2018年蓝桥杯模拟赛第五场题解

by islands

A组第一题 和 B组第一题

题库 > 结果填空：矩阵求和

100% 1000ms 262144K 编辑代码

给你一个从 $n \times n$ 的矩阵，里面填充 1 到 $n \times n$ 。例如当 n 等于 3 的时候，填充的矩阵如下。

```
1 2 3
4 5 6
7 8 9
```

现在我们把矩阵中的每条边的中点连起来，这样形成了一个新的矩形，请你计算一下这个新的矩形的覆盖的数字的和。比如，$n = 3$ 的时候矩形覆盖的数字如下。

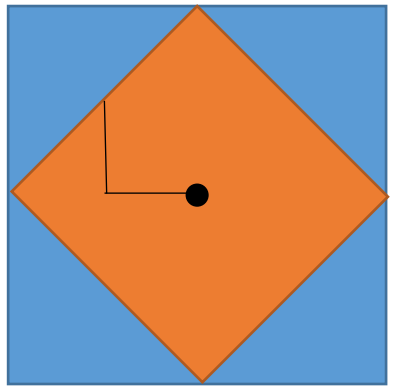
```
  2
4 5 6
  8
```

那么当 n 等于 101 的时候，矩阵和是多少?

- 我们会发现所有需要计算的点到中心点的距离（横坐标加纵坐标的距离和）都是小于等于 n / 2 的。

- 所以我们只需要把满足条件的点加上就可以了。

- 答案是 26020201



```cpp
#include <iostream>
using namespace std;
int main() {
    int sum = 0, n, cnt = 0;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            ++cnt;
            if (abs(n / 2 - i) + abs(n / 2 - j) <= n / 2) {
                sum += cnt;
            }
        }
    }
    cout << sum << endl;
    return 0;
}
```

B组第二题

题库 › 结果填空：素数个数

100%　1000ms　262144K　编辑代码

用 $0, 1, 2, 3 \cdots 7$ 这 8 个数组组成的所有整数中，质数有多少个（每个数字必须用到且只能用一次）。

提示：以 0 开始的数字是非法数字。

- 因为我们需要判断一个数字是否是素数判定次数过多，所以这里使用素数筛法打表，便于查询。

- 然后我们需要把这 8 个数字全排列，这里和之前的全排列不太一样，这里的全排列第一个数字不能是 0，另外我们还需要记录我们全排列的结果。所以我们使用一个数字来记录全排列的结果 num，这样也便于查询是否是素数 。添加一个数字的时候我们通过 10*num + i 完成把先添加的数字放到数字末尾 。对于第一位数字不能为 0，我们可以通过判定 num 是否为 0 来判定第一位数字是否放 0 。
- 这些细节处理完之后，剩下的就是一个简单的递归了。答案是 2668

```
const int M = 7;
bool f[N], vis[10];
int sum = 0;

void init() {
    for (int i = 2; i < N; ++i) {
        f[i] = true;
    }
    for (int i = 2; i * i < N; ++i) {
        if (f[i]) {
            for (int j = i * i; j < N; j += i) {
                f[j] = 0;
            }
        }
    }
}
```

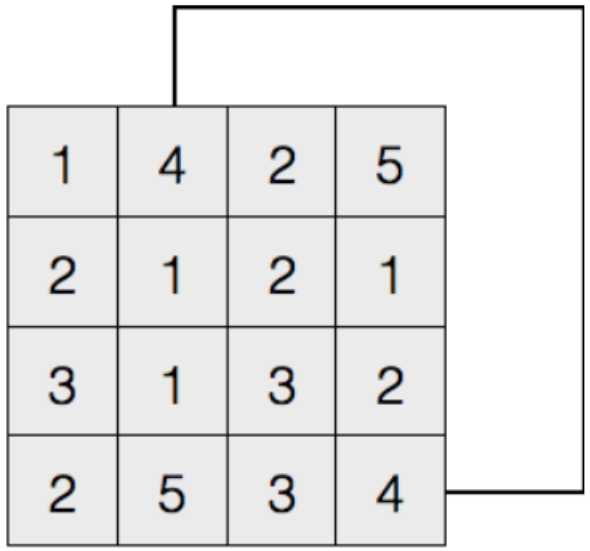
```
void dfs(int num, int cnt) {
    if (cnt == M + 1) {
        if (f[num]) {
            ++sum;
        }
        return ;
    }
    for (int i = num ? 0 : 1; i <= M; ++i) {
        if (!vis[i]) {
            vis[i] = true;
            dfs(num * 10 + i, cnt + 1);
            vis[i] = false;
        }
    }
}

int main() {
    init();
    dfs(0, 0);
    cout << sum  << endl;
}
```

