]);
    }
    vector<int>res;
    res.push_back(1);
    for(int i=0;i<cnt;i++){
        for(int j=0;j<sum[i];j++){
            res=mul(res,prime[i]);

```



```

    }
}
for(int i=res.size()-1;i>=0;i--)cout<<res[i];
}

```

889. 满足条件的 01 序列(卡特兰数+求组合数)

卡特兰数: $C_{2n}^n - C_{2n}^{n-1} = \frac{C_{2n}^n}{n+1}$

给定 n 个 0 和 n 个 1，它们将按照某种顺序排成长度为 $2n$ 的序列，求它们能排列成的所有序列中，能够满足任意前缀序列中 0 的个数都不少于 1 的个数的序列有多少个。

输出的答案对 10^9+7 取模。

输入格式

共一行，包含整数 n 。

输出格式

共一行，包含一个整数，表示答案。

数据范围

$1 \leq n \leq 10^5$

输入样例：

```
3
```

输出样例：

```
5
```

```

#include<iostream>
using namespace std;
const int N=200010,mod=1e9+7;
int fact[N],infact[N],n;
int qi_mi(int a,int b,int p){
    int res=1;
    while(b){
        if(b&1)res=1ll*res*a%p;
        b>>=1;
        a=1ll*a*a%p;
    }
    return res;
}

```

```

void init(int p){
    fact[0]=infact[0]=1;
    for(int i=1;i<N;i++){
        fact[i]=1ll*fact[i-1]*i%p;
        infact[i]=qi_mi(fact[i],p-2,p);
    }
}

int C(int a,int b,int p){
    return 1ll*fact[a]*infact[b]%p*infact[a-b]%p;
}

int main(){
    init(mod);
    cin>>n;
    cout<<1ll*C(2*n,n,mod)*qi_mi(n+1,mod-2,mod)%mod<<endl;
}

```

890. 能被整除的数(容斥原理)

$$\left| \bigcup_{i=1}^n A_i \right| = \sum_{i=1}^n |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + \cdots + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n-1} |A_1 \cap \cdots \cap A_n|.$$

注意:当集合个数为偶数前面符号,为奇数前面正号,代码中枚举每种情况用二进制,1表示集合被选,cnt表示选择集合个数用于判断正负。

1~n 能被 p 整除的个数:n/p 下取整

给定一个整数 n 和 m 个不同的质数 p1,p2,...,pm。

请你求出 1~ n 中能被 p1,p2,...,pm 中的至少一个数整除的整数有多少个。

输入格式

第一行包含整数 n 和 m。

第二行包含 m 个质数。

输出格式

输出一个整数，表示满足条件的整数的个数。

数据范围

1≤m≤16,

1≤n,pi≤10⁹

输入样例：

```
10 2
```

输出样例:

7

```
#include<iostream>
using namespace std;
const int N=20;
int p[N],n,m;
typedef long long LL;
int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++)cin>>p[i];
    int res=0;
    for(int i=1;i<=m;i++){
        int cnt=0;
        LL t=1;
        for(int j=0;j<m;j++){
            if(i>j&1)cnt++,t=t*p[j];
            if(t>n){
                t=-1;
                break;
            }
        }
        if(t!=-1){
            if(cnt%2)res+=n/t;
            else res-=n/t;
        }
    }
    cout<<res<<endl;
    return 0;
}
```

Nim 游戏相关理论

Nim 游戏定义

给定 N 堆物品，第 i 堆物品有 A_i 个。两名玩家轮流行动，每次可以任选一堆，取走任意多个物品，可把一堆取光，但不能不取。取走最后一件物品者获胜。两人都采取最优策略，问先手是否必胜。

我们把这种游戏称为 NIM 博弈。把游戏过程中面临的状态称为局面。整局游戏第一个行动的称为先手，第二个行动的称为后手。若在某一局面下无论采取何种行动，都会输掉游戏，则称该局面必败。

所谓采取最优策略是指，若在某一局面下存在某种行动，使得行动后对面面临必败局面，则优先

采取该行动。同时，这样的局面被称为必胜。我们讨论的博弈问题一般都只考虑理想情况，即两人都无失误，都采取最优策略行动时游戏的结果。

NIM 博弈不存在平局，只有先手必胜和先手必败两种情况。

定理：NIM 博弈先手必胜，当且仅当 $A_1 \oplus A_2 \oplus \dots \oplus A_n \neq 0$

公平组合游戏 ICG

若一个游戏满足：

由两名玩家交替行动；

在游戏进程的任意时刻，可以执行的合法行动与轮到哪名玩家无关；

不能行动的玩家判负；

则称该游戏为一个公平组合游戏。

NIM 博弈属于公平组合游戏，但棋类的棋类游戏，比如围棋，就不是公平组合游戏。因为围棋交战双方分别只能落黑子和白子，胜负判定也比较复杂，不满足条件 2 和条件 3。

有向图游戏

给定一个有向无环图，图中有一个唯一的起点，在起点上放有一枚棋子。两名玩家交替地把这枚棋子沿有向边进行移动，每次可以移动一步，无法移动者判负。该游戏被称为有向图游戏。

任何一个公平组合游戏都可以转化为有向图游戏。具体方法是，把每个局面看成图中的一个节点，并且从每个局面向沿着合法行动能够到达的下一个局面连有向边。

Mex 运算

设 S 表示一个非负整数集合。定义 $\text{mex}(S)$ 为求出不属于集合 S 的最小非负整数的运算，即：
 $\text{mex}(S) = \min\{x\}$, x 属于自然数，且 x 不属于 S

SG 函数

在有向图游戏中，对于每个节点 x ，设从 x 出发共有 k 条有向边，分别到达节点 y_1, y_2, \dots, y_k ，定义 $\text{SG}(x)$ 为 x 的后继节点 y_1, y_2, \dots, y_k 的 SG 函数值构成的集合再执行 $\text{mex}(S)$ 运算的结果，即：
 $\text{SG}(x) = \text{mex}(\{\text{SG}(y_1), \text{SG}(y_2), \dots, \text{SG}(y_k)\})$

特别地，整个有向图游戏 G 的 SG 函数值被定义为有向图游戏起点 s 的 SG 函数值，即 $\text{SG}(G) = \text{SG}(s)$ 。

有向图游戏的和 —— 模板题 AcWing 893. 集合-Nim 游戏

设 G_1, G_2, \dots, G_m 是 m 个有向图游戏。定义有向图游戏 G ，它的行动规则是任选某个有向图游戏 G_i ，并在 G_i 上行动一步。 G 被称为有向图游戏 G_1, G_2, \dots, G_m 的和。

有向图游戏的和的 SG 函数值等于它包含的各个子游戏 SG 函数值的异或和，即：

$$\text{SG}(G) = \text{SG}(G_1) \oplus \text{SG}(G_2) \oplus \dots \oplus \text{SG}(G_m)$$

定理

有向图游戏的某个局面必胜，当且仅当该局面对应节点的 SG 函数值大于 0。

有向图游戏的某个局面必败，当且仅当该局面对应节点的 SG 函数值等于 0。

891. Nim 游戏(经典 Nim 游戏)

定理 1: NIM 博弈先手必胜, 当且仅当 $A_1 \oplus A_2 \oplus \dots \oplus A_n \neq 0$

定理 1 理解: 镜像操作, 先手取多少, 后手模仿先手取多少, 如果 $A_1 \oplus A_2 \oplus \dots \oplus A_n = 0$, 后手是一定可以模仿先手的, 如此反复模仿最后后手取完先手一定没有可取的东西。

定理 2: 经典 Nim 游戏如果 $A_1 \oplus A_2 \oplus \dots \oplus A_n \neq 0$ 那么先手一定可以通过某种方式使 $A_1 \oplus A_2 \oplus \dots \oplus A_n = 0$, 后手不论怎么操作都会使 $A_1 \oplus A_2 \oplus \dots \oplus A_n \neq 0$ 。

定理 2 推导: 若 $A_1 \oplus A_2 \oplus \dots \oplus A_n = x \neq 0$, x 二进制最高位的 1 是第 k 位, 那么至少有 1 个 A_i 第 k 位是 1, $A_i \oplus x < A_i$ 。所以我们必可以在原本 A_i 中拿走 $A_i - A_i \oplus x$ (因为 > 0), 原本 A_i 剩下 $A_i \oplus x$, 剩下的重新带到上面式子异或里面 $x \oplus x$ 一定是 0, 证毕。

给定 n 堆石子, 两位玩家轮流操作, 每次操作可以从任意一堆石子中拿走任意数量的石子 (可以拿完, 但不能不拿), 最后无法进行操作的人视为失败。

问如果两人都采用最优策略, 先手是否必胜。

输入格式

第一行包含整数 n 。

第二行包含 n 个数字, 其中第 i 个数字表示第 i 堆石子的数量。

输出格式

如果先手方必胜, 则输出 **Yes**。

否则, 输出 **No**。

数据范围

$1 \leq n \leq 10^5$,

$1 \leq \text{每堆石子数} \leq 10^9$

输入样例:

```
2
2 3
```

输出样例:

```
Yes
```

```
#include<iostream>
using namespace std;
int n,x,ans;
int main(){
    cin>>n;
    while(n--){
```

```

        cin>>x;
        ans^=x;
    }
    if(ans)cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
    return 0;
}

```

892. 台阶-Nim 游戏

此时我们需要将奇数台阶看做一个经典的 Nim 游戏，如果先手时奇数台阶上的值的异或值为 0，则先手必败，反之必胜

证明：

先手时，如果奇数台阶异或非 0，根据经典 Nim 游戏，先手总有一种方式使奇数台阶异或为 0，于是先手留了奇数台阶异或为 0 的状态给后手

于是轮到后手：

①当后手移动偶数台阶上的石子时，先手只需将对手移动的石子继续移到下一个台阶，这样奇数台阶的石子相当于没变，于是留给后手的又是奇数台阶异或为 0 的状态

②当后手移动奇数台阶上的石子时，留给先手的奇数台阶异或非 0，根据经典 Nim 游戏，先手总能找出一种方案使奇数台阶异或为 0

因此无论后手如何移动，先手总能通过操作把奇数异或为 0 的情况留给后手，当奇数台阶全为 0 时，只留下偶数台阶上有石子。

（核心就是：先手总是把奇数台阶异或为 0 的状态留给对面，即总是将必败态交给对面）

因为偶数台阶上的石子要想移动到地面，必然需要经过偶数次移动，又因为奇数台阶全 0 的情况是留给后手的，因此先手总是可以将石子移动到地面，当将最后一个（堆）石子移动到地面时，后手无法操作，即后手失败。

因此如果先手时奇数台阶上的值的异或值为非 0，则先手必胜，反之必败！

现在，有一个 n 级台阶的楼梯，每级台阶上都有若干个石子，其中第 i 级台阶上有 a_i 个石子($i \geq 1$)。

两位玩家轮流操作，每次操作可以从任意一级台阶上拿若干个石子放到下一级台阶中（不能不拿）。

已经拿到地面上的石子不能再拿，最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

输入格式

第一行包含整数 n 。

第二行包含 n 个整数，其中第 i 个整数表示第 i 级台阶上的石子数 a_i 。

输出格式

如果先手方必胜，则输出 **Yes**。

否则，输出 **No**。

数据范围

$1 \leq n \leq 10^5$,

$1 \leq a_i \leq 10^9$

输入样例：

