# 第 5,6,7 次作业答案

## 5.1

假定一个数据文件由 8 位字符组成，其中所有 256 个字符出现的频率大致相同：最高的频率也低于最低频率的 2 倍。证明：在此情况下，赫夫曼编码并不比 8 位固定长度编码更有效。

## 5.1 答案

对于任意 2 个字符，它们的频率之和超过任何其他字符的频率，因此最初人类编码制作了 128 棵小树，每棵树有 2 片叶子。 在下一阶段，没有内部节点的标签是其他任何节点的两倍以上，因此我们处于与以前相同的设置中。 以这种方式继续，人类编码构建了一个高度lg(256) = 8 的完整二叉树，它并不比普通的 8 位长度编码更有效。

## 5.2

令 $S$ 是一个有限集， $S_1, S_2, ..., S_k$ 是 $S$ 的一个划分，这些集合都是非空且不相交的。定义结构 $(S, \mathcal{I})$ 满足条件 $\mathcal{I} = \{A : |A \cap S_i| \leq 1, i = 1, ..., k\}$。证明：$(S, \mathcal{I})$ 是一个拟阵。也就是说，与划分中所有子集都最多有一个共同元素的集合 $A$ 组成的集合构成了拟阵的独立集。

## 5.2 答案

设 $\mathcal{M} = (S, \mathcal{I})$

S 是一个有限集。

假设 $X \subset Y$ 并且 $Y \in \mathcal{I}$。那么对于任意的 $i$ 有 $(X \cap S_i) \subset (Y \cap S_i)$，因此对于任意的 $1 \leq i \leq k$ 有 $|X \cap S_j| \leq |Y \cap S_i| = 1$。 因此 $\mathcal{I}$ 是遗传的。 ($\mathcal{I}$ 是 $S$ 的子集的一个非空族，这些子集称为 $S$ 的独立子集，使得如果 $B \in \mathcal{I}$ 且 $A \subseteq B$,则 $A \in \mathcal{I}$) 。如果满足此性质，则称之为遗传的。)

现在令 $A, B \in \mathcal{I}$ 满足 $|A| \geq |B| + 1$。那么一定存在 $j$，使得$|A \cap S_j| = 1$ 但 $|B \cap S_j| = 0$。令 $a = A \cap S_j$。有 $a \notin B$ 且 $|(B \cup \{a\}) \cap S_j| = 1$. 由于对所有的 $i \neq j$ 有 $|(B \cup a_i) \cap S_i| = |B \cap S_i| \leq 1$，一定有 $B \cup \{a\} \in \mathcal{I}$，$\mathcal{M}$ 满足交换性质。 (若 $A \in \mathcal{I}, B \in \mathcal{I}$ 且 $|A| < |B|$，那么存在某个元素 $x \in B - A$,使得 $A \cup x \in \mathcal{I}$，则称 $M$ 满足交换性质) 。

因此 $\mathcal{M}$ 是一个拟阵。

# 5.3

$A = a_1, ..., a_n$ 表示一个正整数集合。$A$ 中的元素之和为 $N$。设计一个 $O(n \cdot N)$ 的算法来确定是否存在一个 $A$ 的子集 $B$，使得 $\sum_{a_i \in B} a_i = \sum_{a_i \in A-B} a_i$

# 5.3 答案

我们假定 $N$ 是偶数；因为如果 $N$ 是奇数，则一定不存在满足要求的子集 $B$。

我们接下来确定是否存在 $A$ 的一个子集，它里面的元素相加恰好为 $\frac{N}{2}$。

我们定义

$$\begin{aligned} B[i, j] &= \textbf{true}, \text{ if some subset of } \{a_1, ..., a_i\} \text{ adds up exactly to } j \\ &= \textbf{false}, \text{ otherwise} \end{aligned}$$

如果 $B[n, \frac{N}{2}]$ 是 $\textbf{true}$，则存在一个满足要求的子集。

我们填写条目 $B[i, j]$ 时，要决定是选择 $a_i$ 还是不选择 $a_i$，我们可以将这一点总结为：

$$B[i, j] = B[i-1, j-a_i] \vee B[i-1, j]$$

对于表格中的每一个位置能够在 $O(1)$ 的时间内计算。因为表格一共有 $n \cdot \frac{N}{2}$ 个位置，因此表格可以用 $O(n \cdot N)$ 的时间填满。

# 6.1

假定我们对一个数据结构执行一个由 $n$ 个操作组成的操作序列，当 $i$ 严格为 2 的幂时，第 $i$ 个操作的代价为 $i$，否则代价为 1.

