A:围栏

- 总时间限制:

  1000ms

- 单个测试点时间限制:

  100ms

- 内存限制:

  65536kB

- 描述

  小 A 打算新建一幢楼，这幢楼需要占用一块长方形的面积恰好为 n 平方米的土地，且为了方便测量，这块土地的长宽必须为整数米。小 A 需要在这幢楼外边修围栏，围栏长度为这块长方形的周长。现在想要知道最小的围栏长度。

- 输入

  第一行一个正整数 n($n \le 2 \times 109$)，表示楼的面积。

- 输出

  一行一个数表示答案。数据保证答案在int范围内。

- 样例输入

  `2`

- 样例输出

  `6`

- 提示

  这是一道简单的枚举题，枚举长方形的短边即可，即枚举1到根号n就可以。 最后答案在int范围内，但注意周长中间结果可能会超过int

简单枚举

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;
int main(){
    ll n;
    cin>>n;
    ll min1=4*n;
    for(int i=1;i<=sqrt(n);i++){
        if(n%i==0){
            if(min1>2*(i+n/i)){
                min1=2*(i+n/i);
            }
        }
    }
    cout<<min1<<endl;
    return 0;
```

```
}
```

B:解密

- 总时间限制:

  1000ms

- 单个测试点时间限制:

  100ms

- 内存限制:

  65536kB

- 描述

  有一种简单的加密算法，对于一个长度为n的字符串，这个算法将会以第(n+1)/2（向下取整）个字符为中间轴(最左边的字符算第1个字符），将该字符写在密文的开头，然后对左半部分按照同样的办法进行加密并写下密文，再对右半部分按照同样的办法进行加密并写下密文。以此类推，直到左右部分为空，即完成加密。例如，如果要对12345678进行加密，第一步将选择4作为中间轴，将其写在密文开头，然后继续对左右两边（123和5678）分别继续按这个算法处理并写下，我们可以将其记作4[123][5678]（[]代表待加密处理的部分）。 对于左半部分123，中间轴是2，左半部分为1，右半部分为3因此加密结果为213（1的中间轴为1，左右均为空，因此结果为1，而3同理）。 对于右半部分5678，中间轴是6，左半部分为5，右半部分为78，因此加密结果为65[78] → 6578（78的中间轴为7，左半部分为空，右半部分为8，因此得到78）。 简单来说，整个加密过程如下： 12345678 → 4[123][5678] → 42[1][3][5678]→ 4213[5678] → 42136[5][78] → 42136578 因此，对12345678的加密结果为42136578。 现在给出一个长度为n(1 ≤ n ≤ 50000)的由数字构成的字符串，这个字符串是加密后的密文，请你还原出加密前的明文。

- 输入

  一行，一个长度为n(1 ≤ n ≤ 50000)的由数字构成的字符串字符串，代表加密后的密文。

- 输出

  一行，一个长度同样为n的字符串，代表解密后的明文。

- 样例输入

  `123456789`

- 样例输出

  `324517689`

- 提示

  324517689 → 1[3245][7689] → 12[3][45][7689] → 12345[7689] → 123456[7][89] → 123456789

分治与递归

```cpp
#include <bits/stdc++.h>
using namespace std;
string code;
string x="";
int num=0;
```

```cpp
void div1(int l,int r){
    if(l>r)return ;
    if(l==r){
        x[l]=code[num++];
        return ;
    }
    int mid=(l+r)/2;
    x[mid]=code[num++];
    div1(l,mid-1);
    div1(mid+1,r);
}
int main(){
    cin>>code;
    for(int i=0;i<code.length();i++){
        x+=' ';
    }
    div1(0,code.length()-1);
    cout<<x<<endl;


