基础算法 · 第五章

搜索与回溯算法

枚举所有可能，再聪明地放弃

HGNU-ACM 实验室算法组

截图配套：3 道例题，11 道练习题

本章目录 | 前半

从枚举所有可能开始

DFS 的基本框架

排列、组合与子集

剪枝与可行性

例 5.1　素数环

本章目录｜后半

例 5.2　组合的输出

例 5.3　自然数的拆分

棋盘、迷宫与常见模型

易错点与模板

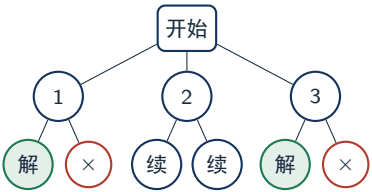
本章全部练习题

什么是搜索：在岔路口系统地试

一句话定义

搜索：把“求一个方案”拆成一次次选择；每个岔路口枚举候选，走不通就退回。

- 状态：现在走到哪了。
- 候选：下一步能选什么。
- 边界：什么时候停。



绿：可行；红：死路

本页要点

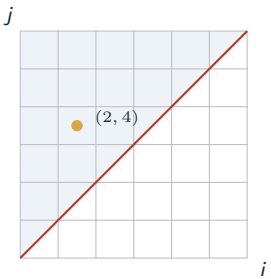
搜索 = 有条理地枚举；回溯 = 走不通时撤销选择、退回分叉点。

枚举：最朴素的搜索

- 例：输出所有满足 $1 \le i < j \le n$ 的数对。
- 固定 i，让 j 从 $i+1$ 走到 n，顺序固定，不重不漏。

```
for (int i = 1; i <= n; ++i)
    for (int j = i + 1; j <= n; ++j)
        printf("%d %d\n", i, j);
```

两层循环 $=$ 两层选择；$n = 6$ 时共 15 对。

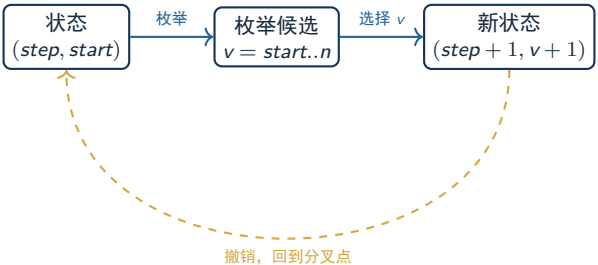


本页要点

枚举的三件事：定顺序、定范围、保证不重不漏。

状态：把“走到哪了”说清楚

- 状态是决定“下一步能怎么选”的最少信息；它把历史压缩成几个参数。
- 组合：$(step, start)$；迷宫：(x, y)；八皇后：$(r, \mathrm{col}, d_1, d_2)$。

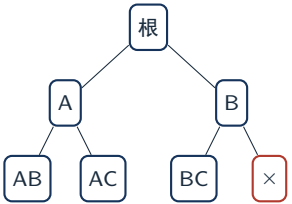


好状态的三条标准

能判断候选是否合法；能推出下一步；不会把两个不同局面混成同一个状态。

解空间与搜索树：把所有方案画出来

- 解空间：所有候选方案组成的集合，包括合法与不合法。
- 把“部分方案”画成结点，把“一次选择”画成边，就得到搜索树。
- 根是空方案；内部结点是部分方案；叶子是完整方案。
- 搜索：从根出发访问整棵树，在叶子上筛选答案。

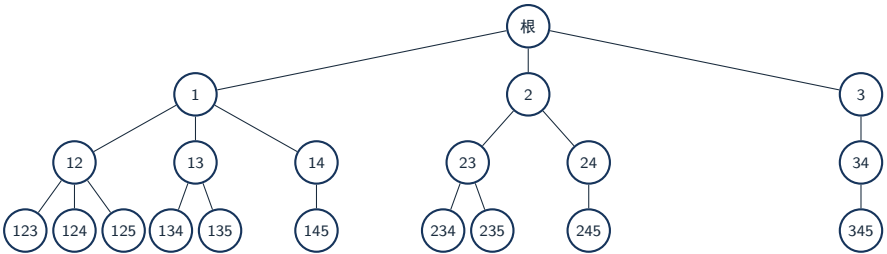


叶子可能合法，也可能不合法

本页要点

搜索树不是数据结构课上的“树”，而是把选择过程画出来的思维工具。

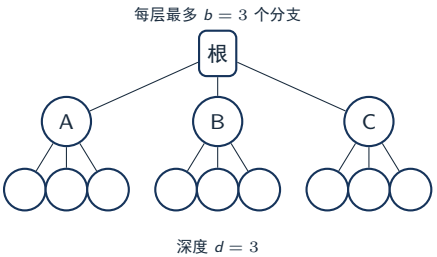
搜索树实例：从 5 个数里选 3 个



- 第 1 层选第一个数，第 2 层选第二个数，第 3 层选第三个数。
- 每个叶子是一条从根到叶的路径，恰好对应一个组合；共 $\binom{5}{3} = 10$ 个叶子。

搜索树有多大：分支因子与深度

- 设树深 d，每层最多 b 个候选。
- 结点总数最多为 $1 + b + b^2 + \cdots + b^d$。
- 叶子最多 b^d 个；每个叶子还要花时间输出或检查。



