期末测试练习题——计算机算法设计与分析

1. 试卷中出现的lg均以2为底数。
2. 对于要求设计算法的题目，需要给出正确性证明和复杂度分析过程。
3. 尽量使用简洁的自然语言描述算法。
4. 本试卷作为练习使用，其题目取自各参考资料。在你决定查阅文献或求助Google之前，请先行思考。

Problem 1: Asymptotic Notation

将以下各表达式按照渐近阶的升序排列。

$$f(n) = n^{\lg 8}, g(n) = 2^{(2\lg n)}, h(n) = (\min(n, 2))^5,$$

$$F(n) = 49F(\frac{n}{25}) + n^{\frac{3}{2}}\lg n, F'(n) = 2F'(\frac{n}{4}) + 1,$$

$$G(n) = G(\sqrt{n}) + 1, H(n) = H(n-1) + \lg n.$$

(提示：对于$G(n)$，可考虑变量代换$m = \lg n$；也可猜一个答案，使用数学归纳法证明。)

Solution:

1. $f(n) = n^{\lg 8} = n^3$
2. $g(n) = 2^{2\lg n} = 2^{\lg n^2} = n^2$
3. $h(n) = (\min(n, 2))^5 = 2^5 = \Theta(1)(n \to \infty)$
4. $F(n) = 49F(\frac{n}{25}) + n^{\frac{3}{2}}\lg n$。$\lg_{25} 49 = \lg_5 7 \approx 1.21 < 1.5$，故

$$n^{\frac{3}{2}}\lg n \in \Omega(n^{\lg_{25} 49+\epsilon}), n^{\frac{3}{2}}\lg n \in O(n^{\lg_{25} 49+\delta})$$

比如$\epsilon = 0.2, \delta = 0.3$。应用Master Theorem第三条，得$F(n) = \Theta(n^{\frac{3}{2}}\lg n)$。

5. $F'(n) = 2F'(\frac{n}{4}) + 1$，直接使用Master Theoerm，得$F'(n) = \Theta(\sqrt{n})$。
6. 变量代换$m = \lg n, n = 2^m$，则有$G(2^m) = G(2^{m/2}) + 1$。令$S(m) = G(2^m)$，则$S(m) = S(\frac{m}{2}) + 1 \Rightarrow S(m) = \Theta(\lg m)$，所以$G(n) = \Theta(\lg \lg n)$。
7. $H(n) = H(n-1) + \lg n$。展开该递归式，得$H(n) = \sum_{k=1}^{k=n} \lg k = \Theta(n \lg n)$。

按渐近阶排序依次为：

$$h(n) = \Theta(1), G(n) = \Theta(\lg \lg n), F'(n) = \Theta(\sqrt{n}), H(n) = \Theta(n \lg n),$$

$$F(n) = \Theta(n^{\frac{3}{2}} \lg n), g(n) = n^2, f(n) = n^3.$$

Problem 2: Mathematical Induction

有两堆棋子，数目相等。两人游戏，轮流取子。每人可以从一堆里取走任意数量的棋子，但是不能同时从两堆里取子。游戏规定取得最后一颗棋子者胜。请证明，后取者有必胜策略。

(提示：当然，你需要先为后取者设计必胜策略。)

Solution: 后取者的必胜策略为：始终保持两堆棋子数目相等。

使用数学归纳法证明该策略的正确性。对每堆棋子的数目n作归纳。

1. 基础步骤：$n = 1$，易见。
2. 归纳假设：假设每堆棋子的数目为$n \leq k$时，上述策略都能保证后取者必胜。
3. 归纳步骤：当每堆棋子数为$n = k+1$时，假设先取者从某堆取走$x \leq (k+1)$颗棋子，后取者按照策略从另外一堆取走同样数目的棋子。则问题规约为$n = k + 1 - x$的子问题。根据归纳假设，上述策略能保证后取者必胜。

Problem 3: Divide and Conquer

给定顶点数为n的完全二叉树$T(n = 2^d - 1$，for some $d)$。树T中每个顶点v都附带着一个实数值x_v。x_v各不相同。如果顶点v的值x_v小于其所有邻居节点(包括其子节点和父节点)附带的实数值，则称顶点v为局部最小值点。

