

2023.3.12

不让古人，是谓有志；不让今人，是谓无量。

## 2023 3月第二周

总体上，依然保持之前的规划方向来学习。

对策略的调整是：每天早上开始就找两道自己力所能及得1500---1600范围内得题。  
来保持自己得信心以及状态。

## 算法学习：

1. 单调队列优化.md

## 刷题

1. atcoder
  1. E RLE.md (dp)
2. luogu(都是水题。)
  1. P1103 书本整理.md
  2. P1564 膜拜.md
  3. P1681 最大正方形II.md
  4. HXY和序列.md

## contest

1. 上海理工大学天梯赛vp：
  1. Setsuna的K数列.md
  2. 叠硬币.md
  3. Wiki with Fake AKGPLT.md
2. div2 857
  1. D. Buying gifts.md
  2. C. The Very Beautiful Blanket.md (未补好)

---

atcoder

E.RLE

## solve

状态定义：

定义状态  $f_{i,j}$  表示S串长度为i，T串长度为j方案个数。

状态转移

由小更新大的贡献转移角度：

1.  $dp_{i,j}$  枚举k。然后贡献后方。

复杂度分析

枚举i，j总体的复杂度应该是： $O(N^3)$

考虑优化

关注从大到小直接统计转移：

当我们枚举计算某个状态时，发现贡献来源于几种不同二维数组中某一段之和。因此可以使用前缀和优化。具体的实现方案是：分析枚举在后面拼接不同的字母长度。将其对k的影响分类成三种情况，分别为1 10 100 1000 10000。假设拼接的长度为1...9。对于j维度关注关注j-2段的下标。假设拼接长度为10...99考虑j-3段的前缀。依此类推。

$$dp_{i,j} += (sum[i-1][j-2] - sum[i-10][j-2]) \times 25;$$

$$dp_{i,j} += (sum[i-10][j-3] - sum[i-1000][j-3]) \times 25 \quad (1)$$

推广到第 $10^k$ 次种长度  $dp_{i,j} += (sum[i-10^k] - sum[i-10^{k+1}]) \times 25$

初始化

考虑那些全放同一个字母的各种方案作为最小状态：

## code

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 const int N = 3E3 + 10;
6
7 ll dp[N][N];
8 int lg(int x) {
9     int res = 1;
10    while (x) {
11        x /= 10;
```

```

12         res++;
13     }
14     return res; //在基础上加1的刚刚好。
15 }
16
17 ll sum[N][N];
18 int main()
19 {
20     ios::sync_with_stdio(false);
21     cin.tie(0);
22     int n , p; cin >> n >> p;
23     int pre[6] = {1 , 10 , 100 , 1000 , 10000 , 100000};
24     for (int i = 1; i <= n; i++) {
25         dp[i][lg(i)] = 26;
26
27     }
28     for (int i = 1; i <= n; i++) {
29         //小心对无关的状态做贡献。计算错误。还是要详细研究转移过程。
30         for (int j = lg(i) + 1; j <= min(n , i * 2); j++) {
31             for (int k = 1; k <= 4; k++) {
32                 dp[i][j] += (((sum[max(0 , i - pre[k - 1]])[j
- k - 1] - sum[max(0 , i - pre[k]])[j - k - 1]) + p) % p) *
25;
33                 dp[i][j] %= p;
34
35             }
36         }
37         for (int j = 1; j <= n; j++) {
38             sum[i][j] = (sum[i - 1][j] + dp[i][j]) % p;
39         }
40     }
41     ll ans = 0;
42     for (int i = 1; i < n; i++) {
43         ans = (ans + dp[n][i]) % p;
44     }
45     cout << ans << '\n';
46 }

```

## 洛谷水题:

### P1103 书本整理

 P1103 书本整理 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)



用一种非常暴力的dp定义方法。把之前的解中可以关注的大多数属性都关注了。对于简单问题这总是可行：

## ● 状态定义如下：



$f_{i,j,k}$ 表示：当前考虑了前*i*个，书架上的上一本书宽度为*j*，已经使用丢掉了*k*本书的最小不整齐代价。

状态转移方程如下

$$\begin{aligned} & \text{if}(a[i] \neq j) \\ & f[i][j][k] = \min(f[i][j][k], f[i-1][j][k-1]) \\ & \text{if}(a[i] == j) \\ & f[i][a[i]][k] = \min(f[i-1][a[i]][k], f[i-1][j][k] + \text{abs}(j - w)) \end{aligned} \quad (2)$$

## code



