第 1 周

Day01

1. 金币

> **题目来源：** 洛谷
>
> **题目链接：** P2669 [NOIP 2015 普及组] 金币

【题目描述】

> 国王将金币作为工资，发放给忠诚的骑士。第一天，骑士收到一枚金币；之后两天（第二天和第三天），每天收到两枚金币；之后三天（第四、五、六天），每天收到三枚金币；之后四天（第七、八、九、十天），每天收到四枚金币……；这种工资发放模式会一直这样延续下去：当连续 n 天每天收到 n 枚金币后，骑士会在之后的连续 $n+1$ 天里，每天收到 $n+1$ 枚金币。
>
> 请计算在前 k 天里，骑士一共获得了多少金币。

【输入描述】

> 一个正整数 k ，表示发放金币的天数。对于 100% 的数据，$1 \leq k \leq 10^4$ 。

【输出描述】

> 一个正整数，即骑士收到的金币数。

【示例一】

> 输入：
>
> 6
>
> 输出：
>
> 14
>
> 样例说明：
>
> 骑士第一天收到一枚金币；第二天和第三天，每天收到两枚金币；第四、五、六天，每天收到三枚金币。因此一共收到 1+2+2+3+3+3=14 枚金币。

【示例二】

> 输入：
>
> 1000
>
> 输出：
>
> 29820

【解法】

解法：模拟

- 搞两个变量维护一下每次需要加的数即可。

时间复杂度：$O(k)$。

【参考代码】

```cpp
#include <iostream>

using namespace std;

int main()
{
    int k; cin >> k;
    int ret = 0, tmp = 1, cnt = 1;

    for(int i = 1; i <= k; i++)
    {
        ret += tmp;
        cnt--;

        if(cnt == 0)
        {
            tmp++;
            cnt = tmp;
        }
    }

    cout << ret << endl;

    return 0;
}
```


2. 接水问题

> **题目来源：** 洛谷
>
> **题目链接：** P1190 [NOIP 2010 普及组] 接水问题

【题目描述】

学校里有一个水房，水房里一共装有 m 个龙头可供同学们打开水，每个龙头每秒钟的供水量相等，均为 1。

现在有 n 名同学准备接水，他们的初始接水顺序已经确定。将这些同学按接水顺序从 1 到 n 编号， i 号同学的接水量为 w_i。接水开始时，1 到 m 号同学各占一个水龙头，并同时打开水龙头接水。当其中某名同学 j 完成其接水量要求 w_j 后，下一名排队等候接水的同学 k 马上接替 j 同学的位置开始接水。这个换人的过程是瞬间完成的，且没有任何水的浪费。即 j 同学第 x 秒结束时完成接水，则 k 同学第 $x+1$ 秒立刻开始接水。若当前接水人数 n' 不足 m，则只有 n' 个龙头供水，其它 $m-n'$ 个龙头关闭。

现在给出 n 名同学的接水量，按照上述接水规则，问所有同学都接完水需要多少秒。

【输入描述】

第一行两个整数 n 和 m，用一个空格隔开，分别表示接水人数和龙头个数。

第二行 n 个整数 $w_1, w_2, ..., w_n$，每两个整数之间用一个空格隔开， w_i 表示 i 号同学的接水量。

$1 \le n \le 104, 1 \le m \le 100, n \le m;$

$1 \le w_i \le 100$。

【输出描述】

一个整数，表示接水所需的总时间。

【示例一】

输入：

5 3

4 4 1 2 1

输出：

4

说明：

第 1 秒，3 人接水。第 1 秒结束时，1,2,3 号同学每人的已接水量为 1,3 号同学接完水，4 号同学接替 3 号同学开始接水。

第 2 秒，3 人接水。第 2 秒结束时，1,2 号同学每人的已接水量为 2,4 号同学的已接水量为 1。

第 3 秒，3 人接水。第 3 秒结束时，1,2 号同学每人的已接水量为 3,4 号同学的已接水量为 2。4 号同学接完水，5 号同学接替 4 号同学开始接水。

第 4 秒，3 人接水。第 4 秒结束时，1,2 号同学每人的已接水量为 4,5 号同学的已接水量为 1。1,2,5 号同学接完水，即所有人完成接水的总接水时间为 4 秒。

【示例二】

> 输入：
>
> 8 4
>
> 23 71 87 32 70 93 80 76
>
> 输出：
>
> 163

【解法】

解法：模拟。

用堆模拟接水的过程即可。

时间复杂度：$O(n \log m)$。

【参考代码】

```
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

int n, m;

int main()
{
    cin >> n >> m;
    priority_queue<int, vector<int>, greater<int>> heap; // 小根堆

    for(int i = 1; i <= m; i++)
    {
        heap.push(0);
    }

    int ret = 0;
    for(int i = 1; i <= n; i++)
    {
        int x; cin >> x;

        int t = heap.top(); heap.pop();
        t += x;
        heap.push(t);
        ret = max(ret, t);
```

```
28        }
29
30        cout << ret << endl;
31
32        return 0;
33    }
```

3. 最接近神的人

> **题目来源：** 洛谷
>
> **题目链接：** P1774 最接近神的人

【题目描述】

> 破解了符文之语，小 F 开启了通往地下的道路。当他走到最底层时，发现正前方有一扇巨石门，门上雕刻着一幅古代人进行某种活动的图案。而石门上方用古代文写着“神的殿堂”。小 F 猜想里面应该就有王室的遗产了。但现在的问题是如何打开这扇门……。
>
> 仔细研究后，他发现门上的图案大概是说：古代人认为只有智者才是最容易接近神明的。而最聪明的人往往通过一种仪式选拔出来。仪式大概是指，即将隐退的智者为他的候选人写下一串无序的数字，并让他们进行一种操作，即交换序列中相邻的两个元素。而用最少的交换次数使原序列变成不下降序列的人即是下一任智者。
>
> 小 F 发现门上同样有着 n 个数字。于是他认为打开这扇门的秘诀就是找到让这个序列变成不下降序列所需要的最小次数。但小 F 不会……只好又找到了你，并答应事成之后与你三七分……

【输入描述】

> 第一行为一个整数 n，表示序列长度。
>
> 第二行为 n 个整数，表示序列中每个元素。
>
> 对于 100% 的数据 $1 \leq n \leq 5 \times 10^5, A_i \in [-2^{31}, 2^{31})$。

