深度优先搜索

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.11

启航舱
中小学拔尖创新人才培养中心

链表和数组的区别包括（ ）。

- A. 数组不能排序，链表可以
- B. 链表比数组能存储更多的信息
- C. 数组大小固定，链表大小可动态调整
- D. 以上均正确

启航舱
中小学拔尖创新人才培养中心

目录

PART-01 概念 TEACHING COURSEWARE
PART-02 例题 TEACHING COURSEWARE
PART-03 教学过程 TEACHING COURSEWARE
PART-04 教学反思 TEACHING COURSEWARE

启航舱
中小学拔尖创新人才培养中心

01

概念

TEACHING
COURSEWARE

TEACH

概念

深度优先搜索（Depth-First Search，DFS）是一种针对状态空间的系统性搜索策略，其核心特征是 “优先深入探索”：在探索过程中，对于当前所处的状态，会优先选择其一个未被探索的邻接状态（即从当前状态可直接转移到的状态），并沿着该方向持续深入探索，直到该路径无法继续延伸（即到达一个没有未被探索的邻接状态的状态）；随后，通过 “回溯” 回到上一个状态，选择其另一个未被探索的邻接状态继续深入，重复这一 “深入 - 回溯” 过程，直至找到目标状态或遍历完所有可能的状态。

概念

核心要素解析：
状态空间：指问题涉及的所有可能状态的集合，每个状态代表问题在某一时刻的具体情况。
邻接状态：从当前状态出发，通过一次合法操作可直接到达的其他状态。
深入探索：优先沿着某一方向的邻接状态持续推进，尽可能“走到底”、。
回溯机制：当一条路径无法继续延伸时，退回至路径上的上一个状态，切换到该状态的其他未探索邻接状态，开启新的探索路径。

简言之，DFS 是 “一条路走到黑，走不通就回头换条路” 的策略，系统性地遍历状态空间以寻找目标，其核心是对 “深度” 的优先选择和回溯机制的结合。

启航舱
中小学拔尖创新人才培养中心

02

例题

TEACHING
COURSEWARE

TEACH

例题（全排列）

启航舱
中小学拔尖创新人才培养中心

题目描述

复制 Markdown　展开　进入 IDE 模式

按照字典序输出自然数 1 到 n 所有不重复的排列，即 n 的全排列，要求所产生的任一数字序列中不允许出现重复的数字。

输入格式

一个整数 n。

输出格式

由 $1 \sim n$ 组成的所有不重复的数字序列，每行一个序列。

每个数字保留 5 个场宽。

输入输出样例

输入 #1　复制

```
3
```

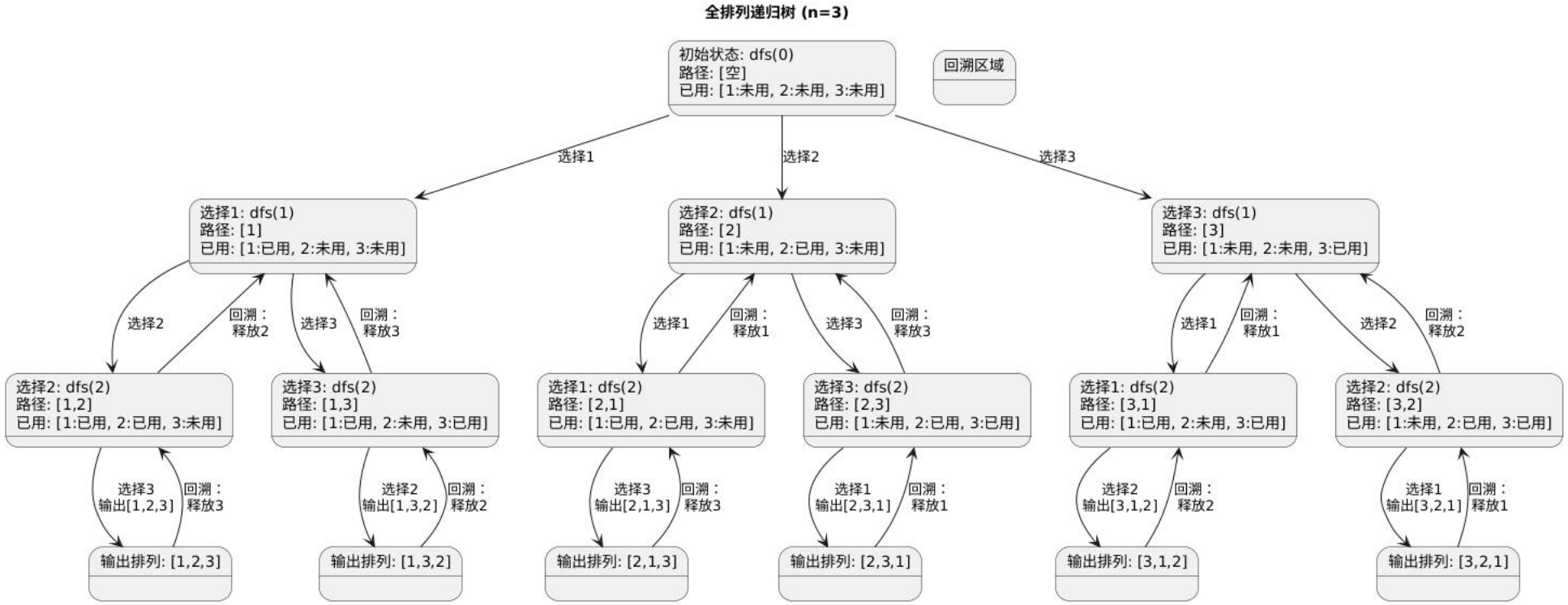
输出 #1　复制

```
    1    2    3
    1    3    2
    2    1    3
    2    3    1
    3    1    2
    3    2    1
```

