# 武林秘籍

### 快读

```
#ifdef LOCAL
    freopen("in", "r", stdin);
    freopen("out", "w", stdout);
#endif
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
#define debug freopen("1.in", "r", stdin), freopen("1.out", "w", stdout)
#define mem(a,b) memset(a,b,sizeof(a))
#define inv(a,p) fpow(a,p-2,p)
#define max(a,b) (a>b?a:b)
#define min(a,b) (a<b?a:b)
#define lb(i) (i&-i)
#define pw(a) (1LL<<a)
#define ls(a) (a<<1LL)
#define rs(a) (a<<1LL|1LL)
#define make make_pair
#define PP push_back
#define xx first
```

## 基础算法

### 归并排序

```
//归并排序
#include<iostream>
using namespace std;
int n;
const int N =1e5+10;
int a[N],tmp[N];//声明一个临时存放的一个数组
void merge_sort(int a[],int l,int r){
    if(l>=r)return ;
    int mid=(l+r)/2;                    //中间位置，不是值
    merge_sort(a,l,mid);                //从左到中间排
    merge_sort(a,mid+1,r);              //从中间到右排
    int k=0,i=l,j=mid+1;                //tmp数组的下标，左右的指针
    while(i<=mid&&j<=r){                //是否到头
        if(a[i]<=a[j])tmp[k++]=a[i++];
        else tmp[k++]=a[j++];           //将两个有序的数组（一个数组的左右）中小的数输入
    }
    while(i<=mid)tmp[k++]=a[i++];
    while(j<=r)tmp[k++]=a[j++];          //剩余的数直接排到尾
```

```
for(int q=l,b=0;q<=r;q++,b++)a[q]=tmp[b]; //放回去
}
int main(){
    scanf("%d",&n);
    for(int i=0;i<n;i++)scanf("%d",&a[i]);
    merge_sort(a,0,n-1);
    for(int i=0;i<n;i++)printf("%d ",a[i]);
    return 0;
}
```

**二分（离散化点的位置，二分边界，二分答案）**

```
int t=lower_bound(vx.begin(),vx.end(),x)-vx.begin();
//取出第一个下标等于x的位置
//整数二分
int a[maxn];
int main() {
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=0;i<n;i++)scanf("%d",&a[i]);
    while(m--){
        int x;
        scanf("%d",&x);
        int l=0,r=n-1;
        while(l<r){
            int mid=(l+r)>>1;
            if(a[mid]>=x)r=mid;    //其实是以mid为判断依据，无论如x在范围内;。
            else l=mid+1;
        }
        if(a[l]!= x)printf("-1 -1\n");
        else{
            printf("%d ",l);
            int l=0,r=n-1;
            while(l<r){
                int mid=(l+r+1)>>1;    //加1是因为下面减一了，要是l和r相差了1，如1和2，不加1mid始终是1;
                if(a[mid]<=x)l=mid;
                else r=mid-1;//是应为向下取余
            }
            printf("%d\n",l);
        }
    }
    return 0;
}
//浮点数二分
int main(){
    double x,m,l=-100,r=100;  //大概估算一下多大找到二分的区间
    scanf("%lf",&x);
    while(r-l>1e-8){          //保留到1e-，多往下取2位
        m=(l+r)/2;
        if(m*m*m>=x)r=m;    //因为是浮点数，所以直接等，没整数二分那样变态
        else l=m;
    }
    printf("%.6lf",m);
```

```
}
```

**高精度系列**

```
//高精度加法
const int N=1e5+10;
vector<int>add(vector<int> &A,vector<int> &B){
    //加&是为了提高效率
    vector<int> C;
    int t=0;
    for(int i=0;i<A.size()||i<B.size();i++){
        //是或的逻辑因为要考虑输出位数
        if(i<A.size())t+=A[i];
        if(i<B.size())t+=B[i];
        C.push_back(t%10);//留的一位数
        t/=10;//剩的是进位
    }
    if(t)C.push_back(1);//判断是不是高位进了1
    return C;
}
int main(){
    string a,b;
    //当作字符串读入数字
    cin>>a>>b;
    vector<int> A,B;
    //像一般声明一样要空格
    for(int i=a.size()-1;i>=0;i--)
        A.push_back(a[i]-'0');
    //注意是i--，别顺手打错了，还有长度-1
    for(int i=b.size()-1;i>=0;i--)
        B.push_back(b[i]-'0');
    //将string转进容器中，想想123456789这数怎么插入
    auto C=add(A,B);
    //以进位方便为主 不是一般的习惯，下表为0的地方放个位
    for(int i=C.size()-1;i>=0;i--)
        cout<<C[i];
    //运行结果输出，是因为高位下标大，在平幕前先输出
    return 0;

}


//高精度减法
bool cmp(vector<int> &A,vector<int> &B){
    //要自己写判断两个数大小的函数，注意！=的使用，太神了
    if(A.size()!= B.size()) return A.size()> B.size();
    for(int i=A.size()-1;i>=0;i--){  //i--
        if(A[i]!=B[i])               //神操作
         return A[i]>B[i];           //注意返回值
    }
    return true;
}
vector<int>sub(vector<int> &A,vector<int> &B){
    vector<int> C;
```

```
    for(int i=0,t=0;i<A.size();i++){
        t=A[i]-t;
        if(i<B.size())t-=B[i];
        //默认了A大只需特判B
        C.push_back((t+10)%10);
        //神操作！把t>0和t<0的情况和二为一了
        if(t<0)t=1;                  //判断是否借位
        else t=0;
    }
    while(C.size()>1&&C.back()==0)C.pop_back();
    //清楚前面空的0
return C;
}
int main(){
    string a,b;
    cin>>a>>b;
    vector<int> 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(cmp(A,B)){                    //cmp的作用
        auto C=sub(A,B);
        for(int i=C.size()-1;i>=0;i--)cout<<C[i];
    }
    else
    {
        auto C=sub(B,A);
        cout<<"-";
        for(int i=C.size()-1;i>=0;i--)cout<<C[i];
    }
    return 0;
}

//高精度乘法
vector<int>mul(vector<int> &A,int B){
    vector<int> C;
    int t=0,k=0;
    //t作为进位，k是自己用来记录最后的进位
    for(int i=0;i<=A.size();i++){
        //到A.szie()-1时算完所有的数，表示的是最大的几位数（不止两位；
        t=A[i]*B+t;
        //到i=A.size()时t以及变成进位了
        C.push_back(t%10);
        //进位小于10输入就完了
        if(i==A.size()&&t>10)k=t;
        //大于十就特判一下记录
        t/=10;
    }
    if(k!=0)C.push_back(k/10);
    //其实可以直接放在后面的
    while(C.size()>1&&C.back()==0)C.pop_back();
    //如果b为0时可以把多余的0去掉
    return C;
}
```

```
int main(){
    string a;
    int b;
    vector<int> A;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)
        A.push_back(a[i]-'0');//注意-‘0’
    auto C=mul(A,b);
    for(int i=C.size()-1;i>=0;i--)cout<<C[i];
    return 0;
}
//高精度除法
vector<int>div(vector<int> &A,int B,int &r){
    vector<int> C;
    r=0;
   for(int i=A.size()-1;i>=0;i--){
        //除法从高往低除，高的在最后
        r=10*r+A[i];
        //接位向下处所以*10，向下进位
        C.push_back(r/B);              //得到商
        r%=B;                          //得到余数
   }
   reverse(C.begin(),C.end());
    //相当于高位先输入了，所以反转
    while(C.size()>1&&C.back()==0)C.pop_back();
    return C;
}
int main(){
    string a;
    int b;
    vector<int> A;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    int r=0;
    auto C=div(A,b,r);
    for(int i=C.size()-1;i>=0;i--)cout<<C[i];
    cout<<endl<<r;
    return 0;
}
```

**前缀和差分(1次修改，多次查询)和（多次修改区间，1次查询）**

```
//二维矩阵和

#include<iostream>
using namespace std;
const int N=1e3+10;
int a[N][N],s[N][N];
int main(){
    int n,m,q;
    int x1,x2,y2,y1;
    cin>>n>>m>>q;
    s[0][0]=0;
```

```
    s[1][0]=0;
    s[0][1]=0;   //预处理
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            cin>>a[i][j];
            s[i][j]=a[i][j]+s[i-1][j]+s[i][j-1]-s[i-1][j-1];
        }
    }
    while(q--){
        cin>>x1>>y1>>x2>>y2;
        cout<<s[x2][y2]-s[x1-1][y2]-s[x2][y1-1]+s[x1-1][y1-1]<<endl;
    }
    return 0;
}
```

```
//二维差分矩阵
#include<iostream>
using namespace std;
const int N=1010;
int cf[N][N],a[N][N];
void suancf(int x1,int y1,int x2,int y2,int c){
    cf[x1][y1]+=c;
    cf[x2+1][y1]-=c;
    cf[x1][y2+1]-=c;
    cf[x2+1][y2+1]+=c;          //原本的输入可以看成是0，然后在1*1的方格里算差分
}
int main(){
    int n,m,q,x,x1,x2,y1,y2,r;
    cin>>n>>m>>q;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            cin>>a[i][j];
            suancf(i,j,i,j,a[i][j]);
        }
    }
    while(q--){
        cin>>x1>>y1>>x2>>y2>>r;
        suancf(x1,y1,x2,y2,r);
    }

    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            cf[i][j]+=cf[i-1][j]+cf[i][j-1]-cf[i-1][j-1]; //注意这里是个累积的过程
            cout<<cf[i][j]<<" ";
        }
    cout<<endl;
   }
   return 0;
}
```

**双指针 （for一遍维护另一端的值）**

最长连续不重复子序列

给定一个长度为 $n$ 的整数序列，请找出最长的不包含重复的数的连续区间，输出它的长度。

```
#include<iostream>
using namespace std;
const int N=1e5+10;
int a[N];
int st[N];
int main(){
    int n;
    cin>>n;
    int ans=0;
    for(int i=1;i<=n;i++) cin>>a[i];
    int j=1;
    for(int i=1;i<=n;i++){
        st[a[i]]++;
        while(st[a[i]]>=2){
            st[a[j]]--;
            j++;  //j只会再从后往前扫一遍
                  //使所有区间里的数字只出现1次
        }
        ans=max(ans,i-j+1);
    }
    cout<<ans;
    return 0;
}
```

**kmp**

```
#include <iostream>

using namespace std;

const int N = 100010, M = 1000010;

int n, m;
int ne[N];
char s[M], p[N];

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)
        {
            printf("%d ", i - n);
            j = ne[j];
        }
    }
    return 0;
}
```

```
const int N=200003;   //一般是范围的2~3倍
const int null=0x3f3f3f3f;  //大概比 1e9大一点
int h[N];
void insert(int x){
    int k=((x%N)+N)%N;    //N小，在C++中负数取模还为负数取完模再加安全
    if(h[k]==null) h[k]=x;
    else{
        while(h[k]!=null){    //插入只需判断是否有空
            k++;
            if(k==N) k=0;    //越界记得返回头
        }
        h[k]=x;              //找到下标后完成插入
    }
 }
int find(int x){
    int k=((x%N)+N)%N;
    if(h[k]==x) return k;
    else{
        while(h[k]!=null && h[k]!=x){   //找虽然是找x但x不一定有，不加是h[k]!=null会死循环
            k++;
            if(k==N) k=0;               //其实可以把insert和find和在一起
        }
        return k;
   }
 }
int main(){
    int n,x;
    char op[2];
    scanf("%d",&n);
    memset(h,0x3f,sizeof h);
    while(n--){
        scanf("%s%d",op,&x);
        if(op[0]=='I') insert(x);
        else{
            if(h[find(x)]==x) cout<<"Yes"<<endl;
            else  cout<<"No"<<endl;
        }
    }
    return 0;
}
```

# 简单搜索 图论

### 三个简单的递归

```
//递归实现指数型枚举
const int N=20;
int n;
bool st[N];
void dfs(int u){
    if(u==n+1){
        //递归的终点从1开始的到n+1
        for(int i=1;i<=n;i++){
            if(st[i]){
                printf("%d ",i);
                //这里算是遍历了原数组
            }
        }
        puts("");
        return ;
    }
    st[u]=true;
    //代码是严格按照从上到下的顺序来的，这里改成用了这个数再向下
    dfs(u+1);
    st[u]=false;
    //上面遍历完全用过，再改成没用过就行
    dfs(u+1);
}

//递归实现排列型枚举
using namespace std;
const int N=10;
int n,q[N];                         //改为存进数组里
bool st[N];
void dfs(int u){
    if(u==n){
        for(int i=0;i<n;i++){
            printf("%d ",q[i]);     //这里遍历数组
        }
        puts("");
        return ;
    }
    for(int i=1;i<=n;i++){
        if(!st[i]){
            q[u]=i;
            st[i]=true;
            //先会一直顺着下去到3，3结束，3改回来，到2改回来，在2处的循环向下到3完成先选1后2
后3，再3再2；
            dfs(u+1);
            st[i]=false;
        }
    }
}
// 递归实现组合型枚举
const int N=30;
int n,m;
```

```
int q[N];
void dfs(int u,int start){   //需要的三个参数，数组存放位置，u为递归的层数，start保证从后向前的顺序
    if(u==m+1){
        for(int i=1;i<=m;i++)printf("%d ",q[i]);
        puts("");
        return ;
    }
    for(int i=start;i<=n;i++){
        q[u]=i;
        dfs(u+1,i+1);
    }
}
int main(){
  scanf("%d%d",&n,&m);
  dfs(1,1);
  return 0;
}
```

**n-皇后问题**

```
#include<iostream>
using namespace std;
const int N=21;
char g[N][N];
int n;
bool row[N],col[N],dg[N],udg[N]; //行，列，对角线
void dfs(int x,int y,int u){
    if(y==n)y=0,x++; //移动y一行一行遍历，判断。到尽头了就拐了，这里偷懒让y越界了
    if(x==n){            //到了最后一行
        if(u==n){        //判断是不是放了n个棋子就是遍历的终点
            for(int i=0;i<n;i++) puts(g[i]);
            puts("");
        }
        return ;
    }
    //不放皇后直接挪到下一个格子
    dfs(x,y+1,u);
    //放皇后要判断
    if(!row[x]&&!col[y]&&!dg[x+y]&&!udg[y-x+n]){
        g[x][y]='Q';
        row[x]=col[y]=dg[x+y]=udg[y-x+n]=true; //改变状态
        dfs(x,y+1,u+1);  //是一行一行遍历的x不能+1;
        g[x][y]='.';
        row[x]=col[y]=dg[x+y]=udg[y-x+n]=false;
    }
}
int main(){
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            g[i][j]='.';
        }
    }
```

```
    dfs(0,0,0);
    return 0;
}
```

**八数码**

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。X与相邻的数字swap

问是否存在能变成这样

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

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

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

        int distance=d[t];   //记录距离

        int k=t.find('x');    //找下标
        int x=k/3,y=k%3;      //一维转二位初边数，取模

        for(int i=0;i<4;i++){
            int a=x+dx[i],b=y+dy[i];

            if(a>=0&&b>=0&&a<3&&b<3){
                swap(t[a*3+b],t[k]);  //交换

                if(!d.count(t)){
                    //判断更新后的t是不是用过了，在d里找一下
                    d[t]=distance+1;
                    //没有遍历过就更新距离而且输入
                    q.push(t);
                }
                swap(t[a*3+b],t[k]);
                //恢复这个t还有3步走法
            }
        }
    }
    return -1;
}
int main(){
```

```
    string start;
    char a[2];
    for(int i=0;i<9;i++){
        cin>>a;
        start+=a;
    }
    cout<<bfs(start);
    return 0;
}
```

**有向图的拓扑排序**

```
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=1e5+10;
priority_queue<int,vector<int>,greater<int>> head;
vector<int> g[N];
vector<int> ans;
int n,m;
int in[N];
void solve(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        g[i].clear();
        in[i]=0;
    }
    //把点按入度分，依次遍历
    for(int i=1;i<=m;i++){
        int a,b;
        scanf("%d%d",&a,&b);
        g[a].push_back(b);
        in[b]++;
    }
    for(int i=1;i<=n;i++) if(in[i]==0) head.push(i);
    while(head.size()){
        int t=head.top();
        head.pop();
        ans.push_back(t);
        for(auto val:g[t]){
            in[val]--; //每遍历一次找到入度为0的点
            if(in[val]==0) head.push(val);
        }
    }
    if(ans.size()!=n) printf("-1");
    else{
        for(auto t:ans) printf("%d ",t);
    }
    return ;
}
int main(){
    solve();
    return 0;
}
```

### Dijkstra

```
using namespace std;
typedef pair<int,int> PII;
int n,m;
const int N=1e6+10;
int ne[N],e[N],idx,h[N],w[N];  //多的w数组用来储存边长，不用邻接矩阵
int dist[N];
bool st[N];
int djs(){
    memset(dist,0x3f,sizeof dist);  //初始化

    priority_queue<PII, vector<PII>, greater<PII>> heap;  //优先对列，first为1到点的距离，second为点

    dist[1]=0;
    heap.push({0,1});     //初始化距离
    while(heap.size()){
        auto t=heap.top();
        heap.pop();

        int ver=t.second,distance=t.first;

        if(st[ver]) continue ;
        st[ver]=true;

        for(int i=h[ver];i!=-1;i=ne[i]){

            int j=e[i];

            if(dist[j]>dist[ver]+w[i]){
                dist[j]=dist[ver]+w[i];
                heap.push({dist[j],j});
            }

        }
    }
    if(dist[n]==0x3f3f3f3f) return -1;
    return dist[n];
}
void add(int a,int b,int c){
    e[idx]=b;
    ne[idx]=h[a];
    w[idx]=c;
    h[a]=idx;
    idx++;
}
int main(){
    cin>>n>>m;
    int a,b,c;
    memset(h,-1,sizeof h);
    while(m--){
        cin>>a>>b>>c;
        add(a,b,c);
    }
```

```
    cout<<djs();
    return 0;
}
```

**树的重心**

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。

```
using namespace std;
const int N=1e5+10,M=2*N;
int n;
bool st[N];
int ans=N;
int h[N],e[M],ne[M],idx;
void add(int a,int b){
    e[idx]=b;
    ne[idx]=h[a];   //链接联通块
    h[a]=idx;
    idx++;
}
int dfs(int u){
    st[u]=true;     //看这个节点是否遍历过了
    int size=0,sum=0;   //记录结果
    for(int i=h[u];i!=-1;i=ne[i]){
        int j=e[i];
        if(st[j]) continue;   //用过就直接跳

        int s=dfs(j);       //没用用过沿着搜
        size=max(size,s);  //
        sum+=s;
    }
    size=max(size,n-sum-1);
    ans=min(size,ans);
    return sum+1;

