《算法设计与分析》
课程实验报告

专业：	计算机科学与技术
班级：	2021211304
姓名：	张梓良
学号：	2021212484

算法设计与分析（第 3 章）：动态规划
编程实验报告

张梓良 2021212484

日期：2023 年 12 月 3 日

1 概述

1.1 实验内容及要求

1. 最长公共子序列
    - 利用“附件 1. 最长公共子序列输入数据-2023”中给出的字符串 A, B, C, D, 分别找出下列两两字符串间的最长公共子串，并输出结果：A-B,C-D,A-D,C-B
2. 最长递减子序列
    - 利用最长公共子序列求解最长递减子序列 (**非严格递减**) 问题
    - 利用“附件 2. 最大子段和输入数据-序列 1-2023”、“附件 2. 最大子段和输入数据-序列 2-2023”，求这两个序列中的最长递减子序列
3. 最大子段和
    - 针对“附件 2. 最大子段和输入数据-序列 1-2023”，“附件 2. 最大子段和输入数据-序列 2-2023”中给出的序列 1、序列 2, 分别计算其最大子段和
    - 指出最大子段和在原序列中的位置并给出最大子段和具体值

1.2 实验环境

- cmake version 3.27.0-rc4
- gcc version 8.1.0
- Visual Studio Code 1.82.2
- OS: Windows_NT x64 10.0.22621

2 最长公共子序列

2.1 算法设计

```
Algorithm 1: LCSLength($m, n, x, y$)
输入: 整数 $m, n$, 字符数组 $x, y$
for $i = 1$ to $m$ do
    for $j = 1$ to $n$ do
        if $x[i] = y[j]$ then
            $c[i][j] \leftarrow c[i-1][j-1] + 1$;
            $b[i][j] \leftarrow 1$;
        end
        else if $c[i-1][j] \geq c[i][j-1]$ then
            $c[i][j] \leftarrow c[i-1][j]$;
            $b[i][j] \leftarrow 2$;
        end
        else
            $c[i][j] \leftarrow c[i][j-1]$;
            $b[i][j] \leftarrow 3$;
        end
    end
end
```

```
Algorithm 2: LCS($i, j, x$)
输入: 整数 $i, j$, 字符数组 $x$
if $i = 0$ or $j = 0$ then
    return;
end
if $b[i][j] = 1$ then
    LCS($i-1, j-1, x$);
    输出 $x[i]$;
end
else if $b[i][j] = 2$ then
    LCS($i-1, j, x$);
end
else
    LCS($i, j-1, x$);
end
```

2.2 算法优化

对以上算法的空间复杂度进行优化，去掉 `b` 数组，得到以下算法：

Algorithm 3: LCSLength(m, n, x, y) (去掉`b`数组)

```
输入: 整数 $m, n$，字符数组 $x, y$
for $i = 1$ to $m$ do
    for $j = 1$ to $n$ do
        if $x[i] = y[j]$ then
            $c[i][j] \leftarrow c[i-1][j-1] + 1$;
        end
        else if $c[i-1][j] \geq c[i][j-1]$ then
            $c[i][j] \leftarrow c[i-1][j]$;
        end
        else
            $c[i][j] \leftarrow c[i][j-1]$;
        end
    end
end
```

Algorithm 4: LCS(i, j, x, y) (去掉`b`数组)

```
输入: 整数 $i, j$，字符数组 $x, y$
if $i = 0$ or $j = 0$ then
    return;
end
if $x[i] = y[j]$ then
    LCS($i-1, j-1, x, y$);
    输出 $x[i]$;
end
else if $c[i][j] = c[i-1][j]$ then
    LCS($i-1, j, x, y$);
end
else
    LCS($i, j-1, x, y$);
end
```

若只需求出最长公共子序列的长度，还可进一步优化空间复杂度，将 `c` 数组优化为一维数组，得到以下算法：

Algorithm 5: LCSLength(m, n, x, y) (c数组优化为一维)

```
   输入: 整数 $m, n$，字符数组 $x, y$
1  初始化 c 数组，设置为 0
2  for $i = 1$ to $m$ do
3      for $j = 1$ to $n$ do
4          $c[j] \leftarrow max(c[j-1], c[j])$
5          if $x[i] = y[j]$ then
6              $c[j] \leftarrow tmp[j-1] + 1$
7          end
8          for $j = 1$ to $n$ do
9              $tmp[j] \leftarrow c[j]$
10         end
11     end
12 end
```