【输出描述】

> 一个整数 ans，即最少操作次数。

【示例一】

> 输入：
>
> 4
>
> 2 8 0 3
>
> 输出：

```
3
```

【解法】

排序的本质就是消灭逆序对的过程，最有体会的应该就是冒泡排序了。在较大的元素不断后移的过程中，逆序对的个数在逐渐减少。

因此最小操作次数就是用最少的交换，使的整个数组中逆序对的个数最小。而消灭一对逆序对，我们最少交换一次就够了。所以最小操作次数就等于逆序对的个数。

时间复杂度：$O(n \log n)$。

【参考代码】

```cpp
#include <iostream>

using namespace std;

typedef long long LL;
const int N = 5e5 + 10;

int n;
int a[N], tmp[N];

LL merge_sort(int l, int r)
{
    if(l >= r) return 0;

    LL ret = 0;
    int mid = (l + r) / 2;
    // [l, mid] [mid + 1, r]
    ret += merge_sort(l, mid);
    ret += merge_sort(mid + 1, r);

    int cur1 = l, cur2 = mid + 1, i = l;
    while(cur1 <= mid && cur2 <= r)
    {
        if(a[cur1] <= a[cur2])
        {
            tmp[i++] = a[cur1++];
        }
        else
        {
            ret += mid - cur1 + 1;
            tmp[i++] = a[cur2++];
```

```
32          }
33      }
34
35      while(cur1 <= mid) tmp[i++] = a[cur1++];
36      while(cur2 <= r) tmp[i++] = a[cur2++];
37
38      for(int j = l; j <= r; j++) a[j] = tmp[j];
39
40      return ret;
41  }
42
43  int main()
44  {
45      cin >> n;
46      for(int i = 1; i <= n; i++) cin >> a[i];
47
48      cout << merge_sort(1, n) << endl;
49
50      return 0;
51  }
```

4. 搭配购买

题目来源： 洛谷

题目链接： P1455 搭配购买

【题目描述】

明天就是母亲节了，电脑组的小朋友们在忙碌的课业之余挖空心思想着该送什么礼物来表达自己的心意呢？听说在某个网站上有卖云朵的，小朋友们决定一同前往去看看这种神奇的商品，这个店里有 n 朵云，云朵已经被老板编号为 $1, 2, 3, .., n$,并且每朵云都有一个价值，但是商店的老板是个很奇怪的人，他会告诉你一些云朵要搭配起来买才卖，也就是说买一朵云则与这朵云有搭配的云都要买，电脑组的你觉得这礼物实在是太新奇了，但是你的钱是有限的，所以你肯定是想用现有的钱买到尽量多价值的云。

【输入描述】

第一行输入三个整数， n, m, w ，表示有 n 朵云， m 个搭配和你现有的钱的数目。

第二行至 $n+1$ 行，每行有两个整数， c_i, d_i 表示第 i 朵云的价钱和价值。

第 $n+2$ 至 $n+1+m$ 行，每行有两个整数 u_i, v_i 。表示买第 u_i 朵云就必须买第 v_i 朵云,同理，如果买第 v_i 朵就必须买第 u_i 朵。

- 对于 30% 的数据，满足 $1 \leq n \leq 100$ ；
- 对于 50% 的数据，满足 $1 \leq n, w \leq 10^3, 1 \leq m \leq 100$ ；
- 对于 100% 的数据，满足 $1 \leq n, w \leq 10^4, 0 \leq m \leq 5 \times 10^3$ 。

【输出描述】

一行，表示可以获得的最大价值。

【示例一】

输入：

```
5 3 10
3 10
3 10
3 10
5 100
10 1
1 3
3 2
4 2
```

输出：

```
1
```

【解法】

解法：01背包 + dfs 求联通块

如果不看搭配购买的话，本质上就是一个 01 背包问题。但是这道题需要把与某一个物品相连的所有物品都选上。这就相当于在图中，把一个联通块内的物品全选上。

因此，可以先用 dfs/bfs/并查集 先将每个联通块合并成大物品。然后在合并之后的每个大物品上来一次 01 背包。

时间复杂度：$O(nm)$ 。

【参考代码】

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

```

```cpp
using namespace std;

const int N = 1e4 + 10;

int n, m, w;
int c[N], d[N];
vector<int> edges[N];

bool st[N]; // 标记 dfs 过程中，哪些点还没有标记过
int cnt;
int cc[N], dd[N];

int f[N];

void dfs(int a)
{
    st[a] = true;
    cc[cnt] += c[a];
    dd[cnt] += d[a];

    for(auto b : edges[a])
    {
        if(!st[b]) dfs(b);
    }
}

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

    for(int i = 1; i <= m; i++)
    {
        int a, b; cin >> a >> b;
        edges[a].push_back(b);
        edges[b].push_back(a);
    }

    for(int i = 1; i <= n; i++)
    {
        if(!st[i])
        {
            cnt++;
            dfs(i);
        }
    }

```

```
51        // 01 背包
52        for(int i = 1; i <= cnt; i++)
53            for(int j = w; j >= cc[i]; j--)
54                f[j] = max(f[j], f[j - cc[i]] + dd[i]);
55
56        cout << f[w] << endl;
57
58        return 0;
59    }
```

Day02

1. 棋盘问题

题目来源： 洛谷

题目链接： P1548[NOIP 1997 普及组] 棋盘问题

【题目描述】

设有一个 $N \times M$ 方格的棋盘($1 \leq N \leq 100, 1 \leq M \leq 100$)

求出改签中包含多少个正方形，多少个长方形 (不包括正方形)

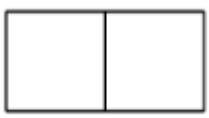
例如： 当 $N = 2, M = 3$ 时：

正方形的个数有 8 个：即边长为 1 的正方形有 6 个；边长为 2 的正方形有 2 个。

长方形的个数有 10 个：

即

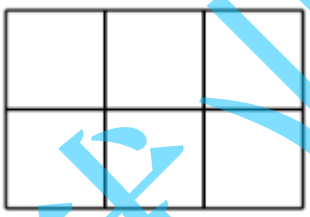
- 2×1 的长方形有 4 个：



- 1×2 的长方形有 3 个：

- 3×1 的长方形有 2 个：

- 3×2 的长方形有 1 个：



【输入描述】

一行两个整数 N, M。

【输出描述】

一行两个整数，表示正方形的个数与长方形的个数。

【示例一】

输入：

2 3

输出：

8 10

【解法】

解法：枚举。

枚举所有的四边形，判断是否是正方形或者长方形。

时间复杂度：$O(n^4)$。

【参考代码】

```cpp
#include <iostream>

using namespace std;