```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5
6  const int N = 2E2 + 10;
7  const int inf = 1E9 + 10;
8  //分别表示当前的尾数，当前去掉的书本的数目。
9  struct node {
10     int h , w;
11 } a[N];
12 int f[N][N][N];
13 int main()
14 {
15     ios::sync_with_stdio(false);
16     cin.tie(0);
17     int n , kk;
18     cin >> n >> kk;
19
20     for (int i = 0; i <= n; i++)
21         for (int j = 0 ; j <= 200 ; j++)
22             for (int k = 0; k <= kk; k++) {
23                 f[i][j][k] = inf;
```

```

24         }
25         //cout << f[0][0][0] << '\n';
26         for (int i = 1; i <= n; i++) {
27             cin >> a[i].h >> a[i].w;
28         }
29         sort(a + 1, a + 1 + n, [&](node & i, node & j) {
30             return i.h < j.h;
31         });
32
33         for (int i = 1; i <= n; i++) {
34             f[i][a[i].w][i - 1] = 0;
35         }
36         for (int i = 1; i <= n; i++) {
37             //考虑状态怎么迁移。
38             //当前新的状态可能由什么转移?
39             for (int k = 1; k <= kk; k++)
40                 for (int j = 1; j <= 200; j++) {
41                     //f[i][j][k]->k = 0.是什么情况?
42                     f[i][j][k] = min(f[i][j][k], f[i - 1][j][k -
1]);
43                 }
44             for (int k = 0; k <= kk; k++)
45                 for (int j = 0; j <= 200; j++) {
46                     f[i][a[i].w][k] = min(f[i][a[i].w][k], f[i -
1][j][k] + abs(a[i].w - j));
47                 }
48         }
49         int ans = (1 << 29);
50         for (int i = 0; i <= 200; i++) {
51             ans = min(ans, f[n][i][kk]);
52         }
53         cout << ans << '\n';
54     }

```

## P1564 膜拜



将一段01字符串分成若干段。对于每一段满足：一段中字符相同，或两种字符数量的差别不超过m。求出最少可以分成多少段。

线性dp，大概是属于被开发烂了的数组模型。但是事实上自己还是做的很慢。

## 1. 状态定义

$dp_i$ 表示1....i最少可以分成多少段。

## 2. 状态转移

$dp_i = \min(dp_j + 1)$ 前提是j...i合法。

## code



```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 3000;
6  int a[N];
7  int f[N];
8  int sum[N];
9  int main()
10 {
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13     int n , m;
14     cin >> n >> m;
15     fill(f + 1 , f + 1 + n , 100000000);
16     for (int i = 1; i <= n; i++) {
17         cin >> a[i];
18         sum[i] = sum[i - 1] + (a[i] == 1);
19     }
20
21     for (int i = 1; i <= n; i++) {
22         for (int j = 1; j <= i; j++) {
23             if (sum[i] - sum[j - 1] == i - j + 1 || sum[i] -
sum[j - 1] == 0 || abs(i - j + 1 - 2 * (sum[i] - sum[j - 1]))
<= m ) {
24                 f[i] = min(f[i] , f[j - 1] + 1);
25             }
26         }
27     }
28     cout << f[n] << '\n';
29 }
```

## P1681 最大正方形II (double experiences)

☁ P1681 最大正方形II - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

这种模型开发有限，但是也不大懂：

类似的问题如下：

☁ P1387 最大正方形 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

## 最大正方形

### solve

#### 状态定义

定义  $f_{i,j}$  表示以  $i, j$  为右下角的满足所有格子都是1的最大正方形边长。

定义  $w_{i,j}$  表示当前点向左走的最大距离。 $h_{i,j}$  表示向上走的最大距离。

#### 状态转移方程

$$f_{i,j} = \min(f_{i-1,j-1}, w_{i,j}, h_{i,j}) + (s_{i,j}=1)$$

### code

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 const int N = 10000 + 10;
6 int a[N][N] , f[N][N] , h[N][N] , w[N][N];
7
8
9 int main()
10 {
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13     int n , m; cin >> n >> m;
14     int ans = 0;
15     for (int i = 1; i <= n; i++)
16         for (int j = 1; j <= m; j++) {
17             cin >> a[i][j];
```

```

18
19         f[i][j] = min({f[i - 1][j - 1] , w[i - 1][j] ,
    h[i][j - 1]}) + a[i][j];
20         if (a[i][j])w[i][j] = w[i - 1][j] + 1 , h[i][j] =
    h[i][j - 1] + 1;
21         ans = max(f[i][j] , ans);
22     }
23     cout << ans << '\n';
24 }

