

Ch2. Recursion. 递归.

math foundation: 数学归纳法.

1. 一阶递归模型.

$$\begin{cases} f(s_1) = m \\ f(s_n) = g(f(s_{n-1}), c_{n-1}) \end{cases}$$

递归时“递变”
出口时“不变”

2. 递归算法设计

- 基于递归数据结构.

• 递归 $\left\{ \begin{array}{l} \cdot \text{缩小规模} \\ \cdot \text{找到出口} \\ \cdot \text{小问题可解} \end{array} \right.$

3. Example.

- n Queen. \Rightarrow 遍历所有解. 无剪枝.

ch3. Divide & Conquer 分治

递归子集

问题特征

① 小规模问题可解 ② 问题可分解 分

③ 分解的子问题解可合并为大问题的解. \Rightarrow 治

④ 子问题求解相互独立

* 不可同时则次
动态规划 / 贪心法

框架

divide-and-conquer(P)

{ if $|P| \leq n_0$ return adhoc(P);

将 P 分解为较小的子问题 P_1, P_2, \dots, P_k ;

for($i=1; i \leq k; i++$) // 循环处理 k 次

$y_i = \text{divide-and-conquer}(P_i)$; // 递归解决 P_i

 // return merge(y_1, y_2, \dots, y_k); // 合并子问题

} // 缩进有些问题.

Example. 排序

① 快速排序

分 插入 pivot 划分为 2 个子序列

治：数组内排序即为治

$$O(n \log n) = O(n \cdot \log n)$$

划分耗时

递归树平均深度

② 归并排序

分 大序列分解成 length 长的子序列，对子序列分别排序

治：数组内排序即为治

$$O(n \log n) = O(n \cdot \log n)$$

子序列比较、Merge 耗时

递归树平均深度

* 特殊情况处理： $\left[\frac{\text{len}(\text{list})}{\text{length}} \right]$ {奇数
偶，但最后一组不足 length}

Example 查找

① 无序列表 找最大、次大元素.

分 list 二分，分别求 $[left, mid], [mid, right]$ 的 max、次 max

合 左右的 max、次 max 比较.

$$O(n) \leq \begin{cases} T(1) = T(2) = 1 \\ T(n) = 2T(n/2) + 1 \quad \text{合并时间为 } O(1). \end{cases}$$

trick. $\text{len}(list) = 1$ 时, 令次 max = $-\infty$.

② 折半查找

③ 无序列表 找第 k 大的元素.

类快排思想. 使用 pivot

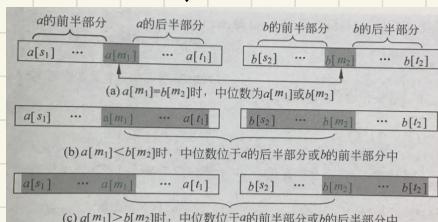
\therefore 为 list. 确定后其 $(下标+1)$ 即为
大于在 list 中序数.

\therefore pivot index = k. return pivot.

$< k \quad \text{search}[pivot.index:]$

$> k \quad \text{search}[:pivot.index]$

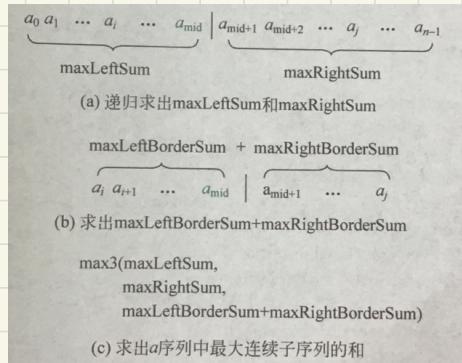
④ 两等长有序数列 中位数



Example · 组合问题

① 最大连续子序列

考虑 子列和可能出现的地方



② L型骨牌相叠覆盖

棋盘 $2^k \times 2^k$ 任意位置放 L 特殊方格。
求 1 种 L 型牌覆盖方法

解：1. 无论如何棋盘能放满。
不受特殊方格位置制约。

2. 分治

程序每轮
仅放置中
心一个 L 型
骨牌

盖其他 3 个象限中各一个方格的 L 形骨牌。图 3.11(b)~图 3.11(d) 是特殊方格在其他象限中放置 L 形骨牌的情况。

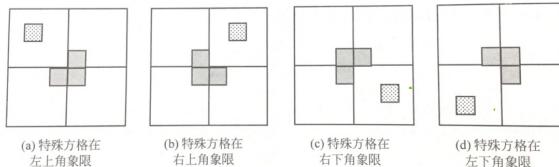


图 3.11 放置一个 L 形骨牌

这样每个象限和包含特殊方格的象限类似，都需要少覆盖一个方格，也与整个问题类似，所以采用分治法求解，将原问题分解为 4 个子问题。

③ 循环日程安排

优雅的数据结构设计 + 巧妙找规律

【问题描述】 设有 $n=2^k$ 个选手要进行网球循环赛, 设计一个满足以下要求的比赛日程表:

- (1) 每个选手必须与其他 $n-1$ 个选手各赛一次。
- (2) 每个选手一天只能赛一次。
- (3) 循环赛在 $n-1$ 天之内结束。

【问题求解】 按问题要求可将比赛日程表设计成一个 n 行 $n-1$ 列的二维表, 其中第 i 行、第 j 列表示和第 i 个选手在第 j 天比赛的选手。

假设 n 位选手被顺序编号为 $1, 2, \dots, n(2^k)$ 。当 $k=1, 2, 3$ 时比赛日程表如图 3.13 所示, 其中第 1 列是增加的, 取值为 $1 \sim n$ 对应各位选手, 这样比赛日程表变成一个 n 行 n 列二维表。



视频讲解

		$i \rightarrow$	1	2	3	4	5	6	7	8
			2	1	4	3	6	5	8	7
		$j \rightarrow$	3	4	1	2	7	8	5	6
			4	3	2	1	8	7	6	5
		$j = i + 2^k$	5	6	7	8	1	2	3	4
			6	5	8	7	2	1	4	3
			7	8	5	6	3	4	1	2
			8	7	6	5	4	3	2	1
(a) $k=1$	(b) $k=2$	(c) $k=3$								

图 3.13 $k=1 \sim 3$ 的比赛日程表

从中可以看出规律, $k=1$ 只有两个选手时比赛安排十分简单, 而 $k=2$ 时可以基于 $k=1$ 的结果进行安排, $k=3$ 时可以基于 $k=2$ 的结果进行安排。

Ch4. Brute Force. 級力法.

Ch5. Backtracking. 回溯法.

用题角度

解空间树 D. 在可能状态中生长.

叶节点为解. (未被剪枝的)

问题类型 < 全部解.
最优解.

设计算法 {
① 树如何生长. (子决策如何确定).
② 树如何搜索.
③ 有效求解? (剪枝) }
约束
限界

tree. <
子集树
排列树

如何回溯? {
① 追溯
② 截至保存中间解来迭代

框架

· 非递归

```
int x[n]; //x存放解向量,全局变量
void backtrack(int n) //非递归框架
{
    int i=1; //根结点层次为1
    while (i>=1) //尚未回溯到头
    {
        if(ExistSubNode(t)) //当前结点存在子结点
        {
            for (j=下界;j<=上界;j++) //对于子集树,j从0到1循环
            {
                x[i]取一个可能的值; //x[i]满足约束条件或界限函数
                if (constraint(i) && bound(i))
                {
                    if (x是一个可行解)
                        输出 x;
                    else i++;
                }
            }
        }
        else i--;
    }
}
```

//不存在子结点,返回上一层,即回溯

· 递归(子集树)

```
int x[n]; //x存放解向量,为全局变量
void backtrack(int i) //求解子集树的递归框架
{
    if(i>n) //搜索到叶子结点,输出一个可行解
        输出结果;
    else
    {
        for(j=下界;j<=上界;j++) //用j枚举i所有可能的路径
        {
            x[i]=j; //产生一个可能的解分量
            ...
            if (constraint(i) && bound(i)) //满足约束条件和限界函数,继续下一层
                backtrack(i+1); //其他操作
        }
    }
}
```

↓ DFS.

“回退时系统递归栈自动还原x[i].”

采用上述算法框架需要注意以下几点：

- (1) i 从 1 开始调用上述回溯算法框架, 此时根结点为第 1 层, 叶子结点为第 $n+1$ 层。当然 i 也可以从 0 开始, 这样根结点为第 0 层, 叶子结点为第 n 层, 所以需要将上述代码中的“ $\text{if } (i>n)$ ”改为“ $\text{if } (i>=n)$ ”。
- (2) 在上述框架中通过 for 循环用 j 枚举 i 的所有可能路径, 如果扩展路径只有两条, 可以改为两次递归调用(如求解 0/1 背包问题、子集和问题等都是如此)。
- (3) 这里回溯框架只有 i 一个参数, 在实际应用中可以根据具体情况设置多个参数。

Example.

