Dengpc

目录

基础

快速排序、归并排序、二分、高精度、前缀和与差分、双指针、位运算、离散化、区间合并、RMQ

快速排序

```
void quick_sort(int q[], int l, int r){
    if (l >= r) return;

    int i = l - 1, j = r + 1, x = q[l + r >> 1];
    while (i < j){
        do i ++ ; while (q[i] < x);
        do j -- ; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }
    quick_sort(q, l, j), quick_sort(q, j + 1, r);
}
```

归并排序

```
void merge_sort(int q[], int l, int r){
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(q, l, mid);
    merge_sort(q, mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (q[i] <= q[j]) tmp[k ++ ] = q[i ++ ];
        else tmp[k ++ ] = q[j ++ ];
    while (i <= mid) tmp[k ++ ] = q[i ++ ];
    while (j <= r) tmp[k ++ ] = q[j ++ ];
    for (i = l, j = 0; i <= r; i ++, j ++ ) q[i] = tmp[j];
}
```

整数二分

```
bool check(int x) {/* ... */} // 检查x是否满足某种性质

// 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用:
int bsearch_1(int l, int r){
    while (l < r){
        int mid = l + r >> 1;
        if (check(mid)) r = mid;    // check()判断mid是否满足性质
        else l = mid + 1;
```

```
    }
    return l;
}
// 区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用:
int bsearch_2(int l, int r){
    while (l < r){
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}
```

浮点二分

```
bool check(double x) {/* ... */} // 检查x是否满足某种性质

double bsearch_3(double l, double r){
    const double eps = 1e-6;   // eps 表示精度，取决于题目对精度的要求
    while (r - l > eps){
        double mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}
```

高精度加法

```
// C = A + B, A >= 0, B >= 0
vector<int> add(vector<int> &A, vector<int> &B){
    if (A.size() < B.size()) return add(B, A);

    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i ++ ){
        t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }

    if (t) C.push_back(t);
    return C;
}
```

高精度加法

```cpp
// C = A - B, 满足A >= B, A >= 0, B >= 0
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];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

高精度乘低精度法

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

高精度除低精度法

```cpp
// A / b = C ... r, A >= 0, b > 0
vector<int> div(vector<int> &A, int b, int &r){
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; i -- ){
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

前缀和

```
S[x2, y2] - S[x1 - 1, y2] - S[x2, y1 - 1] + S[x1 - 1, y1 - 1]
```

差分

```
S[x1, y1] += c, S[x2 + 1, y1] -= c, S[x1, y2 + 1] -= c, S[x2 + 1, y2 + 1] += c
```

位运算

```
求 n 的第 k 位数字: n >> k & 1
返回 n 的最后一位 1：lowbit(n) = n & -n
```

双指针

```
for (int i = 0, j = 0; i < n; i ++ ){
    while (j < i && check(i, j)) j ++ ;
    // 具体问题的逻辑
}
常见问题分类：
    (1) 对于一个序列，用两个指针维护一段区间
    (2) 对于两个序列，维护某种次序，比如归并排序中合并两个有序序列的操作
```

离散化

```
vector<int> alls; // 存储所有待离散化的值
sort(alls.begin(), alls.end()); // 将所有值排序
alls.erase(unique(alls.begin(), alls.end()), alls.end());   // 去掉重复元素

// 二分求出 x 对应的离散化的值
int find(int x){ // 找到第一个大于等于 x 的位置
    int l = 0, r = alls.size() - 1;
    while (l < r){
        int mid = l + r >> 1;
        if (alls[mid] >= x) r = mid;
        else l = mid + 1;
    }
    return r + 1; // 映射到1, 2, ...n
}
```

区间合并

```
// 将所有存在交集的区间合并
void merge(vector<PII> &segs){
    vector<PII> res;
    sort(segs.begin(), segs.end());
    int st = -2e9, ed = -2e9;
    for (auto seg : segs)
        if (ed < seg.first){
```

```
            if (st != -2e9) res.push_back({st, ed});
            st = seg.first, ed = seg.second;
        }
        else ed = max(ed, seg.second);

    if (st != -2e9) res.push_back({st, ed});

    segs = res;
}
```

RMQ

```
int w[N];
int f[N][M];
void init(){
    for (int j = 0; j < M; j ++ )
        for (int i = 1; i + (1 << j) - 1 <= n; i ++ )
            if (!j) f[i][j] = w[i];
            else f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);
}

int query(int l, int r){
    int len = r - l + 1;
    int k = log(len) / log(2);

    return max(f[l][k], f[r - (1 << k) + 1][k]);
}
```

数据结构

链表

```
int h[N], e[M], w[M], ne[M], idx;
void add(int a, int b, int c){
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
}
```

双链表

```
// e[]表示节点的值，l[]表示节点的左指针，r[]表示节点的右指针，idx 表示当前用到了哪个节点
int e[N], l[N], r[N], idx;
// 初始化
void init(){
    //0 是左端点，1 是右端点
    r[0] = 1, l[1] = 0;
    idx = 2;
}
// 在节点 a 的右边插入一个数 x
void insert(int a, int x){
    e[idx] = x;
    l[idx] = a, r[idx] = r[a];
    l[r[a]] = idx, r[a] = idx ++ ;
}
// 删除节点 a
void remove(int a){
    l[r[a]] = l[a];
    r[l[a]] = r[a];
}
```

栈

```
int stk[N], tt = 0;
stk[ ++ tt] = x;tt -- ;
```

队列

```
// hh 表示队头，tt 表示队尾
int q[N], hh = 0, tt = -1;
// 向队尾插入一个数
q[ ++ tt] = x;
// 从队头弹出一个数
hh ++ ;
```

循环队列

```
// hh 表示队头，tt 表示队尾的后一个位置
int q[N], hh = 0, tt = 0;
// 向队尾插入一个数
q[tt ++ ] = x;
if (tt == N) tt = 0;
// 从队头弹出一个数
hh ++ ;
if (hh == N) hh = 0;
```

单调栈

```
常见模型：找出每个数左边离它最近的比它大/小的数
int tt = 0;
for (int i = 1; i <= n; i ++ ){
    while (tt && check(stk[tt], i)) tt -- ;
    stk[ ++ tt] = i;
}
```

单调队列

```
常见模型：找出滑动窗口中的最大值/最小值
int hh = 0, tt = -1;
for (int i = 0; i < n; i ++ ){
    while (hh <= tt && check_out(q[hh])) hh ++ ;  // 判断队头是否滑出窗口
    while (hh <= tt && check(q[tt], i)) tt -- ;
    q[ ++ tt] = i;
}
```

KMP

```
// s[]是长文本，p[]是模式串，n 是 s 的长度，m 是 p 的长度
求模式串的 Next 数组：
for (int i = 2, j = 0; i <= m; i ++ ){
    while (j && p[i] != p[j + 1]) j = ne[j];
    if (p[i] == p[j + 1]) j ++ ;
    ne[i] = j;
}

// 匹配
for (int i = 1, j = 0; i <= n; i ++ ){
    while (j && s[i] != p[j + 1]) j = ne[j];
    if (s[i] == p[j + 1]) j ++ ;
    if (j == m){
        j = ne[j];
        // 匹配成功后的逻辑
```

```
    }
}
```

Tire

```
int son[N][26], cnt[N], idx;
// 0 号点既是根节点，又是空节点
// son[][]存储树中每个节点的子节点
// cnt[]存储以每个节点结尾的单词数量

// 插入一个字符串
void insert(char *str){
    int p = 0;
    for (int i = 0; str[i]; i ++ ){
        int u = str[i] - 'a';
        if (!son[p][u]) son[p][u] = ++ idx;
        p = son[p][u];
    }
    cnt[p] ++ ;
}
// 查询字符串出现的次数
int query(char *str){
    int p = 0;
    for (int i = 0; str[i]; i ++ ){
        int u = str[i] - 'a';
        if (!son[p][u]) return 0;
        p = son[p][u];
    }
    return cnt[p];
}

```

并查集

```
(1)朴素并查集：

    int p[N]; //存储每个点的祖宗节点

    // 返回x的祖宗节点
    int find(int x)
    {
        if (p[x] != x) p[x] = find(p[x]);
        return p[x];
    }

    // 初始化，假定节点编号是1~n
```

```
    for (int i = 1; i <= n; i ++ ) p[i] = i;

    // 合并a和b所在的两个集合：
    p[find(a)] = find(b);


(2)维护size的并查集：

    int p[N], size[N];
    //p[]存储每个点的祖宗节点，size[]只有祖宗节点的有意义，表示祖宗节点所在集合中的点的数量

    // 返回x的祖宗节点
    int find(int x)
    {
        if (p[x] != x) p[x] = find(p[x]);
        return p[x];
    }

    // 初始化，假定节点编号是1~n
    for (int i = 1; i <= n; i ++ )
    {
        p[i] = i;
        size[i] = 1;
    }

    // 合并a和b所在的两个集合：
    size[find(b)] += size[find(a)];
    p[find(a)] = find(b);


(3)维护到祖宗节点距离的并查集：

    int p[N], d[N];
    //p[]存储每个点的祖宗节点，d[x]存储x到p[x]的距离

    // 返回x的祖宗节点
    int find(int x)
    {
        if (p[x] != x)
        {
            int u = find(p[x]);
            d[x] += d[p[x]];
            p[x] = u;
        }
```

```
        return p[x];
    }

    // 初始化，假定节点编号是 1~n
    for (int i = 1; i <= n; i ++ )
    {
        p[i] = i;
        d[i] = 0;
    }

    // 合并 a 和 b 所在的两个集合：
    p[find(a)] = find(b);
    d[find(a)] = distance; // 根据具体问题，初始化 find(a) 的偏移量
```

堆

```
// h[N]存储堆中的值，h[1]是堆顶，x 的左儿子是 2x，右儿子是 2x + 1
// ph[k]存储第 k 个插入的点在堆中的位置
// hp[k]存储堆中下标是 k 的点是第几个插入的
int h[N], ph[N], hp[N], size;

// 交换两个点，及其映射关系
void heap_swap(int a, int b)
{
    swap(ph[hp[a]],ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(h[a], h[b]);
}

void down(int u)
{
    int t = u;
    if (u * 2 <= size && h[u * 2] < h[t]) t = u * 2;
    if (u * 2 + 1 <= size && h[u * 2 + 1] < h[t]) t = u * 2 + 1;
    if (u != t)
    {
        heap_swap(u, t);
        down(t);
    }
}

void up(int u)
{
    while (u / 2 && h[u] < h[u / 2])
    {
```

```
        heap_swap(u, u / 2);
        u >>= 1;
    }
}

// O(n)建堆
for (int i = n / 2; i; i -- ) down(i);
make_heap
push_heap
pop_heap
sort_heap
```

字符串哈希

```
核心思想：将字符串看成 P 进制数，P 的经验值是 131 或 13331，取这两个值的冲突概率低
小技巧：取模的数用 2^64，这样直接用 unsigned long long 存储，溢出的结果就是取模的结果

typedef unsigned long long ULL;
ULL h[N], p[N]; // h[k]存储字符串前 k 个字母的哈希值, p[k]存储 P^k mod 2^64

// 初始化
p[0] = 1;
for (int i = 1; i <= n; i ++ ){
    h[i] = h[i - 1] * P + str[i];
    p[i] = p[i - 1] * P;
}

// 计算子串 str[l ~ r] 的哈希值
ULL get(int l, int r){
    return h[r] - h[l - 1] * p[r - l + 1];
}
```

树状数组

（1）单点修改，区间查询

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define lowbit(x) (x&(-x))
const int N=5e5+10;
int a[N],s[N];
int n,m;
void add(int x,int k){
    for(int i=x;i<=n;i+=lowbit(i)){
        s[i]+=k;
```

```
    }
}
int searchs(int l,int r){
    int ans=0;
    for(int i=r;i;i-=lowbit(i)){
        ans+=s[i];
    }
    for(int i=l-1;i;i-=lowbit(i)){
        ans-=s[i];
    }
    return ans;
}
signed main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        add(i,a[i]);
    }
    while(m--){
        int op;
        cin>>op;
        if(op==1){
            int x,k;
            cin>>x>>k;
            add(x,k);
        }
        else{
            int l,r;
            cin>>l>>r;
            cout<<searchs(l,r)<<endl;
        }
    }
}
```

（2）区间修改，单点查询

```
//存储差分数组，在区间+k 就是把 l 位置+k,r+1 位置-k，然后单点查询就是查询 1 到 x 的和
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define lowbit(x) (x&(-x))
const int N=5e5+10;
int a[N],s[N];
int n,m;
void add(int x,int k){
```

```
    for(int i=x;i<=n;i+=lowbit(i)){
        s[i]+=k;
    }
}
int searchs(int l,int r){
    int ans=0;
    for(int i=r;i;i-=lowbit(i)){
        ans+=s[i];
    }
    for(int i=l-1;i;i-=lowbit(i)){
        ans-=s[i];
    }
    return ans;
}
signed main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        add(i,a[i]-a[i-1]);
    }
    while(m--){
        int op;
        cin>>op;
        if(op==1){
            int x,y,k;
            cin>>x>>y>>k;
            add(x,k);
            add(y+1,-k);
        }
        else{
            int x;
            cin>>x;
            cout<<searchs(1,x)<<endl;
        }
    }
}
```

（3）逆序对个数

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define lowbit(x) (x&(-x))
const int N=5e5+10;
int r[N],tr[N];
```

```
int n,m,ans;
struct T{
    int v,id;
}a[N];
bool cmp(T a1,T a2){
    if(a1.v==a2.v)return a1.id<a2.id;
    return a1.v<a2.v;
}
void add(int x,int k){
    for(int i=x;i<=n;i+=lowbit(i)){
        tr[i]+=k;
    }
}
int searchs(int l,int r){
    int ans=0;
    for(int i=r;i;i-=lowbit(i)){
        ans+=tr[i];
    }
    for(int i=l-1;i;i-=lowbit(i)){
        ans-=tr[i];
    }
    return ans;
}
signed main(){
   cin>>n;
   for(int i=1;i<=n;i++){
    cin>>a[i].v;
    a[i].id=i;
   }
   sort(a+1,a+n+1,cmp);
   for(int i=1;i<=n;i++){
    r[a[i].id]=i;
   }
   for(int i=1;i<=n;i++){
       int y=r[i];
       ans+=searchs(1,n)-searchs(1,y);
       add(y,1);
   }
   cout<<ans<<endl;
}
```

（4）区间修改、区间查询

```
//对于区间修改我们照样用差分存储改为单点修改，区间查询的话我们公式推导后可以发现需要另一个树状数组来存储 i*d 的差分来求前缀和
```

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define lowbit(x) (x&(-x))
const int N=5e5+10;
int a[N],tr[N],tr2[N];
int n,m;
void add(int tr[],int x,int k){
    for(int i=x;i<=n;i+=lowbit(i)){
        tr[i]+=k;
    }
}
int searchs(int tr[],int l,int r){
    int ans=0;
    for(int i=r;i;i-=lowbit(i)){
        ans+=tr[i];
    }
    for(int i=l-1;i;i-=lowbit(i)){
        ans-=tr[i];
    }
    return ans;
}
int sum(int x){
    return searchs(tr,1,x)*(x+1)-searchs(tr2,1,x);
}
signed main(){
   cin>>n>>m;
   for(int i=1;i<=n;i++){
        cin>>a[i];
        int d=a[i]-a[i-1];
        add(tr,i,d);
        add(tr2,i,d*i);
   }
   while(m--){
     char op;
     cin>>op;
     if(op=='Q'){
        int l,r;
        cin>>l>>r;
        cout<<sum(r)-sum(l-1)<<endl;
     }
     else{
        int l,r,d;
        cin>>l>>r>>d;
```

```
            add(tr,l,d);add(tr,r+1,-d);
            add(tr2,l,l*d);add(tr2,r+1,(r+1)*(-d));
        }
    }
}
```

线段树

（1）单点修改，区间查询

```
#include<bits/stdc++.h>
using namespace std;
const int N=5e5+10;
typedef long long int LL;
struct tree{
    LL left,right,sum,num;
}tree[N*4];
LL a[N],ans;
void build(int x,int l,int r){
    tree[x].left=l,tree[x].right=r;
    tree[x].num=0;
    if(l==r){
        tree[x].sum=a[l];
        return ;
    }
    int mid=(l+r)/2;
    build(x*2,l,mid);
    build(x*2+1,mid+1,r);
    tree[x].sum=tree[x*2].sum+tree[x*2+1].sum;
    return ;
}
void add(int i,int x,int k){
    if(tree[i].left==x&&tree[i].right==x){
        tree[i].sum+=k;
        return ;
    }
    if(tree[i*2].right>=x)add(i*2,x,k);
    if(tree[i*2+1].left<=x)add(i*2+1,x,k);
    tree[i].sum=tree[i*2].sum+tree[i*2+1].sum;
}
void searchs(int i,int l,int r){
    if(tree[i].left>=l&&tree[i].right<=r){
        ans+=tree[i].sum;
```