估算顺序

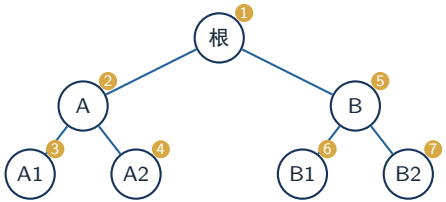
先数每层候选数，再数深度，最后乘上每个结点的工作量。

本页要点

搜索的复杂度首先看树有多大，其次看每个结点做多少事。

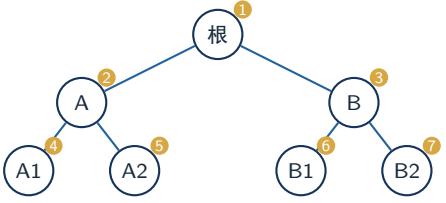
DFS 与 BFS：两种系统地走法

DFS：栈 / 递归



- 一条路走到底，再回头。
- 空间 $O(d)$；适合枚举方案、找一组解、连通块。

BFS：队列

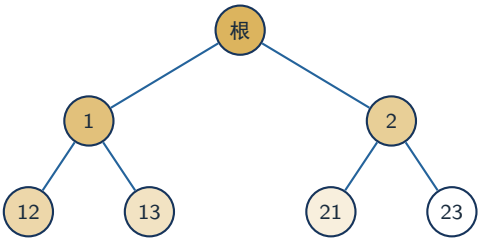


- 一层层扩展，先到先扩展。
- 空间 O(宽度)；适合无权最短路、最少步数。

本页要点

要“所有方案”或“任意一组解”用 DFS；要“最少步数”用 BFS。

DFS 的执行过程：进入、到底、返回



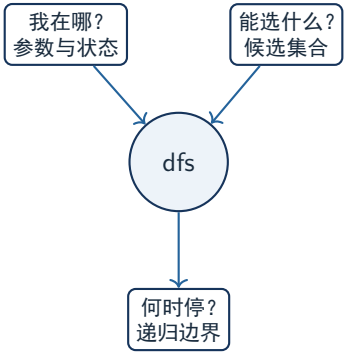
- 访问顺序：根 $\to$ 1 $\to$ 12 $\to$ 13 $\to$ 2 $\to$ 21 $\to$ 23。
- 到叶子后，递归函数返回上一层，`for` 循环继续试下一个候选。
- 返回时执行撤销，把状态恢复到进入这一层之前。

本页要点

DFS 的三个动作：选择（进入）、递归（深入）、撤销（返回）。

递归函数要回答三个问题

1. 我在哪？用参数写出当前状态：填到第几位、还剩多少、在哪个格子。
2. 下一步能选什么？列出全部候选，逐个尝试，不能漏。
3. 什么时候停？写清边界：填满、走到终点、剩余为 0。



顺序不能反

先判断边界，再枚举候选；否则会在已经完成的状态上继续扩展。

DFS 通用模板：五件事写全

```
void dfs(int step, int start) { // step: 已选个数
    if (step == r) { // 边界：选够 r 个
        output(); // 记录或输出答案
        return; // 返回上一层
    }
    for (int v = start; v <= n; ++v) { // 枚举候选
        if (!ok(v)) continue; // 合法性检查
        choose(v); // 选择：标记、记录
        dfs(step + 1, v + 1); // 递归进入下一层
        undo(v); // 撤销：恢复现场
    }
}
```

- 边界、候选、合法性、选择、撤销，五件事缺一不可。
- choose 与 undo 必须成对出现。

选择与撤销：把状态恢复原样

- 回溯：退回分叉点时，把这次选择造成的影响全部撤销。
- 搜索树的不同分支共用同一份数组；进入分支时修改，退回时必须改回。
- 例：进入时 `used[3]=true`，退回后若不改回，其他分支就永远用不了 3。



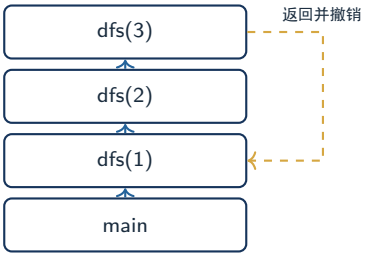
蓝色：选择；金色虚线：撤销。每一步只改一个位置。

本页要点

不撤销 = 把上一条路用过的资源永远占住，后面的分支全被堵死。

递归调用栈：撤销发生在返回时

- 每调用一次 `dfs`，就压入一个栈帧；返回时弹出。
- 局部变量随栈帧消失；全局数组和标记不会自动恢复，必须手动撤销。
- 递归深度就是栈的深度，太深会栈溢出。

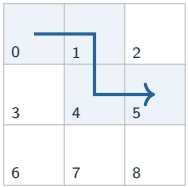


本页要点

函数返回只销毁局部变量；数组里的标记要自己擦掉。

状态数组：把“能不能选”变成一次查询

- 用 `bool used[N]` 标记数字是否用过；用 `vis[N][M]` 标记格子是否走过。
- 标记后查询是 $O(1)$，比每次扫描一遍快得多。
- 进入分支置真，退回置假；只求连通块时可以不撤销。



网格编号

按编号展平成一维数组

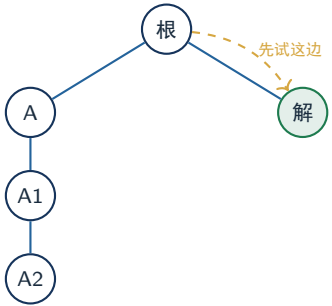
1	1	0	0	1	1	0	0	0

本页要点

状态数组是搜索的“记忆”；记忆用完后要记得擦掉。

搜索顺序：先试谁，效果可能差很多

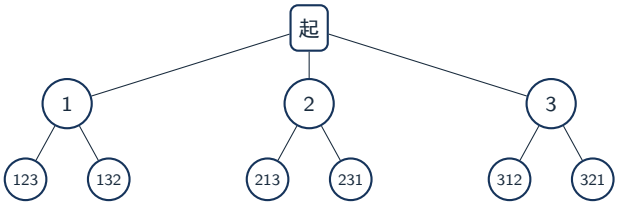
- 候选顺序不改变“是否完备”，但改变找到第一组解的速度。
- 输出题要求字典序时，顺序本身就是题意的一部分。
- 最优化题先试更有希望的候选，能更早拿到好的当前最优解。
- 固定起点、强制升序，可以消掉旋转和排列造成的重复。