```
3
2 1 3
```

输出样例：

```
Yes
```

```
#include<iostream>
using namespace std;
int n,x,ans;
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>x;
        if(i%2==1)ans^=x;
    }
    if(ans)cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
}
```

893. 集合-Nim 游戏

$SG(x) = \text{mex}(\{SG(y_1), SG(y_2), \dots, SG(y_k)\})$ y_i 是 x 一次能到的所有局面

有向图游戏的某个局面(G_i)必胜，当且仅当该局面对应节点的 SG 函数值大于 0。

有向图游戏的某个局面(G_i)必败，当且仅当该局面对应节点的 SG 函数值等于 0。

$SG(G_1) \oplus SG(G_2) \oplus \dots \oplus SG(G_m) \neq 0$ 先手必胜。

回想经典 Nim 游戏,经典 Nim 游戏如果 $A_1 \oplus A_2 \oplus \dots \oplus A_n \neq 0$ 那么先手一定可以通过某种方式使 $A_1 \oplus A_2 \oplus \dots \oplus A_n = 0$, 后手不论怎么操作都会使 $A_1 \oplus A_2 \oplus \dots \oplus A_n \neq 0$ 。(注意,这里先手的操作只是操作 A_i 里面的一堆。)

而 sg 函数则是对于复杂的操作,有限的操作比如该题拿石子个数有限制,就可以分析每堆石子的局面,例如下图是一堆 10 个石子,拿法限制只能拿 2 和 5 个石子的图,其中蓝数字是石子个数,红数字是对应的 sg 值。

输入样例:

```
2
2 5
3
2 4 7
```

输出样例:

```
Yes
```

```
#include<iostream>
#include<cstring>
#include<unordered_set>
using namespace std;
const int N=110,M=100010;
int s[N],f[M],k,n,ans;
int sg(int x){
    if(f[x]!=-1)return f[x];
    unordered_set<int>g;
    for(int i=0;i<k;i++){
        if(x-s[i]>=0)g.insert(sg(x-s[i]));
    }
    for(int i=0;;i++){
        if(!g.count(i))return f[x]=i;
    }
}
int main(){
    memset(f,-1,sizeof f);
    cin>>k;
    for(int i=0;i<k;i++)cin>>s[i];
    cin>>n;
    for(int i=0;i<n;i++){
        int x;
        cin>>x;
        ans^=sg(x);
    }
    if(ans)cout<<"Yes"<<endl;
    else cout<<"No"<<endl;

    return 0;
}
```

894. 拆分-Nim 游戏

思想类似集合-Nim 游戏。首先把初始的 n 堆石子分为 n 个局面单独解决后异或即可判断答案。

对应每堆石子算 sg 值与前面不太相同,因为一堆石子的局面按题目意思会变成两堆石子一起的局面,而两堆石子的 sg 分别是两堆石子分别的局面,通过两堆石子 sg 异或就成了两堆石子一起的局面。又因为每堆石子拆分方法是规模更小的两堆,所以暴力枚举当前一堆石子局面能到的所有两堆石子一起的局面计算 mex 即为当前石子局面的 sg 值。

给定 n 堆石子,两位玩家轮流操作,每次操作可以取走其中的一堆石子,然后放入两堆规模更小的石子(新堆规模可以为 0,且两个新堆的石子总数可以大于取走的那堆石子数),最后无法进行操作的人视为失败。

问如果两人都采用最优策略,先手是否必胜。

输入格式

第一行包含整数 n 。

第二行包含 n 个整数,其中第 i 个整数表示第 i 堆石子的数量 a_i 。

输出格式

如果先手方必胜,则输出 **Yes**。

否则,输出 **No**。

数据范围

$1 \leq n, a_i \leq 100$

输入样例:

```
2
2 3
```

输出样例:

```
Yes
```

```
#include<iostream>
#include<cstring>
#include<unordered_set>
using namespace std;
const int N=110;
int f[N],n,ans,x;
int sg(int x){
    if(f[x]!=-1)return f[x];
    unordered_set<int>S;
```

```

    for(int i=0;i<x;i++)
        for(int j=0;j<=i;j++)S.insert(sg(i)^sg(j));
    for(int i=0;;i++){
        if(!S.count(i))return f[x]=i;
    }
}

int main(){
    memset(f,-1,sizeof f);
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>x;
        ans^=sg(x);
    }
    if(ans)cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
}

```

2.01 背包问题

有 N 件物品和一个容量是 V 的背包。每件物品只能使用一次。

第 i 件物品的体积是 v_i ，价值是 w_i 。

求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

输出最大价值。

输入格式

第一行两个整数， N ， V ，用空格隔开，分别表示物品数量和背包容积。

接下来有 N 行，每行两个整数 v_i, w_i ，用空格隔开，分别表示第 i 件物品的体积和价值。

输出格式

输出一个整数，表示最大价值。

数据范围

$0 < N, V \leq 1000$

$0 < v_i, w_i \leq 1000$

输入样例

```

4 5
1 2
2 4

```

3 4

4 5

输出样例:

8

二维

```
#include<iostream>
using namespace std;
const int N=1010;
int v[N],w[N],n,m,f[N][N];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>v[i]>>w[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            f[i][j]=max(f[i][j],f[i-1][j]);
            if(j>=v[i])f[i][j]=max(f[i][j],f[i-1][j-v[i]]+w[i]);
        }
    }
    cout<<f[n][m]<<endl;
}
```

滚动数组降维,第一种使用两个数组滚动赋值好理解,第二种直接使用一个数组

```
#include<iostream>
#include<cstring>
using namespace std;
const int N=1010;
int v[N],w[N],n,m,f[N],t[N];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>v[i]>>w[i];
    }
    for(int i=1;i<=n;i++){
        memcpy(f,t,sizeof t);
        for(int j=1;j<=m;j++){
            if(j>=v[i])f[j]=max(f[j],t[j-v[i]]+w[i]);
        }
        memcpy(t,f,sizeof f);
    }
    cout<<f[m]<<endl;
}
```

```
}
```

理解方式:状态转移方程 $f[i][j]=\max(f[i-1][j],f[i-1][j-v[i]]+w[i])$

若是顺序遍历 j 。当求解到 $f[j]$ 时,说明 j 前面的已经求解过了,即 j 前面的 f 已经是第 i 层的 f 了,不是第 $i-1$ 层的 f 了,但是转移方程用的是 $i-1$ 层的且在 j 之前的 f ,不可取。

若是逆序遍历 j 。当求解到 $f[j]$ 时,说明 j 后面的已经求解过了,即 j 后面的 f 已经是第 i 层的 f 了,不是第 $i-1$ 层的 f 了,但是 j 前面的 f 还没有求解,是 $i-1$ 层的 f ,又因为转移方程是用的 $i-1$ 层的 j 前面的 f ,可取。

```
#include<iostream>
using namespace std;
const int N=1010;
int f[N],n,m,v[N],w[N];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
    for(int i=1;i<=n;i++){
        for(int j=m;j>=v[i];j--){
            f[j]=max(f[j],f[j-v[i]]+w[i]);
        }
    }
    cout<<f[m]<<endl;
    return 0;
}
```

3. 完全背包问题

有 N 种物品和一个容量是 V 的背包,每种物品都有无限件可用。

第 i 种物品的体积是 v_i ,价值是 w_i 。

求解将哪些物品装入背包,可使这些物品的总体积不超过背包容量,且总价值最大。

输出最大价值。

输入格式

第一行两个整数, N,V ,用空格隔开,分别表示物品种数和背包容积。

接下来有 N 行,每行两个整数 v_i,w_i ,用空格隔开,分别表示第 i 种物品的体积和价值。

输出格式

输出一个整数,表示最大价值。

数据范围

$0<N,V\leq 1000$

$0 < v_i, w_i \leq 1000$

输入样例

```
4 5
1 2
2 4
3 4
4 5
```

输出样例:

```
10
```

二维

```
#include<iostream>
using namespace std;
const int N=1010;
int f[N][N],v[N],w[N],n,m;
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    //f[i][j]=f[i-1][j]+f[i-1][j-v[i]]+f[i-1][j-2*v[i]]+...
    //f[i][j-v[i]]=f[i-1][j-v[i]]+f[i-1][j-2*v[i]]+...
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>v[i]>>w[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            f[i][j]=f[i-1][j];
            if(j>=v[i])f[i][j]=max(f[i][j],f[i][j-v[i]]+w[i]);
        }
    }
    cout<<f[n][m]<<endl;
    return 0;
}
```

滚动数组降维，理解方式同理 01 背包问题的滚动数组。

```
#include<iostream>
using namespace std;
const int N=1010;
int f[N],n,m,v[N],w[N];
int main(){
```

```

cin>>n>>m;
for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
for(int i=1;i<=n;i++){
    for(int j=v[i];j<=m;j++){
        f[j]=max(f[j],f[j-v[i]]+w[i]);
    }
}
cout<<f[m]<<endl;
return 0;
}

```

4. 多重背包问题 I

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 s_i 件，每件体积是 v_i ，价值是 w_i 。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。

输出最大价值。

输入格式

第一行两个整数， N ， V ，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 v_i, w_i, s_i ，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

输出格式

输出一个整数，表示最大价值。

数据范围

$0 < N, V \leq 100$

$0 < v_i, w_i, s_i \leq 100$

输入样例

```

4 5
1 2 3
2 4 1
3 4 3
4 5 2

```

输出样例：

```

#include<iostream>
using namespace std;
const int N=110;
int n,m,w[N],v[N],s[N],f[N][N];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>v[i]>>w[i]>>s[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            for(int k=0;k*v[i]<=j&&k<=s[i];k++){
                f[i][j]=max(f[i][j],f[i-1][j-v[i]*k]+w[i]*k);
            }
        }
    }
    cout<<f[n][m]<<endl;
    return 0;
}

```

5. 多重背包问题 II(二进制优化)

思想:将多重背包拆分为 01 背包做,但是如果全部拆开(例如 7 拆成 7 个 1),复杂度没有变化。01 背包思想是这个物品选或不选,和二进制每位是 1 或是 0 恰好对应。因此我们可以使用二进制优化,例如 7,我们只需要 1,2,4。实际上分别是 3 位二进制的权值,通过每位选是 1 不选是 0,7 以内的数都可以通过某种选法选出来。因此利用二进制也完成了所有情况的考虑。

注意:这里二进制优化例如 10,需要 1,2,4,3 而不是 1,2,4,8 因为后者可以选择 0-15,但是要求是只能选择物品个数 0-10。1,2,4 可以选择 0-7, 1,2,4,3 按前述选择方法每个加 3 可以选择 3-10,既然 0-7 和 3-10 都可以通过 1,2,4,3 选择,即可以选择 0-10 而不能选择其他,符合题意。

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 s_i 件,每件体积是 v_i ,价值是 w_i 。

求解将哪些物品装入背包,可使物品体积总和不超过背包容量,且价值总和最大。

输出最大价值。

输入格式

第一行两个整数, N , V , 用空格隔开, 分别表示物品种数和背包容积。

接下来有 N 行, 每行三个整数 v_i, w_i, s_i , 用空格隔开, 分别表示第 i 种物品的体积、价值和数量。

输出格式

输出一个整数，表示最大价值。

数据范围

$0 < N \leq 1000$

$0 < V \leq 2000$

$0 < v_i, w_i, s_i \leq 2000$

提示：

本题考查多重背包的二进制优化方法。

输入样例