    return 0;
}
```

C:传送法术

- 总时间限制:

  1000ms

- 单个测试点时间限制:

  100ms

- 内存限制:

  65536kB

- 描述

  小明在一个大小为 1 × n 的迷宫中，坐标范围为0,1,2...,n-1，maze[i] 表示迷宫第i格的地形：• 若为 . ，表示可以到达的空地；• 若为 # ，表示不可到达的墙壁；• 若为 S ，表示小明的起点位置；• 若为 T ，表示迷宫的出口位置。 小明每次可以向 左、右 相邻的位置移动一格。 此外，小明还有一个可以无限使用的传送法术，使用后会被传送到镜像的位置。 具体的，当小明在x的位置使用传送法术，会被传送到n-x-1的位置。例如，当n=5时，如果小明在0位置使用传送法术，会被传送到4的位置。 移动或使用法术都需要一步，现在小明想知道，最少几步它可以从起点走到迷宫的出口？ **注意：** • 如果传送后的位置是墙壁，则不能使用传送法术• 传送后的位置可能与原位置相同

- 输入

  第一行包含一个整数 n，表示迷宫的长度。(2 ≤ n ≤ 1000) 第二行包含一个长度为 n 的字符串maze，表示迷宫。

- 输出

一个整数，表示小明从起点走到出口最少的步数。如果无法到达，输出-1

- 样例输入

  ```
  11 S#...#T#...
  ```

- 样例输出

  ```
  7
  ```

经典bfs走迷宫

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn=1005;
char maze[maxn];
int vis[maxn];
int n;
int dx[2]={-1,1};
struct node{
    int pos;
    int step;
    node(int y,int x):pos(y),step(x){
    }
};
int main(){
    cin>>n;
    int s=0;
    for(int i=0;i<n;i++){
        cin>>maze[i];
        if(maze[i]=='S')s=i;
    }
    queue<node> q;
    q.push(node(s,0));
    vis[s]=1;
    int find=0;
    int p=0;
    while(!q.empty()){
        node tmp=q.front();
        q.pop();
        int tag=0;
        for(int i=0;i<2;i++){
            int tx=tmp.pos+dx[i];
            if(tx<0||tx>=n||vis[tx]==1||maze[tx]=='#')continue;
            vis[tx]=1;
            if(maze[tx]=='T'){
                q.push(node(tx,tmp.step+1));
                tag=1;
                find=1;
                p=tmp.step+1;
                break;
            }else{
                q.push(node(tx,tmp.step+1));
            }
```

```
        }
        int tx=n-tmp.pos-1;
        if(maze[tx]=='#'||vis[tx]==1){

        }else if(maze[tx]=='T'){
                q.push(node(tx,tmp.step+1));
                tag=1;
                find=1;
                p=tmp.step+1;
                break;
        }else{
            q.push(node(tx,tmp.step+1));
        }
        if(tag==1)break;
    }
    if(find){
        cout<<p<<endl;
    }else{
        cout<<-1<<endl;
    }
    return 0;
}
```

D:购买优惠券

- 总时间限制:

  7400ms

- 单个测试点时间限制:

  200ms

- 内存限制:

  65536kB

- 描述

  在超市的货架上有n个商品，每个商品都有一个价值ai。小明有m张优惠券，可以使用这些优惠券购买货架上连续的总价值不超过优惠券面值的若干商品。根据特殊的规定，小明必须购买m张面值相同的优惠券，而且优惠券面值越高，价格越贵。请帮助小明计算，如果他需要购买所有商品，最少需要购买的优惠券面值是多少。优惠券面值必须是整数，且一个商品不能用两张优惠券凑起来购买。比如一张优惠券购买了若干商品后还剩2元，然而下一个商品价格超过2元，那么这张优惠券剩下的2元只能作废。请注意：所有商品价格之和可能超过int表示范围，所以答案也可能超过int表示范围。该用long long 的地方就要用long long。

- 输入

  第一行两个整数n,m,分别表示商品的数量(1≤ n ≤ 105)和要分成的组数(1≤ m ≤ n)。 第二行n个整数a1,a2,...,an，表示每个商品的价值 (1 ≤ ai ≤ 105)

- 输出

  一个整数，表示最少需要购买的优惠券面值

- 样例输入

  `7 3 2 5 1 9 3 6 2`

- 样例输出

  `11`

- 提示

  分别购买[2, 5, 1], [9], [3, 6, 2]

二分答案法

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=100005;
int n,m;
int a[maxn]={0};
ll presum[maxn];
int check(ll x){
    int cnt=0;
    int s=0;
    for(int i=1;i<=n;i++){
        if(presum[i]-presum[s]<x){
            continue;
        }else if(presum[i]-presum[s]==x){
            cnt++;
            s=i;
        }else{
            cnt++;
            s=i-1;
        }
    }
    if(s!=n){
        cnt++;
    }
    return cnt;
}
int main(){
    cin>>n>>m;
    ll l=0;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        l= l>=a[i]? l:a[i];
        presum[i]=presum[i-1]+a[i];
    }
    ll r=presum[n];
    ll ans=r;
    while(l<=r){
        ll mid=(l+r)/2;
        int res=check(mid);
        if(res<=m){
```