本页要点

顺序不影响“能不能找到”，只影响“多快找到”和“答案按什么顺序出现”。

排列：位置有序，数字不重复



- 第 i 个位置从 $1..n$ 中选一个没用过的数；每个数恰好用一次。
- 用 `used[v]` 标记，选完递归，返回时撤销。
- n 个位置共有 $n!$ 个叶子；输出每组还要 $O(n)$，时间 $O(n \cdot n!)$。

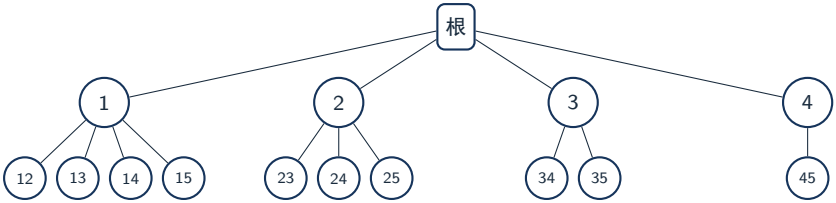
本页要点

排列的关键词：位置有序，数字不重复，用 `used` 去重。

排列模板：used 数组与撤销

```
int n, a[15];
bool used[15];
void dfs(int pos) { // pos：当前填第几位
    if (pos == n) { // 边界：n 位都填满
        print(); // 输出当前排列
        return;
    }
    for (int v = 1; v <= n; ++v) {
        if (used[v]) continue; // 用过的数不能重复选
        used[v] = true; // 选择 v
        a[pos] = v; // 记录到第 pos 位
        dfs(pos + 1); // 填下一位
        used[v] = false; // 撤销
    }
}
```

组合：顺序无关，强制升序



- 组合不分顺序。规定后选的数必须比前一个大，每种组合只出现一次。
- 递归参数带上 `start`：下一层从 `start` 开始枚举。
- $n = 5, r = 2$ 的搜索树只有 $\binom{5}{2} = 10$ 个叶子。

本页要点

组合的关键词：用 `start` 保证升序，避免同一组合被反复计数。

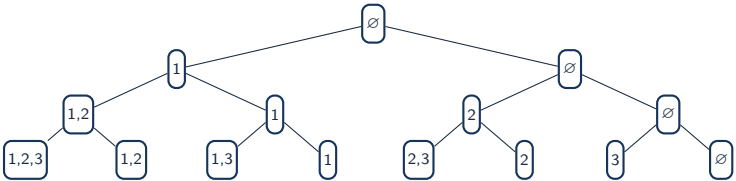
组合模板：start 与上界

```
int n, r, a[25];

void dfs(int step, int start) { // step: 已选个数
    if (step == r) { // 边界：选够 r 个
        for (int i = 0; i < r; ++i)
            cout << a[i] << ' ';
        cout << '\n';
        return;
    }
    for (int v = start; v <= n; ++v) {
        a[step] = v; // 选择：第 step 位放 v
        dfs(step + 1, v + 1); // 下一层只能选更大的数
    }
}
```

- 下一层传 v+1 而不是 start，否则会退化成排列。
- 上界可收紧到“给后面留够数”，例题会讲。

子集：每个元素“选”或“不选”



- 每个元素只有“加入 / 不加入”两种选择，搜索树是二叉树。
- n 个元素共有 2^n 个叶子；子集、背包都能套这个模型。

本页要点

子集的关键词：二叉分叉，叶子数 2^n。

子集模板：两个递归分支

```
int n, a[25];
bool chosen[25];
void dfs(int i) { // i：正在决定第 i 个元素
    if (i == n) { // 所有元素都决定完
        for (int j = 0; j < n; ++j)
            if (chosen[j]) cout << a[j] << ' ';
        cout << '\n';
        return;
    }
    chosen[i] = true; // 分支 1：选第 i 个
    dfs(i + 1);
    chosen[i] = false; // 恢复现场
    dfs(i + 1); // 分支 2：不选第 i 个
}
```

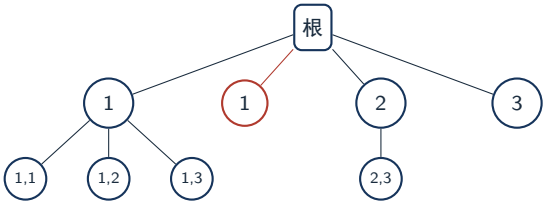
- 两个分支对应两条边，叶子数 2^n；递归写法也更容易加剪枝。

三种模型怎么选

模型	每层候选	去重办法	叶子数
排列	所有未用过的元素	used 数组	$n!$
组合	从 start 到 n	强制升序	$\binom{n}{r}$
子集	选 / 不选	固定元素顺序	2^n

- 顺序有关 $\rightarrow$ 排列；顺序无关且选固定个数 $\rightarrow$ 组合。
- 每个元素只有“要 / 不要” $\rightarrow$ 子集；棋盘逐行放置 $\rightarrow$ 棋盘模型。
- 先写出“叶子长什么样”，再决定每层枚举什么。

有重复元素：排序后跳过同一层的相同值



- 先排序，让相同的数挨在一起。
- 同一层枚举时，如果 `i > start` 且 `a[i] == a[i-1]`，就跳过。
- 图中红色的 1 与同层第一个 1 重复，整枝跳过。
- 跳过的是同一层的重复，不同层仍然可以使用相同的值。

