第 7 章　动态规划

1. 用动态规划解决数塔问题。

C/C++代码：

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_LEVELS100 // 假设数塔的最大层数为 100

//动态规划求解数塔问题
int solveNumberTower(int tower[MAX_LEVELS][MAX_LEVELS], int levels) {
    // 从倒数第二层开始向上计算
    for (int i = levels - 2; i >= 0; i--) {
        for (int j = 0; j <= i; j++) {
            // 选择下方或右下方较大的值累加
            tower[i][j] += (tower[i +1][j] > tower[i + 1][j + 1]) ? tower[i
+ 1][j] : tower[i + 1][j + 1];
        }
    }
    // 返回顶层的最大值
    return tower[0][0];
}

//打印最大路径
void printPath(int tower[MAX_LEVELS][MAX_LEVELS], int levels) {
    int j =0;
    printf("% d", tower[0][0] - (tower[1][j] > tower[1][j + 1] ? tower[1]
[j] : tower[1][j + 1]));
    for (int i = 1; i < levels; i++) {
        if (tower[i][j] > tower[i][j + 1]) {
            printf("->% d", tower[i][j] - (i + 1 < levels ? (tower[i + 1][j]
> tower[i + 1][j + 1] ? tower[i + 1][j] : tower[i + 1][j + 1]) : 0));
        }else {
            printf("->% d", tower[i][j + 1] - (i + 1 < levels ? (tower[i + 1]
[j + 1] > tower[i + 1][j + 2] ? tower[i + 1][j + 1] : tower[i + 1][j + 2]) : 0));
            j++;
        }
```

```
        }
        printf("\n");
    }

    int main() {
        int tower[MAX_LEVELS][MAX_LEVELS] = {0};
        int levels;
        // 输入数塔的层数
        scanf("% d", &levels);
        // 输入数塔的数据
        for (int i = 0; i < levels; i++) {
            for (int j = 0; j <= i; j++) {
                scanf("% d", &tower[i][j]);
            }
        }
        // 计算最大路径和
        int maxSum = solveNumberTower(tower, levels);
        // 输出结果
        printf("% d\n", maxSum);
        printPath(tower, levels);
        return 0;
    }
```

Python 代码：

```
    def solve_number_tower(tower, levels):
        # 从倒数第二层开始向上计算
        for i in range(levels - 2, -1, -1):
            for j in range(i + 1):
                # 选择下方或右下方较大的值累加
                tower[i][j] +=max(tower[i + 1][j], tower[i + 1][j + 1])
        # 返回顶层的最大值
        return tower[0][0]

    def print_path(tower, levels):
        j =0
        path = [tower[0][0] - max(tower[1][j], tower[1][j + 1])]
        for i in range(1, levels):
            if tower[i][j] > tower[i][j + 1]:
                path.append(tower[i][j] - (max(tower[i + 1][j], tower[i + 1][j
+ 1]) if i + 1 < levels else 0))
```

```
            else:
                path.append(tower[i][j +1] - (max(tower[i + 1][j + 1], tower[i
+ 1][j + 2]) if i + 1 < levels else 0))
                j +=1
        print("->".join(map(str, path)))

    def main():
        # 输入数塔的层数
        levels =int(input())
        # 输入数塔的数据
        tower = []
        for i in range(levels):
            row =list(map(int, input().split()))
            tower.append(row)

        # 计算最大路径和
        max_sum = solve_number_tower(tower, levels)

        # 输出结果
        print(max_sum)
        print_path(tower, levels)

    if __name__ == "__main__":
        main()
```

2. 求字符串序列“ABCDBAB”和“BDCABA”的最长公共子序列。

C/C++代码：

```
#include <stdio.h>
#include <string.h>

#define MAX_LEN1000

//定义一个二维数组来保存动态规划表
int dp[MAX_LEN +1][MAX_LEN + 1];

//获取最长公共子序列的函数
void printLCS(char * s1, char * s2, int m, int n) {
    int index = dp[m][n];
    char lcs[index +1];
```

```
    lcs[index] ='\0';   // 最后一位设为空字符，作为字符串的终止符

    // 从 dp 表中回溯，找到 LCS
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (s1[i - 1] == s2[j - 1]) {
            lcs[index -1] = s1[i - 1];
            i--;
            j--;
            index--;
        }else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        }else {
            j--;
        }
    }

    // 打印最长公共子序列
    printf("%s\n", lcs);
}

int main() {
    char s1[MAX_LEN], s2[MAX_LEN];