```
4 5
1 2 3
2 4 1
3 4 3
4 5 2
```

输出样例：

```
10
```

```
#include<iostream>
using namespace std;
const int N=2010;
int f[N],v[13000],w[13000],n,m,cnt,t[N];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        int k=1,a,b,c;
        cin>>a>>b>>c;
        while(k<=c){
            cnt++;
            v[cnt]=k*a;
            w[cnt]=k*b;
            c-=k;
            k*=2;
        }
        if(c>0){
            cnt++;
            v[cnt]=c*a;
            w[cnt]=c*b;
        }
    }
    for(int i=1;i<=m;i++){
        int t=0;
        for(int j=0;j<=t;j++){
            if(f[j]+v[cnt]>f[t]){
                t=j+cnt;
            }
        }
    }
    cout<<f[m]<<endl;
}
```

```

    }
}
n=cnt;
for(int i=1;i<=n;i++){
    for(int j=1;j<=m;j++){
        t[j]=f[j];
        if(j>=v[i])f[j]=max(f[j],t[j-v[i]]+w[i]);
    }
}
cout<<f[m]<<endl;
}

```

9. 分组背包问题

有 N 组物品和一个容量是 V 的背包。

每组物品有若干个，同一组内的物品最多只能选一个。

每件物品的体积是 v_{ij} ，价值是 w_{ij} ，其中 i 是组号， j 是组内编号。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

输入格式

第一行有两个整数 N, V ，用空格隔开，分别表示物品组数和背包容量。

接下来有 N 组数据：

- 每组数据第一行有一个整数 S_i ，表示第 i 个物品组的物品数量；
- 每组数据接下来有 S_i 行，每行有两个整数 v_{ij}, w_{ij} ，用空格隔开，分别表示第 i 个物品组的第 j 个物品的体积和价值；

输出格式

输出一个整数，表示最大价值。

数据范围

$0 < N, V \leq 100$

$0 < S_i \leq 100$

$0 < v_{ij}, w_{ij} \leq 100$

输入样例

```
3 5
```

```
2
1 2
2 4
1
3 4
1
4 5
```

输出样例:

```
8
```

```
#include<iostream>
using namespace std;
const int N=110;
int f[N][N],n,m,v[N][N],w[N][N],s[N];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>s[i];
        for(int j=1;j<=s[i];j++){
            cin>>v[i][j]>>w[i][j];
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            f[i][j]=f[i-1][j];
            for(int k=1;k<=s[i];k++){
                if(j>=v[i][k])f[i][j]=max(f[i][j],f[i-1][j-v[i][k]]+w[i][k]);
            }
        }
    }
    cout<<f[n][m]<<endl;
    return 0;
}
```

241. 楼兰图腾(树状数组)

树状数组使用范围:单点值修改,区间和查询

常见题型:给一个数组,问数组中每个数左边和右边比它大的数或小的数有多少个
在完成了分配任务之后,西部 314 来到了楼兰古城的西部。

相传很久以前这片土地上(比楼兰古城还早)生活着两个部落,一个部落崇拜尖刀(∇),一个部落崇拜铁锹(\wedge),他们分别用 ∇ 和 \wedge 的形状来代表各自部落的图腾。

西部 314 在楼兰古城的下面发现了一幅巨大的壁画,壁画上被标记出了 n 个点,经测量发现这 n 个点的水平位置和竖直位置是两两不同的。

西部 314 认为这幅壁画所包含的信息与这 n 个点的相对位置有关,因此不妨设坐标分别为 $(1,y_1),(2,y_2),\dots,(n,y_n)$,其中 $y_1\sim y_n$ 是 1 到 n 的一个排列。

西部 314 打算研究这幅壁画中包含着多少个图腾。

如果三个点 $(i,y_i),(j,y_j),(k,y_k)$ 满足 $1\leq i<j<k\leq n$ 且 $y_i>y_j,y_j<y_k$,则称这三个点构成 ∇ 图腾;

如果三个点 $(i,y_i),(j,y_j),(k,y_k)$ 满足 $1\leq i<j<k\leq n$ 且 $y_i<y_j,y_j>y_k$,则称这三个点构成 \wedge 图腾;

西部 314 想知道,这 n 个点中两个部落图腾的数目。

因此,你需要编写一个程序来求出 ∇ 的个数和 \wedge 的个数。

输入格式

第一行一个数 n 。

第二行是 n 个数,分别代表 y_1, y_2, \dots, y_n 。

输出格式

两个数,中间用空格隔开,依次为 ∇ 的个数和 \wedge 的个数。

数据范围

对于所有数据, $n\leq 200000$,且输出答案不会超过 int64 。

$y_1\sim y_n$ 是 1 到 n 的一个排列。

输入样例:

```
5
1 5 3 2 4
```

输出样例:

```
3 4
```

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N=200010;
int a[N],n,tr[N],lmin[N],lmax[N],rmin[N],rmax[N];
int lowbit(int x){
```

```

        return x&-x;
    }
    void add(int x,int c){
        for(int i=x;i<=n;i+=lowbit(i))tr[i]+=c;
    }
    int query(int x){
        int res=0;
        for(int i=x;i>0;i-=lowbit(i))res+=tr[i];
        return res;
    }
    int main(){
        cin>>n;
        for(int i=1;i<=n;i++)cin>>a[i];
        for(int i=1;i<=n;i++){
            lmin[i]=query(a[i]-1);
            lmax[i]=query(n)-query(a[i]);
            add(a[i],1);
        }
        memset(tr,0,sizeof tr);
        for(int i=n;i>=1;i--){
            rmin[i]=query(a[i]-1);
            rmax[i]=query(n)-query(a[i]);
            add(a[i],1);
        }
        LL ans1=0,ans2=0;
        for(int i=1;i<=n;i++){
            ans1+=1ll*lmax[i]*rmax[i];
            ans2+=1ll*lmin[i]*rmin[i];
        }
        cout<<ans1<<" "<<ans2<<endl;
    }
}

```

1057. 股票买卖 IV(dp-状态机模型- 2 状态)

给定一个长度为 N 的数组，数组中的第 i 个数字表示一个给定股票在第 i 天的价格。

设计一个算法来计算你能获取的最大利润，你最多可以完成 k 笔交易。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。一次买入卖出合为一笔交易。

输入格式

第一行包含整数 N 和 k ，表示数组的长度以及你可以完成的最大交易笔数。

第二行包含 N 个不超过 10000 的非负整数，表示完整的数组。

输出格式

输出一个整数，表示最大利润。

数据范围

$1 \leq N \leq 10^5$,

$1 \leq k \leq 100$

输入样例 1:

```
3 2
2 4 1
```

输出样例 1:

```
2
```

输入样例 2:

```
6 2
3 2 6 5 0 3
```

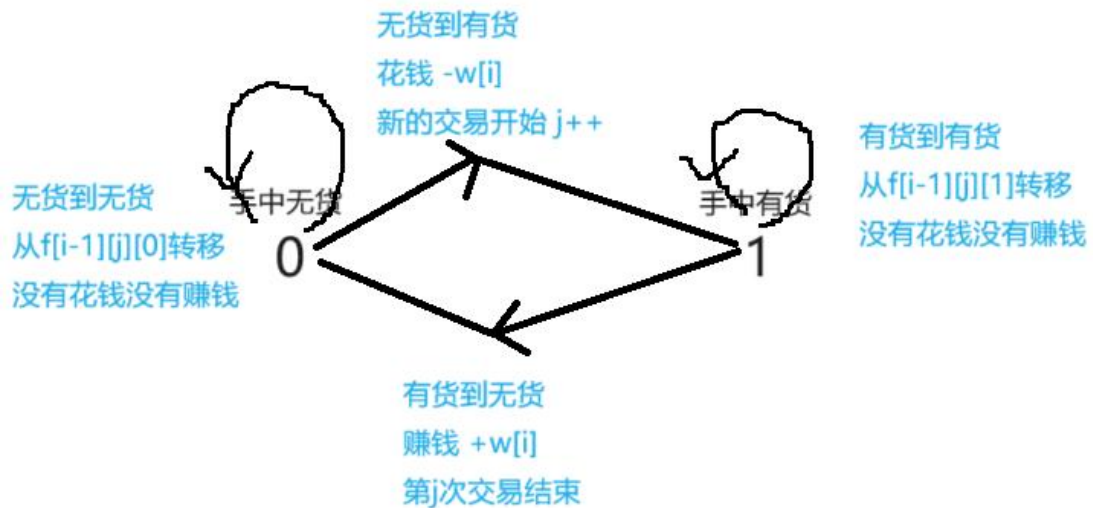
输出样例 2:

```
7
```

样例解释

样例 1: 在第 1 天 (股票价格 = 2) 的时候买入, 在第 2 天 (股票价格 = 4) 的时候卖出, 这笔交易所能获得利润 = $4 - 2 = 2$ 。

样例 2: 在第 2 天 (股票价格 = 2) 的时候买入, 在第 3 天 (股票价格 = 6) 的时候卖出, 这笔交易所能获得利润 = $6 - 2 = 4$ 。随后, 在第 5 天 (股票价格 = 0) 的时候买入, 在第 6 天 (股票价格 = 3) 的时候卖出, 这笔交易所能获得利润 = $3 - 0 = 3$ 。共计利润 $4 + 3 = 7$ 。



```
#include<bits/stdc++.h>
using namespace std;
int f[100010][110][2],w[100010],n,k;
//f[i][j][k] 前i个股票,第j次交易,此时状态为k    k=0 手中无货    k=1 手中有货
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++)cin>>w[i];
    memset(f,-0x3f,sizeof f);
    for(int i=0;i<=n;i++)f[i][0][0]=0;
    //所有状态都只能由没有交易过,且手中无货的状态转移,因此除这个状态外全为负无穷这样
    //就从其他非法状态转移不过来
    for(int i=1;i<=n;i++){
        for(int j=1;j<=k;j++){
            f[i][j][0]=max(f[i-1][j][0],f[i-1][j][1]+w[i]);
            f[i][j][1]=max(f[i-1][j][1],f[i-1][j-1][0]-w[i]);
        }
    }
    int maxx=0;
    for(int i=1;i<=k;i++)maxx=max(maxx,f[n][i][0]);
    cout<<maxx<<endl;
    return 0;
}
```

1058. 股票买卖 V(dp-状态机模型- 3 状态)

给定一个长度为 N 的数组，数组中的第 i 个数字表示一个给定股票在第 i 天的价格。

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）：

- 你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

- 卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

输入格式

第一行包含整数 N ，表示数组长度。

第二行包含 N 个不超过 10000 的正整数，表示完整的数组。

输出格式

输出一个整数，表示最大利润。

数据范围

$1 \leq N \leq 10^5$

输入样例：

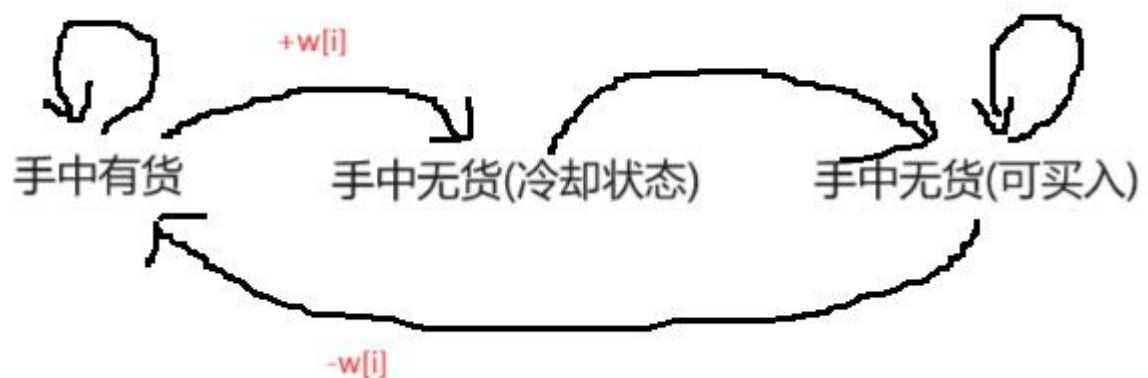
```
5
1 2 3 0 2
```

输出样例：

```
3
```

样例解释

对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出], 第一笔交易可得利润 $2-1=1$, 第二笔交易可得利润 $2-0=2$, 共得利润 $1+2=3$ 。



```
#include<bits/stdc++.h>
using namespace std;
const int N=100010;
int f[N][3],w[N],n;
/*
f[i][j] 第i天所处状态为j j=0 手中有货 j=1 手中无货(冷却) j=2 手中无货(可买入)
```