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

    int x = 0, y = 0;
    for(int x1 = 0; x1 <= n; x1++)
        for(int y1 = 0; y1 <= m; y1++)
            for(int x2 = x1 + 1; x2 <= n; x2++)
                for(int y2 = y1 + 1; y2 <= m; y2++)
                {
                    int dx = x2 - x1, dy = y2 - y1;
                    if(dx == dy) x++;
                    else y++;
                }

    cout << x << " " << y << endl;

    return 0;
}
```

2. 混合牛奶 Mixing Milk

题目来源： 洛谷

题目链接： P1208 [USACO1.3] 混合牛奶 Mixing Milk

【题目描述】

由于乳制品产业利润很低，所以降低原材料（牛奶）价格就变得十分重要。帮助 Marry 乳业找到最优的牛奶采购方案。

Marry 乳业从一些奶农手中采购牛奶，并且每一位奶农为乳制品加工企业提供的价格可能相同。此外，就像每头奶牛每天只能挤出固定数量的奶，每位奶农每天能提供的牛奶数量是一定的。每天 Marry 乳业可以从奶农手中采购到小于或者等于奶农最大产量的整数数量的牛奶。

给出 Marry 乳业每天对牛奶的需求量，还有每位奶农提供的牛奶单价和产量。计算采购足够数量的牛奶所需的最小花费。

> 注：每天所有奶农的总产量大于 Marry 乳业的需求量。

【输入描述】

> 第一行二个整数 n, m ，表示需要牛奶的总量，和提供牛奶的农民个数。
>
> 接下来 m 行，每行两个整数 p_i, a_i ，表示第 i 个农民牛奶的单价，和农民 i 一天最多能卖出的牛奶量。
>
> 对于 100% 的数据：
> $0 \leq n, a_i \leq 2 \times 10^6$，$0 \leq m \leq 5000$，$0 \leq p_i \leq 1000$

【输出描述】

> 单独的一行包含单独的一个整数，表示 Marry 的牛奶制造公司拿到所需的牛奶所要的最小费用。

【示例一】

> 输入：
>
> 100 5
> 5 20
> 9 40
> 3 10
> 8 80
> 6 30
>
> 输出：
>
> 630

【解法】

解法：小贪心。

先选择单价较小的牛奶即可。

注意审题：题目给的是牛奶的单价，不是牛奶的总价~

时间复杂度：$O(m \log m)$。

【参考代码】

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

```cpp

using namespace std;

const int N = 5010;

int n, m;
struct node
{
    int p, x;
}a[N];

bool cmp(node& a, node& b)
{
    return a.p < b.p;
}

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= m; i++) cin >> a[i].p >> a[i].x;

    sort(a + 1, a + 1 + m, cmp);

    int ret = 0, sum = 0;
    for(int i = 1; i <= m; i++)
    {
        int t = min(a[i].x, n - sum);
        ret += t * a[i].p;
        sum += t;
    }

    cout << ret << endl;

    return 0;
}
```


3. 开心的金明

> 题目来源： 洛谷
>
> 题目链接： P1060 [NOIP 2006 普及组] 开心的金明

【题目描述】

金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间他自己专用的很宽敞的房间。更让他高兴的是，妈妈昨天对他说：“你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过 N 元钱就行”。今天一早金明就开始做预算，但是他想买的东西太多了，肯定会超过妈妈限定的 N 元。于是，他把每件物品规定了一个重要度，分为 5 等：用整数 1−5 表示，第 5 等最重要。他还从因特网上查到了每件物品的价格（都是整数元）。他希望在不超过 N 元（可以等于 N 元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第 j 件物品的价格为 v_j，重要度为 w_j，共选中了 k 件物品，编号依次为 $j_1, j_2, ..., j_k$，则所求的总和为：

$$v_{j_1} \times w_{j_1} + v_{j_2} \times w_{j_2} ... + w_{j_k} \times w_{j_k}$$

请你帮助金明设计一个满足要求的购物单。

【输入描述】

第一行，为 2 个正整数，用一个空格隔开： $n, m(n < 30000, m < 24)$ 其中 n 表示总钱数， m 为希望购买物品的个数。

从第 2 行到第 $m + 1$ 行，第 j 行给出了编号为 $j - 1$ 的物品的基本数据，每行有 2 个非负整数 v, p（其中 v 表示该物品的价格 $(v \leq 10000)$， p 表示该物品的重要度 $(1 \leq p \leq 5)$。

【输出描述】

1 个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值 $< 100,000,000$。

【示例一】

输入：

```
1000 5
800 2
400 5
300 5
400 3
200 2
```

输出：

```
3900
```

【解法】

解法：动态规划。

简单的 01 背包问题~

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 30010, M = 30;

int n, m;
int v[M], p[M];

int f[N];

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

    for(int i = 1; i <= m; i++)
        for(int j = n; j >= v[i]; j--)
            f[j] = max(f[j], f[j - v[i]] + v[i] * p[i]);

    cout << f[n] << endl;

    return 0;
}
```

4. 借教室

题目来源： 洛谷

题目链接： P1083 [NOIP2012 提高组] 借教室

【题目描述】

在大学期间，经常需要租借教室。大到院系举办活动，小到学习小组自习讨论，都需要向学校申请借教室。教室的大小功能不同，借教室人的身份不同，借教室的手续也不一样。

面对海量租借教室的信息，我们自然希望编程解决这个问题。

我们需要处理接下来 n 天的借教室信息，其中第 i 天学校有 r_i 个教室可供租借。共有 m 份订单，每份订单用三个正整数描述，分别为 d_j, s_j, t_j，表示某租借者需要从第 s_j 天到第 t_j 天租借教室（包括第 s_j 天和第 t_j 天），每天需要租借 d_j 个教室。

我们假定，租借者对教室的大小、地点没有要求。即对于每份订单，我们只需要每天提供 d_j 个教室，而它们具体是哪些教室，每天是否是相同的教室则不用考虑。

借教室的原则是先到先得，也就是说我们要按照订单的先后顺序依次为每份订单分配教室。如果在分配的过程中遇到一份订单无法完全满足，则需要停止教室的分配，通知当前申请人修改订单。这里的

> 无法满足指从第 s_j 天到第 t_j 天中有至少一天剩余的教室数量不足 d_j 个。
>
> 现在我们需要知道，是否会有订单无法完全满足。如果有，需要通知哪一个申请人修改订单。

【输入描述】

> 第一行包含两个正整数 n, m，表示天数和订单的数量。
>
> 第二行包含 n 个正整数，其中第 i 个数为 r_i，表示第 i 天可用于租借的教室数量。
>
> 接下来有 m 行，每行包含三个正整数 d_j, s_j, t_j，表示租借的数量，租借开始、结束分别在第几天。
>
> 每行相邻的两个数之间均用一个空格隔开。天数与订单均用从 1 开始的整数编号。
>
> 对于 100% 的数据，有 $1 \leq n, m \leq 10^6, 0 \leq r_i, d_j \leq 10^9, 1 \leq s_j \leq t_j \leq n$。

【输出描述】

> 如果所有订单均可满足，则输出只有一行，包含一个整数 0。否则（订单无法完全满足）
>
> 输出两行，第一行输出一个负整数 -1，第二行输出需要修改订单的申请人编号。