本页要点

同一层去重，不同层保留；先排序，再搜索。

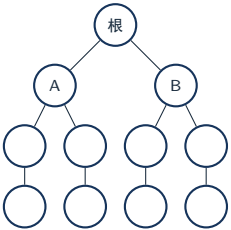
去重模板：i > start 判断

```
int n, r, a[25], x[25];
void dfs(int step, int start) {
    if (step == r) { // 选够 r 个
        print(x);
        return;
    }
    for (int i = start; i < n; ++i) {
        if (i > start && a[i] == a[i-1]) continue; // 同层重复，跳过
        x[step] = a[i]; // 选择
        dfs(step + 1, i + 1); // 下一层从 i+1 开始
    }
}
```

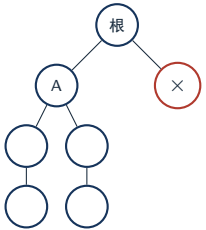
- 排列去重的条件不同：a[i]==a[i-1] && !used[i-1] 时跳过。
- 先画一遍小例子，确认被跳过的分支确实和前面的分支等价。

为什么需要剪枝

- 搜索树可能按指数增长：$2^{20} \approx 10^6$，$10! \approx 3.6 \times 10^6$。
- 很多分支从第一步起就不可能产生答案，继续走只是浪费时间。
- 剪枝：在进入分支前，用廉价判断排除“没有希望”的子树。



不剪枝：15 个结点



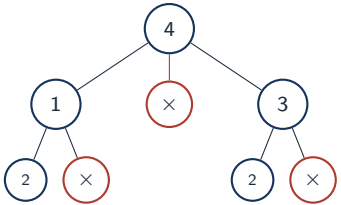
剪枝后：9 个结点

本页要点

剪枝不改变答案，只改变“走多少冤枉路”。

可行性剪枝：这一步已经不可能

- 判断依据：当前部分方案已经违反题意，后面怎么补都救不回来。
- 素数环：相邻两数之和不是素数，立刻换下一个候选。
- 迷宫：越界、撞墙、重复访问，立刻返回。
- 组合：后面位置不够，直接缩小枚举上界。



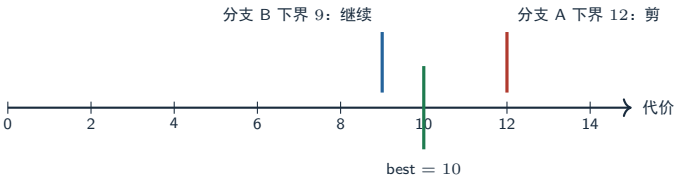
起点 4：$4+2=6$ 不是素数，整枝剪掉

本页要点

可行性剪枝的条件必须“充分”：被剪掉的分支确实不可能有解。

最优性剪枝：已经不可能更好

- 维护当前找到的最优答案 best。
- 若当前分支的代价下界已经 $\geq$ best，继续走也不会更优，剪掉。
- 例：分成互质组，当前组数已经不少于最优组数，停止。



本页要点

最优性剪枝必须能证明“再走也不会更好”，否则会剪掉正确答案。

顺序剪枝：用 start 和固定起点消重复

- 组合与拆分：下一层只从 start（或 low）往后选，强制非递减。
- 环与排列：固定起点或固定第一个元素，消掉旋转造成的重复。
- 这是“避免重复枚举”，不是“排除合法方案”。



本页要点

先排序、再规定“只能往后选”，每种组合就只有一条路径。

剪枝的正确性：不能漏解

- 计数题要求“一个不漏”，剪枝条件必须充分。
- 只有能证明“这个分支无解”或“不会更优”时才能剪。
- 不确定时，先写完整搜索，再用小数据和暴力程序对拍。

分支	剪掉	保留
有解	漏解	正确
无解	正确	低效

判断口诀

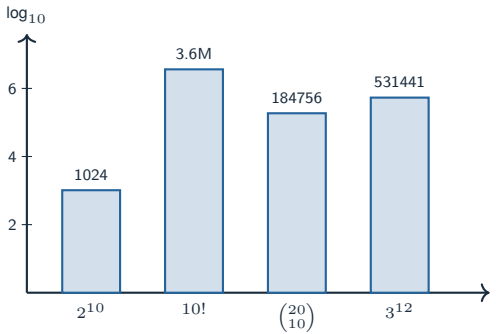
先问“剪掉它，会不会漏掉答案”；答不上来，就先别剪。

本页要点

剪枝是加速手段，不是正确性手段；正确性由“枚举完整”保证。

复杂度估算：先数结点，再乘工作量

- 时间 = 搜索树结点数 × 每个结点的工作量。
- 空间 = 递归深度 + 标记数组。
- 输出本身可能很大，复杂度要和输出规模一起算。



经验范围

2^n：$n \le 20$；$n!$：$n \le 10$；$O(nm)$ 网格：$n, m \le 1000$。

本页要点

搜索题先估算树的大小；树太大，就要剪枝或换算法。

写搜索前，先过四道检查

1. 状态参数够不够？能不能唯一确定“现在走到哪”。
2. 候选范围对不对？有没有漏选或重复枚举。
3. 边界覆盖全不全？空集、单元素、$r = n$、起点即终点。
4. 撤销完不完整？标记数组、计数器、临时答案都要恢复。