（1）使用聚合分析确定每个操作的摊还代价。

（2）使用核算法确定每个操作的摊还代价。

（3）使用势能法确定每个操作的摊还代价。

# 6.1 答案

聚合分析，我们证明对所有 $n$，一个 $n$ 个操作的序列最坏情况下花费的总时间为 $T(n)$, 因此在最坏情况下每个操作的平均代价或摊还代价为 $T(n)/n$。

（1）聚合分析，我们证明对所有 $n$，一个 $n$ 个操作的序列最坏情况下花费的总时间为 $T(n)$, 因此在最坏情况下每个操作的平均代价或摊还代价为 $T(n)/n$。

第 $i$ 个操作的代价为

$$a_i = \begin{cases} i, & \text{如果 } i \text{ 是 2 的幂} \\ 1, & \text{else} \end{cases}$$

$n$ 个操作序列的总代价为 $\sum_{i=1}^{n} a_i$

假设 $k = \lceil \log n \rceil, m = 2^k$，可以判断 $m \geq n \geq 2^{k-1}$，则有

$$\begin{aligned} \sum_{i=1}^{n} a_i \leq \sum_{i=1}^{m} a_i &= m - k + \sum_{t=1}^{k} 2^t \\ &= m - k + 2^{k+1} - 2 \leq m + 2^{k+1} \\ &= 3 \times 2^k \end{aligned}$$

$$(\sum_{i=1}^{n} a_i)/n \leq (3 \times 2^k)/2^{k-1} = 6$$

（2）核算法：进行摊还分析时，我们对不同操作赋予不同费用，赋予某些操作的费用可能多于或少于其实际代价。

假定每个操作支付的摊还代价为 $\hat{c}_i = 6$，实际代价 $c_i = a_i$，则由上题知

$$\sum_{i=1}^{n} c_i \leq 6n = \sum_{i=1}^{n} \hat{c}_i$$

（3）势能法摊还分析并不将预付代价表示为数据结构中的特定对象的信用，而是表示为势能，将势能释放即可用来支付未来操作的代价。

定义 $q = \lfloor \log n \rfloor$，可得 $n \geq 2^q$。定义势能函数如下

$$\Phi(D_n) = 2 \times (n - 2^q)$$

分两类情况讨论。当 $n = 2^q$

$$\begin{aligned} \hat{c}_i &= c_i + \Phi(D_n) - \Phi(D_{n-1}) \\ &= n + 0 - 2 \times (n - 1 - 2^{q-1}) \\ &= n - 2 \times (n - 1 - n/2) = 2 \end{aligned}$$

当 $n > 2^q$

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_n) - \Phi(D_{n-1}) \\ &= 1 + 2 = 3\end{aligned}$$

# 6.2

V.Pan发现一种方法，可以用 $132464$ 次乘法操作完成 $68 \times 68$ 的矩阵相乘，发现另一种方法，可以用 $143640$ 次乘法操作完成 $70 \times 70$ 的矩阵相乘，还发现一种方法，可以用 $155424$ 次乘法操作完成 $72 \times 72$ 的矩阵乘法。当用于矩阵乘法的分治算法时，上述哪种方法会得到最佳的渐近运行时间？与 Strassen 算法相比，性能如何？

# 6.2 答案

$\log_{68} 132464 = 2.795128487$
$\log_{70} 143640 = 2.79512269$
$\log_{72} 155424 = 2.795147391$
$\log_2 7 = 2.807354922$