① 1~9 间插入 '+' '-' 式空格，输出求和为 100 的表达式

子集树。→

② 水盆排列 (排列树) ⇌ 元素交换 非填无

③ 0/1 背包 子集树 & 最优解

剪枝函数设计：左右子树分别设计
起至？ 过程？

思路 实现
← 相近

④ 简单装载 子集树 & 最优解

⑤ 复杂装载 → 转化

⑥ 子集和

⑦ N 皇后 无向图着色 ⇒ 子集树

⑧ 任务分配 几个任务 ⇒ N 叉子集树

⑨ 活动安排 排列树：暴力遍历、剪枝

⑩ 流水作业 排列树 + 剪枝

Ch6. Branch & Bound 分枝限界法

——回溯法求全部解.

分枝限界求 一个/最优解.

广度优先搜索

设计. ① 限界函数.

② 治结点扩展

队列

优先队列 <= 跳跃式搜索, 不再按层

③ 解向量存储

Example

① 0/1 背包.

PL 判决
剪枝 {
 左子树 放入后不超重.
 右子树 设置 bound() 为上界 (剩余可能最大价值)
 上界 \leq 当前最优解. \Rightarrow 剪枝.

优先队列 (大根堆)

以上界 ub 为排序标准.

trick: 队中 struct node 的设计 — 每个 node 都存储当前解向量

② 困单源最短路径.

用 edge.weight 剪枝.

设 prev[] 数组 保存节点 路径

相较于 Dijkstra. 队列自动完成了 已遍历节点的记录

③ 任务分配. 下界 lb 为剩余未分配任务最短完成时间 + 已耗时间

Ch7. Greedy algorithms / 贪心法

- 满足 2 性质。
 - 贪心选择：整体最优 可通过局部最优达到
 - 最优子结构：最优解包含子问题最优解。

思路：建模 → 子问题 → 求子问题局部最优解
→ 合并得全局最优解。

Example.

最大非空子集 $\left\{ \begin{array}{l} \text{① 活动安排问题} \\ \text{② 互斥保留问题} \\ \text{③ 相交区间问题} \end{array} \right.$ *注意区间开闭性

step1. sort. step2. greedy - 取最先结束的

背包问题

一维0/1包. 可拿一部分. greedy: 按 $\frac{\text{Value}}{\text{weight}}$ 排序

more: Dijkstra. Prim. Huffman Tree.

Ch8. Dynamic Programming 动态规划

多阶段决策问题优化

[递推公式 + 初态 + DP存储数据结构] \Rightarrow DP

· 问题特征

“问题由系列子问题构成，而前子问题解基于上一子问题”

① 最优子结构

② 无后效性。——状态确定后不受后续状态影响

③ 子问题重叠。——状态解可复用，效率高

· 设计思路

① 划分阶段：阶段应可排序

② 定状态和状态变量。

③ 求状态转移方程 \leftarrow most important step

④ 定边界条件

① 明确数组含义

\Rightarrow ② 求 dp 数组。
递推公式

③ 确定边界条件

求解时：正序 / 逆序

Example.

1. 整数拆分

设 $f(n, k)$ 为将整数 n 无序拆分为拆分项值 最多不超过 k 的方案个数
讨论如下:

$$\text{① } (n=1) / (k=1), \quad f(n, k) = 1.$$

$$\text{② } n < k, \quad f(n, k) = f(n, n)$$

$$\text{③ } n = k, \quad f(n, k) = f(n, n-1) + 1. \quad \begin{array}{l} \text{④ } n \text{ 拆分为 } 1 \text{ 个 } n. \\ \text{⑤ } n \text{ 的 } n-1 \text{ 拆分} \end{array}$$

$$\text{④ } n > k. \quad \begin{array}{l} \text{a. 拆分为 } n = k + (n-k). \quad \text{其中 } (n-k) \text{ 能继续被拆分. 由 } f(n-k, k) \\ \text{b. 拆分项都小于 } k. \quad \text{即 } n = \sum_{i=1}^r x_i \quad x_i < k \quad (1 \leq i \leq r) \\ \therefore \text{退化为 } f(n, k-1). \end{array}$$

综上.

$$\text{有 } f(n, k) = \begin{cases} 1 & (n=1) / (k=1) \\ f(n, n), & n < k \\ f(n, n-1) + 1, & n = k \\ f(n-k, k) + f(n, k-1), & n > k \end{cases}$$

代码实现.