说明/提示

$1 \leq n \leq 9$。

代码


全排列递归树 (n=3)
初始状态: dfs(0)
路径: [空]
已用: [1:未用, 2:未用, 3:未用]
回溯区域
选择1
选择2
选择3
选择1: dfs(1)
路径: [1]
已用: [1:已用, 2:未用, 3:未用]
选择2: dfs(1)
路径: [2]
已用: [1:未用, 2:已用, 3:未用]
选择3: dfs(1)
路径: [3]
已用: [1:未用, 2:未用, 3:已用]
选择2
回溯：释放2
选择3
回溯：释放3
选择1
回溯：释放1
选择3
回溯：释放3
选择1
回溯：释放1
选择2
回溯：释放2
选择2: dfs(2)
路径: [1,2]
已用: [1:已用, 2:已用, 3:未用]
选择3: dfs(2)
路径: [1,3]
已用: [1:已用, 2:未用, 3:已用]
选择1: dfs(2)
路径: [2,1]
已用: [1:已用, 2:已用, 3:未用]
选择3: dfs(2)
路径: [2,3]
已用: [1:未用, 2:已用, 3:已用]
选择1: dfs(2)
路径: [3,1]
已用: [1:已用, 2:未用, 3:已用]
选择2: dfs(2)
路径: [3,2]
已用: [1:未用, 2:已用, 3:已用]
选择3
输出[1,2,3]
回溯：释放3
选择2
输出[1,3,2]
回溯：释放2
选择3
输出[2,1,3]
回溯：释放3
选择1
输出[2,3,1]
回溯：释放1
选择2
输出[3,1,2]
回溯：释放2
选择1
输出[3,2,1]
回溯：释放1
输出排列: [1,2,3]
输出排列: [1,3,2]
输出排列: [2,1,3]
输出排列: [2,3,1]
输出排列: [3,1,2]
输出排列: [3,2,1]

启航舱
中小学拔尖创新人才培养中心

04

习题

TEACHING
COURSEWARE

TEACH

代码

```cpp
#include <iostream>
using namespace std;
int n;
int path[10];  // 存储当前排列（n最大为9，数组大小10足够）
bool used[10]; // 标记数字是否已使用（索引1~n有效）
// 深度优先搜索函数，参数step表示当前要确定第几个位置（从0开始）
void dfs(int step) {
    // 递归终止条件：已经确定了n个位置
    if (step == n) {
        // 输出当前排列，每个数字占5个场宽
        for (int i = 0; i < n; ++i) {
            printf("%5d", path[i]);
        }
        printf("\n");
        return;
    }
    // 尝试将1~n中未使用的数字放入当前位置
    for (int i = 1; i <= n; ++i) {
        if (!used[i]) {  // 如果数字i未被使用
            used[i] = true;        // 标记为已使用
            path[step] = i;        // 将数字i放入当前位置
            dfs(step + 1);         // 递归处理下一个位置
            used[i] = false;       // 回溯：取消标记
        }
    }
}
```

代码

启航舱
中小学拔尖创新人才培养中心

```
int main() {
    cin >> n;
    // 初始化used数组（默认false，可不显式初始化）
    dfs(0);  // 从第0个位置开始递归
    return 0;
}
```