```
            ans=mid;
            r=mid-1;
        }else{
            l=mid+1;
        }
    }
    cout<<ans<<endl;

    return 0;
}
```

E:建筑修建

- 总时间限制:

  1000ms

- 单个测试点时间限制:

  100ms

- 内存限制:

  65536kB

- 描述

  小雯打算对一个线性街区进行开发，街区的坐标为[0,m)。 现在有n个开发商要承接建筑的修建工作，第i个承包商打算修建宽度为y[i]的建筑，并保证街区包含了x[i]这个整数坐标。 建筑为一个左闭右开的区间，为了方便规划建筑的左侧必须为整数坐标，且左右边界不能超出街区范围。 例如，当m=7, x[i]=5, y[i]=3时，[3,6), [4,7)是仅有的两种合法建筑，[2,5),[5,8)则是不合法的建筑。 两个开发商修建的建筑不能有重叠。例如，[3,5)+[4,6)是不合法的，而[3,5)+[5,7)则是合法的。 小雯想要尽量满足更多开发商的修建工作，请问在合理安排的情况下，最多能满足多少个开发商的需求?

- 输入

  第一行两个整数n,m（n, m ≤ 1000） 之后n行，每行两个整数表示开发商的计划，其中第i行的整数为x[i],y[i]。 输入保证x[i]从小到大排列，且都在[0,m)之间。并且保证y[i] > 0。

- 输出

  一个整数，表示最多能满足多少个开发商的需求。

- 样例输入

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

- 样例输出

  ```
  2
  ```

挺好的贪心题

```
#include <bits/stdc++.h>
using namespace std;
int n,m;
struct node{
```

```
    int x,y;
    int mmin,mmax;
};
bool cmp(const node&x,const node&y){
    if(x.mmin+x.y==y.mmin+y.y)return x.y<y.y;
    return x.mmin+x.y<y.mmin+y.y;
}
void change(vector<node>& lis,int zero,int start){
    for(int i=start;i<n;i++){
        lis[i].mmin=max(zero,lis[i].x+1-lis[i].y);
        lis[i].mmax=min(m-lis[i].y,lis[i].x);
    }
    sort(lis.begin()+start,lis.end(),cmp);
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin>>n>>m;
    vector<node>lis(n);
    for(int i=0;i<n;i++){
        cin>>lis[i].x>>lis[i].y;
    }
    change(lis,0,0);
    int cur=0,total=0;
    for(int i=0;i<n;i++){
        if(lis[i].mmax>=cur){
            if(cur>=lis[i].mmin)cur+=lis[i].y;
            else cur=lis[i].mmin+lis[i].y;
            total++;
            change(lis,cur,i+1);
        }
    }
    cout<<total<<"\n";
    return 0;
}
```

F:预测赢家

- 总时间限制:

  1000ms

- 内存限制:

  65536kB

- 描述

  给定一个整数数组nums，玩家 1 和玩家 2 基于这个数组设计了一个游戏。玩家1和玩家2轮流进行自己的回合，玩家 1 先手。开始时，两个玩家的初始分值都是 0 。每一回合，玩家从数组的任意一端取一个数字（即首个数组元素或者末尾的数组元素），取到的数字将会从数组中移除（数组长度减1）。玩家选中的数字将会加到他的得分上。当数组中没有剩余数字可取时，游戏结束。如果玩家 1 能成为赢家，返回true。如果两个

玩家得分相等，同样认为玩家 1 是游戏的赢家，也返回true。你可以假设每个玩家的玩法都会使他的分数最大化。

- 输入

  第一行为测试组数n（n ≤ 350） 后面n行为数组初始元素个数m 与 m个元素 1 ≤ m <= 20 0 ≤ nums[i] ≤ 107

- 输出

  对于每一组测试数据，输出true 或 false

- 样例输入

  `7 3 1 5 2 4 1 5 233 7 5 242 353 531 22 231 8 231 343 63 543 54 332 541 674 3 423 552 653 11 231 343 63 543 54 332 541 674 423 552 653 6 1 1 1 1 1 1`

- 样例输出

  `false true false true true false true`

动态规划，博弈问题

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=25;
int m;
int num[maxn];
ll dp[maxn][maxn];
ll presum[maxn];
//max score
ll pro(int l,int r){
    if(l==r){
        return num[l];
    }
    //l<r
    if(dp[l][r]!=-1){
        return dp[l][r];
    }
    ll ans=0;
    ll p1=num[l]+(presum[r]-presum[l]-pro(l+1,r));
    ll p2=0;
    if(l==0){
        p2=num[r]+presum[r-1]-pro(l,r-1);
    }else{
        p2=num[r]+presum[r-1]-presum[l-1]-pro(l,r-1);
    }
    return dp[l][r]=p1>p2? p1:p2;
}
int main(){
    int t;
    cin>>t;
    while(t--){
        cin>>m;
        memset(num,0,sizeof(num));
        memset(presum,0,sizeof(presum));
```

```
        for(int i=0;i<m;i++){
            cin>>num[i];
            if(i!=0)
                presum[i]=presum[i-1]+num[i];
            else
                presum[i]=num[i];
        }
        for(int i=0;i<m;i++){
            for(int j=0;j<m;j++){
                dp[i][j]=-1;
            }
        }
        ll win=pro(0,m-1);
        if(presum[m-1]%2==0){
            if(win>=presum[m-1]/2){
                cout<<"true"<<endl;
            }else{
                cout<<"false"<<endl;
            }
        }else{
            if(win>presum[m-1]/2){
                cout<<"true"<<endl;
            }else{
                cout<<"false"<<endl;
            }
        }

    }
    return 0;
}
```

