

第 4 周

Day16

1. 小A的糖果

题目来源：洛谷

题目链接：[P3817 小A的糖果](#)

【题目描述】

小 A 有 n 个糖果盒，第 i 个盒中有 a_i 颗糖果。

小 A 每次可以从其中一盒糖果中吃掉一颗，他想知道，要让任意两个相邻的盒子中糖的个数之和都不大于 x ，至少得吃掉几颗糖。

【输入描述】

输入的第一行是两个用空格隔开的整数，代表糖果盒的个数 n 和给定的参数 x 。

第二行有 n 个用空格隔开的整数，第 i 个整数代表第 i 盒糖的糖果个数 a_i 。

- 对于 30% 的数据，保证 $n \leq 20$ ， $a_i, x \leq 100$ 。
- 对于 70% 的数据，保证 $n \leq 10^3$ ， $a_i, x \leq 10^5$ 。
- 对于 100% 的数据，保证 $2 \leq n \leq 10^5$ ， $0 \leq a_i, x \leq 10^9$ 。

【输出描述】

输出一行一个整数，代表最少要吃掉的糖果的数量。

【示例一】

输入：

3 3

2 2 2

输出：

1

说明：

吃掉第 2 盒中的一个糖果即可。

【示例二】

输入：

6 1

1 6 1 2 0 4

输出：

11

说明：

第 2 盒糖吃掉 6 颗，第 4 盒吃掉 2 颗，第 6 盒吃掉 3 颗。

【示例三】

输入：

5 9

3 1 4 1 5

输出：

0

【解法】

解法：贪心。

从前往后考虑每一个元素 i ，如果 $a[i] + a[i - 1] > x$ ，那么就需要拿走糖果：

- 贪心一：为了拿走的糖果数尽可能少，应该拿 $d = a[i] + a[i - 1] - x$ ；
- 贪心二：为了让后面相邻元素累加尽可能小，应该从 i 位置拿走 d 的糖果，此时 $a[i] = a[i] - d$ ，化简完之后为 $a[i] = x - a[i - 1]$ 。

注意，第一个元素也要和前面加，加的数是 0，加完之后保证自己本身不会超过 x 。后续再出现相邻元素之和超过 x 的情况，必定是后一个元素导致的。

【参考代码】

```
1  #include <iostream>
2
3  using namespace std;
4
5  typedef long long LL;
6
7  const int N = 1e5 + 10;
8
9  LL n, x;
```

```

10  LL a[N];
11
12  int main()
13  {
14      cin >> n >> x;
15
16      LL sum = 0;
17      for(int i = 1; i <= n; i++)
18      {
19          cin >> a[i];
20
21          LL d = a[i] + a[i - 1] - x;
22          if(d > 0)
23          {
24              sum += d;
25              a[i] = x - a[i - 1];
26          }
27      }
28
29      cout << sum << endl;
30
31      return 0;
32  }

```

2. Cow Picnic

题目来源：洛谷

题目链接： [P2853 \[USACO06DEC\] Cow Picnic S](#)

【题目描述】

$K(1 \leq K \leq 100)$ 只奶牛分散在 $N(1 \leq N \leq 1000)$ 个牧场。现在她们要集中起来进餐。牧场之间有 $M(1 \leq M \leq 10000)$ 条有向路连接，而且不存在起点和终点相同的有向路。她们进餐的地点必须是所有奶牛都可到达的地方。那么，有多少这样的牧场可供进食呢？

【示例一】

输入：

2 4 4

2

3

1 2

1 4

2 3

3 4

输出：

2

【解法】

解法：暴搜。

以每一个奶牛的位置作为起点做一次 dfs/bfs，标记能走到的结点。如果结点被标记了 k 次，就是所找的点。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2  #include <cstring>
3  #include <vector>
4
5  using namespace std;
6
7  const int N = 1010;
8
9  int k, n, m;
10 int a[N]; // k 头奶牛的编号
11 vector<int> edges[N];
12
13 int cnt[N];
14 bool st[N];
15
16 void dfs(int u)
17 {
18     st[u] = true;
19     cnt[u]++;
20
21     for(auto v : edges[u])
22     {
23         if(!st[v]) dfs(v);
24     }
25 }
26
27 int main()
```

```

28  {
29      cin >> k >> n >> m;
30      for(int i = 1; i <= k; i++) cin >> a[i];
31      for(int i = 1; i <= m; i++)
32      {
33          int x, y; cin >> x >> y;
34          edges[x].push_back(y);
35      }
36
37      for(int i = 1; i <= k; i++)
38      {
39          // a[i]
40          memset(st, 0, sizeof st);
41          dfs(a[i]);
42      }
43
44      int ret = 0;
45      for(int i = 1; i <= n; i++)
46      {
47          if(cnt[i] == k) ret++;
48      }
49
50      cout << ret << endl;
51
52      return 0;
53  }

```

3. Tallest Cow

题目来源：洛谷

题目链接：[P2879 \[USACO07JAN\] Tallest Cow S](#)

【题目描述】

FarmerJohn 有 n 头牛，它们按顺序排成一列。FarmerJohn 只知道其中最高的奶牛的序号及它的高度，其他奶牛的高度都是未知的。现在 FarmerJohn 手上有 r 条信息，每条信息上有两头奶牛的序号（ a 和 b ），其中 b 奶牛的高度一定大于等于 a 奶牛的高度，且 a, b 之间的所有奶牛的高度都比 a 小。现在 FarmerJohn 想让你根据这些信息求出每一头奶牛的可能的最大的高度。（数据保证有解）

【输入描述】

第一行：四个以空格分隔的整数： n, i, h, r （ n 和 r 意义见题面； i 和 h 表示第 i 头牛的高度为 h ，他是最高的奶牛）

接下来 r 行：两个不同的整数 a 和 b ($1 \leq a, b \leq n$)

- $1 \leq n \leq 10000, 1 \leq h \leq 1000000, 0 \leq r \leq 10000$

【输出描述】

一共 n 行，表示每头奶牛的最大可能高度。

【示例一】

输入：

9 3 5 5

1 3

5 3

4 3

3 7

9 8

输出：

5

4

5

3

4

4

5

5

5

【解法】

解法：贪心 + 差分。

因为题目要求的是所有奶牛的「最大高度」，那我们不妨设所有奶牛的初始高度都是 h 。对于任意一对关系 (a, b) ，让 a 和 b 位置的奶牛高度不变，然后让 $[a + 1, b - 1]$ 区间内所有奶牛的高度「减 1」。这样处理 R 条信息之后，既满足「高低」要求，又可以让所有奶牛的高度「最高」。

涉及「区间」统一减 1 的操作，可以利用「差分」数组。

需要注意两点：

1. a, b 的大小关系；
2. 有可能给出 $(a, b), (b, a), (a, b)$ 这样的信息，此时我们只用修改一次，注意去重。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2  #include <set>
3
4  using namespace std;
5
6  const int N = 1e4 + 10;
7
8  int n, id, h, r;
9  int f[N]; // 差分数组
10
11 int main()
12 {
13     cin >> n >> id >> h >> r;
14
15     set<pair<int, int>> mp; // 帮助去重
16     while(r--)
17     {
18         int a, b; cin >> a >> b;
19         if(a > b) swap(a, b);
20         // [a + 1, b - 1]
21         if(mp.count({a, b})) continue;
22
23         f[a + 1]--; f[b]++;
24         mp.insert({a, b});
25     }
26
27     for(int i = 1; i <= n; i++)
28     {
29         f[i] += f[i - 1];
30         // cout << f[i] << " ";
31         cout << h + f[i] << endl;
32     }
33
34     return 0;
35 }
```