写代码前的一句话

先用一句话说清搜索树长什么样：每层选什么，叶子是什么。

本页要点

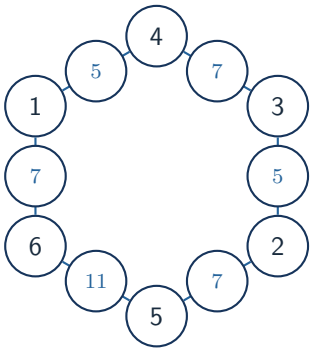
搜索题的错误，多半不是“算错”，而是“漏了一种情况”或“多算了一次”。

例 5.1 | 素数环：题意与样例

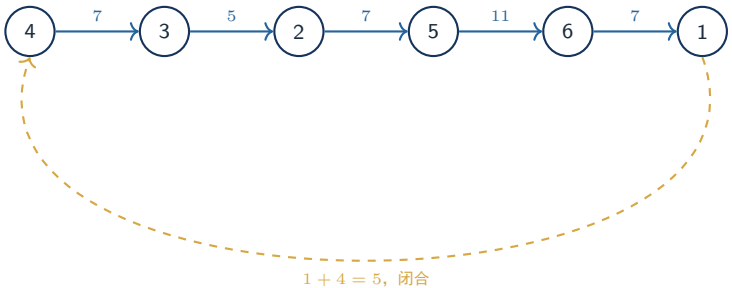
题意

输入 n（$4 \le n \le 30$），把 $1, 2, \ldots, n$ 排成一个环，使相邻两个整数之和都是素数。输出任意一个合法环。

- 样例：输入 6，输出 4 3 2 5 6 1。
- 六条边之和为 $7, 5, 7, 11, 7, 5$，全部是素数。
- 首尾也相邻，最后一条边必须单独检查。
- 旋转算同一种环，可固定起点。



例 5.1 | 样例手推：沿着一条路填下去



- 固定起点为 $n-2=4$，依次填后面的数；每一步只检查“与前一个数之和”。
- 走不通就退回上一位换候选；填满 n 个数后再检查首尾边。
- 样例路径：$4 \to 3 \to 2 \to 5 \to 6 \to 1$，六条边全部合法。

例 5.1 | 搜索思路：固定起点，逐个填环

1. 固定 `ring[0] = n-2`，并把 `used[n-2]` 置真。
2. `dfs(pos)`：填第 `pos` 个位置。
3. 候选 v：没用过，且 `ring[pos-1]+v` 是素数。
4. 填满后返回 `isPrime[ring[n-1]+ring[0]]`。
5. 找到一组就立刻返回，不再继续搜索。

为什么只查相邻边

前面的边在填数时已经逐条检查；填满后只剩首尾这一条没查。

本页要点

搜索顺序：固定起点 $\rightarrow$ 逐位填数 $\rightarrow$ 闭合检查。

例 5.1 | 完整程序（1/2）：搜索函数

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, ring[35];
bool used[35], isPrime[65];
bool dfs(int pos) { // pos：正在填第几个位置
    if (pos == n) // n 个数都填完了
        return isPrime[ring[n-1] + ring[0]]; // 检查首尾边
    for (int v = 1; v <= n; ++v) {
        if (used[v]) continue; // v 已经用过
        if (!isPrime[ring[pos-1] + v]) continue; // 相邻和必须是素数
        used[v] = true; // 选择 v
        ring[pos] = v;
        if (dfs(pos + 1)) return true; // 找到一组就返回
        used[v] = false; // 撤销
    }
    return false; // 这个分支无解
}
```

例 5.1 | 完整程序（2/2）：主函数

```
int main() {
    cin >> n;
    for (int x = 2; x <= 2 * n; ++x) { // 预处理素数表
        isPrime[x] = true;
        for (int d = 2; d * d <= x; ++d)
            if (x % d == 0) isPrime[x] = false;
    }
    used[n-2] = true; // 固定起点，消掉旋转重复
    ring[0] = n - 2;
    if (!dfs(1)) return 0; // 无解则直接结束
    for (int i = 0; i < n; ++i)
        cout << ring[i] << (i + 1 == n ? '\n' : ' ');
    return 0;
}
```

- 素数表只算到 $2n$：相邻和最大为 $n + (n-1) = 2n - 1$。
- 输出时最后一个数后换行，其余数后加空格。

例 5.1 | 复杂度与易错点

- 最坏情况是指数级搜索；$n \le 30$ 且合法环很多，实际很快找到一组。
- 素数表开到 $2n$；相邻和最大为 $2n-1$，不会越界。
- 奇数 $n > 1$ 无解：环长是奇数时，奇偶必须交替，奇数个位置无法闭合。
- 易错：忘记首尾边；忘记撤销 `used`；输出多一个或少一个空格。

本页要点

环问题三件事：固定起点、逐边判合法、最后闭合。

例 5.2 | 组合的输出：题意与样例

题意

输入 n, r，按字典序输出从 $1..n$ 中任取 r 个数的所有组合。每个组合占一行，元素升序，每个元素占三个字符的位置。

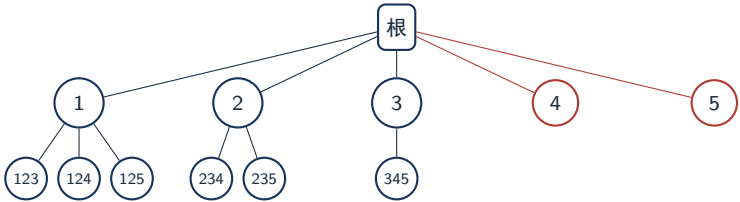
- 范围：$1 < n < 21$，$1 \le r \le n$；样例 $n = 5, r = 3$，共 $\binom{5}{3} = 10$ 行。
- 先固定第 1 位为 1，再把第 2 位从 2 开始增大，最后增大第 3 位。
- 第 1 位为 1 的全部输出完，才把第 1 位改成 2。

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

本页要点

组合题的两个关键词：升序、从 `start` 继续；输出格式也是题意的一部分。

例 5.2 | 搜索树与上界剪枝



- 第 step 层选 v 后，还要选 $r-\texttt{step}$ 个数。
- 所以 v 最大只能到 $n-(r-\texttt{step})$，再大就留不够位置。
- 这个上界就是一条简单的可行性剪枝；不写它，深层也会被边界挡住，但白跑很多。

例 5.2 | 完整程序（1/2）：搜索函数

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, r, a[25];
void dfs(int step, int start) { // step: 已选个数
    if (step > r) { // 已经选够 r 个
        for (int i = 1; i <= r; ++i)
            printf("%3d", a[i]); // 每个数占 3 个字符
        putchar('\n');
        return;
    }
    int last = n - (r - step); // 还要给后面留位置
    for (int v = start; v <= last; ++v) {
        a[step] = v; // 选择 v
        dfs(step + 1, v + 1); // 下一层从 v+1 开始
    }
}
```