G:海拔

- 总时间限制:

  5000ms

- 单个测试点时间限制:

  500ms

- 内存限制:

  65536kB

- 描述

  一片矩形地域被横平竖直地切分成了 n × m 片方形区块. 其中位于第 i 行第 j 列的区块的平均海拔是 hi, j. 某人要从第 1 行第 1 列的区块移动至第 n 行第 m 列的区块. 每次移动时, 她只能选择一个与当前所处区块有公共边的相邻区块, 并移动至该区块. 跨越处于不同海拔的区块是相当耗费体力的. 定义一次移动的体力消耗值为该次移动涉及到的两个区块的海拔之差, 某人希望你能够帮助她找到一条能顺利抵达目的地的路径, 使得所有移动中体力消耗值的最大值尽可能小.

- 输入

  输入数据的第一行包含两个正整数 n, m, 表示矩形地域的大小. 接下来 n 行, 每行 m 个数, 第 i 行第 j 个正整数 hi, j 表示位于第 i 行第 j 列的区块的平均海拔. 1 ≤ n, m, ≤ 400, 1≤ hi, j ≤ 109.

- 输出

  输出一行一个非负整数表示最优情况下的体力消耗的最大值.

- 样例输入

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

- 样例输出

  ```
  4
  ```

- 提示

  样例解释 沿着路径 (1, 1) → (1, 2) → (1, 3) → (1, 4) → (1, 5) → (2, 5) → (3, 5) → (4, 5) 行走, 即可实现体力消耗的最大值为 4.

最短路，Dijkstra

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn=405;
#define check(x,y) (x>=0&&x<n&&y>=0&&y<m)
int h[maxn][maxn],n,m,vis[maxn][maxn],dx[4]={1,-1,0,0},dy[4]={0,0,1,-1};
struct comp{
    bool operator()(array<int,3>& x,array<int,3>& y){
        return x[0]>y[0];
    }
};
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin>>n>>m;
    for(int i=0;i<n;i++)
    for(int j=0;j<m;j++){
        cin>>h[i][j];
        vis[i][j]=numeric_limits<int>::max();
    }
    priority_queue<array<int,3>,vector<array<int,3>>,comp> q;
    q.push({0,0,0});
    vis[0][0]=1;
    int ans=numeric_limits<int>::max();
    while(q.size()){
        auto cur=q.top();
        q.pop();
        if(cur[1]==n-1&&cur[2]==m-1){
            ans=cur[0];
            break;
        }
        for(int i=0;i<4;i++){
            int xx=cur[1]+dx[i],yy=cur[2]+dy[i];
```