```

## 最大正方形二

### solve

1. 关注到关注的解是正方形。

#### 状态定义

定义左下角为  $f_{i,j,0/1}$  表示  $i, j$  作为右下角，该处为 0/1 的最大正方形的面积。

#### 状态转移

如果当前位是 0:

$$f_{i,j,0} = \min(f_{i-1,j-1,0}, f_{i-1,j,1}, f_{i,j-1,1}) + 1$$

如果当前位是 1

$$f_{i,j,1} = \min(f_{i-1,j-1,1}, f_{i-1,j,0}, f_{i,j-1,0}) + 1$$

和上述比较，在指标函数中，就已经有了一些与最大  $h$ , 最大  $w$  相关的长度。

### code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 10000 + 10;
6
7  int a[N][N] , f[N][N][2];
8
9  int main()

```




```

10 {
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13     int n , m; cin >> n >> m;
14     int ans = 0;
15     for (int i = 1; i <= n; i++)
16         for (int j = 1; j <= m; j++) {
17             cin >> a[i][j];
18             if (a[i][j]) {
19                 f[i][j][1] = min({f[i - 1][j - 1][1] , f[i -
20 1][j][0] , f[i][j - 1][0]}) + 1;
21                 ans = max(f[i][j][1] , ans);
22             } else {
23                 f[i][j][0] = min({f[i - 1][j - 1][0] , f[i -
24 1][j][1] , f[i][j - 1][1]}) + 1;
25                 ans = max(f[i][j][0] , ans);
26             }
27         }
28     cout << ans << '\n';
29 }

```

天梯赛训练练习：

Setsuna的K数列

 D-Setsuna的K数列\_2022年中国高校计算机大赛-团队程序设计天梯赛（GPLT）  
上海理工大学校内选拔赛 (nowcoder.com)

时间限制: C/C++ 1秒, 其他语言2秒

空间限制: C/C++ 262144K, 其他语言524288K

64bit IO Format: %lld

### 题目描述

Komorebi非常喜欢数列,但他实在太弱了无法想象出一个数列该如何构造,于是他就去FD(Frog Department)找Setsuna求助, Setsuna是一名光荣的FTCer(Frog, Time Controller),并没有很多空闲,就随口给Komorebi构造了一个神秘的 $K$ 数列,构建方法如下:

- 1、创建一个集合 $A_K$ , 集合内包含 $K$ 的所有整数次幂, 如当 $K = 3$ 时,  $A_3$ 包含 $1, 3, 9, 27 \dots$ , 但不会包含 $2, 4 \dots$ 。
- 2、取 $A_K$ 内任意个元素各一个相加, 并将他们放进一个新的集合 $B_K$ 。
- 3、将 $B_K$ 内的元素从小到大排序, 得到的有序数列即是一个 $K$ 数列 $S_K$ 。

例如当 $K = 3$ 时,  $S_3 = \{1, 3, 4(3 + 1), 9, 10(9 + 1), 12(9 + 3), 13(9 + 3 + 1), \dots\}$ 。

Komorebi拿到这个数列后很开心, 每天都要盯着这个 $K$ 数列看很久。久而久之这个数列有些数被Komorebi弄丢了, Komorebi想复原这个数列, 又不好意思再去找Setsuna, 他就只能来求助你了。

Komorebi想知道 $K$ 数列 $S_K$ 的第 $n$ 项是多少, 你可以帮帮他吗? 当然这个数有可能会非常大, 因此请输出答案对 $10^9 + 7$ 取模后的结果。

输入描述:

## solve

观察出一些现象,

1. 假定一个具体的值  $k^a$ , 对于  $k^{0 \dots a-1}$  无论如何组合, 都小于当前的数。
2. 类比二进制串:

然后找到一个严格的偏序关系。其实第  $n$  大, 各种数字的选择组合情况, 就是  $n$  所表示的二进制的1上的情况。

## code

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 const int N = 1E6 + 10;
6 const int mod = 1E9 + 7;
7
8 int main()
9 {
10     ios::sync_with_stdio(false);
11     cin.tie(0);
12
13     int n, k; cin >> n >> k;
14     ll now = 1;
15     ll ans = 0;
16     while (n) {
17         if (n % 2) {ans = (ans + now) % mod;}
```

```

18         n /= 2;
19         now = now * k % mod;
20     }
21     cout << ans << '\n';
22 }