三个方法都要比 Strassen 方法好，其中效果最好的是第二种方法

# 6.3

我们可以将一维离散傅里叶变换(DFT)推广到$d$维上。这时输入是一个$d$维的数组$A = (a_{j_1,j_2,...,j_d})$，维数分别为$n_1, n_2, ..., n_d$，其中$n_1 n_2 ... n_d = n$。定义$d$维离散傅里叶变换如下：

$$y_{k_1,k_2,...,k_d} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} ... \sum_{j_d=0}^{n_d-1} a_{j_1,j_2,...,j_d} \omega_{n_1}^{j_1 k_1} \omega_{n_2}^{j_2 k_2} ... \omega_{n_d}^{j_d k_d}$$

其中$0 \le k_1 < n_1, 0 \le k_2 < n_2, ..., 0 \le k_d < n_d$。

a.证明：我们可以依次在每个维度上计算一维的DFT来计算一个$d$维的DFT。也就是说，首先沿着第1维计算$n/n_1$个独立的一维DFT。然后，把沿着第1维的DFT结果作为输入，我们计算沿着第2维的$n/n_2$个独立的一维DFT。利用这个结果作为输入，我们计算沿着第三维的$n/n_3$个独立的一维DFT，如此下去，直到第d维。

b.证明：维度的次序并无影响，于是可以通过在d个维度的任意顺序中计算一维DFT来计算一个d维的DFT。

c.证明：如果采用计算快速傅里叶变换计算每个一维的DFT，那么计算一个d维的DFT的总时间是$O(n \lg n)$，与$d$无关。

## 6.3 答案

a.

$$
\begin{aligned}
y_{k_1,k_2,\ldots,k_d} &= \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \cdots \sum_{j_d=0}^{n_d-1} a_{j_1,j_2,\ldots,j_d} w_{n_1}^{j_1 k_1} w_{n_2}^{j_2 k_2} \cdots w_{n_d}^{j_d k_d} \\
&= \sum_{j_d=0}^{n_d-1} \cdots \sum_{j_2=0}^{n_2-1} \left( \sum_{j_1=0}^{n_1-1} a_{j_1,j_2,\ldots,j_d} w_{n_1}^{j_1 k_1} \right) w_{n_2}^{j_2 k_2} \cdots w_{n_d}^{j_d k_d} \\
&= \sum_{j_d=0}^{n_d-1} \cdots \left( \sum_{j_2=0}^{n_2-1} \left( \sum_{j_1=0}^{n_1-1} a_{j_1,j_2,\ldots,j_d} w_{n_1}^{j_1 k_1} \right) w_{n_2}^{j_2 k_2} \right) \cdots w_{n_d}^{j_d k_d} \\
&= \cdots
\end{aligned}
$$

$\left(\sum_{j_1=0}^{n_1-1} a_{j_1,j_2,\ldots,j_d} w_{n_1}^{j_1 k_1}\right)$ 是第 1 维度的计算结果（注意这里的结果不是一个数字）
$\left(\sum_{j_2=0}^{n_2-1} \left(\sum_{j_1=0}^{n_1-1} a_{j_1,j_2,\ldots,j_d} w_{n_1}^{j_1 k_1}\right)\right.$ 是第二维度的计算结果
$\cdots$

b. 按照上面的公式，更换求解顺序实际上就是更换求和顺序，对结果不会产生影响

c. 对第一维度求解，第 $i$ 维度的长度为 $n_i$，第 $i$ 维度数组的个数为 $\frac{n}{n_i}$，总时间复杂度为

$$
O(n_i \log n_i \times \frac{n}{n_i}) = O(n \log n_1)
$$

类似的，所有维度的时间复杂度之和为

$$
O(\sum_{i=1}^{d} n \log n_i) = O(n \log n)
$$

# 7.1

如果将输入的图用邻接矩阵来表示，并修改算法来应对此种形式的输入，请问 BFS的运行时间将是多少?