```
            if(check(xx,yy)){
                int mmax=max(cur[0],abs(h[cur[1]][cur[2]]-h[xx][yy]));
                if(vis[xx][yy]>mmax){
                    vis[xx][yy]=mmax;
                    q.push({mmax,xx,yy});
                }
            }
        }

    }
    cout<<ans<<"\n";
}
```

H:课程安排 II

- 总时间限制:

  1000ms

- 单个测试点时间限制:

  100ms

- 内存限制:

  65536kB

- 描述

  在 Pegion Kingdom University, 学校内一周总共有 18 个时间段可以开设课程。这些时间段被依次编号为 1 ~ 18. Pegion Kingdom University 总共有 n(n ≤ 100) 门课程开设，第 i 门课程需要占用 ki 个互不相同的时间段，能够给学生带来的知识量为 vi。 作为 Pegion Kingdom University 的学生，小 A 想要选择尽量多的课程，使得这些课程上课时间段不会互相冲突，且能够带来的知识量之和尽量的大。

- 输入

  第一行一个正整数 n(n ≤ 100)，表示课程总数。 接下来 2n 行，每两行表示一门课程的信息。具体的，对于每门课程： 第一行两个正整数 ki,vi(ki,vi ≥ 1, vi ≤ 107)，表示课程占用的时间段数，以及课程的知识量。 第二行 ki 个互不相同的正整数 ci,j，表示课程占用的时间段。保证 ci,j ≤ 18。

- 输出

  一行一个正整数，表示保证课程上课时间段不会互相冲突的前提下，最大的知识量之和。

- 样例输入

  ```
  3 1 2 1 1 3 2 2 6 1 2
  ```

- 样例输出

  ```
  6
  ```

状态压缩dp模版题

```
#include<bits/stdc++.h>
using namespace std;
int dp[1<<18];
```

```cpp
int w[101],v[101],n,k;
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>k>>v[i];
        int tmp=0,t;
        for(int j=0;j<k;j++){
            cin>>t;
            tmp+=(1<<(t-1));
        }
        w[i]=tmp;
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<(1<<18);j++){
            if(!(j&w[i])){
                int ww=j|w[i];
                dp[ww]=max(dp[ww],dp[j]+v[i]);
            }
        }
    }
    int ans=0;
    for(int j=0;j<(1<<18);j++)ans=max(ans,dp[j]);
    cout<<ans<<"\n";
    return 0;
}
```

I:取数游戏

- 总时间限制:

  10000ms

- 单个测试点时间限制:

  200ms

- 内存限制:

  65536kB

- 描述

  一个N× M的由非负整数构成的数字矩阵，你需要在其中取出若干个数字，使得取出的任意两个数字不相邻（若一个数字在另外一个数字相邻8个格子中的一个即认为这两个数字相邻），求取出数字和最大是多少。

- 输入

  第1行有一个正整数T，表示了有T组数据。（T ≤ 10） 对于每一组数据，第一行有两个正整数N和M，表示了数字矩阵为N行M列。(N, M<= 6) 接下来N行，每行M个小于100的非负整数，描述了这个数字矩阵。

- 输出

  T行，每行一个非负整数，输出所求得的答案。

- 样例输入

```
3 4 4 67 75 63 10 29 29 92 14 21 68 71 56 8 67 91 25 2 3 87 70 85 10 3 17 3 3 1 1 1 1 99 1 1 1 1
```

- 样例输出

