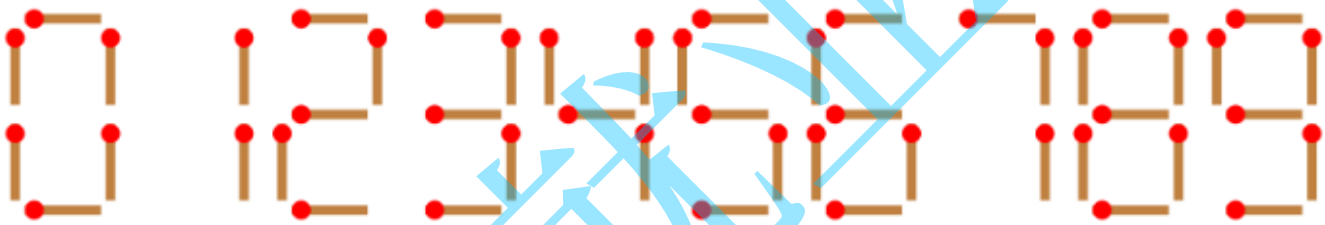
第 3 周

Day11

1. 火柴棒等式

> **题目来源：** 洛谷
>
> **题目链接：** P1149[NOIP 2008 提高组] 火柴棒等式

【题目描述】

> 给你 n 根火柴棍，你可以拼出多少个形如 $A+B=C$ 的等式？等式中的 A,B,C 是用火柴棍拼出的整数（若该数非零，则最高位不能是 0）。用火柴棍拼数字 0~9 的拼法如图所示：
>
> 
>
> 注意：
>
> 1. 加号与等号各自需要两根火柴棍；
> 2. 如果 $A \neq B$，则 $A+B=C$ 与 $B+A=C$ 视为不同的等式（ $A,B,C \geq 0$ ）；
> 3. n 根火柴棍必须全部用上。

【输入描述】

> 一个整数 n ($1 \leq n \leq 24$)。

【输出描述】

> 一个整数，能拼成的不同等式的数目。

【示例一】

> 输入：
>
> 14
>
> 输出：
>
> 2
>
> 说明：

比特就业课

> 2 个等式为
>
> $0+1=1$ 和 $1+0=1$。

【示例二】

> 输入：
>
> 18
>
> 输出：
>
> 9
>
> 说明：
>
> 9 个等式为
>
> $0+4=4$
>
> $0+11=11$
>
> $1+10=11$
>
> $2+2=4$
>
> $2+7=9$
>
> $4+0=4$
>
> $7+2=9$
>
> $10+1=11$
>
> $11+0=11$

【解法】

解法：暴力枚举。

n 的最大值为 24，花费较小的火柴的等式 1111 + 1 = 1112 是 25 根。因此仅需枚举所有的三位数 + 三位数的算式，判断所用的火柴是否等于 n。

【参考代码】

```
#include <iostream>

using namespace std;

int cnt[] = {6, 2, 5, 5, 4, 5, 6, 3, 7, 6};

int calc(int x)
```

比特就业课

```cpp
{
    if(x == 0) return 6;

    int sum = 0;
    while(x)
    {
        sum += cnt[x % 10];
        x /= 10;
    }

    return sum;
}

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

    int ret = 0;
    for(int a = 0; a < 1000; a++)
    {
        for(int b = 0; b < 1000; b++)
        {
            int c = a + b;

            if(calc(a) + calc(b) + calc(c) + 4 == n)
            {
                ret++;
            }
        }
    }

    cout << ret << endl;

    return 0;
}
```

比特就业课

2. 导弹拦截

> 题目来源： 洛谷
>
> 题目链接： P1158 [NOIP 2010 普及组] 导弹拦截

【题目描述】

经过 11 年的韬光养晦，某国研发出了一种新的导弹拦截系统，凡是与它的距离不超过其工作半径的导弹都能够被它成功拦截。当工作半径为 0 时，则能够拦截与它位置恰好相同的导弹。但该导弹拦截系统也存在这样的缺陷：每套系统每天只能设定一次工作半径。而当天的使用代价，就是所有系统工作半径的平方和。

某天，雷达捕捉到敌国的导弹来袭。由于该系统尚处于试验阶段，所以只有两套系统投入工作。如果现在的要求是拦截所有的导弹，请计算这一天的最小使用代价。

【输入描述】

第一行包含 4 个整数 x_1, y_1, x_2, y_2 ，每两个整数之间用一个空格隔开，表示这两套导弹拦截系统的坐标分别为 $(x_1, y_1)(x_2, y_2)$。第二行包含 1 个整数 N ，表示有 N 颗导弹。接下来 N 行，每行两个整数 x, y ，中间用一个空格隔开，表示一颗导弹的坐标 (x, y)。不同导弹的坐标可能相同。

【输出描述】

一个整数，即当天的最小使用代价。

【示例一】

输入：

```
0 0 10 0
2
-3 3
10 0
```

输出：

```
18
```

【示例二】

输入：

```
0 0 6 0
5
-4 -2
-2 3
4 0
6 -2
9 1
```

输出：

```
30
```

【解法】

比特就业课

解法：枚举。

- 枚举第一个圆的半径；
- 在没有包裹的点中，找出距离第二个点的最大半径。

在枚举的过程中，求出半径之和的最小值即可。

快速枚举的方式：

- 预处理出所有点到第一个圆心的距离，并排序；
- 从大到小枚举所有的距离，这样每次只会额外出来一个点。因此就可以用这个点快速更新出第二个圆心的最大距离。

【参考代码】

代码块

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

using namespace std;

const int N = 1e5 + 10;

int x1, y1, x2, y2;
int n;
struct node
{
    int x, y, d;
    // d 表示当前这个点距离第一个导弹拦截系统的距离
}a[N];

int calc(int i, int x, int y)
{
    int dx = x - a[i].x;
    int dy = y - a[i].y;
    return dx * dx + dy * dy;
}

bool cmp(node& a, node& b)
{
    return a.d > b.d;
}

int main()
```

比特就业课

```
29  {
30      cin >> x1 >> y1 >> x2 >> y2;
31      cin >> n;
32      for(int i = 1; i <= n; i++)
33      {
34          cin >> a[i].x >> a[i].y;
35          a[i].d = calc(i, x1, y1);
36      }
37
38      sort(a + 1, a + 1 + n, cmp);
39
40      int ret = a[1].d;
41      int r2 = 0;
42      for(int i = 2; i <= n + 1; i++) // 循环到 n + 1
43      {
44          int r1 = a[i].d;
45          r2 = max(r2, calc(i - 1, x2, y2));
46
47          ret = min(ret, r1 + r2);
48      }
49
50      cout << ret << endl;
51
52      return 0;
53  }
```

3. 铺设道路

> **题目来源：** 洛谷
>
> **题目链接：** P5019 [NOIP 2018 提高组] 铺设道路

【题目描述】

> 春春是一名道路工程师，负责铺设一条长度为 n 的道路。
>
> 铺设道路的主要工作是填平下陷的地表。整段道路可以看作是 n 块首尾相连的区域，一开始，第 i 块区域下陷的深度为 d_i 。
>
> 春春每天可以选择一段连续区间 $[L, R]$ ，填充这段区间中的每块区域，让其下陷深度减少 1。在选择区间时，需要保证，区间内的每块区域在填充前下陷深度均不为 0 。
>
> 春春希望你能帮他设计一种方案，可以在最短的时间内将整段道路的下陷深度都变为 0 。

【输入描述】

比特就业课

> 输入文件包含两行，第一行包含一个整数 n ，表示道路的长度。 第二行包含 n 个整数，相邻两数间用一个空格隔开，第 i 个整数为 d_i 。
>
> 对于 30% 的数据， $1 \leq n \leq 10$;
> 对于 70% 的数据， $1 \leq n \leq 1000$;
> 对于 100% 的数据， $1 \leq n \leq 100000$, $0 \leq d_i \leq 10000$ 。

【输出描述】

> 输出文件仅包含一个整数，即最少需要多少天才能完成任务。

【示例一】

> 输入：
>
> 6
>
> 4 3 2 5 3 5
>
> 输出：
>
> 9
>
> 说明：
>
> 一种可行的最佳方案是，依次选择：[1,6]、[1,6]、[1,2]、[1,1]、[4,6]、[4,4]、[4,4]、[6,6]、[6,6]。

【解法】

解法：贪心。

从前往后遍历所有的坑，对于第 i 个坑：

- 如果 `a[i - 1] < a[i]` ：那么仅需填上 `a[i] - a[i - 1]` 即可，因为 `a[i - 1]` 深度的部分会在处理 `i - 1` 的时候顺手填上；
- 如果 `a[i - 1] >= a[i]` ：无需填补。因此在填 `i - 1` 的时候，顺手就填上了。

【参考代码】

代码块