2.3 算法性能分析

时间复杂度

- LCSLength: $O(m * n)$
- LCS: $O(m + n)$
- total: $O(m * n + m + n)$

空间复杂度

- 无优化：$O(m * n)$
- 去掉 `b` 数组：$O(m * n)$
- 将 `c` 数组优化为一维：$O(n)$

2.4 结果展示及分析

```
=======A-B=======
The length of LCS is: 136
The LCS is:
↪  an+algorithm+is+any+welldefined+computational+procedure+that+takes
+some+values+as+input+and+produces+some+values+as+output20212113xx2023


=======C-D=======
The length of LCS is: 136
The LCS is:
↪ an+algorithm+is+any+welldefined+computational+procedure+that+takes
+some+values+as+input+and+produces+some+values+as+output20212113xx2023

=======A-D=======
The length of LCS is: 136
The LCS is:
↪ an+algorithm+is+any+welldefined+computational+procedure+that+takes
+some+values+as+input+and+produces+some+values+as+output20212113xx2023

=======B-C=======
The length of LCS is: 136
The LCS is:
↪ an+algorithm+is+any+welldefined+computational+procedure+that+takes
+some+values+as+input+and+produces+some+values+as+output20212113xx2023
```

结果符合预期，算法设计正确。

3 最长递减子序列

3.1 算法设计

利用最长公共子序列求解最长递减子序列问题，只需要求解原序列和按降序排序后的序列的最长公共子序列即为原序列的最长递减子序列。

3.2 算法优化

利用 dp+ 二分的思想可以得到时间复杂度为 $O(nlogn)$ 的算法：

Algorithm 6: LDS 算法 (O(nlogn))

```
1  LDS($a$):
       /* 找到大于等于 $a[i]$ 的最小的 $f[l]$                */
       /* $f[l+1]$ 一定小于 $a[i]$                          */
       /* 用 $a[i]$ 去替换 $f[l+1]$                         */
2      $len \leftarrow 0$
3      $f[0] \leftarrow 2 \times 10^9$
4      for $i \leftarrow 1$ to $a.size() - 1$ do
5          $l \leftarrow 0$
6          $r \leftarrow len$
7          while $l < r$ do
8              $mid \leftarrow (l + r + 1)/2$
9              if $f[mid] \geq a[i]$ then
10                 $l \leftarrow mid$
11             else
12                 $r \leftarrow mid - 1$
13             end
14         end
15         $f[l+1] \leftarrow a[i]$
16         if $l + 1 > len$ then
17             $len \leftarrow l + 1$
18         end
19     end
20     cout $\rightarrow len$
```

最终输出的 `len` 即为最长递减子序列的长度，同时 `f` 数组中存储了最长递减子序列。

3.3 算法性能分析

时间复杂度

- 无优化：$O(n^2)$
- dp+ 二分优化：$O(nlogn)$

空间复杂度

- 无优化：$O(n^2)$
- dp+ 二分优化：$O(n)$

3.4 结果展示及分析

```
The length of LDS of sequence1: 40
The LDS of sequence1: 99 99 95 91 89 87 87 79 76 72 65 61 60 56 56 51 50
↪ 47 36 31 27 27 27 19 3 0 0 0 -4 -10 -14 -15 -17 -22 -31 -100 -200
↪ -230 -301 -305
The length of LDS of sequence2: 30
The LDS of sequence2: 100 49 47 47 39 38 37 34 34 34 33 28 27 24 24 5 -2
↪ -6 -10 -11 -25 -25 -32 -38 -41 -42 -44 -70 -304 -307
```

由于题目中对于递减子序列的定义是**允许** $a[i] \geq a[j], i < j$，因此结果中的最长递减子序列中存在值相同的项。

结果符合预期，算法设计正确。

4 最大子段和

4.1 算法设计

Algorithm 7: MaxSum 算法

```
int MaxSum($n$, $a$)
    $max\_sum \leftarrow -\infty$
    $sum \leftarrow 0$
    $cur\_i \leftarrow 0$
    $cur\_j \leftarrow 0$
    for $i \leftarrow 1$ to $n$ do
        if $sum > 0$ then
            $cur\_j \leftarrow i$
            $sum \leftarrow sum + a[i]$
        else
            $cur\_i \leftarrow i$
            $cur\_j \leftarrow i$
            $sum \leftarrow a[i]$
        end
        if $sum > max\_sum$ then
            $best\_i \leftarrow cur\_i$
            $best\_j \leftarrow cur\_j$
            $max\_sum \leftarrow sum$
    end
    return $max\_sum$
```