4. 英雄联盟

题目来源：洛谷

题目链接：P5365 [SNOI2017] 英雄联盟

【题目描述】

正在上大学的小皮球热爱英雄联盟这款游戏，而且打的很菜，被网友们戏称为「小学生」。

现在，小皮球终于受不了网友们的嘲讽，决定变强了，他变强的方法就是：买皮肤！

小皮球只会玩 N 个英雄，因此，他也只准备给这 N 个英雄买皮肤，并且决定，以后只玩有皮肤的英雄。

这 N 个英雄中，第 i 个英雄有 K_i 款皮肤，价格是每款 C_i Q 币（同一个英雄的皮肤价格相同）。

为了让自己看起来高大上一些，小皮球决定给同学们展示一下自己的皮肤，展示的思路是这样的：对于有皮肤的每一个英雄，随便选一个皮肤给同学看。

比如，小皮球共有 5 个英雄，这 5 个英雄分别有 0, 0, 3, 2, 4 款皮肤，那么，小皮球就有 $3 \times 2 \times 4 = 24$ 种展示的策略。

现在，小皮球希望自己的展示策略能够至少达到 M 种，请问，小皮球至少要花多少钱呢？

【输入描述】

第一行，两个整数 N, M 。

第二行， N 个整数，表示每个英雄的皮肤数量 K_i 。

第三行， N 个整数，表示每个英雄皮肤的价格 C_i 。

100% 的数据： $M \leq 10^{17}, 1 \leq K_i \leq 10, 1 \leq C_i \leq 199$ 。保证有解。

【输出描述】

一个整数，表示小皮球达到目标最少的花费。

【示例一】

输入：

3 24

4 4 4

2 2 2

输出：

18

【解法】

解法：动态规划 - 多重背包。

1. 状态表示：

这里有两种状态表示都能解决问题：

- a. $dp[i][j]$ 表示从前 i 个英雄中挑选，总方案数不低于 j 时，此时的最小花费；
- b. $dp[i][j]$ 表示从前 i 个英雄中挑选，总花费为 j 时，此时的最大方案数。

如果选第一种，时间和空间都吃不消，因为第二维的最大值是 10^{17} 。因此，我们选择第二种状态表示。在打表结束之后，从前往后扫描最后一行，找出第一个方案数超过 m 的花费就是结果。

2. 状态转移方程：

根据第 i 个英雄选择皮肤数量分类讨论，假设选了 p 个皮肤，其中 $0 \leq p \leq k[i]$ ，并且 $p \times c[i] \leq j$ ，此时的最大方案数为 $dp[i-1][j-p \times c[i]] \times p$ 。也就是前 $i-1$ 个英雄中挑选，总价值为 $j-p \times c[i]$ 时最大方案数再乘上此时的选择数量。

因为要的是最大值，所以状态转移方程就是所有合法 k 情况下的最大值。

3. 初始化：

$dp[0][0] = 1$ ，因为后续方案数是累乘，所以这里初始化为 1 就能保证后续填表是正确的。

【参考代码】

```
1  #include <iostream>
2
3  using namespace std;
4
5  typedef long long LL;
6
7  const int N = 300, M = 1e6 + 10;
8
9  LL n, m;
10 LL cnt[N], v[N];
11 LL f[M];
12 LL sum; // 最大花费
13
14 int main()
15 {
16     cin >> n >> m;
```

```

17     for(int i = 1; i <= n; i++) cin >> cnt[i];
18     for(int i = 1; i <= n; i++)
19     {
20         cin >> v[i];
21         sum += v[i] * cnt[i];
22     }
23
24     f[0] = 1;
25     for(int i = 1; i <= n; i++)
26     {
27         for(int j = sum; j >= 0; j--)
28         {
29             for(int k = 0; k <= cnt[i] && k * v[i] <= j; k++)
30             {
31                 f[j] = max(f[j], f[j - k * v[i]] * k);
32             }
33         }
34     }
35
36     for(int j = 1; j <= sum; j++)
37     {
38         if(f[j] >= m)
39         {
40             cout << j << endl;
41             break;
42         }
43     }
44
45     return 0;
46 }

```

Day17

1. 寻宝

题目来源：洛谷

题目链接： [P1076 \[NOIP 2012 普及组\] 寻宝](#)

【题目描述】

传说很遥远的藏宝楼顶层藏着诱人的宝藏。小明历尽千辛万苦终于找到传说中的这个藏宝楼，藏宝楼的门口竖着一个木板，上面写有几个大字：寻宝说明书。说明书的内容如下：

藏宝楼共有 $N + 1$ 层，最上面一层是顶层，顶层有一个房间里面藏着宝藏。除了顶层外，藏宝楼另有 N 层，每层 M 个房间，这 M 个房间围成一圈并按逆时针方向依次编号为 $0, \dots, M - 1$ 。其中一些房间有通往上一层的楼梯，每层楼的楼梯设计可能不同。每个房间里有一个指示牌，指示牌上有一个数字 x ，表示从这个房间开始按逆时针方向选择第 x 个有楼梯的房间（假定该房间的编号为 k ），从该房间上楼，上楼后到达上一层的 k 号房间。比如当前房间的指示牌上写着 2，则按逆时针方向开始尝试，找到第 2 个有楼梯的房间，从该房间上楼。如果当前房间本身就有楼梯通向上层，该房间作为第一个有楼梯的房间。

寻宝说明书的最后用红色大号字体写着：“寻宝须知：帮助你找到每层上楼房间的指示牌上的数字（即每层第一个进入的房间内指示牌上的数字）总和为打开宝箱的密钥”。

请帮助小明算出这个打开宝箱的密钥。

【输入描述】

第一行有两个整数 N 和 M ，之间用一个空格隔开。 N 表示除了顶层外藏宝楼共 N 层楼， M 表示除顶层外每层楼有 M 个房间。

接下来 $N \times M$ 行，每行两个整数，之间用一个空格隔开，每行描述一个房间内的情况，其中第 $(i - 1) \times M + j$ 行表示第 i 层 $j - 1$ 号房间的情况 ($i = 1, 2, \dots, N; j = 1, 2, \dots, M$)。第一个整数表示该房间是否有楼梯通往上一层（0 表示没有，1 表示有），第二个整数表示指示牌上的数字。注意，从 j 号房间的楼梯爬到上一层到达的房间一定也是 j 号房间。

最后一行，一个整数，表示小明从藏宝楼底层的几号房间进入开始寻宝（注：房间编号从 0 开始）。

对于 50% 数据，有 $0 < N \leq 1000; 0 \leq x \leq 10^4$ ；

对于 100% 数据，有 $0 < N \leq 10000; 0 < M \leq 100; 0 < x \leq 10^6$ 。

【输出描述】

一个整数，表示打开宝箱的密钥，这个数可能会很大，请输出对 20123 取模的结果即可。

【示例一】

输入：

2 3

1 2

0 3

1 4

0 1

1 5

1 2

1

输出：

5

【解法】

解法：模拟。

根据题意模拟整个爬楼的过程即可。

优化：

- 对于某一个房间的编号 x ，要找的是从这个位置开始第 x 个存在楼梯的房间编号。直接模拟会很大，可以先对该层的楼梯数取模，然后再去模拟。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2
3  using namespace std;
4
5  typedef long long LL;
6
7  const int N = 1e4 + 10, M = 110, MOD = 20123;
8
9  LL n, m;
10 bool st[N][M]; // 标记楼梯信息
11 LL x[N][M]; // 维护指示牌的信息
12
13 LL cnt[N]; // 第 i 层，有楼梯的房间个数
14
15 int main()
16 {
17     cin >> n >> m;
18     for(int i = 1; i <= n; i++)
19     {
20         for(int j = 0; j < m; j++)
21         {
22             int a, b; cin >> a >> b;
23             if(a)
24             {
25                 st[i][j] = true;
26                 cnt[i]++;
27             }
28             x[i][j] = b;
```