```
#include <iostream>

using namespace std;

typedef long long LL;

```

比特就业课

```cpp
const int N = 1e5 + 10;

int n;
int a[N];

int main()
{
    cin >> n;
    LL ret = 0;
    for(int i = 1; i <= n; i++)
    {
        cin >> a[i];
        if(a[i] > a[i - 1])
        {
            ret += a[i] - a[i - 1];
        }
    }

    cout << ret << endl;

    return 0;
}
```

4. Shaass and Bookshelf

题目来源： 洛谷

题目链接： Shaass and Bookshelf

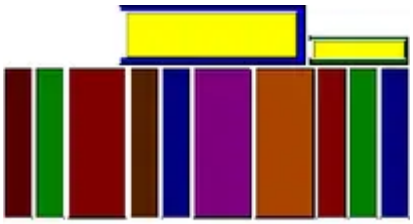
【题目描述】

Shaass 拥有 n 本书。他想为他的所有书制作一个书架，并想让书架的长宽尽量小。第 i 本书的厚度是 $t[i]$，且这本书的纸张宽度是 $w[i]$。书的厚度是 1 或 2，所有书都有同样的高度（即书架的高是均匀的）。

Shaass 以以下的方式摆放这些书籍。

比特就业课

1. 他选择了一些书并竖直摆放它们。
2. 他将剩余的书籍水平纺织于竖直的书上面。 水平放置的书的宽度和不能多于竖直放置的书的总厚度。图中描绘了书籍的样本排列。



帮助 Shaass 找到可以达到的书架长度最小值。

【输入描述】

输入的第一行包含一个 int 型的整数 n $(1 \leq n \leq 100)$ 。 下面的n行分别为 $v[i]$ 和 $w[i]$，对应了书的长度和宽度（即书籍竖直放置与水平放置所占的空间）。 $(1 \leq v[i] \leq 2, 1 \leq w[i] \leq 100)$

【输出描述】

一个整数，为可以达到的最小的长度。

【示例一】

输入：

```
5
1 12
1 3
2 15
2 5
2 1
```

输出：

```
5
```

【示例二】

输入：

```
3
1 10
2 1
2 4
```

输出：

```
3
```

比特就业课

【解法】

解法：动态规划 - 背包问题。

如果在选择每一本书摆放位置的时候，既考虑上面的宽度，又考虑下面厚度，解决起来非常麻烦。但是，如果只盯着水平放置的书籍，当它们确定下来之后，下面的书籍的厚度是可以用总厚度 sum 减去上面书籍的厚度 i 直接计算出来。

这样，我们只用考虑上面书籍的所有厚度下的最小宽度，然后计算对应下面书籍的厚度是否合法即可。计算上面书籍所有可能的厚度下的最小宽度，正好对应 01 背包问题。

1. 状态表示：

   $dp[i][j]$ 表示：从前 i 个书本中挑选，水平放置的时候，总厚度恰好为 j 时，此时的最小宽度。

   最终结果就是所有符合 $sum - i \geq dp[i]$ 里面， $sum - i$ 的最小值。

2. 状态转移方程：

   对于 i 位置的书籍，有选或不选两种情况：

   a. 如果不选：那就是在 $[1, i - 1]$ 区间内，去凑厚度恰好为 j 时的最小宽度 $dp[i - 1][j]$；

   b. 如果选上 i 位置书籍：那就是在 $[i, i - 1]$ 区间内，去凑厚度恰好为 $j - v[i]$ 时的最小宽度，然后再加上 i 位置宽度 $dp[i - 1][j - v[i]] + w[i]$；

   因为要的是最小值，所以取上面两种情况的最小值。其中第二种情况注意判断是否存在。

3. 初始化：

   可以把 dp 表初始化为正无穷 $0x3f3f3f3f$，然后把 $dp[0][0]$ 初始化为 0。

   - 不合法的状态初始化为无穷大，取最小值的时候就不会影响结果；
   - $dp[0][0]$ 是一个合法的位置，也是为了让后续填表是正确的。

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

比特就业课

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

using namespace std;

const int N = 110, M = 10010;

int n;
int t[N], w[N];
int f[M];

int main()
{
    cin >> n;
    int sum = 0;
    for(int i = 1; i <= n; i++)
    {
        cin >> t[i] >> w[i];
        sum += t[i];
    }

    memset(f, 0x3f, sizeof f);
    f[0] = 0;
    for(int i = 1; i <= n; i++)
    {
        for(int j = sum; j >= t[i]; j--)
        {
            f[j] = min(f[j], f[j - t[i]] + w[i]);
        }
    }

    for(int j = sum; j >= 0; j--)
    {
        if(f[j] <= sum - j)
        {
            cout << sum - j << endl;
            break;
        }
    }

    return 0;
}
```

比特就业课

Day12

1. 统计单词数

题目来源： 洛谷

题目链接： P1308 [NOIP 2011 普及组] 统计单词数

【题目描述】

一般的文本编辑器都有查找单词的功能，该功能可以快速定位特定单词在文章中的位置，有的还能统计出特定单词在文章中出现的次数。

现在，请你编程实现这一功能，具体要求是：给定一个单词，请你输出它在给定的文章中出现的次数和第一次出现的位置。注意：匹配单词时，不区分大小写，但要求完全匹配，即给定单词必须与文章中的某一独立单词在不区分大小写的情况下完全相同（参见样例 1），如果给定单词仅是文章中某一单词的一部分则不算匹配（参见样例 2）。

【输入描述】

共 2 行。

第 1 行为一个字符串，其中只含字母，表示给定单词；

第 2 行为一个字符串，其中只可能包含字母和空格，表示给定的文章。

数据范围

$1 \leq$ 第一行单词长度 ≤ 10。

$1 \leq$ 文章长度 $\leq 10^6$ 。

【输出描述】

一行，如果在文章中找到给定单词则输出两个整数，两个整数之间用一个空格隔开，分别是单词在文章中出现的次数和第一次出现的位置（即在文章中第一次出现时，单词首字母在文章中的位置，位置从 0 开始）；如果单词在文章中没有出现，则直接输出一个整数 -1。

【示例一】

输入：

```
To
to be or not to be is a question
```

输出：

```
2 0
```

【示例二】

输入：

```
to
Did the Ottoman Empire lose its power at that time
```

比特就业课

> 输出：
>
> -1

【解法】

解法：模拟 + 字符串函数。

- 将所有字符都转成小写，然后利用 find 函数快速找出单词所在位置。

注意：文章中有的单词内部也会出现目标单词，可以在目标单词以及文章的前后加上一个空格，再去查找。

【参考代码】

```
#include <iostream>

using namespace std;

int main()
{
    string s, t;
    cin >> s;
    getchar();
    getline(cin, t);

    s = ' ' + s + ' ';
    t = ' ' + t + ' ';

    // 全部大写换小写
    for(int i = 0; i < s.size(); i++)
    {
        s[i] = tolower(s[i]);
    }

    for(int i = 0; i < t.size(); i++)
    {
        t[i] = tolower(t[i]);
    }

    int cnt = 0, f = 0;
    if(t.find(s) == -1)
    {
        cout << -1 << endl;
    }
    else
```

比特就业课

```
32      {
33          f = t.find(s);
34          int pos = 0;
35          while(t.find(s, pos) != -1)
36          {
37              pos = t.find(s, pos);
38              cnt++;
39
40              pos++;
41          }
42
43          cout << cnt << " " << f << endl;
44      }
45
46      return 0;
47  }
```

2. 连续自然和

> **题目来源：** 洛谷
>
> **题目链接：** P1147 连续自然数和

【题目描述】

> 对一个给定的正整数 *M*，求出所有的连续的正整数段（每一段至少有两个数），这些连续的自然数段中的全部数之和为 *M*。
>
> 例子：1998+1999+2000+2001+2002=10000，所以从 1998 到 2002 的一个自然数段为 *M*=10000 的一个解。

【输入描述】

> 包含一个整数的单独一行给出 *M* 的值（10≤*M*≤2,000,000）。

【输出描述】

> 每行两个正整数，给出一个满足条件的连续正整数段中的第一个数和最后一个数，两数之间用一个空格隔开，所有输出行的第一个按从小到大的升序排列，对于给定的输入数据，保证至少有一个解。