```
34.         return ;
35.     }
36.     if(tree[i*2].right>=l)searchs(i*2,l,r);
37.     if(tree[i*2+1].left<=r)searchs(i*2+1,l,r);
38. }
39. int main(){
40.     LL n,m;
41.     cin>>n>>m;
42.     for(int i=1;i<=n;i++)cin>>a[i];
43.     build(1,1,n);
44.     while(m--){
45.         int op,x,k,l,r;
46.         cin>>op;
47.         if(op==1){
48.             cin>>x>>k;
49.             add(1,x,k);
50.         }
51.         else {
52.            cin>>l>>r;
53.            ans=0;
54.            searchs(1,l,r);
55.            cout<<ans<<endl;
56.         }
57.     }
58. }
```

（2）区间修改，单点查询

```
1.  #include<bits/stdc++.h>
2.
3.  using namespace std;
4.  const int N=5e5+10;
5.  typedef long long int LL;
6.  struct tree{
7.      LL left,right,sum,num;
8.  }tree[N*4];
9.  LL a[N],ans;
10. void build(int x,int l,int r){
11.     tree[x].left=l,tree[x].right=r;
12.     tree[x].num=0;
13.     if(l==r){
14.         tree[x].sum=a[l];
15.         return ;
16.     }
17.     int mid=(l+r)/2;
```

```
18.      build(x*2,l,mid);
19.      build(x*2+1,mid+1,r);
20.      tree[x].sum=tree[x*2].sum+tree[x*2+1].sum;
21.      //cout<<tree[x].sum<<endl;
22.      return ;
23.  }
24.  void add(int x,int l,int r,int k){
25.      if(tree[x].left>=l&&tree[x].right<=r){
26.          tree[x].num+=k;
27.          return ;
28.      }
29.      if(tree[x*2].right>=l)add(x*2,l,r,k);
30.      if(tree[x*2+1].left<=r)add(x*2+1,l,r,k);
31.  }
32.  void searchs(int x,int p){
33.      ans+=tree[x].num;
34.      if(tree[x].left==tree[x].right)return ;
35.      if(p<=tree[x*2].right)searchs(x*2,p);
36.      if(p>=tree[x*2+1].left)searchs(x*2+1,p);
37.  }
38.  int main(){
39.      LL n,m;
40.      cin>>n>>m;
41.      for(int i=1;i<=n;i++)cin>>a[i];
42.      build(1,1,n);
43.      while(m--){
44.          int op,l,r,k;
45.          cin>>op;
46.          if(op==1){
47.              cin>>l>>r>>k;
48.              add(1,l,r,k);
49.          }
50.          else {
51.              cin>>l;
52.              ans=0;
53.              searchs(1,l);
54.              cout<<ans+a[l]<<endl;
55.          }
56.      }
57.  }
```

（3）区间修改，区间查询

加法

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=5e5+10;
typedef long long int LL;
struct tree{
    LL left,right,sum,lz;
}tree[N*4];
LL a[N],ans;
void build(int x,int l,int r){
    tree[x].left=l,tree[x].right=r;
    if(l==r){
        tree[x].sum=a[l];
        return ;
    }
    int mid=(l+r)/2;
    build(x*2,l,mid);
    build(x*2+1,mid+1,r);
    tree[x].sum=tree[x*2].sum+tree[x*2+1].sum;
    return ;
}
void push_down(int i){
    if(tree[i].lz!=0){
        int mid=(tree[i].left+tree[i].right)/2;
        tree[i*2].sum+=(mid-tree[i*2].left+1)*tree[i].lz;
        tree[i*2+1].sum+=(tree[i*2+1].right-mid)*tree[i].lz;
        tree[i*2].lz+=tree[i].lz;
        tree[i*2+1].lz+=tree[i].lz;
        tree[i].lz=0;
    }
}
void add(int i,int l,int r,int k){
    if(tree[i].left>=l&&tree[i].right<=r){
        tree[i].sum+=(tree[i].right-tree[i].left+1)*k;
        tree[i].lz+=k;
        return;
    }
    push_down(i);
    int mid=(l+r)/2;
    if(tree[i*2].right>=l)add(i*2,l,r,k);
    if(tree[i*2+1].left<=r)add(i*2+1,l,r,k);
    tree[i].sum=tree[i*2].sum+tree[i*2+1].sum;
}

void searchs(int i,int l,int r){
```

```
    if(tree[i].left>=l&&tree[i].right<=r){
        ans+=tree[i].sum;
        return ;
    }
    push_down(i);
    if(tree[i*2].right>=l)searchs(i*2,l,r);
    if(tree[i*2+1].left<=r)searchs(i*2+1,l,r);
}
int main(){
    LL n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    build(1,1,n);
    while(m--){
        int op;
        cin>>op;
        int l,r,k;
        if(op==1){
            cin>>l>>r>>k;
            add(1,l,r,k);
        }
        else{
            cin>>l>>r;
            ans=0;
            searchs(1,l,r);
            cout<<ans<<endl;
        }
    }
}
```

加法+乘法

```
//改了好久
#include<bits/stdc++.h>
using namespace std;
const int N=5e5+10;
#define int long long
struct tree{
    int left,right,sum,mulz,addz;
}tree[N*4];
int a[N],ans,p;
void build(int x,int l,int r){
    tree[x].left=l,tree[x].right=r;
    tree[x].mulz=1;
    if(l==r){
```

```
14.             tree[x].sum=a[l]%p;
15.             return ;
16.         }
17.         int mid=(l+r)/2;
18.         build(x*2,l,mid);
19.         build(x*2+1,mid+1,r);
20.         tree[x].sum=(tree[x*2].sum+tree[x*2+1].sum)%p;
21.         return ;
22.     }
23.     void push_down(int i){
24.         //子线段的加法懒标记=原先的乘以父节点的乘法懒标记+父节点的加法懒标记
25.         // sum 需要在 push_down 中计算，若在 ans 那里计算的话，（倘若第一次就进入会被计算重复），
    push_down 后再计算
26.         tree[i*2].sum=(tree[i*2].sum*tree[i].mulz+tree[i].addz*(tree[i*2].right-tree[i*2].
    left+1)%p)%p;//两次模 p
27.         tree[i*2+1].sum=(tree[i*2+1].sum*tree[i].mulz+tree[i].addz*(tree[i*2+1].right-tree
    [i*2+1].left+1)%p)%p;
28.         tree[i*2].mulz=tree[i*2].mulz*tree[i].mulz%p;
29.         tree[i*2+1].mulz=tree[i*2+1].mulz*tree[i].mulz%p;
30.         tree[i*2].addz=(tree[i*2].addz*tree[i].mulz+tree[i].addz)%p;
31.         tree[i*2+1].addz=(tree[i*2+1].addz*tree[i].mulz+tree[i].addz)%p;
32.         tree[i].mulz=1;
33.         tree[i].addz=0;
34.         return ;
35.     }
36.     void add(int i,int l,int r,int k){
37.         if(tree[i].left>=l&&tree[i].right<=r){
38.             tree[i].sum=(tree[i].sum+(tree[i].right-tree[i].left+1)*k)%p;
39.             tree[i].addz=(tree[i].addz+k)%p;
40.             return;
41.         }
42.         push_down(i);
43.         int mid=(l+r)/2;
44.         if(tree[i*2].right>=l)add(i*2,l,r,k);
45.         if(tree[i*2+1].left<=r)add(i*2+1,l,r,k);
46.         tree[i].sum=(tree[i*2].sum+tree[i*2+1].sum)%p;
47.     }
48.     void mul(int i,int l,int r,int k){
49.         if(tree[i].left>=l&&tree[i].right<=r){
50.             tree[i].sum=(tree[i].sum*k)%p;
51.             tree[i].addz=(tree[i].addz*k)%p;
52.             tree[i].mulz=tree[i].mulz*k%p;
53.             return ;
54.         }
```

```
    push_down(i);
    if(tree[i*2].right>=l)mul(i*2,l,r,k);
    if(tree[i*2+1].left<=r)mul(i*2+1,l,r,k);
    tree[i].sum=(tree[i*2].sum+tree[i*2+1].sum)%p;
}
void searchs(int i,int l,int r){
    if(tree[i].left>=l&&tree[i].right<=r){
        ans=(ans+tree[i].sum)%p;
        return ;
    }
    push_down(i);
    if(tree[i*2].right>=l)searchs(i*2,l,r);
    if(tree[i*2+1].left<=r)searchs(i*2+1,l,r);
}
signed main(){
    int n,m;
    cin>>n>>m>>p;
    for(int i=1;i<=n;i++)cin>>a[i];
    build(1,1,n);
    while(m--){
        int op;
        cin>>op;
        int l,r,k;
        if(op==1){
            cin>>l>>r>>k;
            mul(1,l,r,k);
        }
        else if(op==2){
            cin>>l>>r>>k;
            add(1,l,r,k);
        }
        else{
            cin>>l>>r;
            ans=0;
            searchs(1,l,r);
            cout<<ans%p<<endl;
        }
    }
}
```

区间最大连续字段和

```
tmax 存储最大连续字段和，只有 tmax 转移不过来

rmax 最大后缀和，lmax 最大前缀和
```

```
用左子节点的最大后缀和和右子节点的最大前缀和以及他们的最大连续字段和向父亲 tmax 转移
父亲的 lmax 和 rmax 需要用 sum 以及子节点 lmax 和 rmax 转移
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=5e5+10;
int a[N],ans;
struct Tree{
    int l,r,lmax, rmax, tmax,sum;

}tree[N*4];
void build(int i,int l,int r){
    tree[i].l=l;tree[i].r=r;
    if(l==r){
        tree[i].sum=a[l];
        tree[i].lmax=tree[i].rmax=tree[i].tmax=a[l];
        return ;
    }
    int mid=(l+r)/2;
    build(i*2,l,mid);
    build(i*2+1,mid+1,r);
    tree[i].sum=tree[i*2].sum+tree[i*2+1].sum;
    tree[i].lmax=max(tree[i*2].lmax,tree[i*2].sum+tree[i*2+1].lmax);
    tree[i].rmax=max(tree[i*2+1].rmax,tree[i*2].rmax+tree[i*2+1].sum);
    tree[i].tmax=max(max(tree[i*2].tmax,tree[i*2+1].tmax),tree[i*2].rmax+tree[i*2+1].l
max);
    return ;
}
void xg(int i,int x,int k){
    if(tree[i].l==tree[i].r&&tree[i].l==x){
        tree[i].sum=k;
        tree[i].lmax=tree[i].rmax=tree[i].tmax=k;
        return ;
    }
    if(x<=tree[i*2].r)xg(i*2,x,k);
    if(x>=tree[i*2+1].l)xg(i*2+1,x,k);
    tree[i].sum=tree[i*2].sum+tree[i*2+1].sum;
    tree[i].lmax=max(tree[i*2].lmax,tree[i*2].sum+tree[i*2+1].lmax);
    tree[i].rmax=max(tree[i*2+1].rmax,tree[i*2].rmax+tree[i*2+1].sum);
    tree[i].tmax=max(max(tree[i*2].tmax,tree[i*2+1].tmax),tree[i*2].rmax+tree[i*2+1].l
max);
}
Tree searchs(int i,int l,int r){
    if(tree[i].l>=l&&tree[i].r<=r){
```

```
        return tree[i];
    }
    int mid=(tree[i].l+tree[i].r)/2;
    if(mid>=r)return searchs(i*2,l,r);   //在当前左半区间
    if(mid<l)return searchs(i*2+1,l,r);  //在当前右半区间
    auto left=searchs(i*2,l,r);           //两边都有
    auto right=searchs(i*2+1,l,r);
    Tree v;
    v.sum=left.sum+right.sum;
    v.lmax=max(left.lmax,left.sum+right.lmax);
    v.rmax=max(right.rmax,left.rmax+right.sum);
    v.tmax=max(max(left.tmax,right.tmax),left.rmax+right.lmax);//合并答案
    return v;
}
signed main(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    build(1,1,n);
    while(m--){
        int k,x,y;
        cin>>k>>x>>y;
        if(k==1){
            if(x>y)swap(x,y);
            cout<<searchs(1,x,y).tmax<<endl;
        }
        else{
            xg(1,x,y);
        }
    }
}
```

区间最大公约数

这是一个区间修改区间查询问题，若用 push_down 来写 gcd 没法更新（我不会），当你查询到在向下传递懒标记的话，你还是没法更新，除非 push_down 到最底层（那还不如多次单点修改）。

然后想到用差分数组来写（感觉和树状数组想法有点像？是不可以用线段树套树状数组来写），我们每个建树时存储差分（这样的话就把区间加 d 改成单点修改）

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=5e5+10;
```

```
int a[N],ans;
struct Tree{
    int l,r,sum,v;
}tree[N*4];
void build(int i,int l,int r){
    tree[i].l=l;tree[i].r=r;
    if(l==r){
        tree[i].v=a[r]-a[r-1];
        tree[i].sum=a[r]-a[r-1];
        return ;
    }
    int mid=(l+r)/2;
    build(i*2,l,mid);
    build(i*2+1,mid+1,r);
    tree[i].v=__gcd(tree[i*2].v,tree[i*2+1].v);
    tree[i].sum=tree[i*2].sum+tree[i*2+1].sum;
    return ;
}
void add(int i,int x,int d){
    if(tree[i].l==x&&tree[i].r==x){
        tree[i].v=tree[i].sum+d;
        tree[i].sum+=d;
        return ;
    }
    if(x<=tree[i*2].r)add(i*2,x,d);
    if(x>=tree[i*2+1].l)add(i*2+1,x,d);
    tree[i].sum=tree[i*2].sum+tree[i*2+1].sum;
    tree[i].v=__gcd(tree[i*2].v,tree[i*2+1].v);
    return ;
}
Tree searchs(int i,int l,int r){
    if(tree[i].l>=l&&tree[i].r<=r){
        return tree[i];
    }
    int mid=(tree[i].l+tree[i].r)/2;
    if(r<=mid)return searchs(i*2,l,r);
    if(l>mid)return searchs(i*2+1,l,r);
    auto left=searchs(i*2,l,r);
    auto right=searchs(i*2+1,l,r);
    Tree u;
    u.sum=left.sum+right.sum;
    u.v=__gcd(left.v,right.v);      //更新答案用的，不向上更新
    return u;
}
```

```
signed main(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    build(1,1,n);
    while(m--){
        char op;
        int l,r,d;
        cin>>op>>l>>r;
        if(op=='C'){
            cin>>d;
            add(1,l,d);
            if(r+1<=n)add(1,r+1,-d);
        }
        else{
            auto ll=searchs(1,1,l);
            Tree rr={0,0,0,0};
            if(l+1<=r)rr=searchs(1,l+1,r);
            cout<<abs(__gcd(ll.sum,rr.v))<<endl;
        }

    }
}
```

可持久化数据结构

（1）最大异或和

```
#include <iostream>

using namespace std;

const int N = 600010, M = N * 25;

int n, m;  // 序列初始长度、操作个数
int s[N];  // 原序列的异或前缀和
int tr[M][2];  // trie 中的节点
int max_id[M];  // 以当前节点为根的子树所代表的数据中最晚插入的那个数据插入的时间t(即第几个插入的)
int root[N], idx;  // 记录每个版本，从 1 开始

// i: 前缀和下标，表示插入 Si
// k: 当前处理到Si 的第几位，从 23 为开始处理到第 0 位
```

```
// p: 上一个版本的根节点
// q: 当前版本的根节点，是 p 的复制，差别在于新加了一条链
void insert(int i, int k, int p, int q) {

    if (k < 0) {  // 说明我们处理完了最后一位(第 0 位)，此时 q 就是叶节点的编号
        max_id[q] = i;
        return;
    }
    int v = s[i] >> k & 1;
    // 所有需要修改的节点，不要去修改它，而是拷贝一个新的，然后修改这个新点。
    if (p) tr[q][v ^ 1] = tr[p][v ^ 1];  // 如果上一个版本存在，直接复制过来
    tr[q][v] = ++idx;  // 当前这个二进制位需要开辟一个新的节点
    insert(i, k - 1, tr[p][v], tr[q][v]);
    max_id[q] = max(max_id[tr[q][0]], max_id[tr[q][1]]);  // 左右儿子取大
}

// 从 root 开始查询，需要被异或的数据是 C，区间左端点是 L
int query(int root, int C, int L) {

    int p = root;
    for (int i = 23; i >= 0; i--) {
        int v = C >> i & 1;
        if (max_id[tr[p][v ^ 1]] >= L) p = tr[p][v ^ 1];
        else p = tr[p][v];  // 找不到更好的了，只能将就一下
    }
    return C ^ s[max_id[p]];  // 这个叶子节点只有它自己，因此 max_id 就是 Si 的下标 i
}

int main() {

    scanf("%d%d", &n, &m);

    // 这里 max_id[0] 中的 0 表示空节点，空节点不包含任何点，
    // 所以它的 max_id 需要设置成比所有的 id 都小的数，所以设置成任何负数均可。
    max_id[0] = -1;

    root[0] = ++idx;  // 第 0 个版本在 trie 中的根节点编号为 1
    insert(0, 23, 0, root[0]);  // 在第 0 个版本中插入 S0，不存在上一个版本

    for (int i = 1; i <= n; i++) {
        int x;
        scanf("%d", &x);
        s[i] = s[i - 1] ^ x;
        root[i] = ++idx;  // 第 i 个版本
```

