模拟算法

定义

> 模拟算法就是“代码版的阅读理解”

它不需要推导高深的数学公式，也不需要通过复杂的逻辑（如递归、动态规划）去优化问题。它的核心逻辑只有一条：题目告诉你发生了什么，你就让

计算机如实地去执行一遍发生的过程。

- 题目说： “小明向左走三步，再向右走两步。”
- 模拟算法做： 编写代码更新小明的坐标 x = x - 3; 然后 x = x + 2;。
- 结果： 输出小明最后的位置。

它就像是在拍电影： 题目是剧本，你是导演，代码是演员。你不需要改编剧本，只需要确保演员严格按照剧本的每一个动作去演，不要漏掉任何一个细节。

特点和技巧

特点：

- 代码量大
- 操作多
- 思路复杂
- 较为复杂的模拟题出错后难定位错误

技巧：

- 在动手写代码之前，在草纸上尽可能地写好要实现的流程.
- 在代码中，尽量把每个部分模块化，写成函数、结构体或类
- 对于一些可能重复用到的概念，可以统一转化，方便处理:如，某题给你"YY-MM-DD 时：分" 把它抽取到一个函数，处理成秒，会减少概念混淆
- 调试时分块调试。模块化的好处就是可以方便的单独调某一部分
- 写代码的时候一定要思路清晰，不要想到什么写什么，要按照落在纸上的步骤写

网格/地图类模拟

在二维数组（棋盘）上，根据规则改变坐标 (x, y)。

建立“坐标系直觉”

- 原点： 通常在左上角 (0, 0)。
- X轴（行）： 垂直向下增加（代表第几行）。
- Y轴（列）： 水平向右增加（代表第几列）。

> `a[x][y]` 中，x 管上下，y 管左右。

方向数组

```
// 顺序：上    右    下    左
int dx[] = {-1,  0,  1,  0}; // 行的变化量
int dy[] = { 0,  1,  0, -1}; // 列的变化量
```

假设当前方向是 d（0代表上，1代表右...）：

- 向前走一步：`nx = x + dx[d]; ny = y + dy[d];`
- 向右转（顺时针）：`d = (d + 1) % 4;`
- 向左转（逆时针）：`d = (d + 3) % 4;`

代码实现

```
#include <iostream>
#include <vector>
using namespace std;

const int N = 10;  // 定义网格大小为10×10
int grid[N][N];    // 存储网格中每个位置的数字
bool visited[N][N];  // 标记网格中的位置是否已被访问

// 方向数组：上、右、下、左（顺时针方向）
int dx[] = {-1, 0, 1, 0};
int dy[] = {0, 1, 0, -1};

int main() {
    // 起始位置为左上角(0,0)
    int x = 0, y = 0;
    // 初始方向：1（向右），对应dx[1], dy[1]
    int d = 1;

    // 从1开始填充，已填充数字的计数
    int count = 1;
    // 将起始位置填充为1
    grid[x][y] = count;
    // 标记起始位置已访问
    visited[x][y] = true;

    // 继续填充直到填满整个网格（N*N个数字）
    while (count < N * N) {
        // 根据当前方向计算下一个位置的坐标
        int nx = x + dx[d];
        int ny = y + dy[d];

        // 检查下一个位置是否在网格范围内且未被访问
        if (nx >= 0 && nx < N && ny >= 0 && ny < N && !visited[nx][ny]) {
            // 移动到下一个位置
            x = nx;
            y = ny;
            // 增加计数并填充当前网格
            count++;
            grid[x][y] = count;
            // 标记当前网格已访问
            visited[x][y] = true;
        } else {
            // 如果下一个位置不可达，则顺时针旋转方向（改变d）
            d = (d + 1) % 4;
```

```
            // 注意：这里不移动位置，下一轮循环将用新方向重试
        }
    }

    // 打印填充好的网格，每个数字后跟一个制表符以便对齐
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            cout << grid[i][j] << "\t";
        }
        cout << endl;
    }

    return 0;
}
```

灌溉

题目描述

- 输入：n 行 m 列，有 t 个出水口，需要模拟 k 分钟。
- 规则：每一分钟，有水的格子会向 上、下、左、右 四个方向扩展一格。
- 输出：k 分钟后，共有多少个格子有水。

```
#include <iostream>
#include <cstring>
using namespace std;

const int MAXN = 105;  // 定义最大网格大小，行和列最多为105

int g[MAXN][MAXN];     // 主网格数组，存储每个单元格的状态，1表示有细胞，0表示无细胞
int temp[MAXN][MAXN];  // 临时网格数组，用于在每轮更新中保存新的状态，避免状态更新冲突

// 方向数组：上下左右四个方向的偏移量
// dx[4] = {-1, 1, 0, 0} 对应上、下、不动、不动
// dy[4] = {0, 0, -1, 1} 对应不动、不动、左、右
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};

int main() {
    int n, m;  // 网格的行数n和列数m
    cin >> n >> m;  // 读取网格大小

    int t;  // 初始时有细胞的位置数量
    cin >> t;  // 读取初始细胞数量

    // 初始化网格，将所有位置的状态设为0（无细胞）
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m; j++) {
            g[i][j] = 0;
        }
    }

    // 读取初始有细胞的位置，并将其状态设为1
    for(int i = 0; i < t; i++) {
        int r, c;  // 行坐标和列坐标
        cin >> r >> c;  // 读取位置
        g[r][c] = 1;  // 将该位置标记为有细胞
    }

    int k;  // 细胞扩散的轮数
    cin >> k;  // 读取扩散轮数
```

```
    // 进行k轮细胞扩散
    while (k--) {
        // 将当前网格状态复制到临时网格中，以便同时更新所有细胞
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= m; j++) {
                temp[i][j] = g[i][j];
            }
        }

        // 遍历整个网格，查找有细胞的位置
        for(int x = 1; x <= n; x++) {
            for(int y = 1; y <= m; y++) {
                if (g[x][y] == 1) {  // 如果当前位置有细胞
                    // 遍历四个方向：上、下、左、右
                    for(int d = 0; d < 4; d++) {
                        int nx = x + dx[d];  // 计算相邻位置的行坐标
                        int ny = y + dy[d];  // 计算相邻位置的列坐标

                        // 检查相邻位置是否在网格范围内
                        if(nx >= 1 && nx <= n && ny >= 1 && ny <= m) {
                            temp[nx][ny] = 1;  // 在临时网格中标记相邻位置有细胞
                        }
                    }
                }
            }
        }

        // 将临时网格中的状态复制回主网格，完成本轮更新
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= m; j++) {
                g[i][j] = temp[i][j];
            }
        }
    }

    int ans = 0;  // 统计最终有细胞的位置数量
    // 遍历整个网格，统计状态为1的单元格数量
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m; j++) {
            if (g[i][j] == 1) {
                ans++;  // 有细胞则计数加1
            }
        }
    }

    // 输出最终有细胞的位置总数
    cout << ans << endl;

    return 0;
}
```