```

## 叠硬币

☁ H-叠硬币\_2022年中国高校计算机大赛-团队程序设计天梯赛（GPLT）上海理工大学校内选拔赛 (nowcoder.com)

## solve

定义dp状态：

$f_{i,j}$ 表示考虑前*i*个情况下，当前硬币堆高度之和为*j*的最小的硬币堆数。

状态转移方程：

$f_{i,j} = \min(f_{i-1,j}, f_{i-1,j-w} + 1)$  注意判*j - w*的大小。

但是上面没有解决所有问题：

对于找出转移路径：

定义 $pre_{i,j}$ 表示 $f_{i,j}$ 的解结构的尾部选择。

1. 转移更新时，保持记录即可。

关于还原：

1. 如果已经得到尾部的物品，可以计算上一个子问题状态 $f_{i-1,h-a[pre[i][j]]}$

**但是还没有解决问题：—**

还要求求出字典序最小的方案：

经典问题：倒过来考虑即可。进一步证明，在背包问题中的其它问法的笔记中写过：


```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 3E3 + 10;
6  const int inf = 1 << 29;
7  int f[N][N] , pre[N][N];
8  int a[N];
9
10 int main()
11 {
12     ios::sync_with_stdio(false);
13     cin.tie(0);
14
15     int n , H; cin >> n >> H;
16     for (int i = 0; i <= n + 1; i++) {
17         for (int j = 0; j <= H; j++) {
18             f[i][j] = inf;
19         }
20     }
21     for (int i = 1; i <= n; i++) {
22         cin >> a[i];
23     }
24     sort(a + 1 , a + 1 + n);
25     f[n + 1][0] = 0;
26     for (int i = n; i >= 1; i--) {
27         int x = a[i];
28         for (int j = 0; j <= H; j++) {
29             f[i][j] = f[i + 1][j];
30             pre[i][j] = pre[i + 1][j];
31         }
32         //动态规划中的复原的基本功没有做好:
33         for (int j = H; j >= x; j-- ) {
34             if (f[i + 1][j - x] + 1 <= f[i][j]) {
35                 f[i][j] = f[i + 1][j - x] + 1;
36                 pre[i][j] = i;
37             }
38         }
39     }
40     if (f[1][H] == inf) {cout << -1 << '\n';}
41     else {
42         cout << f[1][H] << '\n';
43         queue<int> que;
```

```

44     int id = pre[1][H];
45     int h = H;
46     while (id) {
47         que.push(id);
48         h -= a[id];
49         id = pre[id + 1][h];
50     }
51     while (que.empty() == false) {
52         cout << a[que.front()] << ' ';
53         que.pop();
54     }
55     cout << '\n';
56 }
57 }
58
59 /* stuff you should look for
60 * int overflow, array bounds
61 * special cases (n=1?)
62 * do smth instead of nothing and stay organized
63 * WRITE STUFF DOWN
64 * DON'T GET STUCK ON ONE APPROACH
65 */
66

```

## Wiki with Fake AKGPLT

 F-Wiki with Fake AKGPLT\_ “2021年中国高校计算机大赛-团队程序设计天梯赛 (GPLT) 上海理工大学校内选拔赛” (nowcoder.com)

