总结

1. 绪论

- **算法的特性**
  - 输入：一个算法有**零个或多个输入**，确定所处理问题的**输入定义域**。
  - 输出：一个算法有**一个或多个输出**（没有输出的算法没有任何意义）。
  - 确定性：每一个步骤不能有**歧义**。
  - 有穷性：一个算法必须总是在执行**有穷步后结束**，且每一步都必须在**有穷时间内完成**。
  - 可行性：所有操作都可以通过已经实现的**基本操作运算**执行**有限次**来实现。
- 具体问题的算法
  - 同一算法可以用多种不同形式藐视，**描述方式不唯一**。
  - 同一问题存在多种不同求解算法，同一问题算法**多样性**。
  - 同一问题多种算法基于不同思路，解题速度显著不同，算法有**多样的效率**。
- 算法重要问题类型
  - 排序
  - 查找
  - 字符串匹配
  - 图问题
  - 组合问题
  - 几何问题
  - 数值问题
- 数据结构
  - 线性结构
  - 树结构
  - 图结构
  - 集合
- 求解框架（步骤）
  - **分析**，建立数学模型
  - 对相关已知知识进行梳理，**分解问题**
  - **设计算法**，建立初步解
  - 对设计的算法进行**正确性证明**
  - 对正确的算法进行**效率分析**
  - 对分析后的算法进行**程序实现**
  - 相应**文档的完善**

- **算法的表示**
  - 自然语言描述法
  - 流程图描述法
  - 伪代码描述法

一个算法就是一个“**有穷规则的集合**”，规定了解决某一特定类型问题的“**步骤**”。

2. 算法效率分析

- 下界渐进符号 -
  ，代表　是　的下界（　，坐标系中　在　下面）。
- 上界渐进符号 -
  ，代表　是　的上界（　，坐标系中　在　上面）。
- 双界渐进符号 -

非递归算法分析方法：

1. **决定**用哪些参数作为**输入规模**。
   > e.g.1. 算法输入规模的度量是输入变量 ，因为通过 控制循环次数。

2. **确定**算法的**基本操作**（一般是算法最内层循环中最耗时的操作）。
   > e.g.1. 算法的基本操作是最内层循环的加法操作。
   > e.g.2. 最内层循环中包含元素的加、乘、赋值等操作，选择复杂度最高的乘作为基本操作。

3. 判断基本操作**是否只与输入规模有关**：是则直接求解；否则考察在另一相关变量不同情况下的最优、最差、平均效率（但实际上只用分析最差即可）。
   > e.g.1. 算法循环次数只接受 控制，所以 确定，循环体的执行次数就能确定。
   > e.g.2. 基本操作的执行次数不仅受 的控制，还与具体的输入有关，但对于时间复杂度可以只用分析其最差效率。

4. 建立“算法基本操作执行次数” 的**求和表达式**。
   > e.g.1.
   > e.g.2.
   > e.g.3. 最差效率：

5. 求解表达式，最终**确定效率类型**。
   > e.g.1.
   > e.g.2.
   > e.g.3. ——

递归算法分析方法：

采用主定理，一盘用来分析“分治”这类把问题划分为子问题-的复杂度。

> 主定理（并不一定严谨，只适合应付考试）：
>
> –
>
> - 是 上界，则： 。
>   若 ，则判断为： ，转换为：
> - 与 同界，则：
>   若 ，则判断为： ，转换为：
> - 是 下界，则： 。
>   若 ，则判断为： ，转换为：

1. **决定**用哪些参数作为**输入规模**。
2. **确定**算法的**基本操作**（一般是递归调用附近最耗时操作）。
3. 判断基本操作**是否只与输入规模有关**：是则直接求解；否则分析最差即可。
4. 建立“算法基本操作执行次数” 的**递推关系式及其初始条件**。
   > e.g.1. – （经典二分查找）
   > e.g.2. –
   > e.g.3. –
5. **求解递推关系式**或确定他的增长次数。（利用主定理）
   > e.g.1. ⟶ ⟶
   > e.g.2. ⟶ ⟶
   > e.g.3. ⟶ ⟶

3. 蛮力法

4. 分治法

- 具有特征
  - **规模缩小**到一定程度就**可以解决**。
  - 问题**可以分解**为若干个规模较小的相同问题。
  - 分解出的**子问题的解可以合并**为该问题的解。