    // 输入两个字符串
    scanf("%s", s1);
    scanf("%s", s2);

    int m = strlen(s1);
    int n = strlen(s2);
    // 初始化 dp 数组
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) {
                dp[i][j] =0;
            }else if (s1[i - 1] == s2[j - 1]) {
                dp[i][j] = dp[i -1][j - 1] + 1;
            }else {
                dp[i][j] = (dp[i -1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i]
```

```
[j - 1];
                }
            }
        }
        // 输出最长公共子序列
        printLCS(s1, s2, m, n);
        return 0;
    }
```

Python 代码：

```
def printLCS(s1, s2, m, n):
    # 构建 dp 数组
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # 填充 dp 数组
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if s1[i - 1] == s2[j - 1]:
                dp[i][j] = dp[i -1][j - 1] + 1
            else:
                dp[i][j] =max(dp[i - 1][j], dp[i][j - 1])

    # 回溯，找到最长公共子序列
    index = dp[m][n]
    lcs = [' '] * (index)
    i, j = m, n

    while i > 0 and j > 0:
        if s1[i - 1] == s2[j - 1]:
            lcs[index -1] = s1[i - 1]
            i -=1
            j -=1
            index -=1
        elif dp[i - 1][j] > dp[i][j - 1]:
            i -=1
        else:
            j -=1

    # 输出最长公共子序列
    print(' '.join(lcs))
```

```
if __name__ == "__main__":
    # 输入两个字符串
    s1 =input()
    s2 =input()
    m =len(s1)
    n =len(s2)
    # 获取并输出最长公共子序列
    printLCS(s1, s2, m, n)
```

3. 给定由 n 个整数(可能为负数)组成的序列 a_1, a_2, …, a_n, 求该序列的最大子段和。当所有整数均为负数, 定义其最大子段和为 0。

C/C++代码:

```
#include <stdio.h>
int MaxSubSeq(int n, int a[], int b[], int max){
    for (int i = 0; i < n; i++){
        if (i == 0){
            b[i] = a[i];
            max = b[i];
        }
        else{
            if (b[i - 1] <= 0)
                b[i] = a[i];
            else
                b[i] = b[i -1] + a[i];
            if (b[i] > max)
                max = b[i];
        }
    }
    return max;
}
int main(){
    int n;
    scanf("% d", &n);
    int a[1000];
    for (int i = 0; i < n; i++)
        scanf("% d", &a[i]);
    int b[100], max = 0;
    max = MaxSubSeq(n, a, b, max);
    printf("% d", max);
```

```
}
```

Python 代码：

```
def MaxSubSeq(n, a):
    b = [0] *  n
    max_sum = a[0]
    b[0] = a[0]
    for i in range(1, n):
        if b[i - 1] <= 0:
            b[i] = a[i]
        else:
            b[i] = b[i -1] + a[i]
        if b[i] > max_sum:
            max_sum = b[i]
    return max_sum

def main():
    n =int(input())  # 读取数组的长度
    a =list(map(int, input().split()))  # 读取数组元素
    result = MaxSubSeq(n, a)
    print(result)

if __name__ == "__main__":
    main()
```

4. 给定一个长度为 n 的数组，找出一个最长的单调递增子序列(不一定连续，但是顺序不能乱)。例如：给定一个长度为 7 的数组 A{5, 6, 7, 1, 2, 8, 9}，则其最长的单调递增子序列为{5, 6, 7, 8, 9}，长度为 5。

C/C++代码：

```
#include <stdio.h>
#include <stdlib.h>

//动态规划求解最长递增子序列
int lengthOfLIS(int * nums, int n) {
    if (n == 0) return 0;

    // dp[i] 表示以 nums[i] 结尾的最长递增子序列的长度
    int * dp = (int * )malloc(n *  sizeof(int));
    for (int i = 0; i < n; i++) {
        dp[i] =1; // 初始化为 1, 因为每个元素本身就是一个长度为 1 的子序列
    }
```

```
    int maxLen =1; // 记录全局最大值

    // 动态规划求解
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (nums[i] > nums[j]) {
                // 如果 nums[i] 可以接在 nums[j] 后面，更新 dp[i]
                if (dp[j] + 1 > dp[i]) {
                    dp[i] = dp[j] +1;
                }
            }
        }
        // 更新全局最大值
        if (dp[i] > maxLen) {
            maxLen = dp[i];
        }
    }
    free(dp); // 释放动态分配的内存
    return maxLen;
}

int main() {
    int n;
    scanf("% d", &n); // 输入数组长度
    int * nums = (int * )malloc(n *  sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("% d", &nums[i]); // 输入数组元素
    }
    // 计算最长递增子序列的长度
    int result = lengthOfLIS(nums, n);
    printf("% d\n", result); // 输出结果
    free(nums); // 释放动态分配的内存
    return 0;
}
```

Python 代码：