【示例一】

> 输入：
>
> 4 3
>
> 2 5 4 3
>
> 2 1 3
>
> 3 2 4
>
> 4 2 4
>
> 输出：
>
> -1
>
> 2

【解法】

力破万法：线段树。但是杀鸡焉用宰牛刀，可以用更简单的做法（其实是因为我们还没学.....）。

设无法完成的订单的天数为 ret。针对某一天 x，根据题意可得：

- 当 $x \geq ret$ 时，处理完 $[1, x]$ 天的订单，一定无法完成；
- 当 $x < ret$ 时，处理完 $[1, x]$ 天的订单，一定可以完成。

在解空间中，根据 ret 的位置，可以将解集分成两部分，具有「二段性」，那么我们就可以「二分答案」。

接下来的问题就是给定一个天数 x ，如何判断能否完成 $[1,x]$ 天内的所有订单：

- 利用「差分」数组，具体细节不多说了，很基础的应用；
- 处理完 $[1,x]$ 区间的修改，还原「原数组」之后判断是否「全部 ≥ 0」即可。

时间复杂度：$O(n \log n)$。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 1e6 + 10;

int n, m;
int r[N];
int d[N], s[N], t[N];

int f[N]; // 差分数组

// 把 [1, x] 所有的订单处理完毕之后，判断是否可行
bool check(int x)
{
    // 初始差分数组
    for(int i = 1; i <= n; i++)
    {
        f[i] = r[i] - r[i - 1];
    }

    // 处理订单
    for(int i = 1; i <= x; i++)
    {
        // s[i] ~ t[i]  -d[i]
        f[s[i]] -= d[i];
        f[t[i] + 1] += d[i];
    }

    for(int i = 1; i <= n; i++)
    {
        f[i] = f[i - 1] + f[i];
        if(f[i] < 0) return false;
```

```
        }

        return true;
    }

    int main()
    {
        cin >> n >> m;
        for(int i = 1; i <= n; i++) cin >> r[i];
        for(int i = 1; i <= m; i++) cin >> d[i] >> s[i] >> t[i];

        int l = 1, r = m;
        while(l < r)
        {
            int mid = (l + r) / 2;
            if(check(mid)) l = mid + 1;
            else r = mid;
        }

        if(check(l)) cout << 0 << endl;
        else cout << -1 << endl << l << endl;

        return 0;
    }
```


Day03

1. Phone numbers

> **题目来源：** 洛谷
>
> **题目链接：** CF25B Phone numbers

【题目描述】

> 给一串长度为 n 的数字串，在两个或三个数字之间加 `-`，使得这个数字串便于记忆。如果有多解，可随意输出其中一个。

【示例一】

> 输入：
>
> 6
>
> 549871
>
> 输出：

> 54-98-71

【解法】

解法：分类讨论。

判断一下 n 的奇偶：

- 如果是偶数，直接两个两个隔开；
- 如果是奇数，两个两个隔开，最后一个分三个。

【参考代码】

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n; string s;
    cin >> n >> s;

    if(n % 2) // 奇数
    {
        for(int i = 0; i < n; i++)
        {
            cout << s[i];
            if(i % 2 && i < n - 3) cout << '-';
        }
    }
    else // 偶数
    {
        for(int i = 0; i < n; i++)
        {
            cout << s[i];
            if(i % 2 && i < n - 2) cout << '-';
        }
    }

    return 0;
}
```


2. zzc 种田

> **题目来源：** 洛谷
>
> **题目链接：** P2660 zzc 种田

【题目描述】

> 田地是一个巨大的矩形，然而 zzc 每次只能种一个正方形,而每种一个正方形时 zzc 所花的体力值是正方形的周长，种过的田不可以再种，zzc 很懒还要节约体力去泡妹子，想花最少的体力值去种完这块田地，问最小体力值。

【输入描述】

> 两个正整数 x, y ，表示田地的长和宽。

【输出描述】

> 输出最小体力值。

【示例一】

> 输入：
>
> 1 10
>
> 输出：
>
> 40

【示例二】

> 输入：
>
> 2 2
>
> 输出：
>
> 8

【解法】

解法：数学。

每次都切出边长为 min(x, y) 正方形，切完之后，计算出新的 x 和 y，然后继续切。

【参考代码】

```
#include <iostream>

using namespace std;

typedef long long LL;

int main()
{
```

```
    LL x, y; cin >> x >> y;

    LL ret = 0;
    while(x && y)
    {
        LL cnt = x / y;
        ret += cnt * y * 4;
        x %= y;
        swap(x, y);
    }

    cout << ret << endl;

    return 0;
}
```

3. 信息传递

> **题目来源：** 洛谷
>
> **题目链接：** P2661 [NOIP 2015 提高组] 信息传递

【题目描述】

> 有 n 个同学（编号为 1 到 n）正在玩一个信息传递的游戏。在游戏里每人都有一个固定的信息传递对象，其中，编号为 i 的同学的信息传递对象是编号为 T_i 的同学。
>
> 游戏开始时，每人都只知道自己的生日。之后每一轮中，所有人会同时将自己当前所知的生日信息告诉各自的信息传递对象（注意：可能有人可以从若干人那里获取信息，但是每人只会把信息告诉一个人，即自己的信息传递对象）。当有人从别人口中得知自己的生日时，游戏结束。请问该游戏一共可以进行几轮？

【输入描述】

> 输入共 2 行。
>
> 第一行包含 1 个正整数 n，表示 n 个人。
>
> 第二行包含 n 个用空格隔开的正整数 $T_1, T_2, ..., T_n$，其中第 i 个整数 T_i 表示编号为 i 的同学的信息传递对象是编号为 T_i 的同学，$T_i \leq n$ 且 $T_i \neq i$。

【输出描述】

> 共一行一个整数，表示游戏一共可以进行多少轮。

【示例一】

> 输入：

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

输出：

```
3
```

【解法】

解法：拓扑排序 + dfs计数。

这是一种比较形象直观的解法，当然还有其余解法，感兴趣的同学可以去瞅瞅题解。

1. 拓扑排序的过程可以将除了环以外的结点全部打上标记；
2. 对所有没有打标记的点做一次 dfs，统计一下环的大小。

【参考代码】

```cpp
#include <iostream>
#include <queue>

using namespace std;

const int N = 2e5 + 10;

int n;
int ne[N];
int in[N];

bool st[N];

int cnt;

void dfs(int a)
{
    cnt++;
    st[a] = true;

    int b = ne[a];
    if(!st[b]) dfs(b);
}

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

```
    for(int i = 1; i <= n; i++)
    {
        cin >> ne[i];
        // i -> ne[i]
        in[ne[i]]++;
    }

    // 1. 利用拓扑排序给环以外的点打上标记
    queue<int> q;
    for(int i = 1; i <= n; i++)
    {
        if(in[i] == 0)
            q.push(i);
    }

    while(q.size())
    {
        auto a = q.front(); q.pop();
        st[a] = true;

        int b = ne[a];
        // a -> b
        in[b]--;
        if(in[b] == 0) q.push(b);
    }