设 dp 二维数组 存放状态.

2. 最大连续子列和

$f(a_i)$ 为第 i 个数读入后，最大连续子列和

分类有：① $a_i < 0$. $f(a_i) = \max\{0, f(a_{i-1}) + a_i\}$

#不选，或选了后与前子列和仍 > 0 . 则后继续可能使子列和继续增长。

② $a_i \geq 0$. $f(a_i) = f(a_{i-1}) + a_i$

整理有：

$$dp[0] = 0. \quad \text{#边界}$$

$$dp[j] = \max\{dp[j-1] + a_j, a_j\}. \quad 1 \leq j \leq n.$$

∴ 最大子列和为 $dp[\max j]$.

子序列从 $\max j$ 的第一个值 ≤ 0 的元素 a_i .

即序列为 $[i, \max j]$.

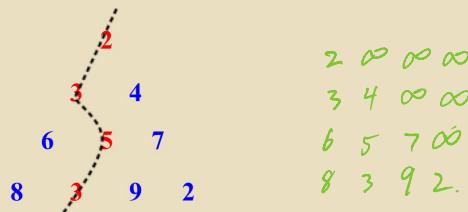
3. 字符串中最长连续数字串。

$$dp[j] = \begin{cases} 0. & a_j \text{ 非数字} \\ dp[j-1] + 1 & a_j \text{ 是数字} \end{cases}$$

4. 来解三角形最小路径.

【问题描述】 给定高度为 n 的一个整数三角形，找出从顶部到底部的最小路径和，只能向左或向右移动相邻的结点。首先输入 n ，接下来的 $1 \sim n$ 行，第 i 行输入 i 个整数，输出分为 2 行，第一行为最小路径，第二行为最小路径和。

例如，下图是一个 $n=4$ 的三角形，输出的路径是 2 3 5 3，最小路径和是 13。



$f(r, c)$. r : row. 行号. c : column. 列号
 $f(r, c)$ 为行至 (r, c) 时的最短路径。

大三角形转为直角三角形
便于矩阵存储表示。

分析：（设起始行 3， index 为 1， $a(r, c)$ 为对应点路径值）

- ① $f(1, 1) = a(1, 1)$. } 初界
- ② $f(r, 1) = a(r, 1) + f(r-1, 1)$. } 边界
- ③ $f(r, c) = a(r, c) + f(r-1, c-1)$. # $r=c$.
- ④ $f(r, c) = a(r, c) + \min(f(r-1, c), f(r-1, c-1))$

∴ 对于 dp 有

$$dp[0][0] = a[0][0] \quad \text{顶部边界}$$

$$dp[i][0] = dp[i-1][0] + a[i][0] \quad \text{考虑第1列的边界, } 1 \leq i \leq n-1$$

$$dp[i][i] = dp[i-1][i-1] + a[i][i] \quad \text{考虑对角线的边界, } 1 \leq i \leq n-1$$

$$dp[i][j] = \min(dp[i-1][j-1], dp[i-1][j]) + a[i][j] \quad i > 1 \text{ 的其他两条达到路径的结点}$$

* $dp[i][j]$ 存储着 a, b 中从头开始的长度分别为
为 i, j 的子序列的 LCS 长度

∴ 解 LCS 存在 $dp[m][n]$ 中

Q: 如何由 dp 求出序列 sub_3 ?

答: sub_3 未解. 由 $dp[m][n]$ 开始回退查找

(1) $dp[i][j] == dp[i-1][j] \Rightarrow i--$
“一样则说明 $a[i-1]$ 不在 sub_3 中”

(2) $dp[i][j] == dp[i][j-1] \Rightarrow j--$
“同理”

(3). $dp[i][j]$ 与上方、左方所有值都不等 $\Rightarrow i--, j--$
且 $a[i-1]$ 在 sub_3 中
(此时 $a[i-1] = b[j-1]$)

6. 最长递增子序列

【问题描述】 给定一个无序的整数序列 $a[0..n-1]$, 求其中最长递增子序列的长度。
例如, $a[] = \{2, 1, 5, 3, 6, 4, 8, 9, 7\}$, $n=9$, 其最长递增子序列为 $\{1, 3, 4, 8, 9\}$, 结果为5。