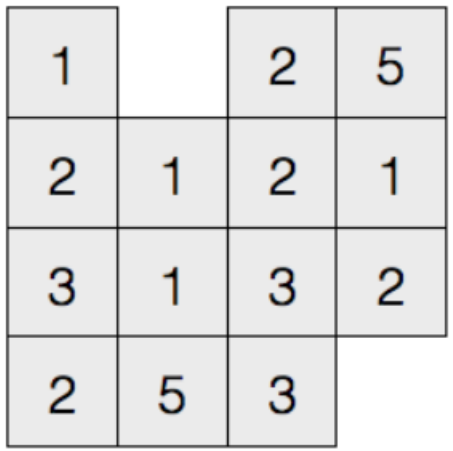
A组第二题 和 B组第三题

连连看是一款非常有意思的游戏。

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

我们可以把任意两个在图的在边界上的相同的方格一起消掉，比如把两个 4 消掉以后，





每次消掉两个方格的时候，都有会获得一个分数，第 i 次消的分数为 $i\times$ 方格的值。比如上面的消法，是第一次消，获得的分数为 $1 \times 4 = 4$。

请你帮忙最优操作情况下，获得的分数最多为多少。

- 用 dfs 求解，每一次搜索，找到两个相同的边界点，标记即可。检测边界点的方法也很简单，只需要一个点的四个方向中有一个点在地图外或者已经被标记，那么这个点就是边界点。
- 检测边界点的方法也很简单，只需要一个点的四个方向中有一个点在地图外或者已经被标记，那么这个点就是边界点。

- 答案是 89，不要手算，手算很难找到最优解。最优的消除方案为(2 2)、(1 1)、(1 1)、(3 3)、(4 4)、(5 5)。

```cpp
#include <iostream>
using namespace std;
char mat[21][21];
bool vis[21][21];
int n;
int dir[4][2] = {0, 1, 1, 0, 0, -1, -1, 0};
bool in(int x, int y) {
    return 0 <= x && x < n && 0 <= y && y < n;
}
bool check(int x, int y) {
    if (vis[x][y]) {
        return false;
    }
    for (int i = 0; i < 4; ++i) {
        int tx = x + dir[i][0];
        int ty = y + dir[i][1];
        if (!in(tx, ty)) {
            return true;
        }
        if (vis[tx][ty] == 1) {
            return true;
        }
    }
    return false;
}
int ans = 0;
void dfs(int step, int s) {
    ans = max(ans, s);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (check(i, j)) {
                for (int p = 0; p < n; ++p) {
                    for (int q = 0; q < n; ++q) {
                        if (p == i && q == j) {
                            continue;
                        }
```

```cpp
int ans = 0;
void dfs(int step, int s) {
    ans = max(ans, s);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (check(i, j)) {
                for (int p = 0; p < n; ++p) {
                    for (int q = 0; q < n; ++q) {
                        if (p == i && q == j) {
                            continue;
                        }
                        if (mat[p][q] == mat[i][j] && check(p, q)) {
                            vis[p][q] = vis[i][j] = 1;
                            dfs(step + 1, s + step *(mat[i]
[j] - '0'));
                            vis[p][q] = vis[i][j] = 0;
                        }
                    }
                }
            }
        }
    }
}
int main() {
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cin >> mat[i];
    }
    dfs(1, 0);
    cout << ans << endl;
    return 0;
}
```

A组第二题 和 B组第三题

题库 › 结果填空：加分二叉树

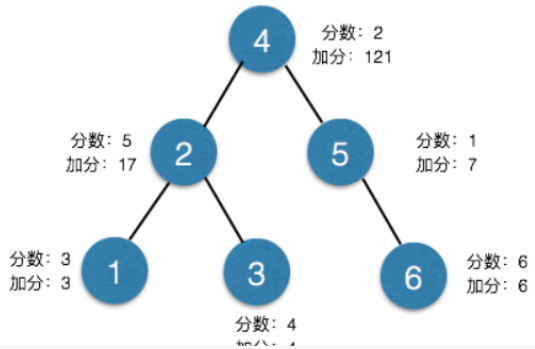
100%　　1000ms　　262144K　　编辑代码

一个有 20 个节点的二叉树，其中序遍历的结果为 $(1,2,3\cdots 20)$，其中 $1,2,3,\cdots 20$ 为节点编号。每个结点都有一个分数，节点 1 到 节点 20 的分数依次为