}
int main(){
    memset(h,-1,sizeof h);
    cin>>n;
    int k=n-1;
    while(k--){
        int a,b;
        cin>>a>>b;
        add(a,b);
        add(b,a);
    }
    dfs(1);
    cout<<ans;
```

```
    return 0;
}
```

**有边数限制的最短路 bellman-ford**

```
using namespace std;
int n,m,k;
const int N=510,M=10010;

 struct{
    int a,b,w;
}edges[M];    //记录的是边的信息
int dist[N];
int backup[N];  //防止多走的备份
void bellman_fold(){

    memset(dist,0x3f,sizeof dist);
    dist[1]=0;

    while(k--){
        memcpy(backup,dist,sizeof dist);   //每次循环直走备份保证只走一步
        for(int i=0;i<m;i++){
            auto e=edges[i];
            dist[e.b]=min(dist[e.b],backup[e.a]+e.w);
        }
    }
}
int main(){
    scanf("%d%d%d",&n,&m,&k);

    for(int i=0;i<m;i++){
        int a,b,w;
        scanf("%d%d%d",&a,&b,&w);
        edges[i]={a,b,w};
    }

    bellman_fold();

    if(dist[n]>0x3f3f3f3f/2)printf("impossible");
    else printf("%d",dist[n]);

    return 0;
}
```

**spfa 最短路**

```
using namespace std;
const int N=1e5+10;
int n,m;
int ne[N],e[N],idx,h[N],w[N];
int dist[N];
bool st[N];
void spfa(){
    memset(dist,0x3f,sizeof dist);
```

```
    dist[1]=0;
    queue<int> q;
    q.push(1);
    st[1]=true;                 //标记，不再遍历
    while(q.size()){
        int t=q.front();
        q.pop();
        st[t]=false;        //如果取出来了，就是变得更小了，清标记
        for(int i=h[t];i!=-1;i=ne[i]){
            int j=e[i];
            if(dist[j]>dist[t]+w[i]){     //i就是对应的t到该节点的下标
                dist[j]=dist[t]+w[i];   //判断是否变小
                if(!st[j]){            //没有遍历过或着不是变小的就不加入queue
                    q.push(j);
                    st[j]=true;              //改变状态
                }
            }
        }
    }
}
void add(int a,int b,int c){
    e[idx]=b;
    w[idx]=c;
    ne[idx]=h[a];
    h[a]=idx;
    idx++;
}
int main(){
    scanf("%d%d",&n,&m);
    int a,b,c;
    memset(h,-1,sizeof h);
    while(m--){
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);
    }
    spfa();
    if(dist[n]==0x3f3f3f3f)printf("impossible");
    else printf("%d",dist[n]);
    return 0;
}
```

**spfa 判断负环**

```
using namespace std;
const int N=1e5+10;
int n,m;
int ne[N],e[N],idx,h[N],w[N];
int dist[N],cnt[N];    //cnt数组记录点最短路的步数
bool st[N];
bool spfa(){
     //不是求距离的只是判断有负权环没有
    queue<int> q;
    for(int i=1;i<=n;i++){
        q.push(i);    //把每个点都输入
```

```
        st[i]=true;
    }
    while(q.size()){
        int t=q.front();
        q.pop();
        st[t]=false;
        for(int i=h[t];i!=-1;i=ne[i]){
            int j=e[i];
            if(dist[j]>dist[t]+w[i]){
                dist[j]=dist[t]+w[i];
                cnt[j]=cnt[t]+1;     //记录到t的边长
                if(cnt[j]>=n)return true;  //比n大就是有重复走过的点在最短路求解的过程中有负权边
                if(!st[j]){
                    q.push(j);
                    st[j]=true;
                }
            }
        }
    }
    return false;
}
void add(int a,int b,int c){
    e[idx]=b;
    w[idx]=c;
    ne[idx]=h[a];
    h[a]=idx;
    idx++;
}
int main(){
    scanf("%d%d",&n,&m);
    int a,b,c;
    memset(h,-1,sizeof h);
    while(m--){
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);
    }
    if(spfa())printf("Yes");
    else printf("No");
    return 0;
}
```

**Floyd 求最短路**

```
using namespace std;
const int N=210,INF=0x3f3f3f3f;
int d[N][N];
int n,m,Q;
void flod(){
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                d[i][j]=min(d[i][j],d[i][k]+d[k][j]);  //三个foe循环先循环k
            }
```

```
        }
    }
}
int main(){
    scanf("%d%d%d",&n,&m,&Q);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){    //这里是记录全部的点，所以j是<n;
            if(i==j) d[i][j]=0;  //防止自环
            else d[i][j]=INF;
        }
    }
    int a,b,c;
    while(m--){
        scanf("%d%d%d",&a,&b,&c);
        d[a][b]=min(d[a][b],c);  //防止重边
    }
    flod();
    while(Q--){
        int i,j;
        scanf("%d%d",&i,&j);

        if(d[i][j]>INF/2) printf("impossible\n");
        else printf("%d\n",d[i][j]);
    }
    return 0;
}。
```

**Prim求最小生成树**

```
using namespace std;
const int N=510,INF=0x3f3f3f3f;
int n,m;
int g[N][N],dist[N];
bool st[N];
int prim(){
    memset(dist,0x3f,sizeof dist);
    //所有边不联通，标记成正无穷
    int res=0;
    for(int i=0;i<n;i++){
        int t=-1;
        for(int j=1;j<=n;j++)
            if(!st[j]&&(t==-1||dist[t]>dist[j]))
                //找出没有遍历过的j的最大点（不在集合里）
             t=j;

    if(i&&dist[t]==INF)return INF;
        //没找到路程
    if(i) res+=dist[t];
    st[t]=true;
       //把没遍历过的j点加入集合
    for(int j=1;j<=n;j++) dist[j]=min(dist[j],g[t][j]);
        //用t更新其他点
    }
    return res;
}
```

```
int main(){
    scanf("%d%d",&n,&m);
    int a,b,c;
    memset(g,0x3f,sizeof g);
    for(int i=0;i<m;i++){
        scanf("%d%d%d",&a,&b,&c);
        g[a][b]=g[b][a]=min(g[a][b],c);
    }
    int t=prim();
    if(t==INF) puts("impossible");
    else printf("%d\n",t);
    return 0;
}
```

**Flood Fill 洪水灌溉法**

山峰与山谷

1. S 的所有格子都有相同的高度。
2. S 的所有格子都连通。
3. 对于 s 属于 S，与 s 相邻的 s 不属于，都有 ws>ws'（山峰），或者 ws<ws''（山谷）。
4. 如果周围不存在相邻区域，则同时将其视为山峰和山谷。

```
#define x first
#define y second

using namespace std;

typedef pair<int, int> PII;

const int N = 1010, M = N * N;

int n;
int h[N][N];
PII q[M];
bool st[N][N];

void bfs(int sx, int sy, bool& has_higher, bool& has_lower)   //重载加上&，不同联通块的种类
{
    int hh = 0, tt = 0;
    q[0] = {sx, sy};
    st[sx][sy] = true;

    while (hh <= tt)
    {
        PII t = q[hh ++ ];

        for (int i = t.x - 1; i <= t.x + 1; i ++ )
            for (int j = t.y - 1; j <= t.y + 1; j ++ )
            {
                if (i == t.x && j == t.y) continue;
                if (i < 0 || i >= n || j < 0 || j >= n) continue;
                if (h[i][j] != h[t.x][t.y]) // 山脉的边界
```

```
                {
                    if (h[i][j] > h[t.x][t.y]) has_higher  = true;
                    else has_lower = true;
                }
                else if (!st[i][j])
                {
                    q[ ++ tt] = {i, j};
                    st[i][j] = true;
                }
            }
    }
}

int main()
{
    scanf("%d", &n);

    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < n; j ++ )
            scanf("%d", &h[i][j]);

    int peak = 0, valley = 0;
    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < n; j ++ )
            if (!st[i][j])
            {
                bool has_higher = false, has_lower = false;
                bfs(i, j, has_higher, has_lower);
                if (!has_higher) peak ++ ;  //一个地方（平面）既可以是山峰又可以是山谷
                if (!has_lower) valley ++ ;
            }

    printf("%d %d\n", peak, valley);

    return 0;
}
```

**迭代加深**

满足如下条件的序列 X（序列中元素被标号为 1、2、3...m）被称为“加成序列”：

1. X[1]=1
2. X[m]=n
3. X[1]<X[2]<...<X[m−1]<X[m]
4. 对于每个 k（2≤k≤m）都存在两个整数 i 和 j （1≤i,j≤k−1，i和 j 可相等），使得 X[k]=X[i]+X[j]

你的任务是：给定一个整数 n，找出符合上述条件的长度 m 最小的“加成序列”。

如果有多个满足要求的答案，只需要找出任意一个可行解。

```
#include<iostream>
#include<cstring>
using namespace std;
const int N=200;
int ans[N];
```

```
bool st[N];
int n;
bool dfs(int u,int depth)
{
    if(u==depth) return ans[u-1]==n;
    memset(st,0,sizeof st);          //每次的一层防止重
    for(int i=u-1;i>=0;i--){           //从后向前枚举好一点
        for(int j=i;j>=0;j--){
            int s=ans[i]+ans[j];
            if(st[s]||s>n||ans[u-1]>=s) continue;      //越界情况，防止重复搜索只搜第一次的，s不能大于n，后面比前面大
            st[s]=true;
            ans[u]=s;
            if(dfs(u+1,depth)) return true;
        }
    }
    return false;
}
int main(){
    ans[0]=1;
    while(cin>>n){
        if(n==0) break;
        int depth;
        for(int i=1;i<=20;i++){
            if(dfs(1,i)){
                depth=i;
                break;
            }
        }
        for(int i=0;i<depth;i++) cout<<ans[i]<<" ";
        cout<<endl;
    }
    return 0;
}
```

**dfs剪枝**

木棒

乔治拿来一组等长的木棒，将它们随机地砍断，使得每一节木棍的长度都不超过 50个长度单位。

然后他又想把这些木棍恢复到为裁截前的状态，但忘记了初始时有多少木棒以及木棒的初始长度。

请你设计一个程序，帮助乔治计算木棒的可能最小长度。

每一节木棍的长度都用大于零的整数表示。

```
using namespace std;
const int N=70;
int a[N],len,n,sum;
bool st[N];
bool cmp(int c,int b){
    return c>b;
}
bool dfs(int u,int s,int start){

    if(u*len==sum) return true;
```

```
    if(s==len) return dfs(u+1,0,0);
    //这里要return 一搜搜到底也会一路返回

    for(int i=start;i<n;i++)
    {
        if(a[i]+s>len)continue;  //比len都长了应该再找小的
        if(st[i]) continue;      //用过了
        st[i]=true;              //标记
        if(dfs(u,s+a[i],i+1)) return true;
        //选这个木棒进下一层
        st[i]=false;
        //这里有else的含义，否则不选恢复现场
        if(s==0) return false;
        //如果s为0说明刚选的是第一根木棍，第一根都选不了，肯定寄了
        if(s+a[i]==len) return false;
        //如果是这组最后一根符合长度还寄了，那肯定寄
        int j=i;
        while(j<n&&a[j]==a[i]) j++;
        //这里优化，既然a[i]这个数值上的数都寄了，那就都跳过
        i=j-1;
    }
    return false;
}
int main(){
    while(cin>>n)
    {
        if(n==0) break;
        memset(st,0,sizeof st);
        sum=0,len=1;
        for(int i=0;i<n;i++) {
            cin>>a[i];
            sum+=a[i];
            len=max(len,a[i]);        //原来相等的木棒一定大于等于截取后的木棒，常数级优
化
        }
        sort(a,a+n,cmp);
        while(true){
            if(sum%len==0){
              if(dfs(0,0,0)){
                  cout<<len<<endl;
                  break;
              }
            }
            len++;
        }
    }
    return 0;
}
```

# 简单数学

### 埃氏筛

```
#include<iostream>
using namespace std;
const int N=1e6+10;
int cnt,primes[N];//建议开成vector
bool st[N];
void get_prime(int x){
    for(int i=2;i<=x;i++){
    if(!st[i]) primes[cnt++]=i;
    for(int j=0;primes[j]<=x/i;j++){
        //晒到的质数直到小于x/i;
        st[primes[j]*i]=true;
        //从小到大枚举素数的·1倍数
        if(i%primes[j]==0)break;
        //保证i不会是prime[j]的倍数，保证只筛一次
     }
    }
}
int main(){
    int n;
    scanf("%d",&n);
    get_prime(n);
    printf("%d",cnt);
    return 0 ;
}
```

### 快速幂 算数基本定理

```
算数基本定理：任何一个≥2的自然数 N都可以唯一分解成有限个素数（质数）的幂的乘积。
N = P1^a1 * P2^a2 * P3^a3 * ……* Pr^ar
这里P1 < P2 < P3 .. … < Pr 均为素数，其中指数ai是正整数。
ll a,b,p;
int imp(ll a,ll b,ll p){
    ll ans=1;
    while(b){
        if(b&1)ans=ans*a%p;
        a=a*a%p;
        b>>=1;
    }
    return ans;
}
```

### 组合计数

```
#include<bits/stdc++.h>
#define ll long long
#define inf 0x3f3f3f3f
using namespace std;
const int maxn=1e5+5;
const ll mod=998244353;
```

```
ll inv[maxn], fac[maxn];  //分别表示逆元和阶乘
//快速幂
ll quickPow(ll a,ll b){
    ll ans=1;
    while(b){
        if(b&1)
            ans=(ans*a)%mod;
        b>>=1;
        a=(a*a)%mod;
    }
    return ans;
}

void init(){
    //求阶乘
    fac[0]=1;
    for(int i=1;i<maxn;i++){
        fac[i]=fac[i-1]*i%mod;
    }
    //求逆元
    inv[maxn-1]=quickPow(fac[maxn-1],mod-2);
    for(int i=maxn-2;i>=0;i--){
        inv[i]=inv[i+1]*(i+1)%mod;
    }
}
ll C(int n,int m){
    if(m>n){
        return 0;
    }
    if(m==0)
        return 1;
    return fac[n]*inv[m]%mod*inv[n-m]%mod;
}
int main(){
    init();
    int n,m;
    scanf("%d%d",&n,&m);
    printf("%lld\n",C(n,m));
}
```

### exgcd

在欧几里得算法中，当$b=0$时，显然存在一对整数解$x=1,y=0$,使得$a*1+0*0=gcd(a,0)$.

令$ax+by=gcd(a,b)$
$\because gcd(b,a\ mod\ b)=gcd(a,b);$
$\therefore a*x+b*y=gcd(a,b)=gcd(b,a\ mod\ b)=b*tx+(a\ mod\ b)*ty$
$\because a\ mod\ b=a-\lfloor a/b\rfloor*b$
$\therefore a*x+b*y=b*tx+(a-\lfloor a/b\rfloor*b)*ty$
$\therefore a*x+b*y=a*ty+b*(tx-\lfloor a/b\rfloor*ty)$
$\therefore x=ty,y=tx-\lfloor a/b\rfloor*ty$
再用数学归纳法进行证明即可。
由于本题$K$不一定等同于$gcd(a,b)$，因此我们要先求到$gcd(a,b)$,后逆推得$x,y$，判断一下$K$是否被$gcd(a,b)$整除，再$x=x*k/d$,$y=(k-a*x)/b$

```
#include<cstdio>
```

```cpp
#include<algorithm>
#include<cmath>
#include<cstdlib>
#define ll long long
using namespace std;
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(!b)
    {
        x=1;y=0;
        return a;
    }
    else
    {
        ll tx,ty;
        ll d=exgcd(b,a%b,tx,ty);
        x=ty;y=tx-(a/b)*ty;
        return d;
    }
}
int main()
{
    ll a,b,k,x,y;scanf("%lld%lld%lld",&a,&b,&k);
    ll d=exgcd(a,b,x,y);
    if(k%d){puts("no solution!");return 0;}
    else
    {
        x=x*k/d;
        y=(k-a*x)/b;
    }
    printf("%lld %lld\n",x,y);
    return 0;
}
```

### 矩阵快速幂

```cpp

```

### 数学上来先打表

```cpp
//https://loj.ac/p/519
```

给你一个图，每个点有点权，最开始没有边。

有一些操作：

添加一条 x 与 y 之间的双向边。

回到第 x 次操作后的状态。（注意这里的 x 可以是 0，即回到初始状态）

查询 x 所在联通块能到的点中点权第 y 小的值，如果不存在，那么输出 -1。

```cpp
#include<bits/stdc++.h>
#define pw(a) (1LL<<(a))
#define L ((p)<<1)
#define R ((p)<<1|1)
#define endl '\n'
using namespace std;

typedef long long LL;
typedef double DD;
typedef pair<int,int> PAIR;

const int INF=0x3f3f3f3f;
const int mod=998244353;
const int N=1e5+10;
const int S=1500;

int n,m,k,t,ans[N],K;
int a[N],sum[N][N/S+100],pre[N],sz[N];
PAIR P[N];
int get(int x)
{
    if(pre[x]==x) return x;//涓嶈壓璽璽 緞鍘嫣缉
    return get(pre[x]);
}
struct Node
{
    int op,x,y,id;
}Q[N];
vector<int>gra[N];
int get(int x,int k)
{
    // for(int i=1;i<=n;i++) cout<<sum[i][1]<<' ';cout<<endl;
    x=get(x);
    if(sz[x]<k) return 0;
    int i=1;
    while(k>sum[x][i]) k-=sum[x][i],i++;
    for(int j=1;j<=S;j++)
    {
        int y=(i-1)*K+j;
        if(get(P[y].second)==x) k--;
        if(not k) return P[y].second;
    }
    return 0;
}
void dfs(int i)
{

    if(Q[i].op==1)
    {
        int u=get(Q[i].x),v=get(Q[i].y);
        if(sz[u]>sz[v]) swap(u,v);
        //  cout<<i<<' '<<u<<" "<<v<<endl;
        if(u!=v)
        {
            for(int i=1;i<=K;i++) sum[v][i]+=sum[u][i];
```

