算法原理 习题参考答案

整理：薛建新 校正：吴琨
Jason_xjx@sjtu.edu.cn

可能有误，有异议请与我联系,谢谢。

1 chap 1 算法分析基本概念

1.5 a)数组元素按升序排列时，MODSELECTIONSORT执行的元素赋值次数最少，为0。
b)数组元素按降序排列时，MODSELECTIONSORT执行的元素赋值次数最多，为3n(n-1)/2。

1.9 就比较次数而言，INSERTIONSORT更有效。
就赋值次数而言，SECLECTIONSORT更有效。
两者的时间复杂性均为$O(n^2)$。 如果输入数组由大量的元素组成，主要比较两者的赋值次数，显然SECLECTIONSORT更有效。

1.13

$$\begin{array}{ccc} F & T & F \\ T & T & T \\ F & T & F \\ T & F & F \\ F & T & F \end{array}$$

1.16 (a) 数组元素按升序排列时，元素比较次数最少，为n-1次。
(b) 数组元素按降序排列时，元素比较次数最多， 为n(n-1)/2次。
(c) 数组元素按升序排列时，赋值次数最少，为0次。
(d) 数组元素按降序排列时，赋值次数最多，为3n(n-1)/2次。
(e) $O(n^2)$, $\Omega(n)$.
(f) 无法用Θ表示。不符合定义。

1.17 $f(n) = n^{n+\sin n}$, $g(n) = n^{n+\cos n}$

1.25 $O(1)$可以表示常数阶 或 低阶；但$\Theta(1)$只能表示常数阶。

1.31 (a) 第6步执行了$\lfloor \log n \rfloor(\lfloor \log n \rfloor + 1)(2\lfloor \log n \rfloor + 1)$.
(b) 用Θ表示更合适。因为$O(\lfloor \log^3 n \rfloor) = \Omega(\lfloor \log^3 n \rfloor)$
(c) 时间复杂性是$\Theta(\lfloor \log^3 n \rfloor)$

1.32 (a) 当n为2的幂时，第6步的最大次数是$n \log n$
(b) ????当n为3的幂时，第6步的最大次数是$n\lfloor \log n \rfloor$
(c) $O(n \log n)$
(d) $\Omega(n)$
(e) 用O表示更合适。根据Θ定义， 该算法不存在一个精确的时间复杂度。

1.33 (a) ????当n为2的幂时，第7步的最大次数是$n(n+1)\lfloor\log_3\rfloor n/2$

(b) 当n为3的幂时，第6步的最大次数是$n(n+1)\log_3 n/2$

(c) $O(n^2\log_3 n)$

(d) $\Omega(n^2)$

(e) 用O表示更合适。根据Θ定义， 该算法不存在一个精确的时间复杂度。

1.37 数组A[0,1,2...n]表示存放$a_0, a_1 ... a_n$

(a) $\Omega(n^2)$.

```
sum ← A[0]
for i ← 1 to n
    temp ← 1
    for j ← 1 to i
        temp ← temp * x
        j ← j+1
    end for
    sum ← sum + A[i] * temp
    i ← i+1
end for
return sum
```

(b) $O(n)$.

```
sum ← 0
for i ← n to 0
    sum ← sum * x + A[i]
    i ← i-1
end for
return sum
```

2 chap2 数学预备知识

2.10 1) 方法一：取$k=\lfloor\log n\rfloor$, 则$\lfloor\log n\rfloor \le \log n\lfloor\log n\rfloor+1$,即$2^k \le n < 2^{k+1}$,则对任意正整数$n$,有

$$2^k < n+1 \le 2^{k+1}$$

取对数得：

$$k \le \log n < k+1 \quad and \quad k < \log n + 1 \le k+1$$

故有$\lfloor\log n\rfloor = k, \quad and \lceil\log(n+1)\rceil = k+1$,
因而有 $\lfloor\log n\rfloor + 1 = \lceil\log(n+1)\rceil$. 2) 方法二，分 $n=2^k$, $\quad n+1=2^k$, $\quad 2^k < n < n+1 < 2^{k+1}$三种情况讨论即可。

2.21 a)$f(n) = 2n^2 - n$
b)$f(n) = \Theta(n^2)$

3 Chap5 归纳法

5.8 参考第1章1.14节和第5张93页第一段。

(a) n的数位长度为$\log n$，设$k = \log n$.
故根据某一位数字把数分到表中去的方法做了k遍，每一遍的代价为$\Theta(n)$, 故总的时间代价为$\Theta(kn) = \Theta(n \log n)$.