【示例一】

> 输入：
>
> 10000
>
> 输出：
>
> 18 142

比特就业课

> 297 328
>
> 388 412
>
> 1998 2002

【解法】

解法：前缀和 + 哈希表。

从前往后遍历每一个数，对于当前的前缀和 `sum[i]`，仅需在前面找到前缀和为 `sum[i] - m` 的位置 `p`。如果存在，则 `[p + 1, i]` 就是要找的一个区间。

可以利用哈希表维护 <前缀和, 下标> 信息。

【参考代码】

代码块

```cpp
#include <iostream>
#include <unordered_map>

using namespace std;

typedef long long LL;

int m;
unordered_map<LL, int> mp;

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

    mp[0] = 0;
    LL sum = 0;
    for(int i = 1; i <= n; i++)
    {
        sum += i;

        // sum - m
        if(mp.count(sum - m))
        {
            cout << mp[sum - m] + 1 << " " << i << endl;
        }

```

比特就业课

```
28            mp[sum] = i;
29        }
30
31        return 0;
32    }
```

3. Meteor Shower

题目来源： 洛谷

题目链接： P2895 [USACO08FEB] Meteor Shower S

【题目描述】

贝茜听说一场特别的流星雨即将到来：这些流星会撞向地球，并摧毁它们所撞击的任何东西。她为自己的安全感到焦虑，发誓要找到一个安全的地方（一个永远不会被流星摧毁的地方）。

如果将牧场放入一个直角坐标系中，贝茜现在的位置是原点，并且，贝茜不能踏上一块被流星砸过的土地。

一共有 M 颗流星 $(1 \leq M \leq 50,000)$ 会坠落在农场上，其中第 i 颗流星会在时刻 T_i $(0 \leq Ti \leq 1000)$ 砸在坐标为 $(Xi, Yi)(0 \leq Xi \leq 300, 0 \leq Yi \leq 300)$ 的格子里。流星的力量会将它所在的格子，以及周围 4 个相邻的格子都化为焦土，当然贝茜也无法再在这些格子上行走。

贝茜在时刻 0 开始行动，她只能在会在横纵坐标 $X, Y \geq 0$ 的区域中，平行于坐标轴行动，每 1 个时刻中，她能移动到相邻的（一般是 4 个）格子中的任意一个，当然目标格子要没有被烧焦才行。如果一个格子在时刻 t 被流星撞击或烧焦，那么贝茜只能在 t 之前的时刻在这个格子里出现。 贝茜一开始在 $(0, 0)$。

请你计算一下，贝茜最少需要多少时间才能到达一个安全的格子。如果不可能到达输出 -1。

【输入描述】

共 $M+1$ 行，第 1 行输入一个整数 M，接下来的 M 行每行输入三个整数分别为 X_i, Y_i, T_i。

【输出描述】

贝茜到达安全地点所需的最短时间，如果不可能，则为 -1。

【示例一】

输入：

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

比特就业课

> 0 3 5
>
> 输出：
>
> 5

【解法】

解法：bfs 求最短路。

本质还是一个迷宫问题。

可以先预处理所有的流星雨，记录所有位置最早的爆炸时间 t 。只要后续 bfs 到某一点时，$dist[x][y] < t$ 就能到达该位置，否则就相当于障碍点。

当第一次搜索到一个没有被摧毁的位置时，就是我们要的最短时间。

【参考代码】

代码块

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

using namespace std;

const int N = 310, INF = 0x3f3f3f3f;

int m;
int t[N][N];
int dist[N][N];

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

int bfs()
{
    if(t[0][0] == INF) return 0;
    memset(dist, 0x3f, sizeof dist);

    queue<pair<int, int>> q;
    q.push({0, 0});
    dist[0][0] = 0;

    while(q.size())
    {
        auto a = q.front(); q.pop();
```

比特就业课

```
28          int i = a.first, j = a.second;
29
30          for(int k = 0; k < 4; k++)
31          {
32              int x = i + dx[k], y = j + dy[k];
33              if(x < 0 || y < 0) continue;
34              if(dist[x][y] != INF) continue;
35              if(dist[i][j] + 1 >= t[x][y]) continue;
36
37              dist[x][y] = dist[i][j] + 1;
38              if(t[x][y] == INF) return dist[x][y];
39
40              q.push({x, y});
41          }
42      }
43
44      return -1;
45  }
46
47  int main()
48  {
49      cin >> m;
50      memset(t, 0x3f, sizeof t);
51      for(int i = 1; i <= m; i++)
52      {
53          int x, y, z; cin >> x >> y >> z;
54          t[x][y] = min(t[x][y], z);
55
56          for(int k = 0; k < 4; k++)
57          {
58              int i = x + dx[k], j = y + dy[k];
59              if(i < 0 || j < 0) continue;
60              t[i][j] = min(t[i][j], z);
61          }
62      }
63
64      cout << bfs() << endl;
65
66      return 0;
67  }
```

比特就业课

4. 打鼹鼠

> **题目来源： 洛谷**

> 题目链接： P2285 [HNOI2004] 打鼹鼠

【题目描述】

> 鼹鼠是一种很喜欢挖洞的动物，但每过一定的时间，它还是喜欢把头探出到地面上来透透气的。根据这个特点阿牛编写了一个打鼹鼠的游戏：在一个 $n \times n$ 的网格中，在某些时刻鼹鼠会在某一个网格探出头来透透气。你可以控制一个机器人来打鼹鼠，如果 i 时刻鼹鼠在某个网格中出现，而机器人也处于同一网格的话，那么这个鼹鼠就会被机器人打死。而机器人每一时刻只能够移动一格或停留在原地不动。机器人的移动是指从当前所处的网格移向相邻的网格，即从坐标为 (i,j) 的网格移向 $(i-1,j),(i+1,j),(i,j-1),(i,j+1)$ 四个网格，机器人不能走出整个 $n \times n$ 的网格。游戏开始时，你可以自由选定机器人的初始位置。
>
> 现在知道在一段时间内，鼹鼠出现的时间和地点，请编写一个程序使机器人在这一段时间内打死尽可能多的鼹鼠。

【输入描述】

> 第一行为 $n,m(n \leq 1000, m \leq 10^4)$，其中 m 表示在这一段时间内出现的鼹鼠的个数，接下来的 m 行中每行有三个数据 $time,x,y$ 表示在游戏开始后 $time$ 个时刻，在第 x 行第 y 个网格里出现了一只鼹鼠。$time$ 按递增的顺序给出。注意同一时刻可能出现多只鼹鼠，但同一时刻同一地点只可能出现一只鼹鼠。

【输出描述】

> 仅包含一个正整数，表示被打死鼹鼠的最大数目。

【示例一】

> 输入：
>
> ```
> 2 2
> 1 1 1
> 2 2 2
> ```
>
> 输出：
>
> ```
> 1
> ```

【解法】

解法：动态规划 - 最长上升子序列模型。

看似是一个暴搜或者路径类 dp，实际上是一个最长上升子序列的变形。

我们不妨盯着所有鼹鼠出现时机的序列：随意取出前后两个鼹鼠，根据坐标的位置关系，其实很容易就能判断出打死前面这个鼹鼠之后，能不能接着打死后面这个鼹鼠。因此，我们不用在网格中一格一格的走，直接在鼹鼠序列中找出最长的能够被连续打死的序列。这不正是最长上升子序列的问题么？

（打死这么血腥，后面改成消灭了~）

比特就业课

1. 状态表示：

   $f[i]$ 表示：如果消灭的最后一只是 i 位置鼹鼠时，最长能消灭多少鼹鼠。

   那么 f 表里面的最大值就是我们要的结果。

2. 状态转移方程：

   对于 $f[i]$，可以分两种情况：

   - 只消灭这一只鼹鼠，此时 $f[i] = 1$；
   - 这只鼹鼠放在前面某只鼹鼠 $j\ (1 \le j < i)$ 后面一起消灭，此时需要满足 $abs(x[i] - x[j]) + abs(y[i] - y[j]) \le t[i] - t[j]$，也就是曼哈顿距离小于等于时间差，那么 $f[i] = f[j] + 1$。做出所有符合要求的最大值即可。

3. 初始化：

   不需要初始化，直接填表即可。

4. 填表顺序：

   从左往右。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 1e4 + 10;

int n, m;
int t[N], x[N], y[N];
int f[N];

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

    int ret = 0;
    for(int i = 1; i <= m; i++)
    {
        cin >> t[i] >> x[i] >> y[i];