$dp[i]$ 表示 $a[0..i]$ 中以 $a[i]$ 结尾的最长递增子序列的长度

$$\begin{cases} dp[i] = 1 & 0 \leq i \leq n-1 \text{ (初始化)} \\ dp[i] = \max(dp[i], dp[j]+1) & \text{if } a[i] > a[j] \quad 0 \leq j \leq i-1 \end{cases}$$

7. 编辑距离

【问题描述】 设 A 和 B 是两个字符串。现在要用最少的字符操作次数，将字符串 A 转换为字符串 B 。这里所说的字符操作共有3种：

- (1) 删除一个字符。
- (2) 插入一个字符。
- (3) 将一个字符替换另一个字符。

例如, $A = "sfdqxbw"$, $B = "gfdgw"$, 结果为4。

$dp[i][j]$ 存储 $a[0..i-1]$ 与 $b[0..j-1]$ 的最优编辑距离。

初始化 $\begin{cases} (1) i=0, dp_a \text{ 为空}, dp[0][j] = j & "A 中插入 } B \text{ 的 } j \text{ 个字符" } \\ (2) j=0, dp_b \text{ 为空}, dp[i][0] = i & "删除 } A \text{ 中的 } i \text{ 个字符" } \end{cases}$

非空 $(3) dp[i][j] = dp[i-1][j-1]$ 当 $a[i-1] = b[j-1]$ 时, 无须任何操作

$(4) dp[i][j] = \min(dp[i-1][j-1]+1, dp[i][j-1]+1, dp[i-1][j]+1)$ $\begin{cases} a[i-1] \neq b[j-1] \\ a[i-1] 替换为 b[j-1] \\ a[i-1] 后插入 b[j-1] \\ 删掉 a[i-1] \end{cases}$

8. 8/1 背包

$$\text{total_value} = \sum_{i=1}^n v_i x_i. \quad (x_i = 0 \text{ 或 } 1)$$

主序： n 个物品，按 $i=1, 2, \dots, n$ 次序确定 x_i 的值

长分子问题： $x_i = 0$. \rightarrow 背包容量 W . 剩余物品装载

$x_i = 1$ \rightarrow 背包容量 $W - w_i$. 剩余物品装载

$\therefore dp[i][r]$ 表示背包 当前最大 容量为 r ($0 \leq r \leq W$)

已考虑 $1, 2, \dots, i$ ($1 \leq i \leq n$) 时装入的最优价值

$$\left\{ \begin{array}{l} dp[i][0] = dp[0][r] = 0. \quad \text{边界条件. 无容量/物品 可装.} \\ dp[i][r] = dp[i-1][r] \quad r < w[i]. \text{ 时, 无法装入 } i \text{ 物品} \\ dp[i][r] = \max [dp[i-1][r], dp[i-1][r-w[i]] + v[i]] \end{array} \right.$$

细节：放入物品后，上一个相关
状态的剩余容量
对应的是 $r - w[i]$

Weakness：由于数组第二维记录背包剩余容量

： 数组可能过大。

且 不支持浮点数

(index 必为整数)

9. 完全背包

每种物品可选任意多件.

$$dp[0][j] = dp[i][0] = 0 \quad \text{#边界条件}$$

$$dp[i][j] = \max \{ dp[i-1][j - k * w[i]] + k * v[i] \}$$

($\Rightarrow dp[i][j] < dp[i-1][j - k * w[i]] + k * v[i]$ 有价值增长)

且 $k * w[i] \leq j$ 时) (不装)

$k=0$: 不装. $k>1$: 装入 k 件.

and. 用 $f[k][i][j]$ 存 $dp[i][j]$ 取 max 后物品 i 装入 k 个数.

more tricks:

使用“滚动数组”压缩空间

(\because 当前状态决策仅依赖过去有限个状态.)

(\therefore 无需保存全部状态.)

Ch9 Graph. 图算法

1. minimal spanning tree

- a. Prim. algorithm
 - b. Kruskal
- } greedy alg.

2. shortest path.

- a. Dijkstra.
- b. Floyd.

多源 - 矩阵规划

$$\left\{ \begin{array}{l} A_{-1}[i][j] = g.edges[i][j] \\ A_k[i][j] = \min \{ A_{k-1}[i][j], A_{k-1}[i][k] + A_{n-1}[k][j] \} \\ \text{选优最近} \\ 0 \leq k \leq n-1 \end{array} \right.$$