    - ☆分解出的各个**子问题是相互独立**的，子问题之间不包含公共子问题。
      （**与动态规划的区别**）
- 基本步骤
    - **分解**：分解为若干个规模较小，相互独立，与原问题形式相同的子问题。
    - **解决**：若子问题规模较小而容易被解决则直接求解，否则递归解决（再划分）。
    - **合并**：各个子问题的解合并为原问题的解。
- **用到分治法的问题**
    - 合并排序
    - 快速排序
      快速排序的性能取决于：**划分的对称性**
    - 二分查找
    - 大整数乘法
    - **Strassen矩阵乘法**
    - 最近点对问题
    - 安排循环赛日程表
    - *快速幂、矩阵快速幂*

分治法设计出的程序，一般都是“**递归算法**”。

5. 动态规划

- 只适用于解决**最优化问题**。
- 执行方式 - “**自底向上**”，递推所有子问题或但阶段的最优值。
  ⚠但个人询问老师后：老师说的是自顶向下也行【？……
  注意一下这道题：

10. 下列算法中通常以自顶向下的方式求解最优解的是（ B ）。

A、分治法　　B、动态规划法　　C、贪心法　　D、回溯法

【虽然书上写的自底向上，网上搜答案都是 orz……

动态规划也是从小规模的计算逐渐扩大到大规模计算，因此是自顶向下的计算过程。

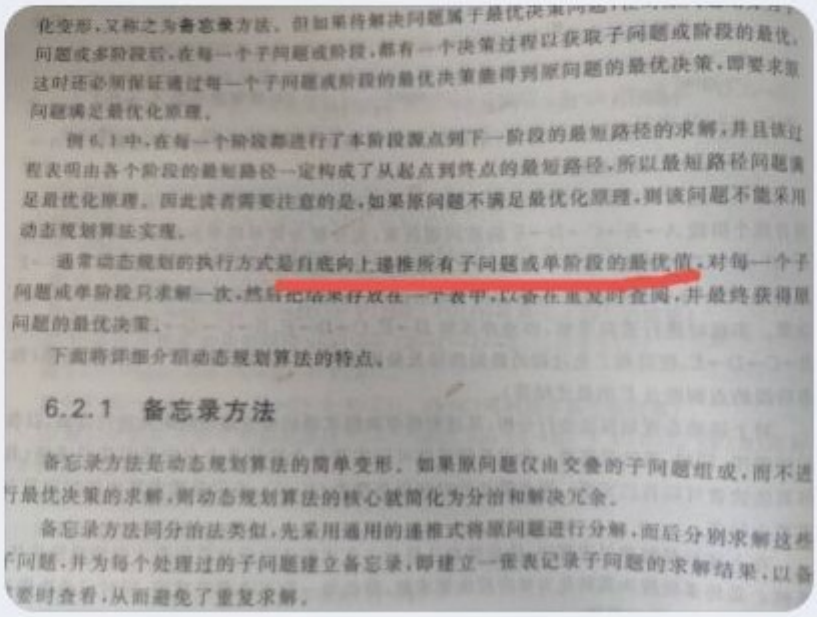
推理过程是自底向上的过程。

也就是逆推的过程

12:17

那贪心法是否是自顶向下的呢？……

而且从小规模（子问题）扩大到大规模，不应该是自底向上的吗？……



化变形，又称之为备忘录方法。但如果待解决问题属于最优决策问题，
问题或多阶段后，在每一个子问题或阶段，都有一个决策过程以获取子问题或阶段的最优，
这时还必须保证通过每一个子问题或阶段的最优决策能得到原问题的最优决策，即要求原
问题满足最优化原理。

例 6.1 中，在每一个阶段都进行了本阶段源点到下一阶段的最短路径的求解，并且该过
程表明由各个阶段的最短路径一定构成了从起点到终点的最短路径，所以最短路径问题满
足最优化原理。因此读者需要注意的是，如果原问题不满足最优化原理，则该问题不能采用
动态规划算法实现。

通常动态规划的执行方式是自底向上递推所有子问题或单阶段的最优值，对每一个子
问题或单阶段只求解一次，然后把结果存放在一个表中，以备在重复时查阅，并最终获得原
问题的最优决策。