4.2 算法性能分析

时间复杂度 $O(n)$

空间复杂度 $O(1)$

4.3 结果展示及分析

```
=====Sequence1=====
Best i = 43, best j = 383
Max subsequence sum of sequence1 is: 6914


=====Sequence2=====
Best i = 75, best j = 210
Max subsequence sum of sequence2 is: 2583

Verifying...
=====Sequence1=====
Best i = 43, best j = 383
Max subsequence sum of sequence1 is: 6914

=====Sequence2=====
Best i = 74, best j = 210
Max subsequence sum of sequence2 is: 2583
```

算法结果和暴力枚举得到的结果相同，算法设计正确。

说明：暴力枚举得到的第二个序列的最大子段的开始位置和算法结果不相同是因为**原序列的第 74 项为** 0，因此不影响最大子段和。

5 总结

实验内容主要涉及最长公共子序列、最长递减子序列以及最大子段和这三个经典的动态规划问题。通过实验，我对这些问题的求解方法有了更深入的理解。

在最长公共子序列部分，我利用给定的字符串进行了两两比较，找出了 A-B、C-D、A-D、C-B 之间的最长公共子串，并成功输出了结果。这一部分让我熟悉了最长公共子序列的求解过程，加深了对动态规划算法的理解。

在最长递减子序列部分，我利用最长公共子序列的思想来解决了最长递减子序列（非严格递减）的问题。通过对附件提供的数据进行求解，我成功地找到了两个序列中的最长递减子序列，进一步巩固了动态规划算法在不同问题中的应用。

在最大子段和部分，我针对附件提供的序列数据，分别计算了序列 1 和序列 2 的最大子段和，并找出了最大子段和在原序列中的位置，给出了具体数值。这一部分让我对最大子段和问题有了更深入的了解，也锻炼了我的算法分析能力。

通过本次实验，我不仅掌握了动态规划算法在最长公共子序列、最长递减子序列和最大子段和等问题中的应用，还提高了自己的编程能力和算法设计水平。这些知识和经验对我今后的学习和工作都将有着重要的指导作用。

A LCS_1.cpp

```cpp
/**
 * @file LCS_1.cpp
 * @author zhang ziliang (ziliangzhang@bupt.edu.cn)
 * @brief 朴素版本最长公共子序列算法
 * @date 2023-11-30
 */
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>

using namespace std;

const int MAX_SIZE = 2000;

int c[MAX_SIZE][MAX_SIZE]; // c[i][j]表示序列Xi和Yj的最长公共子序列的长度
int b[MAX_SIZE][MAX_SIZE]; // b[i][j]记录c[i][j]的值是由哪一个子问题的解得到的

// 求最长公共子序列的长度
void LCSLength(int m, int n, char *x, char *y)
{
    int i, j;
    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++)
        {
            if (x[i] == y[j])
            {
                c[i][j] = c[i - 1][j - 1] + 1;
                b[i][j] = 1;
            }
            else if (c[i - 1][j] >= c[i][j - 1])
            {
                c[i][j] = c[i - 1][j];
                b[i][j] = 2;
            }
            else
            {
                c[i][j] = c[i][j - 1];
                b[i][j] = 3;
            }
        }
}

// 输出最长公共子序列
void LCS(int i, int j, char *x)
```

```cpp
{
    if (i == 0 || j == 0)
        return;
    if (b[i][j] == 1)
    {
        LCS(i - 1, j - 1, x);
        cout << x[i];
    }
    else if (b[i][j] == 2)
        LCS(i - 1, j, x);
    else
        LCS(i, j - 1, x);
}

int main()
{
    char str[4][MAX_SIZE];

    // 读入数据
    ifstream file("DATA/LCS_data.txt", ios::in);
    if (!file.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    string line;
    int i = 0;
    while (getline(file, line))
    {
        if (line.size() > 10)
        {
            strcpy(str[i] + 1, line.c_str());
            i++;
        }
    }

    // 求最长公共子序列
    int size_a = strlen(str[0] + 1), size_b = strlen(str[1] + 1), size_c = strlen(
        str[2] + 1), size_d = strlen(str[3] + 1);

    LCSLength(size_a, size_b, str[0], str[1]);
    cout << "=======A-B=======" << endl;
    cout << "The length of LCS is: " << c[size_a][size_b] << endl;
    cout << "The LCS is: ";
    LCS(size_a, size_b, str[0]);
    cout << endl << endl;

    LCSLength(size_c, size_d, str[2], str[3]);
    cout << "=======C-D=======" << endl;
    cout << "The length of LCS is: " << c[size_c][size_d] << endl;
    cout << "The LCS is: ";
    LCS(size_c, size_d, str[2]);
    cout << endl << endl;

    LCSLength(size_a, size_d, str[0], str[3]);
    cout << "=======A-D=======" << endl;
    cout << "The length of LCS is: " << c[size_a][size_d] << endl;
    cout << "The LCS is: ";
    LCS(size_a, size_d, str[0]);
    cout << endl << endl;

    LCSLength(size_b, size_c, str[1], str[2]);
    cout << "=======B-C=======" << endl;
    cout << "The length of LCS is: " << c[size_b][size_c] << endl;
    cout << "The LCS is: ";
    LCS(size_b, size_c, str[1]);
    cout << endl << endl;

    return 0;
}
```