```
        insert(i, 23, root[i - 1], root[i]);
    }

    char op[2];
    int l, r, x;
    for (int i = 1; i <= m; i++) {
        scanf("%s", op);
        if (*op == 'A') {
            scanf("%d", &x);
            n++;
            s[n] = s[n - 1] ^ x;
            root[n] = ++idx;
            insert(n, 23, root[n - 1], root[n]);
        } else {
            scanf("%d%d%d", &l, &r, &x);
            // p 在[L, R]之间，p-1 在[L-1, R-1]之间
            printf("%d\n", query(root[r - 1], s[n] ^ x, l - 1));
        }
    }

    return 0;
}
```

（2）主席树

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=1e5+10;
int n,m,len,a[N],d[N];
int rout[N],idx;
struct tree{
    int l,r,v;
}t[N*20];
int build(int l,int r){
    int p=++idx;
    int mid=(l+r)>>1;
    if(l<r){
        t[p].l=build(l,mid);
        t[p].r=build(mid+1,r);
    }
    t[p].v=0;
    return p;
}
int inserts(int pre,int l,int r,int v){
```

```
    int p=++idx,mid=(l+r)>>1;
    t[p].l=t[pre].l;
    t[p].r=t[pre].r;
    t[p].v=t[pre].v+1;
    if(l<r){
        if(v<=mid)t[p].l=inserts(t[pre].l,l,mid,v);
        else t[p].r=inserts(t[pre].r,mid+1,r,v);
    }
    return p;
}
int searchs(int x,int y,int l,int r,int k){
    if(l==r)return l;
    int sum=t[t[y].l].v-t[t[x].l].v;
    int mid=(l+r)>>1;
    if(k<=sum)return searchs(t[x].l,t[y].l,l,mid,k);
    else return searchs(t[x].r,t[y].r,mid+1,r,k-sum);
}
signed main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        d[i]=a[i];
    }
    sort(d+1,d+1+n);
    int len=unique(d+1,d+n+1)-(d+1);
    for(int i=1;i<=n;i++){
        a[i]=lower_bound(d+1,d+len+1,a[i])-d;
    }
    rout[0]=build(1,len);
    for(int i=1;i<=n;i++){
        rout[i]=inserts(rout[i-1],1,len,a[i]);
    }
    while(m--){
        int l,r,k;
        cin>>l>>r>>k;
        int ans=searchs(rout[l-1],rout[r],1,len,k);
        cout<<d[ans]<<endl;
    }
}
```

平衡树

插入数值 x，删除数值 x（若有多个相同的数，则只删除一个），查询数值 x 的排名（若有多个相同的数，应输出最小排名），查询排名为 x 的的数值，求数值 x 的前驱

求数值 x 的后继

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10,INF=1e8;
int n,root,idx;
struct Node{
    int l,r;
    int key,val;
    int cnt,size;//当前节点 key 重复个数和子孙结点个数
}tr[N];
//更新父节点信息
void pushup(int p){
    tr[p].size=tr[tr[p].l].size+tr[tr[p].r].size+tr[p].cnt;
}
//创造叶子结点
int get_node(int key){
    tr[++idx].key=key;
    tr[idx].val=rand();
    tr[idx].cnt=tr[idx].size=1;
    return idx;
}
//初始化平衡树
void build(){
    get_node(-INF);get_node(INF);
    root=1,tr[1].r=2;//INF 在-INF 右边
    pushup(root);
}
//右旋
void zig(int &p){
    int q=tr[p].l;
    tr[p].l=tr[q].r;
    tr[q].r=p;
    p=q;
    pushup(tr[p].r);
    pushup(p);
}
//左旋
void zag(int &p){
    int q=tr[p].r;
    tr[p].r=tr[q].l;
    tr[q].l=p;
```

```
    p=q;
    pushup(tr[p].l);
    pushup(p);
}

void inserts(int &p,int key){
    if(!p)p=get_node(key);
    else if(tr[p].key==key)tr[p].cnt++;
    else if(tr[p].key>key){
        inserts(tr[p].l,key);
        if(tr[tr[p].l].val>tr[p].val)zig(p);
    }
    else {
        inserts(tr[p].r,key);
        if(tr[tr[p].r].val>tr[p].val)zag(p);
    }
    pushup(p);
}

void remove(int &p,int key){
    if(!p)return ;
    if(tr[p].key==key){
        if(tr[p].cnt>1)tr[p].cnt--;
        else if(tr[p].l||tr[p].r){
            if(!tr[p].r||tr[tr[p].l].val>tr[tr[p].r].val){
                //只存在左儿子或者左 val 大于右 val
                zig(p);
                remove(tr[p].r,key);
            }
            else {
                //存在右儿子且左 val 小于右 val
                zag(p);
                remove(tr[p].l,key);
            }
        }
        else p=0;
    }
    else if(tr[p].key>key)remove(tr[p].l,key);
    else remove(tr[p].r,key);
    pushup(p);
}

int get_rank_bykey(int p,int key){
    if(!p)return 0;
```

```
85.     if(tr[p].key==key)return tr[tr[p].l].size+1;//左子树 size 一
86.     if(tr[p].key>key)return get_rank_bykey(tr[p].l,key);//大了，去左子树找
87.     return tr[tr[p].l].size+tr[p].cnt+get_rank_bykey(tr[p].r,key);
88. }
89.
90. int get_key_byrank(int p,int rank){
91.     if(!p)return INF;
92.     if(tr[tr[p].l].size>=rank)return get_key_byrank(tr[p].l,rank);
93.     if(tr[tr[p].l].size+tr[p].cnt>=rank)return tr[p].key;
94.     return get_key_byrank(tr[p].r,rank-tr[tr[p].l].size-tr[p].cnt);
95. }
96. //找到严格小于 key 的最大数
97. int get_pre(int p,int key){
98.     if(!p)return -INF;
99.     if(tr[p].key>=key)return get_pre(tr[p].l,key);
100.    return max(tr[p].key,get_pre(tr[p].r,key));
101. }
102. //找到严格大于 key 的最小数
103. int get_next(int p,int key){
104.    if(!p)return INF;
105.    if(tr[p].key<=key)return get_next(tr[p].r,key);
106.    return min(tr[p].key,get_next(tr[p].l,key));
107. }
108.
109. int main(){
110.    build();
111.    int n;cin>>n;
112.    while(n--){
113.        int op,x;
114.        cin>>op>>x;
115.        if(op==1)inserts(root,x);
116.        if(op==2)remove(root,x);
117.        if(op==3)cout<<get_rank_bykey(root,x)-1<<endl;
118.        if(op==4)cout<<get_key_byrank(root,x+1)<<endl;
119.        if(op==5)cout<<get_pre(root,x)<<endl;
120.        if(op==6)cout<<get_next(root,x)<<endl;
121.    }
122.    return 0;
123. }
```

Math

质数

【试除法判定质数】

质数和合数是针对所有大于 1 的 “自然数” 来定义的

所有小于等于 1 的整数既不是质数也不是合数.

```
bool is_prime(int x){
    if(x<2)return false;
    for(int i=2;i<n/i;i++){
        if(n%i==0)return false;
    }
    return true;
}
//  n/i 防止越界，相对于i*i<sqrt(n);
```

【分解质因数】

$$N = p_1^{a1} P_2^{a2} \cdots p_k^a k$$

```
void divide(int x){
    for(int i=2;i<=x/i;i++){
        if(x%i==0){
            int s=0;
            while(x%i==0)x/=i,s++;
            cout<<i<<' '<<s<<endl;
        }
    }
    if(x>1)cout<<x<<' '<<1<endl;
}
```

【筛质数】

调和级数:当 n 趋近于正无穷的时候,1/2+1/3+1/4+1/5+…+1/n=lnn+c.(c 是欧阳常数,约等于 0.577 左右.).

1. 朴素筛

```
void get_primes(int n){
    for(int i=2;i<=n;i++){
        if(st[i])continue;
```

```
            primes[cnt++]=i;
            for(int j=i+i;j<=n;j+=i){
                st[j]=true;
            }
        }
    }
```

2. 线性筛

```
void get_primes(int n){
    for(int i=2;i<=n;i++){
        if(!st[i])primes[cnt++]=i;
        for(int j=0;primes[j]<=n/i;j++){
            st[primes[j]*i]=true;
            if(i%primes[j]==0)break;
        }
    }
}
```

约数

【试除法求约数】

```
vector<int>v;
void solve(int x){
    v.clear();
    for(int i=1;i<=x/i;i++){
        if(x%i==0){
            v.push_back(i);
            if(i!=x/i)  v.push_back(x/i);
        }
    }
    sort(v.begin(),v.end());
    for(int i=0;i<v.size();i++)cout<<v[i]<<' ';
    cout<<endl;
}
```

【约数个数】

N 的正约数个数为：$\prod_{i=1}^{m}(c_i+1)$

```
//自己验算一下上式子正确性
```

```
void solve(int x){
    unordered_map<int,int> prime;
    for(int i=2;i<=x/i;i++){
        while(x%i==0){
            x/=i;
            prime[i]++;
        }
    }
    if(x>1)prime[x]++;
    LL ans=1;
    for(unordered_map<int,int> ::iterator i=prime.begin();i!=prime.end();i++){
        ans=ans*((*i).second+1)%mod;
    }
}
```

【约数之和】

N 的所有正约数和为： $\prod_{i=1}^{m}(\sum_{j=0}^{c_i}(p_i)^j)$

```
void solve(int x){
    map<int,int> prime;
    for(int i=2;i<=x/i;i++){
          while(x%i==0){
              x/=i;
              prime[i]++;
          }
    }
    if(x>1)prime[x]++;
    long long ans=1;
    for(map<int,int>::iterator i=prime.begin();i!=prime.end();i++){
        long long a=(*i).first,b=(*i).second;
        long long t=1;
        while(b--)t=(t*a+1)%mod;
        ans=ans*t%mod;
    }
    cout<<ans<<endl;
}
```

【欧几里得算法】

```
gcd(a,b) = gcd(b,a mod b)
int gcd(int a,int b){
    return b?gcd(b,a%b):a;
```

```
}
```

欧拉函数

【欧拉函数】

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

$$\phi(N) = N * \prod_{\text{质数 } p|N} (1 - \frac{1}{p})$$

互质是指最大公约数是 1

```
//容斥原理证明欧拉函数
//1 到 n 互质的个数即为（总个数-每个约数的倍数的个数，这些肯定不互质，一些会多，所以再加回来）
//奇减偶加
int phi(int x){
    int res = x;
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0){
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);
    return res;
}
```

【筛法求欧拉函数】

```
LL solve(){
    phi[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i])prime[cnt++]=i,phi[i]=i-1;
        for(int j=0;prime[j]<=n/i;j++){
            st[prime[j]*i]=true;
            if(i%prime[j]==0){
                phi[prime[j]*i]=phi[i]*prime[j];
                break;
            }
            phi[prime[j]*i]=phi[i]*(prime[j]-1);
        }
    }
}
```

欧拉定理：如果 a 与 n 互质，则有 a 的 phi[n]次方模 n 等于 1

当 n 为质数时(a 为任意自然数)，phi[n]=n-1;则 a 的 n-1 次方模 n 等于 1（费马定理）

$$a^p \equiv a \pmod{p}$$

快速幂

【快速幂】

```
1.  //快速求出a的k次方mod P 的结果
2.  //预处理a的2的1次方到2的logk次方的结果
3.  //把k拆成若干个2的次幂之和
4.  LL qmi(int a,int k,int p){
5.      LL res=1;
6.      while(k){
7.          if(k&1){            //看 k 的个位，是1的话，让res*a的2的几次幂
8.              res=(LL)res*a%p;
9.          }
10.         k>>=1;              //接着走
11.         a=(LL)a*a%p;        //a 方
12.     }
13.     return res%p;
14. }
```

【快速幂求逆元】

三大余数定理：

加法：(a+b)%c=a%c+b%c

减法：(a-b)%c=a%c-b%c

乘法：(ab)%c=(a%c)(b%c)

乘法逆元的定义

a 存在模 p 的乘法逆元的充要条件是 gcd(a，p) = 1

若整数 b，m 互质，并且对于任意的整数 a，如果满足 b|a，则存在一个整数 x，使得 a/b≡a×x(mod m)，则称 x 为 b 的模 m 乘法逆元，记为 b 的-1 次方(mod m)。 b 存在乘法逆元的充要条件是 b 与模数 m 互质。当模数 m 为质数时，b 的（m-2）次方 即为 b 的乘法逆元。

```
1.  //求其a的p-2次方即为其乘法逆元
2.  if(a%p==0)puts("impossible");
3.  else cout<<qmi(a,p-2,p)<<endl;
```

扩展欧几里得算法

裴蜀定理： 对任何[整数 a、b 和它们的最大公约数 d，关于未知数 x 和 y 的线性丢番图方程（称为裴蜀等式）：

ax + by = m 有解当且仅当 m 是 d 的倍数，d 其实就是最小可以写成 ax+by 的正整数

【扩展欧几里得算法】

```
int exgcd(int a,int b,int &x,int &y){
    if(!b){
        x=1,y=0;
        return a;
    }
    int d=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return d;
}
```

【线性同余方程】

ai×xi≡bi(modmi)

高斯消元

【高斯消元解线性方程组】

```
int gauss(){
    int r,c;
    for(c=0,r=0;c<n;c++){                    //枚举每一列
        int t=r;
        for(int i=r;i<n;i++){                //从 r 行开始（上面的每一行第一个数都已成为一）
            if(fabs(a[i][c])>fabs(a[t][c]))
                t=i;                         //找到 c 这一列最大的数所对应的行，为了后续便又把它交换你上去
        }

        if(fabs(a[t][c])<eps)continue;       //是 0 的话就继续

        for(int i=c;i<=n;i++)swap(a[t][i],a[r][i]);    //将此行交换上去
        for(int i=n;i>=c;i--)a[r][i]/=a[r][c];         //将这行第一个数变为 1

```

```
        for(int i=r+1;i<n;i++){                 //将从 r 到 n 行对应的此列都变为 0
            if(fabs(a[i][c])>eps){
                for(int j=n;j>=c;j--){
                    a[i][j]-=a[r][j]*a[i][c];   //a[i][c]倍
                }
            }
        }
        r++;
    }

    if(r<n){
        for(int i=r;i<n;i++){
            if(fabs(a[i][n])>eps)
            return 2;//无解       一个数=0
        }
        return 1;                               //无穷解 0=0，此方程可以被别的方程表示
    }

    for(int i=n-1;i>=0;i--)                     //求解
        for(int j=i+1;j<n;j++)
            a[i][n]-=a[j][n]*a[i][j];

    return 0;
}
```

组合数学

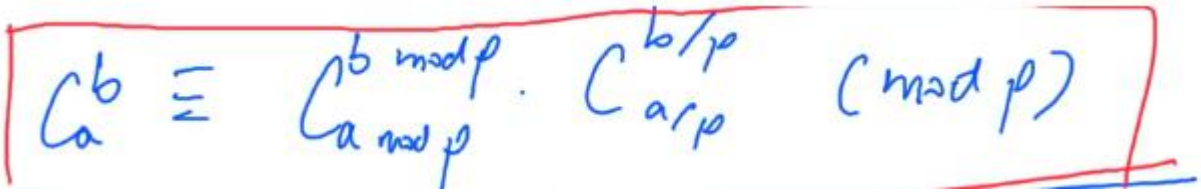
一、直接算

```
void init(){
    for (int i = 0; i < N; i ++ )
        for (int j = 0; j <= i; j ++ )
            if (!j) c[i][j] = 1;
            else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
}
//公式,dp
```

二、公式+逆元

```
fact[0]=1,infact[0]=1;
for(int i=1;i<=N;i++){
    fact[i]=(LL)fact[i-1]*i%mod;
    infact[i]=(LL)infact[i-1]*qmi(i,mod-2,mod)%mod;
}
cout<<fact[a]*infact[b]%mod*infact[a-b]%mod<<endl;
```

三、Lucas 定理



```
int C(int a,int b,int c){
    if(b>a)return 0;
    int res=1;
    for(int i=1,j=a;i<=b;i++,j--){
        res=(LL)res*j%p;
        res=(LL)res*qmi(i,p-2,p)%p;
    }
    return res;
}
int lucas(LL a,LL b,int p){
    if(a<p&&b<p)return C(a,b,p);
    return (LL)C(a%p,b%p,p)*lucas(a/p,b/p,p)%p;
}
```

四、高精度算出结果

```
int primes[N], cnt;
int sum[N];
bool st[N];
void get_primes(int n){
    for (int i = 2; i <= n; i ++ ){
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] <= n / i; j ++ ){
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
int get(int n, int p){
    int res = 0;
    while (n){
        res += n / p;
        n /= p;
    }
    return res;
}
vector<int> mul(vector<int> a, int b){
```