```

29     }
30 }
31
32 int pos = 0; cin >> pos;
33 LL ret = 0;
34 for(int i = 1; i <= n; i++)
35 {
36     ret = (ret + x[i][pos]) % MOD;
37
38     // 优化
39     LL step = x[i][pos] % cnt[i];
40     if(!step) step = cnt[i];
41
42     while(1)
43     {
44         if(st[i][pos]) step--;
45         if(step == 0) break;
46         pos++;
47         if(pos == m) pos = 0;
48     }
49 }
50
51 cout << ret << endl;
52
53 return 0;
54 }

```

2. 村村通

题目来源：洛谷

题目链接：[P1536 村村通](#)

【题目描述】

某市调查城镇交通状况，得到现有城镇道路统计表。表中列出了每条道路直接连通的城镇。市政府"村村通工程"的目标是使全市任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要相互之间可达即可）。请你计算出最少还需要建设多少条道路？

【输入描述】

输入包含若干组测试数据，每组测试数据的第一行给出两个用空格隔开的正整数，分别是城镇数目 n 和道路数目 m ；随后的 m 行对应 m 条道路，每行给出一对用空格隔开的正整数，分别是该条道路直接相连的两个城镇的编号。简单起见，城镇从 1 到 n 编号。

注意：两个城市间可以有多个道路相通。

在输入数据的最后，为一行一个整数 0，代表测试数据的结尾。

【输出描述】

对于每组数据，对应一行一个整数。表示最少还需要建设的道路数目。

【示例一】

输入：

4 2
1 3
4 3
3 3
1 2
1 3
2 3
5 2
1 2
3 5
999 0
0

输出：

1
0
2
998

【解法】

解法：图的性质 + 并查集。

根据图的结构，用并查集维护出联通块的个数 `cnt`，那么 `cnt - 1` 就是最终结果。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2
```

```

3  using namespace std;
4
5  const int N = 1010;
6
7  int n, m;
8  int fa[N];
9
10 int find(int x)
11 {
12     return fa[x] == x ? x : fa[x] = find(fa[x]);
13 }
14
15 void un(int x, int y)
16 {
17     fa[find(x)] = find(y);
18 }
19
20 int main()
21 {
22     while(cin >> n >> m)
23     {
24         for(int i = 1; i <= n; i++) fa[i] = i;
25
26         for(int i = 1; i <= m; i++)
27         {
28             int a, b; cin >> a >> b;
29             un(a, b);
30         }
31
32         int ret = 0;
33         for(int i = 1; i <= n; i++)
34             if(fa[i] == i)
35                 ret++;
36
37         cout << ret - 1 << endl;
38     }
39
40     return 0;
41 }

```

3. Diamond Collector

题目来源：洛谷

题目链接： [\[USACO16OPEN\] Diamond Collector S](#)

【题目描述】

奶牛 Bessie 很喜欢闪亮亮的东西 (Baling~ Baling~)，所以她喜欢在她的空余时间开采钻石！她现在已经收集了 N 颗不同大小的钻石 ($N \leq 50000$)，现在她想在谷仓的两个陈列架上摆放一些钻石。

Bessie 想让这些陈列架上的钻石保持相似的大小，所以她不会把两个大小相差 K 以上的钻石同时放在一个陈列架上（如果两颗钻石的大小差值不大于 K ，那么它们可以同时放在一个陈列架上）。现在给出 K ，请你帮 Bessie 确定她最多一共可以放多少颗钻石在这两个陈列架上。

【输入描述】

The first line of the input file contains N and K ($1 \leq K \leq 10^9$).

The next N lines each contain an integer giving the size of one of the diamonds.

All sizes will be positive and will not exceed 10^9 .

【输出描述】

Output a single positive integer, telling the maximum number of diamonds that Bessie can showcase in total in both cases.

【示例一】

输入：

7 3

10

5

1

12

9

5

14

输出：

5

【解法】

解法：预处理 + 滑动窗口。

错误的解法：先选一段长度「最大」的，再选一段长度「次大」的。但是这样是错误的，因为第一次的选择会影响第二次的选择，两者加起来「不一定是最优」的。比如： $[1, 1, 4, 5, 6, 7, 8, 10]$, $k = 3$ ：

- 如果先选 $[4, 5, 6, 7]$ ，接下来只能选 $[1, 1]$ 或者 $[8, 10]$ ，总长就是 6；
- 但是如果先选 $[5, 6, 7, 8]$ ，接下来可以选 $[1, 1, 4]$ ，总长就是 7。

因此，先选一段最长，再选一段次长的方法是错误的。

那么我们可以「枚举」所有的情况，以 i 位置为「分界点」，「左边」选一段，「右边」选一段：

- 左边选： $[1, i - 1]$ 区间内，符合要求的「最长子串」的长度。
- 右边选： $[i, n]$ 区间内，符合要求的「最长子串」的长度。

这样我们就可以枚举出所有的情况，「左右两部分相加」的最大值就是结果。

接下来考虑，如何快速找到 $[1, i - 1]$ 区间内，符合要求的「最长子串」的长度以及 $[i, n]$ 区间内，符合要求的「最长子串」的长度，小的动态规划预处理：

- 定义数组 $f[i]$ 表示 $[1, i]$ 区间的最大长度， $g[i]$ 表示 $[i, n]$ 区间的最大长度；
- 对于 $f[i]$ ，先找出 $[1, i - 1]$ 区间的最大长度，然后再找到「以 $a[i]$ 为结尾位置」的最长子串的长度，两者最大值即可；
- 对于 $g[i]$ ，先找出 $[i + 1, n]$ 区间的最大长度，然后再找到「以 $a[i]$ 为起始位置」的最长子串的长度，两者最大值即可。

如何找出「以 $a[i]$ 为结尾位置」的最长子串的长度：

- 在滑动窗口的过程中，我们每次找到一段「符合要求的子串」时，都可以知道这段子串的终止位置 $right$ ；
- 因此，做一次「滑动窗口」，就可以把所有位置的信息都「预处理」出来。

「以 $a[i]$ 为起始位置」的最长子串的长度，倒着再来一次即可。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 5e4 + 10;
7
8  int n, k;
9  int a[N];
```