下面将详细介绍动态规划算法的特点。

6.2.1 备忘录方法

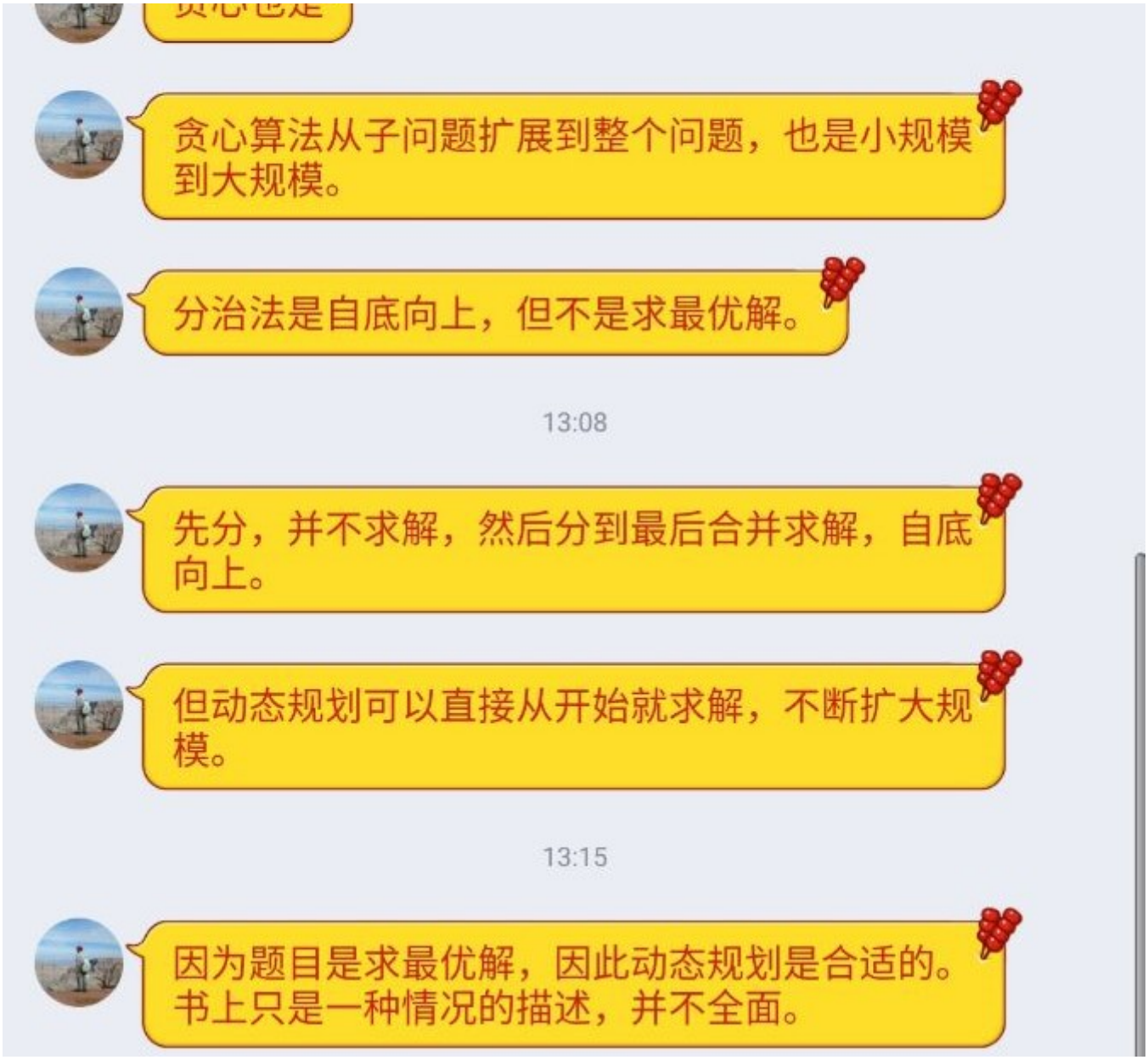
备忘录方法是动态规划算法的简单变形，如果原问题仅由交叠的子问题组成，而不进
行最优决策的求解，则动态规划算法的核心就简化为分治和解决冗余。

备忘录方法同分治法类似，先采用通用的递推式将原问题进行分解，而后分别求解这些
子问题，并为每个处理过的子问题建立备忘录，即建立一张表记录子问题的求解结果，以备
需要时查看，从而避免了重复求解。

如同书中所写的这样……

13:03

贪心也是



- **备忘录方法**：**不是最优化问题**，可采用带有记忆功能的分治法求解（是动态规划法的变形）。对于该方法，既可以**自顶向下**递归方式，也可以**自底向上**递推方式。