9 8 9 9 5 7 4 2 2 4 5 6 7 8 9 3 3 4 5 1

每个结点 i 都有一个加分，如果是叶子节点，这个节点的加分 = 结点分数。否则这个节点的加分 = 左儿子的加分 × 右儿子的加分 + 自己的分数。如果左儿子或者右儿子为空，我们用 1 代替。

比如下面这个树，圈里面的数字代表节点编号，中序遍历为 $(1,2,3,4,5,6)$。



加分：4

那么对于这 20 个点，中序遍历的结果为 $(1,2,3\cdots 20)$ 的所有二叉树中，根节点加分最多的二叉树的加分是多少。

- 可以用 dfs 直接搜索。dfs 搜索方法很好写，对于每棵子树，只需要枚举一下根节点即可，然后取加分最大的一棵。

- 另外的方法就是标程的方法，用 dp[i][j] 表示 i 到 j 组成一棵子树的的最大分数，然后进行区间 dp 即可。

- dfs 的方法，可以转换成动态规划，也就是记忆化搜索。

- 答案是 53071238

dfs 写法

```cpp
#include<iostream>
using namespace std;
int dp[30][30];
int a[30];
int dfs(int l, int r) {
    if (l > r) {
        return 1;
    }
    if (l == r) {
        return a[l];
    }
    int ans = 0;
    for (int i = l; i <= r; ++i) {
        int w = dfs(l, i - 1) * dfs(i + 1, r) + a[i];
        ans = max(ans, w);
    }
    return ans;
}
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    cout << dfs(1, n) << endl;
    return 0;
}
```

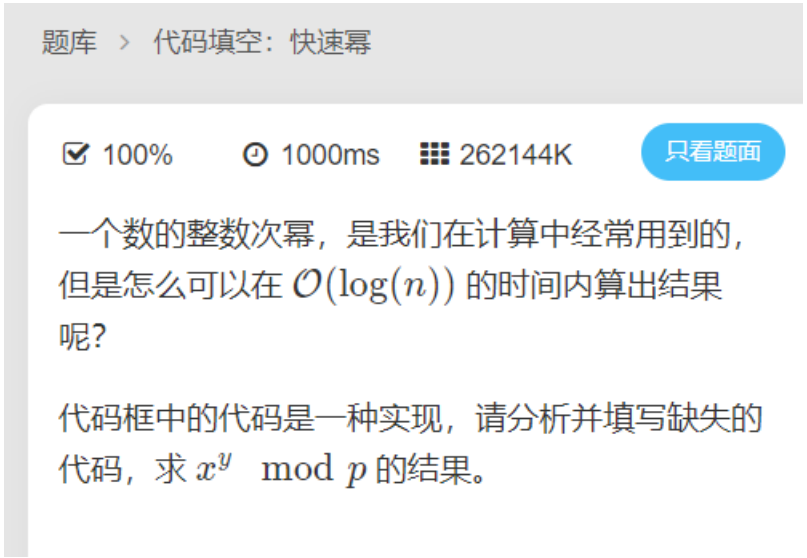
动态规划写法

```cpp
#include<iostream>
using namespace std;
int dp[30][30];
int a[30];
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    for (int i = 0; i <= n + 1; ++i) {
        for (int j = 0; j <= n + 1; ++j) {
            dp[i][j]=1;
        }
        dp[i][i] = a[i];
    }
    for (int i = 2; i <= n; ++i) {
        for (int j = 1; j <= n - i + 1; ++j) {
            for (int k = j; k <= j + i - 1; ++k) {
                int ww = dp[j][k - 1] * dp[k + 1][j + i - 1] + a[k];
                dp[j][j + i - 1] = max(dp[j][j + i - 1], ww);
            }
        }
    }
    cout << dp[1][n] << endl;
    return 0;
}
```

dfs 转记忆化搜索

```cpp
#include<iostream>
using namespace std;
int dp[30][30];
int a[30];
int dfs(int l, int r) {
    if (l > r) {
        return 1;
    }
    if (l == r) {
        return a[l];
    }
    if (dp[l][r]) {
        return dp[l][r];
    }
    int ans = 0;
    for (int i = l; i <= r; ++i) {
        int w = dfs(l, i - 1) * dfs(i + 1, r) + a[i];
        ans = max(ans, w);
    }
    return dp[l][r] = ans;
}
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    cout << dfs(1, n) << endl;
    return 0;
}
```

