枚举与模拟

TEACHING COURSEWARE POWPOINT

授课时间：2025.07.25

目录

PART-01 模拟概念 TEACHING COURSEWARE
PART-02 模拟例题 TEACHING COURSEWARE
PART-03 枚举概念 TEACHING COURSEWARE
PART-04 枚举例题 TEACHING COURSEWARE

01

模拟概念

TEACHING
COURSEWARE

TEACH

模拟

定义：
“模拟算法（Simulation Algorithm）是指：按照问题描述的规则、流程或物理规律，用代码逐步还原过程，最终得到结果的算法。
简单说就是‘照葫芦画瓢’—— 问题让你做什么，你就用代码‘一步步做’，不需要太多技巧，重点是‘忠实还原’。”
核心思想：
“模拟算法的核心只有两个字：‘还原’。具体拆成两步：

第一步：拆解问题步骤。把问题的过程拆成若干个连续的小步骤（比如‘先做 A，再做 B，满足条件则做 C）；
第二步：用代码复现步骤。用循环、条件判断等基础语法，把每个小步骤翻译成代码，让计算机按顺序执行。”

适用条件：
当问题的 过程逻辑（如多条件判断、动态变化、交互规则） 复杂，难以用数学公式直接描述时，模拟算法是直观的解决方案。

模拟的基本释意

[mó nǐ]

也作摹拟。模仿现成的样子。

模拟

优点：
直观易懂：逻辑和现实过程一致，容易理解和实现代码写起来和操作步骤几乎一样；
普适性强：只要能描述清楚过程，就能用模拟解决，几乎不挑问题类型。
缺点：
效率可能较低：如果过程步骤极多（比如模拟 10 亿次掷骰子），会因重复操作过多导致耗时；
依赖规则的准确性：如果对问题的规则理解错了，模拟结果会完全错误。需要对细节把握到位。

02

模拟例题

TEACHING
COURSEWARE

TEACH

扫雷游戏

例（NOIP 2015 普及组）

给出一个 $n \times m$ 的网格，有些格子埋有地雷。求问这个棋盘上每个没有地雷的格子周围（上下左右和斜对角）的地雷数。

输入：第一行两个整数 n 和 m 表示网格的行数和列数，接下来 n 行 m 列的字符矩阵，描述地雷分布情况。字符*表示相应格子是地雷格，字符?表示相应格子是非地雷格。m 和 n 不超过 100。

输出：包含 n 行，每行 m 个字符，描述整个雷区。用*表示地雷格，用周围的地雷个数表示非地雷格。

```
3 3
*??
???
?*?
```

```
*10
221
1*1
```

地图是2维的，因此我们需要借助二维数组。我们也注意到需要对每一个非雷格子进行计算。

Question：请问对于每一个非雷格子，我们可以怎么找到他的八个相邻格？

扫雷游戏

(x-1,y-1)	(x-1,y)	(x-1,y+1)
(x,y-1)	(x,y)	(x,y+1)
(x+1,y-1)	(x+1,y)	(x+1,y+1)

方法1：手写八个方向，正确但麻烦

方法2：如果我们将相对(x, y)的偏移量（+1, 0, -1）的所有组合不遗漏、不重复地使用数组保存，则只需要在这个数组上循环，即可遍历所有的方向

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

Question：如何循环，下标从几开始

```
char g[105][105];//地图数组
int a[105][105];//答案数组
```

```
//假设当前格不是雷并且当前坐标是x,y
for(int i = [ ]; i < [ ]; i++)
{
    int nx = x + [ ];
    int ny = y + [ ];
    if(g[nx][ny] == [ ])//此处不考虑越界
    {
        [     ];
    }
}
//此处为单点求解过程
```

扫雷游戏


```cpp
#include <iostream>
using namespace std;
const int MAXN = 105;
const int MAXM = 105;
char g[MAXN][MAXM];  // 原始地图：'*'是雷，'.'是空地
int ans[MAXN][MAXM]; // 存储每个空地周围的雷数
// 八方向偏移量
const int dx[] = {-1, -1, -1, 0, 0, 1, 1, 1};
const int dy[] = {-1, 0, 1, -1, 1, -1, 0, 1};
int main() {
    int n, m;
    cin >> n >> m;
    // 读取地图
    for (int x = 0; x < n; ++x) {
        for(int y=0;y<m;++y) {
            cin >> g[x][y];
        }
    }
```

扫雷游戏

重点！

```
// 遍历每个格子，空地直接统计周围雷数（无cnt变量）
for (int x = 0; x < n; ++x) {
    for (int y = 0; y < m; ++y) {
        if (g[x][y] == '*') {
            continue; // 雷不处理，输出时直接输出'*'
        }

        // 空地：遍历8个方向，直接累加雷数到ans[x][y]
        for (int i = 0; i < 8; ++i) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            // 检查邻居是否在范围内且是雷
            if (nx >= 0 && nx < n && ny >= 0 && ny < m && g[nx][ny] == '*') {
                ans[x][y]++; // 直接累加，无需cnt
            }
        }
    }
}
```

