
讲义 1: 期末复习

期末考试将以教学大纲中列出的内容为参考范围，本次课将帮助大家回顾本学期算法课程的主要知识点，以下标注超纲的内容一般不会出现在期末考试中。

算法分析基础

- 算法是一种有限、明确、无歧义的指令序列，用于解决特定问题或完成特定任务的指令集合。
- 程序是指通过计算机语言编写的一系列指令，可以让计算机按照一定的规则和条件执行特定的任务或操作。
- 算法可以用各种方式表示，包括文字、伪代码、流程图和编程语言。
- 对同一个算法，使用不同表示不改变这个算法的**正确性**和**渐进效率**。
- 算法执行效率用函数 $T(n)$ 度量，其中 n 是计算问题输入数据的大小。
- 算法的效率一般包括时间效率和空间效率两类。
- 算法的时间效率分析包括**最好**情况、**最坏**情况和**平均**情况下的分析，后两种可应用于不同算法之间算法效率的比较。
- 算法效率分析的渐进表示：
  - 上界 O，$f(n) = O(g(n))$ 表示函数 $f(n)$ 可以看作函数 $g(n)$ 的渐进上界。
  - 下界 Ω，$f(n) = \Omega(g(n))$ 表示函数 $f(n)$ 可以看作函数 $g(n)$ 的渐进下界。
  - 确界 Θ，$f(n) = \Theta(g(n))$ 表示函数 $f(n)$ 可以看作函数 $g(n)$ 的渐进上下界（或确界）。
- $O(1)$ 表示常数，算法指令里面诸如基本数学函数、单个数据的拷贝移动等基本指令 $op()$ 其时间复杂度均为 $O(1)$。

- $O(\lg n)$，有序序列的二分搜索、平衡二叉搜索树和跳跃表的操作等。

```
for (int i = 1; i <= n; i = 2*i)
    op()
```

- $O(n)$，序列搜索及求最大或最小值等。

```
for (int i = 1; i <= n; i++)
    op()
```

- $O(n \lg n)$，合并排序、快速排序和快速傅立叶算法等。

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= n; j=2*j)
      op()
```

- $O(n^2)$，插入排序和冒泡排序等。

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= n; j++)
      op()
```

- $O(n^3)$，两个大小为 $n \times n$ 的矩阵乘法等。

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= n; j++)
    for(int k = 1; k <= n; k++)
      op()
```

- $O(2^n)$，子集和问题等。指数时间复杂度的算法只有在输入数据 n 较小时，算法才有意义。

- 主分析法求递归表达的时间复杂度：

$$T(n) = aT(n/b) + \Theta(n^c) \tag{1}$$

  - 如果 $c < \log_b a$，$T(n) = \Theta(n^{\log_b a})$，渐进增。
  - 如果 $c = \log_b a$，$T(n) = \Theta(n^c \log n)$，渐进等。
  - 如果 $c > \log_b a$，$T(n) = \Theta(n^c)$，渐进减。

- 常用的公式和近似公式
  - 调和级数：$1 + 1/2 + 1/3 + \ldots + 1/n \sim \ln n$
  - $1 + 2 + \ldots + n \sim n^2/2$
  - $1^2 + 2^2 + \ldots + n^2 \sim n^3/3$
  - $1 + r + r^2 + \ldots + r^n = (r^{n+1} - 1)/(r - 1), r \neq 1$