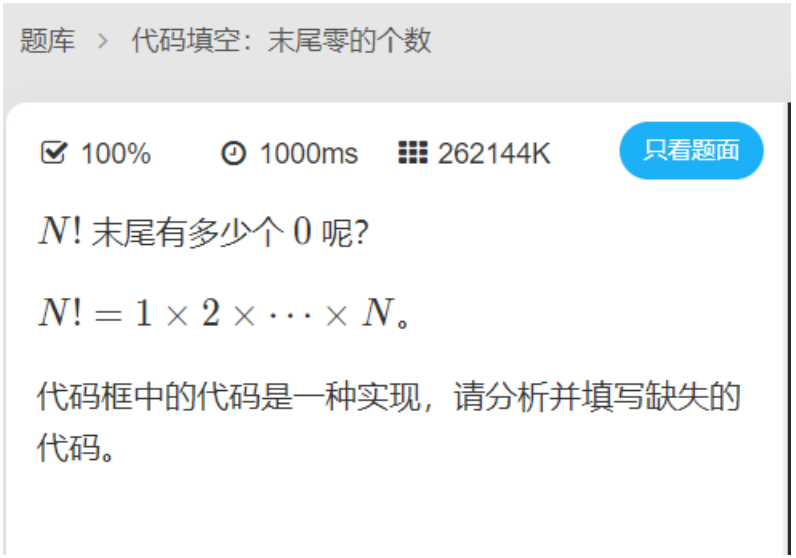
A组第四题 和 B组第四题



- pw(x, y / 2, p) * pw(x, y / 2, p) % p

对于 $x^y \%p$ 我们可以把它写成 $x^{y/2} \times x^{y/2} \%p$ 。如果 y 是奇数我们还需要再乘以 x。

A组第五题 和 B组第五题



- 填空 n = n / 5

- 对于一个数的阶乘(分解成多个素数相乘)，如果想末尾出现 0 的话，只有当 5 和 2 出现的时候，才会在末尾出现 0 。

- 因为 2 的个数一定比 5 多。所以我们就可以得出一个结论，一个数的阶乘，末尾 0 的个数就是看里面 5 的个数。

- 现在变成求 1 到 n 的因子有多少个 5。对于包含 1 个 5 的数字，就是 n/5，包含两个 5 的数字个数为 n / 25。。。通过 n = n / 5 的方式，每次剥掉一层 5。