扫雷游戏


```
// 输出结果
for (int x = 0; x < n; ++x) {
    for (int y = 0; y < m; ++y) {
        if (g[x][y] == '*') {
            cout << '*';
        } else {
            cout << ans[x][y];
        }
    }
    cout << endl;
}
```

模拟总结

DAILY REPORT INTRODUCTION

“模拟算法就像‘让计算机复现现实中的操作步骤’——把问题的过程拆成一步一步，让计算机按顺序执行，最后得到我们想要的结果。它可能不是最高效的，但一定是最‘诚实’的算法，因为它严格遵循现实逻辑。”

Q&A

Q：如果问题步骤很多，还适合用模拟吗？

Q：拆解问题步骤时，如何避免遗漏关键环节？

Q：扫雷游戏中，为什么要用偏移量数组处理 8 个方向？直接手写 8 个条件判断不行吗？

03

枚举概念

TEACHING
COURSEWARE

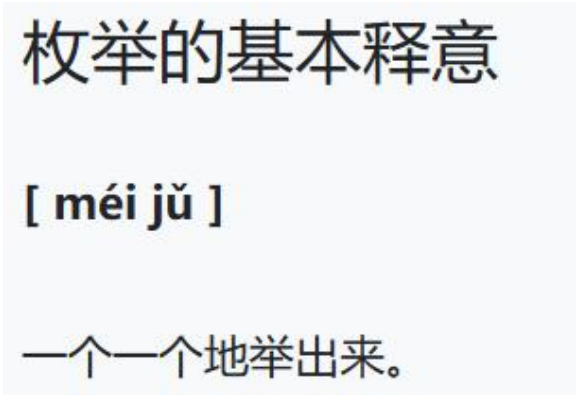
TEACH

枚举概念

“在计算机科学中，枚举算法（也叫‘穷举法’）是指：明确问题的所有可能解的范围（即‘解空间’），逐一列举所有候选解，对每个候选解验证是否满足问题的条件，最终得到所有有效解的算法。
简单说，就是让计算机‘按顺序把所有可能的情况过一遍，符合要求的就记下来。”
核心思想：
“枚举算法的核心可以浓缩成两个动作：

第一个是‘遍历’：系统地、不重不漏地列出解空间里的所有候选解；
第二个是‘验证’：对每个候选解，用问题的条件检查它是否有效。

这两个动作缺一不可：没有遍历，可能漏掉有效解；没有验证，会把无效的情况当成解。”



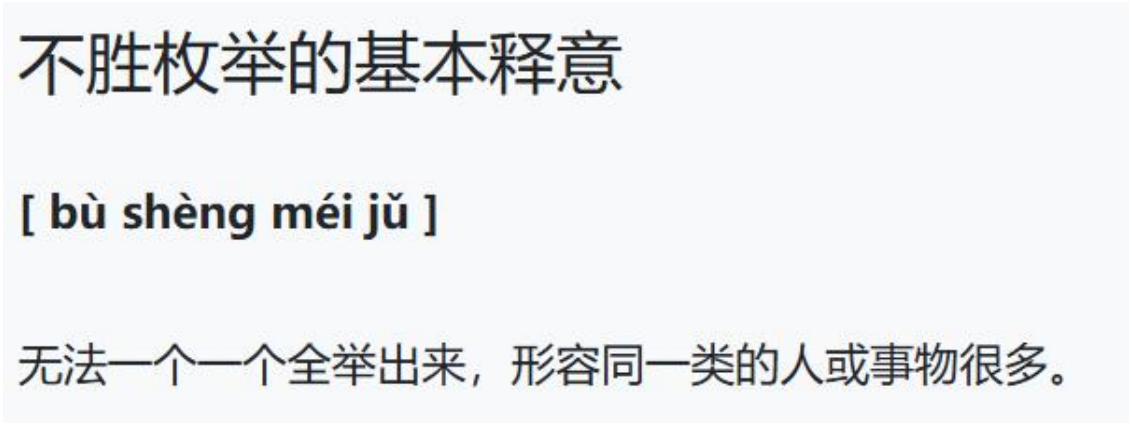
适用场景：当问题的解空间规模较小且可明确界定（如日期范围、数值范围较小），且验证条件简单时，枚举算法是直观高效的解决方案。

枚举概念

枚举算法的基本步骤
把枚举的过程拆解成可操作的步骤，让逻辑更清晰：
“用枚举算法解决问题，通常分三步：
第一步：界定解空间范围。
明确候选解可能存在的边界，范围越明确、越小，枚举的效率越高。
第二步：遍历解空间。
用循环（比如 for 循环、while 循环）逐个访问解空间里的每个候选解。遍历的关键是‘不重不漏’：既不能重复检查同一个候选解，也不能漏掉任何一个可能的情况。
第三步：验证候选解。
对每个遍历到的候选解，用问题的条件判断它是否有效。如果有效，就记录下来。
这三步环环相扣：范围界定错了，遍历的就是无效空间；遍历漏了，会丢解；验证错了，会把无效解当成有效解。”

