ch8_解答

T8.2

8.2 理解如下的随机算法，完成后面的问题。

输入：S={$s_1,s_2,\ldots,s_n \mid s_i\in \boldsymbol{R}$}

输出：$min(S, k)$—S 中第 k 小的元素

Random_Select(S,k)

1. 从 S 中随机选择一个元素 s；
2. S_1={s_i| $s_i\in S$, $s_i<s$}, S_2={s_i| $s_i\in S$, $s_i>s$};
3. IF $|S_1|=k-1$ THEN 返回 s;
4. ELSE IF $|S_1|>k$ THEN 返回 Random_Select(S_1, k);
5. ELSE 返回 Random_Select(S_2, k-$|S_1|$);

（1）该算法属于哪一类随机算法？

（2）证明：存在常数 $b<1$，使得算法递归过程中所考虑集合的大小的数学期望为 bn。

（3）证明：算法时间复杂度的数学期望为 $O(n)$。

（1）该算法运用随机性来优化期望运行时间。通过随机选择元素s，算法可以避免确定性算法中可能遇到的最坏情况。在Random_Select算法中，我们通过随机选择元素s，然后根据s将集合S分为两个子集S1和S2。算法始终返回正确的结果，即S中第k小的元素。

（2）证明：我们需要找到一个常数b<1，使得算法递归过程中所考虑集合的大小的数学期望为bn。

考虑递归调用Random_Select(S1, k)或Random_Select(S2, k–|S1|)。在最坏的情况下，我们将选择S中的最大或最小元素，这将导致递归调用的集合大小为n–1。但是，这种情况的概率相对较小。

在最佳情况下，我们将选择S中的中位数，这将导致递归调用的集合大小为n/2。

$$
\begin{aligned}
E(n) &\leq \sum_{k=1}^{n} E(x_k) \times \max(k-1, n-k) \\
&= \sum_{k=1}^{n} \frac{1}{n} \times \max(k-1, n-k) \\
&\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} k \\
&\leq \frac{n/2+n-1}{2} \\
&= \frac{3}{4}n - \frac{1}{2}
\end{aligned}
$$

综上所述：存在常数 $b<1$, 使得算法递归过程中所考虑集合的大小的数学期望为 bn。

（3）证明：算法时间复杂度的数学期望为O(n)。

我们已经证明了存在常数b<1，使得算法递归过程中所考虑集合的大小的数学期望为bn。

现在，我们来计算算法的时间复杂度。

我们可以将算法的期望运行时间表示为：

E(T(n)) = O(n) + b * E(T(n))

这里，O(n)表示将集合S分割成S1和S2的时间复杂度，E(T(n))表示递归调用的期望时间复杂度。

我们可以解这个递归式子得到：

E(T(n)) = O(n) / (1 – b)

E(T(n)) = O(n)

所以，算法时间复杂度的数学期望为O(n)。

T8.3

8.3 试设计一个随机算法判定输入的阶分别为 m,n,l 的多项式 $p(x),q(x)$和 $r(x)$是否满足 $p(x)\cdot q(x)=r(x)$。分析随机算法的时间复杂度和获得正确解的概率，判断该随机算法的类别。

我们可以使用以下随机算法来判断输入的阶分别为m, n, l的多项式p(x), q(x)和r(x)是否满足p(x) × q(x) = r(x)：

1. 随机选择一个整数x_0，例如在一个合适的范围内（如0到10^6）。
2. 计算p(x_0)、q(x_0)和r(x_0)的值。

3. 检查p(x_0) × q(x_0)是否等于r(x_0)： a. 如果相等，则认为p(x) × q(x)可能等于r(x)； b. 如果不相等，则p(x) × q(x)一定不等于r(x)。

这个算法属于Monte Carlo算法类别，因为它在每次运行时可能会得到不同的结果，但是正确答案的概率可以通过多次运行算法来提高。

分析随机算法的时间复杂度：

对于给定的x_0，计算p(x_0)、q(x_0)和r(x_0)需要O(m)、O(n)和O(l)的时间。检查它们的乘积是否相等需要O(1)的时间。因此，总的时间复杂度为O(m + n + l)。