    // 2. 利用 dfs 计算环的大小
    int ret = n;
    for(int i = 1; i <= n; i++)
    {
        if(!st[i])
        {
            cnt = 0;
            dfs(i);
            ret = min(ret, cnt);
        }
    }

    cout << ret << endl;

    return 0;
}
```


4. Decrease

题目来源： 洛谷

题目链接： 『MdOI R1』Decrease

【题目描述】

给定一个 $n \times n$ 的矩阵，你可以进行若干次操作。

每次操作，你可以将一个 $k \times k$ 的连续子矩阵里的所有数全都加上 1 或者全都减去 1 。

初始时，矩阵中有 m 个位置上的数不为 0 ，其它位置上的数均为 0 。

请你求出至少需要多少次操作，可以将矩形中所有数都变为 0 。

【输入描述】

第一行三个整数 n, m, k ，分别表示矩阵大小，非 0 格数和每次修改的连续子矩阵大小。

接下来 m 行，每行三个整数 x, y, z ，表示初始时矩阵的第 x 行第 y 列上的数为 z 。

对于所有数据，
$1 \leq n \leq 5 \times 10^3, 1 \leq m \leq min(n^2, 5 \times 10^5), 1 \leq k \leq min(n, 10^3), 1 \leq x, y \leq n$ ，每对 (x, y) 至多出现一次， $1 \leq |z| \leq 10^9$ 。

数据保证如果有解，答案不超过 $2^{63} - 1$ 。

本题读入量较大，建议使用较快的读入方式。

【输出描述】

一行一个整数，表示最少操作次数。

特别地，如果无法使矩阵中所有数都变为 0 ，输出 -1 。

【示例一】

输入：

```
4 14 3
1 1 1
1 2 1
1 3 1
2 1 1
2 2 3
2 3 3
2 4 2
3 1 1
3 2 3
```

3 3 3

3 4 2

4 2 2

4 3 2

4 4 2

输出：

3

【示例二】

输入：

3 1 2

1 1 1

输出：

-1

【示例三】

输入：

4 5 1

1 1 5

2 2 -3

2 3 -4

3 3 1

4 4 2

输出：

15

【解法】

为了方便进行「区间修改」操作，我们创建原矩阵的「差分矩阵」，记为 f 。如果让原矩阵的「所有元素」都变成 0 ，相应的差分矩阵中的「所有元素」也都会变成 0 。问题就转化成了把 f 矩阵变成 0 的最小操作次数。对原矩阵进行「区间」操作对应差分矩阵中「点」的操作，因此在原矩阵中把一个边长为 k 的子矩阵修改成 0 ，对应差分矩阵中「四个点」的修改。

那么我们就可以枚举「差分矩阵」中的左上角 (i,j) ，其中 $1 \leq i,j \leq (n-k+1)$ ，然后进行「区间修改」操作：

- 如果 $f[i][j]$ 不等于 0 ，那我们就把这个区间变成 0 ，操作次数就是 $abs(f[i][j])$ ；
- 如果 $f[i][j]$ 等于 0 ，无需操作。

当把差分矩阵中 $[1, n-k+1]$ 区间内的所有元素「都变成 0 」之后，扫描整个差分矩阵：

- 如果全变成 0 ，输出最小次数；
- 如果存在不等于 0 的位置，说明不能把整个矩阵变成 0 ，输出 -1 。

【参考代码】

```cpp
#include <iostream>
#include <cmath>

using namespace std;

typedef long long LL;

const int N = 5e3 + 10;

int n, m, k;
LL f[N][N]; // 差分

void insert(int x1, int y1, int x2, int y2, int k)
{
    f[x1][y1] += k;
    f[x1][y2 + 1] -= k;
    f[x2 + 1][y1] -= k;
    f[x2 + 1][y2 + 1] += k;
}

int main()
{
    cin >> n >> m >> k;
    for(int i = 1; i <= m; i++)
    {
        int x, y, z; cin >> x >> y >> z;
        insert(x, y, x, y, z);
    }

    // for(int i = 1; i <= n; i++)
    // {
    //     for(int j = 1; j <= n; j++)
    //     {
    //         cout << f[i][j] << " ";
    //     }
    //     cout << endl;
```

```
37      // }
38
39      LL ret = 0;
40      for(int x1 = 1; x1 <= n - k + 1; x1++)
41          for(int y1 = 1; y1 <= n - k + 1; y1++)
42          {
43              ret += abs(f[x1][y1]);
44              int x2 = x1 + k - 1, y2 = y1 + k - 1;
45              insert(x1, y1, x2, y2, -f[x1][y1]);
46          }
47
48      for(int i = 1; i <= n; i++)
49          for(int j = 1; j <= n; j++)
50              if(f[i][j] != 0)
51              {
52                  cout << -1 << endl;
53                  return 0;
54              }
55
56      cout << ret << endl;
57
58      return 0;
59  }
```


Day04

1. 陶陶摘苹果

> **题目来源：** 洛谷
>
> **题目链接：** P1046 [NOIP 2005 普及组] 陶陶摘苹果

【题目描述】

> 陶陶家的院子里有一棵苹果树，每到秋天树上就会结出 10 个苹果。苹果成熟的时候，陶陶就会跑去摘苹果。陶陶有个 30 厘米高的板凳，当她不能直接用手摘到苹果的时候，就会踩到板凳上再试试。
>
> 现在已知 10 个苹果到地面的高度，以及陶陶把手伸直的时候能够达到的最大高度，请帮陶陶算一下她能够摘到的苹果的数目。假设她碰到苹果，苹果就会掉下来。

【输入描述】

> 输入包括两行数据。第一行包含 10 个 100 到 200 之间（包括 100 和 200）的整数（以厘米为单位）分别表示 10 个苹果到地面的高度，两个相邻的整数之间用一个空格隔开。第二行只包括一个 100 到 120 之间（包含 100 和 120）的整数（以厘米为单位），表示陶陶把手伸直的时候能够达到的最大高度。

【输出描述】

> 输出包括一行，这一行只包含一个整数，表示陶陶能够摘到的苹果的数目。

【示例一】

> 输入：
>
> 100 200 150 140 129 134 167 198 200 111
>
> 110
>
> 输出：
>
> 5

【解法】

解法：模拟。

嗯……，希望以后遇到的题都这么可爱。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

int a[15];

int main()
{
    for(int i = 1; i <= 10; i++) cin >> a[i];

    int h; cin >> h; h += 30;

    int ret = 0;
    for(int i = 1; i <= 10; i++)
        if(a[i] <= h)
            ret++;

    cout << ret << endl;

    return 0;
}
```

2. 陶陶摘苹果（升级版）

题目来源： 洛谷

题目链接： P1478 陶陶摘苹果（升级版）

【题目描述】

又是一年秋季时，陶陶家的苹果树结了 n 个果子。陶陶又跑去摘苹果，这次他有一个 r 公分的椅子。当他手够不着时，他会站到椅子上再试试。

这次与 NOIp2005 普及组第一题不同的是：陶陶之前搬凳子，力气只剩下 s 了。当然，每次摘苹果时都要用一定的力气。陶陶想知道在 $s<0$ 之前最多能摘到多少个苹果。