初始状态是可以买股票的, 因此是手中无货(可买入), 所以 $f[0][2]=0$, 然后其他置负无穷, 这样就不能从非法操作转移

```
*/  
  
int main(){  
    cin>>n;  
    memset(f,-0x3f,sizeof f);  
    f[0][2]=0;  
    for(int i=1;i<=n;i++)cin>>w[i];  
    for(int i=1;i<=n;i++){  
        f[i][0]=max(f[i-1][0],f[i-1][2]-w[i]);  
        f[i][1]=f[i-1][0]+w[i];  
        f[i][2]=max(f[i-1][2],f[i-1][1]);  
    }  
    cout<<max(f[n][1],f[n][2])<<endl;  
    return 0;  
}
```

91. 最短 Hamilton 路径(dp-状压 dp)

给定一张 n 个点的带权无向图, 点从 $0 \sim n-1$ 标号, 求起点 0 到终点 $n-1$ 的最短 Hamilton 路径。

Hamilton 路径的定义是从 0 到 $n-1$ 不重不漏地经过每个点恰好一次。

输入格式

第一行输入整数 n 。

接下来 n 行每行 n 个整数, 其中第 i 行第 j 个整数表示点 i 到 j 的距离 (记为 $a[i,j]$)。

对于任意的 x,y,z , 数据保证 $a[x,x]=0$, $a[x,y]=a[y,x]$ 并且 $a[x,y]+a[y,z] \geq a[x,z]$ 。

输出格式

输出一个整数, 表示最短 Hamilton 路径的长度。

数据范围

$1 \leq n \leq 20$

$0 \leq a[i,j] \leq 10^7$

输入样例:

```
5  
  
0 2 4 5 1  
2 0 6 5 3  
4 6 0 8 3
```

```
5 5 8 0 5
1 3 3 5 0
```

输出样例:

```
18
```

```
#include<iostream>
#include<cstring>
using namespace std;
const int N=1<<20,M=20;
int f[N][M],n,w[M][M];
// f[i][j] 当前已经用过点的状态 i (二进制表示, 每位相应点已经用过), 处于 j 位
int main(){
    cin>>n;
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)cin>>w[i][j];
    memset(f,0x3f,sizeof f);
    f[1][0]=0;//最开始处于位置 0 点, 状态 i 二进制 0000...0001, 处于 j=0 位
    for(int i=0;i<1<<n;i++){
        for(int j=0;j<n;j++){
            if(!(i>>j&1))continue;//因为处于 j, 那么 i 状态里面 j 点必须已经用过
            for(int k=0;k<n;k++){
                //由 k 点转移到 j 点过来, 那么 k 点的状态 i 中必须没有 j 点, 但是必须要有 k
                //点, 因为还没转移过来
                if(!((i-(1<<j))>>k&1))continue;
                f[i][j]=min(f[i][j],f[i-(1<<j)][k]+w[k][j]);
            }
        }
    }
    cout<<f[(1<<n)-1][n-1];
    return 0;
}
```

285. 没有上司的舞会(树形 dp,打家劫舍算法)

Ural 大学有 N 名职员, 编号为 $1 \sim N$ 。

他们的关系就像一棵以校长为根的树, 父节点就是子节点的直接上司。

每个职员有一个快乐指数, 用整数 H_i 给出, 其中 $1 \leq i \leq N$ 。

现在要召开一场周年庆宴会, 不过, 没有职员愿意和直接上司一起参会。

在满足这个条件的前提下，主办方希望邀请一部分职员参会，使得所有参会职员的快乐指数总和最大，求这个最大值。

输入格式

第一行一个整数 N 。

接下来 N 行，第 i 行表示 i 号职员的快乐指数 H_i 。

接下来 $N-1$ 行，每行输入一对整数 L,K ，表示 K 是 L 的直接上司。（注意一下，后一个数是前一个数的父节点，不要搞反）。

输出格式

输出最大的快乐指数。

数据范围

$1 \leq N \leq 6000$,

$-128 \leq H_i \leq 127$

输入样例：

```
7
1
1
1
1
1
1
1
1 3
2 3
6 4
7 4
4 5
3 5
```

输出样例：

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
int n;
int h[6500],e[6500],ne[6500],a[6500],idx,f[6500][2];
void dfs(int u){
    f[u][1]+=a[u];
    for(int i=h[u];i!=-1;i=ne[i]){
        int t=e[i];
        dfs(t);
        f[u][0]+=max(f[t][0],f[t][1]);
        f[u][1]+=f[t][0];
    }
}
void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
void solve(){
    cin>>n;
    map<int,int>mp;
    memset(h,-1,sizeof h);
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<n;i++){
        int a,b;
        cin>>a>>b;
        add(b,a);
        mp[a]++;
    }
    int root=1;
    while(mp.count(root))root++;
    dfs(root);
    cout<<max(f[root][0],f[root][1])<<endl;
}
signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    //cin>>t;
    while(t--)solve();
    return 0;
}

```

245. 你能回答这些问题吗(线段树,连续子区间和最大,单点修改)

给定长度为 N 的数列 A ，以及 M 条指令，每条指令可能是以下两种之一：

1. $1\ x\ y$ ，查询区间 $[x,y]$ 中的最大连续子段和。

2. $2\ x\ y$ ，把 $A[x]$ 改成 y 。

对于每个查询指令，输出一个整数表示答案。

输入格式

第一行两个整数 N,M 。

第二行 N 个整数 $A[i]$ 。

接下来 M 行每行 3 个整数 k,x,y ， $k=1$ 表示查询（此时如果 $x>y$ ，请交换 x,y ）， $k=2$ 表示修改。

输出格式

对于每个查询指令输出一个整数表示答案。

每个答案占一行。

数据范围

$N \leq 500000, M \leq 100000$,

$-1000 \leq A[i] \leq 1000$

输入样例：

```
5 3
1 2 -3 4 5
1 2 3
2 2 -1
1 3 2
```

输出样例：

```
2
-1
```

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

```

using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
int n,m;
const int N=500010;
int a[N];
struct tr{
    int l,r,tmax,lmax,rmax,sum;
}tr[4*N];
void pushup(int u){
    tr[u].tmax=max(max(tr[u<<1].tmax,tr[u<<1|1].tmax),tr[u<<1|1].lmax+tr[u<<1].
rmax);
    tr[u].sum=tr[u<<1].sum+tr[u<<1|1].sum;
    tr[u].lmax=max(tr[u<<1].lmax,tr[u<<1].sum+tr[u<<1|1].lmax);
    tr[u].rmax=max(tr[u<<1|1].rmax,tr[u<<1|1].sum+tr[u<<1].rmax);
}
void pushup(struct tr &u,struct tr &ul,struct tr &ur){
    u.tmax=max(max(ul.tmax,ur.tmax),ur.lmax+ul.rmax);
    u.sum=ul.sum+ur.sum;
    u.lmax=max(ul.lmax,ul.sum+ur.lmax);
    u.rmax=max(ur.rmax,ur.sum+ul.rmax);
}
void build(int u,int l,int r){
    if(l==r)tr[u]={l,r,a[l],a[l],a[l],a[l]};
    else{
        tr[u]={l,r};
        int mid=l+r>>1;
        build(u<<1,l,mid),build(u<<1|1,mid+1,r);
        pushup(u);
    }
}
void modify(int u,int x,int v){
    if(tr[u].l==tr[u].r&&tr[u].l==x)tr[u]={x,x,v,v,v,v};
    else{
        int mid=tr[u].l+tr[u].r>>1;
        if(x<=mid)modify(u<<1,x,v);
        else modify(u<<1|1,x,v);
        pushup(u);
    }
}
struct tr query(int u,int l,int r){
    if(tr[u].l>=l&&tr[u].r<=r)return tr[u];
    else{

```

```

        int mid=tr[u].l+tr[u].r>>1;
        if(r<=mid)return query(u<<1,l,r);
        else if(l>mid)return query(u<<1|1,l,r);
        else{
            struct tr left=query(u<<1,l,r),right=query(u<<1|1,l,r);
            struct tr res;
            pushup(res,left,right);
            return res;
        }
    }
}

void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    build(1,1,n);
    while(m--){
        int b,c,d;
        cin>>b>>c>>d;
        if(b==1){// 查询
            if(c>d)swap(c,d);
            cout<<query(1,c,d).tmax<<endl;
        }else{// 修改
            modify(1,c,d);
        }
    }
}

signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    //cin>>t;
    while(t--){solve();}
    return 0;
}

```

3165. 第一类斯特林数--圆排序

第一类斯特林数（斯特林轮换数） $s(n,k)$ 表示将 n 个两两不同的元素，划分为 k 个非空圆排列的方案数。

现在，给定 n 和 k ，请你求方案数。

圆排列定义：圆排列是排列的一种，指从 n 个不同元素中取出 m ($1 \leq m \leq n$) 个不同的元素排列成一个环形，既无头也无尾。两个圆排列相同当且仅当所取元素的个数相同并且元素取法一致，在环上的排列顺序一致。

输入格式

两个整数 n 和 k 。

输出格式

输出一个整数表示划分方案数。

答案对 10^9+7 取模。

数据范围

$1 \leq k \leq n \leq 1000$

输入样例：