设计$O(\lg n)$算法确定T中的某个局部最小值顶点。算法的关键操作为Probe(v)：对于每个顶点v，Probe(v)返回x_v。

Solution:

- 如果完全二叉树顶点数为1，即只有一个节点v，直接返回v。
- 否则，考虑根节点r，如果其两个子节点的值均比x_r大，则r为局部最小值点。返回r。
- 否则，两个子节点u, v中至少有一个节点的值比x_r小。比如u节点。如果两个子节点的值均比x_r小，则任取一个即可。
- 递归处理以u为根节点的子树。

算法复杂度分析：

$$T(n) = T(\frac{n}{2}) + 1 = \Theta(\lg n).$$

Problem 4: Search

给定整数数组A，其元素按升序存储。数组大小为n，但是具体数值未知。给定某整数i，设计时间复杂度为$\Theta(\lg n)$ 的算法判断i是否在A中。

Solution:

- 依次检查数组第$1, 2, 4, \ldots, 2^k, 2^{k+1}$个数，直到$2^k < i < 2^{k+1}$(如果$i = 2^k$，则只检查前$k$个元素即可)。
- 在$2^k, 2^{k+1}$区间段采用二分查找，搜索i。

该算法复杂度为$\Theta(\lg n)$，满足条件。

Problem 5: Amortized Analysis

Here is a collection of arrays, where array i has size 2^i. Each array is either empty or full. The issue of which arrays are full and which are empty is based on the binary representation of the number of items. For example, 11 elements are stored as the following array:

$$\begin{aligned} A0 &: [a_1] \\ A1 &: [a_2, a_3] \\ A2 &: empty \\ A3 &: [a_4, a_5, \cdots, a_{11}] \end{aligned}$$

To insert a new element (in the above example, say a_{12}), we first create an array of size 1 that just has this single element in it. We now look to see if A0 is empty. If so we make this be A0. If not (like in the above example) we merge our array with A0 to create a new array (which in the above case would now be $[a_1, a_{12}]$) and look to see if A1 is empty. If A1 is empty, we make this be A1. If not (like in the above example) we merge this with A1 to create a new array and check to see if A2 is empty, and so on. So, inserting a_{12} in the example above, we now have the new collection as shown below:

$$\begin{aligned} A0 &: empty \\ A1 &: empty \\ A2 &: [a_1, a_2, a_3, a_{12}] \\ A3 &: [a_4, a_5, \cdots, a_{11}] \end{aligned}$$

Cost model: To be clear about costs, let's say that creating the initial array of size 1 costs 1, and merging two arrays of size m costs 2m. So, the above insert had cost 1 + 2 + 4.

Please give the complexity of the insertion using amortized analysis.

Solution: 先考虑前几个操作，看看有没有规律可循，如表格1：

Table 1: 插入操作代价示例

操作序号:	代价的计算	代价	累积代价
1	1	1	1
2	1+2	3	4
3	1	1	5
4	1+2+4	7	12
5	1	1	13
6	1+2	3	16
7	1	1	17
8	1+2+4+8	15	32

数据结构初始为空，考虑n个插入操作构成的序列。记$k = \lceil \lg(n+1) \rceil$。

1. 方法一：使用记账方法。对于插入操作，定义其平摊代价为$\hat{c} = k$，其中1份是创建的代价(实际代价)，剩余的$k-1$份(accounting cost，存款)记在要插入的元素上。accounting cost用于以后可能的merge操作。对于n个插入操作构成的序列，任一元素最多参与$k-1$次merge操作。所以，对于整个数据结构而言，存款总是非负的，即$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$。所以，每个插入操作的平摊代价为$k = \Theta(\lg n)$。

2. 方法二：使用聚集分析。从一个空的数据结构开始，我们计算n个插入操作构成的序列的代价之和C_n，那么$\frac{C_n}{n}$即为单个插入操作的平摊代价。