启航舱
中小学拔尖创新人才培养中心

03

习题

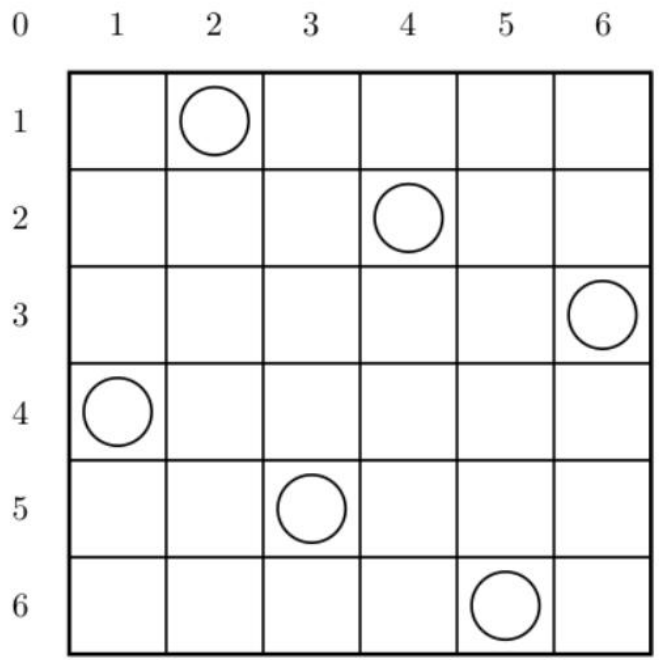
TEACHING
COURSEWARE

TEACH

题目描述

复制 Markdown 展开 进入 IDE 模式

一个如下的 6×6 的跳棋棋盘，有六个棋子被放置在棋盘上，使得每行、每列有且只有一个，每条对角线（包括两条主对角线的所有平行线）上至多有一个棋子。



洛谷

上面的布局可以用序列 $2\ 4\ 6\ 1\ 3\ 5$ 来描述，第 i 个数字表示在第 i 行的相应位置有一个棋子，如下：

行号 $1\ 2\ 3\ 4\ 5\ 6$

列号 $2\ 4\ 6\ 1\ 3\ 5$

启航舱
中小学拔尖创新人才培养中心

这只是棋子放置的一个解。请编一个程序找出所有棋子放置的解。
并把它们以上面的序列方法输出，解按字典顺序排列。
请输出前 3 个解。最后一行是解的总个数。

输入格式

一行一个正整数 n，表示棋盘是 $n \times n$ 大小的。

输出格式

前三行为前三个解，每个解的两个数字之间用一个空格隔开。第四行只有一个数字，表示解的总数。

输入输出样例

输入 #1 复制

```
6
```

输出 #1 复制

```
2 4 6 1 3 5
3 6 2 5 1 4
4 1 5 2 6 3
4
```

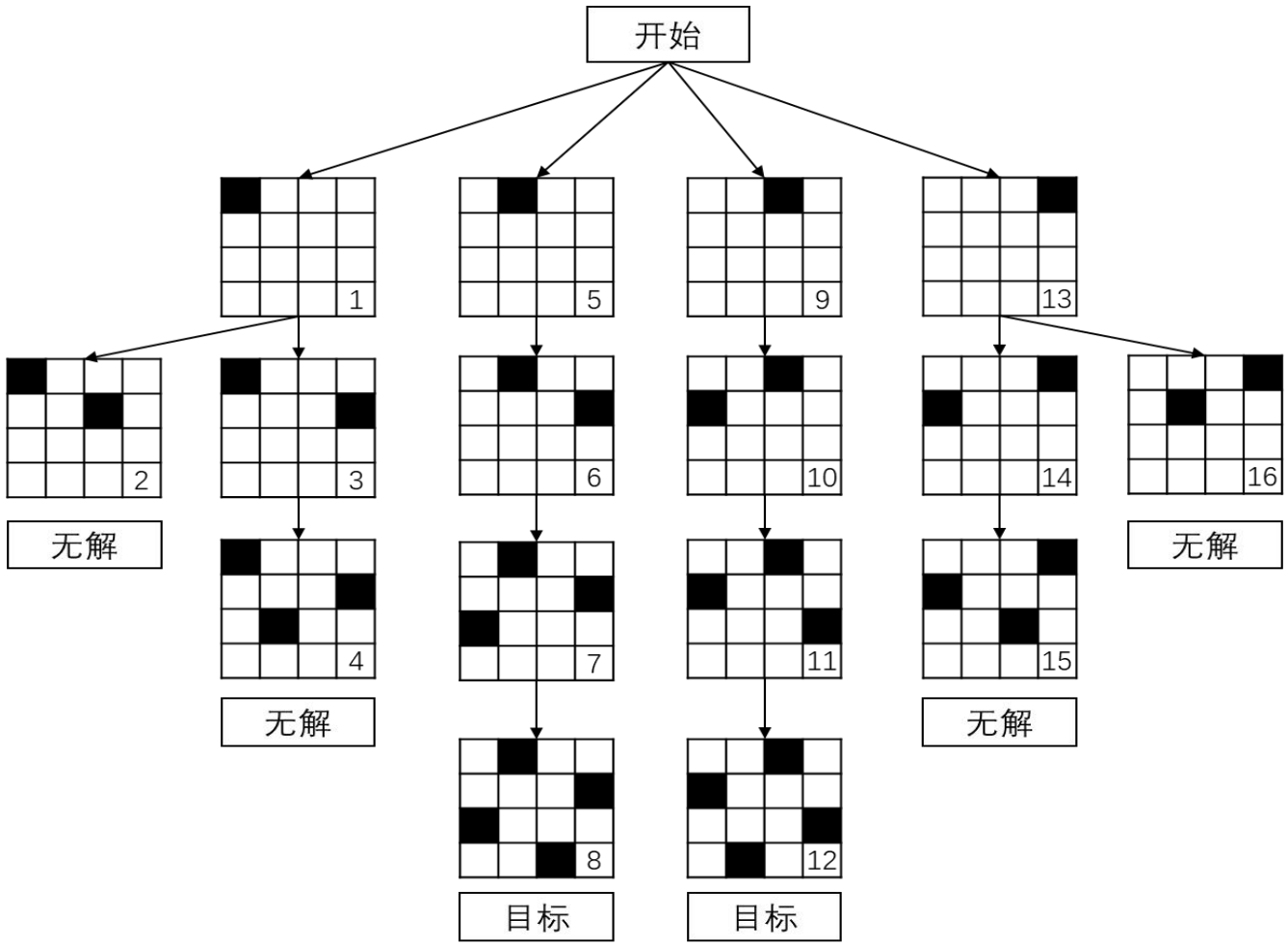
说明/提示

【数据范围】
对于 100% 的数据，$6 \leq n \leq 13$。

题目翻译来自NOCOW。

USACO Training Section 1.5

启航舱
中小学拔尖创新人才培养中心



启航舱
中小学拔尖创新人才培养中心

```
void dfs(int x)
{
    // 递归终止条件：当x > n时，说明前n行皇后已全部放置完毕，形成一个有效方案
    if (x > n)
    {
        // 若当前输出的方案数小于3，则打印该方案
        if (total < 3)
        {
            for (int i = 1; i <= n; i++)
            {
                cout << a[i] << " ";  // 输出第i行皇后所在的列号
            }
            cout << endl;
            total++;  // 方案数加1
            return;
        }
        else
        {
            total++;  // 超过3个方案后，仅计数不打印
        }
    }
```

启航舱
中小学拔尖创新人才培养中心

```