```
            sz[v]+=sz[u];
            pre[u]=v;
        }
        for(int v:gra[i]) dfs(v);
        if(u!=v)
        {
            for(int i=1;i<=K;i++) sum[v][i]-=sum[u][i];
            sz[v]-=sz[u];
            pre[u]=u;
        }
    }
    else if(Q[i].op==3)
    {
        ans[Q[i].id]=get(Q[i].x,Q[i].y);
        for(int v:gra[i]) dfs(v);
    }
    else for(int v:gra[i]) dfs(v);
}
signed main(void)
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin>>n>>m;
    K=((n+S-1)/S);
    for(int i=1;i<=n;i++) pre[i]=i,sz[i]=1;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        P[i]={a[i],i};
    }
    sort(P+1,P+n+1);
    for(int i=1;i<=n;i++) sum[P[i].second][(i-1)/S+1]=1;
    for(int i=1,op,x,y,id;i<=m;i++)
    {
        cin>>op;
        if(op==2)
        {
            cin>>x;
            gra[x].emplace_back(i);
        }
        else
        {
            cin>>x>>y;
            gra[i-1].emplace_back(i);
        }
        Q[i]={op,x,y,i};
    }
    dfs(0);
    a[0]=-1;
    for(int i=1;i<=m;i++)
    if(Q[i].op==3) cout<<a[ans[i]]<<endl;
    return 0;
}
```

# 数据结构

### STL 函数

```
priority_queue <int,vector<int>,less<int> > p;
priority_queue <int,vector<int>,greater<int> > q;
swap(a,b);
#define y1 asdaskjfhksagh
```

### 单调队列

```
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=1e6+10;
typedef pair<int,int> PII;
priority_queue<int> heap;
int a[N];
int q[N];
int he[N];
int n,k;
//长度最长为m的前缀和最大值 Si-Sj, 其实是把每个i前的长为m 的范围里
//求前缀最小值
void solve(){
    cin>>n>>k;
    for(int i=1;i<=n;i++) {
    cin>>a[i];
    he[i]=he[i-1]+a[i];
   }
   int hh=0,tt=0; //这里相等为什么
   int ans=-1e9;
   for(int i=1;i<=n;i++){
    if(i-q[hh]>k) hh++;
    while(hh<=tt&&he[i]<=he[q[tt]]) tt--;
     int s=q[hh];
    ans=max(ans,he[i]-he[s]);
    q[++tt]=i;
   }
   cout<<ans<<endl;
    return ;
}
int main(){
    solve();
    return 0;
}
// 对dp的优化
在某两个城市之间有 n
 座烽火台，每个烽火台发出信号都有一定的代价。
为了使情报准确传递，在连续 m
 个烽火台中至少要有一个发出信号。
现在输入 n,m
 和每个烽火台的代价，请计算在两城市之间准确传递情报所需花费的总代价最少为多少。
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
```

```
const int N=2e5+10;
typedef pair<int,int> PII;
priority_queue<int> heap;
int n,m,d;
int f[N];
int a[N];
int q[N];
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>> a[i];
    //想像一个滑动窗口选值
    int tt=0,hh=0;
    for(int i=1;i<=n;i++){
        if(i>q[hh]+m) hh++;
        f[i]=f[q[hh]]+a[i];
        while(hh<=tt&&f[i]<=f[q[tt]]) tt--;
        q[++tt]=i;
    }
     int res=0x3f3f3f3f;
    for(int i=n-m+1;i<=n;i++) res=min(f[i],res);
    cout<<res<<endl;
    return ;
}
int main(){
 solve();
    return 0;
}
```

**树状数组**

给$n$个数$a_1$到$a_n$

支持$q$个操作：

1. `1 x d`，修改$a_x = d_x$
2. `2 x`，查询$\sum xi = ai$。

```
typedef long long ll;
const int N=2e5+10;
int a[N],n;
ll c[N];
int lowbit(int x){
    return x&-x;
}
ll query(int x){
    ll s=0;
    for(;x;x-=lowbit(x)) s+=c[x];
    return s;
}
//正着求和，已知求和位置，从高到底求和
void modify(int x,int d){
    for(;x<=n;x+=lowbit(x)) c[x]+=d;
}
```

```
//反着加 已知改嫁的点反着从下往上加
int main(){
    int q;
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++)
    {
    scanf("%d",&a[i]);
    modify(i,a[i]);
    }
    while(q--){
        int t,x,d;
        scanf("%d%d",&t,&x);
        if(t==1){
            scanf("%d",&d);
            modify(x,d-a[x]);
            a[x]=d;
        }else  printf("%lld\n",query(x));
    }
    return 0;
}
```

**逆序对数量**

有$n$ 个数 $a_1, a_2, \ldots, a_n$，对于其中的两个数字 $x$,$y$，如果满足 $x$ 出现的位置在 $y$ 出现的位置前面并且 $x$比 $y$ 大，则称 $(x, y)$ 为数组 $a$ 的一个逆序对。请问数组 $a$ 的逆序对一共有多少个？形式化的说，请求出有多少组 $(i, j)$ 满足 $i < j$ 并且 $ai > aj$。

```
typedef long long ll;
const int N=2e5+10;
int a[N];
int n;
ll c[N];
int lowbit(int x){
    return x&-x;
}
ll query(int x){
    ll s=0;
    for(;x;x-=lowbit(x)){
        s+=c[x];
    }
    return s;
}
void modify(int x,int d){
    for(;x<=n;x+=lowbit(x)){
        c[x]+=d;
    }
    return ;
}
//原有思路 扫描线+权值的树状数组
//能算一个位置上的后面的比它小的点的个数
//求后缀和，把n-这个数+1就是 求这个数后面比它小的数
//就是到后面的数时，求比其大的数的个数，从而变成前缀和
//因为树状数组的求和是只能求前缀和的，每个点的权值为1
int main(){
    scanf("%d",&n);
```

```
    ll ans=0;
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        a[i]=n-a[i]+1;
    }
    for(int i=1;i<=n;i++){
        ans+=query(a[i]);
        modify(a[i],1);
    }
    printf("%lld\n",ans);
    return 0;
}
```

**差分树状数组 变为区间改单点查询**

区间加，单点查询

有$n$个数$a_1, a_2, a_3, \ldots, a_n$一开始都是0。

支持$q$个操作：

1. `1 l r d`，令所有的$ai(l \leq i \leq r)$加上$d$。
2. `2 x`，查询$\sum_i^x a_i(i=1) mod 2$的$64$次幂

```
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
typedef pair<int,int> PII;
priority_queue<int> hash;
u64 d[N],di[N]; //u64自动取mod
int n;
ll lowbit(ll x){
    return x&-x;
}
u64 query1(u64 x){
    u64 s=0;
    for(;x;x-=lowbit(x)) s+=d[x];
    return s;
}
u64 query2(u64 x){
    u64 s=0;
    for(;x;x-=lowbit(x)) s+=di[x];
    return s;
}
void modify1(int x,u64 y){
    for(;x<=n;x+=lowbit(x)){
        d[x]+=y;
    }
    return ;
}
void modify2(int x,u64 y){
    for(;x<=n;x+=lowbit(x)){
        di[x]+=y;
    }
}
void solve()
```

```
{
    int q;
    scanf("%d%d",&n,&q);
    while(q--){
        int t;
        scanf("%d",&t);
        if(t==1){
            int l,r;
            u64 d;
            scanf("%d%d%llu",&l,&r,&d);
            modify1(l,d);
            modify1(r+1,-d);
            modify2(l,l*d);
            modify2(r+1,-(r+1)*d);
        }else{
            int x;
            scanf("%d",&x);
            printf("%llu\n",(x+1)*query1(x)-query2(x));
        }

    }
    return ;
}
```

**树状状数组二分单点修改，查询和的最小边界**

给$n$个数$a_1, a_2, a_3, \ldots, a_n$。

支持$q$个操作：

1. `1 x d`，修改$ax = d$。
2. `2 s`，查询最大的$T(0 \le T \le n)$满足 $\sum_i^T a_i <= S(i从1开始)$

```
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
typedef pair<int,int> PII;
priority_queue<int> hash;
ll a[N];
ll c[N];
int n;
int lowbit(int x){
    return x&-x;
}
//int query(int x){
//  ll s=0;
//  for(;x;x-=lowbit(x)) s+=c[x];
//  return s;
//} 一般的
int query(ll s){
    int ans=0;
    for(int j=18;j>=0;j--){
        if((ans+(1<<j))<=n&&c[ans+(1<<j)]<=s){
            //<<的权值小，用要带（）
            // 利用二进制逼近ans，利用c数组的性质
```

```
            // xxx1(第j位)000000 c[i]=a(i-lowbit(i)+1)~a[i];
            //正好是 xxx00001到xxx10000的和
            //和也用二进制逼近化log平方为log
            ans+=1<<j;
            s-=c[ans];
        }
    }
    return ans;
}
void modify(int x,ll d){
    for(;x<=n;x+=lowbit(x)){
        c[x]+=d;
    }
    return ;
}

void solve(){
    int q;
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++) {
    scanf("%lld",&a[i]);
    modify(i,a[i]);
    }
    while(q--){
        int t;
        scanf("%d",&t);
        if(t==1){
            int x;ll d;
            scanf("%d%lld",&x,&d);
            modify(x,d-a[x]);
            a[x]=d;
        }else{
            ll s;
            scanf("%lld",&s);
            printf("%d\n",query(s));
        }
    }
    return ;
}
```

**二维树状数组**

给$n\times m$个数$a_{(1,1)}....a_{(n,m)}$

支持$q$个操作：

1. `1 x y d`，修改$a_{x,y}=d$。
2. `2 x y`，查询$\sum_{i=1}^{x}\sum_{j=1}^{y}a_{i\ j}$。

第一行三个整数$n,m,q(1\le n,m\le 500,q\le 2\times 105)$

```
typedef long long ll;
typedef unsigned long long u64;
const int N=510;
typedef pair<int,int> PII;
```

```
priority_queue<int> hash;
int a[N][N];
ll c[N][N];
int n,m;
int lowbit(int x){
    return x&-x;
}
ll query(int x,int y){
    ll s=0;
    for(int i=x;i;i-=lowbit(i)){
        for(int j=y;j;j-=lowbit(j)){
            s+=c[i][j];
        }
    }
    return s;
}
void modify(int x,int y,ll d){
    for(int i=x;i<=n;i+=lowbit(i)){
        for(int j=y;j<=m;j+=lowbit(j)){
            c[i][j]+=d;
        }
    }
    return ;
}

void solve(){
    int q;
    scanf("%d%d%d",&n,&m,&q);
    for(int i=1;i<=n;i++) {
        for(int j=1;j<=m;j++){
            scanf("%d",&a[i][j]);
            modify(i,j,a[i][j]);
        }
    }
    while(q--){
        int t;
        scanf("%d",&t);
        if(t==1){
            int x,y;
            ll d;
            scanf("%d%d%lld",&x,&y,&d);
            modify(x,y,d-a[x][y]);
            a[x][y]=d;
        }else{
            int x,y;
            scanf("%d%d",&x,&y);
            printf("%lld\n",query(x,y));
        }
    }
    return ;
}
```

**线段树1单点修改查询区间min（维护单点信息）**

给$n$个数

支持$q$个操作：

1. `1 x d`，修改$a_x = d$。
2. 2 l r`，查询$min_l^r a_i$,和区间最下值出现次数
3. 第一行两个整数$n, q(1 \le n, q \le 2\times 105)$

```
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
typedef pair<int,int> PII;
priority_queue<int> heap;
struct info{
    int minv,mincnt;
};
info operator + (const info &a,const info &b){
    info c;
    c.minv=min(a.minv,b.minv);
    if(a.minv==b.minv) c.mincnt=a.mincnt+b.mincnt;
    else if(a.minv<b.minv) c.mincnt=a.mincnt;
    else c.mincnt=b.mincnt;
    return c;
}
// 重定义运算符号，方便操作
struct node{
    info val;
} seg[4*N];
int a[N],n;
//定义线段树开4倍保证不超
void update(int id){
    seg[id].val=seg[id*2].val+seg[2*id+1].val;
    //线段树上的节点都是需要先递归到低从下往上更新
}
void build(int id,int l,int r){
    //id表示节点编号，l,r为左右区间
    //建树即为记录原数组的值
    if(l==r){
        seg[id].val={a[l],1};
    }else{
        int mid=l+r>>1;
        build(id*2,l,mid);
        build(id*2+1,mid+1,r);
        update(id);  //递归结束后要依次的去更新该节点的值
    }
}
void change(int id,int l,int r,int pos,int val){
    //定义单点修改，节点为id，对应区间[l,r],a[pos]->val;
    if(l==r){
        seg[id].val={val,1};
    }else{
        int mid=l+r>>1;
        if(pos<=mid) change(id*2,l,mid,pos,val);
        else change(id*2+1,mid+1,r,pos,val);
```

```
        //这里递归结束的节点也是需要update
        update(id);
    }

}
//void modify() //定义为区间修改

info query(int id,int l,int r,int ql,int qr){
     //区间上的查询，[ql,qr]上的性质
     if(ql==l&&qr==r){
        return seg[id].val;
     }else{
        int mid=l+r>>1;
        //[l,mid],[mid+1,r];
         // [ql,qr];
        if(qr<=mid) return query(id*2,l,mid,ql,qr);
        else if(ql>=(mid+1)) return query(id*2+1,mid+1,r,ql,qr);
        else{
            //那mid就在ql和qr之间
            //改变的是查询范围，递归的范围不能变才符合线段树性质
             return query(id*2,l,mid,ql,mid)+query(id*2+1,mid+1,r,mid+1,qr);
         }
         //查询操作并不需要update，因为没有改变的值
     }
}
void solve(){
    int m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    build(1,1,n);
    while(m--){
        int t;
        scanf("%d",&t);
        if(t==1){
            int x,d;
            scanf("%d%d",&x,&d);
            change(1,1,n,x,d);
        }else{
            int l,r;
            scanf("%d%d",&l,&r);
            auto ans=query(1,1,n,l,r);
            printf("%d %d\n",ans.minv,ans.mincnt);
        }
    }
    return ;
}
```

**线段树2单点修改查询最大子段和（维护区间信息）**

给$n$个数$a_1,,\ldots,a_n$。

支持$q$个操作：

1. `1 x d`，修改$ax=d$。
2. `2 l r`，查询$[l,r]$中的最大子段和，也就是找到$l\le l'\le r'\le r$，使得$\sum_l^r a_i$最大。

```cpp
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
typedef pair<int,int> PII;
priority_queue<int> heap;
int a[N];
int n;
int lowbit(int x){
    return x&-x;
}
struct info{
    ll mss,mssf,mpre,s;
    info(){}
    info(ll a):mss(a),mssf(a),mpre(a),s(a){}
};
///利用结构体压缩便于赋值
struct Node{
    info val;
}seg[4*N];
info operator + (const info &l,const info &r){
    info a;
    a.mss=max({l.mss,r.mss,l.mssf+r.mpre});  //更新三个值，左/友 ，中间分开
    a.mpre=max(l.mpre,l.s+r.mpre); //中间分开的要记录对应的前缀/后缀和
    a.mssf=max(r.mssf,r.s+l.mssf);
    a.s=l.s+r.s;
    return a;
}
void update(int id){
    seg[id].val=seg[id*2].val+seg[id*2+1].val;
}
void build(int id,int l,int r){
    if(l==r) seg[id].val=info(a[l]);
    else{
        int mid=l+r>>1;
        build(id*2,l,mid);
        build(id*2+1,mid+1,r);
        update(id);
    }
}
void change(int id,int l,int r,int pos,int x){
    if(l==r) seg[id].val=info(x);
    else{
        int mid=l+r>>1;
        if(pos<=mid) change(id*2,l,mid,pos,x);
        else change(id*2+1,mid+1,r,pos,x);
        update(id);
    }
}
info query(int id,int l,int r,int ql,int qr){
    if(ql==l&&qr==r) return seg[id].val;
    else{
        int mid=l+r>>1;
        // [l,mid] [mid+1,r]
        // [ql,qr]
```

```
        if(qr<=mid) return query(id*2,l,mid,ql,qr);
        else if(ql>=mid+1) return query(id*2+1,mid+1,r,ql,qr);
        else{
            return query(id*2,l,mid,ql,mid)+query(id*2+1,mid+1,r,mid+1,qr);
        }
    }
}
void solve(){
    int m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    build(1,1,n);
    while(m--){
        int t;
        scanf("%d",&t);
        if(t==1){
            int x;
            ll d;
            scanf("%d%lld",&x,&d);
            change(1,1,n,x,d);
        }else{
            int l,r;
            scanf("%d%d",&l,&r);
            auto ans=query(1,1,n,l,r);
            printf("%lld\n",ans.mss);
        }
    }
    return ;
}
```

**线段树上二分 单点改查询区间的边界（优化logn）**

给$n$个数$a_1, a_2, a_3, \ldots, a_n$。

支持$q$个操作：

1. `1 x d`，修改$a_x = d$。
2. `2 l r d`查询$a_l, a_{l+1}, \ldots, a_r$中大于等于$d$的第一个数的下标，如果不存在，输出$-1$。也就是说，求最小的$i(l \le i \le r)$满足$ai \ge d$。

```
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10,M=3e6+10;
typedef pair<int,int> PII;
priority_queue<int> tmheap;
unordered_map<string,int> mmp;
struct Node{
    ll val;
}seg[N*4];
ll a[N];
void update(int id){
    seg[id].val=max(seg[id*2].val,seg[id*2+1].val);
}
void build(int id,int l,int r){
    if(l==r) seg[id].val=a[l];
    else{
```

```
        int mid=l+r>>1;
        build(id*2,l,mid);
        build(id*2+1,mid+1,r);
        update(id);
    }
}
void change(int id,int l,int r,int x,int d){
    if(l==r&&l==x) seg[id].val=d;
    else{
        int mid=l+r>>1;
        if(x<=mid) change(id*2,l,mid,x,d);
        else change(id*2+1,mid+1,r,x,d);
        update(id);
    }
}
int search(int id,int l,int r,int ql,int qr,int d){
    if(l==ql&&r==qr){
        if(seg[id].val<d) return -1;
        if(l==r) return l;
        int mid=l+r>>1;
        if(seg[id*2].val>=d) return search(id*2,l,mid,ql,mid,d);
        else return search(id*2+1,mid+1,r,mid+1,qr,d);
    }else{
        int mid=l+r>>1;
        // [l,mid] [mid+1,r] [ql,qr]
        if(qr<=mid) return search(id*2,l,mid,ql,qr,d);
        else if(ql>=mid+1) return search(id*2+1,mid+1,r,ql,qr,d);
        else{
            int  pos=search(id*2,l,mid,ql,mid,d); //这里是要分成两边的查找，左儿子不合
适再找右
            if(pos==-1) return search(id*2+1,mid+1,r,mid+1,qr,d);
        }
    }
}
void solve(){
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    build(1,1,n);
    while(m--){
        int t;
        scanf("%d",&t);
        if(t==1){
            int x,d;
            scanf("%d%d",&x,&d);
            change(1,1,n,x,d);
        }else{
            int l,r,d;
            scanf("%d%d%d",&l,&r,&d);
            printf("%d\n",search(1,1,n,l,r,d));
        }
    }
    return ;
}
```

**线段树打标记 区间加查询区间最大值**

给$n$个数$a_1, a_2, a_3, \ldots, a_n$。

支持$q$个操作：

1. `1 l r d`，令所有的$a_i(l \le i \le r)$加上d。
2. `2 l r`，查询$max_l^r a_i$。