  - $(1 + 1/n)^n \sim e$，$(1 - 1/n)^n \sim 1/e$
  - $\int_0^n f(x)dx \leq \sum_{i=1}^n f(i) \leq \int_1^{n+1} f(x)dx$

- P 问题是指多项式（Polynomial）时间可以求解的问题。
- NP（Non-deterministic Polynomial）问题是指多项式时间可以确定解是否正确的问题，输出为 Y 或 F。
- NP 完全（NPC）问题：1）首先是 NP 问题；2）所有 NP 问题都可化约为它。
- NP 难问题指所有 NP 问题都可化约为它的问题，NP 与 NP 难的交集等于 NPC。
- 第一个 NP 完成问题是 Cook 给出的布尔表达式可满足性问题（SAT）。SAT 问题是给定一个布尔表达式，要求找到一种赋值方式，使得该表达式成为真。
- 经典的 NPC 问题有：
  - 3-SAT（3-CNF SAT）是一种 SAT 问题的特殊情况，其中每个子句（clause）最多包含三个文字（literal）。
  - 团问题（Clique Problem）是在给定无向图 G 和一个正整数 k，判定是否存在 G 中大小不小于 k 的一个完全子图的问题。一个完全子图是指在子图中任意两个节点之间都存在一条边。
  - 顶点覆盖问题（Vertex Cover Problem）是找到一个最小的顶点集合，使得这个集合内的顶点可以覆盖图中的所有边。

  - 子集和问题（Subset Sum Problem）是指在一个由 n 个正整数构成的集合中，是否存在一个非空子集，使得该子集中的元素和等于给定的目标数 t。
  - 哈密顿回路指给定无向图，由指定的起点前往指定的终点，途中经过所有其他节点且只经过一次。

- 证明给定问题 Q 是 NPC 问题的步骤：
  - 证明 Q 是 NP 问题；
  - 确定一个已知的 NPC 问题，如 3-SAT；
  - 证明可以从 3-SAT 问题化约到问题 Q。

递归与分治算法

- 递归函数包括边界条件和递归定义，边界条件确保递归执行不会陷入死循环。
- 递归的作用：1）分解问题；2）建立问题解之间的关系。
- 递归算法的典型问题：
  - 求 $n!$。
  - 斐波那契数列问题，递归求斐波那契数的时间复杂度为 $O(2^n)$。该问题也可利用动态规划求解，时间复杂度为 $O(n)$。
  - 全排列。
  - 汉诺塔问题，二进制变化可与汉诺塔问题进行对应。
  - 整数划分问题，给定正整数 n，n 表示成其它整数的和，求整数划分的个数。令 $q(n,m)$ 表示整数 n 的划分中，被加数不超过 m 的划分数。

$$q(n,m)=\begin{cases} q(n,m-1)+1, & m=n \\ q(n,n), & m>n \\ q(n,m-1)+q(n-m,m), & n>m>1 \\ 1, & m=1 \end{cases}$$

- 二分搜索。给定有序序列，求该序列是否包含元素 x。算法的时间复杂度 $T(n) = T(n/2) + c = \lg n$。
- 大整数的乘法，将整数按位分解成高位和低位，利用代数式减少 1 次乘法运算，$T(n) = 3T(n/2) + cn = O(n^{\lg 3})$。
- Strassen 矩阵乘法，将矩阵分块，利用代数式减少 1 次分块矩阵乘法运算，$T(n) = 7T(n/2) + cn^2 = O(n^{\lg 7})$。
- 合并排序算法（重在合并），稳定（相同数排序前后位置不变）但并非原位（原位表示不需要额外空间）排序算法，可以通过自然合并的方式优化合并的效率。合并排序的时间复杂度 $T(n) = 2T(n/2) + cn = O(n \lg n)$。
- 快速排序算法（重在分解），利用支点数对序列进行划分，不稳定但是原位排序算法。可以通过随机选择支点数来避免最坏情况出现，随机快速排序属于舍伍德算法，平均时间复杂度 $O(n \lg n)$。