```

比特就业课

```
            f[i] = 1;
            for(int j = 1; j < i; j++)
            {
                if(t[i] - t[j] >= abs(x[i] - x[j]) + abs(y[i] - y[j]))
                {
                    f[i] = max(f[i], f[j] + 1);
                }
            }
            ret = max(ret, f[i]);
        }

        cout << ret << endl;

        return 0;
    }
```

Day13

1. 神奇的幻方

> **题目来源：** 洛谷
>
> **题目链接：** P2615 [NOIP 2015 提高组] 神奇的幻方

【题目描述】

> 幻方是一种很神奇的 $N \times N$ 矩阵：它由数字 1,2,3,……, $N \times N$ 构成，且每行、每列及两条对角线上的数字之和都相同。
>
> 当 N 为奇数时，我们可以通过下方法构建一个幻方：
>
> 首先将 1 写在第一行的中间。
>
> 之后，按如下方式从小到大依次填写每个数 $K(K = 2, 3, ..., N \times N)$ ：
>
> 1. 若 $(K - 1)$ 在第一行但不在最后一列，则将 K 填在最后一行， $(K - 1)$ 所在列的右一列；
> 2. 若 $(K - 1)$ 在最后一列但不在第一行，则将 K 填在第一列， $(K - 1)$ 所在行的上一行；
> 3. 若 $(K - 1)$ 在第一行最后一列，则将 K 填在 $(K - 1)$ 的正下方；
> 4. 若 $(K - 1)$ 既不在第一行，也不在最后一列，如果 $(K - 1)$ 的右上方还未填数，则将 K 填在 $(K - 1)$ 的右上方，否则将 K 填在 $(K - 1)$ 的正下方。
>
> 现给定 N ，请按上述方法构造 $N \times N$ 的幻方。

【输入描述】

> 一个正整数 N ，即幻方的大小。

北京学就业课

对于 100% 的数据，对于全部数据， $1 \leq N \leq 39$ 且 N 为奇数。

【输出描述】

共 N 行，每行 N 个整数，即按上述方法构造出的 $N \times N$ 的幻方，相邻两个整数之间用单空格隔开。

【示例一】

输入：

3

输出：

8 1 6

3 5 7

4 9 2

【示例二】

输入：

25

输出：

327 354 381 408 435 462 489 516 543 570 597 624 1 28 55 82 109 136 163 190 217 244 271 298 325

353 380 407 434 461 488 515 542 569 596 623 25 27 54 81 108 135 162 189 216 243 270 297 324 326

379 406 433 460 487 514 541 568 595 622 24 26 53 80 107 134 161 188 215 242 269 296 323 350 352

405 432 459 486 513 540 567 594 621 23 50 52 79 106 133 160 187 214 241 268 295 322 349 351 378

431 458 485 512 539 566 593 620 22 49 51 78 105 132 159 186 213 240 267 294 321 348 375 377 404

457 484 511 538 565 592 619 21 48 75 77 104 131 158 185 212 239 266 293 320 347 374 376 403 430

483 510 537 564 591 618 20 47 74 76 103 130 157 184 211 238 265 292 319 346 373 400 402 429 456

509 536 563 590 617 19 46 73 100 102 129 156 183 210 237 264 291 318 345 372 399 401 428 455 482

比特就业课

535 562 589 616 18 45 72 99 101 128 155 182 209 236 263 290 317 344 371 398 425 427 454 481 508

561 588 615 17 44 71 98 125 127 154 181 208 235 262 289 316 343 370 397 424 426 453 480 507 534

587 614 16 43 70 97 124 126 153 180 207 234 261 288 315 342 369 396 423 450 452 479 506 533 560

613 15 42 69 96 123 150 152 179 206 233 260 287 314 341 368 395 422 449 451 478 505 532 559 586

14 41 68 95 122 149 151 178 205 232 259 286 313 340 367 394 421 448 475 477 504 531 558 585 612

40 67 94 121 148 175 177 204 231 258 285 312 339 366 393 420 447 474 476 503 530 557 584 611 13

66 93 120 147 174 176 203 230 257 284 311 338 365 392 419 446 473 500 502 529 556 583 610 12 39

92 119 146 173 200 202 229 256 283 310 337 364 391 418 445 472 499 501 528 555 582 609 11 38 65

118 145 172 199 201 228 255 282 309 336 363 390 417 444 471 498 525 527 554 581 608 10 37 64 91

144 171 198 225 227 254 281 308 335 362 389 416 443 470 497 524 526 553 580 607 9 36 63 90 117

170 197 224 226 253 280 307 334 361 388 415 442 469 496 523 550 552 579 606 8 35 62 89 116 143

196 223 250 252 279 306 333 360 387 414 441 468 495 522 549 551 578 605 7 34 61 88 115 142 169

222 249 251 278 305 332 359 386 413 440 467 494 521 548 575 577 604 6 33 60 87 114 141 168 195

248 275 277 304 331 358 385 412 439 466 493 520 547 574 576 603 5 32 59 86 113 140 167 194 221

274 276 303 330 357 384 411 438 465 492 519 546 573 600 602 4 31 58 85 112 139 166 193 220 247

300 302 329 356 383 410 437 464 491 518 545 572 599 601 3 30 57 84 111 138 165 192 219 246 273

301 328 355 382 409 436 463 490 517 544 571 598 625 2 29 56 83 110 137 164 191 218 245 272 299

【解法】

比特就业课

解法：模拟。

根据题意模拟整个填数过程即可。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 45;

int n;
int a[N][N];

int main()
{
    cin >> n;
    a[1][(n + 1) / 2] = 1;
    int x = 1, y = (n + 1) / 2; // 标记前一个数的横纵坐标

    for(int k = 2; k <= n * n; k++)
    {
        if(x == 1 && y == n) // 条件 3
        {
            a[x + 1][y] = k;
            x++;
        }
        else if(x == 1) // 条件 1
        {
            a[n][y + 1] = k;
            x = n; y++;
        }
        else if(y == n) // 条件 2
        {
            a[x - 1][1] = k;
            x--; y = 1;
        }
        else // 条件 4
        {
            if(a[x - 1][y + 1] == 0)
            {
                a[x - 1][y + 1] = k;
                x--; y++;
```

比特就业课

```
            }
            else
            {
                a[x + 1][y] = k;
                x++;
            }
        }
    }

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

    return 0;
}
```

2. 挖地雷

题目来源： 洛谷

题目链接： P2196 [NOIP 1996 提高组] 挖地雷

【题目描述】

在一个地图上有 $N(N \leq 20)$ 个地窖，每个地窖中埋有一定数量的地雷。同时，给出地窖之间的连接路径。当地窖及其连接的数据给出之后，某人可以从任一处开始挖地雷，然后可以沿着指出的连接往下挖（仅能选择一条路径），当无连接时挖地雷工作结束。设计一个挖地雷的方案，使某人能挖到最多的地雷。

【输入描述】

有若干行。

第 1 行只有一个数字，表示地窖的个数 N。

第 2 行有 N 个数，分别表示每个地窖中的地雷个数。

第 3 行至第 $N+1$ 行表示地窖之间的连接情况：

第 3 行有 $n-1$ 个数（0 或 1），表示第一个地窖至第 2 个、第 3 个 … 第 n 个地窖有否路径连接。如第 3 行为 11000…0，则表示第 1 个地窖至第 2 个地窖有路径，至第 3 个地窖有路径，至第 4 个地窖、第 5 个 … 第 n 个地窖没有路径。

比特就业课

> 第 4 行有 $n-2$ 个数，表示第二个地窖至第 3 个、第 4 个 … 第 n 个地窖有否路径连接。
>
> ……
>
> 第 $n+1$ 行有 1 个数，表示第 $n-1$ 个地窖至第 n 个地窖有否路径连接。（为 0 表示没有路径，为 1 表示有路径）。

【输出描述】

> 第一行表示挖得最多地雷时的挖地雷的顺序，各地窖序号间以一个空格分隔，不得有多余的空格。
>
> 第二行只有一个数，表示能挖到的最多地雷数。

【示例一】

> 输入：
>
> 5
>
> 10 8 4 7 6
>
> 1 1 1 0
>
> 0 0 0
>
> 1 1
>
> 1
>
> 输出：
>
> 1 3 4 5
>
> 27

【解法】

解法：动态规划 - 最长上升子序列求方案。

根据题目给的图结构可得，对于 `i` 号位置，只能从前面的某一个位置转移过来。

因此，原问题就变成：在 `1 ~ n` 中挑出一个可行的序列，这个序列中地雷的总数是最大的。就变成了一个最长上升子序列问题。

只不过，这道题需要求具体方案。用一个 `pre` 数组维护当前位置的最大值是通过谁转移过来的即可。

【参考代码】

代码块