(b) 同(a)分析，时间代价为$\Theta(2n \log n)$= $\Theta(n \log n)$.

(c) 同(a)分析，时间代价为$\Theta(n^2 \log 2)$= $\Theta(n^2)$.

5.12 为分析插入排序的时间代价，设X_i为桶$B[i]$ (i = 1,2,...,k)中元素个数的随机变量。

对桶$B[i]$中的元素进行插入排序，期望时间为$E[O(X_i^2)] = O[E(X_i^2)]$.

由于每个元素落入桶$B[i]$的概率为$1/k$,且每个元素落入桶中的事件相互独立。故随机变量 X 服从二项分布B(n,p). 故$E(X_i) = np = n/k$, 方差$V(X_i) = np(1-p) = (k-1)n/k^2$.

故$E(X_i^2) = V(X_i) + E^2(X_i) = (n^2 + (k-1)n)/k^2$。

总的运行时间为$k \times (n^2 + (k-1)n)/k^2 = (n^2 + (k-1)n)/k$.

可见，若k十分接近n时，算法的时间复杂度为$\Theta(n)$.

5.14 bcdf为稳定算法。

5.33 算法如下：

输入：n个元素已排序的数组A[1...n]和整数x

输出：若A中存在两个数，其和恰好是x，则返回真； 否则返回假。

1. return sumX(1,n)

过程：sumX(s,t)

1. if s = t or s > t
2. return false;
3. else if A[s] + A[t] = x
4. return true;
5. else if A[s] + A[t] > x
6. return sumX(s,t-1);
7. else return sumX(s+1,t);

4 Chap6 分治法

6.52 算法可参考寻找中项（第k小元素）构造。

6.53 用反例说明（4个顶点就ok）。形状为普通的生成树。

6.54 设n $= 2^k, k > 0$.

将$n \times n$棋盘分割为4个$2^{k-1} \times 2^{k-1}$的子棋盘， 则残缺方格必位于4个子棋盘之一，其余三个无残缺方格。 用一个L型条块覆盖这三个无残缺小棋盘的结合处，这样， 3个子棋盘被L条块覆盖的方格就成为该棋盘上的残缺方格。 原问题也就转化为4个较小规模的棋盘覆盖问题。递归的使用这种分割， 直至期盼化简为2×2棋盘。

5 Chap7 动态规划

7.1 用数组A[1 $\cdots$ n]储存f(1)到f(n),输入为n，输出为A(n)。
A[1] $\leftarrow$ 1
A[2] $\leftarrow$ 1
for i=3 to n
A[i]=A[i-1]+A[i-2]
end for
分析：该算法的时间复杂性为$\Theta(n)$,它是指数复杂性算法， 该算法的运行时间取决于第4条语句的执行次数，其关于输入长度复杂性为 $\Theta(2^k), k = \lceil log(n+1) \rceil$(从数论角度分析)。

7.26 算法的时间复杂性为$\Theta(n\lfloor C/K \rfloor)$.反例略。

7.30 讲解.感谢周爱爱同学发现原有方案的问题。
(a)设$c[j], j = 0 \cdots y$,表示用用该硬币系统找钱j的最少硬币个数。
对于任何$0 \le j \le y$及$0 \le i \le n$,若$j - v_i > 0$, 则$c[j - v[i]]$所表示的找钱$j - v[i]$的最优序列，再加1枚面值为 $v[i]$的硬币是一种找钱j的方法，且所用的硬币个数为$c[j - v[i]] + 1$。
由此可得以下递规式：

$$c[j] = \min_{0 \le i \le n} \{c[j - v_i] + 1\}$$

其初始条件为：

$$c[0] = 0$$

算法自己写.
(b) 算法的时间复杂性为$\Theta(ny)$,空间复杂性为$O(y)$。
(c) y相当于背包容量，n种硬币相当于n种体积为$v[i]$、价值为1的物品。 但该题要求背包恰好装满，且总价值最小。