例 5.2 | 完整程序（2/2）：主函数

```
int main() {
    cin >> n >> r;
    dfs(1, 1); // 从第 1 位、候选 1 开始
    return 0;
}
```

- 本题下标从 1 开始，所以边界写 `step > r`，不是 `step == r`。
- `a[step] = v` 后不需要撤销：下一层会覆盖这一位。
- `%3d` 保证每个数占三个字符，和样例对齐。

本页要点

先保证升序，再谈剪枝；输出格式也是题意的一部分。

例 5.2 | 复杂度与易错点

- 叶子数 $\binom{n}{r}$，每个叶子输出 r 个数，时间 $O\left(r\binom{n}{r}\right)$。
- 递归深度为 r，额外空间 $O(r)$（不计输出）。
- 易错 1：下一层从 1 开始而不是 `v+1`，会输出排列而不是组合。
- 易错 2：上界写成 n，会枚举出位置不够的分支，做很多无用功。
- 易错 3：用空格分隔而不是 `%3d`，样例格式不对。

本页要点

先保证升序，再谈剪枝；输出格式也是题意的一部分。

例 5.3 | 自然数的拆分：题意

题意

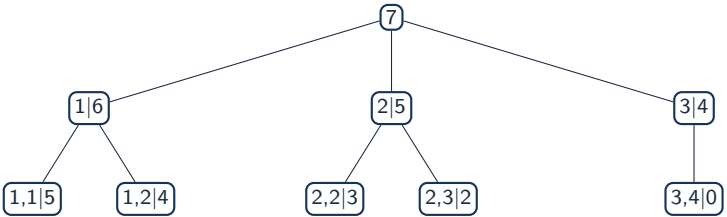
把大于 1 的自然数 n 拆成若干个小于 n 的自然数之和，按字典序输出所有方案。

- 样例 $n = 7$，共 14 种方案，从 `7=1+1+...+1` 到 `7=3+4`。
- 加数按非递减顺序排列：$1 + 2 + 4$ 可以，$2 + 1 + 4$ 不重复输出。
- 不允许 `7=7` 这种“不拆”的方案。

本页要点

拆分题的关键：让加数非递减，一种方案只走一条路径。

例 5.3 | 样例手推：为什么从 low 开始枚举



- 结点中的 $a|b$ 表示“当前已选的加数为 a，还剩 b”；其余分支略。
- 每层从上一层选的数 `low` 开始枚举，保证非递减。
- 选 v 后，剩余 `rem-v`，下一层最小仍为 v。
- 剩余为 0 时输出；只选了一个数（`dep<2`）时不算拆分。

例 5.3 | 完整程序（1/2）：搜索函数

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, a[35], dep; // dep: 当前加数个数
void dfs(int rem, int low) { // rem: 剩余；low: 最小可选
    if (rem == 0) { // 剩下的数拆完了
        if (dep < 2) return; // 至少两个加数
        printf("%d=", n);
        for (int i = 0; i < dep; ++i)
            printf("%d%c", a[i], i + 1 == dep ? '\n' : '+');
        return;
    }
    for (int v = low; v <= rem; ++v) { // 保证加数非递减
        a[dep++] = v; // 选择 v
        dfs(rem - v, v); // 下一层最小仍是 v
        --dep; // 撤销
    }
}
```

例 5.3 | 完整程序（2/2）：主函数

```
int main() {
    cin >> n;
    dfs(n, 1); // 从剩余 n、最小加数 1 开始
    return 0;
}
```

- dep 是全局计数器，进入时加一，返回时减一，必须成对。
- 输出格式由 i+1==dep 判断：最后一个加数后换行，其余加号。
- 只选一个数时提前返回，避免输出 n=n。

本页要点

非递减 + 剩余归零 + 至少两个加数 = 正确的拆分。

例 5.3 | 复杂度与易错点

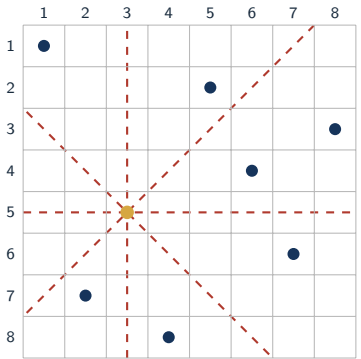
- 搜索树的大小等于拆分数的数量；n 较小时增长很快，但本题只要求输出所有方案。
- 每个合法方案输出一次，时间与方案总长度成正比。
- 易错 1：下一层从 1 开始，导致 $1+2+4$ 和 $2+1+4$ 重复。
- 易错 2：忘记 `dep<2` 判断，多输出 `7=7`。
- 易错 3：先输出再加号或换行，导致行末格式错误。

本页要点

非递减 $+$ 剩余归零 $+$ 至少两个加数 $=$ 正确的拆分。

八皇后：逐行放，标记列与两条对角线

- 一行只放一个皇后，逐行递归。
- 候选列不能有同列、同对角线冲突（金色为当前皇后）。
- 列用 `col[c]`，两条对角线用 $r+c$ 和 $r-c+8$ 编号。
- 放到第 9 行说明得到一组解；八皇后共 92 组。



本页要点

棋盘搜索：把“行、列、对角线”都变成可以直接查询的标记数组。

八皇后：对角线编号怎么来的

- 同一条“/”对角线上的格子，$r+c$ 相同。
- 同一条“\”对角线上的格子，$r-c$ 相同。
- 加一个偏移量 $+8$，把 $r-c$ 变成正下标。
- 三个布尔数组各管一类冲突，查询都是 $O(1)$。

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

$r+c$ $r-c+4$

本页要点

两条对角线各有一套编号；编号相同就会互相攻击。

八皇后：核心递归