```
#include <iostream>

using namespace std;
```

比特就业课

```cpp

const int N = 25;

int n;
int cnt[N];
int e[N][N];
int f[N], pre[N];

void dfs(int x)
{
    if(pre[x]) dfs(pre[x]);

    cout << x << " ";
}

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> cnt[i];
    for(int i = 1; i < n; i++)
        for(int j = i + 1; j <= n; j++)
            cin >> e[i][j];

    for(int i = 1; i <= n; i++)
    {
        f[i] = cnt[i];
        for(int j = 1; j < i; j++)
        {
            if(e[j][i])
            {
                if(f[j] + cnt[i] > f[i])
                {
                    f[i] = f[j] + cnt[i];
                    pre[i] = j;
                }
            }
        }
    }

    int ret = 0, p = 0;
    for(int i = 1; i <= n; i++)
    {
        if(f[i] > ret)
        {
            ret = f[i];
            p = i;
        }
```

比特就业课

```
51        }
52
53        dfs(p);
54        cout << endl << ret << endl;
55
56        return 0;
57    }
```

3. 路标设置

> **题目来源：** 洛谷
>
> **题目链接：** P3853 [TJOI2007] 路标设置

【题目描述】

> 现在政府决定在公路上增设一些路标，使得公路的“空旷指数”最小。他们请求你设计一个程序计算能达到的最小值是多少。请注意，公路的起点和终点保证已设有路标，公路的长度为整数，并且原有路标和新设路标都必须距起点整数个单位距离。

【输入描述】

> 第 1 行包括三个数 L, N, K ，分别表示公路的长度，原有路标的数量，以及最多可增设的路标数量。
>
> 第 2 行包括递增排列的 N 个整数，分别表示原有的 N 个路标的位置。路标的位置用距起点的距离表示，且一定位于区间 $[0, L]$ 内。
>
> 公路原来只在起点和终点处有两个路标，现在允许新增一个路标，应该把新路标设在距起点 50 或 51 个单位距离处，这样能达到最小的空旷指数 51。
>
> 50% 的数据中， $2 \leq N \leq 100$， $0 \leq K \leq 100$ 。
>
> 100% 的数据中， $2 \leq N \leq 100000, 0 \leq K \leq 100000$ 。
>
> 100% 的数据中， $0 < L \leq 10000000$ 。

【输出描述】

> 输出 1 行，包含一个整数，表示增设路标后能达到的最小“空旷指数”值。

【示例一】

> 输入：
>
> 101 2 1
>
> 0 101
>
> 输出：

比特就业课

51

【解法】

解法：二分答案。

二段性：设最优解为 `ret`：

- 如果 `x >= ret`，需要设置路标的数量就小于等于 `k`；
- 如果 `x < ret` ，需要设置路标的数量就大于 `k`。

对于一个数 `x`，如何判断设置路标的数量：

- 设 `d = a[i] - a[i - 1]`，那么需要路标的数量就是 `d / x`；
- 如果 `d % x == 0`，此时就可以少用一个路标。

【参考代码】

代码块

```
#include <iostream>

using namespace std;

const int N = 1e5 + 10;

int len, n, k;
int a[N];

bool check(int x)
{
    int cnt = 0;
    for(int i = 2; i <= n; i++)
    {
        int d = a[i] - a[i - 1];
        cnt += d / x;
        if(d % x == 0) cnt--;
    }

    return cnt <= k;
}

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

比特就业课

```
26      for(int i = 1; i <= n; i++) cin >> a[i];
27
28      int l = 1, r = len;
29      while(l < r)
30      {
31          int mid = (l + r) / 2;
32          if(check(mid)) r = mid;
33          else l = mid + 1;
34      }
35
36      cout << l << endl;
37
38      return 0;
39  }
```

4. 关押罪犯

> **题目来源：** 洛谷
>
> **题目链接：** P1525 [NOIP2010 提高组] 关押罪犯

【题目描述】

> S 城现有两座监狱，一共关押着 N 名罪犯，编号分别为 $1 \sim N$。他们之间的关系自然也极不和谐。很多罪犯之间甚至积怨已久，如果客观条件具备则随时可能爆发冲突。我们用“怨气值”（一个正整数值）来表示某两名罪犯之间的仇恨程度，怨气值越大，则这两名罪犯之间的积怨越多。如果两名怨气值为 c 的罪犯被关押在同一监狱，他们俩之间会发生摩擦，并造成影响力为 c 的冲突事件。
>
> 每年年末，警察局会将本年内监狱中的所有冲突事件按影响力从大到小排成一个列表，然后上报到 S 城 Z 市长那里。公务繁忙的 Z 市长只会去看列表中的第一个事件的影响力，如果影响很坏，他就会考虑撤换警察局长。
>
> 在详细考察了 N 名罪犯间的矛盾关系后，警察局长觉得压力巨大。他准备将罪犯们在两座监狱内重新分配，以求产生的冲突事件影响力都较小，从而保住自己的乌纱帽。假设只要处于同一监狱内的某两个罪犯间有仇恨，那么他们一定会在每年的某个时候发生摩擦。
>
> 那么，应如何分配罪犯，才能使 Z 市长看到的那个冲突事件的影响力最小？这个最小值是多少？

【输入描述】

> 每行中两个数之间用一个空格隔开。第一行为两个正整数 N, M，分别表示罪犯的数目以及存在仇恨的罪犯对数。接下来的 M 行每行为三个正整数 a_j, b_j, c_j，表示 a_j 号和 b_j 号罪犯之间存在仇恨，其怨气值为 c_j。数据保证 $1 < a_j < b_j \leq N, 0 < c_j \leq 10^9$，且每对罪犯组合只出现一次。

【输出描述】

比特就业课

> 共一行，为 Z 市长看到的那个冲突事件的影响力。如果本年内监狱中未发生任何冲突事件，请输出 `0`。

【示例一】

> 输入：
>
> 4 6
>
> 1 4 2534
>
> 2 3 3512
>
> 1 2 28351
>
> 1 3 6618
>
> 2 4 1805
>
> 3 4 12884
>
> 输出：
>
> 3512

【解法】

解法：贪心 + 扩展域并查集。

贪心：从大到小分割每一条边。当某一条边无法分割的时候，这条边就是最终结果。

可以利用扩展域并查集维护每一条边断开后的两个集合，并且判断某条边是否能够断开：

- 先断开：合并 a 和 b + n，b 和 a + n；
- 判断是否能断开：判断 a 和 b 是否在同一个集合中。

【参考代码】

代码块

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

using namespace std;

const int N = 4e4 + 10, M = 1e5 + 10;

int n, m;
struct node
{
    int a, b, c;
```

比特就业课

```cpp
}e[M];

int fa[N];

int find(int x)
{
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}

void un(int x, int y)
{
    fa[find(x)] = find(y);
}

bool cmp(node& x, node& y)
{
    return x.c > y.c;
}

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

    // 初始化
    for(int i = 1; i <= n + n; i++) fa[i] = i;

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

    for(int i = 1; i <= m; i++)
    {
        int a = e[i].a, b = e[i].b, c = e[i].c;
        un(a, b + n); un(b, a + n);

        if(find(a) == find(b))
        {
            cout << c << endl;
            return 0;
        }
    }

    // 细节问题
    cout << 0 << endl;

    return 0;
}
```

比特就业课

Day14

1. Crazy Computer

> **题目来源：** 洛谷
>
> **题目链接：** CF716A Crazy Computer

【题目描述】

> 给你一个N(N<=100000)个字母敲击的时间$a[i]$($a[i]$<=109)，如果在M时间内没有敲击那么屏幕就清零，否则屏幕上就多一个字母，问最后屏幕剩下几个字母。
>
> 打下下一个字母的时候，如果和之前字母打下的时间不超过k的话，则保留前面的继续打，如果超过了，则前面的字母全部消失，只留下这一个字母。

【示例一】

> 输入：
>
> 6 5
>
> 1 3 8 14 19 20
>
> 输出：
>
> 3

【解法】

解法：模拟。

从前往后模拟整个过程即可。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 1e5 + 10;

int n, m;
int a[N];

int main()
{
```