```
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
typedef pair<int,int> PII;
priority_queue<int> heap;
int a[N],n;
struct Node{
    ll t,val;
}seg[N*4];
void update(int id){
    seg[id].val=max(seg[id*2].val,seg[id*2+1].val);
}
void settag(int id,ll t){
    seg[id].val=seg[id].val+t;
    seg[id].t=seg[id].t+t;
}
void pushdown(int id){
    if(seg[id].t!=0){
        settag(id*2,seg[id].t);
        settag(id*2+1,seg[id].t);
        seg[id].t=0;  //向左右边儿子传递标记
    }
}
void bulid(int id,int l,int r){
    if(l==r) seg[id].val={a[l]};
    else {
        int mid=l+r>>1;
        bulid(id*2,l,mid);
        bulid(id*2+1,mid+1,r);
        update(id);  //build这里别忘了更新
    }
}
//当递归到子区间时候直接把懒标记更新了就行
void modify(int id,int l,int r,int ql,int qr,ll t){
    if(l==ql&&qr==r){
        settag(id,t);
        return ;
    }else{
        int mid=l+r>>1;
        pushdown(id);  //对于向下遍历的点也要加上标记
        if(qr<=mid){ modify(id*2,l,mid,ql,qr,t);
        }else if(ql>=mid+1) modify(id*2+1,mid+1,r,ql,qr,t);
        else{
            modify(id*2,l,mid,ql,mid,t);
            modify(id*2+1,mid+1,r,mid+1,qr,t);
        }
```

```
        update(id); //还要跟新这个点的信息
    }
}
ll query(int id,int l,int r,int ql,int qr){
    if(l==ql&&r==qr) return seg[id].val;
        int mid=l+r>>1;
        pushdown(id); //push的树上的标记是id
        // [l,mid] [mid+1,r];
        if(qr<=mid){ return query(id*2,l,mid,ql,qr);
        }else if(ql>=mid+1) return query(id*2+1,mid+1,r,ql,qr);
        else{
            return max(query(id*2,l,mid,ql,mid),query(id*2+1,mid+1,r,mid+1,qr));
        }
}
void solve(){
    int m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    bulid(1,1,n);
    while(m--){
        int t;
        scanf("%d",&t);
        if(t==1){
            int l,r,d;
            scanf("%d%d%d",&l,&r,&d);
            modify(1,1,n,l,r,d);
        }else{
            int l,r;
            scanf("%d%d",&l,&r);
            ll a=query(1,1,n,l,r);
            printf("%lld\n",a);
        }
    }
}
```

**线段树打标记2 区间加法区间乘法**

给$n$个数。

支持$q$个操作:

1. `1 l r d`，令所有的$ai(l \leq i \leq r)$加上d。
2. `2 l r d`，令所有的$ai(l \leq i \leq r)$乘上d。
3. `3 l r d`，令所有的$ai(l \leq i \leq r)$等于d。
4. `4 l r`，查询$(\sum_{l}^{r} a_i) mod(109+7)$

```
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
const ll mod=1000000007;
typedef pair<int,int> PII;
priority_queue<int> heap;
```

```
int a[N],n;
struct tag{
    ll mul,add;  //把标记统一化成乘一个数和加一个数
};
struct Node{
    tag t;
    ll val;
    int sz;
}seg[N*4];
tag operator + (const tag &t1, const tag &t2){
    tag t={(t1.mul*t2.mul)%mod,(t1.add*t2.mul+t2.add)%mod};
    return t;
}
void update(int id){
    seg[id].val=(seg[id*2].val+seg[id*2+1].val)%mod;
}
void settag(int id,tag t){
    seg[id].val=(seg[id].val*t.mul+t.add*seg[id].sz)%mod;  //这里更新区间要乘上区间长
度

    seg[id].t=seg[id].t+t;
}
void pushdown(int id){
    if(seg[id].t.mul!=1||seg[id].t.add!=0){  //乘和加不为1
        settag(id*2,seg[id].t);
        settag(id*2+1,seg[id].t);
        seg[id].t={1,0};  //向左右边儿子传递标记
    }
}
void bulid(int id,int l,int r){
    seg[id].t={1,0};     //这里要赋初值
    seg[id].sz=r-l+1;    //表示线段树节点大小
    if(l==r) seg[id].val={a[l]};
    else {
        int mid=l+r>>1;
        bulid(id*2,l,mid);
        bulid(id*2+1,mid+1,r);
        update(id);  //build这里别忘了更新
    }
}
//当递归到子区间时候直接把懒标记更新了就行
void modify(int id,int l,int r,int ql,int qr,tag t){
    if(l==ql&&qr==r){
        settag(id,t);
        return ;
    }else{
        int mid=l+r>>1;
        pushdown(id);  //对于向下遍历的点也要加上标记
        if(qr<=mid){ modify(id*2,l,mid,ql,qr,t);
        }else if(ql>=mid+1) modify(id*2+1,mid+1,r,ql,qr,t);
        else{
            modify(id*2,l,mid,ql,mid,t);
            modify(id*2+1,mid+1,r,mid+1,qr,t);
        }
        update(id); //还要跟新这个点的信息
```

```
    }
}
ll query(int id,int l,int r,int ql,int qr){
    if(l==ql&&r==qr) return seg[id].val;
        int mid=l+r>>1;
        pushdown(id); //push的树上的标记是id
        // [l,mid] [mid+1,r];
        if(qr<=mid){ return query(id*2,l,mid,ql,qr);
        }else if(ql>=mid+1) return query(id*2+1,mid+1,r,ql,qr);
        else{
            return
(query(id*2,l,mid,ql,mid)+query(id*2+1,mid+1,r,mid+1,qr))%mod;
        }
}
void solve(){
    int m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    bulid(1,1,n);
    while(m--){
        int t;
        scanf("%d",&t);
        if(t==1){
            int l,r,d;
            scanf("%d%d%d",&l,&r,&d);
            modify(1,1,n,l,r,{1,d});
        }else if(t==2){
            int l,r,d;
            scanf("%d%d%d",&l,&r,&d);
            modify(1,1,n,l,r,{d,0});
        }else if(t==3){
            int l,r,d;
            scanf("%d%d%d",&l,&r,&d);
            modify(1,1,n,l,r,{0,d});
        }else{
            int l,r;
            scanf("%d%d",&l,&r);
            printf("%lld\n",query(1,1,n,l,r));
        }

    }
}
```

**扫面线 二维数点问题 基础扫描线（可持续化树状数组）**

平面上有$n$个点$(x_i, y_i)$。

回答$q$个询问，每个询问给定一个矩形$[X_1, X_2]\times[Y_1, Y_2]$，询问矩形里面有多少个点。

```
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
#define y1 abcdewqw   //这里是防止把y1读成关键字,换变量名
int n,q;
vector<array<int,4>> event; //把查询和插入当作事件，记录所有的事件
```

```cpp
vector<int> vx;
int c[N],ans[N];
int lowbit(int x){
    return x&-x;
}
int query(int i){
    int sum=0;
    for(;i;i-=lowbit(i)) sum+=c[i];
    return sum;
}
void add(int i,int x){
    for(;i<=n;i+=lowbit(i)) c[i]+=x;
}
//重要的关键是离散化和事件的表示
void solve(){
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++){
        int x,y;
        scanf("%d%d",&x,&y);
        vx.push_back(x);           //离散化 x
        event.push_back({y,0,x}); //关于x建立离散化的坐标轴，所有按y轴sort
        //y坐标，类型，x坐标
    }
    for(int i=1;i<=q;i++){
        int x1,x2,y1,y2;
        scanf("%d%d%d%d",&x1,&x2,&y1,&y2); //查询矩阵范围
        //范围<=x2 <=y2 减<=x1-1 减<=y1-1 -...+.. 树状数组只能算<=（二维前缀和）
        //y坐标 1- 2+，x坐标,询问是哪一个询问
        // 1和2也不是随便选的 应该是先把点加入再查询所以是 0 1 2
        //选状态表示时应该要考虑当y相同时哪一个先发生
        event.push_back({y2,2,x2,i});
        event.push_back({y1-1,2,x1-1,i});
        event.push_back({y2,1,x1-1,i});
        event.push_back({y1-1,1,x2,i});
    }
    sort(event.begin(),event.end());
    sort(vx.begin(),vx.end());
    vx.erase(unique(vx.begin(),vx.end()),vx.end());
    int m=vx.size();
    //y坐标只表示顺序，排完序后没有用了
    //array<typedef,size> 和pai<int,int>一样也是按第1位和第2位往后比较
    for(auto evt:event){
        if(evt[1]==0){
            int y=lower_bound(vx.begin(),vx.end(),evt[2])-vx.begin()+1;
            //找到第一个比x小的数，就是他的位置(vector下标从0开始),加1防止 树状数组死循环
            add(y,1);
        }else{
            int y=upper_bound(vx.begin(),vx.end(),evt[2])-vx.begin();
            //要在vx里找到第一个<=他的 那就是第一>他的位置减一
            int tmp=query(y);
            if(evt[1]==1) ans[evt[3]]-=tmp;
            else ans[evt[3]]+=tmp;
        }
    }
    for(int i=1;i<=q;i++){
```

```
        printf("%d\n",ans[i]);
    }
    return ;
}
```

**矩形面积并 可持续化线段树**

平面上有$n$个矩形$[Xi,1,Xi,2]\times[Yi,1,Yi,2]$，也就是左下角$(Xi,1,Yi,1)$右上角$(Xi,2,Yi,2)$的矩形。问面积的并是多少。

```
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
int n,m;
vector<int> vx;
vector<array<int,4>> event;
struct Node{
    int minv,mincnt;
    int t;
} seg[N*8]; //离散化出来就有两倍端点

void update(int id){
    seg[id].minv=min(seg[id*2].minv,seg[id*2+1].minv);
    if(seg[id*2].minv==seg[id*2+1].minv)
seg[id].mincnt=seg[id*2].mincnt+seg[id*2+1].mincnt;
    else if(seg[id*2].minv<seg[id*2+1].minv) seg[id].mincnt=seg[id*2].mincnt;
    else seg[id].mincnt=seg[id*2+1].mincnt;
}
void settag(int id,int t){
    seg[id].minv=seg[id].minv+t;
    seg[id].t=seg[id].t+t;
}
void pushdown(int id){
    if(seg[id].t!=0){
        settag(id*2,seg[id].t);
        settag(id*2+1,seg[id].t);
        seg[id].t=0;
    }
}
void build(int id,int l,int r){
    if(l==r) seg[id]={0,vx[r]-vx[r-1]}; //这里的mincnt是长度
    else{
        int mid=l+r>>1;
        build(id*2,l,mid);
        build(id*2+1,mid+1,r);
        update(id);
    }
}
//int query(int id,int l,int r,int ql,int qr) 只对整段查询
void  modify(int id,int l,int r,int ql,int qr,int t){
    if(ql==l&&qr==r){
        settag(id,t);
        return ;
    }else{
```

```
        int mid=l+r>>1;
        pushdown(id);
        if(qr<=mid)  modify(id*2,l,mid,ql,qr,t);
        else if(ql>=mid+1) modify(id*2+1,mid+1,r,ql,qr,t);
        else{
            modify(id*2,l,mid,ql,mid,t);
            modify(id*2+1,mid+1,r,mid+1,qr,t);
        }
        update(id);
    }
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        int x1,x2,y1,y2;
        scanf("%d%d%d%d",&x1,&x2,&y1,&y2);
        vx.push_back(x1);
        vx.push_back(x2);
        event.push_back({y1,1,x1,x2});
        event.push_back({y2,-1,x1,x2}); //这里插入删除操作无所为写成1/-1
    }
    sort(vx.begin(),vx.end());
    vx.erase(unique(vx.begin(),vx.end()),vx.end());
    sort(event.begin(),event.end());
    m=vx.size()-1;
    build(1,1,m);
    int prey=0; //记录前一个x的坐标
    int totlen=seg[1].mincnt;
    ll ans=0;
    for(auto evt:event){
        int cov=totlen; //覆盖的长度
         if(seg[1].minv==0) cov=totlen-seg[1].mincnt;
        ans+=(ll)(evt[0]-prey)*cov;//?
        prey=evt[0];
        int x1=lower_bound(vx.begin(),vx.end(),evt[2])-vx.begin()+1;
        int x2=lower_bound(vx.begin(),vx.end(),evt[3])-vx.begin();//右端点不加一
        if(x1>x2) continue; //防止死循环 l<r;
        modify(1,1,m,x1,x2,evt[1]);
    }
    printf("%lld\n",ans);
    return 0;
}
```

**字典树查询字符串，异或和**

```
#include<iostream>
using namespace std;
const int N=1e5+10;
int p[N][27];
int n;
int idx=1;
int cnt[N];
void insert(string str){
    int s=0;
    for(int i=0;i<str.size();i++){
```

```
        int u=str[i]-'a';
        if(!p[s][u]) p[s][u]=idx++;
        s=p[s][u];
    }
    cnt[s]++;
}
int query(string str){
    int s=0;
    for(int i=0;i<str.size();i++){
        int u=str[i]-'a';
        if(!p[s][u]) return 0;
        s=p[s][u];
    }
    return cnt[s];
}
int main(){
    cin>>n;
    string op,str;
    while(n--){
        cin>>op>>str;
        if(op[0]=='I'){
            insert(str);
        }else cout<<query(str)<<endl;
    }
    return 0;
}
////////
#include<iostream>
using namespace std;
typedef long long ll;
const int N=1e5+10;
ll a[N];
int son[30*N][3],idx=1;
void insert(ll x){
    int p=0;
    for(int i=30;i>=0;i--){
        int t=(x>>i)&1;
        if(!son[p][t]) son[p][t]=idx++;
        p=son[p][t];
    }
}
ll query(ll x){
    int p=0;
    ll ans=0;
    for(int i=30;i>=0;i--){
        int t=(x>>i)&1;
        x-=(t<<i);
        if(son[p][!t]){
            ans+=(1<<i);
            p=son[p][!t];
        }else{
            p=son[p][t];
        }
    }
    return ans;
```

```
}
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        insert(a[i]);
    }
    ll ans=0;
    for(int i=1;i<=n;i++){
        ans=max(ans,query(a[i]));
    }
    cout<<ans<<endl;
    return 0;
}
```

**异或第K小 字典树**

给$n$个数字$a1, a2, \ldots, an$。

你要回答$m$个询问，每次给定两个数$x, k$，询问$a_1 xor x, a_2 xor x, \ldots, a_n xor x$中从小到大排序中第k小的元素。

```
using namespace std;
const int N=2e5+10,M=11*N;
typedef long long ll;

int a[N];
int idx;
struct Node{
    int sz;
    int s[3];
}seg[N*32];
void solve(){
    int n,m;
    scanf("%d%d",&n,&m);
    int root=++idx;
    for(int i=1;i<=n;i++){
        ll x;
        scanf("%lld",&x);
        int p=1;
        for(int j=30;j>=0;j--){
            seg[p].sz+=1;
            int w=(x>>j)&1;
            if(seg[p].s[w]==0) seg[p].s[w]=++idx;
            p=seg[p].s[w];
        }
        seg[p].sz+=1;
    }
    for(int i=0;i<m;i++){
        int x,k;
        scanf("%d%d",&x,&k);
        int p=root;
```

```
        int ans=0;
        for(int j=30;j>=0;j--){
            int t=(x>>j)&1;
            if(seg[seg[p].s[t]].sz>=k) p=seg[p].s[t];       //这里应该是下一层的小的大
小
            else{
                k-=seg[seg[p].s[t]].sz;
                //这里要减去大小
                p=seg[p].s[1^t];
                ans+=(1<<j);
            }
        }
        printf("%d\n",ans);
    }
    return ;
}
int main(){
#ifdef LOCAL
    freopen("E:/input.txt","r",stdio);
#endif
    solve();
    return 0;
}
```

**莫队 算法**

> 一个数列，每次给出一个询问求【L,R】 的区间和，强制使用莫队来做 【2，5】 区间已知 求 【2，6】 直接+a[6] 就行


https://www.bilibili.com/video/BV1zE411673h/?spm_id_from=333.337.search-card.all.click

**zoto**

题目大意：在二维平面内有 n 个点，表示为 ( i , f [ i ] ) 需要回答 m 次询问，每次询问会给出一个矩形，问矩形内有多少个不同的 y 值

```
//https://acm.hdu.edu.cn/showproblem.php?pid=6959
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
#define debug freopen("1.in", "r", stdin), freopen("1.out", "w", stdout)
#define mem(a,b) memset(a,b,sizeof(a))
#define inv(a,p) fpow(a,p-2,p)
#define max(a,b) (a>b?a:b)
#define min(a,b) (a<b?a:b)
#define lb(i) (i&-i)
#define pw(a) (1LL<<a)
#define ls(a) (a<<1LL)
#define rs(a) (a<<1LL|1LL)
#define make make_pair
#define PP push_back
#define xx first
#define yy second
using namespace std;
```

```
typedef long long LL;
typedef double DD;
typedef bool BB;
typedef pair<string,LL> PAIR;

const LL INF=1e18;
const LL mod=1e9+7;
const LL N=2e5+10;

LL n,m,k,T,sum,bkn,l,r,vl,vr,Max;
LL tre[N];
LL arr[N];
LL ans[N],cnt[N];
LL id[N];

struct Node
{
    LL l,r,vl,vr,num;
    bool operator <(const Node x) const{
        if(id[l]!=id[x.l]) return id[l]<id[x.l];
        return r<x.r;
    }
}q[N];

void modify(LL x,LL w)
{
    if(x<=0) return ;
    for(LL i=x;i<=n;i+=lb(i)) tre[i]+=w;
}
LL get(LL x)
{
    LL res=0;
    for(LL i=x;i>=1;i-=lb(i)) res+=tre[i];
    return res;
}
void add(LL x)
{
    cnt[x]++;
    if(cnt[x]==1) modify(x,1);
}
void remove(LL x)
{
    cnt[x]--;
    if(cnt[x]==0) modify(x,-1);
}
LL gcd(LL a,LL b){return (b==0)?a:gcd(b,a%b);}
int main(void)
{
    scanf("%lld",&T);
    while(T--)
    {
        scanf("%lld%lld",&n,&m);
        for(LL i=0;i<=Max;i++) cnt[i]=0;
        for(LL i=1;i<=n;i++) tre[i]=0;
```