```
22.     vector<int> c;
23.     int t = 0;
24.     for (int i = 0; i < a.size(); i ++ ){
25.         t += a[i] * b;
26.         c.push_back(t % 10);
27.         t /= 10;
28.     }
29.     while (t){
30.         c.push_back(t % 10);
31.         t /= 10;
32.     }
33.     return c;
34. }
35. int main(){
36.     int a, b;
37.     cin >> a >> b;
38.     get_primes(a);
39.     for (int i = 0; i < cnt; i ++ ){
40.         int p = primes[i];
41.         sum[i] = get(a, p) - get(a - b, p) - get(b, p);
42.     }
43.     vector<int> res;
44.     res.push_back(1);
45.     for (int i = 0; i < cnt; i ++ )
46.         for (int j = 0; j < sum[i]; j ++ )
47.             res = mul(res, primes[i]);
48.     for (int i = res.size() - 1; i >= 0; i -- ) printf("%d", res[i]);
49.     puts("");
50.     return 0;
51. }
```

五、Catalan 数

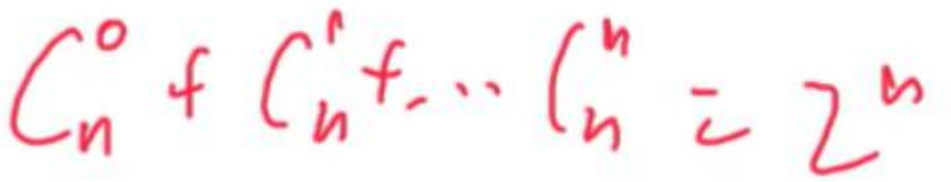
$$C_{2n}^{n} - C_{2n}^{n-1} = \frac{C_{2n}^{n}}{n+1}$$

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

画图，0 往右走，1 往上走，所有到达 n, n 点的合法（即 0 的个数比 1 多方案数

就是总的减去终点经过上一条线对称的点的方案数

容斥原理



容斥原理

```
for(int i=0;i<m;i++)cin>>p[i];
    for(int i=1;i<1<<m;i++){     //2 的 m 次方，1 表示选此数
        int cnt=0,t=1;           //cnt 统计 1 个数，奇加偶减
        for(int j=0;j<m;j++){    //t 统计选的这些数相乘
            if(i>>j&1){
                if((LL)t*p[j]>n){
                    t=-1;
                    break;
                }
                t*=p[j];
                cnt++;
            }
        }
        if(t!=-1){
            if(cnt%2){
                res+=n/t;
            }
            else res-=n/t;
        }
    }
    cout<<res<<endl;
```

博弈论

Nim 游戏

给定 n 堆石子，两位玩家轮流操作，每次操作可以从任意一堆石子中拿走任意数量的石子（可以拿完，但不能不拿），最后无法进行操作的人视为失败。
n 堆石子亦或为 1 时，先手必胜，否则必败

```
while(n--){
        int x;
        cin>>x;
        res^=x;
    }
```

```
        if(res)puts("Yes");
        else puts("No");
```

搜索

BFS

FLood Fill、最短路模型、多源 bfs、最小步数模型

双端队列广搜

算法分析

双端队列

解决问题：

双端队列主要解决图中边的权值只有 `0` 或者 `1` 的最短路问题

操作：

每次从队头取出元素，并进行拓展其他元素时

- 1、若拓展某一元素的边权是 `0` ，则将该元素插入到队头
- 2、若拓展某一元素的边权是 `1` ，则将该元素插入到队尾

```
#include<bits/stdc++.h>
#define x first
#define y second
using namespace std;
typedef pair<int, int> PII;

const int N = 510, M = N * N;

int n, m;
char g[N][N];
int dist[N][N];
bool st[N][N];

int bfs(){
```

```
15.     memset(dist, 0x3f, sizeof dist);
16.     memset(st, 0, sizeof st);
17.     dist[0][0] = 0;
18.     deque<PII> q;
19.     q.push_back({0, 0});
20.
21.     char cs[] = "\\/\\/";
22.     int dx[4] = {-1, -1, 1, 1}, dy[4] = {-1, 1, 1, -1};
23.     int ix[4] = {-1, -1, 0, 0}, iy[4] = {-1, 0, 0, -1};
24.
25.     while (q.size()){
26.         PII t = q.front();
27.         q.pop_front();
28.
29.         if (st[t.x][t.y]) continue;
30.         st[t.x][t.y] = true;
31.
32.         for (int i = 0; i < 4; i ++ ){
33.             int a = t.x + dx[i], b = t.y + dy[i];
34.             if (a < 0 || a > n || b < 0 || b > m) continue;
35.
36.             int ca = t.x + ix[i], cb = t.y + iy[i];
37.             int d = dist[t.x][t.y] + (g[ca][cb] != cs[i]);
38.
39.             if (d < dist[a][b]){
40.                 dist[a][b] = d;
41.
42.                 if (g[ca][cb] != cs[i]) q.push_back({a, b});
43.                 else q.push_front({a, b});
44.             }
45.         }
46.     }
47.
48.     return dist[n][m];
49. }
50.
51. int main(){
52.     int T;
53.     scanf("%d", &T);
54.     while (T -- ){
55.         scanf("%d%d", &n, &m);
56.         for (int i = 0; i < n; i ++ ) scanf("%s", g[i]);
57.
58.         int t = bfs();
```

```
        if (t == 0x3f3f3f3f) puts("NO SOLUTION");
        else printf("%d\n", t);
    }
    return 0;
}
```

双向广搜

已知有两个字串 A, B 及一组字串变换的规则（至多 6 个规则）：

$A_1 \to B_1$

$A_2 \to B_2$

...

规则的含义为：在 A 中的子串 A_1 可以变换为 B_1、A_2 可以变换为 B_2 ...。

```
#include<bits/stdc++.h>
using namespace std;
const int N=6;
string a[N],b[N];
int n;
string st,ed;

int extend(queue<string>& q, unordered_map<string, int>&mpa, unordered_map<string, int>& mpb, string a[N], string b[N]){

    int d=mpa[q.front()];
    while(q.size()&&mpa[q.front()]==d){
        auto t=q.front();
        q.pop();
        for(int i=0;i<n;i++){
            for(int j=0;j<t.size();j++){
                if (t.substr(j, a[i].size()) == a[i]){
                    string r = t.substr(0, j) + b[i] + t.substr(j + a[i].size());
                    if (mpb.count(r)) return mpa[t] + mpb[r] + 1;
                    if (mpa.count(r)) continue;
                    mpa[r] = mpa[t] + 1;
                    q.push(r);
                }
            }
        }
```

```
        }
        return 11;
    }
    int bfs(){
        if(st==ed)return 0;
        queue<string> qa,qb;
        unordered_map<string,int> mpa,mpb;
        qa.push(st);mpa[st]=0;
        qb.push(ed);mpb[ed]=0;

        int step=0;
        while(qa.size()&&qb.size()){
            int t;
            if(qa.size()<qb.size())t=extend(qa,mpa,mpb,a,b);
            else t=extend(qb,mpb,mpa,b,a);
            if(t<=10)return t;
            if(++step==10)return -1;
        }
        return -1;

    }
    int main(){
        cin>>st>>ed;
        while(cin>>a[n]>>b[n])n++;
        int x=bfs();
        if(x==-1)cout<<"NO ANSWER!"<<endl;
        else cout<<x<<endl;
    }
```

DFS

连通性模型，搜索顺序，
剪枝与优化：
1. 优化搜索顺序
2. 排除等效冗余
3. 可行性剪枝
4. 最优化剪枝
5. 记忆化搜索

翰翰和达达饲养了 N 只小猫，这天，小猫们要去爬山。

经历了千辛万苦，小猫们终于爬上了山顶，但是疲倦的它们再也不想徒步走下山了（呜咕>_<）。

翰翰和达达只好花钱让它们坐索道下山。

索道上的缆车最大承重量为 W，而 N 只小猫的重量分别是 C_1、C_2……C_N。

当然，每辆缆车上的小猫的重量之和不能超过 W。

每租用一辆缆车，翰翰和达达就要付 1 美元，所以他们想知道，最少需要付多少美元才能把这 N 只小猫都运送下山？

```
#include<bits/stdc++.h>
using namespace std;
int n,w;
const int N=20;
int g[N],a[N];
int ans=N;
void dfs(int x,int cnt){
    if(cnt>=ans)return ;
    if(x==n){
        ans=min(ans,cnt);
        return ;
    }
    for(int i=0;i<cnt;i++){
        if(g[i]+a[x]<=w){
            g[i]+=a[x];
            dfs(x+1,cnt);
            g[i]-=a[x];
        }
    }
    g[cnt]=a[x];
    dfs(x+1,cnt+1);
    g[cnt]=0;
}
int main(){
    cin>>n>>w;
    for(int i=0;i<n;i++){
        cin>>a[i];
    }
    sort(a,a+n);
    reverse(a,a+n);
    dfs(0,0);
    cout<<ans<<endl;
}
```

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。

```
#include<bits/stdc++.h>
using namespace std;

const int N=70;

int w[N],sum,length,n;
bool st[N];

bool dfs(int u,int part,int start){ //第u组,part第u组的已有长度,start表示第u组的枚举位置;
    if(u*length==sum) return true;//如果总长度到达了,返回true
    if(part==length) return dfs(u+1,0,0);//true要一直持续不断的返回,false同理

    for(int i=start;i<=n;i++){
        if(st[i]) continue;
        if(w[i]+part>length) continue;
        st[i]=true;//标记已经使用
        if(dfs(u,w[i]+part,i+1)) return true;//因为前i个棍子都在第u组枚举了,所以直接从第i+1根棍子开始枚举
        st[i]=false;//返回上层分支时要恢复现场(枚举该组不选择第i根棍子的方案)

        if(!part||w[i]+part==length) return false;//如果第一根失败了或者最后一根失败了,就一定失败

        int j=i;//如果i失败了,那么长度跟i一样的棍子也一定失败
        while(j<=n&&w[j]==w[i]) j++;
        i=j-1;
    }

    return false;//枚举完了还没有成功,就返回失败
}

int main(){
    while(cin>>n,n){
        //初始化
        memset(st,0,sizeof st);
        sum=0,length=1;
```

```
35.
36.         for(int i=1;i<=n;i++){
37.             scanf("%d",&w[i]);
38.             sum+=w[i];//总和
39.         }
40.
41.         //倒着排序,以减少分支
42.         sort(w+1,w+n+1);
43.         reverse(w+1,w+n+1);
44.
45.         while(1){  //枚举 length 的长度
46.             if(sum%length==0&&dfs(0,0,0)){
47.                 printf("%d\n",length);
48.                 break;
49.             }
50.             length++;
51.         }
52.     }
53. }
```

迭代加深搜索

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

1. $X[1]=1$
2. $X[m]=n$
3. $X[1]<X[2]<\ldots<X[m-1]<X[m]$
4. 对于每个 k（$2\le k\le m$）都存在两个整数 i 和 j（$1\le i,j\le k-1$，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。

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.  int n;
4.  const int N=120;
5.  int p[N];
6.  int dfs(int u,int maxx){
7.      if(u>maxx)return 0;
8.      if(p[u-1]==n)return 1;
9.      int st[N]={0};
10.     for(int i=u-1;i>=0;i--){
11.         for(int j=i;j>=0;j--){
12.             int s=p[i]+p[j];
13.             if(st[s]||s<p[u-1]||s>n)continue;
14.             st[s]=1;
15.             p[u]=s;
```

```
            if(dfs(u+1,maxx))return 1;
        }
    }
    return 0;
}
int main(){
    p[0]=1;
    while(cin>>n&&n){
        int k=1;
        while(!dfs(1,k))k++;
        for(int i=0;i<k;i++){
            cout<<p[i]<<' ';
        }
        cout<<endl;
    }
}
```

双向 DFS

达达帮翰翰给女生送礼物，翰翰一共准备了 N 个礼物，其中第 i 个礼物的重量是 $G[i]$。

达达的力气很大，他一次可以搬动重量之和不超过 W 的任意多个物品。

达达希望一次搬掉尽量重的一些物品，请你告诉达达在他的力气范围内一次性能搬动的最大重量是多少。

输入格式

第一行两个整数，分别代表 W 和 N。

以后 N 行，每行一个正整数表示 $G[i]$。

输出格式

仅一个整数，表示达达在他的力气范围内一次性能搬动的最大重量。

数据范围

$1 \le N \le 46$,
$1 \le W, G[i] \le 2^{31} - 1$

首先这道题目肯定是要搜索的，因为如果说用 DP 做的话，那么这个 W 值太大了，但是如果说只是普通搜索的话，那么 O(2^N)的复杂度足以超时，而且这道题目重点就是，我们已经知道了初态而且还知道了终态，既然如此的话，我们可以选择双向搜索.

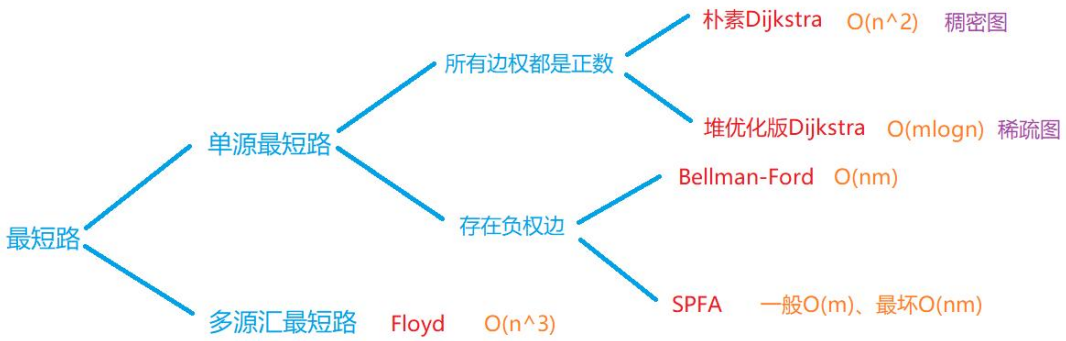
根据双向搜索的性质，我们大致可以确定当前搜索的范围，首先从前一半个物品中，挑选任意多个物品，然后将这些物品的权值总和加入到数组 S 中，然后我们就会发现在这个 S 数组中有很多很多的重复的数值，既然如此，我们不妨将他们统统都删掉，至于如何删掉，相信 STL 中的 unique 可以满足你的需求. 然后我们现在前半部分搜索完后，开始搜索后半部分，至于后半部分搜索，其实和前半部分一模一样，只是我们需要二分找到当前可以填写的最大值.

现在我们已经确定好了搜索后,我们就需要剪枝了.

```
#include <bits/stdc++.h>
using namespace std;
const int N=(1<<24)+1;
long long ans,n,m,a[N],s[N],n_2;
long long find(int val){  //二分查找
    int l=1,r=n_2,check=m-val;//最大可以承受的值
    while(l<r){
        int mid=(l+r+1)>>1;
        if (s[mid]<=check)
            l=mid;
        else
            r=mid-1;
    }
    ans=max(ans,s[l]+val);//当前最大值与全局最大值开始比较
}
int cmp(long long a,long long b){
    return a>b;
}
int dfs(int x,long long sum){
    if (x==(n/2+2)+1){
        s[++n_2]=sum;//新的权值出现,压入数组中
        return true;//返回必不可少,否则RE
    }
    dfs(x+1,sum);//不放这个进去
    if (sum+a[x]<=m)//可以放进去
        dfs(x+1,sum+a[x]);
}
int dfs2(int x,int sum){
//  cout<<x<<endl;
    if (x==n+1)
    {
        find(sum);//求出当前可以填充的最大值
        return true;
    }
    dfs2(x+1,sum);
    if (sum+a[x]<=m)//如果可以放进去
        dfs2(x+1,sum+a[x]);
}
int main(){
    ios::sync_with_stdio(false);
    cin>>m>>n;
    for(int i=1; i<=n; i++)
        cin>>a[i];
```

```
    sort(a+1,a+1+n,cmp);//从大到小排序
    dfs(1,0);
    sort(s+1,s+n_2+1);
    n_2=unique(s+1,s+n_2+1)-(s+1);//去掉重复后,多少个数
    dfs2(n/2+3,0);
    cout<<ans<<endl;
    return 0;
}
```

图论



Dijkstra

朴素 O(n*n+m)

```
int g[N][N];  // 存储每条边
int dist[N];  // 存储1号点到每个点的最短距离
bool st[N];   // 存储每个点的最短路是否已经确定

// 求1号点到n号点的最短路，如果不存在则返回-1
int dijkstra(){
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    for (int i = 0; i < n - 1; i ++ ){
        int t = -1;     // 在还未确定最短路的点中，寻找距离最小的点
        for (int j = 1; j <= n; j ++ )
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
```

```
                t = j;

        // 用t更新其他点的距离
        for (int j = 1; j <= n; j ++ )
            dist[j] = min(dist[j], dist[t] + g[t][j]);

        st[t] = true;
    }

    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}
```

堆优化 O(mlogN)

```
typedef pair<int, int> PII;

int n;      // 点的数量
int h[N], w[N], e[N], ne[N], idx;       // 邻接表存储所有边
int dist[N];        // 存储所有点到1号点的距离
bool st[N];     // 存储每个点的最短距离是否已确定

// 求1号点到n号点的最短距离，如果不存在，则返回-1
int dijkstra(){
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, 1});      // first存储距离，second存储节点编号

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

```
        }
    }

    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}
```

Bellman-Ford

O(n*m)

```
int n, m;       // n 表示点数，m 表示边数
int dist[N];        // dist[x]存储 1 到 x 的最短路距离

struct Edge     // 边，a 表示出点，b 表示入点，w 表示边的权重
{
    int a, b, w;
}edges[M];