```
3 2
```

输出样例：

```
3
```

相对第二类斯特林数就像是子集变成圆排序,也就是圆排序内要分顺序,1 2 3 和 1 3 2 不一样
 $f[i][j]$ 表示把 i 个数按第一类斯特林数分配到 j 个盒子里面,注意 j 个盒子没有顺序区分,且每个盒子内部有顺序标识,也就是单个盒子内部符合圆排序

递推式: $f[i][j]=f[i-1][j-1]+(i-1)*f[i-1][j];$

首先现在要放第 i 个数,那么之前肯定都只放了 $i-1$ 个数,也就是从那里转移,然后分为第 i 个数是放新盒子里面还是旧盒子里面,如果是新盒子里面,那么之前的 $i-1$ 个数就应该放到前 $j-1$ 个盒子,这样才能腾出第 j 个新盒子,如果是旧盒子里面,那么之前 $i-1$ 个数就应该全部盒子都放,也就是 j 个盒子,这样这些盒子才能变成旧盒子。但是在放新盒子的时候,因为盒子之前没有区别,所以数量就是 $f[i-1][j-1]$,在放旧盒子的时候, $f[i-1][j]$ 子问题也保证了 $i-1$ 分配到 j 个盒子盒子是无区别的,但是由于旧盒子有 j 个,且每个盒子内部顺序是有区别的,第 i 个数总共有 $i-1$ 种插法(因为每个盒子是圆排序), $(i-1)*f[i-1][j]$

假设盒子有区别又会怎么样? 这个问题不属于斯特林数问题,在此只是一起探讨区分不同的区别。

递推式: $f[i][j]=(i-1)*(f[i-1][j-1]+f[i-1][j]);$

如果有区别,那么定义 $f[i][j]$ 就变成了 i 个数放进 j 个盒子,盒子有区别,那么相对于上面盒子无区别,如果放入新盒子(第 j 个盒子是新盒子),注意前 $j-1$ 个盒子都满足所有盒子都有区别,也就是满足全排列,所以你不能只放入 j 就完事了,因为你的 $f[i][j]$ 也要保证前 j 个盒子有区别,因此你目前的 j 和 $j-1$ 交换, $j-2$ 交换.....盒子的顺序不同,都是有区别的,但是前 $j-1$ 个盒子不用互相交换了,因为子问题 $f[i-1][j-1]$ 已经保证了 $i-1$ 放进 $j-1$ 个盒子且盒子有区别了,因此相对于上面就要多乘 $i-1$ 。但是放到旧盒子却是跟之前是一样的,因为旧盒子之前就全部满足了有区别,然后你一个盒子插入的情况都算了,所有旧盒子还是有区别的。说白了,就是旧盒子的子问题已经保证了所有 j 盒子有区别,因此父问题就不用管了,但是新盒子由于是你新加的,还要自己维护。

无符号Stirling数有如下性质:

$$\textcircled{1} s_u(0,0) = 1$$

$$\textcircled{2} s_u(n,0) = 0$$

$$\textcircled{3} s_u(n,n) = 1$$

$$\textcircled{4} s_u(n,1) = (n-1)!$$

$$\textcircled{5} s_u(n,n-1) = C(n,2)$$

$$\textcircled{6} s_u(n,2) = (n-1)! \cdot \sum_{i=1}^{n-1} \frac{1}{i}$$

$$\textcircled{7} s_u(n,n-2) = 2 \cdot C(n,3) + 3 \cdot C(n,4)$$

$$\textcircled{8} \sum_{k=0}^n s_u(n,k) = n! \text{ 证明可令升阶函数中的 } x=1, \text{ 比较两边系数。}$$

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
const int N=1010,mod=1e9+7;
int n,k;
int f[N][N];
void solve(){
    cin>>n>>k;
    f[0][0]=1;//特殊规定
    for(int i=1;i<=n;i++){
        for(int j=1;j<=i;j++){//j 从1 开始是因为 s(i,0)=0 i>0 时
            f[i][j]=f[i-1][j-1]+(i-1)*f[i-1][j];
            f[i][j]%=mod;
        }
    }
    cout<<f[n][k]<<endl;
}
signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    //cin>>t;
    while(t--)<math>solve();</math>
    return 0;
}
```

3166. 第二类斯特林数--子集

第二类斯特林数（斯特林子集数） $S(n,k)$ 表示将 n 个两两不同的元素，划分为 k 个非空子集的方案数。

现在，给定 n 和 k ，请你求方案数。

输入格式

两个整数 n 和 k 。

输出格式

输出一个整数表示划分方案数。

答案对 10^9+7 取模。

数据范围

$1 \leq k \leq n \leq 1000$

输入样例：

```
3 2
```

输出样例：

```
3
```

相对第一类斯特林数就像是圆排序变成子集，也就是子集内不分顺序，只要有就行，第二类斯特林数相对第一还有通项公式，也就是可以直接求，但是还是建议递推求，递推的复杂度一定是更小的。某些详细解释还可以参考第一类斯特林数的红色字体。

$$\textcircled{1} S(n, 0) = 0^n$$

$$\textcircled{2} S(n, 1) = 1$$

$$\textcircled{3} S(n, n) = 1$$

$$\textcircled{4} S(n, 2) = 2^{n-1} - 1$$

$$\textcircled{5} S(n, n-1) = C(n, 2)$$

$$\textcircled{6} S(n, n-2) = C(n, 3) + 3 \cdot C(n, 4)$$

$$\textcircled{7} S(n, 3) = \frac{1}{2}(3^{n-1} + 1) - 2^{n-1}$$

$$\textcircled{8} S(n, n-3) = C(n, 4) + 10 \cdot C(n, 5) + 15 \cdot C(n, 6)$$

$$\textcircled{9} \sum_{k=0}^n S(n, k) = B_n, \quad B_n \text{ 是贝尔数。}$$

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
const int N=1010,mod=1e9+7;
int n,k;
int f[N][N];
void solve(){
    cin>>n>>k;
    f[0][0]=1;//特殊规定
    for(int i=1;i<=n;i++){
        for(int j=1;j<=i;j++){//j 从1 开始是因为 S(i,0)=0 i>0 时
            f[i][j]=f[i-1][j-1]+j*f[i-1][j];
            f[i][j]%=mod;
        }
    }
    cout<<f[n][k]<<endl;
}
signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    //cin>>t;
    while(t--){solve();}
    return 0;
}
```

1270. 数列区间最大值(st 表)

输入一串数字，给你 M 个询问，每次询问就给你两个数字 X,Y ，要求你说出 X 到 Y 这段区间内的最大数。

输入格式

第一行两个整数 N,M 表示数字的个数和要询问的次数；

接下来一行为 N 个数；

接下来 M 行，每行都有两个整数 X,Y 。

输出格式

输出共 M 行，每行输出一个数。

数据范围

$$1 \leq N \leq 10^5$$

$$1 \leq M \leq 10^6$$

$$1 \leq X \leq Y \leq N$$

数列中的数字均不超过 $2^{31}-1$

输入样例：

```
10 2
3 2 4 5 6 8 1 2 9 7
1 4
3 8
```

输出样例：

```
5
8
```

ST 表（Sparse Table，稀疏表）是用于解决 **可重复贡献问题** 的数据结构。

查询一次**可重复贡献问题**(区间最大,最小,gcd)复杂度为 $O(1)$, 只可查询不可像线段树一样修改

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
int n,m;
int a[100010],f[100010][32],lg2[100010];
```

```

//f[i][j]以 i 开始长度为2^j 的区间中最大值
int query(int l,int r){
    //用重叠的区间O(1)解决查询,若不用重叠区间,把区间长度拆分二进制一个一个算则O(Logn)
    //直接用小于等于区间长度的最大2 的幂次方长度解决,因为求的最大值,区间重合没影响
    int len=r-l+1,k=lg2[len];
    return max(f[l][k],f[r-(1<<k)+1][k]);
}
void solve(){
    cin>>n>>m;
    //预处理 log2(x) 下取整的值
    for(int i=2;i<=n;i++)lg2[i]=lg2[i/2]+1;
    //初始化 dp 中最初的每个a[i]
    for(int i=1;i<=n;i++)cin>>a[i],f[i][0]=a[i];
    //状态转移得先枚举每个区间的长度
    for(int j=1;j<=lg2[n];j++){//倍增枚举长度,lg2 下取整,若带小数则也只能下取整因为
        是倍增
        for(int i=1;(i+(1<<j)-1)<=n;i++){//每个区间的右边界不得>n
            //以 i 开始长度为2^j 可以从区间 i-i+2^j-1 中对半分,每个区间长度2^j-1 一样
            且满足2 的幂次方
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
        }
    }
    while(m--){
        int b,c;
        cin>>b>>c;
        cout<<query(b,c)<<endl;
    }
}
signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    // cin>>t;
    while(t--)solve();
    return 0;
}

```

有向图 dfs 判环

有向图 $st[u]$ 有三个状态:

- 0: 表示节点 u 尚未被访问。
- 1: 表示节点 u 正在被访问 (在递归调用栈中)。
- 2: 表示节点 u 已经被完全处理完毕。(无向图 dfs 判环没有这个状态,主要是因为无向图边是单向的,即使是连通的环,但也有可能不是有向图的环,环内边的方向有不一致的情况)

```

bool ok=true;
function<void(string)> dfs=[&](string u)->void{

```

```

    if(st[u])return ;
    st[u]=1;
    for(int i=h[u];i!=-1;i=ne[i]){
        string s=e[i];
        if(st[s]==1)ok=false;
        else if(st[s]==0)dfs(s);
    }
    st[u]=2;
};

```

1819. 序列中不同最大公约数的数目

给你一个由正整数组成的数组 `nums` 。

数字序列的 **最大公约数** 定义为序列中所有整数的共有约数中的最大整数。

例如，序列 `[4,6,16]` 的最大公约数是 2 。

数组的一个 **子序列** 本质是一个序列，可以通过删除数组中的某些元素（或者不删除）得到。

例如，`[2,5,10]` 是 `[1,2,1,2,4,1,5,10]` 的一个子序列。

计算并返回 `nums` 的所有 非空 子序列中 不同 最大公约数的 数目 。

示例 1：

Subsequence	GCD
[6]	6
[10]	10
[3]	3
[6,10]	2
[6,3]	3
[10,3]	1
[6,10,3]	1

输入： `nums = [6,10,3]` 输出： 5 解释： 上图显示了所有的非空子序列与各自的 最大公约数。

不同的最大公约数为 6 、 10 、 3 、 2 和 1 。

示例 2：

输入: nums = [5,15,40,5,6]输出: 7

1 <= nums.length <= 10⁵

1 <= nums[i] <= 2 * 10⁵

思路:最终 gcd 一定小于所有数的最大值,因为 gcd 一定越取越小,然后我们**暴力枚举 1-maxx**,枚举的是这个最大公约数 i 且是否能通过某种操作得到,内部**枚举这个 i 的倍数**,因为只有 i 的倍数取 gcd 才有可能*是 i*,同时确保 i 的倍数要在原数组中,这样才能表示是从原数组中通过 gcd 而来的,复杂度**不是 O(n²)**,外层循环加上内层循环是**调和级数**,复杂度是 **O(nlogn)**

```
class Solution {
public:
    int countDifferentSubsequenceGCDs(vector<int> &nums) {
        int ans = 0, mx = *max_element(nums.begin(), nums.end());
        bool has[mx + 1]; memset(has, 0, sizeof(has));
        for (int x : nums) has[x] = true;
        for (int i = 1; i <= mx; ++i) {
            int g = 0; // 0 和任何数 x 的最大公约数都是 x
            for (int j = i; j <= mx && g != i; j += i) // 枚举 i 的倍数 j
                if (has[j]) // 如果 j 在 nums 中
                    g = gcd(g, j); // 更新最大公约数
            if (g == i) ++ans; // 找到一个答案
        }
        return ans;
    }
};
```

365. 水壶问题(裴蜀定理)

有两个水壶,容量分别为 x 和 y 升。水的供应是无限的。确定是否有可能使用这两个壶准确得到 target 升。

你可以:

- 1.装满任意一个水壶
- 2.清空任意一个水壶
- 3.将水从一个水壶倒入另一个水壶,直到接水壶已满,或倒水壶已空。

示例 1:

输入: x = 3,y = 5,target = 4 输出: true 解释: 按照以下步骤操作,以达到总共 4 升水:

1. 装满 5 升的水壶(0, 5)。
2. 把 5 升的水壶倒进 3 升的水壶,留下 2 升(3, 2)。
3. 倒空 3 升的水壶(0, 2)。

4. 把 2 升水从 5 升的水壶转移到 3 升的水壶(2, 0)。
5. 再次加满 5 升的水壶(2, 5)。
6. 从 5 升的水壶向 3 升的水壶倒水直到 3 升的水壶倒满。5 升的水壶里留下了 4 升水(3, 4)。
7. 倒空 3 升的水壶。现在, 5 升的水壶里正好有 4 升水(0, 4)。

参考: 来自著名的 "Die Hard"

示例 2:

输入: x = 2, y = 6, target = 5 输出: false

$1 \leq x, y, \text{target} \leq 10^3$

思路: 观察题目会有一个结论 1, 不会出现两个水壶都不满的情况

1. 装满任意一个水壶(+x 或 +y) 若装满之前不为 0, 则不会+x 或+y, 但是根据结论 1, 此时另一个水壶要么是 0(空), 要么是 x 或 y(满), 那么装满之后要么是 x, 要么是 x+y, 也是可以表示成 $a*x+b*y$
2. 清空任意一个水壶(-x 或 -y)若清空前不是 x 或 y(满), 则不会-x 或-y, 但不影响结果, 原因同上
3. 将水从一个水壶倒入另一个水壶, 直到接水壶已满, 或倒水壶已空。(总量不变)

很明显, 操作后最终的总量可以表示为 $a*x+b*y$, 而根据裴蜀定理 $ax + by = m$ 有解当且仅当 m 是 d 的倍数。 $d=\text{gcd}(x,y)$, a 和 b 是任何整数

「裴蜀定理」的内容为: 对于不全为零的任意整数 a 和 b, 记 $g=\text{gcd}(a,b)$, 其中 $\text{gcd}(a,b)$ 为 a 和 b 的最大公约数, 则对于任意整数 x 和 y 都满足 $ax+by$ 是 g 的倍数, 特别地, 存在整数 x 和 y 满足 $ax+by=g$ 。

```
class Solution {
public:
    bool canMeasureWater(int x, int y, int target) {
        if(target>x+y)return false;
        else{
            int g=gcd(x,y);
            if(target%g==0)return true;
            else return false;
        }
    }
};
```

P1452 求凸包+旋转卡壳

题目描述

给定平面上 n 个点, 求凸包直径。

输入格式

第一行一个正整数 n。

接下来 n 行，每行两个整数 x,y ，表示一个点的坐标。保证所有点的坐标两两不同。

输出格式

输出一行一个整数，表示答案的平方。

输入输出样例

输入

```
4
0 0
0 1
1 1
1 0
```

输出

```
2
```

说明/提示

【数据范围】

对于 100% 的数据， $2 \leq n \leq 50000$ ， $|x|, |y| \leq 10^4$ 。

思路:题目只给了 n 个点,得先把凸包求出来([andrew 算法](#)),再求凸包直径([旋转卡壳算法](#))

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
int n;
PII p[50010];
vector<PII>s;//vector 模拟栈,因为要取顶上两个元素而且不一定删除
int cross(PII a,PII b,PII c){//向量a到b叉乘向量a到c的值
    int a_to_b_x=b.first-a.first,a_to_b_y=b.second-a.second;
```

```

        int a_to_c_x=c.first-a.first,a_to_c_y=c.second-a.second;
        return a_to_b_x*a_to_c_y-a_to_b_y*a_to_c_x;
    }
    int dis(PII a,PII b){//距离平方和
        return (a.first-b.first)*(a.first-b.first)+(a.second-b.second)*(a.s
econd-b.second);
    }
    void andrew(){//求凸包
        sort(p+1,p+1+n);//首先按x,y 升序排序
        //求下凸包 必定包含1 和n 点(因为排过序)
        for(int i=1;i<=n;i++){
            //首先保底栈中有两个元素,s[s.size()-2]到s[s.size()-1]形成
            的向量,如果p[i]在这个向量方向的右边或共线则出栈
            //为什么? 因为p[i]在右边,把s[s.size()-2]删了连
            s[s.size()-1]和p[i]才有更大潜力使凸包更凸
            //直到在它的左边才迫不得已只能选他
            while(s.size()-2>=1&&cross(s[s.size()-2],s[s.size()-1],p[
i])<=0)s.pop_back();
            s.push_back(p[i]);
        }
        //求上凸包,求的点不重不漏
        int k=s.size();
        for(int i=n-1;i>=1;i--){//1 和 n 在求下凸包必进栈,因为排序的性质导致必
        是下凸包的点,所以上凸包1 会多入一次栈,最后删除即可
            //为啥s.size()-2>=k+1,因为排序导致上面求下凸包n 点必是凸包的
            点,所以求上凸包不能删掉下凸包求出来的点,最多只能把n 点的后一个点删了留下n 点
            while(s.size()-2>=k+1&&cross(s[s.size()-2],s[s.size()-1],
p[i])<=0)s.pop_back();
            s.push_back(p[i]);
        }
        s.pop_back();//删除重复的1 点
    }
    int rotate(){//旋转卡壳,求凸包最长直径,也就是凸包最长对角线
        //andrew 函数求出了所有的凸包逆时针点坐标
        int res=0;//最长直径
        for(int i=0,j=1;i<s.size();i++){
            while(cross(s[i],s[(i+1)%s.size()],s[j])<cross(s[i],s[(i+1)
%s.size()],s[(j+1)%s.size()])))j=(j+1)%s.size();
            //通过叉乘算 s[i],s[i+1],s[j] 形成三角形面积,前两个为底,面积最大,
            此时以前两个为底的三角形高最大,因此对于凸包 s[j]点,最长对角线是(s[j],s[i])或
            (s[j],s[i+1])
            res=max(res,max(dis(s[i],s[j]),dis(s[(i+1)%s.size()],s[j])))
        }
    }

```

```

        return res;
    }
    void solve(){
        cin>>n;
        for(int i=1;i<=n;i++){
            auto &[x,y]=p[i];
            cin>>x>>y;
        }
        andrew();
        cout<<rotate()<<endl;
    }
    signed main(){
        int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
        //    cin>>t;
        while(t--){solve();}
        return 0;
    }
}

```

nlogn 求 1-n 范围的每个数的所有质因子

优化复杂度:**nlogn**

暴力复杂度:已知 867.分解质因数对一个数复杂度是 \sqrt{n} ,那么 n 个数则是 $n\sqrt{n}$

思想:先通过**欧拉筛法**用 $O(n)$ 的时间筛查 1-n 区间的所有质数,同时记录每个数的**最小质因子**,最后通过**最小质因子**慢慢求出所有质因子,步骤如下

1. 对于数 X

2. 获取 min_prime[X]为X目前的最小质因子(记录所有该值则是最初X的所有质因子,每个质因子有几次就会出现几次,需要去重自行去重)

3. $X=X / \text{min_prime}[X]$

4. 如果 $X=1$,退出,否则回到步骤 2

```

for(int i=2;i<=n;i++){
    if(!st[i]){
        prime[cnt++]=i;
        min_prime[i]=i;
    }
    for(int j=0;prime[j]<=n/i;j++){
        st[i*prime[j]]=1;
        min_prime[i*prime[j]]=prime[j];
        if(i%prime[j]==0)break;
    }
}
vector mp(nums.size()+10,vector<int>(0));

```

```

for(int i=0;i<nums.size();i++){
    int x=nums[i];
    while(x!=1){
        int min_pr=min_prime[x];
        mp[i].push_back(min_pr);
        x/=min_pr;
    }
    sort(mp[i].begin(),mp[i].end());
    mp[i].erase( unique(mp[i].begin(),mp[i].end()),mp[i].end());
    for(auto i : mp[i])cout<<i<<" ";
    cout<<endl;
}

```

243. 一个简单的整数问题 2(线段树,懒标记)

给定一个长度为 N 的数列 A ，以及 M 条指令，每条指令可能是以下两种之一：

1. **C l r d**，表示把 $A[l], A[l+1], \dots, A[r]$ 都加上 d 。

2. **Q l r**，表示询问数列中第 $l \sim r$ 个数的和。

对于每个询问，输出一个整数表示答案。

输入格式

第一行两个整数 N, M 。

第二行 N 个整数 $A[i]$ 。

接下来 M 行表示 M 条指令，每条指令的格式如题目描述所示。

输出格式

对于每个询问，输出一个整数表示答案。

每个答案占一行。

数据范围

$1 \leq N, M \leq 10^5$

$|d| \leq 10000$

$|A[i]| \leq 10^9$

输入样例：

```

10 5
1 2 3 4 5 6 7 8 9 10

```

```

Q 4 4
Q 1 10
Q 2 4
C 3 6 3
Q 2 4

```

输出样例:

```

4
55
9
15

```

注意:`spread(u)`表示更新 `u` 结点的子结点,意味着当前 `u` 已经更新了懒标记,如果 `u` 有 `lazy` 值,这个值是子节点需要的 `lazy`,而不包括自己

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
typedef struct node{
    int l,r,sum,lazy;
}node;
void solve(){
    int n,q;
    cin>>n>>q;
    vector<int>a(n+10);
    vector<node>tr(4*n);
    for(int i=1;i<=n;i++)cin>>a[i];
    function<void(int)> up=[&](int u){
        tr[u].sum=tr[u<<1].sum+tr[u<<1|1].sum;
    };
    function<void(int,int,int)> build=[&](int u,int l,int r){
        if(l==r)tr[u]={l,l,a[l],0};
        else{
            int mid=l+r>>1;
            build(u<<1,l,mid),build(u<<1|1,mid+1,r);
            tr[u].l=l,tr[u].r=r;
            up(u);
        }
    }
}

```

```

};
function<void(int)> spread=[&](int u){
    if(tr[u].lazy){
        tr[u<<1].sum+=tr[u].lazy*(tr[u<<1].r-tr[u<<1].l+1);
        tr[u<<1|1].sum+=tr[u].lazy*(tr[u<<1|1].r-tr[u<<1|1].l+1);
        tr[u<<1].lazy+=tr[u].lazy;
        tr[u<<1|1].lazy+=tr[u].lazy;
        tr[u].lazy=0;
    }
};
function<void(int,int,int,int)> modify=[&](int u,int l,int r,int c){
    if(tr[u].l>=l&&tr[u].r<=r){
        tr[u].lazy+=c;
        tr[u].sum+=c*(tr[u].r-tr[u].l+1);
    }else{
        spread(u);
        int mid=tr[u].l+tr[u].r>>1;
        if(l<=mid)modify(u<<1,l,r,c);
        if(r>mid)modify(u<<1|1,l,r,c);
        up(u);
    }
};
function<int(int,int,int)> query=[&](int u,int l,int r){
    if(tr[u].l>=l&&tr[u].r<=r){
        return tr[u].sum;
    }else{
        spread(u);
        int mid=tr[u].l+tr[u].r>>1,sum=0;
        if(l<=mid)sum+=query(u<<1,l,r);
        if(r>mid)sum+=query(u<<1|1,l,r);
        return sum;
    }
};
build(1,1,n);
while(q--){
    char c;
    int l,r,d;
    cin>>c>>l>>r;
    if(c=='Q'){
        cout<<query(1,l,r)<<endl;
    }else{
        cin>>d;
        modify(1,l,r,d);
    }
}

```

```

    }
}
signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    //    cin>>t;
    while(t--){solve();}
    return 0;
}

```

255. 第 K 小数(主席树)