```
        bkn=sqrt(n);
        for(LL i=1;i<=n;i++)
        {
            scanf("%lld",&arr[i]);
            arr[i]++;
            Max=max(Max,arr[i]);
            id[i]=(i-1)/bkn+1;
        }
        for(LL i=1;i<=m;i++)
        {
            scanf("%lld%lld%lld%lld",&l,&vl,&r,&vr);
            q[i]={l,r,vl+1,vr+1,i};
        }
        sort(q+1,q+m+1);
        l=r=0;
        for(LL i=1;i<=m;i++)
        {
            while(l<q[i].l) remove(arr[l++]);
            while(l>q[i].l) add(arr[--l]);
            while(r<q[i].r) add(arr[++r]);
            while(r>q[i].r) remove(arr[r--]);
            ans[q[i].num]=get(q[i].vr)-get(q[i].vl-1);
        }
        for(LL i=1;i<=m;i++) printf("%lld\n",ans[i]);
    }
    system("pause");
    return 0;
}
///自己的
#include<iostream>
#include<algorithm>
#include<array>
using namespace std;
const int N=1e5+100;
typedef long long ll;
typedef array<int,6> ari4;
int a[N],ans[N];
int cnt[N];  //记录每个点的种类
int tr[N*4];
ari4 query[N];
int n,m;
// 题目大意在（数组下表）区间 [l,r] 中 ，介于 [ql,qr] 之间的数的种类  kind[qr]-kind[ql-1],用树状数组维护
int B=500;
bool cmp(const ari4 &a,const ari4 &b){
    if(a[0]/B!=b[0]/B) return a[0]/B<b[0]/B;
    return a[1]<b[1];
}
int lowbit(int x){
    return x&-x;
}
void addtr(int x,int t){
    for(;x<=n;x+=lowbit(x)) tr[x]+=t;
}
int qtr(int x){
```

```
    int sum=0;
    for(;x;x-=lowbit(x)) sum+=tr[x];
    return sum;
}
void add(int x){
    if(cnt[a[x]]==0) addtr(a[x],1);
    cnt[a[x]]++;
}
void del(int x){
    cnt[a[x]]--;
    if(cnt[a[x]]==0) addtr(a[x],-1);
}
void solve(){
    int maxn=0;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) {
        scanf("%d",&a[i]);
        a[i]++; //用权值树状数组 避免0要+1;
        cnt[i]=0;
        tr[i]=0;
        maxn=max(a[i],maxn);
    }
    n=maxn;
    for(int i=1;i<=m;i++){
        int l,r,ql,qr;
        scanf("%d%d%d%d",&l,&ql,&r,&qr);
        ql++,qr++; //查询边界也要+1
        query[i]={l,r,ql,qr,i};
    }
    sort(query+1,query+m+1,cmp);
    int l=0,r=0;
    for(int i=1;i<=m;i++){
        int tl=query[i][0],tr=query[i][1],ql=query[i][2],qr=query[i]
[3],j=query[i][4];
        // [l,r] [tr,tl];
        while(tl<l) add(--l);
        while(tr>r) add(++r);
        while(tl>l) del(l++);
        while(tr<r) del(r--);
        ans[j]=qtr(qr)-qtr(ql-1);
    }
    for(int i=1;i<=m;i++) printf("%d\n",ans[i]);
    return ;
}
int main(){
    #ifdef LOCAL
    freopen("E:/input.txt","r",stdio);
    #endif
    int t;
    scanf("%d",&t);
    while(t--) solve();
    system("pause");
    return 0;
}
```

**小Z的袜子 莫队板子**

> 加一个数/删一个数好做，但是合并区间难做类似上面的例题，把区间查询变成区间的扩展，先把区间分块
>
> 怎么维护一个分块的区间，对于一个不在块里的区间，边界直接从上个区间的右边界加到下个区间的左，解决一个询问需要2*sqrt（n）的代价（移动边界），把求区间信息变成边界的移动，所以把查询区间 sort一遍可以大大加快查询速度

```
// http://oj.daimayuan.top/course/15/problem/768
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=5e4+100;
int n,q;
ll ansq[N],ans[N];
int a[N];
ll cnt[N];
array<int,4> qr[N];
ll gcd(ll a,ll b){
    if(b == 0) return a;
    return gcd(b, a % b);
}
 /* bool cmp(ari4 c, ari4 b){
    int t=c[0]/B;
    if((c[0]/B)!=(b[0]/B)) return c[0]<b[0];  //分奇数前偶数后可以优化时间复杂度
    if(t&1) return c[1]<b[1];
    else return c[1]>b[1];
  }*/
int main(){
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=0;i<q;i++){
        int l,r;
        scanf("%d%d",&l,&r);
        qr[i]={l,r,i};
        ansq[i]=(ll)(r-l)*(r-l+1)/2; //这里要开 ll
    }
    int B=500;
    sort(qr,qr+q,[&](array<int,4> b,array<int,4> c){
        if(b[0]/B!=c[0]/B) return b[0]/B<c[0]/B;  //看在不在同一个分块里
        return b[1]<c[1];
    }); //匿名函数
    ll tmp=0;
    auto add=[&](int x){
        tmp+=cnt[a[x]]; //算的是对数，该位数与前面有的加
        cnt[a[x]]++;
    };
    auto del=[&](int x){
        cnt[a[x]]--;
        tmp-=cnt[a[x]];
    };
    int l=1,r=0; //初始化
    for(int i=0;i<q;i++){
        int tl=qr[i][0],tr=qr[i][1],t=qr[i][2];
```

```
        // 已知[l,r],要算[tl,tr];
        while(tl<l) add(--l);
        //先左移再加上，因为是已知区间[L,R];
        //是闭区间所以先移动后加，因为本位上算过了
        while(tr>r) add(++r);
        while(tl>l) del(l++); //当前点也要删，先删再移动i
        while(tr<r) del(r--);
        ans[t]=tmp;
    }
    for(int i=0;i<q;i++){
        ll d=gcd(ans[i],ansq[i]);
        printf("%lld/%lld\n",ans[i]/d,ansq[i]/d);
    }
    return 0;
}
```

**第K大数查询 链表**

给定一个1~n的排列$a_1,a_2,a_3,...,a_n，想知道所有的区间[l,r][�,�]满足r−l+1≥k，这样的区间里的数中第k大的元素，输出它们的和。

```
#include<iostream>
#include<cmath>
#include<algorithm>
#include<queue>
#include<map>
#include<vector>
#include<string>
#include<cstring>
#include<deque>
#include<random>
#include<ctime>
#include<array>
#define fi first
#define se second7
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e6+10;
#define y1 abcdewqwa
#define left ajghkjgad
#define right asdklksgahlkh
int l[N],r[N];
int left[N],right[N];
int pos[N];
ll ans=0;
void solve(){
    int n,k;
    scanf("%d%d",&n,&k);
    int x;
    for(int i=1;i<=n;i++) scanf("%d",&x),pos[x]=i; //做一次hash
    for(int i=0;i<=n+1;i++){
        r[i]=max(i-1,0);
```

```
        l[i]=min(n+1,i+1);  //下标
    }
    for(int i=1;i<=n;i++){
        int x=pos[i];
        left[0]=x;
        for(int j=1;j<=k;j++){
            x=l[x];
            left[j]=x;
        }
        x=pos[i];
        right[0]=x;
        for(int j=1;j<=k;j++){
            x=r[x];
            right[j]=x;
        }
        x=pos[i];
        l[r[x]]=l[x];
        r[l[x]]=r[x];
        ll seg=0;
        for(int j=1;j<=k;j++){
            seg+=(ll)(left[j-1]-left[j])*(right[k-j+1]-right[k-j]);
        }
        ans+=seg*i;
    }
    printf("%lld",ans);
}
int main(){
#ifdef LOCAL
    freopen("E:/input.txt", "r", stdin);
#endif
   solve();
    return 0;
}
```

**重复的数 莫队**

给定一个数列$A=(a_1,a_2,\ldots,a_n)$，给出若干询问，每次询问某个区间$[l_i,r_i]$内恰好出现$k$次的数有多少个。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
typedef long long ll;
typedef array<int,5> ari4;
int a[N],ans[N];
int cnt[N];
int s[N];
ari4 query[N];
bool cmp(const ari4& a,const ari4 &b){
    if(a[0]/500!=b[0]/500) return a[0]/500<b[0]/500;
    return a[1]>b[1];
}
int tmp=0;
```

```
int n,m;
void add(int x){
    if(cnt[a[x]]!=0)s[cnt[a[x]]]--;
    cnt[a[x]]++;
    s[cnt[a[x]]]++;
}
void del(int x,int y){
    if(cnt[a[x]]!=0)s[cnt[a[x]]]--;
    cnt[a[x]]--;
    s[cnt[a[x]]]++;
}
// 把次数变成权值数组
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    scanf("%d",&m);
    for(int i=1;i<=m;i++){
        int l,r,x;
        scanf("%d%d%d",&l,&r,&x);
        query[i]={l,r,x,i};
    }
    sort(query+1,query+m,cmp); //可以想想cmp函数这么写
    int l=1,r=0;
    for(int i=1;i<=m;i++){
        int tl=query[i][0],tr=query[i][1],x=query[i][2],t=query[i][3];
        // [l,r] [tl,tr]
        while(tl<l) add(--l);
        while(tr>r) add(++r);
        while(tl>l) del(l++);
        while(tr<r) del(r--);
        ans[t]=s[x];
    }
    for(int i=1;i<=m;i++) printf("%d\n",ans[i]);

    return 0;
}
```

**分块powerful array**

给定一个长度N的数组a

要求:询问区间1<=L<=R<=N中，每个数字出现次数的平方与当前数字的乘积和

```
//https://vjudge.net/problem/CodeForces-86D
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
#define debug freopen("1.in", "r", stdin), freopen("1.out", "w", stdout)
#define mem(a,b) memset(a,b,sizeof(a))
#define inv(a,p) fpow(a,p-2,p)
#define max(a,b) (a>b?a:b)
#define min(a,b) (a<b?a:b)
#define pw(a) (1LL<<a)
#define ls(a) (a<<1LL)
#define rs(a) (a<<1LL|1LL)
```

```
#define make make_pair
#define PP push_back
#define xx first
#define yy second
using namespace std;

typedef long long LL;
typedef double DD;
typedef bool BB;
typedef pair<string,LL> PAIR;

const LL INF=1e18;
const LL mod=1e9+7;
const LL N=2e5+10;

LL n,m,k,t,T,now,sum,l,r;
LL arr[N],cnt[10*N],ans[N];
LL id[N];

struct Node
{
    LL l,r;
    LL num;
    bool operator <(const Node x) const{
        if(id[l]!=id[x.l]) return id[l]<id[x.l];
        return r<x.r;
    }
}q[N];

void remove(LL x)
{
    now-=(2*cnt[x]-1)*x;
    cnt[x]--;
}
void add(LL x)
{
    now+=(2*cnt[x]+1)*x;
    cnt[x]++;
}
LL gcd(LL a,LL b){return (b==0)?a:gcd(b,a%b);}
int main(void)
{
    scanf("%lld%lld",&n,&m);
    for(LL i=1;i<=n;i++) scanf("%lld",&arr[i]);
    LL bk=LL(sqrt(n));
    for(LL i=1;i<=n;i++) id[i]=(i-1)/bk+1;
    for(LL i=1;i<=m;i++)
    {
        scanf("%lld%lld",&q[i].l,&q[i].r);
        q[i].num=i;
    }
    sort(q+1,q+m+1);
    for(LL i=1;i<=m;i++)
    {
        while(l<q[i].l) remove(arr[l++]);
```

```
        while(l>q[i].l) add(arr[--l]);
        while(r>q[i].r) remove(arr[r--]);
        while(r<q[i].r) add(arr[++r]);
        ans[q[i].num]=now;
    }
    for(LL i=1;i<=m;i++) cout<<ans[i]<<endl;
    system("pause");
    return 0;
}
```

题目大意：给出n nn个图层的RGB参数，其参数用十六进制表示，有两个操作：
m = 1 m=1m=1 ， 总图层为这层的RGB参数，
m = 2 m=2m=2 ， 总图层为[ l , r ] [l,r][l,r]的RGB总和，(不会超过255)。
有q qq次询问，求[ l , r ] [l,r][l,r]的总图层RGB
思路：多次询问求和区间，很容易想到是前缀和，但操作难度就在于十六进制相互转换，赛后才知道有%X直接成十六进制，用r [ i ] , g [ i ] , b [ i ] r[i],g[i],b[i]r[i],g[i],b[i]分别表示到这层的RGB总和，并用pos记录到当前i ii最近的操作为一的位置，若右端点对应的m等于1，直接输出当前图层，否则取m a x ( l , p o s ) max(l,pos)max(l,pos)，最后输出判断有没有超过255

```
1
4 2
1 0 3 1
1 0 4 3
1 0 4 2
// 3 2
//https://acm.dingbacode.com/contests/contest_showproblem.php?pid=1007&cid=986
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
#define debug freopen("1.in", "r", stdin), freopen("1.out", "w", stdout)
#define mem(a,b) memset(a,b,sizeof(a))
#define inv(a,p) fpow(a,p-2,p)
#define max(a,b) (a>b?a:b)
#define min(a,b) (a<b?a:b)
#define pw(a) (1LL<<a)
#define ls(a) (a<<1LL)
#define rs(a) (a<<1LL|1LL)
#define make make_pair
#define PP push_back
#define xx first
#define yy second
using namespace std;

typedef long long LL;
typedef double DD;
typedef bool BB;
typedef pair<string,LL> PAIR;

const LL INF=1e18;
const LL mod=1e9+7;
const LL N=2e5+10;

LL n,m,k,T,cnt,q,l,r,t,bkn;
struct Node
{
    LL r,g,b;
```

```
    LL mo;
}st[N],sum[350],ans;

LL id[N];
bool fla[350];
LL num[300];

char s[10];
char alp[]="0123456789ABCDEF";
char c1,c2;
void cal(LL x)
{
    c1=alp[x%16];
    x/=16;
    c2=alp[x%16];
    return ;
}

Node qur(LL l,LL r)
{
    Node ans={0,0,0,0};
    if(id[l]==id[r])
    {
        for(LL i=r;i>=l;i--)
        {
            ans.r+=st[i].r;ans.g+=st[i].g;ans.b+=st[i].b;
            if(st[i].mo==1) return ans;
        }
        return ans;
    }
    LL j=r;
    while(id[j]==id[r])
    {
        ans.r+=st[j].r;ans.g+=st[j].g;ans.b+=st[j].b;
        if(st[j].mo==1) return ans;
        j--;
    }
    for(LL k=id[j];k>=id[l]+1;k--)
    {
        if(ans.r>=255&&ans.g>=255&&ans.b>=255) return ans;
        if(fla[k])
        {
            while(id[j]==k)
            {
                ans.r+=st[j].r;ans.g+=st[j].g;ans.b+=st[j].b;
                if(st[j].mo==1) return ans;
                j--;
            }
            return ans;
        }
        ans.r+=sum[k].r;ans.g+=sum[k].g;ans.b+=sum[k].b;
        j-=bkn;
    }
    while(j>=l)
    {
```

```
        ans.r+=st[j].r;ans.g+=st[j].g;ans.b+=st[j].b;
        if(st[j].mo==1) return ans;
        j--;
    }
    return ans;
}
LL gcd(LL a,LL b){return (b==0)?a:gcd(b,a%b);}
int main(void)
{
    for(LL i=0;i<16;i++) num[alp[i]]=i;
    scanf("%lld",&T);
    while(T--)
    {
        mem(sum,0);
        mem(fla,0);
        scanf("%lld%lld",&n,&q);
        bkn=sqrt(n);
        for(LL i=1;i<=n;i++) id[i]=(i-1)/bkn+1;

        for(LL i=1;i<=n;i++)
        {
            scanf("%lld%s",&st[i].mo,s);
            if(st[i].mo==1) fla[id[i]]=1;
            st[i].r=num[s[0]]*16+num[s[1]];
            st[i].g=num[s[2]]*16+num[s[3]];
            st[i].b=num[s[4]]*16+num[s[5]];
            sum[id[i]].r+=st[i].r;
            sum[id[i]].g+=st[i].g;
            sum[id[i]].b+=st[i].b;
            // cout<<st[i].r<<' '<<st[i].g<<' '<<st[i].b<<endl;
        }
        while(q--)
        {
            // ans={0,0,0,0};
            scanf("%lld%lld",&l,&r);
            ans=qur(l,r);
            // cout<<ans.r<<' '<<ans.g<<' '<<ans.b<<endl;
            ans.r=min(ans.r,255);
            ans.g=min(ans.g,255);
            ans.b=min(ans.b,255);
            cal(ans.r);
            printf("%c%c",c2,c1);
            cal(ans.g);
            printf("%c%c",c2,c1);
            cal(ans.b);
            printf("%c%c\n",c2,c1);
        }
    }
    system("pause");
    return 0;
}
```

**分块 序列维护**

[序列维护 - 题目 - Daimayuan Online Judge]

你要维护n个序列$p_1, p_2, \ldots, p_n$，一开始都是空的。接下来需要支持$q$个操作：

1. `1 x y`，往序列$p_x$末尾插入一个数$y$，每个序列的下标都是从0开始的。
2. `2 z`，对于所有非空的序列$p_i$，求出$p_{i,j}\ j = z mod |pi|$和，其中$|pi|$表示序列$p_i$中数字个数。

```
#include<bits/stdc++.h>
#define fi first
#define se second7
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10,M=510;
#define y1 abcdewqwa
#define left ajghkjgad
#define right asdklksgahlkh
vector<int> g[N],large;
ll sum[M+22][M+22];
//分块神题
//把体积小的序列直接加起来直接求和了
//更新小体积时，先把原来位置上的数剪掉
//再加入数更新到下一个体积大小的位置
//超过M视为大体积直接挨个算
vector<ll> ans;
void solve(){
    int n,q;
    scanf("%d%d",&n,&q);
    while(q--){
        int op,x,y;
        scanf("%d%d",&op,&x);
        if(op==1){
            scanf("%d",&y);
            int t=g[x].size();
            if(t<=M){
            for(int i=0;i<t;i++) sum[t][i]-=g[x][i];  //本层先减
            g[x].push_back(y);
            t++;  //多一个数体积大一
            if(t<=M) for(int i=0;i<t;i++) sum[t][i]+=g[x][i]; //下一层加
            else large.push_back(x); //视为大体积放到large记录下标
        }else g[x].push_back(y);
        }else{
            ll tmp=0;
            for(int i=1;i<=M;i++) tmp+=sum[i][x%i];  //先求小区间里的
            for(auto y:large) tmp+=g[y][x%g[y].size()]; //再求大区间里的
            ans.push_back(tmp);
        }
    }
    for(auto t:ans){
        printf("%lld\n",t);
    }
    return ;
}
int main(){
```

```
#ifdef LOCAL
    freopen("E:/input.txt", "r", stdin);
#endif
   solve();
    return 0;
}
```