## solve

看代码即可：

## code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6

```

```
7  string t = "AKGPLT";
8
9  int main()
10 {
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13
14     int n;
15     cin >> n;
16     while (n --) {
17         string s;
18         cin >> s;
19         if (s > t) {
20             cout << 0 << '\n';
21             continue;
22         }
23         bool flag = true;
24         for (int i = 0 ; i < (int)s.length(); i++) {
25             if (i >= 2 && s[i] > 'K') {
26                 cout << i - 1 << '\n';
27                 flag = false;
28                 break;
29             }
30             if (s[i] != 'A') {
31                 cout << i << '\n';
32                 flag = false;
33                 break;
34             }
35         }
36         if (flag) { cout << -1 << '\n';}
37
38
39     }
40 }
41
42 /* stuff you should look for
43 * int overflow, array bounds
44 * special cases (n=1?)
45 * do smth instead of nothing and stay organized
46 * WRITE STUFF DOWN
47 * DON'T GET STUCK ON ONE APPROACH
48 */
```

## 1. 前缀的字符串（不相等）的字典序比其本身小

### 摘苹果

☁ H-摘苹果\_2023年中国高校计算机大赛-团队程序设计天梯赛（GPLT）上海理工大学校内选拔赛（同步赛）(nowcoder.com)

## solve



注意到这种操作最多可以进行20次。维护未到尽头的点进行的单点修改即可。

赛时变量写错：惨痛经验。如果确定自己思路是对的。但是一直没有调出来。有可能是低级错误。

### 1. 变量名写错。

解决方法可以是：

1. 重新写一遍。
2. 再次审查代码中变量名称的意义。

## code



```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E5 + 10;
6  set<int> rec;
7  int n , m;
8  ll a[N];
9  class BIT {
10     ll c[N];
11
12 public:
13     ll query(int x) {
14         ll res = 0;
15         for (; x ; x -= x & (-x))
16             res += c[x];
17         return res;
18     }
```

```

19     void modify(int x, ll d) {
20         for (; x <= n; x += x & (-x)) {
21             c[x] += d;
22         }
23     }
24 };
25
26 BIT d1, d2;
27
28 //树状数组求区间和公式:
29 //cout << (x + 1)*d1.query(x) - d2.query(x) - (x)*d1.query(x -
30 //1) + d2.query(x - 1) << '\n';
31 //区间修改仔细点，前加后减。小心记错结论。
32 //求和问题非常容易溢出。
33
34 int main()
35 {
36     ios::sync_with_stdio(false);
37     cin.tie(0);
38
39     cin >> n >> m;
40     for (int i = 1; i <= n; i++) {
41         cin >> a[i];
42         if (a[i] >= 10) {
43             rec.insert(i);
44         }
45         d1.modify(i, a[i]);
46         if (a[i] < 100) {
47             d2.modify(i, 1);
48         }
49     }
50     for (int i = 1; i <= m; i++) {
51         int op, l, r; cin >> op >> l >> r;
52         if (op == 1) {
53             auto ptr = rec.lower_bound(l);
54             while (ptr != rec.end() && *ptr <= r) {
55                 ll t = a[*ptr];
56                 int no = *ptr;
57                 a[no] -= a[no] / 3 + (a[no] % 3 != 0);
58                 d1.modify(no, a[no] - t);
59                 if (a[no] < 100 && t >= 100) {
60                     d2.modify(no, 1);
61                 }
62                 ptr++;
63                 if (a[no] < 10) {
64                     rec.erase(no);
65                 }
66             }
67         }
68     }
69 }

```



```

64         }
65     }
66     } else if (op == 2) {
67         cout << d2.query(r) - d2.query(l - 1) << '\n';
68     }
69     } else if (op == 3) {
70         cout << d1.query(r) - d1.query(l - 1) << '\n';
71     }
72 }
73 }
74
75 /* stuff you should look for
76 * int overflow, array bounds
77 * special cases (n=1?)
78 * do smth instead of nothing and stay organized
79 * WRITE STUFF DOWN
80 * DON'T GET STUCK ON ONE APPROACH
81 */

```

div2 857

## D. Buying gifts

### solve



考虑对解空间进行枚举优化：对于第一个朋友的礼物选择。枚举一个价值最大值。于是发现可以对解空间进行。只要对每一个最大值d的所有可能枚举完全即可。

1. 关注某个最大值的解空间：
  1. 用有序的眼光考察：发现同一种（属于第一个朋友）比当前枚举最大值大的，都要选 $b_j$ ，而比它小的任意选：考虑最优条件：必选最大值，以及通过可自由选择的一些最大值进行调整。
    1. 如果必选b中最大值大于max.那么最优方案已经确定
    2. 如果必选b中最大值小于。然后从自由选择的值中，找第一个比max大的，以及第一个小于等于max的。

1. contest中，满脑子想着二分。但是二分的途中就已经确定了解。
2. 有一些问题，二分check的过程中，可能就已经重复了所有的解。我相当于使用了一个最蠢的方法。每一次枚举所有的最优子集。然后看是否出过check值。但其实已经得到了所有的最优子集。在一次check的过程中就已经可以算出来。

平常的普通问题：最优子集比较庞大，或者不容易枚举。所有采用一种局部枚举的二分操作。

## code