B组第六题

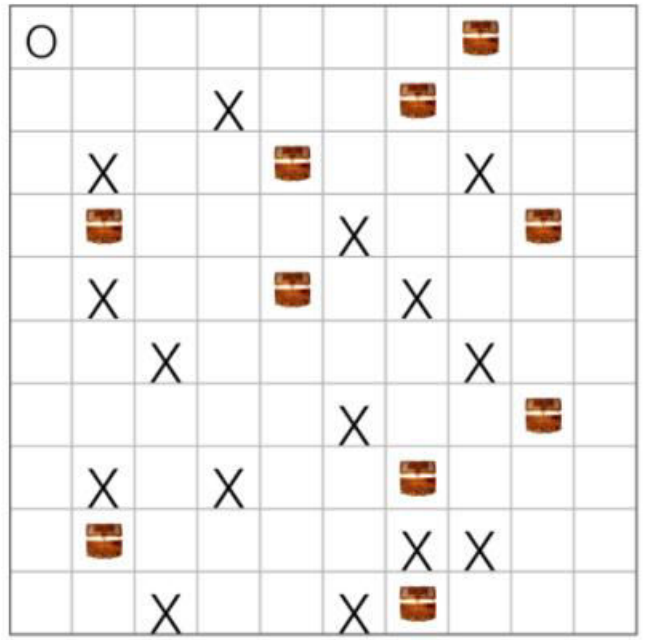
题库 > 结果填空：藏宝图

100%　　1000ms　　262144K　　编辑代码

蒜头君得到一张藏宝图。藏宝图是一个 10 × 10 的方格地图，图上一共有 10 个宝藏。有些方格地形太凶险，不能进入。

整个图只有一个地方可以出入，即是入口也是出口。蒜头君是一个贪心的人，他规划要获得所有宝藏以后才从出口离开。

藏宝图上从一个方格到相邻的上下左右的方格需要 1 天的时间，蒜头君从入口出发，找到所有宝藏以后，回到出口，最少需要多少天。



O 出入点　X 不能进入的点

宝藏

- 常规方法：状态压缩 bfs，每个点状态用 d[x][y][state] 表示，state 表示每个宝藏是否获取的状态压缩。然后 bfs 即可。

- 其他方法：全排列枚举选取宝藏的顺序，然后用 dfs 求出一个宝藏到另外一个宝藏之间的最短距离。优化的话，就是预处理任意两个宝藏之间的最短路距离。

```cpp
#include <iostream>
#include <queue>
using namespace std;
char mat[11][11] = {
                        "O......9..",
                        "...X..0...",
                        ".X..8..X..",
                        ".7...X..6.",
                        ".X..5.X...",
                        "..X....X..",
                        ".....X..3.",
                        ".X.X..1...",
                        ".4....XX..",
                        "..X..X2...",
                    };

int dir[4][2] = {0, 1, 1, 0, 0, -1, -1, 0};
int d[11][11][1 << 10];
struct node {
    int x, y, s;
    node(int xx, int yy, int ss): x(xx), y(yy), s(ss) {}
};
bool in(int x, int y) {
    return 0 <= x && x < 10 && 0 <= y && y < 10 && mat[x][y] != 'X';
}
void bfs() {
    queue<node> q;
    memset(d, -1, sizeof(d));
    q.push(node(0, 0, 0));
    d[0][0][0] = 0;
    while (!q.empty()) {
        int x = q.front().x;
        int y = q.front().y;
```

```cpp
}
void bfs() {
    queue<node> q;
    memset(d, -1, sizeof(d));
    q.push(node(0, 0, 0));
    d[0][0][0] = 0;
    while (!q.empty()) {
        int x = q.front().x;
        int y = q.front().y;
        int s = q.front().s;
        q.pop();
        for (int i = 0; i < 4; ++i) {
            int tx = x + dir[i][0];
            int ty = y + dir[i][1];
            int ts = s;
            if (in(tx, ty)) {
                if ('0' <= mat[tx][ty] && mat[tx][ty] <= '9') {
                    ts = s | (1 << (mat[tx][ty] - '0'));
                }
                if (d[tx][ty][ts] == -1) {
                    d[tx][ty][ts] = d[x][y][s] + 1;
                    q.push(node(tx, ty, ts));
                }
            }
        }

    }
}
int main() {
    bfs();
    cout << d[0][0][(1 << 10) - 1] << endl;
    return 0;
}
```

A组第六题

题库 › 结果填空：最强团队 问答

100% 1000ms 262144K 编辑代码

蒜头君的是学校创新班的一员，创新班的成员经常参加各类算法比赛。创新班有 30 人，每个人都用一个整数值来表示他的算法水平的强弱。这 30 个人的算法水平如下

258055 69760 163908 249856 53440

151684 77958 176134 8262 229446

245953 20676 45189 69826 131075

28672 155717 118851 221318 254150

135235 86083 41089 28743 32772

225475 118855 249862 184320 217154

现在学校要参加一个算法团体赛，要在创新班中选出若干的成员参赛。学校经过精挑细选，选出一个最强团队，这个团队的整体算法水平。总所周知，算法比赛中，不是不是强强得强，一个团队的算法水平定义为所有成员的算法水平的异或。比如如果选出来 3 个人参赛，他们算法水平分别为 $2, 3, 4$，那么这个团队的整体水平为 $2 \oplus 3 \oplus 4 = 5$。

请你帮忙计算最强团队的整体算法水平为多少。

- 方法 1：二进制枚举暴力，只要有耐心，肯定能得到结果。

```
int bao() {
    int ans = 0;
    for (int i = 0; i < (1 << n); ++i) {
        int cnt = 0;
        for (int j = 0; j < n; ++j) {
            if (i & (1 << j)) {
                cnt ^= a[j];
            }
        }
        ans = max(ans, cnt);
    }
    return ans;
}
```

- 方法 2：01 背包。

```
dp[0] = true;
for (int i = 1; i <= 30; ++i) {
    for (int j = 0; j <= 262144; ++j) {
        dp[j] |= dp[j ^ a[i]];
    }
}
int ans = 0;
for (int i = 262144; i >= 0; --i) {
    if (dp[i]) {
        ans = i;
        break;
    }
}
```