7.34 DAG的最长路径就是把权值变成负的求最短路径，有多项式解。
单源最短路径可用dijstra算法，可用$O(n^2)$时间解决。

6 chap9 图的遍历

9.3 参照例9.1。

9.5 更改算法9.1的$dfs(v)$过程即可。

9.7 主要是利用定义。 假设存在前向边，则由前向边定义可设w是v的后裔，且探索(v,w), w已被访问。由于G为无向图，故(v,w)即(w,v)，当深度优先搜索 树构建到w时，首先考虑由w出发的边，故(w,v)在(v,w)之前被探索， 则该边被记为回边，而非前向边。

假设存在横跨边，则存在v，w都不互为祖先，且w在v之前被访问，否则 (v,w)将成树边；由于访问w时，未访问到v，故(w,v)不存在，矛盾。 故无横跨边。

9.14 利用关节点的定义和性质。记v为深度优先生成树的根节点。
1) 如果v只有一个儿子，则移除v后，得到的子图为连通子图。 而删去关节点，则会生成不连通子图。矛盾。
2) 如果v有2个或2个以上儿子，则存在不同于v的顶点，其间的任何路径 都必须经过顶点v，符合关节点定义。

9.15 证明：充分性。v不是根，且v有一个儿子w， 使$\beta[w] \geqslant \alpha[v]$，则从$w$到$v$的祖先顶点的路径都必须经过$v$， 即$v$是关节点。
必要性。即证v不是根，且它是关节点，则v有一个儿子w，使 $\beta[w] \geqslant \alpha[v]$。反正法，假设对于$v$的所有儿子$w'$， $\beta[w'] < \alpha[v]$，则必有：v的后裔与v的祖先之间存在回边，与v是关节点矛盾。

9.17 重点讲解。分析：只需证明两种情况：

- 强连通的任何点必然出现在该算法得到的一个强连通支中；
- 该算法划分为强连通的点，必然都出现在强连通支中。

证明：

- 先证强连通的任何点必然出现在该算法得到的一个强连通支中。 假设a,b是强连通的点，则必有a $\to$ b 或者 b $\to$ a， 则不管选用任何顶点做起始点执行深度优先搜索遍历，a和b的predfn 和postdfn必然存在一种大小序。不失一般性，假设先遍历a，后遍历b， 则必有predfn(b)> predfn(a)和postdfn(b)<postdfn(a)，则在新图(边已换向)，必然会找到一条从a到b的路径， 所以a与b必然出现在由算法得到的一个强连通支中。
- 证明由该算法划分为强连通的点，必然都出现在同一强连通支中。 假设该算法将a，b两点划分为强连通的点，则只需证明它们在连通分支C中即可。
反正法。假设算法认为a,b同属于连通支C，但实际上a $\in$C1,b $\in$C2，即算法将两个原本不属于同一连通分支的顶点归类为强连通分支。 不失一般性，假设算法先遍历a，后遍历b。 考虑两种情况：
  * C1 $\to$ C2, 但是C2 $\not\to$ C1， 则必有predfn(b) > predfn(a) 和 postdfn(b) < postdfn(a)。 边反向后，有C1 $\leftarrow$ C2 和 C2 $\not\leftarrow$ C1 ,显然算法认为a,b不强连通。矛盾。
  * C1和C2之间无树边构成的路径。必然有 predfn(b) > predfn(a) 和 postdfn(b) > postdfn(a)。 显然算法认为a,b不强连通。矛盾。

7 chap10 NP完全问题

10.3 修改深度优先算法即可得。略。

10.5 根据题意，已经完成了一个不确定算法的猜测阶段， 我们只需完成验证阶段,确定性算法描述如下：

- 检查s所用的颜色是否超出着色的颜色种类
- 如果超出，则算法停下并回答no
- 否则，检查s中相邻顶点的颜色是否相同
- 如果存在两邻接点，颜色一样，则算法停下，并回答no
- 否则，算法停下并回答yes.