```
1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  const int oo = 0xffffffff;
6  const int N = 1E6 + 10;
7
8  struct node {
9      int x;
10     int y;
11 } a[N];
12 int n;
13
14 // int pre[N];
15 int sux[N];
16 int solve() {
17     set<int> rec1;
18     //然后就是一直选择，一直check
19     int ans = 1E9;
20     for (int i = 1; i <= n; i++) {
21         int mx = 0;
22         if (i != n) {
23             mx = sux[i + 1];
24             ans = min(ans , abs(a[i].x - mx));
25             //考虑从前面找一个较大值。
26             if (mx < a[i].x ) {
27                 auto ptr = rec1.lower_bound(a[i].x);
28                 if (ptr != rec1.end()) ans = min(abs(*ptr -
a[i].x) , ans);
29                 if (ptr != rec1.begin()) -- ptr;
30                 ans = min(ans , abs(*ptr - a[i].x));
31             }
32         } else {
33             auto ptr = rec1.lower_bound(a[i].x);
```

```

34         if (ptr != rec1.end()) ans = min(abs(*ptr -
a[i].x) , ans);
35         if (ptr != rec1.begin()) -- ptr;
36         ans = min(ans , abs(*ptr - a[i].x));
37     }
38     rec1.insert(a[i].y);
39 }
40 return ans;
41 }
42
43 void work(int testNo)
44 {
45     cin >> n;
46     for (int i = 1; i <= n; i++) {
47         cin >> a[i].x >> a[i].y;
48     }
49     sort(a + 1 , a + 1 + n , [&](const node & i , const node &
j) {
50         return i.x < j.x;
51     });
52     sux[n + 1] = 0;
53     for (int i = n; i >= 1; i--) {
54         sux[i] = max(sux[i + 1] , a[i].y);
55     }
56     cout << solve() << '\n';
57 }
58
59
60 int main()
61 {
62     ios::sync_with_stdio(false);
63     cin.tie(0);
64
65     int t; cin >> t;
66     for (int i = 1; i <= t; i++) work(i);
67 }

```

## C. The Very Beautiful Blanket

 Problem - C - Codeforces

## 简介



构造一个举证。满足矩阵中的每一个4\*4的矩阵中满足如下条件

1.  $a_{11} \oplus a_{12} \oplus a_{21} \oplus a_{22} = a_{33} \oplus a_{34} \oplus a_{43} \oplus a_{44}$
2.  $a_{13} \oplus a_{14} \oplus a_{23} \oplus a_{24} = a_{31} \oplus a_{32} \oplus a_{41} \oplus a_{42}$

找出一个元素种数最多的矩阵。

## solve



特殊的，构造一个矩阵满足以下形式：任何一个四块的异或和都为0.

如果一个数字上的某个位置上有1怎么消除？并且要保证两两不同：

1. 对于最高的位置上：不同的行打1个1.于是同行的两个必然可以消掉。行与行的数字之间就不会有交叉。
2. 对于较低的位置上，从1开始填比较低的位置。这样任意两行之间，上下两个元素就可以把低位的1给中和掉。于是无论怎么圈，都可以找出一个异或和为0的正方形：

## code



```
1 void work(int testNo)
2 {
3     int n , m;
4     cin >> n >> m;
5     cout << n*m << '\n';
6     for (int i = 0; i < n; i++) {
7         for (int j = 0 ; j < m; j++)
8             cout << (i << 10) + j << " \n"[j == m - 1];
9     }
10 }
```