日期与时间类模拟

闰年判断

规则：四年一闰，百年不闰，四百年再闰。

```cpp
bool is_leap(int y) {
    return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);
}
```

月份天数表

```cpp
// 平年的每个月天数，2月先填28，函数里动态修正
int days[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

int get_days(int y, int m) {
    if (m == 2 && is_leap(y)) return 29; // 只有这里特殊
    return days[m];
}
```

“下一天”函数

```cpp
void next_day(int &y, int &m, int &d) {
    d++; // 先加一天
    if (d > get_days(y, m)) { // 如果日子爆了（比如1月32日）
        d = 1; // 变成下个月1号
        m++;   // 月份加1
        if (m > 12) { // 如果月份爆了（比如13月）
            m = 1; // 变成1月
            y++;   // 年份加1
        }
    }
}
```

回文日期

题目背景： 给你一个日期（如 20200202），让你找到在这个日期之后（不含当天）的：

- 第一个回文日期（如 20211202，正着读倒着读一样）。
- 第一个 ABABBABA 型的回文日期。

解题思路：

不要试图去构造数字，对于 C 组，最不易出错的方法就是：从给定日期的下一天开始，一天一天往后翻，每翻一天就检查一下是不是回文。

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

using namespace std;

// 月份天数表，索引0不用，从1开始对应1月到12月
int days[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

// 判断是否为闰年
bool is_leap(int y) {
    // 闰年条件：能被4整除但不能被100整除，或者能被400整除
    return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);
```

```
}

// 获取指定年份和月份的天数
int get_days(int y, int m) {
    // 如果是2月且是闰年，返回29天
    if (m == 2 && is_leap(y)) return 29;
    // 否则返回月份天数表中对应的天数
    return days[m];
}

// 计算下一天的日期（按天数累加）
void next_day(int &y, int &m, int &d) {
    d++;  // 天数加1
    // 如果天数超过当前月的最大天数，进入下个月
    if (d > get_days(y, m)) {
        d = 1;  // 天数重置为1
        m++;    // 月份加1
        // 如果月份超过12，进入下一年
        if (m > 12) {
            m = 1;  // 月份重置为1
            y++;    // 年份加1
        }
    }
}

// 检查字符串是否为回文
bool check_palindrome(string s) {
    string rev = s;          // 复制原字符串
    reverse(rev.begin(), rev.end());  // 反转字符串
    return s == rev;         // 比较原字符串和反转后的字符串
}

// 检查字符串是否为ABABBABA型回文
bool check_ababbaba(string s) {
    // 首先必须是回文
    if (!check_palindrome(s)) return false;

    char A = s[0];  // 第一个字符为A
    char B = s[1];  // 第二个字符为B

    // A和B不能相同
    if (A == B) return false;

    // 检查字符串格式是否为ABABBABA
    // 对于8位日期字符串：位置0-7分别对应A B A B B A B A
    return s[2] == A && s[3] == B && s[4] == B &&
           s[5] == A && s[6] == B && s[7] == A;
}

int main() {
    int N;  // 输入的日期，格式为YYYYMMDD
    cin >> N;  // 读取输入日期

    // 从输入日期中分离出年、月、日
    int y = N / 10000;              // 前4位是年份
    int m = (N % 10000) / 100;      // 中间2位是月份
    int d = N % 100;                // 最后2位是天数

    // 标记是否已找到两种类型的回文日期
    bool found1 = false;  // 标记是否找到普通回文日期
    bool found2 = false;  // 标记是否找到ABABBABA型回文日期

    // 循环查找，直到两种回文日期都找到
    while (!found1 || !found2) {
        // 计算下一天的日期
```

```
        next_day(y, m, d);

        // 将日期转换为8位字符串格式
        char buf[10];  // 缓冲区，存储格式化后的字符串
        // 格式化日期为YYYYMMDD格式
        sprintf(buf, "%04d%02d%02d", y, m, d);
        string s = buf;  // 将缓冲区内容转换为字符串

        // 检查并输出普通回文日期
        if (!found1 && check_palindrome(s)) {
            cout << s << endl;  // 输出找到的回文日期
            found1 = true;      // 标记已找到普通回文日期
        }

        // 检查并输出ABABBABA型回文日期
        if (!found2 && check_ababbaba(s)) {
            cout << s << endl;  // 输出找到的ABABBABA型回文日期
            found2 = true;      // 标记已找到ABABBABA型回文日期
        }
    }

    return 0;  // 程序正常结束
}
```