University of Chinese Academy of Sciences

# 计算机算法设计与分析

**083500M01001H-2**

# 2021 期末试卷解答

2022 年 12 月 15 号

*Professor:* 刘玉贵

**学生: 周胤昌**
**学号: 202228018670052**
**学院: 网络空间安全学院**
**所属专业: 网络空间安全**
**方向: 安全协议理论与技术**

# 一、多选填空 (10 分, 每题 2 分)

**1.** 给定函数 $f(n)$ 和 $g(n)$, 判断这两个函数阶的关系:_________.
**Solution:** 熟练掌握渐进上界、渐进同阶及渐进下界的概念即可解决. 此处题目不清, 就不解答了.

**2.** 下面属于遗传算法实现的关键技术问题的有 ______.

(1) .解的编码　　(2) .初始种群的选择　　(3) .邻域定义　　(4) .适应函数

**Solution:** 遗传算法实现的关键技术问题有**解的编码、适应函数、初始种群的选取、交叉规则以及终止规则**. 因此选择 (1), (2), (3).

**3.** 实现 $n$ 皇后问题求解, 有下几种方法哪几种:_______(四选二).

(1) .贪心算法　　(2) .回溯算法　　(3) .分支限界算法　　(4) .Las Vegas 算法

**Solution:** $n$ 皇后问题的求解可以用回溯算法 (第 5-7 章 PPT 内容), 也可用 Las Vegas 算法 (第 9 章 PPT 内容). 因此选择 (2) 和 (4).

**4.** 对于判定问题 $\Pi$ 的 Monte Carlo 算法, 当返回 false(true) 时解总是正确的, 但当返回 true(false) 时解可能有错误, 该算法是 ______.

(1) .偏真的*Monte Carlo*算法　　(2) .偏假的*Monte Carlo*算法
(3) .一致的*Monte Carlo*算法　　(4) .不一致的*Monte Carlo*算法

**Solution:** 答案选 (2), 只要将偏真的 Monte Carlo 算法的定义中的 true/false 互换即可得到偏真的 Monte Carlo 算法的定义.

**5.** 下面说法, 正确的是 ______.

(1) .$\mathcal{P}$类问题是存在多项式时间算法的问题　　(2) .$\mathcal{NP}$类问题是不存在多项式时间算法的问题
(3) .$\mathcal{P}$类问题一定也是$\mathcal{NP}$类问题　　(4) .$\mathcal{NP}$类问题比$\mathcal{P}$类问题难解

**Solution:** 注意一下 PPT 中的概念定义即可, 答案选择 (1) 和 (3).

# 二、判断正误 (10 分, 每题 2 分)

这里只记得 3 个, 填空题和判断题基本上都是作业里面的选择题和判断题[1].
**1.** 0/1 背包问题的贪心算法 (单位价值高优先装入...) 是绝对近似算法. (　)
**Solution: 错误** (✗), 0/1 背包问题的贪心算法**不是**绝对近似算法, 是 $\epsilon$-近似算法 (其中 $\epsilon = 2$ 并且该算法是 **GKK** 算法).

**2.** 多机调度问题的贪心近似算法 (按输入顺序将作业分配给当前最小负载机器) 是 $\epsilon$-近似算法. (　)
**Solution: 正确** (✓), 多机调度问题的贪心近似算法有 GMPS 和 DGMPS 分别是 2-近似和 3/2-近似算法.

**3.** 禁忌搜索中, 禁忌某些对象是为了避免邻域中的不可行解. (　)
**Solution: 错误** (✗), 选取禁忌对象是为了引起解的变化, 根本目的在于避开邻域内的局部最优解而不是不可行解.

[1]主要是集中在第八章至十一章 ($\mathcal{NP}$ 完全理论, 近似算法, 启发式算法里面), 都是一些概念性问题.

