回溯法的核心思想

按选优条件向前搜索，以达到目标。如果探索到某一步时发现原先选择并不优或达不到目标，就退回到上一步，重新选择。

1. 问题的解空间
2. DFS
3. 递归回溯
4. 迭代回溯
5. 子集树与排列树

问题 5-2

5-2 最小长度电路板排列问题。

问题描述：最小长度电路板排列问题是大规模电子系统设计中提出的实际问题。该问题的提法是，将 n 块电路板以最佳排列方案插入带有 n 个插槽的机箱中。n 块电路板的不同的排列方式对应于不同的电路板插入方案。

设 $B=\{1, 2, \cdots, n\}$ 是 n 块电路板的集合。集合 $L=\{N_1, N_2, \cdots, N_m\}$ 是 n 块电路板的 m 个连接块。其中每个连接块 N_i 是 B 的一个子集，且 N_i 中的电路板用同一根导线连接在一起。在最小长度电路板排列问题中，连接块的长度是指该连接块中第 1 块电路板到最后 1 块电路板之间的距离。

试设计一个回溯法，找出所给 n 个电路板的最佳排列，使得 m 个连接块中最大长度达到最小。

算法设计：对于给定的电路板连接块，设计一个算法，找出所给 n 个电路板的最佳排列，使得 m 个连接块中最大长度达到最小。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 m（$1 \leqslant m, n \leqslant 20$）。接下来的 n 行中，每行有 m 个数。第 k 行的第 j 个数为 0 表示电路板 k 不在连接块 j 中，为 1 表示电路板 k 在连接块 j 中。

结果输出：将计算的电路板排列最小长度及其最佳排列输出到文件 output.txt。文件的第一行是最小长度；接下来的 1 行是最佳排列。

思路：回溯法，尝试每种排列

1. 排列状态：通过数组arrangement表示当前排列，used数组标记某块电路板是否已经被使用；

2. 长度计算：函数len根据排列arrangement，计算每个连接块的长度，返回所有连接块长度的最大值；
3. 递归搜索：每层递归尝试将未使用的电路板放置在当前位置；深度达到 nnn 时计算最大长度，并更新全局最优解；

伪代码：

核心模块

```
Function LEN(arrangement)
    max_len ← 0
    for each connection_block j do
        find min_pos and max_pos in arrangement where block j has connection
        if valid:
            update max_len with max(max_len, max_pos - min_pos)
    end for
    return max_len
End Function
Function BACKTRACK(depth)
  if depth = n then
        current_length ← LEN(arrangement)
        if current_length < best_length then
            best_length ← current_length
            best_arrangement ← arrangement
        end if
        return
    end if
```

```
    for i from 0 to n-1 do
        if not used[i] then
            used[i] ← true
            arrangement[depth] ← i
            BACKTRACK(depth + 1)
            used[i] ← false
        end if
    end for
```

End Function

时间复杂度：O(n!×nm)

回溯法需要生成所有电路板的排列，共有n!种可能；

len对每种排列需检查m个连接块。对每个连接块，需要扫描n个电路板的排列，复杂度为O(nm);

空间复杂度：O(nm+n)

存储connections矩阵需要O(n×m)；

辅助变量arrangement, used, best_arrangement各需要O(n);

问题 5-20

5-20　部落卫队问题。

问题描述：原始部落 byteland 中的居民们为了争夺有限的资源，经常发生冲突。几乎每个居民都有他的仇敌。部落酋长为了组织一支保卫部落的队伍，希望从部落的居民中选出最多的居民入伍，并保证队伍中任何 2 个人都不是仇敌。

算法设计：给定 byteland 部落中居民间的仇敌关系，计算组成部落卫队的最佳方案。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 m，表示 byteland 部落中有 n 个居民，居民间有 m 个仇敌关系。居民编号为 $1, 2, \cdots, n$。接下来的 m 行中，每行有 2 个正整数 u 和 v，表示居民 u 与居民 v 是仇敌。

结果输出：将计算的部落卫队的最佳组建方案输出到文件 output.txt。文件的第 1 行是部落卫队的人数；第 2 行是卫队组成 x_i（$1 \leqslant i \leqslant n$）。$x_i$=0 表示居民 i 不在卫队中，x_i=1 表示居民 i 在卫队中。

思路：回溯法，尝试每名居民是否入伍

1. 递归状态：当前正在考虑的居民编号resident
2. 决策点：每个居民可以选择入伍或不入伍；
3. 可行性检查：当前居民是否与已选择入伍的居民存在仇敌关系；
4. 剪枝：如果当前已选择人数加剩余人数不可能超过当前最佳解，可以提前终止递归；
5. 更新解：当所有居民都处理完时，更新最佳解（最多入伍人数和对应的方案）

伪代码：

核心模块

```
Function BACKTRACK(resident)
    if resident > n then
        if cnt > best_cnt then
            best_cnt ← cnt
            best_solution ← solution
        end if
```

```
        return
    end if
    if cnt + (n - resident + 1) ≤ best_cnt then
        return
    end if
    if CAN_JOIN(resident) then
        solution[resident] ← 1
        cnt ← cnt + 1
        BACKTRACK(resident + 1)
        cnt ← cnt - 1
        solution[resident] ← 0
    end if
    solution[resident] ← 0
    BACKTRACK(resident + 1)
End Function
```

时间复杂度：$O(n \cdot 2^n)$ （最坏）

递归树深度：每次递归处理一个居民，递归树深度为n；

状态数：每个居民有两种选择（入伍或不入伍）共2^n种可能；

冲突检测：每次判断是否有仇敌关系需扫描所有居民，复杂度为O(n)；

空间复杂度：$O(n^2)$

存储矩阵：存储enemies矩阵需要$O(n^2)$；

递归栈：递归深度为O(n)；