比特就业课

```
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> a[i];

    int cnt = 1;
    for(int i = 2; i <= n; i++)
    {
        if(a[i] - a[i - 1] <= m) cnt++;
        else cnt = 1;
    }

    cout << cnt << endl;

    return 0;
}
```

2. 转圈游戏

> **题目来源：** 洛谷
>
> **题目链接：** P1965 [NOIP 2013 提高组] 转圈游戏

【题目描述】

> n 个小伙伴（编号从 0 到 $n-1$）围坐一圈玩游戏。按照顺时针方向给 n 个位置编号，从 0 到 $n-1$。最初，第 0 号小伙伴在第 0 号位置，第 1 号小伙伴在第 1 号位置，……，依此类推。游戏规则如下：每一轮第 0 号位置上的小伙伴顺时针走到第 m 号位置，第 1 号位置小伙伴走到第 $m+1$ 号位置，……，依此类推，第 $n-m$ 号位置上的小伙伴走到第 0 号位置，第 $n-m+1$ 号位置上的小伙伴走到第 1 号位置，……，第 $n-1$ 号位置上的小伙伴顺时针走到第 $m-1$ 号位置。
>
> 现在，一共进行了 10^k 轮，请问 x 号小伙伴最后走到了第几号位置。

【输入描述】

> 共一行，包含四个整数 n, m, k, x，每两个整数之间用一个空格隔开。
>
> 对于 30% 的数据，$0 < k < 7$；
>
> 对于 80% 的数据，$0 < k < 10^7$；
>
> 对于 100% 的数据，$1 < n < 10^6, 0 < m < n, 0 \le x \le n, 0 < k < 10^9$。

【输出描述】

> 一个整数，表示 10^k 轮后 x 号小伙伴所在的位置编号。

【示例一】

比特就业课

> 输入：
>
> 10 3 4 5
>
> 输出：
>
> 5

【解法】

解法：数学 + 快速幂。

计算 $(x + m * 10^k) \% n$ 的值即可。

其中 10^k 非常大，需要使用能取模的快速幂求。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

LL n, m, k, x;

LL qpow(LL a, LL b, LL p)
{
    LL ret = 1;
    while(b)
    {
        if(b & 1) ret = ret * a % p;
        b >>= 1;
        a = a * a % p;
    }

    return ret;
}

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

    cout << (x + m * qpow(10, k, n)) % n << endl;

    return 0;
```

比特就业课

```
29  }
```

3. System Administrator

题目来源： 洛谷

题目链接： CF22C System Administrator

【题目描述】

给定 n 个顶点，可以在这些顶点之间连接 m 条双向边，要求连接后整个图联通，并且去掉 v 点后整个图不连通。

若有解，输出所连得的 m 条边，随意输出一种方案即可；若无解，输出 `-1`。

【输入描述】

一行三个整数表示 n, m, v。

- $n, m \leq 10^5$

【输出描述】

共 m 行，每行两个整数 u_i 和 v_i，表示 u_i 向 v_i 连一条边。

【示例一】

输入：

```
5 6 3
```

输出：

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

【示例二】

输入：

```
6 100 1
```

输出：

比特就业课

```
-1
```

【解法】

解法：构造。

如果边的数量小于 `n - 1`，那么连通图就不存在。当有一个顶点与 `v` 相连，并且与其他顶点都不相连时，就达到了最大可能的边数，这些边可以构成完整的图，此时最大的边数是 `(n - 1)*(n - 2) / 2 + 1`。如果 `m` 在这个范围内，那么所需的图总是存在的。

如果图存在，可以先在 `v` 的一边放置一个顶点（设为 `x`），在另一边放置其他顶点。接下来将所有这些顶点连接到 `v` 上，然后将它们相互连接起来（除了 `x`）。

【参考代码】

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

LL n, m, v;

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

    if(m < n - 1 || (n - 1) * (n - 2) / 2 + 1 < m)
    {
        cout << -1 << endl;
        return 0;
    }

    for(int i = 1; i <= n; i++)
    {
        if(i != v)
        {
            cout << i << " " << v << endl;
            m--;
        }
    }

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j < i; j++)
```

比特就业课

```
31          {
32              if(!m) return 0;
33              if(i == v || j == v) continue;
34              cout << i << " " << j << endl;
35              m--;
36          }
37      }
38
39      return 0;
40  }
```

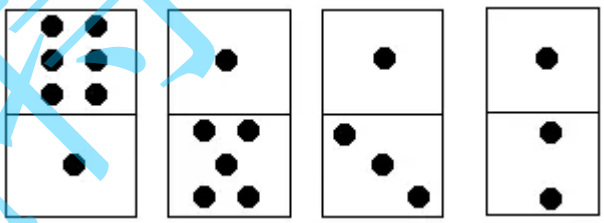
4. 多米诺骨牌

> **题目来源：** 洛谷
>
> **题目链接：** P1282 多米诺骨牌

【题目描述】

多米诺骨牌由上下 2 个方块组成，每个方块中有 $1\sim6$ 个点。

现有排成行的上方块中点数之和记为 S_1，下方块中点数之和记为 S_2，它们的差为 $|S1-S2|$。如图，$S_1=6+1+1+1=9$，$S_2=1+5+3+2=11$，$|S1-S2|=2$。每个多米诺骨牌可以旋转 180°，使得上下两个方块互换位置。请你计算最少旋转多少次才能使多米诺骨牌上下 2 行点数之差达到最小。



对于图中的例子，只要将最后一个多米诺骨牌旋转 180°，即可使上下 2 行点数之差为 0。

【输入描述】

输入文件的第一行是一个正整数 $n(1\le n\le 1000)$，表示多米诺骨牌数。接下来的 n 行表示 n 个多米诺骨牌的点数。每行有两个用空格隔开的正整数，表示多米诺骨牌上下方块中的点数 a 和 b，且 $1\le a,b\le 6$。

【输出描述】

输出文件仅一行，包含一个整数。表示求得的最小旋转次数。

【示例一】

输入：

比特就业课

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

输出：

```
1
```

【解法】

解法：动态规划。

对于每一个骨牌，设上面的点数是 x ，下面的点数是 y ：

- 如果不旋转，对于总差值的贡献就是 $x-y$ ；
- 如果旋转，对于总差值的贡献就是 $y-x=-(x-y)$ ；

也就是说，对于每个骨牌，我们统计一下上下点数的差值 $a[i]$ ，然后的操作就是要么旋转，要么不旋转，最终想看看总和最接近 0 的值的最小旋转次数，正好对应 01 背包问题。

但是，此时会有一个问题，就是我们的总和有可能是负数，对应不到 dp 表里面的下标。对于这种问题，我们常见的处理操作就是统一将第二维加上一个偏移量，相当于将整个 dp 表向右平移，直到所有的下标都是正数。

本题的偏移量可以设置成 5000 ，因为最小值不会低于 -5000 。

1. 状态表示：

   $dp[i][j]$ 表示：考虑前 i 个骨牌，确定好它们的状态之后，上下差值正好为 j 时，此时的最小旋转次数。

   最终结果就是 $dp[n][j]$ 中，状态合法，并且 j 的绝对值最小时的值。

2. 状态转移方程：

   对于 i 位置的骨牌，我们有不旋转或者旋转两种状态：

   a. 不旋转：此时 i 位置骨牌对上下差值的贡献就是 $a[i]$ ，那么就需要去 $[1,i-1]$ 中凑总和为 $j-a[i]$ 的最小旋转次数，即 $dp[i-1][j-a[i]]$ ；

   b. 旋转：此时 i 位置骨牌对上下差值的贡献就是 $-a[i]$ ，那么就需要去 $[1,i-1]$ 中凑总和为 $j+a[i]$ 的最小旋转次数再加上本次旋转，即 $dp[i-1][j-a[i]]+1$ ；

   综上， $dp[i][j]$ 应该为上面两种情况的最小值。

比特就业课

3. 初始化：

先把所有位置初始化为正无穷大 $0x3f3f3f3f$ ，然后把 $dp[0][0]$ 初始化为 0 。

4. 填表顺序：

从上往下每一行，每一行从左往右。

【参考代码】

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

using namespace std;

const int N = 1010, M = 1e4 + 10, INF = 0x3f3f3f3f;

int n, m = 5000;
int a[N];
int f[N][M];