# 三、简答题 (25 分)

**1.** 写出模拟退火算法的主要步骤. (5 分)

**Solution:** 模拟退火算法的主要步骤如下算法1中所示:

**Algorithm 1** 模拟退火算法步骤

任选初始解 $x_0$ 并初始化 $x_i \leftarrow x_0, k \leftarrow 0, t_0 \leftarrow t_{\max}$(初始温度);
**while** $k \le k_{\max}$ && $t_k \ge T_f$ **do**
  从邻域 $N(x_i)$ 中随机选择 $x_j$, 即 $x_j \leftarrow_R N(x_i)$;
  计算 $\Delta f_{ij} = f(x_j) - f(x_i)$;
  **if** $\Delta f_{ij} \le 0 \,\|\, \exp\left(-\Delta f_{ij}/t_k\right) > \text{RANDOM}(0,1)$ **then**
    $x_i \leftarrow x_j$;
  **end if**
  $t_{k+1} \leftarrow d(t_k)$;
  $k \leftarrow k+1$;
**end while**

**2.** 写出一个贪心算法, 来近似估计多机调度问题的解, 并说明该解是什么近似的 (不需要证明) (5 分)

**Solution:** 贪心算法 GMPS 为: 按输入的顺序分配作业, 把每一项作业分配给当前负载最小的机器. 并且 GMPS 算法的近似比为 $\epsilon = 2$. 也可以有 DGMPS 算法: 将任务按处理时间从大到小排序, 然后按 GMPS 算法进行调度. 该算法的近似比为 $\epsilon = 3/2$.

**3.** 证明相遇集 (碰撞集) 问题是 $\mathcal{NPC}$ 问题. (10 分)

**Solution:** **碰撞集问题:** 给定一组集合 $\{S_1, S_2, \cdots, S_n\}$ 和预算 $b$, 问是否存在一个集合 $H$, 其大小不超过 $b$, 且 $H$ 和所有 $S_i(i = 1, 2, \cdots, n)$ 相交非空.

- **先证明该问题是一个 $\mathcal{NP}$ 问题**: 假设给出集合 $H$ 的所有元素, 显然可以在**多项式时间内**验证该集合 $H$ 是否满足条件要求 (和 $S_i$ 逐一比较是否有交集并检查规模是否超过 $b$), 所以该问题 $\in \mathcal{P}$ 问题 $\subseteq \mathcal{NP}$ 问题.
- **再利用一个已知的 $\mathcal{NPC}$ 问题, 将其 (在多项式时间内) 归约到目标问题**: 已知**图的顶点覆盖问题 (VC)** 是 $\mathcal{NPC}$ 问题, 只要找到一种把 VC 问题归约到碰撞集问题的多项式方法, 即可证明碰撞集问题是 $\mathcal{NPC}$ 问题. 具体的归约方式构造如下:

  假设有图 $G = (V, E)$, 则把该图的每一条边对应一个集合 $S_i$, 该边的两点作为该集合的元素, 即每个集合都有两个元素 (如 $S_1 = \{v_1, v_2\}$). 这样就可以构造出 $|E|$ 个集合 $\left\{S_1, S_2, \cdots, S_{|E|}\right\}$, 再将 VC 问题中覆盖的顶点数上限 $K$ 作为预算 $b$, 并把图 $G = (V, E)$ 的顶点覆盖中的所有点作为集合 $H$ 的元素. 另外还需要说明一下上述多项式变换的充分必要性:

  - **当碰撞集问题有解时, 则顶点覆盖问题就有解:** 只需要选取碰撞集的解 $H$ 对应的所有点, 即为对应顶点覆盖问题的解;
  - **当顶点覆盖问题有解时, 则碰撞集问题就有解:** 当顶点覆盖问题有解 $V$ 时, 则将 $V$ 中的每个顶点对应到所生成的那一组集合 (即 $\left\{S_1, S_2, \cdots, S_{|E|}\right\}$) 中的元素, 从而得到集合 $H$, 即为碰撞集问题的解.