```

10  int f[N], g[N];
11
12  int main()
13  {
14      cin >> n >> k;
15      for(int i = 1; i <= n; i++) cin >> a[i];
16      sort(a + 1, a + 1 + n);
17
18      // 预处理 [1, i]
19      for(int left = 1, right = 1; right <= n; right++)
20      {
21          while(a[right] - a[left] > k) left++;
22
23          f[right] = max(f[right - 1], right - left + 1);
24      }
25
26      // 预处理 [i, n]
27      for(int left = n, right = n; left >= 1; left--)
28      {
29          while(a[right] - a[left] > k) right--;
30
31          g[left] = max(g[left + 1], right - left + 1);
32      }
33
34      int ret = 0;
35      for(int i = 2; i <= n; i++)
36      {
37          ret = max(ret, f[i - 1] + g[i]);
38      }
39
40      cout << ret << endl;
41
42      return 0;
43  }

```

4. Apple Catching

题目来源：洛谷

题目链接： [P2690 \[USACO04NOV\] Apple Catching G](#)

【题目描述】

很少有人知道奶牛爱吃苹果。农夫约翰的农场上有两棵苹果树（编号为 1 和 2），每一棵树上都长满了苹果。奶牛贝茜无法摘下树上的苹果，所以她只能等待苹果从树上落下。但是，由于苹果掉到地

上会摔烂，贝茜必须要在半空中接住苹果（没有人爱吃摔烂的苹果）。贝茜吃东西很快，她接到苹果后仅用几秒钟就能吃完。每一分钟，两棵苹果树其中的一棵会掉落一个苹果。贝茜已经过了足够的训练，只要站在树下就一定能接住这棵树上掉落的苹果。同时，贝茜能够在两棵树之间快速移动（移动时间远少于 1 分钟），因此当苹果掉落时，她必定站在两棵树其中的一棵下面。此外，奶牛不愿意不停地往返于两棵树之间，因此会错过一些苹果。苹果每分钟掉落一个，共 T ($1 \leq T \leq 1000$) 分钟，贝茜最多愿意移动 W ($1 \leq W \leq 30$) 次。现给出每分钟掉落苹果的树的编号，要求判定贝茜能够接住的最多苹果数。开始时贝茜在 1 号树下。

【输入描述】

第一行 2 个数， T 和 W 。接下来的 t 行，每行一个数，代表在时刻 t 苹果是从 1 号苹果树还是从 2 号苹果树上掉下来的。

【输出描述】

对于每个测试点，输出一行，一个数，为奶牛最多接到的苹果的数量。

【示例一】

输入：

7 2

2

1

1

2

2

1

1

输出：

6

【解法】

解法：线性 dp。

1. 状态表示：

$f[i][j]$ 表示：当时间为 i 时，移动次数为 j 时，能拿到的最大分数。

2. 状态转移方程：

先计算当前这个时刻，移动 j 次之后，能否拿到苹果：

- 如果 `j % 2 == 0 && a[i] == 1` 或者 `j % 2 == 1 && a[i] == 2`，那就能拿到，`c = 1`；
- 否则拿不到，`c = 0`。

- 当前这个时刻和上一个时刻的位置不同： $f[i-1][j-1] + c$ ；
- 当前这个时刻和上一个时刻的位置一样： $f[i-1][j] + c$ 。

3. 最终结果：

$f[n]$ 这一行中的最大值。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int N = 1010, M = 35;
6
7  int n, m;
8  int a[N];
9  int f[N][M];
10
11 int main()
12 {
13     cin >> n >> m;
14     for(int i = 1; i <= n; i++) cin >> a[i];
15
16     for(int i = 1; i <= n; i++)
17     {
18         for(int j = 0; j <= m; j++)
19         {
20             int c = 0;
21             if(j % 2 == 0 && a[i] == 1 || (j % 2 == 1 && a[i] == 2)) c = 1;
22
23             f[i][j] = f[i-1][j] + c;
24             if(j) f[i][j] = max(f[i][j], f[i-1][j-1] + c);
25         }
26     }
27
28     int ret = 0;
```

```
29     for(int j = 0; j <= m; j++)
30     {
31         ret = max(f[n][j], ret);
32     }
33
34     cout << ret << endl;
35
36     return 0;
37 }
```

Day18

1. 天天爱跑步

题目来源：洛谷

题目链接：[B3655 \[语言月赛202208\] 天天爱跑步](#)

【题目描述】

为了推广《天天爱跑步》，洛咕公司开发了活跃值系统，同其他游戏一样，连续签到天数越多，每次签到获得的活跃值也就越多。但只要一天不签到，连续天数就要清零。

当连续签到天数达到以下天数时，活跃值奖励将发生如下变化。

- 1 天：由 0 变为 v_1
- 3 天：由 v_1 变为 v_3
- 7 天：由 v_3 变为 v_7
- 30 天：由 v_7 变为 v_{30}
- 120 天：由 v_{30} 变为 v_{120}
- 365 天：由 v_{120} 变为 v_{365}
- 366 天或更多天数：均为 v_{365}

例如，以下为某游戏玩家连续签到情况及活跃值奖励情况，有助于进一步理解题意。

日期	是否签到	连续签到天数	当天获得活跃值奖励
8月8日	是	1	v_1
8月9日	是	2	v_1
8月10日	是	3	v_3
8月11日	是	4	v_3
8月12日	否	0	0
8月13日	是	1	v_1
8月14日	是	2	v_1
8月15日	是	3	v_3

即，签到系统按照如下顺序处理奖励：

1. 累计连续签到天数
2. 根据规则确定当日奖励的活跃值
3. 发放活跃值

现在给出 超高校级的游戏玩家 ☒七海千秋☒ 连续 n 天的签到情况，请你求出这 n 天七海千秋获得的活跃值奖励一共为多少。

连续签到天数从 0 开始计算。

【输入描述】

输入共 $n+2$ 行。

输入的第一行为一个正整数 n ，代表 ☒七海千秋☒ 游戏的天数。

输入的第二行为六个正整数，分别为 $v_1, v_3, v_7, v_{30}, v_{120}, v_{365}$ ，含义如题。

接下来的 n 行，每行一个整数 1 或 0。其中第 i 行为 1 表示第 i 天 ☒七海千秋☒ 签到了，输入为 0 表示第 i 天 ☒七海千秋☒ 没有签到。

【输出描述】

输出一行一个整数，代表 ☒七海千秋☒ n 天签到所获得的活跃值。

【示例一】

输入：

12

1 2 3 4 5 6

1

1

1

1

1

0

0

0

1

0

1

1

输出：

11

【解法】

解法：模拟。

模拟计算分数的流程即可。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2
3  using namespace std;
4
5  int n;
6  int v[10];
7
8  int main()
9  {
10     cin >> n;
11     for(int i = 1; i <= 6; i++) cin >> v[i];
```

```

12
13     int sum = 0, d = 0;
14     for(int i = 1; i <= n; i++)
15     {
16         int x; cin >> x;
17         if(x) d++;
18         else d = 0;
19
20         int y = 0;
21         if(d == 0) y = 0;
22         else if(d < 3) y = v[1];
23         else if(d < 7) y = v[2];
24         else if(d < 30) y = v[3];
25         else if(d < 120) y = v[4];
26         else if(d < 365) y = v[5];
27         else y = v[6];
28
29         sum += y;
30     }
31
32     cout << sum << endl;
33
34     return 0;
35 }

```

2. Subsequences Summing to Sevens

题目来源：洛谷

题目链接： [P3131 \[USACO16JAN\] Subsequences Summing to Sevens S](#)

【题目描述】

Farmer John 的 N 头奶牛站成一排，这是它们时不时会做的事情。每头奶牛都有一个独特的整数 ID 编号，以便 Farmer John 能够区分它们。Farmer John 希望为一组连续的奶牛拍照，但由于童年时与数字 $1 \cdots 6$ 相关的创伤事件，他只希望拍摄一组奶牛，如果它们的 ID 加起来是 7 的倍数。

请帮助 Farmer John 确定他可以拍摄的最大奶牛组的大小。

【输入描述】

输入的第一行包含 N ($1 \leq N \leq 50,000$)。接下来的 N 行每行包含一头奶牛的整数 ID（所有 ID 都在 $0 \cdots 1,000,000$ 范围内）。

【输出描述】

请输出 ID 之和为 7 的倍数的最大连续奶牛组中的奶牛数量。如果不存在这样的组，则输出 0。

【示例一】

输入：

7
3
5
1
6
2
14
10

输出：

5

【解法】

解法：前缀和 + 同余。

对于前缀和 $f[i]$ ，仅需找到 $[1, i - 1]$ 区间内，前缀和模 7 等于 $f[i] \% 7$ 的最左位置 j 。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2  #include <cstring>
3
4  using namespace std;
5
6  const int N = 10;
7
8  int n;
9  int id[N];
10
11 int main()
12 {
13     cin >> n;
14     memset(id, -1, sizeof id);
15     id[0] = 0;
16
```

```

17     int sum = 0, ret = 0;
18     for(int i = 1; i <= n; i++)
19     {
20         int x; cin >> x;
21         sum = (sum + x) % 7;
22
23         if(id[sum] != -1) ret = max(ret, i - id[sum]);
24         else id[sum] = i;
25     }
26
27     cout << ret << endl;
28
29     return 0;
30 }