// 求 1 到 n 的最短路距离，如果无法从 1 走到 n，则返回 -1。
int bellman_ford(){
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    // 如果第 n 次迭代仍然会松弛三角不等式，就说明存在一条长度是 n+1 的最短路径，由抽屉原理，路径中至少存在两个相同的点，说明图中存在负权回路。
    for (int i = 0; i < n; i ++ ){
        for (int j = 0; j < m; j ++ ){
            int a = edges[j].a, b = edges[j].b, w = edges[j].w;
            if (dist[b] > dist[a] + w)
                dist[b] = dist[a] + w;
        }
    }

    if (dist[n] > 0x3f3f3f3f / 2) return -1;
    return dist[n];
}
```

Spfa

平均情况 O(m)，最坏 O(n*m)

```
int n;      // 总点数
int h[N], w[N], e[N], ne[N], idx;       // 邻接表存储所有边
int dist[N];        // 存储每个点到1号点的最短距离
bool st[N];     // 存储每个点是否在队列中

// 求1号点到n号点的最短路距离，如果从1号点无法走到n号点则返回-1
int spfa(){
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    queue<int> q;
    q.push(1);
    st[1] = true;

    while (q.size()){
        auto 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];
                if (!st[j]) {  // 如果队列中已存在j，则不需要将j重复插入
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }

    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}
```

Floyd

```
初始化：
    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= n; j ++ )
            if (i == j) d[i][j] = 0;
            else d[i][j] = INF;

// 算法结束后，d[a][b]表示a到b的最短距离
```

```
void floyd(){
    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]);
}
```

朴素版 prim 算法

O(n*n+m)

```
int n;      // n 表示点数
int g[N][N];        // 邻接矩阵，存储所有边
int dist[N];        // 存储其他点到当前最小生成树的距离
bool st[N];     // 存储每个点是否已经在生成树中


// 如果图不连通，则返回 INF(值是 0x3f3f3f3f)，否则返回最小生成树的树边权重之和
int prim(){
    memset(dist, 0x3f, sizeof dist);

    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]))
                t = j;

        if (i && dist[t] == INF) return INF;

        if (i) res += dist[t];
        st[t] = true;

        for (int j = 1; j <= n; j ++ ) dist[j] = min(dist[j], g[t][j]);
    }

    return res;
}
```

Kruskal

O(mlogm)

```
int n, m;       // n 是点数，m 是边数
```

```
int p[N];       // 并查集的父节点数组

struct Edge {   // 存储边
    int a, b, w;

    bool operator< (const Edge &W)const{
        return w < W.w;
    }
}edges[M];

int find(int x){    // 并查集核心操作
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

int kruskal(){
    sort(edges, edges + m);

    for (int i = 1; i <= n; i ++ ) p[i] = i;    // 初始化并查集

    int res = 0, cnt = 0;
    for (int i = 0; i < m; i ++ ){
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;

        a = find(a), b = find(b);
        if (a != b) {  // 如果两个连通块不连通，则将这两个连通块合并
            p[a] = b;
            res += w;
            cnt ++ ;
        }
    }

    if (cnt < n - 1) return INF;
    return res;
}
```

染色法判二分图

O(n+m)

```
int n;      // n 表示点数
int h[N], e[M], ne[M], idx;     // 邻接表存储图
int color[N];       // 表示每个点的颜色，-1 表示未染色，0 表示白色，1 表示黑色

```

```
// 参数：u 表示当前节点，c 表示当前点的颜色
bool dfs(int u, int c){
    color[u] = c;
    for (int i = h[u]; i != -1; i = ne[i]){
        int j = e[i];
        if (color[j] == -1){
            if (!dfs(j, !c)) return false;
        }
        else if (color[j] == c) return false;
    }
    return true;
}

bool check(){
    memset(color, -1, sizeof color);
    bool flag = true;
    for (int i = 1; i <= n; i ++ )
        if (color[i] == -1)
            if (!dfs(i, 0)){
                flag = false;
                break;
            }
    return flag;
}
```

应用

给定一个 N 行 N 列的棋盘，已知某些格子禁止放置。

求最多能往棋盘上放多少块的长度为 2、宽度为 1 的骨牌，骨牌的边界与格线重合（骨牌占用两个格子），并且任意两张骨牌都不重叠。

输入格式

第一行包含两个整数 N 和 t，其中 t 为禁止放置的格子的数量。

接下来 t 行每行包含两个整数 x 和 y，表示位于第 x 行第 y 列的格子禁止放置，行列数从 1 开始。

输出格式

输出一个整数，表示结果。

```
#include<bits/stdc++.h>
#define x first
#define y second
using namespace std;
typedef pair<int, int> PII;
const int N = 110;int n, m;
PII match[N][N];
bool g[N][N], st[N][N];
```

```
int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};

bool find(int x, int y){
    for (int i = 0; i < 4; i ++ ){
        int a = x + dx[i], b = y + dy[i];
        if (a && a <= n && b && b <= n && !g[a][b] && !st[a][b]){
            st[a][b] = true;
            PII t = match[a][b];
            if (t.x == -1 || find(t.x, t.y)){
                match[a][b] = {x, y};
                return true;
            }
        }
    }
    return false;
}

int main(){
    cin >> n >> m;
    while (m -- ){
        int x, y;
        cin >> x >> y;
        g[x][y] = true;
    }

    memset(match, -1, sizeof match);

    int res = 0;
    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= n; j ++ )
            if ((i + j) % 2 && !g[i][j]){
                memset(st, 0, sizeof st);
                if (find(i, j)) res ++ ;
            }
    cout << res << endl;
    return 0;
}
```

匈牙利算法

O(n*m)

```
int n1, n2;     // n1 表示第一个集合中的点数，n2 表示第二个集合中的点数
```

```
int h[N], e[M], ne[M], idx;     // 邻接表存储所有边，匈牙利算法中只会用到从第一个集合指向第二个集合的边，所以这里只用存一个方向的边
int match[N];       // 存储第二个集合中的每个点当前匹配的第一个集合中的点是哪个
bool st[N];     // 表示第二个集合中的每个点是否已经被遍历过

bool find(int x){
    for (int i = h[x]; i != -1; i = ne[i]){
        int j = e[i];
        if (!st[j]){
            st[j] = true;
            if (match[j] == 0 || find(match[j])){
                match[j] = x;
                return true;
            }
        }
    }
    return false;
}

// 求最大匹配数，依次枚举第一个集合中的每个点能否匹配第二个集合中的点
int res = 0;
for (int i = 1; i <= n1; i ++ ){
    memset(st, false, sizeof st);
    if (find(i)) res ++ ;
}
```

拓扑排序

O(n+m)

```
bool topsort(){
    int hh = 0, tt = -1;

    // d[i] 存储点 i 的入度
    for (int i = 1; i <= n; i ++ )
        if (!d[i])
            q[ ++ tt] = i;

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

        for (int i = h[t]; i != -1; i = ne[i]){
            int j = e[i];
            if (-- d[j] == 0)
```

```
                q[ ++ tt] = j;
            }
        }

        // 如果所有点都入队了，说明存在拓扑序列；否则不存在拓扑序列。
        return tt == n - 1;
    }
```

综合运用:DFS，二分，DP，拓扑排序

扩展:虚拟源点，最短路计数

最短路计数

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10,mod=100003;
vector<int> v[N];
int dis[N];
int cnt[N];
void bfs(){
    memset(dis,0x3f,sizeof(dis));
    dis[1]=1;
    cnt[1]=1;
    queue<int> q;
    q.push(1);
    while(!q.empty()){
        auto t=q.front();
        q.pop();
        for(int i=0;i<v[t].size();i++){
            if(dis[v[t][i]]>dis[t]+1){
                q.push(v[t][i]);
                dis[v[t][i]]=dis[t]+1;
                cnt[v[t][i]]=cnt[t];
            }
            else if(dis[v[t][i]]==dis[t]+1){
                cnt[v[t][i]]=(cnt[t]+cnt[v[t][i]])%mod;
            }
        }
    }
}
int main(){
    int n,m;
    cin>>n>>m;
```

```
    while(m--){
        int a,b;
        cin>>a>>b;
        v[a].push_back(b);
        v[b].push_back(a);
    }
    bfs();
    for(int i=1;i<=n;i++)
    {
        cout<<cnt[i]<<endl;
    }
}
```

求图中的起点到终点的最小路径和次小路径

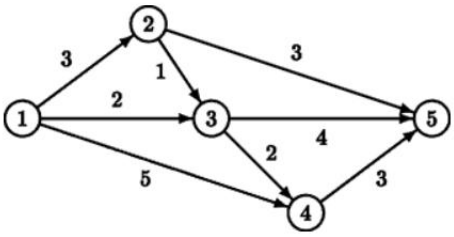
每天公共汽车都会从一座城市开往另一座城市。

沿途汽车可能会在一些城市（零或更多）停靠。

旅行社计划旅途从 S 城市出发，到 F 城市结束。

由于不同旅客的景点偏好不同，所以为了迎合更多旅客，旅行社将为客户提供多种不同线路。

游客可以选择的行进路线有所限制，要么满足所选路线总路程为 S 到 F 的最小路程，要么满足所选路线总路程仅比最小路程多一个单位长度。



如上图所示，如果 $S=1$，$F=5$，则这里有两条最短路线 $1\to 2\to 5, 1\to 3\to 5$，长度为 6；有一条比最短路程多一个单位长度的路线 $1\to 3\to 4\to 5$，长度为 7。

现在给定比荷卢经济联盟的公交路线图以及两个城市 S 和 F，请你求出旅行社最多可以为旅客提供多少种不同的满足限制条件的线路。

枚举城市t可通往的城市j时，有四种情况：`(dijkstra只要dist发生改变，就将该点入队)`

1. $dist[j][0] > dist[v][type] + w[i]$：当前最短路变成次短路，更新最短路，将最短路和次短路加入优先队列
   到达j的最短路个数和到达t是一样的：$cnt[j][0] = cnt[t][type]$
2. $dist[j][0] = dist[v][type] + w[i]$：找到一条新的最短路，更新最短路条数
   到达j的最短路个数应该加上到达t的最短路个数，从t经过的最短路，在j上经过的时候也是最短路：$cnt[j][0] += cnt[t][type]$
3. $dist[j][1] > dist[v][type] + w[i]$：找到一条更短的次短路，覆盖掉当前次短路，加入优先队列
   到达j的最短路个数和到达t是一样的：$cnt[j][1] = cnt[t][type]$
4. $dist[j][1] = dist[v][type] + w[i]$：找到一条新的次短路，更新次短路条数
   到达j的最短路个数应该加上到达t的最短路个数，从t经过的最短路，在j上经过的时候也是最短路：$cnt[j][1] += cnt[t][type]$

```cpp
#include <bits/stdc++,h>
using namespace std;
const int N = 1010, M = 20010;
struct Ver{
    int id, type, dist;
    bool operator> (const Ver &W) const
    {
        return dist > W.dist;
    }
};

int n, m, S, T;
int h[N], e[M], w[M], ne[M], idx;
int dist[N][2], cnt[N][2];
bool st[N][2];

void add(int a, int b, int c){
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
}

int dijkstra(){
    memset(st, 0, sizeof st);
    memset(dist, 0x3f, sizeof dist);
    memset(cnt, 0, sizeof cnt);

    dist[S][0] = 0, cnt[S][0] = 1;
    priority_queue<Ver, vector<Ver>, greater<Ver>> heap;
    heap.push({S, 0, 0});

    while (heap.size()){
        Ver t = heap.top();
```

```
32.         heap.pop();
33.
34.         int ver = t.id, type = t.type, distance = t.dist, count = cnt[ver][type];
35.         if (st[ver][type]) continue;
36.         st[ver][type] = true;
37.
38.         for (int i = h[ver]; ~i; i = ne[i])
39.         {
40.             int j = e[i];
41.             if (dist[j][0] > distance + w[i])
42.             {
43.                 dist[j][1] = dist[j][0], cnt[j][1] = cnt[j][0];
44.                 heap.push({j, 1, dist[j][1]});
45.                 dist[j][0] = distance + w[i], cnt[j][0] = count;
46.                 heap.push({j, 0, dist[j][0]});
47.             }
48.             else if (dist[j][0] == distance + w[i]) cnt[j][0] += count;
49.             else if (dist[j][1] > distance + w[i])
50.             {
51.                 dist[j][1] = distance + w[i], cnt[j][1] = count;
52.                 heap.push({j, 1, dist[j][1]});
53.             }
54.             else if (dist[j][1] == distance + w[i]) cnt[j][1] += count;
55.         }
56.     }
57.
58.     int res = cnt[T][0];
59.     if (dist[T][0] + 1 == dist[T][1]) res += cnt[T][1];
60.
61.     return res;
62. }
63.
64. int main(){
65.     int cases;
66.     scanf("%d", &cases);
67.     while (cases -- ){
68.         scanf("%d%d", &n, &m);
69.         memset(h, -1, sizeof h);
70.         idx = 0;
71.
72.         while (m -- ){
73.             int a, b, c;
74.             scanf("%d%d%d", &a, &b, &c);
75.             add(a, b, c);
```

```
        }
        scanf("%d%d", &S, &T);

        printf("%d\n", dijkstra());
    }
    return 0;
}
```

作为一个城市的应急救援队伍的负责人，你有一张特殊的全国地图。在地图上显示有多个分散的城市和一些连接城市的快速道路。每个城市的救援队数量和每一条连接两个城市的快速道路长度都标在地图上。当其他城市有紧急求助电话给你的时候，你的任务是带领你的救援队尽快赶往事发地，同时，一路上召集尽可能多的救援队。

输入格式:

输入第一行给出4个正整数N、M、S、D，其中N（$2 \le N \le 500$）是城市的个数，顺便假设城市的编号为0 ~ ($N-1$)；M是快速道路的条数；S是出发地的城市编号；D是目的地的城市编号。

第二行给出N个正整数，其中第i个数是第i个城市的救援队的数目，数字间以空格分隔。随后的M行中，每行给出一条快速道路的信息，分别是：城市1、城市2、快速道路的长度，中间用空格分开，数字均为整数且不超过500。输入保证救援可行且最优解唯一。

输出格式:

第一行输出最短路径的条数和能够召集的最多的救援队数量。第二行输出从S到D的路径中经过的城市编号。数字间以空格分隔，输出结尾不能有多余空格。

```
#include<bits/stdc++.h>
using namespace std;
int n,m,s,d;
const int N=520;
int a[N],g[N][N],st[N],dis[N],fa[N],pnum[N],rnum[N];
void dijkstra(){
    memset(dis,0x3f,sizeof(dis));
    dis[s]=0;
    rnum[s]=a[s];
    pnum[s]=1;
    fa[s]=-1;
    for(int i=0;i<n-1;i++){
        int t=-1;
        for(int j=0;j<n;j++){
            if(!st[j]&&(t==-1||dis[j]<dis[t]))
                t=j;
        }
        if(t==-1)return ;
        st[t]=1;
        for(int j=0;j<n;j++){
            if(!st[j]&&dis[j]>dis[t]+g[t][j]){
                dis[j]=dis[t]+g[t][j];
                fa[j]=t;
                pnum[j]=pnum[t];
                rnum[j]=rnum[t]+a[j];
            }
```

```
            else if(!st[j]&&dis[j]==dis[t]+g[t][j]){
                pnum[j]+=pnum[t];
                if(rnum[t]+a[j]>=rnum[j]){
                      rnum[j]=rnum[t]+a[j];
                      fa[j]=t;
                }

            }
        }
    }
}
int main(){
    cin>>n>>m>>s>>d;
    memset(g,0x3f,sizeof(g));
    for(int i=0;i<n;i++)cin>>a[i];
    while(m--){
        int x,y,z;
        cin>>x>>y>>z;
        g[x][y]=g[y][x]=min(g[x][y],z);
    }
    dijkstra();
    cout<<pnum[d]<<' '<<rnum[d]<<endl;
    vector<int> v;
    while(d!=-1){
        v.push_back(d);
        d=fa[d];
    }
    for(int i=v.size()-1;i>=0;i--){
        if(i==v.size()-1)cout<<v[i];
        else cout<<' '<<v[i];
    }
    cout<<endl;
}
```

次小生成树

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 510, M = 10010;
int n, m;
struct Edge{
    int a, b, w;
```

```
        bool f;
        bool operator< (const Edge &t) const{
            return w < t.w;
        }
    }edge[M];
    int p[N];
    int dist1[N][N], dist2[N][N];
    int h[N], e[N * 2], w[N * 2], ne[N * 2], idx;

    void add(int a, int b, int c){
        e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
    }

    int find(int x){
        if (p[x] != x) p[x] = find(p[x]);
        return p[x];
    }

    void dfs(int u, int fa, int maxd1, int maxd2, int d1[], int d2[]){
        d1[u] = maxd1, d2[u] = maxd2;
        for (int i = h[u]; ~i; i = ne[i]){
            int j = e[i];
            if (j != fa){
                int td1 = maxd1, td2 = maxd2;
                if (w[i] > td1) td2 = td1, td1 = w[i];
                else if (w[i] < td1 && w[i] > td2) td2 = w[i];
                dfs(j, u, td1, td2, d1, d2);
            }
        }
    }

    int main(){
        scanf("%d%d", &n, &m);
        memset(h, -1, sizeof h);
        for (int i = 0; i < m; i ++ ){
            int a, b, w;
            scanf("%d%d%d", &a, &b, &w);
            edge[i] = {a, b, w};
        }

        sort(edge, edge + m);
        for (int i = 1; i <= n; i ++ ) p[i] = i;