这样就把 VC 问题归约到碰撞集问题了, 而 VC 问题是 $\mathcal{NPC}$ 问题, 因此碰撞集问题是 $\mathcal{NPC}$ 问题, $\square$.

**4.** 读程序题, 读懂下列快速 Fourier 变换的程序, 并使用迭代法分析该程序的时间复杂度. (5 分)

**Algorithm 2** 算法 **FFT**$(N, a, w, A)$

**Input:** $N = 2n$,$w$ 为 $n$ 次单位根, $a$ 是 $N$ 元数组 (多项式 $a(x)$ 的系数向量)
**Output:** $N$ 元数组 $A$, $A[j] = a\left(w^j\right), j = 0, 1, \cdots, N-1$

**integer** $j$;
**real** $b[\;], c[\;]$;
**complex** $B[\;], C[\;], wp[\;]$;
**if** $N = 1$ **then**
    $A[0] := a[0]$;
**else**
    $n := N/2$;
    **for** $j$ **from** 0 **to** $n-1$ **do**
        $b[j] := a[2j+1], c[j] := a[2j]$;
    **end for**
**end if**
**FFT** $(n, b, w \cdot w, B)$ , **FFT** $(n, c, w \cdot w, C)$;
$wp[0] := 1$;
**for** $j$ **from** 0 **to** $n-1$ **do**
    $wp[j+1] := w \cdot wp[j]$;
    $A[j] := C[j] + B[j] \cdot wp[j]$;
    $A[j+n] := C[j] - B[j] \cdot wp[j]$;
**end for**
**end {FFT}**

**Solution:** 问题划分为 2 个规模为 $n/2$ 的子问题, 算法内部的两个 for 循环 (循环体内部都是常数时间的操作) 都只需要耗时 $O(n)$. 因此 $T(n) = 2T(n/2) + cn$. 不妨设 $n = 2^k$(否则 $\exists k \in \mathbb{N}^*$, 使得 $2^k \le n < 2^{k+1}$). 于是有如下:

$$
\begin{aligned}
T\left(2^k\right) &= 2^1 T\left(2^{k-1}\right) + c \cdot 2^k \\
&= 2^2 T\left(2^{k-2}\right) + 2c \cdot 2^k \\
&= 2^3 T\left(2^{k-3}\right) + 3c \cdot 2^k \\
&\vdots \\
&= 2^k T(1) + kc \cdot 2^k
\end{aligned}
$$

不妨设 $T(1) = 0$, 因此 $T(n) = \Theta(n\log n)$.

# 四、算法设计题 (55 分)

**1.** 有 $n$ 个进程 $p_1, p_2, .., p_n$, 进程 $p_i$ 的开始时间为 $s[i]$, 截止时间为 $d[i]$. 可以通过检测程序 Test 来测试正在运行的进程, Test 每次测试时间很短, 可以忽略不计, 即如果 Test 在时刻 $t$ 测试, 那么它将对满足 $s[i] \le t \le d[i]$ 的所有进程同时取得测试数据. 问: 如何安排测试时刻, 使得对每个进程至少测试一次, Test 测试的次数达到最少? 设计算法并证明正确性, 分析算法复杂度. (20 分)

**Solution: 贪心策略**: 将进程按照 ddl 进行排序. 取第 1 个进程的 ddl 作为第一个测试点, 然后顺序检查后续能够被这个测试点检测的进程 (这些进程的开始时间 $\le$ 测试点), 直到找到下一个不能被测试到的进程为止. 伪码见如下算法3:

**Algorithm 3** **Test** 算法

**Input:** 开始时间的数组 $s[1, \cdots, n]$, 截止时间的数组 $d[1, \cdots, n]$
**Output:** 数组 $t$: 顺序选定的测试点构成的数组

```
1:  将进程按照 d[i] 递增的顺序进行排序 (使得 d[1] ≤ d[2] ≤ ··· ≤ d[n]);
2:  i := 1; t[i] := d[1]; j := 2              ▷ 第一个测试点是最早结束进程的 ddl
3:  while j ≤ n && s[j] ≤ t[i] do             ▷ 检查进程 j 是否可以在时刻 t[i] 被测试
4:      j++;
5:  end while
6:  if j > n then
7:      return t;
8:  else
9:      t[++i] := d[j++], goto 3;             ▷ 找到待测进程中结束时间最早的进程 j
10: end if
11: end {Test}
```

**结论: 对于任意正整数 $k$, 存在最优解包含算法前 $k$ 步选择的测试点.**

证明. $k = 1$ 时, 设 $S = \{t[i_1], t[i_2], \cdots\}$ 是最优解, 不妨设 $t[i_1] < t[1]$. 设 $p_u$ 是在时刻 $t[i_1]$ 被测到的任意进程, 那么 $s(u) \le t[i_1] \le d[u]$, 从而有

$$s[u] \le t[i_1] < t[1] = d[1] \le d[u] \tag{1}$$

因此 $p_u$ 也可以在 $t[1]$ 时刻被测试. 于是在 $S$ 中用 $t[1]$ 替换掉 $t[i_1]$ 后也可得到一个最优解.

假设对于任意 $k$, 算法在前 $k$ 步选择了 $k$ 个测试点 $t[1], t[i_2], \cdots, t[i_k]$ 且存在最优解

$$T = \{t[1], t[i_2], \cdots, t[i_k]\} \cup T' \tag{2}$$

设算法前 $k$ 步选择的测试点不能测到的进程构成集合 $Q \subseteq P$, 其中 $P$ 为全体进程集合. 不难证明 $T'$ 是子问题 $Q$ 的最优解[2]. 根据归纳假设可得知, $\exists Q$ 的最优解 $T^*$ 包含测试点 $t[i_{k+1}]$, 即

$$T^* = \{t[i_{k+1}]\} \cup T'' \tag{3}$$

因此有

$$\{t[1], t[i_2], \cdots, t[i_k]\} \cup T^* = \{t[1], t[i_2], \cdots, t[i_{k+1}]\} \cup T'' \tag{4}$$

也是原问题的最优解, 根据归纳法可知命题成立. □

算法的时间复杂度为 $T(n) = O(n \log n) + O(n) = O(n \log n)$.

[2]反证法: 假设 $T'$ 不是子问题 $Q$ 的最优解, 则可以构造出比 $T$ 更优的解, 这显然与 $T$ 的最优性相矛盾.

**2.** 请设计动态规划算法解决该双机调度问题: 用两台处理机 $A$ 和 $B$ 处理 $n$ 个作业. 设第 $i$ 个作业交给机器 $A$ 处理时所需要的时间是 $a_i$, 若由机器 $B$ 来处理, 则所需要的时间是 $b_i$. 现在要求每个作业只能由一台机器处理, 每台机器都不能同时处理两个作业. 设计一个动态规划算法, 使得这两台机器处理完这 $n$ 个作业的时间最短 (从任何一台机器开工到最后一台机器停工的总时间). (20 分)

**Solution:** 在完成前 $k$ 个作业时, 设机器 $A$ 工作了 $x$ 时间 (注意 $x$ 限制为正整数), 则机器 $B$ 此时最小的工作时间是 $x$ 的函数. 设 $F[k][x]$ 表示完成前 $k$ 个作业时, 机器 $B$ 最小的工作时间, 则有