给定长度为 N 的整数序列 A ，下标为 $1 \sim N$ 。

现在要执行 M 次操作，其中第 i 次操作为给出三个整数 li,ri,ki ，求 $A[li],A[li+1],\dots,A[ri]$ (即 A 的下标区间 $[li,ri]$)中第 ki 小的数是多少。

输入格式

第一行包含两个整数 N 和 M 。

第二行包含 N 个整数，表示整数序列 A 。

接下来 M 行，每行包含三个整数 li,ri,ki ，用以描述第 i 次操作。

输出格式

对于每次操作输出一个结果，表示在该次操作中，第 k 小的数的数值。

每个结果占一行。

数据范围

$N \leq 10^5, M \leq 10^4, |A[i]| \leq 10^9$

输入样例：

```

7 3
1 5 2 6 3 7 4
2 5 3
4 4 1
1 7 3

```

输出样例：

```

5

```

6

3

注意:

1.主席树不同于线段树,线段树 `struct` 结点中的 `l` 和 `r` 表示该结点的对于题目数组的左右边界下标,主席树 `struct` 结点中的 `l` 和 `r` 表示该结点的左右儿子指针,且并未存储该结点表示的值域范围,而是通过函数传参每次传 `l` 和 `r` 计算 $\text{mid}=(l+r)/2$ 推出来,节省了内存,但是传参一定要传 `l` 和 `r` 的值域。

2.由于主席树需要记录多个历史版本,而每个历史版本最多改 $\log n$ 个节点(也就是从根结点走到子节点的某条路径),因此每个版本只用多记录 $\log n$ 个节点,同时必须记录每个版本的根节点 `root[i]` 才能在之后访问,由于主席树记录多个历史版本,导致一个节点可能有多个父节点,因此不能像线段树一样静态开点(父节点为 `u`,左儿子 `u*2`,右儿子 `u*2+1`),因为一个左儿子 `u*2` 只能对应一个父节点 `u`。所以主席树需要动态开点,也就是类似于邻接表或 `trie` 树用 `idx` 分配节点。

3.该题 `a[i]` 范围可高达 $1e9$,由 1 知主席树的每个结点也能表示某个值域(只是没有存,节约内存),那么必然子结点会有 $1e9$ 个,因为会有值域 1 到 1,2 到 2..... $1e9$ 到 $1e9$ 。所以要离散化到 $1e5$,这样空间就不会爆了。

不封装版本

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
typedef struct{
    int l,r,cnt;//这个l和r是指向子节点的指针,不在是线段树的边界
}node;
void solve(){
    int n,q;
    cin>>n>>q;
    vector<int>a(n+10),alls,root(n+10);//root是每个历史版本的根结点,root[0]是建的空树
    for(int i=1;i<=n;i++)cin>>a[i],alls.push_back(a[i]);
    sort(alls.begin(),alls.end());
    alls.erase(unique(alls.begin(),alls.end()),alls.end());
    auto find=[&](int x){//主席树函数传参的l和r是值域(tr[u].l和r是指针),所以得先离散化,不然得开1e9的值域
        //默认最开始的下标为1,因为之后求l-r版本会用类似前缀和s[r]-s[l-1],保证l-1>=0
        return lower_bound(alls.begin(),alls.end(),x)-alls.begin()+1;
    };
    vector<node>tr(4*n+n*((int)log2(n)+1));//主席树,空树版本还是跟线段树4*n 然后插入n次,每次改log个点
    int idx=0;//分配主席树结点,动态开点,因为历史版本的根结点也会连空树版本的节点
    //所以一个节点就可能有多多个父亲,不能再线段树静态开点
    function<void(int&,int,int)>build=[&](int &u,int l,int r){
        u++;idx;
```



```

        if(l==r)return ;
        int mid=l+r>>1;
        build(tr[u].l,l,mid),build(tr[u].r,mid+1,r);
    };
    //x 上一个版本的节点,y 这个版本的节点,需要分配新节点,先拷贝旧节点包括左右儿子的指针
    //然后看插入的值在左还是右,改一个指针指向新节点继续递归
    //对第 i 个数 a[i] 单独插入并建立一个历史版本的主席树,需要完全拷贝上一个版本的插入点
    在左或在右的 log2 个点,并且每个新结点的 cnt 要加 1
    function<void(int,int&,int,int,int)>insert=[&](int x,int&y,int l,int r,int
c){
        y++;
        tr[y]=tr[x];//先拷贝前一个版本和他同等的节点,注意儿子指针也拷贝了,然后下面判
左和右,只改其中一指针
        tr[y].cnt++;
        if(l==r)return ;
        int mid=l+r>>1;
        if(c<=mid)insert(tr[x].l,tr[y].l,l,mid,c);
        else insert(tr[x].r,tr[y].r,mid+1,r,c);
    };
    //此处 x 和 y 表示第 x 和第 y 版本的根结点,含义是下标 x+1 到 y 中第 k 小数
    function<int(int,int,int,int,int,int)>query=[&](int x,int y,int l,int r,int k){
        if(l==r)return l;
        int mid=l+r>>1;
        //x+1 到 y 版本的主席树中,值域为 l-r 的结点的左儿子(l-mid)有 cnt 个
        int cnt=tr[tr[y].l].cnt-tr[tr[x].l].cnt;
        if(k<=cnt)return query(tr[x].l,tr[y].l,l,mid,k);
        else return query(tr[x].r,tr[y].r,mid+1,r,k-cnt);
    };
    build(root[0],1,alls.size());//建空树,离散化过,alls 中的数全变成了从 1 开始的下标
    for(int i=1;i<=n;i++){
        insert(root[i-1],root[i],1,alls.size(),find(a[i]));
    }
    while(q--){
        int l,r,c;
        cin>>l>>r>>c;
        int ans=query(root[l-1],root[r],1,alls.size(),c);
        cout<<alls[ans-1]<<endl;
    }
}
signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    // cin>>t;
    while(t--){solve();

```

```

    return 0;
}

封装版本 求第 k 小数和前 k 小数和
typedef struct{
    int l,r,cnt,sum;
}node;
class zhu_xi_tree{
public:
    /*
        zhu_xi_tree tr(n,a)    n: 需要查找第k 小的数组的长度n a:vector<int>(n+10) 先
        存a 里面然后传参
        tr.q(l,r,c) 下标1-n    返回pair<int,int> {区间l-r 的第c 小数, 区间l-r 的前c
        小数之和}
    */
    vector<int>a,alls,root;
    vector<node>tr;
    int idx;
    zhu_xi_tree(int n,vector<int>a):a(a),root(n+10),tr(4*n+n*((int)log2(n)+
1)){
        for(int i=1;i<=n;i++)alls.push_back(a[i]);
        sort(alls.begin(),alls.end());
        alls.erase(unique(alls.begin(),alls.end()),alls.end());
        idx=0;
        build(root[0],1,alls.size());
        for(int i=1;i<=n;i++){
            insert(root[i-1],root[i],1,alls.size(),find(a[i]));
        }
    }
    int find(int x){
        return lower_bound(alls.begin(),alls.end(),x)-alls.begin()+1;
    };
    void build(int &u,int l,int r){
        u==++idx;
        if(l==r)return ;
        int mid=l+r>>1;
        build(tr[u].l,l,mid),build(tr[u].r,mid+1,r);
    };
    void insert(int x,int&y,int l,int r,int c){
        y==++idx;
        tr[y]=tr[x];//先拷贝前一个版本和他同等的节点,注意儿子指针也拷贝了,然
        后下面判左和右,只改其中一指针
        tr[y].cnt++;
        tr[y].sum+=alls[c-1];
    };
};

```

```

        if(l==r)return ;
        int mid=l+r>>1;
        if(c<=mid)insert(tr[x].l,tr[y].l,l,mid,c);
        else insert(tr[x].r,tr[y].r,mid+1,r,c);
    };

    pair<int,int> query(int x,int y,int l,int r,int k,int &sum){//返回{第k
        小数编号,最后一个没加上的数的sum}
        if(l==r)return {l,tr[y].sum-tr[x].sum};
        int mid=l+r>>1;
        //x+1 到 y 版本的主席树中, 值域为 l-r 的结点的左儿子(l-mid)有 cnt 个
        int cnt=tr[tr[y].l].cnt-tr[tr[x].l].cnt;
        if(k<=cnt)return query(tr[x].l,tr[y].l,l,mid,k,sum);
        else {
            sum+=tr[tr[y].l].sum-tr[tr[x].l].sum;
            return query(tr[x].r,tr[y].r,mid+1,r,k-cnt,sum);
        }
    };

    pair<int,int> q(int l,int r,int c){//返回{第k 小数, 前k 小数之和}
        int sum=0;
        auto [ans,sum2]=query(root[l-1],root[r],1,alls.size(),c,sum);
        return {alls[ans-1],sum+sum2};
    }
};

```

1169. 糖果(差分约束)

幼儿园里有 N 个小朋友，老师现在想要给这些小朋友们分配糖果，要求每个小朋友都要分到糖果。

但是小朋友们也有嫉妒心，总是会提出一些要求，比如小明不希望小红分到的糖果比他的多，于是在分配糖果的时候，老师需要满足小朋友们的 K 个要求。

幼儿园的糖果总是有限的，老师想知道他至少需要准备多少个糖果，才能使得每个小朋友都能够分到糖果，并且满足小朋友们所有的要求。

输入格式

输入的第一行是两个整数 N,K 。

接下来 K 行，表示分配糖果时需要满足的关系，每行 3 个数字 X,A,B 。

- 如果 $X=1$ ，表示第 A 个小朋友分到的糖果必须和第 B 个小朋友分到的糖果一样多。
- 如果 $X=2$ ，表示第 A 个小朋友分到的糖果必须少于第 B 个小朋友分到的糖果。
- 如果 $X=3$ ，表示第 A 个小朋友分到的糖果必须不少于第 B 个小朋友分到的糖果。
- 如果 $X=4$ ，表示第 A 个小朋友分到的糖果必须多于第 B 个小朋友分到的糖果。
- 如果 $X=5$ ，表示第 A 个小朋友分到的糖果必须不多于第 B 个小朋友分到的糖果。

小朋友编号从 1 到 N。

输出格式

输出一行，表示老师至少需要准备的糖果数，如果不能满足小朋友们的所有要求，就输出 -1。

数据范围

$1 \leq N \leq 10^5$,

$1 \leq K \leq 10^5$,

$1 \leq X \leq 5$,

$1 \leq A, B \leq N$,

输入数据完全随机。

输入样例：