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

    memset(f, 0x3f, sizeof f);
    f[0][0 + m] = 0;

    for(int i = 1; i <= n; i++)
    {
        for(int j = -m; j <= m; j++)
        {
            f[i][j + m] = min(f[i - 1][j - a[i] + m], f[i - 1][j + a[i] + m] + 1);
        }
    }

    int ret = INF;
```

比特就业课

```
33        for(int i = 0; i <= m; i++)
34        {
35            ret = min(f[n][i + m], f[n][-i + m]);
36            if(ret != INF) break;
37        }
38
39        cout << ret << endl;
40
41        return 0;
42    }
```

Day15

1. 排队接水

> **题目来源：** 洛谷
>
> **题目链接：** P1223 排队接水

【题目描述】

> 有 n 个人在一个水龙头前排队接水，假如每个人接水的时间为 T_i，请编程找出这 n 个人排队的一种顺序，使得 n 个人的平均等待时间最小。

【输入描述】

> 第一行为一个整数 n。
>
> 第二行 n 个整数，第 i 个整数 T_i 表示第 i 个人的接水时间 T_i。
>
> $1 \le n \le 1000$，$1 \le t_i \le 10^6$，不保证 t_i 不重复。

【输出描述】

> 输出文件有两行，第一行为一种平均时间最短的排队顺序；第二行为这种排列方案下的平均等待时间（输出结果精确到小数点后两位）。

【示例一】

> 输入：
>
> 10
>
> 56 12 1 99 1000 234 33 55 99 812
>
> 输出：
>
> 3 2 7 8 1 4 9 6 10 5

比特就业课

> 291.90

【解法】

解法：贪心。

显然是从小到大接水，对于后面的人来说等待时间更短。

【参考代码】

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

using namespace std;

const int N = 1e3 + 10;

int n;
struct node
{
    int t, id;
}a[N];

bool cmp(node& x, node& y)
{
    return x.t < y.t;
}

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        cin >> a[i].t;
        a[i].id = i;
    }

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

    double sum = 0, wait = 0;
    for(int i = 1; i <= n; i++)
    {
        cout << a[i].id << " ";
        sum += wait;
        wait += a[i].t;
```

比特就业课

```
36        }
37
38        printf("\n%.2lf\n", sum / n);
39
40        return 0;
41    }
```

2. 健康的荷斯坦奶牛

题目来源： 洛谷

题目链接： P1460 [USACO2.1] 健康的荷斯坦奶牛 Healthy Holsteins

【题目描述】

农民 John 以拥有世界上最健康的奶牛为傲。他知道每种饲料中所包含的牛所需的最低的维他命量是多少。请你帮助农夫喂养他的牛，以保持它们的健康，使喂给牛的饲料的种数最少。

给出牛所需的最低的维他命量，输出喂给牛需要哪些种类的饲料，且所需的饲料剂量最少。

维他命量以整数表示，每种饲料最多只能对牛使用一次，数据保证存在解。

【输入描述】

第一行一个整数 v，表示需要的维他命的种类数。
第二行 v 个整数，表示牛每天需要的每种维他命的最小量。

第三行一个整数 g，表示可用来喂牛的饲料的种数。
下面 g 行，第 n 行表示编号为 n 饲料包含的各种维他命的量的多少。

【输出描述】

输出文件只有一行，包括牛必需的最小的饲料种数 p；后面有 p 个数，表示所选择的饲料编号（按从小到大排列）。

如果有多个解，输出饲料序号最小的（即字典序最小）。

【示例一】

输入：

```
4
100 200 300 400
3
50  50  50  50
200 300 200 300
900 150 389 399
```

比特就业课

> 输出：
>
> 2 1 3

【解法】

解法：暴搜 dfs。

从前往后考虑每一个饲料，对于当前饲料要么选，要么不选，dfs 可以把所有情况枚举出来。如果从小到大考虑，最终结果天然是按照字典序排列的。

剪枝：

- 如果当前选择的饲料总数"大于等于"之前已经搜到过的最优解，减掉；
- 如果当前选择的饲料已经满足要求，判断是否是最优解之后，减掉。

【参考代码】

代码块

```
#include <iostream>

using namespace std;

const int N = 30;

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

int cnt; // 记录当前选了多少个
int path; // 记录选了哪些饲料

int ret = N; // 记录最少选了多少
int st;

// 当前的决策是 path 是否满足奶牛的需要
bool check()
{
    for(int i = 1; i <= n; i++)
    {
        int sum = 0;
        for(int j = 1; j <= m; j++)
        {
            if((path >> j) & 1) sum += g[j][i];
```

七特就业课

```
26          }
27          if(sum < v[i]) return false;
28      }
29
30      return true;
31  }
32
33  void dfs(int pos)
34  {
35      // 最优性剪枝
36      if(cnt >= ret) return;
37      if(check())
38      {
39          ret = cnt;
40          st = path;
41          return;
42      }
43
44      if(pos > m) return;
45
46      // 选
47      cnt++;
48      path |= (1 << pos);
49      dfs(pos + 1);
50      cnt--;
51      path &= ~(1 << pos);
52
53      // 不选
54      dfs(pos + 1);
55  }
56
57  int main()
58  {
59      cin >> n;
60      for(int i = 1; i <= n; i++) cin >> v[i];
61      cin >> m;
62      for(int i = 1; i <= m; i++)
63          for(int j = 1; j <= n; j++)
64              cin >> g[i][j];
65
66      dfs(1);
67
68      cout << ret << " ";
69      for(int i = 1; i <= m; i++)
70      {
71          if((st >> i) & 1) cout << i << " ";
72      }
```

比特就业课

```
73
74
75        return 0;
76    }
```

3. 税收与补贴问题

> **题目来源：** 洛谷
>
> **题目链接：** P1023 [NOIP 2000 普及组] 税收与补贴问题

【题目描述】

> 你是某家咨询公司的项目经理，现在你已经知道政府对某种商品的预期价格，以及在各种价位上的销售情况。要求你确定政府对此商品是应收税还是补贴的最少金额（也为整数），才能使商家在这样一种政府预期的价格上，获取相对其他价位上的最大总利润。
>
> - 总利润 = 单位商品利润 × 销量
> - 单位商品利润 = 单位商品价格 − 单位商品成本（减去税金 或者 加上补贴）

【输入描述】

> 输入的第一行为政府对某种商品的预期价；
>
> 第二行有两个整数，第一个整数为商品成本，第二个整数为以成本价销售时的销售量；
>
> 接下来若干行，每行都有两个整数，第一个为某价位时的单价，第二个为此时的销量，以一行 `-1 -1` 表示所有已知价位及对应的销量输入完毕；
>
> 输入的最后一行为一个单独的整数表示在已知的最高单价外每升高一块钱将减少的销量。
>
> 保证输入的所有数字均小于 10^5

【输出描述】

> 输出有两种情况：若在政府预期价上能得到最大总利润，则输出一个单独的整数，数的正负表示是补贴还是收税，数的大小表示补贴或收税的金额最小值。若有多解，取绝对值最小的输出。
>
> 如在政府预期价上不能得到最大总利润，则输出 `NO SOLUTION`。