**笛卡尔树**

```
using namespace std;
const int N=1e6+10;
typedef long long ll;
int a[N];
int l[N],r[N];
int ans[N];
int tot=0;
int n;
void dfs(int u){
    ans[u]=++tot;
    if(l[u]) dfs(l[u]);
    if(r[u]) dfs(r[u]);
    //只要迪卡尔树相同就是符合题意的
    //根节点是一定要小的先序遍历
}
void build(){
    int root=0,last;
    stack<int> ss;
    for(int i=1;i<=n;i++){
        last=0;
        while(ss.size()&&a[ss.top()]>a[i]){
            last=ss.top();
            ss.pop();
        }//单调栈的写法
        if(ss.empty()) root=i;
        else r[ss.top()]=i; //这里是把这个点放到前一个最小值的右边
        l[i]=last; //没有last为0
        ss.push(i);
    }
    dfs(root);
}
void solve(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    build();
    for(int i=1;i<=n;i++) printf("%d ",ans[i]);
}
int main(){
#ifdef LOCAL
    freopen("E:/input.txt","r",stdio);
#endif
   solve();
    return 0;
```

```
}
```

**RMQ**

nlogn 预处里，O（1）查询区间最大值

```
using namespace std;
const int N=1010000;
typedef long long ll;
typedef pair<int,int> PII;
unsigned int A, B, C;
unsigned int f[22][N],a[N],ans;
int n,q;
inline unsigned int rng61() {
    A ^= A << 16;
    A ^= A >> 5;
    A ^= A << 1;
    unsigned int t = A;
    A = B;
    B = C;
    C ^= t ^ A;
    return C;
}
int main(){
    scanf("%d%d%u%u%u", &n, &q, &A, &B, &C);
    for (int i = 1; i<=n; i++) {
        a[i] =rng61();
        f[0][i]=a[i];
    }
    for(int j=1;j<=20;j++){
        for(int i=1;i+(1<<j)-1<=n;i++){
            f[j][i]=max(f[j-1][i],f[j-1][i+(1<<(j-1))]);
        }
    }
    for (int i = 1; i <= q; i++) {
        unsigned int l = rng61() % n + 1, r = rng61() % n + 1;
        if (l > r) swap(l, r);
        int len=__lg(r-l+1);
        ans^=max(f[len][l],f[len][r-(1<<len)+1]);
    }
    printf("%u",ans);
    return 0;
}
/*
void ST_prework()
{
    for(int i = 1; i <= 20; i ++){  //倍增处理不同长度区间
        for(int j = 1; j + (1 << i) - 1 <= n; j ++)
            f[j][i] = min(f[j][i-1], f[j + (1 << (i-1))][i-1]);
//由上一个长度递推取值
    }
    lg[1] = 0;
    for(int i = 2; i <= n; i ++)    lg[i] = lg[i/2] + 1;
//预处理lg函数
}
```

```
unsigned int ST_query(int l, int r){
    int len = lg[r - l + 1];
    return min(f[l][len], f[r - (1<<len)+1][len]);
}
*/
/*
ll query(int l,int r){
    int len=__lg(r-l+1);
    return min(f[len][l],f[len][r-(1<<len)+1]);
}
int main(){
    cin>>n>>c;
    for(int i=1;i<=n;i++){
        cin>>f[0][i];
    }
    for(int j=1;j<=20;j++){
        for(int i=1;i+(1<<j)-1<=n;i++){
            f[j][i]=min(f[j-1][i],f[j-1][i+(1<<(j-1))]);
        }
    }
   */
```

**DFS序 树上改点权查询点权和**

给一棵$n$个点的树，每个点有点权$w_i$，1号点为根。给$q$个操作：

1. `1 x y`，将$x$点的点权改成$y$。
2. `2 x`，询问$x$点子树的点权和，到根的路径的点权和（都包含x这个点）。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=2e5+10;
int a[N],l[N],r[N];
int tot=0;
int n,q;
ll c1[N*4],c2[N*4];
int lowbit(int x){
    return x&-x;
}
void addc1(int x,int t){
    for(;x<=n;x+=lowbit(x)) c1[x]+=t;
}
void addc2(int x,int t) {
    for(;x<=n;x+=lowbit(x)) c2[x]+=t;
}
ll queryc1(int x){
    ll sum=0;
    for(;x;x-=lowbit(x)) sum+=c1[x];
    return sum;
}
ll queryc2(int x){
    ll sum=0;
    for(;x;x-=lowbit(x)) sum+=c2[x];
```

```
    return sum;
}

vector<int> g[N];
void dfs(int u,int fa){
    l[u]=++tot;     //从0开始先++
    for(auto t:g[u]){
        if(t==fa) continue;
        dfs(t,u);
    }
    r[u]=tot;  //这里的tot要和上面的一样
    return ;
}
void solve(){
    // 把树上问题转化成区间问题，每次可以看成对区间的修改
    // 用树状数组作查询与修改
    scanf("%d%d",&n,&q);
    for(int i=1;i<n;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(1,0);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=1;i<=n;i++){
        addc1(l[i],a[i]);
        addc2(l[i],a[i]);
        addc2(r[i]+1,-a[i]);
    }
    while(q--){
        int op,x,y;
        scanf("%d",&op);
        if(op==1){
            scanf("%d%d",&x,&y);
            int d=y-a[x];
            a[x]=y;
            addc1(l[x],d);
            addc2(l[x],d);
            addc2(r[x]+1,-d);
        }else{
            scanf("%d",&x);
            printf("%lld %lld\n",queryc1(r[x])-queryc1(l[x]-1),queryc2(l[x]));
        }
    }
    return ;
}
int main(){
    solve();
    return 0;
}
```

**DFS序 查询子树点权和，并且换根**

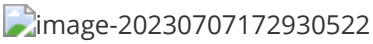
给一棵$n$个点的树，每个点有点权$w_i$，1号点为根。给$q$个操作：

1. `1 x y`，将$x$点的点权改成$y$。
2. `2 x`，询问$x$点子树的点权和。
3. `3 x`，将根换到$x$位置。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int,int> PII;
const int N=2e5+10;
const ll mod=1e9+7;
vector<int> g[N];
vector<PII> vx[N];
int l[N],r[N],w[N];
int tot=0;
int n,q;
int root=1;
ll c[N];
int lowbit(int x){
    return x&-x;
}
void add(int x,ll t){
    for(;x<=n;x+=lowbit(x))c[x]+=t;
}
ll query(int x){
    ll ans=0;
    for(;x;x-=lowbit(x)) ans+=c[x];
    return ans;
}
void dfs(int u,int fa){
    l[u]=++tot;
    for(auto t:g[u]){
        if(t==fa) continue;
        dfs(t,u);
        vx[u].push_back({l[t],r[t]});
    }
    r[u]=tot;
}
int main() {
    scanf("%d%d",&n,&q);
    for(int i=1;i<n;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for(int i=1;i<=n;i++) scanf("%d",&w[i]);
    dfs(1,0);
    for(int i=1;i<=n;i++){
        add(l[i],w[i]);
    }
    while(q--){
```

```
        int op,x,y;
        scanf("%d",&op);
        if(op==1){
          scanf("%d%d",&x,&y);
          int d=y-w[x];
          w[x]=y;
          add(l[x],d);
        }else if(op==3){
            scanf("%d",&root);
        }else{
              scanf("%d",&x);
              if(x==root){
                printf("%lld\n",query(n));
              }else{
                if(l[x]<l[root]&&r[x]>=r[root]){ //根节点在原dfs的子树里
                    auto tmp=*prev(upper_bound(vx[x].begin(),vx[x].end(),(PII)
{l[root],r[root]}));
                      //找包含根节点的前一个节点（倍增也能求）
                    printf("%lld\n",query(n)-(query(tmp.second)-query(tmp.first-
1)));// 左边界都要减一
                  }else{
                      printf("%lld\n",query(r[x])-query(l[x]-1));
                  }
              }
        }
    }
    return 0;
}
```

**倍增 LCA**



```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int,int> PII;
const int N=2e5+10;
const int LOGN=20;
vector<PII> g[N];
bool st[N];
int a[N],deep[N],f[N][LOGN+1];
int val[N][LOGN+1];
int n,q;
void dfs(int u,int fa){
    deep[u]=deep[fa]+1;
    for(auto t:g[u]){
        int v=t.first,w=t.second;
        if(v==fa) continue;
        f[v][0]=u;
        val[v][0]=w;
        dfs(v,u);
    }
```

```
}
int query(int u,int v){
    int ans=1<<30;
    if(deep[u]>deep[v]) swap(u,v);
    int d=deep[v]-deep[u]; //v深度深向上跳
    for(int j=LOGN;j>=0;j--){
        if(d&(1<<j)){
            ans=min(ans,val[v][j]);
            v=f[v][j];
        }
    }
    if(u==v) return ans;
    for(int j=LOGN;j>=0;j--){
        if(f[u][j]!=f[v][j]){
            ans=min(ans,min(val[u][j],val[v][j]));
            u=f[u][j];
            v=f[v][j];
        }
    }
    ans=min(ans,min(val[u][0],val[v][0]));
    return ans;
}
int main(){
    scanf("%d%d",&n,&q);
    for(int i=1;i<n;i++){
        int v,u,w;
        scanf("%d%d%d",&u,&v,&w);
        g[u].push_back({v,w});
        g[v].push_back({u,w});
    }
    dfs(1,0);
    for(int j=1;j<=LOGN;j++){
        for(int i=1;i<=n;i++){
            f[i][j]=f[f[i][j-1]][j-1];  //倍增求第i个节点上的第j级祖先
            val[i][j]=min(val[i][j-1],val[f[i][j-1]][j-1]);//算两段路径的最小值
        }
    }
    for(int i=1;i<=q;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        printf("%d\n",query(u,v));
    }
    return 0;
}
```

**启发式合并 梦幻布丁**

$n$个布丁摆成一行，进行$m$次操作。每次将某个颜色的布丁全部变成另一种颜色的，然后再询问当前一共有多少段颜色。

例如颜色分别为$1,2,2,1$的四个布丁一共有3段颜色。

第一行两个整数$n,m(1 \leq n,m \leq 105)$表示布丁的个数和操作次数。

第二行$n$个数$a_1,a_2,\ldots,a_n(1 \leq a_i \leq 106))$表示第$i$个布丁的颜色。

接下来$m$行，每行描述一次操作。每行首先有一个整数$op$表示操作类型：

- 若$op = 1$，则后有两个整数$x, y$，表示将颜色$x$的布丁全部变成颜色$y$。保证$1 \leq x, y \leq 10^6$($x$可能等于$y$)
- 若$op = 2$，则表示一次询问。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
const int M=1e6+10;
typedef long long ll;
const ll mod=1e9+7;
int a[N];
vector<int> g[M];
int n,m;
int ans;
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        g[a[i]].push_back(i); //记录每种颜色的位置
    }
    for(int i=1;i<=n;i++){
     ans=ans+(a[i]!=a[i-1]); // 求一开始的位置上的颜色，会多算一个，但一定不会少算
    }
    while(m--){
        int op,x,y;
        scanf("%d",&op);
        if(op==1){
            scanf("%d%d",&x,&y);
            // 改颜色等于合并集合，小跟大
            if(x==y) continue;
            if(g[y].empty()) continue;
            if(g[x].size()>g[y].size()){
                swap(g[x],g[y]);
                // g[u].swap(g[y]);
                // 颜色没换记得换
            }
            int col=a[g[y][0]]; //跟大集合的颜色
            auto modify=[&](int p,int col){
                ans-=(a[p]!=a[p+1])+(a[p]!=a[p-1]);
                a[p]=col;  //每次改颜色合并
                ans+=(a[p]!=a[p+1])+(a[p]!=a[p-1]);
            };  //匿名函数
            for(auto t:g[x]){
                modify(t,col);
                g[y].push_back(t);
            }
            g[x].clear(); //小的要清空
        }else {
            printf("%d\n",ans);
        }
    }
    return 0;
}
```

**启发式合并维护查询**

树上路经两点路径中的最小路径

```
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
using namespace std;
typedef long long  ll;
typedef array<int,3>  AII;
typedef pair<int,int> PII;
const int N=2e5+100;
int ans[N],belong[N],f[N]; //表示点的集合归属
int n,q;
vector<int> vec[N]; //记录集合
AII E[N]; //记录最小的边权对应的节点
vector<PII> que[N]; //记录每个点查询的点
int find(int x){
    return x==f[x]?x:f[x]=find(f[x]);
}
int main(){
    ios;
    cin>>n>>q;
    for(int i=1;i<n;i++) cin>>E[i][1]>>E[i][2]>>E[i][0];
    sort(E+1,E+n);
    reverse(E+1,E+n);
    for(int i=1;i<=n;i++){
        f[i]=i;
        belong[i]=i;
        vec[i].push_back(i);
    }
    for(int i=1;i<=q;i++){
        int u,v;
        cin>>u>>v;
        que[u].push_back({v,i});
        que[v].push_back({u,i});
    }
    for(int i=1;i<n;i++){
        int u=find(E[i][1]),v=find(E[i][2]);
        //从最小的边开始加入集合
        if(vec[u].size()>vec[v].size()) swap(u,v);      //u为小要合并的集合
        for(auto t:que[u]){
            int qu=t.first,j=t.second;
            //(u，qu)对应要查询的两个位置
            if(ans[j]!=0) continue;
            //答案更新过了，不需要再遍历了
            if(belong[qu]==v){
                //要查询的点在里面，直接更新答案
              ans[j]=E[i][0];
            }else{
            //小集合的答案没有就把这个问题移动到大集合上
                que[v].push_back({qu,j});
            }
        }
        //答案更新完后要和并区间,清空小集合的所有信息
```

```
        for(auto t:vec[u]){
            vec[v].push_back(t);
            belong[t]=v; //点的归属
        }
        que[u].clear(); //为解决的问题以及转移了
        vec[u].clear();  //清空
        f[u]=v;          //更改并查集
    }
    for(int i=1;i<=q;i++){
        cout<<ans[i]<<endl;
    }
    return 0;
}
```

**DSU on tree 树上启发式合并 板子**

```
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
using namespace std;
typedef long long  ll;
typedef array<int,3>  AII;
typedef pair<int,int> PII;
const int N=2e5+100;
int n,q;
int l[N],r[N],id[N],sz[N],hs[N],tot;
//dfs序的左右边界(包括右边界不包左)，dfs第i位上的点坐标，每个节点做根的大小
// hs找所有子节点中最重的子节点（重儿子）
void dfs_init(int u,int fa){
    l[u]=++tot;
    id[tot]=u;
    sz[u]=1;
    hs[u]=-1;
    for(auto v:g[N]){
        if(v==fa) continue;
        dfs_init(v,u);
        sz[u]+=sz[v];
        for(hs[u]==-1||sz[v]>sz[hs[u]]) hs[u]=v;
    }
    r[u]=tot;
}
// v不为父节点，不为重节点就为轻儿子
// u有重儿子，就单独遍历重儿子的集合
void dfs_solve(int u,int fa,bool keep){ //keep来判断是不是为重儿子
    for(auto v:g[u]){
        if(v!=fa&&v!=hs[u]){
            dfs_solve(v,u,false);
        }
    }
    if(hs[u]!=-1){
        dfs_solve(hs[u],u,true);
    }
    for(auto v:g[u]){
        if(v!=fa&&v!=hs[u]){
            // v是轻儿子
            // 把v子树里所有点加入到重儿子的集合里
```

```
            for(int x=l[v];x<=r[v];x++){
                add(id[x]);
            }
        }
    }
    add(u);
    if(!keep){
        // 清空
        for(int x=l[u];x<=r[u];x++){
            del[id[x]];
        }
    }
}
int main(){
    ios;
    dfs_init(1,0);
    dfs_solve(1,0,false);
    return 0;
}
```

**带权并查集**

给你$n$个变量$a1, a2, \ldots, an$，一开始不知道这些数是多少。

你要执行$q$次操作。

- `1 l r x`，给你一条线索，表示$a_l - a_r = x$，如果与已知条件矛盾，那么忽略这次操作。
- `2 l r`回答$al - ar$，如果现在的线索无法推出答案，那么忽略这次操作。

```
using namespace std;
const int N=2e5+100;
typedef long long ll;
typedef pair<int,int> PII;
ll w[N];
int f[N];
//带权值的并查集只需要在路径压缩中更新权值即可
// f[f[l]]与f[l]的关系 w[l]记录路径的值
// w[l]=a[l]-a[r] f[f[l]]=a[l]-w[l]; //记录路径的值
int find(int x){
    if(f[x]==x) return x;
    int p=f[x];
    find(p);
    //这里先对x路径压缩找到父节点
    // int q=find(p); //f[x]-> q,f[p]->a;
    //q
    //w[x]: a[x]-a[p];
    // f[p]=q w[p] : a[p]-a[q];
    // f[x]=q w[x]: a[x] -a[q]
    // a[x]-a[p] +a[p]-a[q]=a[x]-a[p];
     w[x]=w[x]+w[p]; //geng'x
     return f[x]=f[p];
}
void solve(){
    int n,m;
```

```
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        f[i]=i;
        w[i]=0;
    }
    ll t=0;
    for(int i=1;i<=m;i++){
        int tms,l,r;
        scanf("%d%d%d",&tms,&l,&r);
        l=(l+t)%n+1;
        r=(r+t)%n+1;
        if(tms==1){
        int x;
        scanf("%d",&x);
        if(find(l)==find(r)) continue;
        w[f[l]]=x-w[l]+w[r];//这里不能先改父节点
        f[f[l]]=f[r];
        // w[f[l]]=a[f[l]]-a[f[r]]
        // a[l]-a[r]=x
        // a[l]-w[l]-(a[r]-w[r])=w[f[l]]
        // x-w[l]+w[r]=w[f[l]]

        }else{
            if(find(l)!=find(r)) continue;
            printf("%lld\n",w[l]-w[r]);
            t=abs(w[l]-w[r]);
        }
    }
}
int main(){
#ifdef LOCAL
    freopen("E:/input.txt","r",stdio);
#endif
    solve();
    return 0;
}
```

**等差数列加 分块1**

给你一个序列$a1, a2, \ldots, an$，一开始都是00。

你需要支持$q$个操作：

1. `1 x y d`，对数组中所有下标$i \equiv y$的位置加上$d$。
2. `2 x`，单点询问$a_x$的值。