```

3. 图的遍历

题目来源：洛谷

题目链接：P3916 图的遍历

【题目描述】

给出 N 个点， M 条边的有向图，对于每个点 v ，求 $A(v)$ 表示从点 v 出发，能到达的编号最大的点。

【输入描述】

第 1 行 2 个整数 N, M ，表示点数和边数。

接下来 M 行，每行 2 个整数 U_i, V_i ，表示边 (U_i, V_i) 。点用 $1, 2, \dots, N$ 编号。

【输出描述】

一行 N 个整数 $A(1), A(2), \dots, A(N)$ 。

【示例一】

输入：

4 3

1 2

2 4

4 3

输出：

4 4 3 4

【解法】

解法：正难则反 - 反图。

从任意一点出发，去找能到达的最大结点，时间复杂度巨高。但是如果对原图建一个反图，从节点编号最大的点出发，能走到的点都是原图能到达的点。

因此，建一个反图，按照节点编号从大到小搜索一遍即可。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  const int N = 1e5 + 10;
7
8  int n, m;
9  vector<int> edges[N];
10
11 int ret[N];
12
13 void dfs(int u, int r)
14 {
15     ret[u] = r;
16
17     for(auto v : edges[u])
18     {
19         if(ret[v]) continue;
20         dfs(v, r);
21     }
22 }
23
24 int main()
25 {
26     cin >> n >> m;
27     for(int i = 1; i <= m; i++)
28     {
29         int a, b; cin >> a >> b;
30         // 建反图
31         edges[b].push_back(a);
32     }
33
34     for(int i = n; i >= 1; i--)
```

```

35     {
36         if(ret[i]) continue;
37         dfs(i, i);
38     }
39
40     for(int i = 1; i <= n; i++)
41     {
42         cout << ret[i] << " ";
43     }
44
45     return 0;
46 }

```

4. Space Elevator

题目来源：洛谷

题目链接：[P6771 \[USACO05MAR\] Space Elevator 太空电梯](#)

【题目描述】

奶牛们要去太空了！它们打算用方块建造一座太空电梯。现在它们有 N 种方块，第 i 种方块有一个特定的高度 h_i ，一定的数量 c_i 。为了防止宇宙射线破坏方块，第 i 种方块的任何部分不能超过高度 a_i 。

请用这些方块堆出最高的太空电梯。

【输入描述】

第一行，一个整数 N ；

第二行到 $N+1$ 行，第 $i+1$ 行三个整数 h_i, a_i, c_i ，数字之间用空格分隔。

对于 100% 的数据： $1 \leq N \leq 400, 1 \leq h_i \leq 100, 1 \leq c_i \leq 10, 1 \leq a_i \leq 4 \times 10^4$ 。

【输出描述】

共一行，一个整数，为太空电梯的高度。

【示例一】

输入：

3

7 40 3

5 23 8

2 52 6

输出：

48

【解法】

解法：贪心 + 动态规划。

一定要先搞懂题意！

贪心：当我们从前往后考虑每一个方块的时候，限定高度 $a[i]$ 小的应该优先考虑。因为如果先放限定高度大的，这些限定高度小的就没法放了。因此，先对所有的方块按照限定高度 $a[i]$ 从小到大排序。

接下来的问题就是挑一些方块出来，在不超过每一种方块的限定高度下，看看能堆成的最大高度是多少。正好是多重背包问题。

1. 状态表示：

$dp[i][j]$ 表示：从前 i 个方块中挑选，总高度不超过 j 的情况下，最大的高度是多少。

（这道题也可以定义 *bool* 类型的状态表示：从前 i 个方块中挑选，是否能凑成高度为 j ，可以尝试一下）

那么整个 dp 表中的最大值，就是我们要的结果。这里要注意，并不是 $dp[n][m]$ ，因为有可能考虑不到第 n 个方块根本考虑不进去，最后一行根本就不会更新。

2. 状态转移方程：

根据第 i 个方块选的数量，可以分成 $c[i] + 1$ 种情况，要的是所有情况的最大值。设选了 k 个方块，那么最大高度为 $dp[i - 1][j - k \times h[i]] + k \times h[i]$ 。

注意限定条件，循环高度的时候不能超过 $a[i]$ ，并且 $j - k \times h[i] \geq 0$ 。

【参考代码】

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 410, M = 4e4 + 10;
7
8  int n;
9  struct node
10 {
```

```

11     int h, a, c;
12 }e[N];
13
14 int f[M];
15
16 bool cmp(node& x, node& y)
17 {
18     return x.a < y.a;
19 }
20
21 int main()
22 {
23     cin >> n;
24     for(int i = 1; i <= n; i++) cin >> e[i].h >> e[i].a >> e[i].c;
25
26     sort(e + 1, e + 1 + n, cmp);
27
28     int ret = 0;
29     for(int i = 1; i <= n; i++)
30     {
31         int h = e[i].h, a = e[i].a, c = e[i].c;
32         for(int j = a; j >= 0; j--)
33         {
34             for(int k = 0; k <= c && k * h <= j; k++)
35             {
36                 f[j] = max(f[j], f[j - k * h] + k * h);
37             }
38             ret = max(ret, f[j]);
39         }
40     }
41
42     cout << ret << endl;
43
44     return 0;
45 }

```

Day19

1. 跳跳!

题目来源：洛谷

题目链接： [P4995 跳跳!](#)

【题目描述】

你是一只小跳蛙，你特别擅长在各种地方跳来跳去。

这一天，你和朋友小 F 一起出去玩耍的时候，遇到了一堆高矮不同的石头，其中第 i 块的石头高度为 h_i ，地面的高度是 $h_0 = 0$ 。你估计着，从第 i 块石头跳到第 j 块石头上耗费的体力值为 $(h_i - h_j)^2$ ，从地面跳到第 i 块石头耗费的体力值是 $(h_i)^2$ 。

