算法设计与分析——作业四

2253206 韩明洋

# 概念梳理题

1. 子问题
2. 自顶向下；递归；自底向上；迭代；子问题的解；查询；重复计算
3. 1最优子结构；2重叠子问题；3无后效性
4. 状态转移方程；子问题的最优解
5. 边界条件
6. 选择；全局最优解
7. 局部最优选择
8. 子问题；最优子结构性质
9. 不一定

# 解答题

## 1. 台阶问题

将问题分解为子问题。设 **dp[i]** 表示走到第 **i** 级台阶的方法数。由于最后一步是 1 级或 2 级，**dp[i]** 依赖于 **dp[i-1]** 和 **dp[i-2]**，子问题重叠且具有最优子结构。

**动态转移方程：**

dp[i]=dp[i−1]+dp[i−2]*dp*[*i*]=*dp*[*i*−1]+*dp*[*i*−2]

**边界条件：**

**dp[0] = 1**（没有台阶时，只有一种方法：不动）

**dp[1] = 1**（只有 1 级台阶时，只能走 1 步）

**逐步计算：**

| **i*i*** | **计算过程** | **dp[i]*dp*[*i*]** |
| --- | --- | --- |
| 0 | 边界条件 | 1 |
| 1 | 边界条件 | 1 |
| 2 | dp[1]+dp[0]=1+1 | 2 |
| 3 | dp[2]+dp[1]=2+1 | 3 |
| 4 | dp[3]+dp[2]=3+2 | 5 |
| 5 | dp[4]+dp[3]=5+3 | 8 |
| 6 | dp[5]+dp[4]=8+5 | 13 |
| 7 | dp[6]+dp[5]=13+8 | 21 |
| 8 | dp[7]+dp[6]=21+13 | **34** |

**所以** 34 种方法。

## 2. Dijkstra 算法问题

（1）**基本思想：** 迪杰斯特拉算法是一种用于在加权图中找到从单一源点到所有其他顶点的最短路径的算法。其核心思想是贪心策略，即在每一步选择当前已知最短路径的顶点，然后更新与该顶点相邻的顶点的最短路径估计。

**基本流程：**

1. 初始化：将所有顶点标记为未访问，设置源点到自身的距离为0，到其他所有顶点的距离为无穷大。
2. 选择未访问顶点中距离最小的顶点作为当前顶点。
3. 更新当前顶点的邻接顶点的距离：如果通过当前顶点到达某个邻接顶点的距离小于已知的距离，则更新该距离。
4. 标记当前顶点为已访问。
5. 重复步骤2-4，直到所有顶点都被访问。

（2）解答

1. **初始化：**
   * 距离：A=0, B=∞, C=∞, D=∞, E=∞
   * 已访问：A（已访问），B（未访问），C（未访问），D（未访问），E（未访问）
2. **从A开始：**
   * 更新B的距离：A->B = 3
   * 更新C的距离：A->C = 2
   * 更新D的距离：A->D = 9
   * 距离：A=0, B=3, C=2, D=9, E=∞
   * 已访问：A（已访问），B（未访问），C（未访问），D（未访问），E（未访问）
3. **选择C（距离最小）：**
   * 更新D的距离：C->D = 6（2+6=8，比9小）
   * 距离：A=0, B=3, C=2, D=8, E=∞
   * 已访问：A（已访问），B（未访问），C（已访问），D（未访问），E（未访问）
4. **选择B（距离最小）：**
   * 更新D的距离：B->D = 4（3+4=7，比8小）
   * 更新E的距离：B->E = 5（3+5=8）
   * 距离：A=0, B=3, C=2, D=7, E=8
   * 已访问：A（已访问），B（已访问），C（已访问），D（未访问），E（未访问）
5. **选择D（距离最小）：**
   * 更新E的距离：D->E = 1（7+1=8，与B->E相同，不更新）
   * 距离：A=0, B=3, C=2, D=7, E=8
   * 已访问：A（已访问），B（已访问），C（已访问），D（已访问），E（未访问）
6. **选择E（距离最小）：**
   * 距离：A=0, B=3, C=2, D=7, E=8
   * 已访问：A（已访问），B（已访问），C（已访问），D（已访问E），（已访问）

**最短路径：** 从A到E的最短路径是 A -> B -> E，总距离为8。

## 3. 现金找零问题

**(1) 模拟贪心算法找零的过程**

给定集合 *M*={1,5,10,20,50,100}，要找零67元。

**步骤：**

1. 从最大面额开始，尽可能多地使用该面额。
2. 剩余金额继续使用下一个最大面额，直到找零完成。

**过程：**

* 使用50元：剩余 67−50=17 元
* 使用10元：剩余 17−10=7 元
* 使用5元：剩余 7−5=2 元
* 使用1元：剩余 2−1=1 元
* 使用1元：剩余 1−1=0 元

**结果：** 需要6张纸币（50元1张，10元1张，5元1张，1元3张）。

**(2) 集合***M*={1,2,4,8,16}**的贪心选择性质**

**是否满足：** 是的，这个集合满足贪心选择性质。

**原因：** 这个集合是2的幂次方集合，任何数都可以表示为这些数的和（即二进制表示），每次选择最大的数可以保证使用的纸币数量最少。

**(3) 更一般化的猜想**

**猜想：** 如果一个集合 *M* 是某个数 *b* 的幂次方集合（即 *M*={*b*0,*b*1,*b*2,…,*bk*}），那么这个集合满足贪心选择性质。

**(4) 不满足贪心选择性质的集合***M*

**例子：** *M*={3,5,7}

**原因：** 例如，要找零12元：

* 贪心选择：5 + 5 + 2（需要3张）
* 最优选择：3 + 3 + 3 + 3（需要4张）

这个例子中，贪心选择并不总是能得到最少的纸币数量。

**(5) 动态规划的找零算法**

**算法描述：**

1. 创建一个数组 dp，其中 dp[i] 表示找零 *i* 元所需的最少纸币数量。
2. 初始化 dp[0] = 0（找零0元需要0张纸币），其他 dp[i] = \infty（初始化为一个很大的数）。
3. 对于每个金额 *i* 从1到 *n*：
   * 对于每个面额 *m* 在集合 *M* 中：
     + 如果 *m*≤*i*，则更新 dp[i] = min(dp[i], dp[i - m] + 1)。
4. 最后，dp[n] 就是找零 *n* 元所需的最少纸币数量。

**时间复杂度：** *O*(*nk*)，其中 *n* 是目标金额，*k* 是集合 *M* 中元素的数量。

**(6) 贪心算法和动态规划算法的区别**

* **贪心算法：** 每一步都做出局部最优选择，希望这样能导致全局最优解。它简单、快速，但不总是能得到最优解。
* **动态规划：** 通过考虑所有可能的子问题和它们的答案，确保找到全局最优解。它更复杂、可能更慢，但总是能得到最优解。