特点：

- 经分解后的子问题**往往不是互相独立的**（子问题重叠性质）。（与分治的区别）
- **保存**已解决的**子问题**的**结果**，避免大量重复计算。
- **必须保证**原问题满足“**最优化原理**”。
- 满足**无后效性**。

求解步骤：

1. **最优化分析**，分析最优值的结构，刻画其结构特征。
2. 递归**定义最优值**

3. **重叠子问题**分析。
4. 按**自底向上**或自顶向下记忆化方式（？）**计算**最优值和最优解。

两个基本要素：

- ☆**最优子结构**（显著特征）
- **重叠子问题**

1. 矩阵连乘问题

> 个矩阵 ，其中 可乘（即 是个 矩阵， 是个 矩阵，需要乘 次）
> 根据矩阵乘法结合律，如何确定乘法的顺序，使得所需的乘法次数 最少。
>
> 如 ，分别为 ，
> 若为 ，次数为： ；
> 若为 ，次数为： 。

- 定义 - ：矩阵 大小为
- 定义 - ：得到矩阵 的最小乘法次数，
- 动态规划方程：

- 边界条件：

2. 最长公共子序列

- 定义 - 两序列 ， 。
- 定义 - ： 与 之前的最长公共子序列。
- 动态规划方程：

- 边界条件：

3. 最长递增子序列

- 定义 - ：序列
- 定义 - ：记录 位置之前的最长子序列。
- 动态规划方程：

- 边界条件：

4. 最大子段和

- 定义 -　：序列
- 动态规划方程

  或者简记为：
- 边界条件：

5. 01背包

6. 完全背包

6. 贪心法

- 分治法、动态规划法、贪心法，都是将原问题**归纳为更小规模**的相似**子问题**。
  对分支和动规好理解，贪心法可以理解为每按贪心做一个决策，问题规模缩小。
- 所做的选择只是**局部最优解**。
- 性质
  - **贪心选择性质**：问题的整体最优解可以通过一系列局部最优选择得到。（**必须满足**）
    与动态规划不同的地方，动规选择要依赖于相关子问题的解；贪心只看当前最佳选择。
  - 最优子结构性质（跟动规一样）：问题最优解包含子问题最优解。
- 求解过程：
  从初始状态出发，构造问题的解，以“满足约束方程”为条件，运用贪心策略不断扩充解集合，直至得到问题的解。
- 几个注意的方面
  - 候选集合
  - 解集合
  - 解决函数
  - 选择函数

- 可行函数（判断是否满足约束条件）

7. 回溯法

可以看作蛮力法的改进。
【个人觉得相当于深搜dfs……

- 蛮力法：先穷举搜索，生成问题的所有可能解（解空间树），再评估是否满足约束条件。
- 回溯法x分支限界法：只构造可能解的一部分，然后评估这个部分解，如果可能导致一个可行解再进一步构造，否则不必构造。
- 几个重要概念
  - 目标函数
  - 约束条件
  - 解空间树：是所有的可能解（不一定是可行解），不考虑任何约束条件（此时还未作决策），因此一般是一个完全树。
  - 搜索空间树：回溯法的搜索空间树，会先沿着解空间树一条分支一直走下去，直到判断不符合约束条件，则回溯到上一结点换一条分支dfs。

回溯算法是一种既带有“**系统性**”和“**跳跃性**”的搜索算法。

8. 分支限界法

可以看作蛮力法的改进。
【个人觉得相当于广搜bfs+一点点启发式搜索的思想（优先队列法）……

- 只使用于求解**最优化问题**（因为要计算目标函数的上界或下界）。
- **种类**
  - 队列式（只有广搜）
  - **优先队列式**（带启发搜索）（一般都用这个）
- 扩展的重要概念
  - 活结点：队列中所有可以扩展（进行广搜）的结点。
  - 限界函数：最小值问题计算下界取最小下界，最大值问题计算上界取最大上界
- 与回溯的区别
  - 回溯采用dfs
    分支限界采用bfs
  - 回溯采用栈
    分支限界采用优先队列（堆）

- 回溯找出解空间满足约束条件的一个解
  分支限界找出满足约束条件且使目标函数获得最大或最小的解
- 回溯一个活结点可以执行多次扩展（回溯几次扩展几次）
  分支限界一个活结点只有一次机会成为扩展结点，一次性产生所有儿子结点。