```
int col[9], d1[17], d2[17]; // 列、/ 对角线、\ 对角线
int row[9], total;
void dfs(int r) { // 正在放第 r 行
    if (r == 9) { // 8 行都放好了
        ++total;
        if (total <= 3) print(); // 按题意输出前 3 组
        return;
    }
    for (int c = 1; c <= 8; ++c) {
        if (col[c] || d1[r+c] || d2[r-c+8]) continue;
        col[c] = d1[r+c] = d2[r-c+8] = 1; // 选择：占领列和两条对角线
        row[r] = c;
        dfs(r + 1);
        col[c] = d1[r+c] = d2[r-c+8] = 0; // 撤销
    }
}
```

八皇后：复杂度与易错点

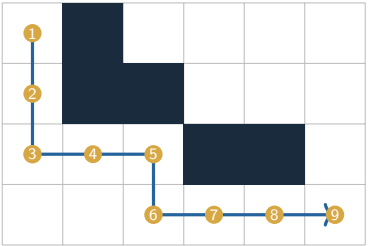
- 八皇后只有 92 组解，逐行搜索很快；n 皇后最坏仍是指数级。
- 每行最多 8 个候选，树高 8，结点数远小于 8^8，因为冲突剪枝很强。
- 易错 1：对角线下标越界，数组要开到 17。
- 易错 2：只标记列，忘记两条对角线。
- 易错 3：撤销不完整，下一行沿用上一行的标记。

本页要点

逐行放 $+$ 三个标记数组 $+$ 成对撤销 $=$ 八皇后模板。

迷宫：网格上的 DFS

- 从起点出发，向上下左右走。
- 不越界、不穿墙、不重复访问。
- 只问“能不能到”用 DFS/BFS 都行；问“最少步数”用 BFS。
- 多组数据要清空标记和答案。



数字表示访问顺序；黑格是墙

本页要点

网格题先写方向数组和越界判断，再写访问标记。

迷宫：路径计数程序

```
int n, m, sx, sy, ans;
char g[105][105];
bool vis[105][105];
int dx[4] = {-1, 1, 0, 0}; // 上、下、左、右
int dy[4] = {0, 0, -1, 1};
void dfs(int x, int y) {
    if (x < 1 || x > n || y < 1 || y > m) return; // 越界
    if (g[x][y] == '#' || vis[x][y]) return; // 墙或已访问
    vis[x][y] = true; // 标记已访问
    if (g[x][y] == 'T') { // 到达终点
        ++ans;
    } else {
        for (int k = 0; k < 4; ++k) // 四个方向
            dfs(x + dx[k], y + dy[k]);
    }
    vis[x][y] = false; // 撤销，换一条路
}
```

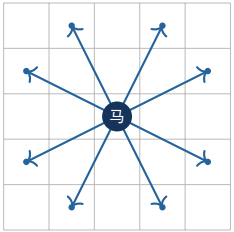
红与黑：只求连通块，不必撤销

```
int w, h, sx, sy, ans;
char g[25][25];
bool vis[25][25];
void dfs(int x, int y) {
    if (x < 0 || x >= h || y < 0 || y >= w) return; // 越界
    if (g[x][y] == '#' || vis[x][y]) return; // 红砖或已访问
    vis[x][y] = true;
    ++ans; // 多到达一块黑砖
    dfs(x - 1, y); // 上
    dfs(x + 1, y); // 下
    dfs(x, y - 1); // 左
    dfs(x, y + 1); // 右
}
```

- 只求从起点能到达多少格，每个格子只算一次，所以不需要撤销。
- 多组数据读到 `0 0` 结束，每组都要清空 `vis` 和 `ans`。

马走日：八个方向与步数

- 马走“日”：横一格竖两格，或横两格竖一格。
- 八个方向写成两个常量数组。
- 用 `step` 记录已经走过的格子数。
- 走遍所有格子时答案加一；走不到就回溯换方向。



本页要点

棋盘题先画方向图，再写方向数组；顺序要与题目要求一致。

马走日：核心递归

```
int n, m, sx, sy, ans;
bool vis[15][15];
int dx[8] = {-2, -1, 1, 2, 2, 1, -1, -2};
int dy[8] = {1, 2, 2, 1, -1, -2, -2, -1};
void dfs(int x, int y, int step) {
    if (step == n * m) { // 所有格子都走过了
        ++ans;
        return;
    }
    for (int k = 0; k < 8; ++k) {
        int nx = x + dx[k], ny = y + dy[k];
        if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
        if (vis[nx][ny]) continue;
        vis[nx][ny] = true;
        dfs(nx, ny, step + 1);
        vis[nx][ny] = false; // 撤销
    }
}
```

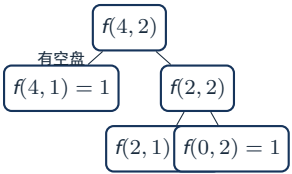
棋盘问题：逐行放，也可以跳过