现在已知 n 个苹果到达地上的高度 x_i，椅子的高度 a，陶陶手伸直的最大长度 b，陶陶所剩的力气 s，陶陶摘一个苹果需要的力气 y_i，求陶陶最多能摘到多少个苹果。

【输入描述】

第 1 行：两个数 苹果数 n，力气 s。

第 2 行：两个数 椅子的高度 a，陶陶手伸直的最大长度 b。

第 3 行~第 $3+n-1$ 行：每行两个数 苹果高度 x_i，摘这个苹果需要的力气 y_i。

对于 100% 的数据，$n\le 5000, a\le 50, b\le 200, s\le 1000, x_i\le 280, y_i\le 100$。

【输出描述】

只有一个整数，表示陶陶最多能摘到的苹果数。

【示例一】

输入：

```
8 15
20 130
120 3
150 2
110 7
180 1
50 8
200 0
140 3
120 2
```

输出：

> 4

【解法】

解法：贪心。

把所有能够到的苹果收齐起来，按费力指数排序，从最省力的开始摘。

【参考代码】

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

using namespace std;

const int N = 5010;

int n, s;
int a, b;

int cnt;
int t[N];

int main()
{
    cin >> n >> s;
    cin >> a >> b;
    a += b;

    for(int i = 1; i <= n; i++)
    {
        int x, y; cin >> x >> y;
        if(x <= a) t[++cnt] = y;
    }

    sort(t + 1, t + 1 + cnt);

    int ret = 0, sum = 0;
    for(int i = 1; i <= cnt; i++)
    {
        sum += t[i];
        if(sum <= s) ret++;
    }

    cout << ret << endl;

```

```
37        return 0;
38    }
```

3. Music Notes

> **题目来源：** 洛谷
>
> **题目链接：** [USACO09DEC] Music Notes S

【题目描述】

> 约翰准备教他的奶牛们弹一首歌。这首歌由 N 个音阶组成，第 i 个音阶要敲击 b_i 次。奶牛从第 0 时刻开始弹，因此他从 0 时刻到 b_{i-1} 时刻都是敲第 1 个音阶，然后他从 b_1 时刻到 b_1+b_2-1 时刻敲第 2 个音阶，从 b_1+b_2 到 $b_1+b_2+b_3-1$ 时刻敲第 3 个音阶……现在有 Q 个问题：在时间段区间 $t,t+1$ 内，奶牛敲的是哪个音阶？
>
> $1 \le N \le 50000, 1 \le Q \le 50000, 1 \le b_i \le 10000$

【输入描述】

> 第一行： N 和 Q
>
> 第 2 ~ N + 1 行： $b_1, b_2 ..., b_n$
>
> 第 N + 2 ~ N + Q + 1 行：Q 次询问： t_i

【输出描述】

> 共 Q 行，每行一个整数，表示奶牛在 t_i 时刻正在弹奏的音阶

【示例一】

> 输入：
>
> ```
> 3 5
> 2
> 1
> 3
> 2
> 3
> 4
> 0
> 1
> ```
>
> 输出：

```
2
3
3
1
1
```

【解法】

解法：前缀和 + 二分。

- 对所有音阶的敲击次数做一次「前缀和」，那么前缀和数组中每一个位置 $f[i]$ 表示：第 i 个音符「结束的时刻」。
- 那么对于每一次询问 t ，我们在前缀和数组中二分出「$>t$ 的第一个位置」的下标，就是结果。

注意是 $>t$，不是 $\geq t$，因为等于 t 的位置这个音阶已经结束了。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 5e4 + 10;

int n, q;
int f[N];

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

    for(int i = 1; i <= n; i++)
    {
        int x; cin >> x;
        f[i] = f[i - 1] + x;
    }

    while(q--)
    {
        int t; cin >> t;
        int l = 1, r = n;
        while(l < r)
        {
            int mid = (l + r) / 2;
```

```
27                 if(f[mid] > t) r = mid;
28                 else l = mid + 1;
29             }
30             cout << l << endl;
31         }
32
33         return 0;
34     }
```

4. 字串变换

> **题目来源：** 洛谷
>
> **题目链接：** P1032 [NOIP2002 提高组] 字串变换

【题目描述】

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），形如：

- $A1 \to B1$。
- $A2 \to B2$。

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 $B_2 \cdots$。

例如：$A = abcd$，$B = xyz$，

变换规则为：

- $abc \to xu$，$ud \to y$，$y \to yz$。

则此时，A 可以经过一系列的变换变为 B，其变换的过程为：

- $abcd \to xud \to xy \to xyz$。

共进行了 3 次变换，使得 A 变换为 B。

【输入描述】

第一行有两个字符串 A, B。

接下来若干行，每行有两个字符串 A_i, B_i，表示一条变换规则。

【输出描述】

若在 10 步（包含 10 步）以内能将 A 变换为 B，则输出最少的变换步数；否则输出 `NO ANSWER!`。

【示例一】

输入：

```
abcd xyz
```

> abc xu
>
> ud y
>
> y yz
>
> 输出：
>
> 3

【解法】

解法：bfs解决最短路问题。

一个字符串如果能变成另一个字符串，变化的代价是 1 。所以我们可以把所有字符串当成一个点，字符串与字符串之间相当于有一条边长为 1 的边，然后跑一遍 bfs 即可。

这道题的难点在于：

- 如何记录最短路？可以用 $unordered\_map<string, int>$ 来记录最短路；
- 如何判断字符串能够变换，以及变换成什么字符串？
  - $string$ 里面的 $find$ 能够找出可以变换之后的位置；
  - 然后用 $substr$ 完成字符串的拼接操作。

  但是要注意，一个字符串可能有多个位置都能转换，所以要多次 $find$。

【参考代码】

```
#include <iostream>
#include <queue>
#include <string>
#include <unordered_map>

using namespace std;

string a, b;
// 用哈希表来记录最短路
unordered_map<string, int> dist;
string x[10], y[10];
int cnt;

int bfs()
{
    if(a == b) return 0;
    queue<string> q;
```

```cpp
    q.push(a);
    dist[a] = 0;

    while(q.size())
    {
        string t = q.front(); q.pop();

        if(t == b) return dist[t];
        if(dist[t] >= 10) continue; // 超过 10 步，就不用搜了

        for(int i = 0; i < cnt; i++)
        {
            int pos = 0;
            // 搜索所有能变的位置
            while(t.find(x[i], pos) != -1)
            {
                pos = t.find(x[i], pos);
                // 记录变换后的字符串
                string tmp = t.substr(0, pos) + y[i] + t.substr(pos + x[i].size());
                pos++; // 别忘了++，寻找下一个位置

                if(dist.count(tmp)) continue;
                dist[tmp] = dist[t] + 1;
                q.push(tmp);
            }
        }
    }
    return -1;
}

int main()
{
    cin >> a >> b;
    // 一边读数据，一边记录数据个数
    while(cin >> x[cnt] >> y[cnt]) cnt++;

    int ret = bfs();

    if(ret != -1) cout << ret << endl;
    else cout << "NO ANSWER!" << endl;

    return 0;
}
```