   我们考虑某特定插入操作，该操作的代价与当前数据结构的状态有关，也就是与当前数据结构中已有元素的个数有关。假设当前数据结构已有n个元素，我们要分析插入第$n+1$个元素的代价。含有n个元素的数据结构的状态由数字n的二进制表示形式所决定：$\langle n_{k-1}, n_{k-1}, \ldots, n_0 \rangle$。假设该二进制表示最右边的0处于第$r$位，即$n_j = 1, 0 \leq j \leq r-1; n_r = 0$，更高位可以为0也可以为1。在这种状态下，插入第$n+1$个元素的代价为：

$$1 + \sum_{j=0}^{r-1} 2 \cdot 2^j = O(2^{r+1}).$$

   考虑n个插入操作构成的序列：$r = 0$的情况约有$\frac{n}{2}$次，$r = 1$的情况约有$\frac{n}{4}$次，推而广之，对于每个r，最多有$\lceil \frac{n}{2^r} \rceil$次。所以，$n$次插入操作的总代价为：

$$O(\sum_{r=0} \lceil \lg(n+1) \rceil (\lceil \frac{n}{2^r} \rceil) 2^{r+1}) = O(n \lg n).$$

所以，每个插入操作的平摊代价为$O(\lg n)$。

Problem 6: Decompostion of Graphs

给定有向图G，设计$\Theta(n+m)$的算法查找G中的奇圈(边数/顶点数为奇数的圈；给出一个即可)。如果G中不存在奇圈，则返回false。

(提示：考虑各个强连通分支。)

Solution:

1. 使用SCC算法求有向图G的各个强连通分支，时间复杂度为$\Theta(n+m)$。
2. 对于每一个强连通分支，运行DFS算法。在DFS遍历过程中，对连续访问(以discover访问为准)的节点交替染上黑白二色。下面证明某back edge，forward edge, cross edge (u,v)，u,v两个顶点着色相同当且仅当存在奇圈。

   证明如下：

   "$\Rightarrow$ "某back edge，forward edge, cross edge (u,v)，u,v两个顶点着色相同$\Rightarrow$存在奇圈。考虑DFS算法中的各种边：

   - back edge：易见。
   - forward edge $(v_1 \to v_2)$：表明从v_1到v_2 有两条路径，且长度的奇偶性不同。那么从v_2到v_1的路径不管奇偶性如何，总可以与v_1到v_2的其中一条路径构成奇圈。
   - cross edge $(u \to v)$：取u,v的最小公共祖先w(公共祖先必存在，因为根节点为u,v的公共祖先)。u,v同色，分情况考虑w的颜色：
     - w与u,v同色。则从w到v存在两条不同路径：$w \rightsquigarrow u \rightsquigarrow v$与$w \rightsquigarrow v$，其长度奇偶性不同。在该强连通分支中，$v$到$w$存在有向路径。不管该路径长度如何，总可以与$w$到$v$的其中一条路径构成奇圈。
     - w与u,v不同色。该情况与"w与u,v同色"类似。

   "$\Leftarrow$ " 存在奇圈$\Rightarrow$ 某back edge，forward edge, cross edge (u,v)，u,v两个顶点着色相同。否则，只可能构成偶圈。

另外，在网页http://algs4.cs.princeton.edu/42directed/中，提供了另外一种类似的方法："A digraph has an odd-length directed cycle if and only if one (or more) of its strong components is nonbipartite (when treated as an undirected graph)."

Problem 7: Minimum Spanning Tree

考虑如下问题：给定无向连通带权图G，所有边的权值各不相同。给定边$e = (v, w)$，请设计一个复杂度为$O(m + n)$的算法判断是否存在G的某个最小生成树包含边e。

有人对此设计了如下算法：

- 删除图G中所有权值大于等于$w(e)$的边，得到图G'。
- 使用BFS或者DFS测试v, w是否连通。
- e属于某个最小生成树当且仅当v, w不连通。

已知上述算法是正确的，请证明其正确性。

Solution:

- $\Rightarrow$ (cycle property) 要证明：e属于某个最小生成树T，则v, w不连通。

  反设v, w连通。$T-e$中v, w不连通，v, w分属两个不同的连通分支。记与v相连的连通分支为V，与w相连的连通分支为$W(V = V_G - W)$。但是在G'中v, w连通(反设)，则在G'中必存在某条边e'连接V, W，且$w(e') < w(e)$ (因为G'中的所有边的权值都小于$w(e)$)。考虑树$T - e + e'$，得到权值更小的生成树，矛盾。

- $\Leftarrow$ (cut property) 要证明：v, w不连通，则e属于某个最小生成树T。

  反设所有最小生成树都不包含e，考虑不包含e的最小生成树T。因为G'中v, w不连通，则v, w分处在两个不同的连通分支中。记与v仅通过权值小于$w(e)$的边相连的连通分支为V，其余所有顶点构成集合$S = V_G - V$。T是生成树，故其必含有某条边e'连接V与S，且$w(e') > w(e)$。考虑生成树$T' = T - e' + e$，其权值比$W(T)$更小。矛盾。

Problem 8: Graph Modeling

给定n张长方形纸张$P_1, P_2, \ldots, P_n$。纸张P_i的长度和宽度分别为l_i和w_i。今将任意纸张$A, B, C, \ldots$依次叠放成一堆，如果A纸张完全包含B纸张(允许旋转纸张)，而B纸张完全包含C纸张，(依次类推)，则称$A, B, C, \ldots$构成一个"纸张栈"，栈的大小即为栈中纸张的数目。

设计算法求出最大纸张栈。分析算法复杂度。

Solution:

1. 图论建模：建立有向图$G = (V, E)$。每张纸对应于图中一个顶点。如果纸张P_i可以完全包含纸张P_j，则在G中添加边$V_i \to V_j$。该图G为有向无环图，因为"完全包含关系"蕴含纸张面积的大小关系。如果给定的n张纸中有完全相同的纸，则可以先去重。
2. 求有向无环图G的最长路径。该最长路径即对应最大纸张栈。如果该最长路径中的每个顶点对应的纸张有重数，则将重数加上。
   (a) 方法一：添加一个顶点v，将其单向连接到其它所有顶点(权重为0)，然后使用DFS遍历(也是动态规划思想)求v到其它所有顶点的最长路径。算法复杂度为$\Theta(n+m)$。
   (b) 方法二：先对G作拓扑排序。按照拓扑序求每个顶点对其它所有顶点的最长路径(动态规划思想)。算法复杂度为$\Theta(n+m)$。

Problem 9: Greedy Algorithm: Storing Files on Tape

现将n个文件$f_1, f_2, \ldots, f_i, \ldots, f_n$顺序存储在磁带上。文件$f_i$的长度为$L_i$。存储在$k$位置的文件的访问代价为：$C(k) = \sum_{i=1}^{k} L_i$。假设每个文件的访问概率均相同，则$n$个文件的平均访问代价为：

$$E = \frac{1}{n}\sum_{k=1}^{n} C(k) = \frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{k} L_i.$$

1. 应如何安排文件的存储顺序使得平均访问代价最小？
2. 假设已知各文件f_i的访问频率为F_i(不一定相等)，应如何安排文件的存储顺序使得平均访问代价最小？
   (提示：考虑$\frac{L_i}{F_i}$)

Solution:

1. 以文件长度的递增顺序存储。证明如下：假设文件存储顺序存在逆序，那么必存在两个相邻的文件构成逆序，即$L_a > L_b$，但是f_a存储在f_b之前。交换这两个文件，平均访问代价将增加$L_b - L_a < 0$。
2. 已经证明如果文件访问频率都相等，应该按照文件长度的递增顺序存储；对应地，如果文件长度都相等，那么应该按照文件的访问频率的递减顺序存储。那么，我们猜测在文件访问频率不等，文件长度也不等的情况下，文件应该按照$\frac{L_i}{F_i}$的升序存储。其证明与第一个问题类似，过程如下：假设文件存储顺序存在逆序，那么必存在两个相邻的文件构成逆序，即$\frac{L_a}{F_a} > \frac{L_b}{F_b}$，但是$f_a$存储在$f_b$之前。交换这两个文件，平均访问代价将增加$L_bF_a - L_aF_b < 0$。