```
271 172 99
```

类似stick的dfs+剪枝

```cpp
#include <bits/stdc++.h>
using namespace std;
int n,m,vis[10][10],num[10][10],dx[4]={0,1,1,1},dy[4]={1,-1,0,1},ans;
#define check(x,y) (x>=0&&x<n&&y>=0&&y<m)
void made(int x,int y,vector<int>& cont,int tag=0){
    for(int i=0;i<4;i++){
        int xx=x+dx[i],yy=y+dy[i];
        if(!check(xx,yy))continue;
        if(!tag)vis[xx][yy]=cont[i];
        else vis[xx][yy]=1;
    }
}
void dfs(int cur,int mm){
    if(cur>=n*m){
        ans=max(ans,mm);
        return ;
    }
    int x=cur/m,y=cur%m,tmp=cur+1;
    vector<int>cont;
    for(int i=0;i<4;i++){
        int xx=x+dx[i],yy=y+dy[i];
        if(!check(xx,yy))continue;
        cont.push_back(vis[xx][yy]);
    }
    while(tmp<n*m&&vis[tmp/m][tmp%m])tmp++;
    if(tmp>=n*m){
        ans=max(ans,mm+num[x][y]);
        return ;
    }
    vis[x][y]=1;
    dfs(tmp,mm);
    made(x,y,cont,1);
    while(tmp<n*m&&vis[tmp/m][tmp%m])tmp++;
    if(tmp>=n*m){
        ans=max(ans,mm+num[x][y]);
        vis[x][y]=0;
        made(x,y,cont);
        return ;
    }
    dfs(tmp,mm+num[x][y]);
    vis[x][y]=0;
    made(x,y,cont);
```

```
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    int t;
    cin>>t;
    while(t--){
        cin>>n>>m;
        ans=0;
        for(int i=0;i<n;i++)
        for(int j=0;j<m;j++){
            cin>>num[i][j];
            vis[i][j]=0;
        }
        dfs(0,0);
        cout<<ans<<"\n";
    }
    return 0;
}
```

J:分配工作

- 总时间限制:

  6000ms

- 单个测试点时间限制:

  1000ms

- 内存限制:

  65536kB

- 描述

  给定n项工作和k个工人，以及完成每项工作需要花费的时间。每个工人可以完成任意项工作，但每项工作都只能分配给一个工人完成，且所有工作都应被分配给工人完成。工人的工作时间是完成分配给他们的所有工作需要花费时间的总和。请设计一套工作分配方案，使得工人的最大工作时间最小化。

- 输入

  第一行为两个正整数n（2≤ n ≤12）和k（1≤ k≤ n），描述工作的数量和工人的数量。 接下来一行有n个数，描述完成每项工作需要花费的时间t[i]（1≤t[i]≤ 2×105）。

- 输出

  输出一个整数，描述最小的最大工作时间

- 样例输入

  ```
  5 2 1 2 4 7 8
  ```

- 样例输出

  ```
  11
  ```

- 提示

样例解释 样例中，可以给其中一个工人分配完成时间为1、2、8的工作，另一个工人分配完成时间为4、7的工作，此时最大工作时间为min(1+2+8,4+7)=11。其他的分配方案都使得至少有一个工人的工作时间超过11。

二分+dfs搜索

```cpp
#include <bits/stdc++.h>
using namespace std;
bool check(vector<int>& work, int k, int max_time) {
    vector<int> workers(k, 0);
    function<bool(int)> dfs = [&](int index) {
        if (index == work.size()) return true;
        for (int i = 0; i < k; ++i) {
            if (workers[i] + work[index] <= max_time) {
                workers[i] += work[index];
                if (dfs(index + 1)) return true;
                workers[i] -= work[index];
            }
            if (workers[i] == 0) break;
        }
        return false;
    };

    return dfs(0);
}
int min_max_work_time(int n, int k, vector<int>& times) {
    int left = *max_element(times.begin(), times.end());
    int right = accumulate(times.begin(), times.end(), 0);
    while (left < right) {
        int mid = (left + right) / 2;
        if (check(times, k, mid)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, k;
    cin >> n >> k;
    vector<int> times(n);
    for (int i = 0; i < n; ++i)cin >> times[i];
    int result = min_max_work_time(n, k, times);
    cout << result << "\n";
    return 0;
}
```

K:蛇入迷宫

- 总时间限制:

  3000ms

- 单个测试点时间限制:

  500ms

- 内存限制:

  65536kB

- 描述

  给定一个n*n的网格迷宫，一条身体长度为2的蛇处在迷宫的左上角（即尾巴在(0,0)，头在(0,1)），需要移动到迷宫的右下角（即尾巴在(n-1,n-2)，头在(n-1,n-1)）。迷宫中用0表示蛇可以经过的单元格，1表示障碍物。蛇只能处于水平或竖直状态，蛇的移动方式有如下几种：• 如果不碰到障碍物，则可以向下移动一个单元格，且保持身体的水平/竖直状态• 如果不碰到障碍物，则可以向右移动一个单元格，且保持身体的水平/竖直状态• 如果蛇处于水平状态，且其下方两个单元格都是空的，则可以以尾巴为轴顺时针旋转90度，即从(r,c)(r,c+1)移动到(r,c)(r+1,c)• 如果蛇处于竖直状态，且其右方两个单元格都是空的，则可以以尾巴为轴逆时针旋转90度，即从(r,c)(r+1,c)移动到(r,c)(r,c+1) 请你求出蛇抵达目的地所需的最少移动次数；如果无法到达目的地，请输出-1。

- 输入

  第一行为正整数n（2 ≤ n ≤ 100），描述迷宫的大小 接下来n行，每行n个数，描述单元格的状态（0表示空地，1表示障碍物）

- 输出

  输出一个整数，描述蛇抵达目的地所需的最少移动次数；如果无法到达目的地，请输出-1。

- 样例输入