枚举概念

优点：
“最突出的优点是简单直接。
逻辑容易理解：不需要复杂的推导或公式，跟着‘遍历→验证’的思路写代码就行；
实现门槛低：只要会用循环和条件判断，就能写出枚举代码；
结果可靠：只要范围界定对、验证条件对，就不会漏解或错解。”
缺点：
最大的局限是效率，依赖解空间大小。
如果解空间太大（比如需要遍历 1 亿个候选解），即使每个验证只要 1 毫秒，总时间也会达到 1 亿毫秒（约 28 小时），完全不可接受；
本质上是暴力思路，没有利用问题的深层规律，所以在解空间大时，效率很低。



04

枚举例题

TEACHING
COURSEWARE

TEACH

枚举例题

水仙花数

题目描述
在自然数中，如果一个三位数等于自身各位数字之立方和，则这个三位数就称为是水仙花数。
如：153=1+125+27，所以153是一个水仙花数。
输入两个整数a，b，求这两个整数间的水仙花数数量。

输入格式
一行，两个整数，中间用空格隔开。

输出格式
一行，一个整数，表示a和b两个整数间的水仙花数数量。

输入：
283 964

输出：
3

Question：
这个题除了枚举思想外，还有没有其他的思想，体现在哪里。

枚举例题

水仙花数

Question:
这个题除了枚举思想外，还有没有其他的思想，体现在哪里。

```
int main() {
    int a, b;
    cin >> a >> b;
    int ans = 0;
    for (int num = a; num <= b; ++num) {
        if (check(num)) {
            ans++;
        }
    }

    cout << ans << endl;
    return 0;
}
```

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

// 判断一个数是否为水仙花数
bool check(int num) {
    int h = num / 100;          // 百位
    int t = (num / 10) % 10;    // 十位
    int u = num % 10;           // 个位
    int sum = h*h*h + t*t*t + u*u*u;
    return sum == num;
}
```

枚举例题

NOIP2016 普及组 T2 回文日期

题目描述

在日常生活中，通过年、月、日这三个要素可以表示出一个唯一确定的日期。

牛牛习惯用 8 位数字表示一个日期，其中，前 4 位代表年份，接下来 2 位代表月份，最后 2 位代表日期。显然：一个日期只有一种表示方法，而两个不同的日期的表示方法不会相同。

牛牛认为，一个日期是回文的，当且仅当表示这个日期的 8 位数字是回文的。现在，牛牛想知道：在他指定的两个日期之间（包含这两个日期本身），有多少个真实存在的日期是回文的。

一个 8 位数字是回文的，当且仅当对于所有的 i（1≤i≤8）从左向右数的第 i 个数字和第 9−i 个数字（即从右向左数的第 i 个数字）是相同的。

回文：正着反着都一样

例如：

对于 2016 年 11 月 19 日，用 8 位数字 20161119 表示，它不是回文的。

对于 2010 年 1 月 2 日，用 8 位数字 20100102 表示，它是回文的。

枚举例题

NOIP2016 普及组 T2 回文日期

输入：
20000101
20101231

输出：
2

符合条件的日期是 20011002 和 20100102。

Question：
我们有几种枚举策略，试对比一下这几种策略。

枚举例题

NOIP2016 普及组 T2 回文日期

Question:
我们有几种枚举策略，试对比一下这几种策略。

方法1：
日期其实就是8位数字，我们从数字a枚举到数字b是否可行呢。

方法2：
按照年，月，日枚举，可以大大减少枚举量。

Question:
分析一下方法1和2的优劣

对于样例：
20000101
29991231

方法1：
29991231−20000101+1=9991131 次

方法2：
2. 计算从 2000 年 1 月 1 日到 2999 年 12 月 31 日的总天数（所有合法日期的数量）。
总天数为：闰年天数 + 平年天数 = 243 * 366 + 757 * 365 = 365243天

按数字枚举的次数约是按年月日枚举的 27.3 倍，差异近 962 万次。这进一步说明：在处理特定问题时，优先枚举合法可能，比盲目遍历所有数字高效得多。

枚举总结

DAILY REPORT INTRODUCTION

枚举算法是遍历解空间→验证候选解的暴力思路，核心是不重不漏地列情况，按条件筛有效解

它适合解空间小、范围明确、验证简单的问题，优点是简单可靠，缺点是解空间大时效率低。

理解枚举的本质，不是为了盲目使用，而是为了在遇到问题时能判断：‘这个问题的解空间适合枚举吗？’—— 这才是掌握枚举算法的关键。

Q&A

Q：枚举时 “解空间” 和 “候选解” 分别指什么？如何界定解空间范围？

Q：为什么说 “遍历” 和 “验证” 缺一不可？

Q：回文日期例题中，按 “年 + 月 + 日” 枚举比按 8 位数字枚举高效的核心原因是什么？

Q：枚举算法效率低时，有哪些优化思路？