```
using namespace std;
typedef long long ll;
const int N=2e5+10;
const int M=500;
ll tag[M+10][M+10];
ll val[N];
int main(){
    int n,q;
    scanf("%d%d",&n,&q);
    for(int i=1;i<=q;i++){
```

```
        int ty,x,y,d;
        scanf("%d%d",&ty,&x);
        if(ty==1){
            scanf("%d%d",&y,&d);
            if(x>M){
                for(int j=y;j<=n;j+=x) val[j]+=d;
                //大于sqrt(x)直接暴力算
            }else{
                tag[x][y]+=d;
                //小标记和开始标记的位置 %mod
            }
        }else{
            ll ans=val[x];
            for(int i=1;i<=M;i++){
                ans+=tag[i][x%i];
                //这里for一遍这个点可我能会打小标记的值
            }
            printf("%lld\n",ans);
        }
    }
    return 0;
}
```

**图染色 树上分块**

给一个$n$个点$m$条边的图，每个点一开始都是白色。

要求支持$q$个操作：

1. `1 x`，翻转$x$点的颜色，从黑到白，从白到黑。
2. `2 x`，查询$x$点周围有多少黑色的邻居。

```
//当莫个图论莫名奇妙时可以写
using namespace std;
typedef long long ll;
const int N=2e5+10;
const int M=500; //sqrt(n)值左右
ll tag[M+10][M+10];
ll val[N];
int col[N],big[N],ans[N];
vector<int> g[N],bige[N];
int main(){
    int n,m,q;
    scanf("%d%d%d",&n,&m,&q);
    for(int i=1;i<=m;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for(int i=1;i<=n;i++)
        if(g[i].size()>=M) big[i]=1; //标记哪些是大点
    for(int u=1;u<=n;u++){
        for(auto v:g[u]){
            if(big[v]){
```

```
                bige[u].push_back(v); //把大度数的节点放的bige里面每次只更新大点
            }
        }
    }
    for(int i=0;i<q;i++){
        int ty,x;
        scanf("%d%d",&ty,&x);
        if(ty==1){
            col[x]^=1;
            for(auto u:bige[x]){
                //只跟新最大值点周围的点
                if(col[x]) ans[u]++;
                else ans[u]--;
            }
        }else{
            if(big[x]) printf("%d\n",ans[x]);
            //大点直接输出
            else{
                int t=0;
                for(auto u:g[x]){
                    if(col[u])t++;
                    //小点直接挨个查
                }
                printf("%d\n",t);
            }
        }
    }
    return 0;
}
```

**最大和上升子序列2**

利用权值线段树/树状数组优化一维

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long u64;
const int N=2e5+10;
int a[N];
ll tr[N*4];
ll f[N];
int n;
int lowbit(int x){
    return x&-x;
}
ll query(int x){
    ll s=0;
    for(;x;x-=lowbit(x)) s=max(s,tr[x]);
    return s;
}
void add(int x,ll s){
    for(;x<=n;x+=lowbit(x)){
        tr[x]=max(tr[x],s);
```

```
    }
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    ll ans=0;
    for(int i=1;i<=n;i++){
        f[i]=query(a[i]-1)+a[i];
        add(a[i],f[i]);
        ans=max(ans,f[i]);
    }
    printf("%lld",ans);
    return 0;
}
```

**区间lower_bound查询 扫描线**

给你$n$个数$a1, a2, \ldots, an$和$q$个询问:

- `l r x`，询问$al, al+1, \ldots, ar$中大于等于$x$的最小的数。

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
int n,q;
int seg[N*4];
int ans[N];
//vector<pair<int,int>> vx;插入是和事件一起做的，不用单开数组
vector<array<int,5> > event;
void update(int id){
    seg[id]=min(seg[id*2],seg[id*2+1]);
}
void change(int id,int l,int r,int x,int val){
    if(l==r){
        seg[id]=val;
        return ;
    }
    int mid=l+r>>1;
    if(x<=mid) change(id*2,l,mid,x,val);
    else change(id*2+1,mid+1,r,x,val);
    update(id);
}
int query(int id,int l,int r,int ql,int qr,int x){
    if(l==ql&&qr==r) return seg[id];
    int mid=l+r>>1;
    if(qr<=mid){
        return query(id*2,l,mid,ql,qr,x);
    }else if(ql>mid) return query(id*2+1,mid+1,r,ql,qr,x);
    else{
        return min(query(id*2,l,mid,ql,mid,x),query(id*2+1,mid+1,r,mid+1,qr,x));
    }
}
int main(){
    scanf("%d%d",&n,&q);
    memset(seg,0x3f,sizeof seg);
    int x;
```

```
    for(int i=1;i<=n;i++)
    {
    scanf("%d",&x);
    event.push_back({-x,0,i,0});
    }
    // 希望从大到小插入,前面取负数直接sort就行
    // 插入在查询前发生
    for(int i=1;i<=q;i++){
        int l,r,x;
        scanf("%d%d%d",&l,&r,&x);
        event.push_back({-x,i,l,r}); //以x为主
    }
    sort(event.begin(),event.end());
    for(auto evt:event)
    {
    int op=evt[1];
    if(op==0){
        change(1,1,n,evt[2],-evt[0]);
    }else{
        int t=query(1,1,n,evt[2],evt[3],-evt[0]);
        if(t==0x3f3f3f3f) t=-1;
        ans[evt[1]]=t;
    }
  }
  for(int i=1;i<=q;i++) printf("%d\n",ans[i]);
    return 0;
}
```

## 动态规划 例题

### 记忆化搜索

```
 1  2  3  4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9
```

```
// 滑雪
#include<bits/stdc++.h>
using namespace std;
const int N=310;
int g[N][N];
int n,m;
int f[N][N];
int dx[5]={1,0,-1,0};
int dy[5]={0,1,0,-1};
int dfs(int x,int y,int fa,int fb){
    if(f[x][y]!=-1) return f[x][y];
```

```
    f[x][y]=1;
    for(int i=0;i<=4;i++){
        int tx=dx[i]+x,ty=dy[i]+y;
        if(tx<1||ty<1||tx>n||ty>m||(tx==fa&&ty==fb)) continue;
        if(g[tx][ty]<g[x][y]){
            dfs(tx,ty,x,y);
            f[x][y]=max(f[x][y],f[tx][ty]+1);
        }
    }
    return f[x][y];
}
int main(){
    cin>>n>>m;
    memset(f,-1,sizeof f);
    for(int i=1;i<=n;i++)
     for(int j=1;j<=m;j++) cin>>g[i][j];
     int ans=1;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            ans=max(dfs(i,j,-1,-1),ans);
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

**方格取数**

```
#include<iostream>
#include<algorithm>
#include<cstring>
const int N=25;
const int M=25;
int g[N][N];
int f[M][N][N];  //多加一维用于同步
using namespace std;
int main(){
    int n;
    cin>>n;
    int x,y,w;
    while(cin>>x>>y>>w){
        if(x==y&&y==w&&w==0) break;
        g[x][y]=w;
    }
    for(int i1=1;i1<=n;i1++){
        for(int i2=1;i2<=n;i2++){
            for(int k=2;k<=2*n;k++){
                //此处可能越界，不过求最大值没事，所以数据范围开大了
                if(i1==i2){
                    f[k][i1][i2]=max(f[k][i1][i2],max(f[k-1][i1-1][i2-
1],max(f[k-1][i1][i2-1],max(f[k-1][i1-1][i2],f[k-1][i1][i2]))))+g[i1][k-i1]);
                }else{
```

```
                    f[k][i1][i2]=max(f[k][i1][i2],max(f[k-1][i1-1][i2-
1],max(f[k-1][i1][i2-1],max(f[k-1][i1-1][i2],f[k-1][i1][i2])))+g[i1][k-i1]+g[i2]
[k-i2]);
                }
            }
        }
    }
    cout<<f[2*n][n][n];
}
```

**玉米田 简单状压dp**

农夫约翰的土地由 M×N 个小方格组成，现在他要在土地里种植玉米。

非常遗憾，部分土地是不育的，无法种植。

而且，相邻的土地不能同时种植玉米，也就是说种植玉米的所有方格之间都不会有公共边缘。

现在给定土地的大小，请你求出共有多少种种植方法。

土地上什么都不种也算一种方法。

```
2 3
1 1 1
0 1 0
// 9
```

```
using namespace std;
const int N=13,M=1<<13,mod=1e8;
int f[N][M];
vector<int> state[N];
vector<int> state2[N];
int g[N][N];
int n,m;
bool check(int x){      //判断相邻为1的check函数
    for(int i=0;i<m;i++){   （i+1越界没事）
        if((x>>i)&1 && (x>>(i+1))&1 )return false;
    }
    return true;
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        for(int j=0;j<m;j++) cin>>g[i][j]; //存图
    }
    for(int i=1;i<=n;i++){
        for(int j=0;j<1<<m;j++){
            bool flag=true;
            for(int k=0;k<m;k++){
                if(g[i][k]==0&&(j>>k)&1) {  //检查是否符合田地
                    flag=false;
                }
            }

            if(flag) state[i].push_back(j);
```

```
        }
    }
    for(int i=1;i<=n;i++){
        for(auto t:state[i]){
            if(check(t)){
                state2[i].push_back(t);  //删掉有1相连的状态
            }
        }
    }

    f[0][0]=1;
    for(auto t:state2[1]) f[1][t]=1;  //第一行只有一种情况
    for(int i=2;i<=n;i++){
        for(auto t1:state2[i]){         //枚举该行
            for(auto t2:state2[i-1]){  //前一行
                if(!(t1&t2)){         //上下没有相邻的
                    f[i][t1]=(f[i][t1]+f[i-1][t2])%mod;
                }
            }
        }
    }
    long long ans=0; //记数总状态
    for(auto t:state2[n]){
        ans=(ans+f[n][t])%mod;
    }
    cout<<ans;
    return 0;
}
```

**最长公共上升自序列**

```
#include<iostream>
#include<algorithm>
using namespace std;
const int N=3010;
int f[N][N],a[N],b[N];  //f表示a数组前i个最长和不超过前i个的b数组的公共最长上升子序列且以b[i]结尾
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++) cin>>a[i];
    for(int i=1;i<=n;i++) cin>>b[i];

    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            f[i][j]=f[i-1][j];    //a[i]前没有相同的
            if(a[i]==b[j]){
            f[i][j]=max(f[i][j],1);  //防一手0
            for(int k=1;k<j;k++)     //a[i]与b[j]相同要枚举b[j]前面的数保证最长
                if(a[i]>b[k]) f[i][j]=max(f[i][j],f[i-1][k]+1); //这个if可以把a[i]换成b[j] 求maxn
            }
        }
    }
```

```
    int res=0;
    for(int i=1;i<=n;i++) res=max(res,f[n][i]);
    cout<<res;
    return 0;
}
```

**树上背包 系类**

```
using namespace std;  //1
const int N = 2200;
const int INF = 1 << 29;
vector<int> son[N];
int n,q,a[N],dp[N][N],sz[N]; //第i个子树，大小为j的包含点i的最大权值
void dfs(int u){
    static int tmp[N];
    sz[u]=0;
    for (auto v : son[u]) { //遍历子节点
        dfs(v);
        for (int i = 0; i <= sz[u] + sz[v]; i++) tmp[i] = -INF;
        for (int i = 0; i <= sz[u]; i++) { //合并两个区间
            for (int j = 0; j <=sz[v]; j++) { //u里面选i个和v里面选j个
                tmp[i + j] = max(tmp[i + j], dp[u][i] + dp[v][j]);
            }
        }
        for (int i = 0; i <= sz[u] + sz[v]; i++) dp[u][i] = tmp[i];
        sz[u] += sz[v];
    }
    //把u放入集合中
    sz[u] += 1;
    for (int i = sz[u]; i >= 1; i--) dp[u][i] = dp[u][i - 1] + a[u];
    dp[u][0] = 0;
}
int main(){

    scanf("%d%d",&n,&q);
    for (int i =2; i <=n; i++) {
        int x;
        scanf("%d", &x);
        son[x].push_back(i);
    }
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    dfs(1);
    while (q--){
        int v, u;
        scanf("%d%d", &v, &u);
        printf("%d\n", dp[v][u]);
    }
    return 0;
}

using namespace std;
const int N = 50100;  //2
const int M = 100;
```

```cpp
const int INF = 1 << 29;
vector<int> son[N];
int n, q, a[N], dp[N][M+10], sz[N]; //第i个子树，大小为j的包含点i的最大权值
void dfs(int u) {
    static int tmp[N];
    sz[u] = 0;
    for (auto v : son[u]) { //从子节点向上遍历
        dfs(v);
        for (int i = 0; i <= sz[u] + sz[v]&&i<=M; i++) tmp[i] = -INF;  //选的点少了，就没必要超过100个点
        for (int i = 0; i <= sz[u]&&i<=M; i++) { //合并两个区间
            for (int j = 0; j <= sz[v]&&i+j<=M; j++) { //u里面选i个和v里面选j个
                tmp[i + j] = max(tmp[i + j], dp[u][i] + dp[v][j]);
            }
        }
        for (int i = 0; i <= sz[u] + sz[v]&&i<=M; i++) dp[u][i] = tmp[i]; //这
        sz[u] += sz[v];
    }
    //把u放入集合中
    sz[u] += 1;
    for (int i = min(sz[u],M); i >= 1; i--) dp[u][i] = dp[u][i - 1] + a[u];
    dp[u][0] = 0;
}
int main() {

    scanf("%d%d", &n, &q);
    for (int i = 2; i <= n; i++) {
        int x;
        scanf("%d", &x);
        son[x].push_back(i);
    }
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    dfs(1);
    while (q--) {
        int v, u;
        scanf("%d%d", &v, &u);
        printf("%d\n", dp[v][u]);
    }
    return 0;
}
// 3
using namespace std;
const int N = 1010;
const int M = 10010;
const int INF = 1 << 29;
vector<int> son[N];
int n,m, a[N], dp[N][M],l[N],r[N],tot;  //dp含义与前面相同
int w[N],id[N];
void dfs(int u){   //这里是DFS的特殊用法是把其搜索顺序记录下来
    l[u]=++tot;
    id[tot]=u;  //每个节点都有
    for(auto t:son[u]) dfs(t);
    r[u]=tot;
}
int main() {
```

```
    scanf("%d%d",&n,&m);
    for(int i=2;i<=n;i++){
        int x;
        scanf("%d",&x);
        son[x].push_back(i);
    }
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=1;i<=n;i++) scanf("%d",&w[i]);
    dfs(1);

    for(int j=1;j<=m;j++) dp[n+1][j]=-INF;  //是正好有，所以预处里边界
    for(int i=n;i>=1;i--){  //枚举每个tot
        int u=id[i];       //当前节点
        for(int j=0;j<=m;j++){  //相当于背包体积
            dp[i][j]=dp[r[u]+1][j]; //不选这个点，那它的字节点也别选
            if(j>=w[u])   //选的话更新
              dp[i][j]=max(dp[i][j],dp[i+1][j-w[u]]+a[u]);

        }
    }
    for(int i=0;i<=m;i++){
        if(dp[1][i]>=0) printf("%d\n",dp[1][i]);
        else puts("0");
    }
    return 0;
}
```

**树上路径**

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=3000+100;
const ll mod=1e9+7;
ll pw[N];
char s[N];
ll c;
int main(){
    scanf("%s%lld",s,&c);
    int n=strlen(s);
    pw[0]=1;
    for(int i=1;i<=n;i++) pw[i]=pw[i-1]*(c+1)%mod; //求每个数位的个数+求和
    int pre=1;
    ll ans=0;
    for(int i=0;i<n;i++){
        if(s[i]=='1'){ //更新0？？？的答案
            ans=(ans+pre*pw[n-i-1]%mod)%mod; //哪一位的情况
            pre=pre*c%mod;
        }
    }
    ans=(ans+pre)%mod; //算上s自己的情况
    printf("%lld",ans);
    return 0;
```

```
}
```

### 点分治

```
https://www.luogu.com.cn/problem/P3806
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+100,M=1e7+10;
typedef long long ll;
int sz[N],maxp[N],q[N],tmp[N],dis[N];
int root=0;
int sum=0;
int e[N],ne[N],idx,h[N];
int w[N];
int n,m,cnt;
bool ans[N],judge[M],st[N];
void add(int a,int b,int c){
    e[idx]=b,ne[idx]=h[a],w[idx]=c,h[a]=idx++;
}
//树的重心是求 删除该点后，最大的连通块大小最小
// 因为这样以其为根向下遍历的层数最小，完全二叉树 log
void dfsroot(int u,int fa){
    sz[u]=1,maxp[u]=0;
    for(int i=h[u];i!=-1;i=ne[i]){
        int j=e[i];
        if(j==fa||st[j]) continue; //已经删过的点也不遍历
        dfsroot(j,u);
        sz[u]+=sz[j];
        if(sz[j]>maxp[u]) maxp[u]=sz[j];
    }
    maxp[u]=max(maxp[u],sum-sz[u]);  //求删除u点以后的最大分块大小
    if(maxp[u]<maxp[root]) root=u;
    return ;
}
void getdis(int u,int f){
    tmp[cnt++]=dis[u];     //存入u的所有节点距离
    for(int i=h[u];i!=-1;i=ne[i]){
        int j=e[i];
        if(j==f||st[j]) continue;
        dis[j]=dis[u]+w[i];  //跟新1从节点u到节点j的距离
        getdis(j,u);          //循环到子树中
    }
}
void solve(int u){
    queue<int> que;
    for(int i=h[u];i!=-1;i=ne[i]){
        int v=e[i];
        if(st[v]) continue;
        cnt=0;        //把tmp指针归0，相当于清空数组
        dis[v]=w[i];  // 当前的根的距离就是 边权
        getdis(v,u);  // 把所有到当前节点的距离都递归一遍
```

```
        for(int j=0;j<cnt;j++)
            for(int k=0;k<m;k++)
                if(q[k]>=tmp[j]) ans[k]|=judge[q[k]-tmp[j]]; //把所有的查询都对应更
新一遍
        //更新一遍已有长度的标记
        for(int j=0;j<cnt;j++){
            if(tmp[j]<M)
            {
               que.push(tmp[j]);
               judge[tmp[j]]=true;
            }