- 线性时间选择问题，需要从序列中找出第 k 小的数。利用分治算法求解该问题的过程包括：
  - 将序列按 5 个元素一组，分成若干组；
  - 求出每组元素的中间数；
  - **递归**求每组中间数序列的中间数，将该数作为支点数；
  - 根据支点数的索引与 k 之间的关系，要么直接返回，要么选择支点数的左边（或右边）部分继续递归求解。
  - 该算法时间复杂度 $T(n) = T(n/5) + T(7n/10) + cn = O(n)$
- 线性时间选择问题也可利用随机算法确定支点数，此时平均情况下的时间复杂度依然是 $O(n)$。
- 最接近点对问题（重在解跨界），给定平面上的 n 个点，找出距离为最小的一对点。利用分治法求解时需要考虑解可能是跨界的情况，为了提升跨界情况下计算效率，根据问题的特点可以确定只需要在一个固定大小的区域内搜索即可。时间复杂度 $T(n) = 2T(n/2) + cn = O(n \lg n)$。

- 逆序对问题，给定一个序列，求出其中逆序对（索引升序，而对应值降序）的个数。利用分治算法求解该问题，同样需要考虑解跨界的情况。算法与合并排序类似，边计数边排序，利用排序来提升跨界情况的搜索效率。时间复杂度 $T(n) = 2T(n/2) + cn = O(n \lg n)$。

- 快速傅立叶变换（超纲），利用分治算法高效求解多项式乘法，从系数计算转换为采样点计算（问题变换），为了实现递归利用复平面点对称和平方计算后的对称性，利用了复矩阵求逆运算高效实现 IFFT。时间复杂度 $T(n) = 2T(n/2) + cn = O(n \lg n)$。

贪心算法

- 常用于求解优化问题，利用对局部最优的策略期望获得全局最优解。贪心算法一般较简单，但证明该贪心算法能获得最优解往往较难。

- 贪心算法证明方法一般采用 Cut-Paste，也就是将贪心算法的部分解与优化算法的部分解进行交换，发现优化算法的部分解替换为贪心算法的部分解后依然获得全局最优，从而证明贪心算法的部分解就是优化算法的部分解。

- 贪心算法往往会需要对数据进行排序，其目的是便于实现贪心策略。

- 贪心算法实现往往使用优先队列（Heap），Heap 常用于频繁取最小（大）值。建堆时间复杂度 $O(n)$，插入和删除节点时间复杂度 $O(\lg n)$。

- 贪心算法时间复杂度分析，需要注意使用了什么数据结构。

- 活动安排问题，要求给出相容的活动子集。贪心策略是选择结束时间最早的活动。按结束时间对活动进行排序，时间复杂度 $O(n \lg n)$。

- 最优装载问题，将 n 个重量为 w_i 个集装箱装入载重为 c 的船，尽可能多的将集装箱装船。贪心策略为重量最轻者先装。对 n 个集装箱按重量排序，其时间复杂度 $O(n \lg n)$。

- 哈夫曼编码问题，自底向上的方式构造表示最优前缀码的二叉树，从而使得出现频次高的字符出现在接近根节点的位置。贪心策略是合并当前最小频次的子树，合并

后频次为子树频次的和。平均码长等于字符频次乘以字符在前缀码二叉树深度的累加和。在算法执行过程包括 $O(n)$ 次堆操作 $O(\lg n)$，时间复杂度 $O(n \lg n)$。

- Shannon Source Coding 定理（超纲），字符的熵（Entropy）是平均码长的下界。$H(x) = \sum_{i=1}^{n} -p(x_i) \log_2 p(x_i)$。
- 最短路径系列问题：
  - 图中权重值均大于 0，且各个权重值相等，利用宽度优先 BFS 算法求最短路径，时间复杂度 $O(|E| + |V|)$。
  - 有向无环（DAG）图，权重有正有负，按节点拓扑排序执行松弛（Relaxation）计算来求最短路径，时间复杂度 $O(|E| + |V|)$。松弛计算是基于三角不等式，对节点距离值更新的算法。拓扑排序可采用深度优先算法，时间复杂度 $O(|E| + |V|)$。