```
def length_of_LIS(nums):
    if not nums:
        return 0
```

```
    n =len(nums)
    dp = [1] *  n   # dp[i] 表示以 nums[i] 结尾的最长递增子序列的长度
    max_len =1   # 记录全局最大值

    # 动态规划求解
    for i in range(1, n):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] =max(dp[i], dp[j] + 1)
        max_len =max(max_len, dp[i])

    return max_len


if __name__ == "__main__":
    n =int(input())   # 输入数组长度
    nums =list(map(int, input().split()))   # 输入数组元素
    # 计算最长递增子序列的长度
    result = length_of_LIS(nums)
    print(result)   # 输出结果
```

5. 求一个序列的最长递增子序列。设$\{a_1, a_2, \cdots, a_n\}$是 n 个不同实数构成的序列，$\{a_{k_1}, a_{k_2}, \cdots, a_{k_m}\}$是一个递增子序列，其中 $k_1<k_2<\cdots<k_m$，并且 $a_{k_1}, a_{k_2}, \cdots, a_{k_m}$

C/C++代码：

```
#include <stdio.h>
#include <stdlib.h>

//打印最长递增子序列
void printLIS(int * nums, int * dp, int * parent, int n) {
    int max_len =0;
    int max_index =0;

    // 找到最长递增子序列的最大长度和结束位置
    for (int i = 0; i < n; i++) {
        if (dp[i] > max_len) {
            max_len = dp[i];
            max_index = i;
        }
    }
```

```
    // 回溯打印最长递增子序列
    int lis[max_len];
    int index = max_len -1;
    while (max_index != -1) {
        lis[index--] = nums[max_index];
        max_index = parent[max_index];
    }

    // 输出最长递增子序列
    for (int i = 0; i < max_len; i++) {
        printf("%d ", lis[i]);
    }
    printf("\n");
}

int main() {
    int n;
    scanf("%d", &n);   // 输入数组的长度

    int * nums = (int * )malloc(n *  sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("%d", &nums[i]);   // 输入数组元素
    }

    int * dp = (int * )malloc(n *  sizeof(int));
    int * parent = (int * )malloc(n *  sizeof(int));

    // 初始化 dp 和 parent 数组
    for (int i = 0; i < n; i++) {
        dp[i] =1;   // 每个元素最小的 LIS 长度为 1
        parent[i] = -1;   // -1 表示没有父节点
    }

    // 动态规划求解 LIS
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (nums[i] > nums[j] && dp[i] < dp[j] + 1) {
                dp[i] = dp[j] +1;
                parent[i] = j; // 更新 parent 数组，记录前驱元素
```

```
            }
        }
    }
    // 打印最长递增子序列
    printLIS(nums, dp, parent, n);
    free(nums);
    free(dp);
    free(parent);
    return 0;
}
```

Python 代码：

```
def print_LIS(nums, dp, parent, n):
    max_len =0
    max_index =0

    # 找到最长递增子序列的最大长度和结束位置
    for i in range(n):
        if dp[i] > max_len:
            max_len = dp[i]
            max_index = i

    # 回溯打印最长递增子序列
    lis = [0] *  max_len
    index = max_len -1
    while max_index ! = -1:
        lis[index] = nums[max_index]
        index -=1
        max_index = parent[max_index]

    # 输出最长递增子序列
    print(" ".join(map(str, lis)))

def main():
    n =int(input())  # 输入数组的长度
    nums =list(map(int, input().split()))  # 输入数组元素
    dp = [1] *  n  # dp[i] 表示以 nums[i] 结尾的最长递增子序列的长度
    parent = [-1] *  n  # parent[i] 记录 nums[i] 的前驱元素
    # 动态规划求解 LIS
    for i in range(1, n):
```

```
            for j in range(i):
                if nums[i] > nums[j] and dp[i] < dp[j] + 1:
                    dp[i] = dp[j] +1
                    parent[i] = j# 更新 parent 数组，记录前驱元素
        # 打印最长递增子序列
        print_LIS(nums, dp, parent, n)

if __name__ == "__main__":
    main()
```