else
{
    // 尝试在第x行的第i列放置皇后（i从1到n遍历）
    for (int i = 1; i <= n; i++)
    {
        // 检查是否冲突：
        // 1. 第i列无皇后（b[i] == 0）
        // 2. 主对角线（i+x）无皇后（c[i+x] == 0）
        // 3. 副对角线（x-i+n）无皇后（d[x-i+n] == 0）
        if (!b[i] && !c[i + x] && !d[x - i + n])
        {
            a[x] = i;  // 在第x行第i列放置皇后

            // 标记冲突：当前列、主对角线、副对角线已被占用
            b[i] = 1;
            c[i + x] = 1;
            d[x - i + n] = 1;

            dfs(x + 1);  // 递归放置第x+1行的皇后

            // 回溯：撤销当前放置，恢复标记（供其他方案尝试）
            b[i] = 0;
            c[i + x] = 0;
            d[x - i + n] = 0;
        }
    }
}
```

启航舱
中小学拔尖创新人才培养中心

```cpp
#include<bits/stdc++.h>
using namespace std;
int n;                          // 表示n皇后问题中的n（棋盘大小为n× n）
const int N = 1e3 + 10; // 数组最大容量，用于存储皇后位置及冲突标记
// 全局数组作用：
// a[x] = i 表示第x行的皇后放在第i列
// b[i] 标记第i列是否已有皇后（1表示有，0表示无）
// c[i + x] 标记主对角线（从左上到右下）是否已有皇后（i+x为对角线唯一标识）
// d[x - i + n] 标记副对角线（从右上到左下）是否已有皇后（x-i+n避免负数索引）
int a[N], b[N], c[N], d[N];
int total;                      // 记录所有可行的皇后摆放方案总数
// 深度优先搜索函数：递归放置第x行的皇后
```

启航舱
中小学拔尖创新人才培养中心

```
int main()
{
    cin >> n;          // 输入n，确定皇后数量和棋盘大小
    dfs(1);            // 从第1行开始放置皇后
    cout << total << endl;  // 输出所有可行方案的总数
    return 0;
}
```