分析获得正确解的概率：

如果p(x) × q(x)确实等于r(x)，那么对于任意x，p(x) × q(x)都等于r(x)。所以，只要我们找到一个x_0使得p(x_0) × q(x_0) = r(x_0)，我们就可以认为答案是正确的。然而，如果p(x) × q(x)不等于r(x)，那么至少有一个x_0使得p(x_0) × q(x_0) ≠ r(x_0)。选择一个这样的x_0的概率取决于多项式系数和x_0的范围。

通过多次运行算法，我们可以提高找到这样一个x_0的概率，从而提高算法给出正确答案的概率。例如，如果我们运行算法k次，并且在每次运行中都找到一个使得p(x) × q(x) = r(x)的x_0，那么我们可以认为答案是正确的。然而，如果在任何一次运行中找到一个使得p(x) × q(x) ≠ r(x)的x_0，我们可以立即得出答案是错误的。通过增加运行次数k，我们可以降低错误概率。

T8.4

> 8.4. 试设计一个随机算法判定输入的阶分别为 $p\times q, q\times r, p\times r$ 的矩阵 A,B 和 C 是否满足 $A\cdot B=C$。分析随机算法的时间复杂度和获得正确解的概率，判断该随机算法的类别。

我们可以使用以下随机算法来判断输入的阶分别为p×q、q×r和p×r的矩阵A、B和C是否满足A × B = C：

1. 随机选择一个q维向量v（元素范围可以是0到10^6之间的整数）。
2. 计算Av和Bv的值。
3. 计算Cv的值。
4. 检查是否满足A(Bv) = Cv： a. 如果满足，则认为A × B可能等于C； b. 如果不满足，则A × B一定不等于C。

这个算法属于Monte Carlo算法类别，因为它在每次运行时可能会得到不同的结果，但是正确答案的概率可以通过多次运行算法来提高。

分析随机算法的时间复杂度：

对于给定的向量v，计算Av、Bv和Cv需要O(pq)、O(qr)和O(pr)的时间。检查它们是否满足A(Bv) = Cv需要O(pr)的时间。因此，总的时间复杂度为O(pq + qr + 2pr)。

分析获得正确解的概率：

如果A × B确实等于C，那么对于任意向量v，都有A(Bv) = Cv。所以，只要我们找到一个v使得A(Bv) = Cv，我们就可以认为答案是正确的。然而，如果A × B不等于C，那么至少有一个v使得A(Bv) ≠ Cv。选择一个这样的v的概率取决于矩阵元素和v的范围。

通过多次运行算法，我们可以提高找到这样一个v的概率，从而提高算法给出正确答案的概率。例如，如果我们运行算法k次，并且在每次运行中都找到一个使得A(Bv) = Cv的v，那么我们可以认为答案是正确的。然而，如果在任何一次运行中找到一个使得A(Bv) ≠ Cv的v，我们可以立即得出答案是错误的。通过增加运行次数k，我们可以降低错误概率。

T8.5

8.5. 证明：最小割问题的如下随机算法输出最小割的概率为$\Omega(1/n^2)$。（ 提示：将该算法与 9.6 节的算法关联起来。）

输入： 一个多重无向连通图 $G=(V,E)$；

输出： G 的一个最小边割。

Random_Mincut

1. 为图 G 的任意边赋予一个随机独立的正权值；
2. 找出 G 的最小生成树 T；
3. 删除 T 中权值最大的一条边得到两棵树 T_1,T_2；
4. 令 T_1 的顶点集为 C，则 T_2 的顶点集为 $V\text{-}C$；
5. $cut=\{uv \mid uv\in E, u\in C, v\in V\text{-}C\}$
6. 输出 cut.

为了证明Random_Mincut算法输出最小割的概率为Ω(1/n^2)，我们可以通过将这个算法与9.6节的Karger's min-cut算法关联起来。

Karger's min-cut算法：

1. 当图G的顶点数量大于2时，重复以下步骤：

   a. 从图G中随机选择一条边(u, v)；

   b. 将顶点u和顶点v合并成一个新顶点，并删除所有连接这两个顶点的边；

   c. 更新所有指向u或v的边，使其指向新顶点；

2. 输出剩余的两个顶点之间的边集作为一个割。

Karger证明了这个算法输出最小割的概率至少为Ω(1/n^2)。

现在我们分析Random_Mincut算法：

1. 随机生成边的权值；

2. 找出最小生成树T；
3. 删除权值最大的一条边，将T分为两个子树T1和T2；
4. 输出割cut。

我们可以观察到Random_Mincut算法与Karger's min-cut算法有一定的相似性。在Karger的算法中，我们通过随机选择边来逐渐缩减图，而在Random_Mincut算法中，我们通过随机生成边的权值来随机选择最小生成树T，然后删除权值最大的一条边来得到一个割。在两种算法中，割的选择都是基于随机性的。