Problem 10: Dynamic Programming: Opening Restaurants

X先生打算沿某高速公路开设多家餐馆。餐馆位置有n种选择$x_1 < x_2 < \ldots < x_n$ (单位为公里)，且有两条约束：

- 在每个位置，最多只能开一家餐馆(开还是不开)。在x_i位置开设餐馆的利润为$p_i > 0$。
- 餐馆之间最少要相距$k > 0$公里。

优化目标：设计一个算法帮助X先生安排餐馆位置，使其利润最大化。

提示：子问题参数个数为1。

Solution: 子问题定义：$P(i)$表示仅考虑前i个位置时可获取的最大利润。原问题为$P(n)$。

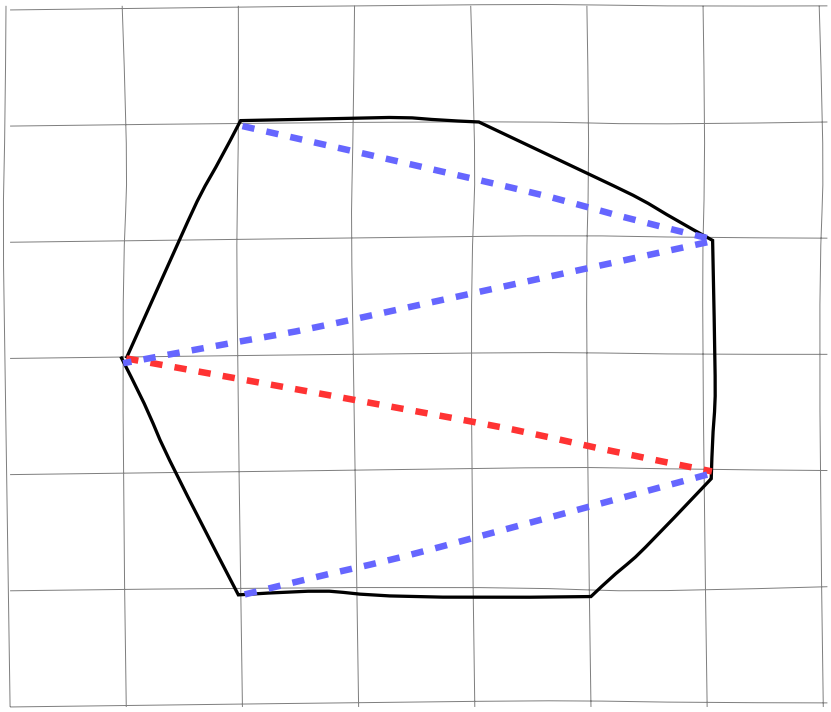
递归公式：考虑是否在x_i位置开设餐馆。

$$P(0) = 0; P(i) = \max_{x_i - x_j \geq k} (P(i-1), p_i + P(j)).$$

Problem 11: Dynamic Programming: Minimum Weight Triangulation

给定n个顶点$p_1, p_2, \ldots, p_i = (x_i, y_i), \ldots, p_n$构成的凸多边形$P$。为方便起见，$P$的$n$个顶点已按照顺时针方向标号$1, 2, \ldots, n$。凸多边形的任意一种“三角化”方案$T$由$P$的$n-3$条互不交叉(顶点除外)的对角线构成，如图1所示。“三角化”方案T的代价为$n-3$条对角线的长度之和。

Figure 1: 凸多边形三角化示例。



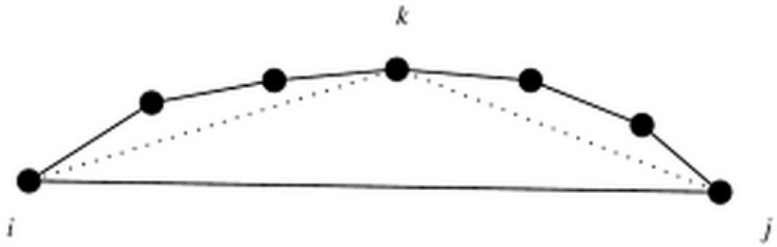
设计算法给出具有最小代价的“三角化”方案。

(提示：矩阵链相乘问题、最优二叉搜索树。)