        }
    }
    // 算完这个清空根节点的所有的权值，要算新的以根为节点的东西
    while(que.size()){
        judge[que.front()]=0;
        que.pop();
    }
}
void divide(int u){
    // 从root依次往下删点
    st[u]=judge[0]=true;
    solve(u);  //算一下这个包含当前节点的所有可能的长度
    for(int i=h[u];i!=-1;i=ne[i]){
        int j=e[i];
        if(st[j]) continue;
        maxp[root=0]=sum=sz[j];
        dfsroot(j,0);       //以当前节点遍历
        dfsroot(root,0);  //找新根
        divide(root);    // 删点
    }
}
int main(){
    scanf("%d%d",&n,&m);
    memset(h,-1,sizeof h);
    for(int i=1;i<n;i++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        add(u,v,w);
        add(v,u,w);
    }
    for(int i=0;i<m;i++) scanf("%d",&q[i]);
    sum=n;
    maxp[0]=n;
    dfsroot(1,-1);
    dfsroot(root,-1);
    divide(root);
    for(int i=0;i<m;i++){
        if(ans[i]) puts("AYE");
        else puts("NAY");
    }
    return 0;
}
```

## 数位 dp

### 数数3

```
typedef long long ll;
ll f[20][2][4][10]; //后面还能填i位，前面结尾数字为j，是否存在连续，前面已有个连续的数字的个数
ll dfs(int sheng,int cunzai,int geshu,int qian){
    if(sheng==0) return cunzai;
    if(f[sheng][cunzai][geshu][qian]!=-1) return f[sheng][cunzai][geshu][qian];
    ll &ans=f[sheng][cunzai][geshu][qian]; //这里取地址不用写很长的状态
    ans++; //这里要先还原
    for(int i=0;i<10;i++){
        int g=(i>qian)?min(geshu+1,3):1;
        ans+=dfs(sheng-1,cunzai|(g==3),g,i); //填问号 123????
    }
    return ans;
}
ll dp(ll x){// 1...x
    x=x+1; //算1...x-1
    vector<int> shuzi;
    while(x){
        int t=x%10;
        shuzi.push_back(t);
        x/=10;
    }
    reverse(shuzi.begin(),shuzi.end());
    int m=shuzi.size();
    int last=0,cunzai=0,geshu=0;
    ll ans=0;
    // 含前导0的状态
    for(int i=1;i<=m-1;i++){
        for(int j=1;j<=9;j++){
            ans+=dfs(i-1,0,1,j); //应该从小到大先把递增的for一遍
        }
    }
    for(int i=0;i<m;i++){
        for(int j=(i==0)?1:0;j<shuzi[i];j++){
            int g=(j>last)?min(3,geshu+1):1;
            ans+=dfs(m-i-1,cunzai|(g==3),g,j);
        }
        geshu=(shuzi[i]>last)?min(geshu+1,3):1; //这是选到最后一维
        cunzai|=(geshu==3);
        last=shuzi[i];
    }
    return ans;
}
int main(){
    ll l,r;
    memset(f,-1,sizeof f);
    cin>>l>>r;
    cout<<dp(r)-dp(l-1);
    return 0;
}
```

####### cfti

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
vector<int> shuzi;
int st[11];
int m;
int l,k;
bool dfs(int wei,int begger,int cnt,int ans){
    if(wei==m){
        printf("%d\n",ans);
        return true;
    }else{
        //注意传的参数和遍历数字的下标别搞混了
        for(int i=(begger)?0:shuzi[wei];i<=9;i++){ //枚举数字前面有大的后面都填 0
            st[i]+=1;
            if(st[i]==1) cnt++;                    //int ncnt=cnt;
                                //if(st[i]==1) ncnt++; //多记个变量不回溯
            if(cnt<=k&&dfs(wei+1,begger|(i>shuzi[wei]),cnt,ans*10+i)) return
true; //更新当前填的位置和bigger
            st[i]-=1;
            if(st[i]==0) cnt--;
        }
        return false;
    }
}
int solve(int x,int k){
    shuzi.clear();
    for(int i=0;i<10;i++) st[i]=0;
    while(x){
        shuzi.push_back(x%10);
        x/=10;
    }
    reverse(shuzi.begin(),shuzi.end());
     m=shuzi.size();
    dfs(0,0,0,0);
}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
    scanf("%d%d",&l,&k);
    solve(l,k);
   }
    return 0;
}
```

**快速乘法**

```
#include <bits/stdc++.h>
#ifndef ONLINE_JUDGE
#else
  #define debug(...) 0
```

```
#endif
using namespace std;
const int N = (int) 1e6 + 5;
const int inf = 1e9;
string a;
int dp[2][N][2];
int n;

int dfs(int is, int i, int exist) { // 使用两次花费
  int cost = exist ? 2 : 1; //是不是确定答案了
  int& ans= dp[is][i][exist];
  if (ans!= -1) return ans;
  if (i == n) {
    if (a[i] == '1') ans=cost;
    else ans= is ? inf : 0; //最后这个点位为前的值为0就不用
    return ans;
  }
  ans= inf;
  for (int x = -1; x <= 1; ++x) { //3种状态
    int val = is * 2 + x - (a[i] == '1'); // 算平成
    if (val == 0 || val == 1) { //值不能小只能大
      int z = (x != 0); //不选为0,每次操作的花费
      ans= min(ans, cost * z + dfs(val, i + 1, exist | z ) );
      //
    }
  }
  return ans;
}
void solve() {
  cin >> a;
  n = (int) a.size();
  a = ' ' + a;
  memset(dp, -1, sizeof(dp));
  cout << dfs(0, 0, 0) << '\n';
}
signed main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  solve();
  return 0;
}
```

### chengdu

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=3000+100;
const ll mod=1e9+7;
ll pw[N];
char s[N];
ll c;
int main(){
    scanf("%s%lld",s,&c);
    int n=strlen(s);
```

```
    pw[0]=1;
    for(int i=1;i<=n;i++) pw[i]=pw[i-1]*(c+1)%mod; //求每个数位的个数+求和
    int pre=1;
    ll ans=0;
    for(int i=0;i<n;i++){
        if(s[i]=='1'){ //更新0？？？的答案
            ans=(ans+pre*pw[n-i-1]%mod)%mod; //哪一位的情况
            pre=pre*c%mod;
        }
    }
    ans=(ans+pre)%mod; //算上s自己的情况
    printf("%lld",ans);
    return 0;
}
```

**洛谷 dp杂项**

**旅游计划 topsort+dp**

小明要去一个国家旅游。这个国家有*N*个城市，编号为1至*N*，并且有*M*条道路连接着，小明准备从其中一个城市出发，并只往东走到城市i停止。

所以他就需要选择最先到达的城市，并制定一条路线以城市i为终点，使得线路上除了第一个城市，每个城市都在路线前一个城市东面，并且满足这个前提下还希望游览的城市尽量多。

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

```
const int N=1e5+10;
vector<int> g[N];
int in[N];
queue<int> a;
int ans[N];
void solve() {
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++){
        int c,b;
        scanf("%d%d",&c,&b);
        g[c].push_back(b);
        in[b]++;
    }
    for(int i=1;i<=n;i++) if(!in[i]) a.push(i),ans[i]=1;
    while(a.size()){
        int u=a.front();
        a.pop();
        for(auto t:g[u]){
        ans[t]=max(ans[u]+1,ans[t]);
         in[t]--;
         if(in[t]==0){
```

```
            a.push(t);
          }
        }
    }
    for(int i=1;i<=n;i++) printf("%d\n",ans[i]);
}
int main(){
#ifdef LOCAL
    freopen("E:/input.txt", "r", stdin);
#endif
    solve();
    return 0;
}
```

**垃圾陷阱 背包**

卡门——农夫约翰极其珍视的一条 `Holsteins` 奶牛——已经落了到“垃圾井”中。“垃圾井”是农夫们扔垃圾的地方，它的深度为 D（2≤D*≤100）英尺。

卡门想把垃圾堆起来，等到堆得高度大等于于井的深度时，她就能逃出井外了。另外，卡门可以通过吃一些垃圾来维持自己的生命。

每个垃圾都可以用来吃或堆放，并且堆放垃圾不用花费卡门的时间。

假设卡门预先知道了每个垃圾扔下的时间 $t$（1≤t*≤1000），以及每个垃圾堆放的高度 $h$（1≤$h$≤25）和吃进该垃圾能维持生命的时间 $f$（1≤$f$≤30），要求出卡门最早能逃出井外的时间，假设卡门当前体内有足够持续 1010 小时的能量，如果卡门 1010 小时内（不含 1010 小时，维持生命的时间同）没有进食，卡门就将饿死

```
20 4
5 4 9
9 3 2
12 6 10
13 1 1  //13
```

```
using namespace std;
typedef long long ll;
const int N=110;
struct Node{
    int t,h,l;
}c[N];
int d,n;
int ti[N];
int f[N]; //表示当前的垃圾高度和剩余生存时间
bool cmp(const Node &a,const Node &b)
{
    return a.t<b.t;
}
int main()
{
    cin>>d>>n;
    for(int i=1;i<=n;i++) cin>>c[i].t>>c[i].l>>c[i].h;
    sort(c+1,c+1+n,cmp);
    f[0]=10;
    for(int i=1;i<=n;i++)
```

```
      for(int j=d;j>=0;j--)
        if(f[j]>=c[i].t)
        {
            if(j+c[i].h>=d)  //能出去就直接输出
            {
                cout<<c[i].t;
                return 0;
            }                //不然更新剩余时间
            f[j+c[i].h]=max(f[j+c[i].h],f[j]);
            f[j]+=c[i].l;
        }
    cout<<f[0];
    return 0;
}
```

**有线电事网**

某收费有线电视网计划转播一场重要的足球比赛。他们的转播网和用户终端构成一棵树状结构，这棵树的根结点位于足球比赛的现场，树叶为各个用户终端，其他中转站为该树的内部节点。

从转播站到转播站以及从转播站到所有用户终端的信号传输费用都是已知的，一场转播的总费用等于传输信号的费用总和。

现在每个用户都准备了一笔费用想观看这场精彩的足球比赛，有线电视网有权决定给哪些用户提供信号而不给哪些用户提供信号。

写一个程序找出一个方案使得有线电视网在不亏本的情况下使观看转播的用户尽可能多。

```
5 3
2 2 2 5 3
2 3 2 4 3
3 4 2   //2
```

```
#include<iostream>
#include<algorithm>
#include<bits/stdc++.h>
using namespace std;
const int N=3000+10;
typedef long long ll;
typedef pair<int,int> PII;
vector<PII> g[N];
int f[N][N]; //第i个节点选j个用户
int a[N];
int n,m;
int dfs(int u){
    if(u>n-m){
        f[u][1]=a[u];
        return 1;
    }
    int sz=0;
    for(auto node:g[u]){
        int val=node.first,w=node.second;
        int t=dfs(val);
        sz=t+sz;
        for(int j=sz;j>0;j--){
```

```
            for(int i=1;i<=t;i++){
                if(j>=i) f[u][j]=max(f[u][j],f[u][j-i]+f[val][i]-w);
            }
        }
    }//这里做一手树上背包
    return sz;
}
void solve(){
    memset(f,~0x3f,sizeof(f));//初始化一个极大负值，因为f可能为负
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n-m;i++){
        int k;
        scanf("%d",&k);
        int x,w;
        for(int j=1;j<=k;j++){
            scanf("%d%d",&x,&w);
            g[i].push_back({x,w});
        }
    }
    for(int i=n-m+1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=0;i<=n;i++) f[i][0]=0;
    dfs(1);
    int ans=0;
    for(int i=m;i>=0;i--){
        if(f[1][i]>=0){
            printf("%d",i);
            return ;
        }
    }
    return ;
}
int main(){
#ifdef LOCAL
    freopen("E:/input.txt","r",stdio);
#endif
    solve();
    return 0;
}
```

**三角形牧场**

和所有人一样，奶牛喜欢变化。它们正在设想新造型的牧场。奶牛建筑师 Hei 想建造围有漂亮白色栅栏的三角形牧场。她拥有 *n* 块木板，每块的长度 /**i 都是整数，她想用所有的木板围成一个三角形使得牧场面积最大。

请帮助 Hei 小姐构造这样的牧场，并计算出这个最大牧场的面积。

```
5
1
1
3
3
4   //692
```

```
using namespace std;
typedef long long ll;
const int N=900;
bool f[N][N];
int a[50];
double ans;
bool check(int x,int y,int z){
    if(x+y>z&&y+z>x&&x+z>y)return true;
    return false;
}
double ji_m(double x,double y,double z)
{
    double p=(x+y+z)/2;
    return sqrt(p*(p-x)*(p-y)*(p-z));
}
int main(){
    int n,sum=0;
    cin>>n;
    for(int i=1;i<=n;i++) cin>>a[i],sum+=a[i];
    f[0][0]=true;
    for(int k=1;k<=n;k++)
    for(int i=sum/2;i>=0;i--)
        for(int j=sum/2;j>=0;j--){
            if(i-a[k]>=0&&f[i-a[k]][j]) f[i][j]=true;
            if(j-a[k]>=0&&f[i][j-a[k]]) f[i][j]=true;
        }
    ans=-1;
    for(int i=sum/2;i>0;i--)
        for(int j=sum/2;j>0;j--){
            if(f[i][j]){
                if(check(i,j,sum-i-j)){
                    ans=max(ans,ji_m(i,j,sum-i-j));
                }
            }
        }
    if(ans!=-1) cout<<(long long)(ans*100);
    else cout<<ans;

    return 0;
}
```

**没有上司的舞会**

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int w[N];
vector<int> son[N];
int t=0;
bool st[N];
int dfs(int u,bool st){
    int ans=0;
    if(st==0){
        for(auto t:son[u]){
        int tm=dfs(t,1);
```

```
        ans+=tm;
    }
    return ans;
  }else{
    for(auto t:son[u]){
        int tm=dfs(t,0);
        ans+=tm;
    }
    return ans+max(w[u],0);
  }
}
int main(){
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",&w[i]);
    for(int i=1;i<=n-1;i++){
        int l,k;
        scanf("%d%d",&l,&k);
        son[k].push_back(l);
        st[l]=true;
    }
    for(int i=1;i<=n;i++){
        if(!st[i]){
            t+=max(dfs(i,0),dfs(i,1));
        }
    }
    printf("%d",t);
    return 0;
}
```

**序列取数**

给定一个长为n的整数序列(n<=1000)，由A和B轮流取数（A先取）。每个人可从序列的左端或右端取若干个数（至少一个），但不能两端都取。所有数都被取走后，两人分别统计所取数的和作为各自的得分。假设A和B都足够聪明，都使自己得分尽量高，求A的最终得分。

```
2
1 -1
2 1 2  // -1 3
```

```
#include<cstdio>
const int N=1002;
int T,n,i,j,a[N],s[N],l[N],r[N];
int max(int x,int y){
    return x>y?x:y;
}
int main(){
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        for(i=1;i<=n;i++){
            scanf("%d",a+i);
            s[i]=s[i-1]+a[i];
            l[i]=r[i]=a[i];
```

```
        }
        for(j=1;j<n;j++)
            for(i=1;i+j<=n;i++){
                r[i]=a[i+j]+max(r[i],s[i+j-1]-s[i-1]-max(l[i],r[i]));
                l[i]=a[i]+max(l[i+1],s[i+j]-s[i]-max(l[i+1],r[i+1]));
                //注意要先转移r[i]，再转移l[i],因为r[i]的转移需要l[i]前一维的值
            }
        printf("%d\n",max(l[1],r[1]));
    }
}
```

**无聊的数列 线段树**

维护一个数列 *a**i*，支持两种操作：

- `1 l r K D`：给出一个长度等于 *r*−*l*+1 的等差数列，首项为 *K*，公差为 *D*，并将它对应加到 [*l*,*r] 范围中的每一个数上。
- `2 p`：询问序列的第 *p* 个数的值 a_p。

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

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod=998244353;
const int N=1e5+10;
int n,m;
struct Node{
    ll val;
    ll tagd,tagk;
    ll sz;
}seg[N*4]; //下传的时候改K就行
// a1+K a2+K+D  a2+K+2D
// a1+K a2+K1 A3+K1+D
ll a[N];
void settag(int id,int k,int d){
    seg[id].tagk+=k;
    seg[id].tagd+=d;
}
void pushdown(int id){
    if(seg[id].tagd!=0||seg[id].tagk!=0){
        seg[id*2].tagk+=seg[id].tagk;
        seg[id*2].tagd+=seg[id].tagd;
        seg[id*2+1].tagk+=(seg[id].tagd*seg[id*2].sz+seg[id].tagk);
        seg[id*2+1].tagd+=seg[id].tagd;
        seg[id].tagd=0;
        seg[id].tagk=0;
    }
}
void build(int id,int l,int r){
    seg[id].sz=(r-l)+1;
    if(l==r){
```

```
        seg[id].val=a[l];
        return ;
    }
    int mid=l+r>>1;
    build(id*2,l,mid);
    build(id*2+1,mid+1,r);
}
void modify(int id,int l,int r,int ql,int qr,ll k,ll d){
    if(l==ql&&qr==r){
        settag(id,k,d);
        return ;
    }
    pushdown(id);
    int mid=l+r>>1;
    if(qr<=mid) modify(id*2,l,mid,ql,qr,k,d);
    else if(ql>=mid+1) modify(id*2+1,mid+1,r,ql,qr,k,d);
    else{
        modify(id*2,l,mid,ql,mid,k,d);
        modify(id*2+1,mid+1,r,mid+1,qr,k+d*(mid-ql+1),d);
    }
}
ll query(int id,int l,int r,int x){
    if(l==r) return seg[id].val+seg[id].tagk;
    else{
        pushdown(id);
        int mid=l+r>>1;
        if(x<=mid) return query(id*2,l,mid,x);
        else return query(id*2+1,mid+1,r,x);
    }
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%lld",&a[i]);
    build(1,1,n);
    while(m--){
        int op,l,r;
        ll k,d;
        scanf("%d%d",&op,&l);
        if(op==1){
            scanf("%d%lld%lld",&r,&k,&d);
            modify(1,1,n,l,r,k,d);
        }else printf("%lld\n",query(1,1,n,l));
    }
    return 0;
}
```

**绝世好题**

给定一个长度为 $n$ 的数列 $a_i$*，求 a**i 的子序列 b**i 的最长长度 $k$，满足 $b_i \& b_{i-1}=0$，其中$2\le i\le k$， &表示位运算取与。

```
#include<iostream>
#include<algorithm>
#include<cmath>
#include<cstring>
```

```cpp
using namespace std;
typedef long long ll;
const int N=1e5+10;
int f[N];
int a[N];
//看似是最长上升子序列实际是分组的线性dp
int main(){
    int n,ans=0;
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        int k=1;
        for(int j=0;j<=30;j++)
            if((1<<j)&a[i]) k=max(f[j]+1,k);
        for(int j=0;j<=30;j++)
            if((1<<j)&a[i]) f[j]=max(f[j],k);
        ans=max(ans,k);
    }
    printf("%d",ans);
    return 0;
}
```