  - 图的权重非负，则可利用 Dijkstra 算法求最短路径，贪心策略是从备选节点中选择距离值最小的节点加入到解集中。采用优先队列实现 Dijkstra 算法，时间复杂度 $O(|V| \lg |V|)$，如果使用数组实现则时间复杂度为 $O(|V|^2)$。
  - 当图中存在累加权重为负值的环，采用 Bellman-Ford 算法求最短路径（超纲）。Bellman-Ford 是一类动态规划算法，可利用该算法判断给定图是否存在累加和为负值的环。需要将图构造成 DAG 图，DAG 图有 $|V|(|V| + 1)$ 个节点，$|V|(|V| + |E|)$ 条边。时间复杂度 $O(|V||E|)$。
  - 给定没有负环的图（图的权重仍然有正有负），求图中各个节点之间相互的最短路径，可采用 Johnson 算法（超纲）。把图转换成权重只有正值的图，调用 Dijkstra 算法，因此时间复杂度 $O(|V|(|V| \lg |V|))$。在执行图转换时采用出边累加 h，出边减去 h 的策略，类似最大流算法中流量保持。
- 最小生成树问题：
  - 生成树是包含图 G 所有节点的树。图 G 的各个生成树中，节点累加权重最小的即为 MST。
  - Prim 算法和 Kruskal 算法都属于贪心算法。
  - Prim 算法的贪心策略为选择节点相邻最小的边，将它所连接的另一个节点添加到最小生成树中。采用堆实现 Prim 算法的时间复杂度 $O(|E| \lg |V|)$。

  - Kruskal 算法先将各个节点看做独立的子树，然后按照边的权重从小到大（贪心策略）将边加入到生成树中，直到生成树中含有所有节点为止。**添加边的过程要保证不会形成环**。使用并查集实现算法的时间复杂度 $O(|E|\lg|E|)$。
  - 稠密图中边的数量可能会达到 $O(|V|^2)$，这时 Prim 算法优于 Kruskal 算法。
  - 并查集支持合并两个元素所在的集合，以及查询两个元素是否连通的操作。

- 贪心算法其它有趣的问题可见 `https://mooc1-1.chaoxing.com/course/232924554.html?edit=true&knowledgeId=718069253&module=2#content`。

动态规划算法

- 动态规划算法求解的基本步骤 SRTBOT：
  - Subproblem：定义子问题，将输入看成序列，将序列的前缀（后缀）或中缀当作子问题的输入。
  - Relate：建立子问题解之间的递归关系，先猜再穷举。
  - Topology sort：子问题求解符合拓扑排序，即当前问题解只依赖已经求出问题的解，可利用自底向上方式实现。
  - Base case：边界条件，一般是输入数据大小为 0 或 1 时的解。
  - Original problem：原问题的解，一般是动态规划表中的某个位置。
  - Time：时间复杂度是子问题个数 × 每一子问题求解的时间。
- 最长递增子序列

$$DP[i] = \max\{DP[j] + 1\}, j = i-1, i-2, \ldots, 1$$

- 木料切割

$$DP[i] = \max\{v[j] + DP[L-j]\}, j = 1, 2, \ldots, L$$

- 最大子串和

$$DP[i] = \max\{DP[i-1] + A[i], A[i]\}$$

- 单词排版问题

$$DP[i] = \min\{DP[j] + badness([j,i])\}, j = i-1, i-2, \ldots, 1.$$

超出页面宽度的划分时设 DP 值为无穷大。

- 0-1 背包问题

$$DP[i,x] = \max\{DP[i-1,x], DP[i-1,x-w[i]] + v[i]\},$$

$x < w[i]$ 表示物品 i 放不下，此时可设 DP 值为无穷小，时间复杂度为伪多项式时间。