因此，我们可以推断Random_Mincut算法输出最小割的概率至少与Karger's min-cut算法的概率相当。所以，Random_Mincut算法输出最小割的概率为Ω(1/n^2)。

T8.7

8.7.设 $a_1,a_2,\ldots,a_n$ 是 n 个不同数构成的列表。如果 $i<j$ 且 $a_i>a_j$ 则称 a_i 和 a_j 是倒置的。冒泡排序算法的实质是不断交换列表中相邻的倒置元素，直到列表中没有倒置元素为止。假设冒泡排序算法的输入是一个随机排列，等可能地是 $n!$ 个排列中的任意一个。确定冒泡排序算法需要交换的倒置元素个数的数学期望。

用符号 X 表示列表 a1, a2, ..., an 中逆序对的个数，用符号 ai 表示列表中第 i

大的数，用符号 Xi 表示列表中的数 aj < ai 与 ai 构成逆序对的个数。

很明显：$X = X_2 + X_3 + \cdots + X_n$，所以可以得到：

$$\begin{aligned}
E[X] &= E[X_2 + X_3 + \cdots + X_n] \\
&= E[X_n] + E[X_{n-1}] + \cdots + E[X_2] \\
&= \frac{1}{n}\sum_{i=1}^{n}(n-i) + \frac{1}{n-1}\sum_{i=1}^{n-1}(n-1-i) + \cdots + \frac{1}{2}\sum_{i=1}^{2}(2-i) \\
&= n - \frac{n-1}{2} + n - \frac{n-2}{2} + \cdots + n - \frac{1}{2} \\
&= n^2 - \sum_{i=1}^{n-1}\frac{i}{2} \\
&= n^2 - \frac{n(n-1)}{4} \\
&= \frac{3}{4}n + \frac{1}{4}n
\end{aligned}$$

T8.8

8.8.有一个函数 $F:\{0,1,\ldots,n\text{-}1\}\to\{0,1,\ldots,m\text{-}1\}$，且 $F((x+y) \bmod n) = F(x)+F(y) \quad \bmod m$ 对$\forall x,y\in\{0,1,\ldots,n\text{-}1\}$成立。设 $F(x)$存储在一个数组中，数组下标表示自变量的值，数组元素的值表示函数值；由于某种意外，数组中 1/5 的函数值被恶意串改。试设计一个随机算法使其对$\forall z\in\{0,1,\ldots,n\text{-}1\}$算法能够以大于 1/2 的概率计算出正确的 $F(z)$。如果运行算法 3 次，你应该返回什么样的值，此时算法得到正确 $F(z)$的概率有什么变化？

为了设计一个随机算法来解决这个问题，我们可以利用F函数的线性性质。算法如下：

1. 对于给定的z∈{0,1,…,n−1}，从{0,1,…,n−1}均匀地随机选择一个数x，且y=(z+n−x)mod n
2. 计算F(x) + F(y) mod m。
3. 返回计算结果作为F(z)的估计值。

首先，我们分析算法是否能以大于1/2的概率计算出正确的F(z)。由于1/5的函数值被串改，4/5的函数值仍然是正确的。

F(z)正确：1.x,y都对;2. x,y都错且碰巧F(x)+F(y)凑对了)，P(F(z)正确)>4/5*4/5 = 16/25>1/2

当我们运行这个算法一次时，正确计算F(z)的概率大于1/2。如果我们运行算法3次，我们可以采用多数投票法，即选择三次运行结果中出现次数最多的值作为最终结果。这样，我们可以进一步提高算法得到正确F(z)的概率。

设P表示运行算法一次时计算正确F(z)的概率。那么，运行算法三次，至少有两次结果正确的概率为：

Q = C(3,2) * P^2 * (1−P) + C(3,3) * P^3

其中C(n,k)表示组合数。因为我们知道P > 1/2，可以证明Q > P。所以，运行算法三次，我们可以得到更高的正确F(z)的概率。