算法设计与分析 (2017 年秋季学期)

第四次作业参考答案

1 基站选址问题 (20 分)

假定有一条很长的直线形河流，在河岸上有 n 个房子。可以将该河流看作一条坐标轴，房子的位置以一维坐标的形式按严格递增的顺序给出 (单位为公里)。某通信公司希望在河岸的某些位置搭建手机基站，使得对任意一间房子都存在一个基站到它的距离小于等于 c 公里，c 为给定的一个常数。请给出一个有效的算法来最小化需要搭建的基站的数量，并证明算法的正确性。(注：时间复杂度为 $O(n)$ 的正确的算法可得满分)

解：

设第一个房子的坐标为 x，那么将第一个站点放在 $x+c$ 位置处，之后删除所有这个站点能够覆盖到的房子；重复这样做直到所有的房子都被删除为止。

算法正确性证明: 令 X 为上述算法所计算出的站点集合，Y 为最优情况下所选择的站点集合，若这两个集合不相等，那么考虑 X 和 Y 中第一对不同的站点，设 X 中的该站点坐标为 x,Y 中的该站点坐标为 y，我们的贪心策略可以保证，$x > y$，因此我们可以将 y 移动到 x 的位置，而此时 Y 集合仍可以覆盖所有的房子。重复这样做可以使 Y 变得和 X 相同。因此我们找出的站点集合中的元素个数是最少的。

2 最长递增子序列问题 (20 分)

请设计一个 $O(n^2)$ 的动态规划算法来求出一个包含 n 个元素的序列中的最长递增子序列。所谓递增子序列是指，从原序列中按顺序挑选出某些元素组成一个新序列，并且该新序列中的任意一个元素均大于该元素之前的所有元素。例如，输入的序列为 $< 5, 24, 8, 17, 12, 45 >$，那么该序列的最长递增子序列为 $< 5, 8, 12, 45 >$ 或 $< 5, 8, 17, 45 >$。请分析你的算法的正确性并验证其时间复杂度。

解：

令 $X =< x_1, \ldots, x_n >$ 为给定的包含 n 个元素的序列，我们需要找到序列 X 的最长递增子序列。

我们首先给出求解最长递增子序列长度的算法，之后再介绍如何找到该递增的上升子序列。

令 $X_i =< x_1, \ldots, x_i >$ 表示序列 X 的前 i 个元素，定义状态 $c[i]$ 表示以 x_i 为结尾的最长递增子序列的长度，显然整个序列的最长递增子序列的长度为 $\max\limits_{1 \le i \le n} c[i]$。

考虑 $c[i]$ 的更新过程，若存在某个 $x_r < x_i \square 1 \le r < i$，那么以 x_r 为结尾的最长递增子序列加上 x_i 就构成了一个新的最长递增子序列，因此我们要选择满足上述条件同时 $c[r]$ 最大的 r 来更新 $c[i]$。据此可以写出如下递归式：

$$c[i] = \begin{cases} 1 & i = 1 \\ 1 & x_r \ge x_i \forall 1 \le r < i \\ \max\limits_{1 \le r < i, x_r < x_i} c[r] + 1 & i > 1 \end{cases}$$

递归式的终止条件基于如下事实：以 x_i 结尾的仅包含一个数字的最长递增子序列就是它本身。

按照递增的顺序对每个 i 依次计算 $c[i]$ 的值，在计算完 c 数组后，其中的最大元素即是序列 X 的最长递增子序列的长度。

为了输出所求出的最长递增子序列,我们在计算 $c[i]$ 时,需要同时记录 $r[i] = \arg \max\limits_{1 \le r < i, x_r < x_i} c[r]$。令 $c[k] = \max\limits_{1 \le i \le n} c[i]$，那么 x_k 就是所求最长递增子序列的最后一个元素，之后我们依次找出