10.9 讲解证明：着重要理解输入规模的大小的概念。
设算法A是实现规约$\Pi_1 \propto_{poly} \Pi_2$的算法, 算法B是求解$\Pi_2$的算法。 由于A可在$O(n^j)$时间内完成， 而且算法执行的每一步最多只能输出c个字符(c>0)，算法A的输出规模不会超过cn^j。
由于对于输入规模为n的问题， 算法B能在$O(n^k)$时间解出。 所以对于输入规模为cn^j的问题， 算法B需要的求解时间是$(cn^j)^k = c^k n^{jk}$,其中，c^k为系数。 即用先规约再求解的方法所需的时间是$O(n^{jk})$。 证毕。

10.19 虽然背包问题能在时间$\Theta(nC)$内解决， 但它仍然是NP完全问题，两者并不矛盾。因为C同问题的输入大小并不成线性关系， 而是指数关系.原因在于问题的输入大小仅仅取决于表达输入所需的比特数$logC$. 同时，考虑到它的运行时间关于输入值（并非输入大小）是多项式的， 因此它是伪多项式算法。

10.22 证明：只需证明P $\subseteq$ NP 和NP $\subseteq$ P 即可。

- 由P和NP的定义可知，P $\subseteq$ NP。因为一个判定问题，如果可以用一个确定性算法在多项式时间内判定或解出， 就可以在多项式时间内容检查或验证其解。
- 证明NP $\subseteq$ P。因为Π是NP完全问题，所以NP中的任何问题Π'都可以多项式规约到 Π,另$\Pi \in$ P, 因此有 $\forall \Pi', \Pi' \in$ P(多项式规约具有传递性),即NP $\subseteq$ P.证毕。

8 chap13 回溯法

13.6 修改3着色问题的递规算法，或者4皇后算法即可。

13.12 判定子集和问题。题目要求判定是否存在，不是枚举全部。

13.17 使用分支限界法。

13.21 (i,j)对应(1,3) (2,4) (3,1) (4,2), bound = 13

9 chap14 随机算法

14.2 问题可以转换为随机序列的生成问题。本题中使用丢硬币的方法来生成随机数。（补充知识：通常，计算机不能生成真随机数，一般采用线性同余法生成伪随机序列）

14.3 “随机算法在最坏情况下以期望时间T(n)运行”定义如下： 假设对于随机算法而言，实例I是求解的最坏情况，也就是说，用该算法求解I 的代价是最大的。则用该算法反复求解I的平均时间，就是最坏情况下的期望运行时间。

14.4 “平均期望时间”定义如下：如果已知实例I的每个输入的分布情况 （即输入出现的概率），则用随机算法反复求解I的平均时间，就是平均期望运行时间。

14.8 分析：主要考察2个知识点。

- Monte Carlo算法和Las Vegas算法的性质。
- 期望值的概念。

解：Las Vegas总是给出正解，或者无解；Monte Carlo总是给出解，但总是出错。因此运行一次Monte Carlo就给出正确解的时间是 T(n)+T′(n)。由于Monte Carlo给出正确解的概率是p(n),所以有期望值的定义（Definition 14.1, Page 236）知,其期望值为1/p(n),又该算法的期望运行时间与期望值成正比， 所以Monte carlo总能在(T(n)+T′(n))/p(n)内给出正确解，即Las Vegas的最大运行时间。

10 chap15 近似算法

15.6 算法有误。u_s应为值最大的项，而不是size最大的项。
分两步分析。以下分析有误，待日后补上。思路上课时已介绍。

- 分析取得最优解OPT时的size，设为C',显然有：$C' > C/2$
  否则，如果$C' < C/2$, 则必然存在u_i, $s_i < C/2$. 必然有$C' + s_i < C$,所以最优解为$OPT + v_i$.Contradiction.
-

