1. 基站选址问题

假设已知前x个房子需要的最小基站数为，如果需要设立新基站，则设在第一个当前所有基站所无法控制的房子的坐标+c的位置

状态转移方程：

2. 最长递增子序列问题

首先，利用排序算法对n个元素的序列做一次排序，其时间复杂度为至，将原数组与新数组利用基于动态规划的最长公共子序列算法求出LCS，该LCS即为原数组的最长递增子序列。由于利用动态规划的最长公共子序列算法的时间复杂度为，其中m为序列A的字符个数，n为序列B的字符个数，而由于当前求解的问题的两个序列的字符个数均为n个，故其时间复杂度为

Longest-Increasing-subsequence(X):

**Input**: string

**Output**: Longest increasing subsequence in the string

← quicksort();

← length();

Letandbe two new 2-dimensional arrays;

// Initialization

**for**←to**do**:

| ;

**end**

**for**←to**do**:

| ;

**End**

// Dynamic Programming

**for**←to**do**:

| **for**←to**do**:

| | **ifthen**

| | | ;

| | | ; // ”LU” indicates left up arrow

| | **end**

| | **else ifthen**

| | | ;

| | | ; // ”U” indicates up arrow

| | **end**

| | **else**

| | | ;

| | | ; // ”L” indicates left arrow

| | **end**

| **end**

**end**

**return**;

Print-LIS()

**Input**: Arraygenerated from Longest-Increasing-Subsequence, string, indexand.

**Output**: output the longest increasing subsequence of

**ifthen**

| **return**;

**end**

**ifthen**

| Print-LIS();

| print

**end**

**else** **ifthen**

| Print-LIS();

**end**

**else**

| Print-LIS();

**end**

3. 最长回文子序列问题

初始化：

状态转移方程：

public int Longest-Palindrome-Sub-Seq(String s) {

if(s == null || s.length() == 0)

return 0;

int n = s.length();

char[] str = s.toCharArray();

int[][] dp = new int[n][n];

// initialization

for(int i = 0; i < n; i++)

dp[i][i] = 1;

// dynamic programming

for(int j = 1; j < n; j++){

for(int i = j - 1; i >= 0; i--){

if(str[i] == str[j]){

if(i == j - 1)

dp[i][j] = 2;

else

dp[i][j] = dp[i + 1][j - 1] + 2;

}

else{

dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);

}

}

}

return dp[0][n-1];

}

4. 餐厅选址问题

状态转移方程：

表示选择第i个位置设立餐厅时，在前i个位置能够得到的总期望盈利

为在第i个位置设立餐厅可以的期望盈利

def get\_maximum\_profit(n, k, m, p):

maximum = 0

chosen\_loc = []

for i in range(n):

before\_max = 0

for j in range(i):

if a[i] - a[j] > k and before\_max < dp[j]:

before\_max = dp[j]

dp[i] = before\_max + p[i]

maximum = math.max(maximum, dp[i])

if maximum == dp[i]:

chosen\_loc.append(i)

return maximum, chosen\_loc