$x_r k, x_{r[r[k]]}$，将这些元素逆序输出即为原序列 X 的最长递增子序列。

时间复杂度分析：在计算 $c[i]$ 时，需要花费 $O(i)$ 的时间，因此，总的运行时间为 $O(\sum i) = O(n^2)$。之后需要 $O(n)$ 的时间来确定最长递增子序列的每个元素，因此，总的时间复杂度为 $O(n^2)$。

3 最长回文子序列问题 (20 分)

一个子序列被称为回文序列是指：该序列从右往左读和从左往右读的结果是一样的。例如，如下所示序列：

$$< A, C, G, T, G, T, C, A, A, A, A, T, C, G >$$

包含了许多回文子序列，如 $< A, C, G, C, A >$，$< A, A, A, A >$。请设计一个动态规划的算法来计算序列 $x[1..n]$ 的最长回文子序列。该算法的时间复杂度应为 $O(n^2)$。

解：

令 $L[i, j]$ 表示子串 $x[i, \ldots, j]$ 的最长回文子序列的长度，考虑子串 $x[i, \ldots, j]$ 的两端的元素，若 $x[i] = x[j]$，那么 $x[i]$ 和 $x[j]$ 必定为该子串的最长回文子序列的两端，只需求出子串 $x[i+1, \ldots, j-1]$ 的最长回文子序列，之后加上 $x[i]$ 和 $x[j]$ 就构成了原串的最长回文子序列；如果 $x[i] \neq x[j]$，那么我们仅需分别考虑子串 $x[i+1, \ldots, j]$ 和子串 $x[i, \ldots, j-1]$。当然，每个长度为 1 的序列都是回文序列。因此，我们可以写出如下递归式：

$$L[i][j] = \begin{cases} L[i+1][j-1] + 2 & x[i] = x[j] \\ \max\{L[i+1, j], L[i, j-1]\} & x[i] \neq x[j] \end{cases}$$

$$L[i, i] = 1 \forall i \in (1, \ldots, n)$$

$$L[i, i-1] = 0 \forall i \in (2, \ldots, n)$$

为了保证算法正确运行，我们可以按照子串长度从短到长的顺序来计算 L 数组，即先计算所有满足 $j - i = 1$ 的元素 $L[i, j]$，之后计算所有满足 $j - i = 2$ 的元素 $L[i, j]$，以此类推。为了输出最终的最长回文序列，我们还需要使用 $r[i, j]$ 来存储 $L[i, j]$ 的转移过程，注意 $L[i, j]$ 只有三种转移方式。

在处理完所有 $L[i, j]$ 后，我们可以从 $r[1, n]$ 开始，输出最长的回文子序列，输出最长回文子序列的算法见 Algorithm 1。

Algorithm 1 $PrintMove(L, r, i, j)$

```
if L[i, j] = 1 then
    print x[i];
    return ;
end if
if r[i, j] = 1 then
    print x[i];
    PrintMove(L, r, i + 1, j − 1);
    print x[j];
end if
if r[i, j] = 2 then
    PrintMove(L, r, i + 1, j);
end if
if r[i, j] = 3 then
    PrintMove(L, r, i, j − 1);
end if
```

4 餐厅选址问题 (20 分)

某公司希望在高速公路上设立一些餐厅。现有 n 个位置可选，距高速公路的起点分别为 $m_1, m_2, \ldots, m_n (m_1 < m_2 < \cdots < m_n)$。对每个位置 m_i，该公司如果在这个位置设立餐厅的话，期望盈利为 $p_i (p_i > 0)$。此外，公司设立的任意两个餐厅间的距离应大于等于 $k (k > 0)$，但设立

的餐厅个数没有限制。请设计一个动态规划的方法来帮助该公司选址，使得期望盈利和最大。请写出你的动态规划转移方程并给出伪代码，最后分析该算法的时间复杂度。

解：

令状态 Ti 表示仅考虑前 i 个位置可获得的最大收益，$R[i]$ 表示第 i 个位置有没有被选作餐厅，如果被选作餐厅则 $R[i]=1$ 否则 $r[i]=0$。