        LL sum = 0;
```

```
    for (int i = 0; i < m; i ++ ){
        int a = edge[i].a, b = edge[i].b, w = edge[i].w;
        int pa = find(a), pb = find(b);
        if (pa != pb){
            p[pa] = pb;
            sum += w;
            add(a, b, w), add(b, a, w);
            edge[i].f = true;
        }
    }

    for (int i = 1; i <= n; i ++ ) dfs(i, -1, -1e9, -1e9, dist1[i], dist2[i]);

    LL res = 1e18;
    for (int i = 0; i < m; i ++ )
        if (!edge[i].f){
            int a = edge[i].a, b = edge[i].b, w = edge[i].w;
            LL t;
            if (w > dist1[a][b])
                t = sum + w - dist1[a][b];
            else if (w > dist2[a][b])
                t = sum + w - dist2[a][b];
            res = min(res, t);
        }

    printf("%lld\n", res);

    return 0;
}
```

环+01 分数规划

> 给定一张 L 个点、P 条边的有向图，每个点都有一个权值 $f[i]$，每条边都有一个权值 $t[i]$。
>
> 求图中的一个环，使“环上各点的权值之和”除以“环上各边的权值之和”最大。
>
> 输出这个最大值。
>
> **注意**：数据保证至少存在一个环。

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

using namespace std;

```

```
const int N = 1010, M = 5010;

int n, m;
int wf[N];
int h[N], e[M], wt[M], ne[M], idx;
double dist[N];
int q[N], cnt[N];
bool st[N];

void add(int a, int b, int c){
    e[idx] = b, wt[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
}

bool check(double mid){
    memset(dist, 0, sizeof dist);
    memset(st, 0, sizeof st);
    memset(cnt, 0, sizeof cnt);

    int hh = 0, tt = 0;
    for (int i = 1; i <= n; i ++ ){
        q[tt ++ ] = i;
        st[i] = true;
    }

    while (hh != tt){
        int t = q[hh ++ ];
        if (hh == N) hh = 0;
        st[t] = false;

        for (int i = h[t]; ~i; i = ne[i]){
            int j = e[i];
            if (dist[j] < dist[t] + wf[t] - mid * wt[i]){
                dist[j] = dist[t] + wf[t] - mid * wt[i];
                cnt[j] = cnt[t] + 1;
                if (cnt[j] >= n) return true;
                if (!st[j]){
                    q[tt ++ ] = j;
                    if (tt == N) tt = 0;
                    st[j] = true;
                }
            }
        }
    }

```

```
51.     return false;
52. }
53. 
54. int main(){
55.     cin >> n >> m;
56.     for (int i = 1; i <= n; i ++ ) cin >> wf[i];
57. 
58.     memset(h, -1, sizeof h);
59.     for (int j = 0; j < m; j ++ ){
60.         int a, b, c;
61.         cin >> a >> b >> c;
62.         add(a, b, c);
63.     }
64. 
65.     double l = 0, r = 1e6;
66.     while (r - l > 1e-4){
67.         double mid = (l + r) / 2;
68.         if (check(mid)) l = mid;
69.         else r = mid;
70.     }
71.     printf("%.2lf\n", l);
72.     return 0;
73. }
```

最近公共祖先

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.  int root;
4.  const int N=4e4+10;
5.  vector<int> v[N];
6.  int depth[N];
7.  int fa[N][20];
8.  void bfs(){
9.      memset(depth,0x3f,sizeof(depth));
10.     depth[0]=0;
11.     depth[root]=1;
12.     queue<int> q;
13.     q.push(root);
14.     while(!q.empty()){
15.         auto t=q.front();
16.         q.pop();
17.         for(int i=0;i<v[t].size();i++){
```

```
            int j=v[t][i];
            if(depth[j]>depth[t]+1){
                depth[j]=depth[t]+1;
                q.push(j);
                fa[j][0]=t;
                for(int k=1;k<=15;k++){
                    fa[j][k]=fa[fa[j][k-1]][k-1];
                }
            }
        }
    }
}
int lca(int a,int b){
    if(depth[a]<depth[b]){
        swap(a,b);
    }
    for(int i=15;i>=0;i--){
        if(depth[fa[a][i]]>=depth[b]){
            a=fa[a][i];
        }
    }
    if(a==b)return a;
    for(int i=15;i>=0;i--){
        if(fa[a][i]!=fa[b][i]){
            a=fa[a][i];
            b=fa[b][i];
        }
    }
    return fa[a][0];
}
int main(){
    int n;
    cin>>n;
    for(int i=0;i<n;i++){
        int a,b;
        cin>>a>>b;
        if(b==-1)root=a;
        else{
            v[a].push_back(b);
            v[b].push_back(a);
        }
    }
    bfs();

```

```
    int m;
    cin>>m;
    while(m--){
        int a,b;
        cin>>a>>b;
        int x=lca(a,b);
        if(x==a)cout<<1<<endl;
        else if(x==b)cout<<2<<endl;
        else cout<<0<<endl;
    }
}
```

Tarjan 求最近公共祖先

```
#include<bits/stdc++.h>
using namespace std;

const int N=1e4+10;

int ans[N*2];
vector<pair<int,int>> q[N];
vector<pair<int,int>> v[N];
int p[N];
int st[N];
int dis[N];

void dfs(int x,int fa){

    for(int i=0;i<v[x].size();i++){
        int j=v[x][i].first;
        int w=v[x][i].second;
        if(j==fa)continue;
        dis[j]=dis[x]+w;
        dfs(j,x);
    }

}
int find(int x){
    if(p[x]!=x)p[x]=find(p[x]);
    return p[x];
}
void tarjan(int  x){

    st[x]=1;
    for(int i=0;i<v[x].size();i++){
```

```
32.         int j=v[x][i].first;
33.         if(st[j])continue;
34.         tarjan(j);
35.         p[j]=x;
36.     }
37. 
38.     for(auto t:q[x]){
39.         int y=t.first,id=t.second;
40.         if(st[y]==2){
41.             int fa=find(y);
42.             ans[id]=dis[x]+dis[y]-dis[fa]*2;
43.         }
44.     }
45. 
46.     st[x]=2;
47. }
48. 
49. int main(){
50.     int n,m;
51.     cin>>n>>m;
52.     memset(dis,0x3f,sizeof(dis));
53.     for(int i=0;i<n-1;i++){
54.         int a,b,c;
55.         cin>>a>>b>>c;
56.         v[a].push_back({b,c});
57.         v[b].push_back({a,c});
58.     }
59. 
60.     for(int i=1;i<=n;i++)p[i]=i;
61.     dfs(1,-1);
62. 
63.     for(int i=0;i<m;i++){
64.         int a,b;
65.         cin>>a>>b;
66.         q[a].push_back({b,i});
67.         q[b].push_back({a,i});
68.     }
69. 
70.     tarjan(1);
71. 
72.     for(int i=0;i<m;i++){
73.         cout<<ans[i]<<endl;
74.     }
75. }
```

LCA 求次小生成树

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 100010, M = 300010, INF = 0x3f3f3f3f;
int n, m;
struct Edge{
    int a, b, w;
    bool used;
    bool operator< (const Edge &t) const{
        return w < t.w;
    }
}edge[M];
int p[N];
int h[N], e[M], w[M], ne[M], idx;
int depth[N], fa[N][17], d1[N][17], d2[N][17];
int q[N];

void add(int a, int b, int c){
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
}

int find(int x){
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

LL kruskal(){
    for (int i = 1; i <= n; i ++ ) p[i] = i;
    sort(edge, edge + m);
    LL res = 0;
    for (int i = 0; i < m; i ++ ){
        int a = find(edge[i].a), b = find(edge[i].b), w = edge[i].w;
        if (a != b){
            p[a] = b;
            res += w;
            edge[i].used = true;
        }
    }
    return res;
}

void build(){
```

```
    memset(h, -1, sizeof h);
    for (int i = 0; i < m; i ++ )
        if (edge[i].used){
            int a = edge[i].a, b = edge[i].b, w = edge[i].w;
            add(a, b, w), add(b, a, w);
        }
}

void bfs(){
    memset(depth, 0x3f, sizeof depth);
    depth[0] = 0, depth[1] = 1;
    q[0] = 1;
    int hh = 0, tt = 0;
    while (hh <= tt){
        int t = q[hh ++ ];
        for (int i = h[t]; ~i; i = ne[i]){
            int j = e[i];
            if (depth[j] > depth[t] + 1){
                depth[j] = depth[t] + 1;
                q[ ++ tt] = j;
                fa[j][0] = t;
                d1[j][0] = w[i], d2[j][0] = -INF;
                for (int k = 1; k <= 16; k ++ ){
                    int anc = fa[j][k - 1];
                    fa[j][k] = fa[anc][k - 1];
                    int distance[4] = {d1[j][k - 1], d2[j][k - 1], d1[anc][k - 1], d2[anc][k - 1]};
                    d1[j][k] = d2[j][k] = -INF;
                    for (int u = 0; u < 4; u ++ ){
                        int d = distance[u];
                        if (d > d1[j][k]) d2[j][k] = d1[j][k], d1[j][k] = d;
                        else if (d != d1[j][k] && d > d2[j][k]) d2[j][k] = d;
                    }
                }
            }
        }
    }
}

int lca(int a, int b, int w){
    static int distance[N * 2];
    int cnt = 0;
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 16; k >= 0; k -- )
```

```
86.         if (depth[fa[a][k]] >= depth[b]){
87.             distance[cnt ++ ] = d1[a][k];
88.             distance[cnt ++ ] = d2[a][k];
89.             a = fa[a][k];
90.         }
91.     if (a != b){
92.         for (int k = 16; k >= 0; k -- )
93.             if (fa[a][k] != fa[b][k]){
94.                 distance[cnt ++ ] = d1[a][k];
95.                 distance[cnt ++ ] = d2[a][k];
96.                 distance[cnt ++ ] = d1[b][k];
97.                 distance[cnt ++ ] = d2[b][k];
98.                 a = fa[a][k], b = fa[b][k];
99.             }
100.        distance[cnt ++ ] = d1[a][0];
101.        distance[cnt ++ ] = d1[b][0];
102.    }
103.
104.    int dist1 = -INF, dist2 = -INF;
105.    for (int i = 0; i < cnt; i ++ ){
106.        int d = distance[i];
107.        if (d > dist1) dist2 = dist1, dist1 = d;
108.        else if (d != dist1 && d > dist2) dist2 = d;
109.    }
110.
111.    if (w > dist1) return w - dist1;
112.    if (w > dist2) return w - dist2;
113.    return INF;
114. }
115.
116. int main(){
117.    scanf("%d%d", &n, &m);
118.    for (int i = 0; i < m; i ++ ){
119.        int a, b, c;
120.        scanf("%d%d%d", &a, &b, &c);
121.        edge[i] = {a, b, c};
122.    }
123.
124.    LL sum = kruskal();
125.    build();
126.    bfs();
127.
128.    LL res = 1e18;
129.    for (int i = 0; i < m; i ++ )
```

```
        if (!edge[i].used){
            int a = edge[i].a, b = edge[i].b, w = edge[i].w;
            res = min(res, sum + lca(a, b, w));
        }
    printf("%lld\n", res);
    return 0;
}
```

有向图的强连通分量

每一头牛的愿望就是变成一头最受欢迎的牛。

现在有 N 头牛，编号从 1 到 N，给你 M 对整数 (A,B)，表示牛 A 认为牛 B 受欢迎。

这种关系是具有传递性的，如果 A 认为 B 受欢迎，B 认为 C 受欢迎，那么牛 A 也认为牛 C 受欢迎。

你的任务是求出有多少头牛被除自己之外的所有牛认为是受欢迎的。

输入格式

第一行两个数 N,M；

接下来 M 行，每行两个数 A,B，意思是 A 认为 B 是受欢迎的（给出的信息有可能重复，即有可能出现多个 A,B）。

输出格式

输出被除自己之外的所有牛认为是受欢迎的牛的数量。

```
#include<bits/stdc++.h>
using namespace std;

const int N = 10010, M = 50010;

int n, m;
int h[N], e[M], ne[M], idx, dfn[N], low[N], timestamp,stk[N], top;
bool in_stk[N];
int id[N], scc_cnt, Size[N], dout[N];

void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

void tarjan(int u){
    dfn[u]=low[u]=++timestamp;
    stk[++top]=u;
    in_stk[u]=true;
    for(int i=h[u];i!=-1;i=ne[i]){
```

```
        int j=e[i];
        if(!dfn[j]){
            tarjan(j);
            low[u]=min(low[j],low[u]);
        }
        else if(in_stk[j]){
            low[u]=min(low[u],dfn[j]);
        }
    }
    if(dfn[u]==low[u]){
        scc_cnt++;
        int y;
        do{
            y=stk[top--];
            Size[scc_cnt]++;
            in_stk[y]=false;
            id[y]=scc_cnt;
        }while(y!=u);
    }
}
int main(){
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    while (m -- ){
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
    }

    for (int i = 1; i <= n; i ++ )
        if (!dfn[i])
            tarjan(i);

    for (int i = 1; i <= n; i ++ )
        for (int j = h[i]; ~j; j = ne[j]){
            int k = e[j];
            int a = id[i], b = id[k];
            if (a != b) dout[a] ++ ;
        }

    int zeros = 0, sum = 0;
    for (int i = 1; i <= scc_cnt; i ++ )
        if (!dout[i]){
            zeros ++ ;
```

```
            sum += Size[i];
            if (zeros > 1){
                sum = 0;
                break;
            }
        }
    printf("%d\n", sum);
    return 0;
}

```

无向图的强联通分量

```
#include<bits/stdc++.h>
using namespace std;

using namespace std;

const int N = 5010, M = 20010;

int n, m;
int h[N], e[M], ne[M], idx;
int dfn[N], low[N], timestamp;
int stk[N], top;
int id[N], dcc_cnt;
bool is_bridge[M];
int d[N];

void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

void tarjan(int u, int from){
    dfn[u] = low[u] = ++ timestamp;
    stk[ ++ top] = u;

    for (int i = h[u]; ~i; i = ne[i]){
        int j = e[i];
        if (!dfn[j]){
            tarjan(j, i);
            low[u] = min(low[u], low[j]);
            if (dfn[u] < low[j])
                is_bridge[i] = is_bridge[i ^ 1] = true;
        }
        else if (i != (from ^ 1))
```

```
33.             low[u] = min(low[u], dfn[j]);
34.     }
35.
36.     if (dfn[u] == low[u]){
37.         ++ dcc_cnt;
38.         int y;
39.         do {
40.             y = stk[top -- ];
41.             id[y] = dcc_cnt;
42.         } while (y != u);
43.     }
44. }
45.
46. int main(){
47.     cin >> n >> m;
48.     memset(h, -1, sizeof h);
49.     while (m -- ){
50.         int a, b;
51.         cin >> a >> b;
52.         add(a, b), add(b, a);
53.     }
54.
55.     tarjan(1, -1);
56.
57.     for (int i = 0; i < idx; i ++ )
58.         if (is_bridge[i])
59.             d[id[e[i]]] ++ ;
60.
61.     int cnt = 0;
62.     for (int i = 1; i <= dcc_cnt; i ++ )
63.         if (d[i] == 1)
64.             cnt ++ ;
65.     printf("%d\n", (cnt + 1) / 2);
66.     return 0;
67. }
```

欧拉回路

给定一张图，请你找出欧拉回路，即在图中找一个环使得每条边都在环上出现恰好一次。

输入格式

第一行包含一个整数 t，$t \in \{1,2\}$，如果 $t=1$，表示所给图为**无向**图，如果 $t=2$，表示所给图为**有向**图。

第二行包含两个整数 n,m，表示图的结点数和边数。

接下来 m 行中，第 i 行两个整数 v_i,u_i，表示第 i 条边（从 1 开始编号）。

- 如果 $t=1$ 则表示 v_i 到 u_i 有一条无向边。
- 如果 $t=2$ 则表示 v_i 到 u_i 有一条有向边。

图中可能有重边也可能有自环。

点的编号从 1 到 n。

输出格式

如果无法一笔画出欧拉回路，则输出一行：NO。

否则，输出一行：YES，接下来一行输出 **任意一组** 合法方案即可。

- 如果 $t=1$，输出 m 个整数 $p_1,p_2,\dots,p_m$。令 $e=|p_i|$，那么 e 表示经过的第 i 条边的编号。如果 p_i 为正数表示从 v_e 走到 u_e，否则表示从 u_e 走到 v_e。
- 如果 $t=2$，输出 m 个整数 $p_1,p_2,\dots,p_m$。其中 p_i 表示经过的第 i 条边的编号。

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
const int N=100010,M=400010;
int type;
int n,m;
int h[N],e[M],ne[M],idx;
bool used[M];
int ans[M>>1],cnt; //因为无向图边扩了一倍，但答案不需要，所以这里除以二
int din[N],dout[N];
void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
void dfs(int u){
    //这里不能写 for(int i=h[u];~i;i=ne[i]) 而是写成现在的样子
    //因为 删边实际上只是在修改 h[u] 的值，但是 i 本身还是在遍历
    //如果遇到自环的情况还是遍历完所有边，从而被卡成 O(m^2) 故需要更改写法
    //这里的写法等同于每次删掉离 h[u] 最近的那条边
    for(int &i=h[u];~i;){
        if(used[i]){
            i=ne[i]; //一旦用过一条边，就把它删掉
            continue;
        }
        used[i]=true;
```