  ```
  6 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 0 1 1 0 0 0
  ```

- 样例输出

  ```
  11
  ```

bfs走迷宫

```cpp
#include <bits/stdc++.h>
using namespace std;
int vis[105][105],bd[105][105],dx[2]={1,0},dy[2]={0,1},n;
#define check(x,y)(x>=0&&x<n&&y>=0&&y<n)
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin>>n;
    for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)cin>>bd[i][j];
    queue<array<int,4>> q;
```

```
    q.push({0,1,2,0});
    int ans=-1;
    while(q.size()){
        auto cur=q.front();
        int x1=cur[0],y1=cur[1],pos=cur[2];
        q.pop();
        if(x1==n-1&&y1==n-1&&pos==2){
            ans=cur[3];
            break;
        }
        for(int i=0;i<2;i++){
            int xx1=x1+dx[i],yy1=y1+dy[i];
            int xx2=xx1-dx[pos-1],yy2=yy1-dy[pos-1];
            if(check(xx1,yy1)&&!bd[xx1][yy1]&&!bd[xx2][yy2]&&((vis[xx1]
[yy1]&pos)==0)){
                vis[xx1][yy1]+=pos;
                q.push({xx1,yy1,pos,cur[3]+1});
            }
            if((pos==2&&i==0)||(pos==1&&i==1)){
                int tmp=((pos==2)?1:2);
                if(check(xx2,yy2)&&!bd[xx1][yy1]&&!bd[xx2][yy2]&&((vis[xx1]
[yy1]&tmp)==0)){
                    vis[xx2][yy2]+=tmp;
                    q.push({xx2,yy2,tmp,cur[3]+1});
                }
            }
        }
    }
    cout<<ans;
    return 0;
}
```

L:最长奇异子序列

- 总时间限制:

  1000ms

- 内存限制:

  65536kB

- 描述

  给出一个由个正整数组成的序列a1,a2,...an，请输出这个序列的最长奇异子序列的长度。奇异子序列是指，从原序列中按照顺序取出一些数字排在一起组成的子序列ab1,...,abm满足，b1 < b2 < ... < bm,∀1 ≤ i < m, abi & abi+1≠0

- 输入

  第一行为一个整数T，表示有T组测试样例(T ≤ 100) 之后对于每组测试样例有两行，其中第一行为序列长度n(n ≤ 100000)，第二行为n个正整数ai (ai ≤ 109)，表示这个序列。

- 输出

  输出T行，表示每组测试数据的最长奇异子序列的长度。

- 样例输入

  ```
  2 3 1 2 3 5 1 10 100 1000 10000
  ```

- 样例输出

  ```
  2 3
  ```

位运算dp

```cpp
#include <bits/stdc++.h>
using namespace std;
int n,dp[32],tmp;
int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int t;
    cin>>t;
    while(t--){
        memset(dp,0,sizeof(dp));
        cin>>n;
        for(int i=1;i<=n;i++){
            cin>>tmp;
            vector<int> pos;
            int mmax=0;
            for(int j=0;j<32;j++){
                if(tmp&(1<<j)){
                    pos.push_back(j);
                    mmax=max(mmax,dp[j]);
                }
            }
            for(auto x:pos)dp[x]=mmax+1;
        }
        int ans=0;
        for(int i=0;i<32;i++)ans=max(ans,dp[i]);
        cout<<ans<<"\n";
    }
    return 0;
}
```