## 7.1 答案

如果我们使用邻接矩阵，当每个顶点出队时，我们需要检查所有顶点来确定是否与这一顶点相邻。 这使得处理每一个顶点的时间复杂度变为 $O(V)$。在连通图中，我们会将图中的每个顶点排入队列，因此最坏情况的运行时间变为 $O(V^2)$。

# 7.2

对于有向图 $G = (V, E)$ 来说，如果 $u \rightsquigarrow v$ 意味着图 $G$ 至多包含一条从 $u$ 到 $v$ 的简单路径，则图 $G$ 是单连通图。请给出一个有效算法来判断一个有向图是否是单连通图。

## 7.2 答案

算法思想：以图中每一个结点为根结点进行 DFS 染色，对于第一次访问的结点染为灰色，退出的结点染为黑色，如果在进行深度优先遍历时出现一个正在访问的结点的子节点为黑色结点，说明存在两条简单路径的起止点相同。当以一个结点为根结点进行 DFS 后未出现问题，将所有结点重置为白色，更换根结点重复上述过程。

对图进行一次深度优先搜索的时间复杂度为 $O(|E| + |V|)$，故上述算法的时间复杂度为 $O(|V|(|E| + |V|))$

# 7.3

假定图中的边权重全部为整数，且在范围 $1 \sim |V|$ 内，在此情况下， Kruskal 算法最快能多快？如果变得权重取值范围在 $1$ 到某个常数 $W$ 之间呢?

基本思想:按照权值从小到大的顺序选择n-1条边,并保证这n-1条边不构成回路。

## 7.3 答案

```
MST-KRUSKAL(G, w)
1: A = ∅
2: for each vertex v ∈ G.V do
3:     MAKE-SET(v)        // O(V) MAKE-SET
4: sort the edges of G.E into nondecreasing order by weight w
   // O(E log E)
5: for each edge (u, v) ∈ G.E, taking in nondecreasing order by
   weight w, do
6:     if FIND-SET(u) ≠ FIND-SET(v) then
7:         A = A ∪ {(u, v)}
8:         UNION(u, v)        //totally O(E) FIND-SET and UNION
9: return A
```

边权重是 $1$ 到 $|V|$ 范围内的整数，我们可以通过使用计数排序在线性时间内按权重对边进行排序，从而使 Kruskal 算法在 $O(E\alpha(V))$ 时间内运行。 如果边权重是由常数限定的整数，可以采用相同的方法，因为运行时算法需要决定边是否加入不相交的森林，而这与边的权重无关。

# 7.4

假定图中的边权重全部为整数，且在范围 $1 \sim |V|$ 内，在此情况下， Prim 算法最快能多快？如果变得权重取值范围在 $1$ 到某个常数 $W$ 之间呢?

## 7.4 答案

先选择一个顶点作为树的根节点，把这个根节点当成一棵树，选择图中距离这棵树最近但是没有被树收录的一个顶点，把他收录在树中，并且保证不构成回路，按照这样的方法，把所有的图的顶点一一收录进树中。

```
MST-PRIM(G, w, r)
 1: for each vertex u ∈ G, V do
 2:     u.key = ∞;        //u.key stores the minimum weight of any
    edge connecting v to a vertex in the current tree
 3:     u.π = NIL
 4: r.key = 0
 5: Q = G.V        // Q contains nodes not yet joining the tree
 6: while Q ≠ ∅ do
 7:     u = EXTRACT-MIN(Q)        //adding (u.π, u) to the tree
 8:     for each v ∈ G.Adj[u] do
 9:         if v ∈ Q and w(u, v) < v.key then        //updating keys
10:             v.π = u
11:             v.key = w(u, v)
```

算法导论第 20 章 vEB 树。 vEB 树支持优先队列操作以及一些其他操作，每个操作最坏情况运行时间为 $O(\log\log n)$。vEB 树支持在动态集合上运行时间为 $O(\log\log n)$ 的操作：SEARCH、INSERT、DELETE、MINIMUM、MAXIMUM、SUCCESSOR 和 PREDECESSOR。

使用 vEB 树来处理 Prim 算法中的优先队列这一问题的时间复杂度为 $O(E\log\log V)$ 以及 $O(E\log\log W)$