```
27.         if(type==1) used[i^1]=true;
28.         //记录边的编号
29.         int t;
30.         if(type==1){
31.             t=i/2+1;
32.             if(i & 1) t=-t;
33.         } else t=i+1;
34.
35.         int v=e[i];
36.         i=ne[i];
37.         dfs(v);
38.         ans[++cnt]=t;
39.     }
40. }
41. int main(){
42.     scanf("%d",&type);
43.     scanf("%d %d",&n,&m);
44.     memset(h,-1,sizeof h);
45.     for(int i=0;i<m;++i){
46.         int a,b;
47.         scanf("%d %d",&a,&b);
48.         add(a,b);
49.         if(type==1) add(b,a);
50.         din[b]++,dout[a]++;
51.     }
52.     //对于条件1. 的判断
53.     if(type==1){
54.         for(int i=1;i<=n;++i)
55.             if(din[i]+dout[i] &1){
56.                 puts("NO");
57.                 return 0;
58.             }
59.     } else {
60.         for(int i=1;i<=n;++i)
61.             if(din[i]!=dout[i]){
62.                 puts("NO");
63.                 return 0;
64.             }
65.     }
66.     //因为只要求边联通，不要求点联通，故找到第一个不孤立点
67.     for(int i=1;i<=n;++i)
68.         if(h[i]!=-1){
69.             dfs(i);
70.             break;
```

```
        }
    //判断遍历到的边的数量和m是不是相等的
    if(cnt<m){
        puts("NO");
        return 0;
    }
    puts("YES");
    for(int i=cnt;i;--i) printf("%d ",ans[i]);
    return 0;
}
```

动态规划

背包问题

01 背包

```
const int N=2e5+10;
int n, m;
int v[N],w[N];
int f[N];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
    for(int i=1;i<=n;i++){
        for(int j=m;j>=v[i];j--){
                f[j]=max(f[j],f[j-v[i]]+w[i]);
        }
    }
    cout<<f[m]<<endl;
}
```

完全背包

```
int main(){
    cin >> n >> m;
    for (int i = 1; i <= n; i ++ ) cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i ++ )
        for (int j = v[i]; j <= m; j ++ )
            f[j] = max(f[j], f[j - v[i]] + w[i]);
```

```
7.
8.      cout << f[m] << endl;
9.  }
```

多重背包

```
1.  const int N = 110;
2.
3.  int n, m;
4.  int v[N], w[N], s[N];
5.  int f[N][N];
6.
7.  int main()
8.  {
9.      cin >> n >> m;
10.
11.     for (int i = 1; i <= n; i ++ ) cin >> v[i] >> w[i] >> s[i];
12.
13.     for (int i = 1; i <= n; i ++ )
14.         for (int j = 0; j <= m; j ++ )
15.             for (int k = 0; k <= s[i] && k * v[i] <= j; k ++ )
16.                 f[i][j] = max(f[i][j], f[i - 1][j - v[i] * k] + w[i] * k);
17.
18.     cout << f[n][m] << endl;
19.     return 0;
20. }
```

优化

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.  const int N=2e5+10;
4.  int v[N],s[N],w[N];
5.  int f[N];
6.  int n,m;
7.  int main(){
8.      cin>>n>>m;
9.      int cnt=0;
10.     for(int i=1;i<=n;i++){
11.         int a,b,c;
12.         cin>>a>>c>>b;
13.         int k=1;
14.         while(k<=b){
15.             v[++cnt]=a*k;
16.             w[cnt]=c*k;
```

```
            b-=k;
            k*=2;
        }
        if(b>0){
            v[++cnt]=a*b;
            w[cnt]=c*b;
        }
    }
    for(int i=1;i<=cnt;i++){
        for(int j=m;j>=v[i];j--){
            f[j]=max(f[j],f[j-v[i]]+w[i]);
        }
    }
    cout<<f[m]<<endl;

}
```

优化优化

```
const int N = 20010;
int n, m;
int f[N], g[N], q[N];
int main(){
    cin >> n >> m;
    for (int i = 0; i < n; i ++ ){
        int v, w, s;
        cin >> v >> w >> s;
        memcpy(g, f, sizeof f);
        for (int j = 0; j < v; j ++ ){
            int hh = 0, tt = -1;
            for (int k = j; k <= m; k += v){
                if (hh <= tt && q[hh] < k - s * v) hh ++ ;
                while (hh <= tt && g[q[tt]] - (q[tt] - j) / v * w <= g[k] - (k - j) / v * w) tt -- ;
                q[ ++ tt] = k;
                f[k] = g[q[hh]] + (k - q[hh]) / v * w;
            }
        }
    }
    cout << f[m] << endl;
}
```

分组背包

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+10;
int v[N][N],w[N][N],cnt[N];
int f[N][N];
int n,m;
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        int x;
        cin>>x;
        cnt[i]=x;
        for(int j=1;j<=x;j++){
            cin>>v[i][j]>>w[i][j];
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=0;j<=m;j++){
            for(int k=1;k<=cnt[i];k++){
                f[i][j]=max(f[i][j],f[i-1][j]);
                if(j>=v[i][k])
                f[i][j]=max(f[i][j],f[i-1][j-v[i][k]]+w[i][k]);
            }
        }
    }
    cout<<f[n][m]<<endl;
}
```

混合背包

有 N 种物品和一个容量是 V 的背包。

物品一共有三类：

- 第一类物品只能用1次（01背包）；
- 第二类物品可以用无限次（完全背包）；
- 第三类物品最多只能用 s_i 次（多重背包）；

每种体积是 v_i，价值是 w_i。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
输出最大价值。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

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

```
const int N = 1010;
int n, m;
int f[N];

int main(){
    cin >> n >> m;

    for (int i = 0; i < n; i ++ ){
        int v, w, s;
        cin >> v >> w >> s;
        if (!s){
            for (int j = v; j <= m; j ++ )
                f[j] = max(f[j], f[j - v] + w);
        }
        else{
            if (s == -1) s = 1;
            for (int k = 1; k <= s; k *= 2){
                for (int j = m; j >= k * v; j -- )
                    f[j] = max(f[j], f[j - k * v] + k * w);
                s -= k;
            }
            if (s){
                for (int j = m; j >= s * v; j -- )
                    f[j] = max(f[j], f[j - s * v] + s * w);
            }
        }
    }
    cout << f[m] << endl;
}
#include<bits/stdc++.h>
```

```
using namespace std;
#define int long long
int n,m;
const int N=1e7+10;
int w[N],v[N],f[N];
signed main(){
    cin>>n>>m;
    int cnt=0;
    for(int i=1;i<=n;i++){
        int a,b,x;
        cin>>a>>b>>x;
        if(x==-1){
            v[++cnt]=a;
            w[cnt]=b;
            continue;
        }
        if(x==0)x=m/a+1;
        int k=1;
        while(k<=x){
            v[++cnt]=a*k;
            w[cnt]=b*k;
            x-=k;
            k*=2;
        }
        if(x>0){
            v[++cnt]=a*x;
            w[cnt]=b*x;
        }
    }
    for(int i=1;i<=cnt;i++){
        for(int j=m;j>=v[i];j--){
            f[j]=max(f[j],f[j-v[i]]+w[i]);
        }
    }
    cout<<f[m]<<endl;
}
```

二维费用的背包问题

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+10;
int v1[N],v2[N],w[N];
int dp[N][N];
int main(){
```

```
    int n,v,m;
    cin>>n>>v>>m;
    for(int i=1;i<=n;i++){
        cin>>v1[i]>>v2[i]>>w[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=v;j>=v1[i];j--){
            for(int k=m;k>=v2[i];k--){
                dp[j][k]=max(dp[j][k],dp[j-v1[i]][k-v2[i]]+w[i]);
            }
        }
    }
    cout<<dp[v][m]<<endl;
}
```

背包问题求方案数

有 N 件物品和一个容量是 V 的背包。每件物品只能使用一次。

第 i 件物品的体积是 v_i，价值是 w_i。

求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

输出 **最优选法的方案数**。注意答案可能很大，请输出答案模 10^9+7 的结果。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品数量和背包容积。

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

输出格式

输出一个整数，表示 **方案数** 模 10^9+7 的结果。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1010, mod = 1e9 + 7;

int n, m;
int f[N], g[N];

int main(){
    cin >> n >> m;

    memset(f, -0x3f, sizeof f);
    f[0] = 0;
    g[0] = 1;
```

```
14.
15.     for (int i = 0; i < n; i ++ ){
16.         int v, w;
17.         cin >> v >> w;
18.         for (int j = m; j >= v; j -- ){
19.             int maxv = max(f[j], f[j - v] + w);
20.             int s = 0;
21.             if (f[j] == maxv) s = g[j];
22.             if (f[j - v] + w == maxv) s = (s + g[j - v]) % mod;
23.             f[j] = maxv, g[j] = s;
24.         }
25.     }
26.
27.     int res = 0;
28.     for (int i = 1; i <= m; i ++ )
29.         if (f[i] > f[res])
30.             res = i;
31.
32.     int sum = 0;
33.     for (int i = 0; i <= m; i ++ )
34.         if (f[i] == f[res])
35.             sum = (sum + g[i]) % mod;
36.
37.     cout << sum << endl;
38.
39.     return 0;
40. }
```

背包问题求具体方案

有 N 件物品和一个容量是 V 的背包。每件物品只能使用一次。

第 i 件物品的体积是 v_i，价值是 w_i。

求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

输出 **字典序最小的方案**。这里的字典序是指：所选物品的编号所构成的序列。物品的编号范围是 $1 \dots N$。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品数量和背包容积。

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

输出格式

输出一行，包含若干个用空格隔开的整数，表示最优解中所选物品的编号序列，且该编号序列的字典序最小。

物品编号范围是 $1 \dots N$。

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

int main(){
    cin >> n >> m;
    for (int i = 1; i <= n; i ++ ) cin >> v[i] >> w[i];

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

    int j = m;
    for (int i = 1; i <= n; i ++ )
        if (j >= v[i] && f[i][j] == f[i + 1][j - v[i]] + w[i]){
            cout << i << ' ';
            j -= v[i];
        }
}
```

有依赖的背包问题

如果选择物品5，则必须选择物品1和2。这是因为2是5的父节点，1是2的父节点。

每件物品的编号是 i，体积是 v_i，价值是 w_i，依赖的父节点编号是 p_i。物品的下标范围是 $1 \dots N$。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

输入格式

第一行有两个整数 N，V，用空格隔开，分别表示物品个数和背包容量。

接下来有 N 行数据，每行数据表示一个物品。
第 i 行有三个整数 v_i, w_i, p_i，用空格隔开，分别表示物品的体积、价值和依赖的物品编号。
如果 $p_i = -1$，表示根节点。 **数据保证所有物品构成一棵树。**

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 110;
int n, m;
int v[N], w[N];
int h[N], e[N], ne[N], idx;
```

```
int f[N][N];
void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

void dfs(int u){
    for (int i = h[u]; ~i; i = ne[i]){  // 循环物品组
        int son = e[i];
        dfs(e[i]);

        // 分组背包
        for (int j = m - v[u]; j >= 0; j -- )  // 循环体积
            for (int k = 0; k <= j; k ++ )  // 循环决策
                f[u][j] = max(f[u][j], f[u][j - k] + f[son][k]);
    }

    // 将物品 u 加进去
    for (int i = m; i >= v[u]; i -- ) f[u][i] = f[u][i - v[u]] + w[u];
    for (int i = 0; i < v[u]; i ++ ) f[u][i] = 0;
}

int main(){
    cin >> n >> m;

    memset(h, -1, sizeof h);
    int root;
    for (int i = 1; i <= n; i ++ ){
        int p;
        cin >> v[i] >> w[i] >> p;
        if (p == -1) root = i;
        else add(p, i);
    }
    dfs(root);
    cout << f[root][m] << endl;

}

```

线性 DP

最长上升子序列

```
#include<bits/stdc++.h>
using namespace std;
```

```
const int N=1e3+10,INF=1e9;
int a[N],dp[N];
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    a[0]=-INF;
    for(int i=1;i<=n;i++){
        dp[i]=1;
        for(int j=1;j<i;j++){
            if(a[j]<a[i]){
                dp[i]=max(dp[i],dp[j]+1);
            }
        }
    }
    int ans=0;
    for(int i=1;i<=n;i++)ans=max(ans,dp[i]);
    cout<<ans<<endl;
}
```

优化

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
int n,a[N],q[N];

int main(){
    scanf("%d", &n);
    for (int i = 0; i < n; i ++ ) scanf("%d", &a[i]);

    int len = 0;
    for (int i = 0; i < n; i ++ ){
        int l = 0, r = len;
        while (l < r){
            int mid = l + r + 1 >> 1;
            if (q[mid] < a[i]) l = mid;
            else r = mid - 1;
        }
        len = max(len, r + 1);
        q[r + 1] = a[i];
    }
   printf("%d\n", len);
}
```

最长公共子序列

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+10;
int dp[N][N];
int main(){
    int n,m;
    cin>>n>>m;
    string s1="?",s2="?";
    string s;
    cin>>s;s1+=s;
    cin>>s;s2+=s;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
            if(s1[i]==s2[j])dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
        }
    }
    cout<<dp[n][m]<<endl;
}
```

同时走

设有 N×N 的方格图，我们在其中的某些方格中填入正整数，而其它的方格中则放入数字0。如下图所示：

A

0	0	0	0	0	0	0	0
0	0	13	0	0	6	0	0
0	0	0	0	7	0	0	0
0	0	0	14	0	0	0	0
0	21	0	0	0	4	0	0
0	0	15	0	0	0	0	0
0	14	0	0	0	0	0	0
0	0	0	0	0	0	0	0

B

某人从图中的左上角 A 出发，可以向下行走，也可以向右行走，直到到达右下角的 B 点。

在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字0）。

此人从 A 点到 B 点共走了两次，试找出两条这样的路径，使得取得的数字和为最大。

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 15;
int n;
int w[N][N];
int f[N * 2][N][N];

int main(){
```

```
    scanf("%d", &n);
    int a, b, c;
    while (cin >> a >> b >> c, a || b || c) w[a][b] = c;

    for (int k = 2; k <= n + n; k ++ )
        for (int i1 = 1; i1 <= n; i1 ++ )
            for (int i2 = 1; i2 <= n; i2 ++ ){
                int j1 = k - i1, j2 = k - i2;
                if (j1 >= 1 && j1 <= n && j2 >= 1 && j2 <= n){
                    int t = w[i1][j1];
                    if (i1 != i2) t += w[i2][j2];
                    int &x = f[k][i1][i2];
                    x = max(x, f[k - 1][i1 - 1][i2 - 1] + t);
                    x = max(x, f[k - 1][i1 - 1][i2] + t);
                    x = max(x, f[k - 1][i1][i2 - 1] + t);
                    x = max(x, f[k - 1][i1][i2] + t);
                }
            }
    printf("%d\n", f[n + n][n][n]);
}
```

传纸条

```
#include<bits/stdc++.h>
using namespace std;
const int N = 55;
int g[N][N];
int n, m;
int dp[N*2][N][N];
int main() {
 cin >> n >> m;
 for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= m; j++) {
   cin >> g[i][j];
  }
 }

    for(int k=2;k<=n+m;k++){
        for(int i=1;i<=n&&i<k;i++){
            for(int j=1;j<=n&&j<k;j++){
                if(k!=2&&k!=n+m&&i==j)continue;
                int t=g[i][k-i]+g[j][k-j];
                int maxv=max(dp[k-1][i][j],dp[k-1][i-1][j-1]);
                maxv=max(maxv,dp[k-1][i-1][j]);maxv=max(maxv,dp[k-1][i][j-1]);
                dp[k][i][j]=maxv+t;
```

```
            }
        }
    }
 cout<<dp[n+m][n][n]<<endl;
}
```

最长上升子序列和

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+10;
int a[N],dp1[N],dp2[N];

int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i],dp1[i]=1,dp2[i]=1;
    for(int i=1;i<=n;i++){
        for(int j=1;j<i;j++){
            if(a[j]<a[i])dp1[i]=max(dp1[i],dp1[j]);
        }
        dp1[i]+=a[i];
    }

    int ans=0;

    for(int i=1;i<=n;i++){
        ans=max(ans,dp1[i]);
    }
    cout<<ans-1<<endl;
}
```

导弹防御系统

为了对抗附近恶意国家的威胁，R 国更新了他们的导弹防御系统。

一套防御系统的导弹拦截高度要么一直 **严格单调** 上升要么一直 **严格单调** 下降。

例如，一套系统先后拦截了高度为 3 和高度为 4 的两发导弹，那么接下来该系统就只能拦截高度大于 4 的导弹。

给定即将袭来的一系列导弹的高度，请你求出至少需要多少套防御系统，就可以将它们全部击落。

输入格式

输入包含多组测试用例。

对于每个测试用例，第一行包含整数 n，表示来袭导弹数量。

第二行包含 n 个**不同的**整数，表示每个导弹的高度。

当输入测试用例 $n = 0$ 时，表示输入终止，且该用例无需处理。

输出格式

对于每个测试用例，输出一个占据一行的整数，表示所需的防御系统数量。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 55;
int n,h[N],up[N], down[N],ans;

void dfs(int u, int su, int sd){
    if (su + sd >= ans) return;
    if (u == n){
        ans = min(ans, su + sd);
        return;
    }

    int k = 0;
    while (k < su && up[k] >= h[u]) k ++ ;
    if (k < su){
        int t = up[k];
        up[k] = h[u];
        dfs(u + 1, su, sd);
        up[k] = t;
    }
    else{
        up[k] = h[u];
        dfs(u + 1, su + 1, sd);
    }

    k = 0;
    while (k < sd && down[k] <= h[u]) k ++ ;
    if (k < sd){
        int t = down[k];
```

```
            down[k] = h[u];
            dfs(u + 1, su, sd);
            down[k] = t;
        }
        else{
            down[k] = h[u];
            dfs(u + 1, su, sd + 1);
        }
    }

int main(){
    while (cin >> n, n){
        for (int i = 0; i < n; i ++ ) cin >> h[i];
        ans = n;
        dfs(0, 0, 0);
        cout << ans << endl;
    }
    return 0;
}