【示例一】

> 输入：
>
> 31
>
> 28 130
>
> 30 120

31 110

-1 -1

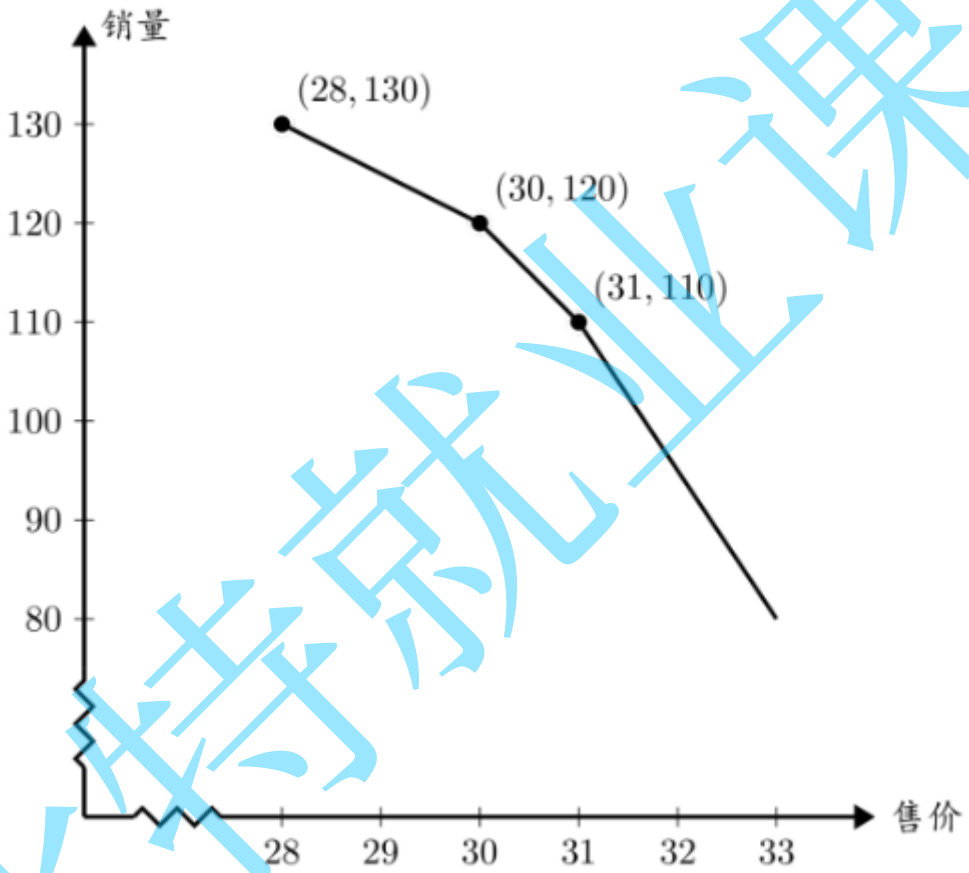
15

输出：

4

说明：

如下图所示是输入样例所对应的价格变化图，横轴表示销售价格，纵轴表示销量



根据题意，28 元是商品的成本。销售价格不应该低于 28 元；当销售价格大于给出的价格的最大值 31 元后，按照售价每提高一元，销量降低 15 计算，例如当售价为 33 元时，销量为 110－15×(33－31)=80。在给出来的价位之间，销量呈线性变化。

当政府给该商品补贴 4 元后，企业将该商品定价为 31 元时，取得的利润为 31－28+4=7 元，销量为 110 件，总利润为 7×110=770 元，是企业在所有定价下能够取得的最大的总利润。此时企业的售价为政府的期望售价，因此是一个合法方案。

【解法】

解法：数学。

【参考代码】

比特就业课

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

using namespace std;

const int N = 1e5 + 10;

int aim, a;
int c[N]; // c[i] 表示：售价为 i 时，销量为 c[i]

int main()
{
    cin >> aim;
    int x, y; cin >> x >> y;
    a = x; c[x] = y;
    int prev = x; // 前一个定价

    while(cin >> x >> y, x != -1 && y != -1)
    {
        int d = (c[prev] - y) / (x - prev);

        for(int i = prev + 1, j = c[prev] - d; i <= x; i++, j -= d)
        {
            c[i] = j;
        }
        prev = x;
    }

    int d; cin >> d;
    for(int i = prev + 1, j = c[prev] - d; j >= 0; i++, j -= d)
    {
        c[i] = j;
        prev = i;
    }

    // for(int i = a; i <= prev; i++)
    // {
    //     cout << i << " " << c[i] << endl;
    // }

    // a ~ prev
    double left = -1e9, right = 1e9;
    for(int i = a; i <= prev; i++)
    {
        // i -> c[i];
        double x = 1.0 * ((i - a) * c[i] - (aim - a) * c[aim]) / (c[aim] - c[i]);
```

比特就业课

```
47
48          if(c[aim] > c[i]) left = max(left, x);
49          else if(c[aim] < c[i]) right = min(right, x);
50      }
51
52      if(left > right) cout << "NO SOLUTION" << endl;
53      else if(right < 0) cout << (int)floor(right) << endl;
54      else if(left > 0) cout << (int)ceil(left) << endl;
55      else cout << 0 << endl;
56
57      return 0;
58  }
```

4. 传纸条

> **题目来源：** 洛谷
>
> **题目链接：** P1006 [NOIP2008 提高组] 传纸条

【题目描述】

> 小渊和小轩是好朋友也是同班同学，他们在一起总有谈不完的话题。一次素质拓展活动中，班上同学安排坐成一个 m 行 n 列的矩阵，而小渊和小轩被安排在矩阵对角线的两端，因此，他们就无法直接交谈了。幸运的是，他们可以通过传纸条来进行交流。纸条要经由许多同学传到对方手里，小渊坐在矩阵的左上角，坐标 $(1,1)$，小轩坐在矩阵的右下角，坐标 (m,n)。从小渊传到小轩的纸条只可以向下或者向右传递，从小轩传给小渊的纸条只可以向上或者向左传递。
>
> 在活动进行中，小渊希望给小轩传递一张纸条，同时希望小轩给他回复。班里每个同学都可以帮他们传递，但只会帮他们一次，也就是说如果此人在小渊递给小轩纸条的时候帮忙，那么在小轩递给小渊的时候就不会再帮忙。反之亦然。
>
> 还有一件事情需要注意，全班每个同学愿意帮忙的好感度有高有低（注意：小渊和小轩的好心程度没有定义，输入时用 0 表示），可以用一个 $[0,100]$ 内的自然数来表示，数越大表示越好心。小渊和小轩希望尽可能找好心程度高的同学来帮忙传纸条，即找到来回两条传递路径，使得这两条路径上同学的好心程度之和最大。现在，请你帮助小渊和小轩找到这样的两条路径。

【输入描述】

> 第一行有两个用空格隔开的整数 m 和 n，表示班里有 m 行 n 列。
>
> 接下来的 m 行是一个 $m \times n$ 的矩阵，矩阵中第 i 行 j 列的整数表示坐在第 i 行 j 列的学生的好心程度。每行的 n 个整数之间用空格隔开。
>
> 对于 100% 的数据，满足 $1 \leq m,n \leq 50$。

优特就业课

【输出描述】

> 输出文件共一行一个整数，表示来回两条路上参与传递纸条的学生的好心程度之和的最大值。

【示例一】

> 输入：
>
> 3 3
>
> 0 3 9
>
> 2 8 5
>
> 5 7 0
>
> 输出：
>
> 34

【解法】

解法：动态规划 - 路径类dp。

类似方格取数，但与方格取数的不同点在于处理相遇点的策略。

1. 状态表示：

   $f[st][i_1][i_2]$ 表示：第一条路在 $[i_1, st - i_1]$，第二条路在 $[i_2, st - i_2]$ 时，两者的路径最大和。

   那我们的最终结果就是 $f[n + m][n][n]$。

2. 状态转移方程：

   第一条路可以从上 $[i_1 - 1, st - i_1]$ 或者左 $[i_1, st - i_1 - 1]$ 走到 $[i_1, st - i_1]$ 位置；第二条路可以从上 $[i_2 - 1, st - i_2]$ 或者左 $[i_2, st - i_2 - 1]$ 走到 $[i_2, st - i_2]$ 位置。排列组合一下一共 4 中情况，分别是：

   - 上 + 上，此时的最大和为：$f[st - 1][i_1 - 1][i_2 - 1]$；
   - 上 + 左，此时的最大和为：$f[st - 1][i_1 - 1][i_2]$；
   - 左 + 上，此时的最大和为：$f[st - 1][i_1][i_2 - 1]$；
   - 左 + 左，此时的最大和为：$f[st - 1][i_1][i_2]$；

   取上面四种情况的最大值，然后再加上 $a[i_1][j_1]$ 和 $a[i_2][j_2]$。

   但是要注意，如果两个路径当前在同一位置时，直接 $continue$，因为两条路不能走在相同的位置。不过，如果是最后一个格子的话，还是要计算的。

3. 初始化：

北京就业课

算的是路径和，0 不会影响最终结果，直接填表。

4. 填表顺序：

先从小到大循环横纵坐标之和，然后依次从小到大循环两者的横坐标。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 55;

int m, n;
int a[N][N];
int f[N + N][N][N];

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

    for(int s = 2; s <= n + m; s++)
    {
        for(int i1 = 1; i1 <= m; i1++)
        {
            for(int i2 = 1; i2 <= m; i2++)
            {
                if(i1 == i2 && s != n + m) continue;
                int j1 = s - i1, j2 = s - i2;
                if(j1 < 1 || j1 > n || j2 < 1 || j2 > n) continue;

                int& t = f[s][i1][i2];
                t = max(t, f[s - 1][i1 - 1][i2 - 1]);
                t = max(t, f[s - 1][i1 - 1][i2]);
                t = max(t, f[s - 1][i1][i2 - 1]);
                t = max(t, f[s - 1][i1][i2]);

                t += a[i1][j1] + a[i2][j2];
            }
        }
```

比特就业课

```
37        }
38
39        cout << f[n + m][m][m] << endl;
40
41        return 0;
42    }
```

比特就业课