Day05

1. 优秀的拆分

> **题目来源：** 洛谷
>
> **题目链接：** P7071[CSP-J2020] 优秀的拆分

【题目描述】

> 一般来说，一个正整数可以拆分成若干个正整数的和。
>
> 例如，$1=1$，$10=1+2+3+4$ 等。对于正整数 n 的一种特定拆分，我们称它为“优秀的”，当且仅当在这种拆分下，n 被分解为了若干个**不同**的 2 的**正整数**次幂。注意，一个数 x 能被表示成 2 的正整数次幂，当且仅当 x 能通过正整数个 2 相乘在一起得到。
>
> 例如，$10=8+2=2^3+2^1$ 是一个优秀的拆分。但是，$7=4+2+1=2^2+2^1+2^0$ 就不是一个优秀的拆分，因为 1 不是 2 的正整数次幂。
>
> 现在，给定正整数 n，你需要判断这个数的所有拆分中，是否存在优秀的拆分。若存在，请你给出具体的拆分方案。

【输入描述】

> 输入只有一行，一个整数 n，代表需要判断的数。

【输出描述】

> 如果这个数的所有拆分中，存在优秀的拆分。那么，你需要从大到小输出这个拆分中的每一个数，相邻两个数之间用一个空格隔开。可以证明，在规定了拆分数字的顺序后，该拆分方案是唯一的。
>
> 若不存在优秀的拆分，输出 `-1`。

【示例一】

> 输入：
>
> ```
> 6
> ```
>
> 输出：
>
> ```
> 4 2
> ```
>
> 说明：
>
> $6=4+2=2^2+2^1$ 是一个优秀的拆分。注意，$6=2+2+2$ 不是一个优秀的拆分，因为拆分成的 3 个数不满足每个数互不相同。

【示例二】

> 输入：
>
> ```
> 7
> ```
>
> 输出：

```
-1
```

【解法】

解法：数的二进制表示。

二进制表示中为 1，就加上对应的权重。如果最后一位是 1，则输出 -1。

【参考代码】

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n; cin >> n;
    if(n & 1)
    {
        cout << -1 << endl;
        return 0;
    }

    for(int i = 30; i >= 0; i--)
    {
        if((n >> i) & 1)
        {
            cout << (1 << i) << " ";
        }
    }

    return 0;
}
```


2. 删数问题

题目来源： 洛谷

题目链接： P1106 删数问题

【题目描述】

键盘输入一个高精度的正整数 n （不超过 250 位），去掉其中任意 k 个数字后剩下的数字按原左右次序将组成一个新的非负整数。编程对给定的 n 和 k ，寻找一种方案使得剩下的数字组成的新数最小。

【输入描述】

> 输入两行正整数。
>
> 第一行输入一个高精度的正整数 n 。
>
> 第二行输入一个正整数 k ，表示需要删除的数字个数。
>
> 用 $len(n)$ 表示 n 的**位数**，保证 $1 \leq k < len(n) \leq 250$ 。

【输出描述】

> 输出一个整数，最后剩下的最小数。

【示例一】

> 输入：
>
> 175438
>
> 4
>
> 输出：
>
> 13

【示例二】

> 输入：
>
> 输出：

【示例三】

> 输入：
>
> 输出：

【解法】

解法：贪心。

循环 k 次：

- 每次从高位开始遍历整个数位，如果当前位的数字大于下一位，删除当前位。

注意处理前导 0。

【参考代码】

```cpp
#include <iostream>

using namespace std;

int main()
{
    string s; int k; cin >> s >> k;

    for(int i = 1; i <= k; i++)
    {
        bool flag = false;
        for(int j = 0; j < s.size() - 1; j++)
        {
            if(s[j] > s[j + 1])
            {
                s.erase(j, 1);
                flag = true;
                break;
            }
        }

        if(!flag) s.pop_back();
    }

    while(s.size() > 1 && s[0] == '0') s.erase(0, 1);

    cout << s << endl;