```

最长公共上升子序列

```
#include <iostream>
using namespace std;
const int N = 3010;
int n,a[N], b[N],f[N][N];
int main(){
    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 ++ ){
        int maxv = 1;
        for (int j = 1; j <= n; j ++ ){
            f[i][j] = f[i - 1][j];
            if (a[i] == b[j]) f[i][j] = max(f[i][j], maxv);

            if (b[j] < a[i])
                maxv = max(maxv, f[i - 1][j] + 1);
        }
    }
    int res = 0;
    for (int i = 1; i <= n; i ++ ) res = max(res, f[n][i]);
```

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

区间 DP

石子合并

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

using namespace std;
const int N=300+10;
int s[N],f[N][N];
int n;
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>s[i],s[i]=s[i-1]+s[i];

    for(int len=2;len<=n;len++){
        for(int i=1;i+len-1<=n;i++){
            int j=i+len-1;
            f[i][j]=2e9;
            for(int k=i;k<j;k++){
                f[i][j]=min(f[i][j],f[i][k]+f[k+1][j]+s[j]-s[i-1]);
            }
        }
    }
    cout<<f[1][n]<<endl;
}
```

计数类 DP

```
#include<bits/stdc++.h>
using namespace std;

int n;
const int N=1e3+10,mod=1e9+7;
int f[N][N];

int main(){
    cin>>n;
    f[0][0]=1;
    for(int i=1;i<=n;i++){
        for(int j=0;j<=n;j++){
            f[i][j]=f[i-1][j];
            if(j>=i)
```

```
            f[i][j]=(f[i][j]+f[i][j-i])%mod;
        }
    }
    cout<<f[n][n]<<endl;
}
```

数位统计 DP

给定两个整数 a 和 b，求 a 和 b 之间的所有数字中 $0 \sim 9$ 的出现次数。

例如，$a = 1024$，$b = 1032$，则 a 和 b 之间共有 9 个数如下：

```
1024 1025 1026 1027 1028 1029 1030 1031 1032
```

其中 `0` 出现 10 次，`1` 出现 10 次，`2` 出现 7 次，`3` 出现 3 次等等…

输入格式

输入包含多组测试数据。

每组测试数据占一行，包含两个整数 a 和 b。

当读入一行为 `0 0` 时，表示输入终止，且该行不作处理。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 10;
/*
001~abc-1, 999
abc
    1. num[i] < x, 0
    2. num[i] == x, 0~efg
    3. num[i] > x, 0~999
*/

int get(vector<int> num, int l, int r){
    int res = 0;
    for (int i = l; i >= r; i -- ) res = res * 10 + num[i];
    return res;
}

int power10(int x){
    int res = 1;
    while (x -- ) res *= 10;
    return res;
}
```

```
23.
24.  int count(int n, int x){
25.      if (!n) return 0;
26.
27.      vector<int> num;
28.      while (n){
29.          num.push_back(n % 10);
30.          n /= 10;
31.      }
32.      n = num.size();
33.      int res = 0;
34.      for (int i = n - 1 - !x; i >= 0; i -- ){
35.          if (i < n - 1){
36.              res += get(num, n - 1, i + 1) * power10(i);
37.              if (!x) res -= power10(i);
38.          }
39.          if (num[i] == x) res += get(num, i - 1, 0) + 1;
40.          else if (num[i] > x) res += power10(i);
41.      }
42.      return res;
43.  }
44.
45.  int main(){
46.      int a, b;
47.      while (cin >> a >> b , a){
48.          if (a > b) swap(a, b);
49.
50.          for (int i = 0; i <= 9; i ++ )
51.              cout << count(b, i) - count(a - 1, i) << ' ';
52.          cout << endl;
53.      }
54.  }
```

状态压缩 DP

求把 $N \times M$ 的棋盘分割成若干个 1×2 的长方形，有多少种方案。

例如当 $N = 2$，$M = 4$ 时，共有 5 种方案。当 $N = 2$，$M = 3$ 时，共有 3 种方案。

如下图所示：

输入格式

输入包含多组测试用例。

每组测试用例占一行，包含两个整数 N 和 M。

当输入用例 $N = 0$，$M = 0$ 时，表示输入终止，且该用例无需处理。

```
typedef long long LL;
const int N = 12, M = 1 << N;
int n, m;
LL f[N][M];
vector<int> state[M];
bool st[M];
int main(){
    while (cin >> n >> m, n || m){
        for (int i = 0; i < 1 << n; i ++ ){
            int cnt = 0;
            bool is_valid = true;
            for (int j = 0; j < n; j ++ )
                if (i >> j & 1){
                    if (cnt & 1){
                        is_valid = false;
                        break;
                    }
                    cnt = 0;
                }
                else cnt ++ ;
            if (cnt & 1) is_valid = false;
            st[i] = is_valid;
        }

        for (int i = 0; i < 1 << n; i ++ ){
            state[i].clear();
            for (int j = 0; j < 1 << n; j ++ )
                if ((i & j) == 0 && st[i | j])
                    state[i].push_back(j);
        }

```

```
        memset(f, 0, sizeof f);
        f[0][0] = 1;
        for (int i = 1; i <= m; i ++ )
            for (int j = 0; j < 1 << n; j ++ )
                for (auto k : state[j])
                    f[i][j] += f[i - 1][k];

        cout << f[m][0] << endl;
    }
    return 0;
}
```

树形 DP

```
#include <bits/stdc++.h>
using namespace std;
const int N = 6010;
int n,h[N], e[N], ne[N], idx
int happy[N],f[N][2];
bool has_fa[N];
void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

void dfs(int u){
    f[u][1] = happy[u];
    for (int i = h[u]; ~i; i = ne[i]){
        int j = e[i];
        dfs(j);
        f[u][1] += f[j][0];
        f[u][0] += max(f[j][0], f[j][1]);
    }
}

int main(){
    scanf("%d", &n);
    for (int i = 1; i <= n; i ++ ) scanf("%d", &happy[i]);
    memset(h, -1, sizeof h);
    for (int i = 0; i < n - 1; i ++ ){
        int a, b;
        scanf("%d%d", &a, &b);
        add(b, a);
        has_fa[a] = true;
    }
```

```
    int root = 1;
    while (has_fa[root]) root ++ ;
    dfs(root);
    printf("%d\n", max(f[root][0], f[root][1]));
    return 0;
}
```

STL

```
vector, 变长数组，倍增的思想
    size()  返回元素个数
    empty()  返回是否为空
    clear()  清空
    front()/back()
    push_back()/pop_back()
    begin()/end()
    []
    支持比较运算，按字典序

pair<int, int>
    first, 第一个元素
    second, 第二个元素
    支持比较运算，以 first 为第一关键字，以 second 为第二关键字（字典序）

string，字符串
    size()/length()  返回字符串长度
    empty()
    clear()
    substr(起始下标，(子串长度))  返回子串
    c_str()  返回字符串所在字符数组的起始地址

queue，队列
    size()
    empty()
    push()  向队尾插入一个元素
    front()  返回队头元素
    back()  返回队尾元素
    pop()  弹出队头元素

priority_queue，优先队列，默认是大根堆
```

```
    size()
    empty()
    push()  插入一个元素
    top()  返回堆顶元素
    pop()  弹出堆顶元素
    定义成小根堆的方式：priority_queue<int, vector<int>, greater<int>> q;

stack，栈
    size()
    empty()
    push()  向栈顶插入一个元素
    top()  返回栈顶元素
    pop()  弹出栈顶元素

deque，双端队列
    size()
    empty()
    clear()
    front()/back()
    push_back()/pop_back()
    push_front()/pop_front()
    begin()/end()
    []

set，map，multiset，multimap，基于平衡二叉树（红黑树），动态维护有序序列
    size()
    empty()
    clear()
    begin()/end()
    ++，-- 返回前驱和后继，时间复杂度 O(logn)

    set/multiset
        insert()  插入一个数
        find()  查找一个数
        count()  返回某一个数的个数
        erase()
            (1) 输入是一个数 x，删除所有 x   O(k + logn)
            (2) 输入一个迭代器，删除这个迭代器
        lower_bound()/upper_bound()
            lower_bound(x)  返回大于等于 x 的最小的数的迭代器
            upper_bound(x)  返回大于 x 的最小的数的迭代器
    map/multimap
        insert()  插入的数是一个 pair
        erase()  输入的参数是 pair 或者迭代器
```

```
        find()
        [] 注意 multimap 不支持此操作。 时间复杂度是 O(logn)
        lower_bound()/upper_bound()

unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
    和上面类似，增删改查的时间复杂度是 O(1)
    不支持 lower_bound()/upper_bound()， 迭代器的++, --

bitset, 压位
    bitset<10000> s;
    ~, &, |, ^
    >>, <<
    ==, !=
    []

    count() 返回有多少个 1

    any() 判断是否至少有一个 1
    none() 判断是否全为 0

    set() 把所有位置成 1
    set(k, v) 将第 k 位变成 v
    reset() 把所有位变成 0
    flip() 等价于~
    flip(k) 把第 k 位取反
```

Vector

```
#include<iostream>
#include<vector>
using namespace std;
int main () {
    //几种初始化的方法
     vector<int> a;//定义一个 vector  未初始化 输出》 0

     vector<int> a(3);//定义一个长度为 3 的 vector  未初始化 输出》0 0 0

     vector<int> a(10, 3); //定义一个长度为 10，且每个数赋值为 3

    //将向量 b 中从下标 0 1 2（共三个）的元素赋值给 a，a 的类型为 int 型
    //它的初始化不和数组一样
    vector<int>a(b.begin(),b.begin+3);

    //从数组中获得初值
```

```
    int b[7]={1,2,3,4,5,6,7};
    vector<int> a(b,b+7) ;

    for(auto x : a) {//遍历输出
        cout << x << " ";
    }
    return 0;
}
```

```
a.resize( )//改变大小
a.front();//返回 a 的第 1 个元素,当且仅当 a 存在
a.back(); //返回 vector 的最后一个数
int main () {
    //支持比较运算
    vector<int> a(4, 3), b(3, 4);
    //a: 3 3 3 3   b:4 4 4
    //比较原理字典序 (根据最前面那个判断，如果一样就往后比较)
    if (a < b) {
        puts("a < b");
    }
    return 0;
}
a.pop_back();//删除 a 向量的最后一个元素
a.push_back(5);//在 a 的最后一个向量后插入一个元素，其值为 5
```

Pair

当 pair 结合 sort()函数使用的时候， pair 默认对 first 升序，当 first 相同时对 second 升序（从小到大）。 也可以通过修改 cmp 函数达到对 second 就行排序

String

```
int main () {
    string a = "ac";
    a += "w";//支持比较操作符>,>=,<,<=,==,!=
    cout << a << endl; //输出子串 a :acw

    a += "ing";
    cout << a << endl;
    //以字符串数组理解
    cout << a.substr(0, 3) << endl; //当第一个数是 0 则后一位数:输出从
```

```
头开始的长度为 3 的子串
    cout << a.substr(0, 3) << endl; //当第一个数是 1 则输出下标为 1 到
下标为 3 的子串
    cout << a.substr(0, 9) << endl;//如果超出长度范围 则输出原子串
    cout << a.substr(1) << endl; //从下标为 1 开始输出
    cout << a.substr(0) << endl; //原子串
    printf("%s\n", a.c_str());//如果用 printf 输出

    return 0;
}


// 尾插一个字符
    a.push_back('a');
// insert(pos,char):在制定的位置 pos 前插入字符 char
    a.insert(a.begin(),'1'); //输出 1a
  //插入字符串
  string str2="hello";
    string s2="weakhaha";
    str2.insert(0,s2,1,3);
//将字符串 s2 从下标为 1 的 e 开始数 3 个字符，分别是 eak，插入原串的下标
为 0 的字符 h 前
```

queue【队列】 和 priority_queue 【优先队列、堆】

```
//queue <类型> 变量名
//priority_queue <类型> 变量名;
queue <int> q; //定义一个名为 q 队列
priority_queue <int> q; //默认是大根堆
//定义小根堆
小根堆：priority_queue <类型,vecotr <类型>,greater <类型>> 变量名
```

deque【双向队列】

```
dq.size(); //返回这个双端队列的长度
dq.empty(); //返回这个队列是否为空空则返回 t 非空则返回 f
dq.clear(); //清空这个双端队列
dq.front(); //返回第一个元素
dq.back(); //返回最后一个元素
dq.push_back(); //向最后插入一个元素
dq.pop_back(); //弹出最后一个元素
dq.push_front(); //向队首插入一个元素
```

```
dq.pop_front();//弹出第一个元素
dq.begin(); //双端队列的第 0 个数
dq.end(); //双端队列的最后一个的数的后面一个数
```

set 【集合】和 multiset

```
size();// 返回元素个数
empty(); //返回 set 是否是空的
clear(); //清空
begin(); //第 0 个数，支持++或--，返回前驱和后继
end(); //最后一个的数的后面一个数,支持+或-，返回前和后
insert(); //插入一个数
find(); //查找一个数
count(); //返回某一个数的个数
erase(x); //删除所以 x  时间复杂度 O(k + logn)
erase(s.begin(),s.end());//删除一个迭代器
lower_bound(x); //返回大于等于 x 的最小的数的迭代器
upper_bound(x); //返回大于 x 的最小的数的迭代器  不存在返回 end()
```

map 【映射】 /multimap

```
insert(); //插入一个数，插入的数是一个 pair
erase();
    //（1）输入是 pair
    //（2）输入一个迭代器，删除这个迭代器
find(); //查找一个数
lower_bound(x); //返回大于等于 x 的最小的数的迭代器
upper_bound(x); //返回大于 x 的最小的数的迭代器
```

bitset 【压位】

```
count(); //返回 1 的个数
any(); //判断是否至少有一个 1
none(); //判断是否全为 0
set(); //把所有位置赋值为 1
set(k,v); //将第 k 位变成 v
reset(); //把所有位变成 0
flip(); //把所有位取反，等价于~
flip(k); //把第 k 位取反
```

常用函数

```
__gcd()
reverse(v.begin(),v.end());//v 的值为 5，4，3，2，1  倒置
```

```
ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);
#define INF  0x3f3f3f3f; //无穷大 10^9
#define x first ;//结合 pair
#define y second;
next_permutation
// 1 结合 数组
int a[] = {1, 2, 3, 4, 5};
    do{
      for(int i = 0; i < 5; i ++) cout << a[i] << " ";
      cout << endl;
    }while(next_permutation(a, a + 5));

// 2 结合 vector
vector<int> a = {1, 2, 3, 4, 5};
    do{
      for(int i = 0; i < a.size(); i ++) cout << a[i] << " ";
      cout << endl;
    }while(next_permutation(a.begin(), a.end()));
```

1. $n \leq 30$, 指数级别, dfs+剪枝, 状态压缩dp
2. $n \leq 100$ => $O(n^3)$, floyd, dp, 高斯消元
3. $n \leq 1000$ => $O(n^2)$, $O(n^2 logn)$, dp, 二分, 朴素版Dijkstra、朴素版Prim、Bellman-Ford
4. $n \leq 10000$ => $O(n * \sqrt{n})$, 块状链表、分块、莫队
5. $n \leq 100000$ => $O(nlogn)$ => 各种sort, 线段树、树状数组、set/map、heap、拓扑排序、dijkstra+heap、prim+heap、Kruskal、spfa、求凸包、求半平面交、二分、CDQ分治、整体二分、后缀数组、树链剖分、动态树
6. $n \leq 1000000$ => $O(n)$, 以及常数较小的 $O(nlogn)$ 算法 => 单调队列、 hash、双指针扫描、并查集, kmp、AC自动机, 常数比较小的 $O(nlogn)$ 的做法: sort、树状数组、heap、dijkstra、spfa
7. $n \leq 10000000$ => $O(n)$, 双指针扫描、kmp、AC自动机、线性筛素数
8. $n \leq 10^9$ => $O(\sqrt{n})$, 判断质数
9. $n \leq 10^{18}$ => $O(logn)$, 最大公约数, 快速幂, 数位DP
10. $n \leq 10^{1000}$ => $O((logn)^2)$, 高精度加减乘除
11. $n \leq 10^{100000}$ => $O(logk \times loglogk)$, k表示位数, 高精度加减、FFT/NTT