- 方法 3：线性基。

- 对于一个数集 V，它的线性基β是它的一个子集，满足β中所有数互相异或得到的集合等价于 V 中所有数互相异或得到的集合。也就是说，β 可以看成是 V 的压缩。

- 线性基的二进制最高位互不相同，每一个二进制位对应一个基，该基的二进制最高位就是该二进制位。线性基的求法可以看代码。通过线性基求解最大值很简单，从最高位的线性基开始，如果异或后更大，就异或上这个线性基。

```cpp
#include <iostream>
#include <assert.h>
using namespace std;
int a[40];
int p[20];
int n;
int main() {
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        int x = a[i];
        for (int j = 19; j >= 0; --j) {
            if (x & (1 << j)) {
                if (!p[j]) {
                    p[j] = x;
                    break;
                }
                x ^= p[j];
            }
        }
    }
    int ans2 = 0;
    for (int i = 19; i >= 0; --i) {
        ans2 = max(ans2, ans2 ^ p[i]);
    }
    cout << ans2 << endl;
    return 0;
}
```

B组第七题

题库 > 程序设计：合并数字

100% 1000ms 262144K 编辑代码

蒜头君得到了 n 个数，他想对这些数进行下面这样的操作，选出最左边的相邻的差的绝对值为 1 的两个数，只保留较小的数，删去较大的数，直到没有两个相邻的差的绝对值为 1 的数，问最多可以进行多少次这样的操作?

输入格式

输入第一行为一个整数 $n(1 \leq n \leq 10^5)$，表示数字的总数

第二行为 n 个整数 $x_1, x_2, ..., x_n(1 \leq x_i \leq 10^9)$，表示这些数。

输出格式

输出一行，为一个整数，表示蒜头君最多可以进行多少次这样的操作。

- 用栈来维护每次合并完的数，每入栈一个数以后栈顶和次栈顶比较，如果可以合并就合并为新的栈顶，并且再次与次栈顶比较直至无法合并，在合并过程中统计次数即可。

```cpp
#include <iostream>
#include <stack>
using namespace std;
const int maxn = 100010;
stack<int> s;
int main() {
    int n, cnt = 0, x;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> x;
        while (!s.empty() && s.top() - x == 1) {
            s.pop();
            ++cnt;
        }
        if (!s.empty() && x - s.top() == 1) {
            ++cnt;
        } else {
            s.push(x);
        }
    }
    cout << cnt << endl;
    return 0;
}
```

A组第七题 和 B组第八题

题库 › 程序设计：蒜头君下棋

100% 1000ms 262144K 编辑代码

蒜头君喜欢下棋。最近它迷上了国际象棋。国际象棋的棋盘可以被当做一个 8×8 的矩阵，棋子被放在格子里面(不是和中国象棋一样放在线上)。

蒜头君特别喜欢国际象棋里面的马，马的移动规则是这样的：横着走两步之后竖着走一步，或者横着走一步之后竖着走两步。例如，一匹马在 $(3,3)$ 的位置，则它可以到达的地方有 $(1,2)$，$(2,1)$，$(1,4)$，$(4,1)$，$(5,2)$，$(2,5)$，$(5,4)$，$(4,5)$ 八个地方。蒜头君想要把整个棋盘都放上马，并且让这些马不能相互攻击(即任何一匹马不能走一步之后就到达另一匹马的位置)。蒜头君当然知道在 8×8 的棋盘上怎么放马，但如果棋盘变为 $n \times m$ 的，蒜头君就不懂了。他希望你来帮忙他计算一下究竟能放多少匹马。

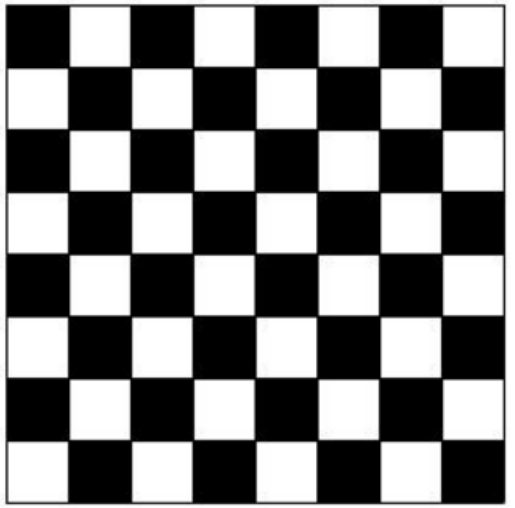
输入格式

共一行，两个整数n和$m(1 \leq n, m \leq 1000)$，代表棋盘一共有 n 行 m 列。

输出格式

输出一个整数，代表棋盘上最多能放的马的数量。

- 当棋盘只有一行时，棋盘上全放上棋子即可。
- 当棋盘只有两行时，参考如下放的方法：



- OOXXOOXXOO.....
- OOXXOOXXOO.....