为了给小 F 展现你超级跳的本领，你决定跳到每个石头上各一次，并最终停在任意一块石头上，并且小跳蛙想耗费尽可能多的体力值。

当然，你只是一只小跳蛙，你只会跳，不知道怎么跳才能让本领更充分地展现。

不过你有救啦！小 F 给你递来了一个写着 AK 的电脑，你可以使用计算机程序帮你解决这个问题，万能的计算机告诉你怎么跳。

那就请你——会写代码的小跳蛙——写下这个程序，为你 NOIp AK 踏出坚实的一步吧！

【输入描述】

输入一行一个正整数 n ，表示石头个数。

输入第二行 n 个正整数，表示第 i 块石头的高度 h_i 。

对于 $1 \leq i \leq n$ ，有 $0 < h_i \leq 10^4$ ，且保证 h_i 互不相同。

对于 10% 的数据， $n \leq 3$ ；

对于 20% 的数据， $n \leq 10$ ；

对于 50% 的数据， $n \leq 20$ ；

对于 80% 的数据， $n \leq 50$ ；

对于 100% 的数据， $n \leq 300$ 。

【输出描述】

输出一行一个正整数，表示你可以耗费的体力值的最大值。

【示例一】

输入：

2

2 1

输出：

5

【示例二】

输入：

3

6 3 5

输出：

49

【解法】

解法：贪心。

- 每次都跳距离当前位置最远的位置，也就是排完序之后，一左一右地跳。

【参考代码】

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  typedef long long LL;
7
8  const int N = 310;
9
10 int n;
11 LL h[N];
12
13 int main()
14 {
15     cin >> n;
16     for(int i = 1; i <= n; i++) cin >> h[i];
17
18     sort(h + 1, h + 1 + n);
19
20     int l = 0, r = n;
21     LL sum = 0;
22     while(l < r)
23     {
24         sum += (h[l] - h[r]) * (h[l] - h[r]);
25         l++;
26         sum += (h[l] - h[r]) * (h[l] - h[r]);
27         r--;
28     }
29
30     cout << sum << endl;
31
32     return 0;
33 }
```


2. 数列分段 Section II

题目来源：洛谷

题目链接：[P1182 数列分段 Section II](#)

【题目描述】

对于给定的一个长度为 N 的正整数数列 $A_{1\sim N}$ ，现要将其分成 M ($M \leq N$) 段，并要求每段连续，且每段和的最大值最小。

关于最大值最小：

例如一数列 4 2 4 5 1 要分成 3 段。

将其如下分段：

[4 2][4 5][1]

第一段和为 6，第 2 段和为 9，第 3 段和为 1，和最大值为 9。

将其如下分段：

[4][2 4][5 1]

第一段和为 4，第 2 段和为 6，第 3 段和为 6，和最大值为 6。

并且无论如何分段，最大值不会小于 6。

所以可以得到要将数列 4 2 4 5 1 要分成 3 段，每段和的最大值最小为 6。

【输入描述】

第 1 行包含两个正整数 N, M 。

第 2 行包含 N 个空格隔开的非负整数 A_i ，含义如题目所述。

对于 20% 的数据， $N \leq 10$ 。

对于 40% 的数据， $N \leq 1000$ 。

对于 100% 的数据， $1 \leq N \leq 10^5$ ， $M \leq N$ ， $A_i < 108$ ，答案不超过 10^9 。

【输出描述】

一个正整数，即每段和最大值最小为多少。

【示例一】

输入：

5 3

4 2 4 5 1

输出：

6

【解法】

解法：二分答案。

题目中有很明显的题眼：最大值最小，并且也有明显的二段性：

- 当分的段数越多时的时候，最大的和越小；
- 当分的段数越少的时候，最大的和越大。

因此，可以用二分答案来解决。

关于 calc 函数，传入一个和 x ，求出最少能分多少段：

- 从前往后累加，只要和小于 x ，就一直加；
- 直到和超过 x ，之前的为一段，然后从该位置继续累加。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2
3  using namespace std;
4
5  typedef long long LL;
6
7  const int N = 1e5 + 10;
8
9  int n, m;
10 LL a[N];
11
12 int calc(LL x)
13 {
14     int cnt = 0, sum = 0;
15     for(int i = 1; i <= n; i++)
16     {
17         sum += a[i];
18         if(sum > x)
19         {
20             cnt++;
21             sum = a[i];
22         }
```

```

23     }
24     return cnt + 1;
25 }
26
27 int main()
28 {
29     cin >> n >> m;
30     LL l = 0, r = 0;
31     for(int i = 1; i <= n; i++)
32     {
33         cin >> a[i];
34         l = max(l, a[i]); r += a[i];
35     }
36
37     while(l < r)
38     {
39         LL mid = (l + r) / 2;
40         if(calc(mid) <= m) r = mid;
41         else l = mid + 1;
42     }
43
44     cout << l << endl;
45
46     return 0;
47 }

```

3. 修理牛棚

题目来源：洛谷

题目链接：[P1209 \[USACO1.3\] 修理牛棚 Barn Repair](#)

【题目描述】

在一个月黑风高的暴风雨夜，Farmer John 的牛棚的屋顶、门被吹飞了。好在许多牛正在度假，所以牛棚没有住满。

牛棚一个紧挨着另一个被排成一行，牛就住在里面过夜。有些牛棚里有牛，有些没有。所有的牛棚有相同的宽度。

自门遗失以后，Farmer John 必须尽快在牛棚之前竖立起新的木板。他的新木材供应商将会供应他任何他想要的长度，但是吝啬的供应商只能提供有限数目的木板。Farmer John 想将他购买的木板总长度减到最少。

给出 m, s, c ，表示木板最大的数目、牛棚的总数、牛的总数；以及每头牛所在牛棚的编号，请算出拦住所有有牛的牛棚所需木板的最小总长度。

【输入描述】

一行三个整数 m, s, c ，意义如题目描述。

接下来 c 行，每行包含一个整数，表示牛所占的牛棚的编号。

【输出描述】

输出一行一个整数，表示所需木板的最小总长度。

【示例一】

输入：

4 50 18

3

4

6

8

14

15

16

17

21

25

26

27

30

31

40

41

42

43

输出：

25

【解法】

解法：正难则反 + 贪心。

可以先来一块足够长的木板挡住所有的牛棚，然后按照挖出 $m - 1$ 个最长的窟窿。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 210;
7
8  int m, s, c;
9  int a[N];
10 int b[N]; // 存间隔
11
12 bool cmp(int a, int b)
13 {
14     return a > b;
15 }
16
17 int main()
18 {
19     cin >> m >> s >> c;
20     for(int i = 1; i <= c; i++) cin >> a[i];
21
22     sort(a + 1, a + 1 + c);
23
24     for(int i = 1; i < c; i++)
25     {
26         b[i] = a[i + 1] - a[i] - 1;
27     }
28
29     sort(b + 1, b + 1 + c, cmp);
30
31     int ret = a[c] - a[1] + 1;
32     for(int i = 1; i < m && i < c; i++)
33     {
34         ret -= b[i];
35     }
36
37     cout << ret << endl;
38
39     return 0;
40 }
```

4. 货币系统

题目来源：洛谷

题目链接：[P5020 \[NOIP2018 提高组\] 货币系统](#)

【题目描述】

在网友的国度中共有 n 种不同面额的货币，第 i 种货币的面额为 $a[i]$ ，你可以假设每一种货币都有无穷多张。为了方便，我们把货币种数为 n 、面额数组为 $a[1..n]$ 的货币系统记作 (n, a) 。

在一个完善的货币系统中，每一个非负整数的金额 x 都应该可以被表示出，即对每一个非负整数 x ，都存在 n 个非负整数 $t[i]$ 满足 $a[i] \times t[i]$ 的和为 x 。然而，在网友的国度中，货币系统可能是不完善的，即可能存在金额 x 不能被该货币系统表示出。例如在货币系统 $n = 3, a = [2, 5, 9]$ 中，金额 $1, 3$ 就无法被表示出来。

两个货币系统 (n, a) 和 (m, b) 是等价的，当且仅当对于任意非负整数 x ，它要么均可以被两个货币系统表出，要么不能被其中任何一个表出。

现在网友们打算简化一下货币系统。他们希望找到一个货币系统 (m, b) ，满足 (m, b) 与原来的货币系统 (n, a) 等价，且 m 尽可能的小。他们希望你来协助完成这个艰巨的任务：找到最小的 m 。

【输入描述】

输入文件的第一行包含一个整数 T ，表示数据的组数。

接下来按照如下格式分别给出 T 组数据。每组数据的第一行包含一个正整数 n 。接下来一行包含 n 个由空格隔开的正整数 $a[i]$ 。

对于 100% 的数据，满足 $1 \leq T \leq 20, n, a[i] \geq 1$ 。

【输出描述】

输出文件共有 T 行，对于每组数据，输出一行一个正整数，表示所有与 (n, a) 等价的货币系统 (m, b) 中，最小的 m 。

【示例一】

输入：

2

4

3 19 10 6

5

11 29 13 19 17

输出：

2

5

【解法】

首先搞懂题意：就是在这一堆数中挑选出来最少的一些数，使的这些数能组合出所有的数。如果想选出最少的数，可以简单归纳出来两个性质：

1. 较大的数只能由较小的一些数组合出来，因此我们可以先对原数组排序；
2. 如果一个数 $a[i]$ 能被 $[1, i - 1]$ 区间内的数表示，那么就可以舍去；如果不能，那就必须保留。

那解法就显而易见了，先对整个数组排序，然后从前往后做完全背包问题。状态表示为从前 i 个数中挑选，是否能凑成整数 j 。在循环到 $a[i]$ 的时候，去看看 $dp[i - 1][a[i]]$ 是否是 *true*。

完全背包的逻辑就不再赘述了，做了那么多背包问题，分析起来应该很简单了~

【参考代码】