- 最长公共子序列

$$DP[i,j] = \begin{cases} DP[i-1,j-1], A[i] == B[j] \\ \max\{DP[i-1,j], DP[i,j-1]\}, \text{其它} \end{cases}$$

- 矩阵连乘问题

$$DP[i,j] = \min\{DP[i,k-1] + DP[k,j] + r[i] \times r[k] \times c[j]\}, k = i+1, \ldots, j$$

- 最优二叉搜索树

$$DP[i,j] = \begin{cases} w[i], i = j. \\ \min\{DP[i,r-1] + DP[r+1,j] + (w[i] + \ldots + w[j])\}, r = i, i+1, \ldots, j. \end{cases}$$

- 子集和问题 (整数规划)，给定大小为 n 的正整数集合 A，问是否存在集合 A 的子集，其累加和等于 S。这是一个决策问题，即问题的输出是 Y 或者 F。

$$DP[i,x] = \text{or} \begin{cases} DP[i-1,x-A[i]], x \geq A[i] \\ DP[i-1,x] \end{cases}$$

其中 $i = 1, 2, \ldots, n$，$x = 1, 2, \ldots, S$，时间复杂度为伪多项式时间。

回溯算法

回溯算法与分支界限算法都可以看作是一类穷举算法。这两个算法之间也存在不同：

- 回溯算法是一种通过不断回溯来寻找问题的解的算法，它依赖于深度优先搜索和剪枝技术。而分支界限算法是一种通过逐步扩展搜索状态空间，利用剪枝来提高求解效率的算法。
- 回溯算法沿着一个子集空间自顶向下搜索，在搜索过程中需要反复地回溯和重复计算。分支界限算法沿着一棵搜索树进行搜索，并且每次只考虑解空间的一个局部范围。
- 回溯算法适用于在所有可能方案中搜索一组特定的解，常用于求解排列组合、子集、图的遍历等问题。而分支界限算法适用于在大规模解空间（如线性规划、背包问题、旅行商问题等）中找最优解或次优解的情况。
- 回溯算法的剪枝方式主要是通过约束条件或者可行性条件来剪枝。而分支界限算法则根据目标函数，采用限界法剪枝。

回溯算法求解问题的 3 个关键：

- 选择，即确定解的形式或者解空间。判断是排列还是 0/1 选择。
- 约束，解需要满足的约束，可用于剪枝。
- 目标。

典型的回溯算法求解的问题包括：

- 装载问题，n 个集装箱要装上 2 艘不同载重的轮船，其中轮船的总载重大于等于集装箱的总重量。该问题不一定有解，即不一定能将所有集装箱装船。最优装载方案是先将第一艘轮船尽可能装满，然后将剩余的集装箱装上第二艘轮船。算法时间复杂度为 $O(2^n)$。
- 子集和问题，算法时间复杂度为 $O(2^n)$。该问题是经典 NP 完全问题。
- n 皇后问题，算法时间复杂度为 $O(n!)$。

- 0-1 背包问题，算法时间复杂度为 $O(n2^n)$。
- 节点数为 n 图的 m 着色问题，算法时间复杂度为 $O(nm^n)$。该问题是经典 NP 完全问题。
- 旅行售货员问题，算法时间复杂度为 $O(n!)$。

分支界限算法

是一种用于求解组合优化问题的算法，它通过不断地缩小问题的解空间，逐步逼近全局最优解或次优解。分支界限算法使用一棵搜索树来维护解空间，每个结点表示一组可能的解，搜索过程就是对这个搜索树的深度、宽度或者优先遍历。分支界限算法常用的数据结构有队列和优先队列。

典型的分支界限算法有：

- 单源最短路径问题，时间复杂度 $O(b^V)$，b 表示分支因子。
- 装载问题，算法时间复杂度为 $O(2^n)$。
- 布线问题，印刷电路板将布线区域划分成 $n \times m$ 个方格阵列。精确的电路布线问题要求确定连接方格 a 的中点到方格 b 的中点的最短布线方案。在布线时，电路只能沿直线或直角布线。算法时间复杂度为 $O(mn)$。