- 即放两排空两排。

- 其它情况下，参考国际象棋棋盘的颜色，马跳一次一定会从白色格子变为黑色，因此把所有马放在同一颜色下即可。

```cpp
#include <iostream>
using namespace std;
int main() {
    int n, m;
    cin >> n >> m;
    if (n > m) {
        swap(n, m);
    }
    if (n == 1) {
        cout << m << endl;
    } else if (n == 2) {
        cout << m / 4 * 4 + min(m % 4, 2) * 2 << endl;
    } else {
        cout << (n * m + 1) / 2 << endl;
    }
    return 0;
}
```

A组第八题 和 B组第九题

题库 > 程序设计：蒜头君的数轴

100% 1000ms 262144K 编辑代码

今天蒜头君拿到了一个数轴，上边有 n 个点，但是蒜头君嫌这根数轴不够优美，想要通过加一些点让它变优美，所谓优美是指考虑相邻两个点的距离，最多只有一对点的距离与其它的不同。

蒜头君想知道，他最少需要加多少个点使这个数轴变优美。

输入格式

输入第一行为一个整数 $n(1 \le n \le 10^5)$，表示数轴上的点数。

第二行为 n 个不重复的整数 $x_1, x_2, ..., x_n(-10^9 \le x_i \le 10^9)$，表示这些点的坐标，点坐标乱序排列。

输出格式

输出一行，为一个整数，表示蒜头君最少需要加多少个点使这个数轴变优美。

- 考虑两对相邻点距离怎么由不同的通过加点变为全部相同，也就是对线段进行分割，分割成相同长度的若干段，这个长度最大也就是原来两条线段长度的最大公约数，我们先对点排好序然后做差，得到相邻点距离，因为最多允许有一段距离跟其它不同，可以枚举不同的是哪一段，对其它段求最大公约数。

- 怎么快速求出这个值，可以运用前缀和的思路，预处理出前缀 GCD 和后缀 GCD，那么其它段的 GCD 就可以通过这两个 GCD 再求一遍 GCD 得到。得到分割后的距离后，每一段需要加的点就可以算了，但是如果每一次都去看每一段要加多少点还是太慢了，我们可以预处理得到最左点和最右点的距离，减去枚举中的那一段，再除以最大公约数得到一共会被分成多少小段，需要加的点数就是这个数字减去原来有多少段，也就是减去 n - 2。

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int maxn = 100010;
int X[maxn];
int dist[maxn];
int gcd1[maxn], gcd2[maxn];
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> X[i];
    }
    if (n <= 3) {
        cout << 0 << endl;
        return 0;
    }
    sort(X, X + n);
    int sum = 0;
    for (int i = 0; i < n - 1; i++) {
        dist[i] = X[i + 1] - X[i];
        sum += dist[i];
    }
    gcd1[0] = dist[0];
    for (int i = 1; i < n - 1; i++) {
        gcd1[i] = gcd(gcd1[i - 1], dist[i]);
    }
    gcd2[n - 2] = dist[n - 2];
        sum += dist[i];
    }
    gcd1[0] = dist[0];
    for (int i = 1; i < n - 1; i++) {
        gcd1[i] = gcd(gcd1[i - 1], dist[i]);
    }
    gcd2[n - 2] = dist[n - 2];
    for (int i = n - 3; i >= 0; i--) {
        gcd2[i] = gcd(gcd2[i + 1], dist[i]);
    }
    int ans = 0x3f3f3f3f;
    for (int i = 0; i < n - 1; i++) {
        int d;
        if (i == 0) {
            d = gcd2[i + 1];
        } else if (i == n - 2) {
            d = gcd1[i - 1];
        } else {
            d = gcd(gcd1[i - 1], gcd2[i + 1]);
        }
        ans = min(ans, (sum - dist[i]) / d - n + 2);
    }
    cout << ans << endl;
    return 0;
}
```

A组第九题 和 B组第十题

题库 › 程序设计：划分整数

100%　1000ms　262144K　编辑代码

蒜头君特别喜欢数学。今天，蒜头君突发奇想：如果想要把一个正整数 n 分解成不多于 k 个正整数相加的形式，那么一共有多少种分解的方式呢?

蒜头君觉得这个问题实在是太难了，于是他想让你帮帮忙。

输入格式

共一行，包含两个整数 $n(1 \leq n \leq 300)$ 和 $k(1 \leq k \leq 300)$，含义如题意所示。

输出格式

一个数字，代表所求的方案数。

- 采用动态规划的方式。令 dp(n, k) 表示将 n 分解为不超过 k 个数之和的方案数。
- 分成四种情况讨论，
- 1. n = 1 或者 k = 1 的时候，只有一种方案。
- 2. n < k 的时候，实际上方案树就是 dp(n ,n)，因为 n 不可能分解超过 n 个数。
- 3. n > k 的时候，这时候，我们情况，如果一定拆成 k 个数，那么实际上每个数必须大于等于 1，我们把每个数都减去 1，那么对应的方案数是 dp(n - k, k)，如果拆成小于 k 个数，那么就是 dp(n, k - 1)，这时候方案数是 dp(n, k - 1) + dp(n - k, k)。
- 4. n = k 的时候，沿用 3 结论，这时候，如果一定要拆成 k 个数，实际上只有一种方案，那么总方案为 dp(n, k-1) + 1。

那么对应的转移如下。

$$dp(n,k)=\begin{cases}1 & n=1 \text{ or } k=1\\ dp(n,n) & k>n\\ dp(n,k-1)+1 & k=n\\ dp(n,k-1)+dp(n-k,k) & k<n\end{cases}$$

时间复杂度：$\mathcal{O}(n*k)$

需要注意的是，因为数据过大，本题需要使用 long long 类型。