```
int n, k, ans;
char g[10][10];
bool col[10];
void dfs(int r, int placed) { // 第 r 行，已放 placed 个
    if (placed == k) { // 放够 k 个棋子
        ++ans;
        return;
    }
    if (r == n) return; // 行用完了
    if (placed + (n - r) < k) return; // 剩下的行不够放
    for (int c = 0; c < n; ++c) {
        if (g[r][c] != '#' || col[c]) continue; // 不能放或同列冲突
        col[c] = true;
        dfs(r + 1, placed + 1); // 这一行放一个
        col[c] = false; // 撤销
    }
    dfs(r + 1, placed); // 这一行不放
}
```

- 每行可放一个（列不冲突）或跳过；`placed+(n-r)<k` 是可行性剪枝。

更多模型：博弈、字符串与分组

- 取石子游戏：不是棋盘搜索，而是博弈递归；$\lfloor a/b \rfloor \geq 2$ 时先手必胜，否则只有唯一走法。
- 单词接龙：先预处理任意两个单词的最小重叠长度，再 DFS 接龙，每个单词最多用两次。
- 分成互质组：放已有组或新开一组，用 gcd 检查互质。
- 放苹果：按“有没有空盘”分类，$f(m,n) = f(m,n-1) + f(m-n,n)$。



搜索题最常见的十个错误

1. 边界写成 `step == n` 还是 `step > n`，没有按自己的下标体系核对。
2. 忘记撤销标记，后面的分支被错误地堵死。
3. 组合从 1 开始枚举，输出了排列或重复方案。
4. 输出顺序不是字典序，或每个元素没有按题目要求对齐。
5. 剪枝条件太强，把还有希望的分支提前剪掉。
6. 棋盘越界、起点或终点本身是障碍，没有特判。
7. 多组数据没有清空标记数组和答案。
8. 递归深度过大导致栈溢出（例如 100×100 迷宫）。
9. 方案数超过 `int`，没有使用 `long long`。
10. 环、闭合路径忘记检查最后一条边。

复杂度速查

模型	搜索树规模	备注
全排列	$n!$	$n \le 10$ 量级
组合	$\binom{n}{r}$	与输出规模同阶
子集	2^n	$n \le 20$ 量级
八皇后	92 组解	逐行搜索很快
迷宫 / 红与黑	$O(nm)$	每个格子最多访问一次
马走日	指数级	棋盘越大增长越快
素数环	指数级	$n \le 30$，合法解多

- 先估算搜索树大小，再决定是否需要剪枝或换算法。

调试与对拍

- 小数据手算：$n = 3$ 的排列、$n = 5, r = 3$ 的组合、$n = 7$ 的拆分，先写出期望输出。
- 打印搜索树：在 `dfs` 入口输出层数和当前方案，检查顺序和重复。
- 对拍：写一个暴力程序和一个优化程序，随机小数据比较输出。
- 计时：大数据用 `clock()` 或运行时间判断是否需要剪枝。

本页要点

搜索题的 bug 大多不是“算错”，而是“漏了一种情况”或“多算了一次”。

通用模板：把五件事写全

```
void dfs(int step, int start) { // step: 已选个数
    if (step == r) { // 边界
        output(); // 记录答案
        return;
    }
    for (int v = start; v <= n; ++v) { // 枚举候选
        if (!ok(v)) continue; // 合法性
        choose(v); // 选择
        dfs(step + 1, v + 1); // 递归
        undo(v); // 撤销
    }
}
```

- 边界、候选、合法性、选择、撤销，五件事缺一不可。
- 先把模板写对，再考虑剪枝和优化。

练习安排：从网格到组合搜索

1. 先做 **LETTERS**、迷宫、红与黑，熟悉网格 DFS 和访问标记。
2. 再做 八皇后问题、八皇后、棋盘问题，练逐行放置与列/对角线标记。
3. 接着做 马走日，练八方向搜索和计数。
4. 最后做 取石子游戏、单词接龙、分成互质组、放苹果，练博弈递归、字符串重叠与分组搜索。

做题笔记

每道题写四行：状态是什么、候选是什么、边界是什么、哪里撤销。

本章练习题索引 | 1–6 / 11

1. P5212 | LETTERS（一本通 1212）
2. P5213 | 八皇后问题（一本通 1213）
3. P5214 | 八皇后（一本通 1214）
4. P5215 | 迷宫（一本通 1215）
5. P5216 | 红与黑（一本通 1216）
6. P5217 | 棋盘问题（一本通 1217）

点击蓝色题名进入学校 OJ；本章截图中的练习全部列在索引中。

本章练习题索引 | 7–11 / 11

7. P5218 | 取石子游戏（一本通 1218）
8. P5219 | 马走日（一本通 1219）
9. P5220 | 单词接龙（一本通 1220）
10. P5221 | 分成互质组（一本通 1221）
11. P5189 | 放苹果（一本通 1222）

点击蓝色题名进入学校 OJ；本章截图中的练习全部列在索引中。

练习提示（1）：网格 DFS

- **LETTERS**：从左上角出发，向四个方向走，不能走已经经过的字母；用 `vis` 标记，返回时撤销，答案为最长路径长度。
- 迷宫：问从 `S` 到 `T` 的路径条数；每走一格标记，返回时撤销，否则只能数到一条路。
- 红与黑：只问从 `@` 能到达多少块黑砖；每个格子只算一次，标记后不需要撤销。

共同点

先写方向数组 `dx/dy`，再写“越界、障碍、已访问”三个返回条件。

练习提示（2）：棋盘、博弈与分组

- 八皇后问题 / 八皇后：逐行放，列和两条对角线各用一个数组标记；注意题目要求的输出顺序和前 3 组。
- 棋盘问题：每行可以放一个棋子，也可以跳过；列不能重复。
- 马走日：八个方向，步数等于格子总数时计数；走不通就撤销。
- 取石子游戏：$\lfloor a/b \rfloor \geq 2$ 时先手必胜，否则递归到唯一走法。
- 单词接龙：先算两词最小重叠，再 DFS；每个词最多用两次。
- 分成互质组：放已有组或新开组，组数达到当前最优就剪枝。
- 放苹果：按“有空盘 / 没空盘”分类，得到 $f(m, n) = f(m, n-1) + f(m-n, n)$。