```
5 7
1 1 2
2 3 2
4 4 1
3 4 5
5 4 5
2 3 5
4 5 1
```

输出样例：

```
11
```

注意 1:

(1) 求不等式组的可行解

源点需要满足的条件：从源点出发，一定可以走到所有的边。

<1>步骤： 以最短路(求每个不等式所有变量的最大值)为例说明步骤

[1] 先将每个不等式 $x_i \leq x_j + c_k$ 转化成一条从 x_j 走到 x_i , 长度为 c_k 的一条边

[2] 找一个超级源点，使得该源点一定可以遍历到所有边

[3] 从源点做一遍单源最短路径

结果 1：如果存在负环，则原不等式组一定无解

结果 2：如果没有负环，则 $dist[i]$ 就是原不等式组的一个可行解

<2>步骤： 以最长路(求每个不等式所有变量的最小值)为例说明步骤

[1] 先将每个不等式 $x_i \geq x_j + c_k$ 转化成一条从 x_j 走到 x_i , 长度为 c_k 的一条边

[2] 找一个超级源点，使得该源点一定可以遍历到所有边

[3] 从源点做一遍单源最短路径

结果 1：如果存在正环，则原不等式组一定无解

结果 2: 如果没有正环, 则 `dist[i]` 就是原不等式组的一个可行解

(2) 如何求最大值或最小值, 这里的最值指的是每个变量的最值

结论: 如果求的是**最小值**, 则应该求**最长路**; 如果求的是**最大值**, 则应该求**最短路**;

注意 2:

Spfa 算法在有正负环的情况下, 栈比队列更快, 但是如果没有正负环的一般情况下, 队列更快。

由于差分约束必判环, 用栈一般更快

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f
typedef pair<int,int> PII;
/*
X=1 A>=B B>=A
X=2 B>=A+1
X=3 A>=B
X=4 A>=B+1
X=5 B>=A
分到的糖果xi>=1
题目是求至少即所有xi的最小值之和, 所以跑最长路, 正环无解
*/
void solve(){
    int n,q,idx=0;
    cin>>n>>q;
    vector<int>dis(n+10,-1e18),st(n+10),ne(2*q+n+10),e(2*q+n+10),h(n+10,-1),w(2
*q+n+10),cnt(n+10);
    auto add=[&](int a,int b,int c){
        e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
    };
    while(q--){
        int x,a,b;
        cin>>x>>a>>b;
        if(x==1)add(b,a,0),add(a,b,0);
        else if(x==2)add(a,b,1);
        else if(x==3)add(b,a,0);
        else if(x==4)add(b,a,1);
        else if(x==5)add(a,b,0);
    }
    for(int i=1;i<=n;i++)add(0,i,1); // 建立超级原点, 能访问到所有边, 也是确定绝对关系的, 不然就只有相对关系
    auto spfa=[&]() {
        dis[0]=0;
        st[0]=1;
        stack<int>s;
        // 在有正负环的情况下, 栈比队列更快, 但是如果没有正负环的一般情况下, 队列更快。
```

```

        s.push(0);
        while(s.size()){
            auto t=s.top();
            s.pop();
            st[t]=0;
            for(int i=h[t];i!=-1;i=ne[i]){
                if(dis[e[i]]<dis[t]+w[i]){
                    dis[e[i]]=dis[t]+w[i];
                    cnt[e[i]]=cnt[t]+1;
                    if(!st[e[i]]){
                        s.push(e[i]);
                        st[e[i]]=1;
                    }
                    if(cnt[e[i]]>=n+1){//有 0-n n+1 个点,若成正环则不存在解
                        return false;
                    }
                }
            }
        }
        return true;
    };
    if(!spfa())cout<<-1<<endl;
    else{
        int ans=0;
        for(int i=1;i<=n;i++)ans+=dis[i];
        cout<<ans<<endl;
    }
}

signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    //    cin>>t;
    while(t--)solve();
    return 0;
}

```

取不相邻元素的方法总数(组合数)

1.从排成一行的 n 个球中选出互不相邻的 r 个球,有多少种取法?

$C[n-(r-1)][r]$ 种,首先在 n 个球中留 $r-1$ 个球用来隔开 r 个球,同时在 $n-(r-1)$ 个球中取出 r 个需要被隔开的球,

当 $C[n][m]$ 不合法时当做 0 处理,例如 $C[4][5]=0$

2.从围成一圈的 n 个球中选出互不相邻的 r 个球有多少种取法?

$n(C[n-r][r])/(n-r)$ 种

<1>包含 1 号球,只需要在(3 到 $n-1$)中取互不相邻的 $r-1$ 个球(因为 2 和 n 与 1 相邻),此时已经将圈转化为第一个行的问题,答案为 $C[n-r-1][r-1]$

<2>不包含 1 号球,还需要在(2 到 n)中取互不相邻的 r 个球,也将圈转化为第一个行的问题,答案为 $C[n-r][r]$

两种情况相加整理后即为上述的总答案 $n(C[n-r][r])/(n-r)$ 种

当 $C[n][m]$ 不合法时当做 0 处理,例如 $C[4][5]=0$

3164. 线性基(任意异或和最大)

给定 n 个整数（可能重复），现在需要从中挑选任意个整数，使得选出整数的异或和最大。

请问，这个异或和的最大可能值是多少。

输入格式

第一行一个整数 n 。

第二行包含 n 个整数。

输出格式

输出一个整数，表示所选整数的异或和的最大可能值。

数据范围

$1 \leq n \leq 10^5$,

$0 \leq S_i \leq 2^{63}-1$

输入样例：

```
3
5 2 8
```

输出样例：

```
15
```

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
void solve(){
    int n;
    cin>>n;
    vector<int>a(n);
    for(int i=0;i<n;i++)cin>>a[i];
```

```

auto get=[&](vector<int>a){//求线性基
    int k=0;//k 之后会增加,k 之前的默认都已经暂时确定了
    for(int i=62;i>=0;i--){
        for(int j=k;j<n;j++){//第 k 个正要确定,如果有第 j 位为1 的那么第 k 个就是这
            if(a[j]>>i&1){
                swap(a[j],a[k]);
                break;
            }
        }
        if(!(a[k]>>i&1))continue;//如果跑完 k 到 n-1,第 j 位还是 0 那么所有数的第 j
        位都是 0 就不用管了
        //否则第 j 位为1 的个数>=1 把除第 k 个的第 j 位为1 的其他的都抵消掉
        for(int j=0;j<n;j++){
            if(j==k)continue;
            if(a[j]>>i&1){
                a[j]^=a[k];
            }
        }
        k++;
        if(k==n)break;
    }
    a.erase(a.begin()+k,a.end());
    return a;
};

auto res=get(a);
int ans=0;
for(auto i : res){
    ans^=i;
}
cout<<ans<<endl;
}

signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    //    cin>>t;
    while(t--)>solve();
    return 0;
}

```

210. 异或运算(线性基,任意异或和第 k 小)

给你由 N 个整数构成的整数序列,你可以从中选取一些(至少一个)进行异或(xor)运算,从而得到很多不同的结果。

请问，所有能得到的不同的结果中第 k 小的结果是多少。

输入格式

第一行包含整数 T ，表示共有 T 组测试数据。

对于每组测试数据，第一行包含整数 N 。

第二行包含 N 个整数(均在 1 至 10^{18} 之间)，表示完整的整数序列。

第三行包含整数 Q ，表示询问的次数。

第四行包含 Q 个整数 k_1, k_2, \dots, k_Q ，表示 Q 个询问对应的 k 。

输出格式

对于每组测试数据，第一行输出 `Case #C:`，其中 C 为顺序编号（从 1 开始）。

接下来 Q 行描述 Q 次询问的结果，每行输出一个整数，表示第 i 次询问中第 k_i 小的结果。

如果能得到的不同结果的总数少于 k_i ，则输出 -1 。

数据范围

$1 \leq N, Q \leq 10000$,

$1 \leq k_i \leq 10^{18}$

输入样例：

```
2
2
1 2
4
1 2 3 4
3
1 2 3
5
1 2 3 4 5
```

输出样例：

```
Case #1:
1
2
```

3
-1
Case #2:
0
1
2
3
-1

注意:只选取一个数字进行运算,则结果为该数字本身。

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define inf 0x3f3f3f3f3f3f3f3f
typedef pair<int,int> PII;
int t=1;
void solve(){
    int n;
    cin>>n;
    vector<int>a(n);
    for(int i=0;i<n;i++)cin>>a[i];
    auto get=[&](vector<int>a){//求线性基
        int k=0;//k之后会增加,k之前的默认都已经暂时确定了
        for(int i=62;i>=0;i--){
            for(int j=k;j<n;j++){//第k个正要确定,如果有第j位为1的那么第k个就是这个
                if(a[j]>>i&1){
                    swap(a[j],a[k]);
                    break;
                }
            }
            if(!(a[k]>>i&1))continue;//如果跑完k到n-1,第j位还是0那么所有数的第j位都是0就不用管了
            //否则第j位为1的个数>=1 把除第k个的第j位为1的其他的都抵消掉
            for(int j=0;j<n;j++){
                if(j==k)continue;
                if(a[j]>>i&1){
                    a[j]^=a[k];
                }
            }
        }
    }
```

```

        k++;
        if(k==n)break;
    }
    a.erase(a.begin()+k,a.end());
    return a;
};

auto res=get(a);//vector 存的线性基
reverse(res.begin(),res.end());//之前 res[0] 存大的, res[n-1] 存小的, 现在调换顺序
在提问时求第 k 小方便
cout<<"Case #"<<t++<<":"<<endl;
int q;
cin>>q;
while(q--){
    int x;
    cin>>x;
    if(res.size()<n)x--;//这个条件说明原序列线性相关, 但是线性基线性无关, 所以可以多凑出一个0, 第 k 小往前错一位
    //线性无关的 res.size() 个线性基可以构成(1ll<<res.size())-1 个不同的数, 有点类似 3 位二进制可以构成 7 个不同的数, 因为线性基互相不会抵消掉
    if(x>(1ll<<res.size())-1)cout<<-1<<endl;
    else{
        int ans=0;
        for(int i=0;i<res.size();i++){
            if(x>>i&1){
                ans^=res[i];
            }
        }
        cout<<ans<<endl;
    }
}
}

signed main(){
    int t=1;ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    cin>>t;
    while(t--){solve();}
    return 0;
}

```

线段树板子集合

1.区间 gcd 线段树,单点修改,求区间的 gcd,支持三种操作(下标是从 0 开始的)。

声明 `SegmentTreeGCD tr(b);` `b` 是 `vector<int>` 下标 0 到 `n-1` 传参

把下标为 `x` 的修改为 `v` `tr.update(x,v);`

查询 `l` 到 `r` 区间的所有值求 gcd `tr.query(l,r);`

若要删除某个值再取 gcd,可以把该点的值赋值为 0,gcd 就为别人了

```
class SegmentTreeGCD {
private:
    std::vector<int> tree;
    std::vector<int> data;
    int n;

    int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b);
    }

    void build(int node, int start, int end) {
        if (start == end) {
            tree[node] = data[start];
        } else {
            int mid = (start + end) / 2;
            build(2 * node, start, mid);
            build(2 * node + 1, mid + 1, end);
            tree[node] = gcd(tree[2 * node], tree[2 * node + 1]);
        }
    }

    void update(int node, int start, int end, int idx, int value) {
        if (start == end) {
            data[idx] = value;
            tree[node] = value;
        } else {
            int mid = (start + end) / 2;
            if (start <= idx && idx <= mid) {
                update(2 * node, start, mid, idx, value);
            } else {
                update(2 * node + 1, mid + 1, end, idx, value);
            }
            tree[node] = gcd(tree[2 * node], tree[2 * node + 1]);
        }
    }

    int query(int node, int start, int end, int L, int R) {
        if (R < start || end < L) {
            return 0;
        }
        if (L <= start && end <= R) {
            return tree[node];
        }
    }
}
```

```

        int mid = (start + end) / 2;
        int left_gcd = query(2 * node, start, mid, L, R);
        int right_gcd = query(2 * node + 1, mid + 1, end, L, R);
        return gcd(left_gcd, right_gcd);
    }

public:
    SegmentTreeGCD(const std::vector<int>& input) {
        data = input;
        n = input.size();
        tree.resize(4 * n);
        build(1, 0, n - 1);
    }

    void update(int idx, int value) {
        update(1, 0, n - 1, idx, value);
    }

    int query(int L, int R) {
        return query(1, 0, n - 1, L, R);
    }
};

```