```
1  #include <iostream>
2  #include <algorithm>
3  #include <cstring>
4
5  using namespace std;
6
7  const int N = 110, M = 25010;
8
9  int n;
10 int a[N];
11 bool f[M];
12
13 void solve()
14 {
15     cin >> n;
16     for(int i = 1; i <= n; i++) cin >> a[i];
17     sort(a + 1, a + 1 + n);
18
19     memset(f, 0, sizeof f);
20     f[0] = true;
21
22     int ret = 0;
23     for(int i = 1; i <= n; i++)
24     {
25         if(!f[a[i]]) ret++;
```

```

26         for(int j = a[i]; j <= a[n]; j++)
27         {
28             f[j] = f[j] || f[j - a[i]];
29         }
30     }
31
32     cout << ret << endl;
33 }
34
35 int main()
36 {
37     int T; cin >> T;
38     while(T--)
39     {
40         solve();
41     }
42
43     return 0;
44 }

```

Day20

1. 生活大爆炸版石头剪刀布

题目来源：洛谷

题目链接：[P1328 \[NOIP 2014 提高组\] 生活大爆炸版石头剪刀布](#)

【题目描述】

石头剪刀布是常见的猜拳游戏：石头胜剪刀，剪刀胜布，布胜石头。如果两个人出拳一样，则不分胜负。在《生活大爆炸》第二季第8集中出现了一种石头剪刀布的升级版游戏。

升级版游戏在传统的石头剪刀布游戏的基础上，增加了两个新手势：

斯波克：《星际迷航》主角之一。

蜥蜴人：《星际迷航》中的反面角色。

这五种手势的胜负关系如表一所示,表中列出的是甲对乙的游戏结果。

甲 \ 乙 甲对乙的结果	剪刀	石头	布	蜥蜴人	斯波克
剪刀	平	输	赢	赢	输
石头		平	输	赢	输
布			平	输	赢
蜥蜴人				平	赢
斯波克					平

现在，小 A 和小 B 尝试玩这种升级版的猜拳游戏。已知他们的出拳都是有周期性规律的，但周期长度不一定相等。例如：如果小 A 以 石头-布-石头-剪刀-蜥蜴人-斯波克 长度为 6 的周期出拳,那么他的出拳序列就是 石头-布-石头-剪刀-蜥蜴人-斯波克-石头-布-石头-剪刀-蜥蜴人-斯波克-...，而如果小 B 以 剪刀-石头-布-斯波克-蜥蜴人 长度为 5 的周期出拳,那么他出拳的序列就是 剪刀-石头-布-斯波克-蜥蜴人-剪刀-石头-布-斯波克-蜥蜴人-...。

已知小 A 和小 B 一共进行 N 次猜拳。每一次赢的人得 1 分，输的得 0 分；平局两人都得 0 分。现请你统计 N 次猜拳结束之后两人的得分。

【输入描述】

第一行包含三个整数： N, NA, NB ，分别表示共进行 N 次猜拳、小 A 出拳的周期长度，小 B 出拳的周期长度。数与数之间以一个空格分隔。

第二行包含 NA 个整数,表示小 A 出拳的规律,第三行包含 NB 个整数，表示小 B 出拳的规律。其中，0 表示 剪刀，1 表示 石头，2 表示 布，3 表示 蜥蜴人，4 表示 斯波克。数与数之间以一个空格分隔。

【输出描述】

输出一行，包含两个整数，以一个空格分隔，分别表示小 A、小 B 的得分。

【示例一】

输入：

10 5 6

0 1 2 3 4

0 3 4 2 1 0

输出：

6 2

【解法】

解法：模拟。

- 模拟整个剪刀石头布的过程即可。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int N = 210;
6
7  int n, n1, n2;
8  int a[N], b[N];
9
10 int c[5][5] = {
11     0, -1, 1, 1, -1,
12     1, 0, -1, 1, -1,
13     -1, 1, 0, -1, 1,
14     -1, -1, 1, 0, 1,
15     1, 1, -1, -1, 0
16 };
17
18 int main()
19 {
20     cin >> n >> n1 >> n2;
21     for(int i = 0; i < n1; i++) cin >> a[i];
22     for(int i = 0; i < n2; i++) cin >> b[i];
23
24     int A = 0, B = 0;
25     for(int i = 0; i < n; i++)
26     {
27         int x = i % n1, y = i % n2;
28         int t = c[a[x]][b[y]];
29
30         if(t > 0) A++;
31         else if(t < 0) B++;
32     }
33
34     cout << A << " " << B << endl;
35
36     return 0;
37 }
```

2. 语文成绩

题目来源：洛谷

题目链接： [P2367 语文成绩](#)

【题目描述】

语文老师总是写错成绩，所以当她修改成绩的时候，总是累得不行。她总是要一遍遍地给某些同学增加分数，又要注意最低分是多少。你能帮帮她吗？

【输入描述】

第一行有两个整数 n, p ，代表学生数与增加分数的次数。

第二行有 n 个数， $a_1 \sim a_n$ ，代表各个学生的初始成绩。

接下来 p 行，每行有三个数， x, y, z ，代表给第 x 个到第 y 个学生每人增加 z 分。

对于 40% 的数据，有 $n \leq 10^3$ 。

对于 60% 的数据，有 $n \leq 10^4$ 。

对于 80% 的数据，有 $n \leq 10^5$ 。

对于 100% 的数据，有 $n \leq 5 \times 10^6, p \leq n$ ，学生初始成绩 $\leq 100, z \leq 100$ 。

【输出描述】

输出仅一行，代表更改分数后，全班的最低分。

【示例一】

输入：

3 2

1 1 1

1 2 1

2 3 1

输出：

2

【解法】

解法：差分。

- 差分模板题换了一个外壳~

【参考代码】

▼ 代码块

```
1  #include <iostream>
2
```

```

3  using namespace std;
4
5  const int N = 5E6 + 10;
6
7  int n, p;
8  int f[N];
9
10 int main()
11 {
12     cin >> n >> p;
13     for(int i = 1; i <= n; i++)
14     {
15         int x; cin >> x;
16         f[i] += x; f[i + 1] -= x;
17     }
18
19     while(p--)
20     {
21         int x, y, z; cin >> x >> y >> z;
22         f[x] += z; f[y + 1] -= z;
23     }
24
25     int ret = 1e9;
26     for(int i = 1; i <= n; i++)
27     {
28         f[i] += f[i - 1];
29         ret = min(ret, f[i]);
30     }
31
32     cout << ret << endl;
33
34     return 0;
35 }