初始状态 $T[0]=0$，因为此时没有任何一个地点可以被选作餐厅，因此收益为 0。

若 $i>0$，则有如下两种选择方案：

1. 不在位置 i 设立餐厅，此时的收益和仅考虑前 $i-1$ 个位置的收益相同，即 $T[i]=T[i-1]$
2. 在位置 i 设立餐厅，此时我们需要寻找在 i 之前且到 i 的距离大于等于 k 的最后的一个位置，即 $c[i]=\max\limits_{0\le j<i, m_j<m_i-k} j$，这个位置之后均不能再设立餐厅，仅需考虑前 $c[i]$ 个站点的餐厅设立情况，那么有 $T[i]=T[c[i]]+p_i$

根据如上分析，可写出递归式：

$$T[i]=\begin{cases}0 & i=0\\ \max\{T[i-1],T[c[i]]+p_i\} & i>0\end{cases}$$

同时如果 $T[i]=T[i-1]$，那么令 $R[i]=0$，否则令 $R[i]=1$，$c[i]$ 则可以使用 $two-pointer$ 的方法在 $O(n)$ 的时间内预处理出来 (见 Algorithm 2)。

Algorithm 2 $ComputeCi(m,k)$

```
for i ← 1 to n do
    m'_i ← m_i − k;
end for
i ← 1;
j ← 1;
while i ≤ n do
    if m'_i < m_j then
        c_i ← j − 1;
        i ← i + 1;
    else
        j ← j + 1;
    end if
end while
```

求解最大收益的动态规划伪代码见 Algorithm 3。

Algorithm 3 $FindOptimalPos(m,p,c)$

```
T[0] ← 0;
for i ← 1 to n do
    if T[i − 1] > p_i + T[c[i]] then
        T[i] ← T[i − 1];
        R[i] ← 0;
    else
        T[i] ← p_i + T[c[i]];
        R[i] ← 1;
    end if
end for
return T[n]
```

在有了 R 数组后，我们就可以用类似前面两题的思路来输出最优方案下我们所选择的餐厅位置，这里不再赘述。

时间复杂度分析：预处理 c、动态规划和输出餐厅位置均可以在 $O(n)$ 的时间内完成，因此总的时间复杂度为 $O(n)$。

5 子图同构问题 (20 分)

子图同构问题是指，对给出的两个图 G_1 和 G_2，判断 G_1 是否是 G_2 的子图。请证明子图同构问题是 NP 完全的。(注：为了证明一个问题是 NP 完全的，首先需要证明该问题是 NP 的，其次需要将一个已知的 NP 完全问题在多项式时间规约到该问题上。)

解：

首先证明该问题为 NP 问题。要判断 G_1 是否是 G_2 的子图，可描述为是否能找到一个 G_1 中的节点到 G_2 中的节点的映射 M，使得对 G_1 中存在的每条边 (u,v)，在 G_2 中也存在 $(M(u), M(v))$ 这条边。若要判断给定的映射是否是该问题的解，最多仅需要 $O(m_1 \cdot m_2)$ 的时间内就可以完成判定 (m_1 为图 G_1 中的边数，m_2 为图 G_2 中的边数)。因此，子图同构问题属于 NP 问题。

接下来要将一个已知的 NP 完全问题在多项式时间规约到该问题上。这里使用团问题进行规约，即寻找一个多项式时间的函数，将给出的团问题的实例 (包含一个图 G 和一个整数 k) 转化为一个子图同构问题的实例 (包含两个图 G_1 和 G_2)，使得图 G 包含大小为 k 的团当且仅当 G_1 是 G_2 的子图。

转化方法：令 G_1 为一个节点个数为 k 的完全图，这需要花费 $O(k^2)$ 的时间，令 G_2 为和图 G 完全相同的图。这样，我们就把团问题多项式规约到了子图同构问题上。

正确性：若图 G 包含一个大小为 k 的团，由于 G_1 为一个大小为 k 的团，那么 $G_2 = G$ 也一定包含 G_1；反之，若 G_1 是 G_2 的子图，那么图 G 必定包含一个大小为 k 的团。