    return 0;
}
```


3. 集合 Subset Sums

> **题目来源：** 洛谷
>
> **题目链接：** P1466 [USACO2.2] 集合 Subset Sums

【题目描述】

> 对于从 $1 \sim n$ 的连续整数集合，能划分成两个子集合，且保证每个集合的数字和是相等的。举个例子，如果 $n = 3$，对于 $\{1, 2, 3\}$ 能划分成两个子集合，每个子集合的所有数字和是相等的：
>
> $\{3\}$ 和 $\{1, 2\}$ 是唯一一种分法（交换集合位置被认为是同一种划分方案，因此不会增加划分方案总数）
>
> 如果 $n = 7$，有四种方法能划分集合 $\{1, 2, 3, 4, 5, 6, 7\}$，每一种分法的子集合各数字和是相等的:

> $\{1,6,7\}$ 和 $\{2,3,4,5\}$
>
> $\{2,5,7\}$ 和 $\{1,3,4,6\}$
>
> $\{3,4,7\}$ 和 $\{1,2,5,6\}$
>
> $\{1,2,4,7\}$ 和 $\{3,5,6\}$
>
> 给出 n ，你的程序应该输出划分方案总数。
>
> 对于 100% 的数据， $1 \le n \le 39$ 。

【输入描述】

> 输入文件只有一行，且只有一个整数 n

【输出描述】

> 输出划分方案总数。

【示例一】

> 输入：
>
> 7
>
> 输出：
>
> 4

【解法】

01 背包问题求方案数的变形。

问题转化：

- 这道题本质上是让我们从 $[1,n]$ 中挑一些数出来，凑成总和的一半，并且每一个数在一次挑选中只能被选一次，正好是 01 背包问题求方案数。
- 其中每一个数的大小就是体积，总和就是价值。

但是在 dp 之前要先判断一下是否有解，因为总和可能是奇数，没有一半。

1. 状态表示：

   $dp[i][j]$ 表示：从 $[1,i]$ 中挑选，总和正好是 j 的方案数。

   那么 $dp[n][m] / 2$ 就是我们要的结果。一定要注意除以 2 ，因为我们统计的方案数是结果的两倍。其中 m 是总和的一半。

2. 状态转移方程：

   根据第 i 个数选或者不选，分成两种情况讨论：

a. 不选 i 这个数：那就是去 $[1, j-1]$ 里面凑 j ，此时的方案数是 $dp[i-1][j]$ ；

b. 选 i 这个数：那就要去 $[1, j-1]$ 里面凑 $j-i$ ，此时的方案数是 $dp[i-1][j-i]$ ；

把两者加起来就是 $dp[i]$ 。但是要注意第二个状态需要判断是否存在。

3. 初始化：

$dp[0][0] = 1$ ，这是一个合法的状态，也是为了让后面的填表是正确的。

4. 填表顺序：

从上往下每一行，每一行从左往右。

空间优化版本：每一行从右往左。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

typedef long long LL;
const int N = 45, M = 410;

int n;
LL f[M];

int main()
{
    cin >> n;
    int m = (1 + n) * n / 2;

    if(m & 1)
    {
        cout << 0 << endl;
        return 0;
    }
    m /= 2;

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

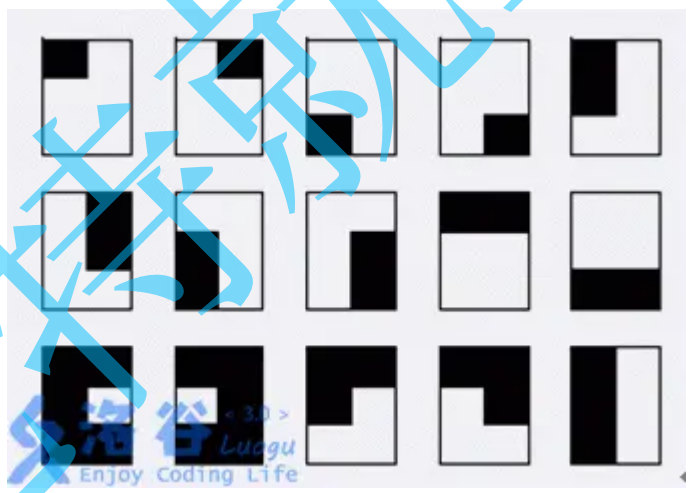
```
27          {
28              f[j] += f[j - i];
29          }
30      }
31      cout << f[m] / 2 << endl;
32
33      return 0;
34  }
```

4. 矩形

> **题目来源：** 洛谷
>
> **题目链接：** P4537 [CQOI2007] 矩形

【题目描述】

> 给一个 $a \times b$ 矩形，由 $a \times b$ 个单位正方形组成。你需要沿着网格线把它分成非空的两部分，每部分所有格子连通，且至少有一个格子在原矩形的边界上。“连通”是指任两个格子都可以通过水平或者竖直路径连在一起。 求方案总数。例如 3×2 的矩形有 15 种方案。



【输入描述】

> 输入仅一行，为两个整数 a, b， $1 \leq a \leq 6, 2 \leq b \leq 7$

【输出描述】

> 输出仅一行，即方案总数。

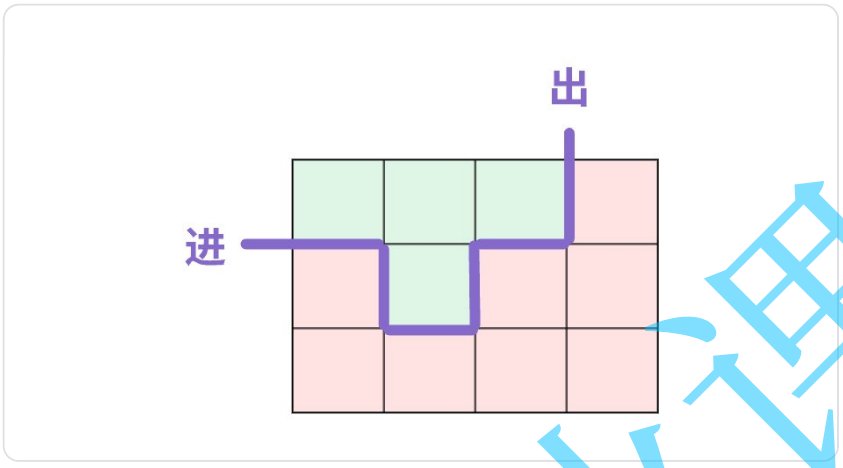
【示例一】

> 输入：
>
> 3 2
>
> 输出：

15

【解法】

分割的过程就像是从矩阵的四边，从不是四角的点中选择一个点进入，然后在矩阵里面随意走，直到走到一个点出去。所以总的分割方案应该和走法是一一对应的。



因此，原始的问题就转化成：在矩阵的四边选择一个点进入，然后走到矩阵边界的总方案数。

总方案数可能由如下四种情况组成：

- 从「左」边界进入，然后出去的总方案数，记作 x_1；
- 从「右」边界进入，然后出去的总方案数，记作 x_2；
- 从「上」边界进入，然后出去的总方案数，记作 y_1；
- 从「下」边界进入，然后出去的总方案数，记作 y_2。

根据对称性可知：

- $x_1 = x_2$，$y_1 = y_2$，所以我们只用计算上边界和左边界两种情况，然后再乘以 2 就是四种情况之和；
- 左进上出与上进左出，是同一种方案（同理还有左和下，左和右.....）。所以 $x_1 + x_2 + y_1 + y_2$ 会把所有情况都计算两遍，我们应该取总和的一半。

综上所述，我们只用算「一次上边界」的情况和「一次左边界」的情况即可。

搜索时候的细节问题：

- 递归的时候，我们判断走到边界的时候算一种方案。如果开始搜索的时候从 $[1, 0]$ 开始，那么刚开始就会统计上一种方案。为了避免这种情况，我们从 $[1, 1]$ 的位置开始。
- 上下左右四个方向走的时候，进入递归之前不用判断是否越界。因为递归出口就是走到边界上就返回，不会发生越界访问的情况。
- 搜索一个分支结束之后要恢复现场。大家可以和《奶酪》对比一下，思考为啥《奶酪》不需要恢复现场，而这道题必须要恢复现场。

【参考代码】

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

using namespace std;

int n, m;
int ret;
bool st[10][10];

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

void dfs(int a, int b)
{
    if(a < 1 || a >= n || b < 1 || b >= m)
    {
        ret++;
        return;
    }
    for(int i = 0; i < 4; i++)
    {
        int x = a + dx[i], y = b + dy[i];
        // 不用判断越界，我们要的就是走到边上
        if(!st[x][y])
        {
            st[x][y] = true;
            dfs(x, y);
            st[x][y] = false;
        }
    }
}

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

    // 统计从左边进入一共有多少种出法
    for(int i = 1; i < n; i++)
    {
        st[i][0] = st[i][1] = true;
        dfs(i, 1); // 从 [i, 1] 开始搜，避免出入口一样的情况
        st[i][0] = st[i][1] = false;
    }

```

```cpp
        // 统计从上边进入一共有多少种出法
        for(int j = 1; j < m; j++)
        {
            st[0][j] = st[1][j] = true;
            dfs(1, j);
            st[0][j] = st[1][j] = false;
        }

        cout << ret << endl;

        return 0;
    }
```