$$F[k][x] = \min\{F[k-1][x] + b_k, F[k-1][x-a_k]\}$$

其中 $F[k-1][x]+b_k$ 对应的是第 $k$ 个作业由机器 $B$ 来处理, 此时完成前 $k-1$ 个作业时机器 $A$ 的工作时间仍是 $x$, 则 $B$ 在 $k-1$ 阶段用时为 $F[k-1][x]$; 而 $F[k-1][x-a_k]$ 对应第 $k$ 个作业由机器 $A$ 处理 (完成 $k-1$ 个作业, 机器 $A$ 工作时间时 $x-a[k]$, 而 $B$ 完成 $k$ 阶段与完成 $k-1$ 阶段用时都为 $F[k-1][x-a_k]$). 于是完成前 $k$ 个作业所需要的时间为 $T=\max\{x, F[k][x]\}$. 根据上述递推关系很容易证得问题满足最优子结构性质. 并且通过下述算法代码可知时间复杂度为 $O\left(n\cdot\min\left\{\sum_{i=1}^{n}a_i, \sum_{i=1}^{n}b_i\right\}\right)$.

```
int schedule() {
int sumA = a[1], time[n];
//k = 1 的情况
for(int x = 0; x < a[1]; x++) {
    F[1][x] = b[1];
}
F[1][a[1]] = min(b[1],a[1]);
//初始化
for(int i = 2; i <= n; i++) {
    for(int j = 0; j <= n; j++) {
        F[i][j] = INT_MAX;
    }
}
//k >= 2 的情况
for(int k = 2; k <= n; k++) {
    sumA += a[k];
    time[k] = INT_MAX;
    for(int x = 0; x <= sumA; x++) {
        if(x < a[k]) {
            F[k][x] = F[k-1][x] + b[k];
        } else {
            F[k][x] = min(F[k-1][x] + b[k], F[k-1][x-a[k]]);
        }
        //判断完成作业 k 时, 到底是机器 B 所需最小时间小, 还是 A 所需时间小
        time[k] = min(time[k],max(x,F[k][x]));
    }
}
return time[n];
}
```

**3.** 最佳调度问题: 假设有 $n$ 个任务要由 $k$ 个可并行工作的机器来完成, 完成任务 $i$ 需要的时间为 $t_i$. 试设计一个分枝限界算法 (要求写出剪枝过程), 找出完成这 $n$ 个任务的最佳调度, 使得完成全部任务的时间 (从机器开始加工任务到最后停机的时间) 最短. (15 分)

**Solution: 限界函数:** 将 $n$ 个任务按照所需时间非递减排序, 得到任务序列 $1, 2, \cdots, n$, 满足时间关系 $t[1] < t[2] < \cdots < t[n]$. 将 $n$ 个任务中的前 $k$ 个任务分配给当前 $k$ 个机器, 然后将第 $k+1$ 个任务分配给最早完成已分配任务的机器, 依次进行, 最后找出这些机器最终分配任务所需时间最长的, 此时间作为分支限界函数. 如果一个扩展节点所需的时间超过这个已知的最优值, 则删掉以此节点为根的子树. 否则更新最优值.

**优先级:** 哪台机器完成当前任务的时间越早, 也就是所有机器中最终停机时间越早, 优先级就越高, 即被选作最小堆中的堆顶, 作为扩展节点. 分支限界算法如下所示:

```
Node{
    int Path[n];
    int T[k];
    int Time;
    int length;
}
Proc BestDispatch(int n, int k, int t[]){
    Node Boot, X, P, result;
    int f;
    f = n * max(t[]);
    Boot.T[n] = {0};
    Boot.Time = 0;
    Boot.Path[n] = {0};
    Boot.length=0;
    AddHeap(Boot);
    while (!Heap.empty()) do {
        P = DeleteMinHeap();
        for i = 1 to k do {
            X = Newnode(P.Path[], P.T[], P.length + 1);
            X.Path[X.length] = i;
            X.T[i] = X.T[i] + t[X.length];
            X.Time = max(X.T[]);
            if X.length == n then {
                if X.Time < f then {
                    f = X.Time;
                    result = X;
                }
            }
            else {
                if X.Time < f then {
                    AddHeap(X);
                }
            }
        }
    }
}
end {BestDispatch}
```