- 0-1 背包问题，算法时间复杂度为 $O(2^n)$。
- 最大团问题是无向图中找到一个完全子图，使得该子图中的所有节点都相互连通，并且包含的节点数最多。该问题是经典 NP 完全问题。
- 旅行售货员问题。该问题是经典 NP 完全问题。

随机算法

- 示性随机变量在求期望中的应用。比如抛硬币若干次，平均需要多少次会出现第一个 H。通过示性函数，可以发现这个问题对应一个几何序列 $1 + 1/2 + 1/4 + \ldots = 2$。

- 生日悖论问题，房间人数 k 与天数 n 之间的关系 $k = \Theta(\sqrt{n})$。
- 最优停时问题算法。采样空间（$1/e \approx 37\%$）用于对整体的观察，之后的决策空间尽快作出决策。
- 随机重排。Fisher-Yates 随机采样算法，从最后一个元素开始，每次从剩余元素中随机选择一个并交换位置，直到第一个元素被选择。算法时间复杂度 $O(n)$。
- 伪随机数生成的简单算法 $a[n] = (b \times a[n] + c) \mod m, n = 1, 2, \ldots$。
- 随机算法主要可以分为三大类：蒙特卡洛算法（Monte Carlo）、拉斯维加斯算法（Las Vegas）和舍伍德算法 (Sherwood)。
  - Monte Carlo 一般利用随机采样求解问题，算法效率一般是多项式时间，但其结果是以高概率正确。
  - Las Vegas 不会返回不正确的解，但在有些轮次随机选择可能不返回任何结果，运行时间一般是平均情形下的多项式时间。该算法主要利用随机性进行提速。
  - Sherwood 通过随机性避免算法陷入到最糟糕的情形，总是得到正确结果。
- 典型 Monte Carlo 算法，如求 π、求定积分、验证矩阵乘积是否等于另一个矩阵、素数测试等。
- 典型 Sherwood 算法，如线性时间选择、跳跃表 ($O(\lg n)$) 等。
- 跳跃表（Skip List）是在有序链表的基础上增加多级索引（跳跃列表），以提高搜索的效率数据结构。常用于 Redis 这类非关系型数据库。比较跳跃表与 AVL 树之间的异同。AVL 树的平衡因子是左子树深度减去右子树的深度。区别树节点的深度和高度的定义。
- 典型 Las Vegas 算法，n 皇后问题：
  - 可以先在棋盘的若干行中随机地放置皇后，然后在后继行中用回溯法继续放置，直至找到一个解或宣告失败。
  - 随机放置的皇后越多，后继回溯搜索所需的时间就越少，但失败的概率就越大。
- 典型 Las Vegas 算法，Pollard 算法是一种用于快速分解整数的算法，其主要思想是利用随机化技术，选取一个随机数序列对该整数进行分解。

量子算法 (超纲)

量子算法是运用量子力学原理来实现运算的算法。量子力学是能够描述微观物理世界的理论，但是人们发现量子力学原理也可以用来解决一些经典计算机无法解决的问题，例如因子分解、求解线性方程组、求解优化问题等。

与经典计算机不同的是，量子算法使用的基本单位是量子比特 (qubit)，其值不仅可以是 0 或 1，还可以是两者的叠加态，即量子叠加态。这意味着在某些情况下，量子算法可以同时处理多个值，从而在某些问题上比经典计算机更快速。

- 对比特的操作是通过矩阵乘法作用于比特向量来实现。
- 量子计算机只使用可逆的运算。
- 多个比特状态可以写作比特向量的张量积。
- 量子比特可以看作超球体上的一个点。
- 处于叠加态的量子比特，以概率塌缩为 Cbits。
- Cbits 是特殊的量子比特。
- 量子比特的操作同样是利用矩阵乘法。
- 量子门可以看作为矩阵。
- 矩阵是映射的工具。
- 量子算法设计简单来看，就是组合量子门使得它满足计算要求。
- 并非所有的 NP 完全问题都有高效的量子算法。
- 研究量子算法并不一定要在量子计算机。