```

3. 花匠

题目来源：洛谷

题目链接：[P1970 \[NOIP 2013 提高组\] 花匠](#)

【题目描述】

花匠栋栋种了一排花，每株花都有自己的高度。花儿越长越大，也越来越挤。栋栋决定把这排中的一部分花移走，将剩下的留在原地，使得剩下的花能有空间长大，同时，栋栋希望剩下的花排列得比较别致。

具体而言，栋栋的花的高度可以看成一系列整数 h_1, h_2, \dots, h_n 。设当一部分花被移走后，剩下的花的高度依次为 g_1, g_2, \dots, g_m ，则栋栋希望下面两个条件中至少有一个满足：

条件 A：对于所有的 $1 \leq i \leq \frac{m}{2}$ ，有 $g_{2i} > g_{2i-1}$ ，同时对于所有的 $1 \leq i \leq \frac{m}{2}$ ，有 $g_{2i} > g_{2i+1}$ ；

条件 B：对于所有的 $1 \leq i \leq \frac{m}{2}$ ，有 $g_{2i} < g_{2i-1}$ ，同时对于所有的 $1 \leq i \leq \frac{m}{2}$ ，有 $g_{2i} < g_{2i+1}$ 。

注意上面两个条件在 $m = 1$ 时同时满足，当 $m > 1$ 时最多有一个能满足。

请问，栋栋最多能将多少株花留在原地。

【输入描述】

第一行包含一个整数 n ，表示开始时花的株数。

第二行包含 n 个整数，依次为 h_1, h_2, \dots, h_n ，表示每株花的高度。

对于 20% 的数据， $n \leq 10$ ；

对于 30% 的数据， $n \leq 25$ ；

对于 70% 的数据， $n \leq 1000$ ， $0 \leq h_i \leq 1000$ ；

对于 100% 的数据， $1 \leq n \leq 10^5$ ， $0 \leq h_i \leq 10^6$ ，所有的 h_i 随机生成，所有随机数服从某区间内的均匀分布。

【输出描述】

输出一行，包含一个整数，表示最多能留在原地的花的株数。

【示例一】

输入：

5

5 3 2 1 2

输出：

3

说明：

有多种方法可以正好保留 3 株花，例如，留下第 1、4、5 株，高度分别为 5、1、2，满足条件 B。

【解法】

解法：贪心。

对于某一个位置来说：

- 如果接下来呈现上升趋势的话，我们让其上升到波峰的位置；

- 如果接下来呈现下降趋势的话，我们让其下降到波谷的位置。

因此，如果把整个数组放在「折线图」中，我们统计出所有的波峰以及波谷的个数即可。

【参考代码】

▼ 代码块

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int N = 1e5 + 10;
6
7  int n;
8  int h[N];
9
10 int main()
11 {
12     cin >> n;
13     for(int i = 1; i <= n; i++) cin >> h[i];
14
15     int prev = 0, cnt = 0;
16     for(int i = 1; i < n; i++)
17     {
18         int d = h[i + 1] - h[i];
19         if(d == 0) continue;
20
21         d = (d > 0 ? 1 : -1);
22         if(d != prev) cnt++;
23         prev = d;
24     }
25
26     cout << cnt + 1 << endl;
27
28     return 0;
29 }
```

4. Zuma

题目来源：洛谷

题目链接：[Zuma](#)

【题目描述】

Genos 最近在他的手机上下载了祖玛游戏。在祖玛游戏里，存在 n 个一行的宝石，第 i 个宝石的颜色是 C_i 。这个游戏的目标是尽快的消灭一行中所有的宝石。

在一秒钟，Genos 能很快的挑选出这些有颜色的宝石中的一个回文的、连续的子串，并将这个子串移除。每当一个子串被删除后，剩余的宝石将连接在一起，形成一个新的行列。

你的任务是：求出把整个宝石串都移除的最短时间。

【输入描述】

第一行包含一个整数 $n(1 \leq n \leq 500)$ ，表示宝石串的长度。

第二行包含 n 个被空格分开的整数，第 $i(1 \leq i \leq n)$ 个表示这行中第 i 个珠子的颜色。

【输出描述】

输出一个整数，把这行珠子移除的最短时间。

【示例一】

输入：

3

1 2 1

输出：

1

样例说明：

Genos 可以在一秒钟就把这行珠子全部移走。

【示例二】

输入：

3

1 2 3

输出：

3

样例说明：

Genos 一次只能移走一个珠子，所以移走三个珠子花费他三秒。

【示例三】

输入：

7

1 4 4 2 3 2 1

输出：

样例说明：

可以达到 2 秒的最快时间，先移除回文串 4 4,再移除回文串 1 2 3 2 1。

【解法】

1. 状态表示：

$dp[i][j]$ 表示：将区间 $[i, j]$ 完全消除，所需的最短时间。

那么 $dp[1][n]$ 就是最终结果。

2. 状态转移方程：

枚举区间的分割点 k ，将整个区间合并的最小步数就是左区间消除掉的最小步数 + 右区间消除掉的最小步数。即 $dp[i][k] + dp[k + 1][j]$ 。我们取所有情况下的最小值即可。

但是，还有一种情况，如果 $a[i] = a[j]$ ，那么我们在消除区间 $[i + 1, j]$ 的最后一步时，顺带就可以把 $a[i]$ 和 $a[j]$ 带走，此时的最小步数有可能是 $dp[i + 1][j - 1]$ 。在写状态转移方程的时候，这种情况也要考虑进去。

3. 初始化：

根据状态转移方程，我们要初始化长度为 1 和 2 时的状态，易得：

- a. $dp[i][i] = 1$;
- b. 如果 $a[i] = a[i + 1]$ ， $dp[i][i + 1] = 1$;
- c. 如果 $a[i] \neq a[i + 1]$ ， $dp[i][i + 1] = 2$;

4. 填表顺序：

先枚举区间长度，再枚举左端点，右端点通过计算。

【参考代码】

```
1  #include <iostream>
2  #include <cstring>
3
4  using namespace std;
5
6  const int N = 510;
7
8  int n;
```



```

9  int a[N];
10 int f[N][N];
11
12 int main()
13 {
14     cin >> n;
15     for(int i = 1; i <= n; i++) cin >> a[i];
16
17     memset(f, 0x3f, sizeof f);
18     for(int i = 1; i <= n; i++) f[i][i] = 1;
19     for(int i = 1; i + 1 <= n; i++)
20     {
21         int j = i + 1;
22         if(a[i] == a[j]) f[i][j] = 1;
23         else f[i][j] = 2;
24     }
25
26     for(int len = 3; len <= n; len++)
27     {
28         for(int i = 1; i + len - 1 <= n; i++)
29         {
30             int j = i + len - 1;
31             for(int k = i; k < j; k++)
32             {
33                 // [i, k] [k + 1, j]
34                 f[i][j] = min(f[i][j], f[i][k] + f[k + 1][j]);
35             }
36
37             if(a[i] == a[j]) f[i][j] = min(f[i][j], f[i + 1][j - 1]);
38         }
39     }
40
41     cout << f[1][n] << endl;
42
43     return 0;
44 }

```