```cpp
#include <iostream>
using namespace std;
const int maxn = 310;
long long dp[maxn][maxn];
int main() {
    int n, k;
    cin >> n >> k;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= k; ++j) {
            if (i == 1 || j == 1) {
                dp[i][j] = 1;
            } else if (j > i) {
                dp[i][j] = dp[i][i];
            } else if (i == j) {
                dp[i][j] = dp[i][j - 1] + 1;
            } else {
                dp[i][j] = dp[i][j - 1] + dp[i - j][j];
            }
        }
    }
    cout << dp[n][k] << endl;
    return 0;
}
```

A组第十题

题库 > 程序设计：取石子

100% 1000ms 262144K 编辑代码

蒜头君和花椰妹今天都很无聊。两个人相约一起玩游戏。

蒜头君取出了一堆奇形怪状的石子，并且把它分成了三堆。他和花椰妹轮流从里面取石子，取出最后一颗石子的人胜利。花椰妹觉得这样没意思，于是她要求加入一个限制条件：每个人每次只能取出 $1, 3, 7$ 或 9 颗石子。石子数目不够的时候不能多取，如还剩 2 颗石子的情况下，只能拿走 1 颗，而不能选择 $3, 7, 9$。蒜头君答应了，但是他要让花椰妹先拿。

已知蒜头君和花椰妹都很聪明，总是会做出最好的拿石子办法。现在你能知道，花椰妹究竟能不能赢吗？

输入格式

共一行，包括三个整数 n, m 和 k $(1 \leq n, m, k \leq 1000)$，代表三堆石子的数目。

输出格式

如果花椰妹能赢，则输出 `"win"`；否则则输出 `"lose"`。

- 用 sg 函数求出每一堆石子的 sg 函数值，异或若不为 0 则后手胜，否则先手胜。

- 具体的请自行学习博弈论相关的知识。

- 当然，对于小数据，可以利用必败和必胜来进行动态规划，用 dp[i][j][k] 表示三堆石为 i, j, k 的时候是必胜还是必败。如果一个点转移所有点都是必胜，那么这个点是必败点。如果一个点能转移到一个必败点，那么这个点是必胜点。

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const int maxn = 1010;
int f[10] = {1, 3, 7, 9}, sg[maxn], s[maxn];
void getsg(int n) {
    memset(sg, 0, sizeof(sg));
    for (int i = 1; i <= n; i++){
        memset(s, 0, sizeof(s));
        for (int j = 0; f[j] <= i && j < 4; j++) s[sg[i - f[j]]] = 1;
        for (int j = 0; ; j++) {
            if (!s[j]) {
                sg[i] = j; break;
            }
        }
    }
}
int main() {
    int n, m, k;
    cin >> n >> m >> k;
    getsg(1000);
    if (sg[n] ^ sg[m] ^ sg[k]) printf("win");
    else printf("lose");
    return 0;
}
```