由题意知，$OPT(I) = VAL$, and $A_k(I) \geq v_s$. 当$v_s > V$时，必然有$v_s > VAL/2$(否则必然$\exists v_j \leq v_s$, 使得$V \leftarrow V + v_j$). 故有$A_k(I) \geq v_s > C/2$,因此$R_{KNAPSACKGREEDY} = OPT(I)/A_k(I) < 2$。

15.10 反例说明。

15.12 反例说明。

15.27 证明：令项集合$U = \{u_1, ..., u_n\}$, 大小分别为$s_1, ..., s_n$，背包容量为C.
设X是实例I对应于最优解的项的集合，根据题意有$OPT(I) = \Sigma_{s_i \in X} s_i < C/2$.
由于X是最优解， 故有$\Sigma_{s_j \in U-X} s_j \leq \Sigma_{s_i \in X} s_i < C/2$.故有 $\Sigma_{i=1}^{n} s_i < C$,则直接可得最优解即为U.

11 chap16 网络流

16.4 举反例。

16.5 举反例。注：16.4和16.5两题其实只需证明一个否定结果， 另一个否定结果可根据最大流最小割定理（定理16.1，page 262）给出。

16.6 额外添加一个源点S_0,以S_0连接原图中的多个源点， 每条连接路径的长度为1，容量为$+\infty$，再对所得的新图运用已有算法， 即可解得多个源点的最大流。
多条边是指无自环但有重边的非简单图的情况。也就是说，有多条边同时连接2个顶点。 仔细考虑该怎么解决。

16.15 分析：明确题中给出的是无向含权图， 而网络流是针对有向图的一个概念， 因此如果要用网络流的方法来解决无向含权图的最小权重问题， 首先要解决如何将其转换为有向图，然后用有向图的相关算法进行求解。 因此，可算法可分两步骤进行：

- 将原图转换为新图，转换方式如下：
  - 所有与s关联的边 $\rightarrow$ 从s出发的有向边
  - 所有与t关联的边 $\rightarrow$ 指向t的有向边
  - 内部节点之间的边 $\rightarrow$ 两条方向相反、 权值与原图相同的有向边
- 利用现有求解最大流的算法即可求得新图的最大流， 即原图中分离s和t的最小权重的割。

12 chap17 匹配

17.1 证明：要证明必要性和充分性。
必要性。若$\exists S \subseteq X$， 有$| S |>| \Gamma(S) |$ and $\Gamma(S) \subseteq Y$, 则$\exists v \in S$无法被匹配。
充分性。$\forall S \subseteq X$, 若$| \Gamma(S) | \geq | S |$，则$\forall V \in S$, 有$| \Gamma(\{V\}) | \geq 1$ and 在$\Gamma(\{V\})$至少存在一个顶点与V匹配。

17.2 举反例。取S $= \{a,c,e,i\}$，$\Gamma(S)\{b,d,f\}$, 故$|S|>|\Gamma(S)|$,不存在完全匹配。

17.3 证明：

1) X中的顶点都被匹配。$\forall S \subseteq X$,分2种情况讨论。

- 若$|S| \leq k$,则必有$|\Gamma(S)| \geq k \geq |S|$
- 若$|S| \geq k$，$|\Gamma(S)| \geq |S|$（为什么？自己想）
- 可见，以上2种情况均满足Hall定理，即X中所有顶点均被匹配。

2)同理，Y中的顶点亦均被匹配。 综上，$X \cup Y$中的顶点均被匹配， 符合完全匹配的定义（P272）.

17.7 $n!$.

17.9 反证法。假设存在树T，有两个不同的完美匹配M_1和M_2， 则$M_1 \oplus M_2$非空， 且子图$T[M_1 \oplus M_2]$中， 每个顶点的度都是2（为什么？）， 因此$T[M_1 \oplus M_2]$包含圈（为什么？）， 从而树T中包含圈，与树的定义矛盾。得证。

算法思路：从叶子节点开始匹配。