Solution: 定义子问题：$T(i, j)$表示仅考虑凸多边形在顶点v_i到v_j部分的三角化的最小代价(不包括v_i与v_j距离d_{ij})。递归公式为(见图2)：

$$T(i, j) = \min_{k=i+1}^{j-1} (T(i, k) + T(k, j) + d_{ik} + d_{kj}).$$

Figure 2: 凸多边形三角化递归公式示意。



Problem 12: NP-Complete

无向图G是3-可着色(3-color)的，是指可以使用3种不同的颜色对图G的所有顶点进行着色，使得任意两个相邻顶点的颜色不同。

1. 给出一种从3-可着色问题(3-color)到可满足性问题(SAT)的多项式规约。
2. 如果已知3-可着色问题(3-color)是NP-Complete问题，证明4-可着色问题(4-color)也是NP-Complete问题。

Solution:

1. 3-color问题的输入为无向图G，SAT问题的输入为CNF。多项式规约R的目的就在于使用CNF逻辑公式来表达3-color问题中的约束。

   假设无向图$G = (V, E)$顶点数为n，边数为m。3种颜色记为R, G, B。按照如下步骤构造SAT实例F：

   - 变量集合为$\{ x_{vc} \mid 1 \leq v \leq n, c \in \{ R, G, B \} \}$，其中$x_{vc}$表示顶点$v$着色$c$。变量个数为$3n$。
   - 相邻顶点不能着相同颜色：

     $$\forall (i, j) \in E, c \in R, G, B : \neg(x_{ic} \wedge x_{jc}) \equiv (\neg x_{ic} \vee \neg xjc).$$

     共计$3m$个子句，每个子句包含2个文字。
   - 每个顶点都必须着色。

     $$\forall v \in V.(x_{vR} \vee x_{vG} \vee x_{vB}).$$

     共计n个子句，每个子句包含3个文字。

由上可知，该规约过程可在多项式时间内完成。

下面证明G是3-可着色的当且仅当F可满足。

- $\Rightarrow$ 如果在图G中顶点v被着色为c，则在F中将x_{vc}设置为true；
- $\Leftarrow$ 如果F可满足，则存在可满足性赋值A。对于任何一个v，至少有一个x_{vc}为true，在G中将v着色为c(如果对于v，有多个x_{vc}为true，则任选一个)。对于每一条边(i,j)，如果x_{ic} 为true，则x_{jc}必为false，所以i,j着色不同。

2. (a) 易证4-color $\in NP$。
   - 非确定性地为G中每个顶点着色(四种既定颜色之一)。
   - 检查每条边，如果某条边的两个顶点着色相同，则返回false；否则返回true。
   - 以上步骤可在多项式时间内完成。

   (b) 规约如下：将3-color问题的输入$G=(V,E)$转化为4-color问题的输入$G'=(V',E'), V'=V\cup\{v'\}, E'=E\cup\{(v',v)\mid v\in V\}$。

   (c) 证明G是3-可着色的当且仅当G'是4-可着色的。
   - “$\Rightarrow$:” G是3-可着色的$\Rightarrow$ G'是4-可着色的。
     为v'着第四种颜色即可。
   - “$\Leftarrow$:” G'是4-可着色的$\Rightarrow$ G是3-可着色的。
     在G'的四着色方案中，因为v'与V中所有顶点相连，故v'独占一种颜色。删除v'及其相应边，即得G的三着色方案。

   (d) 以上规约可在多项式时间内完成。

思考：由3-color问题是NP-Complete问题，可以按照以上规约证明4-color问题也是NP-Complete问题；同理，也可以证明5-color，6-color问题，k-color问题都是NP-Complete问题。但是，如果取$k=n$，该n-color问题是显然易解的。我们证明了“P=NP”吗？

可参见http://cs.stackexchange.com/q/7671/4911。