B LCS_2.cpp

```
/**
 * @file LCS_2.cpp
 * @author zhang ziliang (ziliangzhang@bupt.edu.cn)
 * @brief 去掉b数组，减少空间复杂度
 * @date 2023-11-30
 */
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>

using namespace std;

const int MAX_SIZE = 2000;

int c[MAX_SIZE][MAX_SIZE];

// 求最长公共子序列的长度
void LCSLength(int m, int n, char *x, char *y)
{
```

```cpp
    int i, j;
    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++)
        {
            if (x[i] == y[j])
                c[i][j] = c[i - 1][j - 1] + 1;
            else if (c[i - 1][j] >= c[i][j - 1])
                c[i][j] = c[i - 1][j];
            else
                c[i][j] = c[i][j - 1];
        }
}

// 输出最长公共子序列
void LCS(int i, int j, char *x, char *y)
{
    if (i == 0 || j == 0)
        return;
    if (x[i] == y[j])
    {
        LCS(i - 1, j - 1, x, y);
        cout << x[i];
    }
    else if (c[i][j] == c[i - 1][j])
        LCS(i - 1, j, x, y);
    else
        LCS(i, j - 1, x, y);
}

int main()
{
    char str[4][MAX_SIZE];

    // 读入数据
    ifstream file("DATA/LCS_data.txt", ios::in);
    if (!file.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    string line;
    int i = 0;
    while (getline(file, line))
    {
        if (line.size() > 10)
        {
            strcpy(str[i] + 1, line.c_str());
            i++;
        }
    }

    // 求最长公共子序列
    int size_a = strlen(str[0] + 1), size_b = strlen(str[1] + 1), size_c = strlen(str[2] + 1), size_d = strlen(str[3] + 1);

    LCSLength(size_a, size_b, str[0], str[1]);
    cout << "========A-B========" << endl;
    cout << "The length of LCS is: " << c[size_a][size_b] << endl;
    cout << "The LCS is: ";
    LCS(size_a, size_b, str[0], str[1]);
    cout << endl << endl;

    LCSLength(size_c, size_d, str[2], str[3]);
    cout << "========C-D========" << endl;
    cout << "The length of LCS is: " << c[size_c][size_d] << endl;
    cout << "The LCS is: ";
    LCS(size_c, size_d, str[2], str[3]);
    cout << endl << endl;

    LCSLength(size_a, size_d, str[0], str[3]);
    cout << "========A-D========" << endl;
    cout << "The length of LCS is: " << c[size_a][size_d] << endl;
    cout << "The LCS is: ";
    LCS(size_a, size_d, str[0], str[3]);
    cout << endl << endl;

    LCSLength(size_b, size_c, str[1], str[2]);
    cout << "========B-C========" << endl;
    cout << "The length of LCS is: " << c[size_b][size_c] << endl;
    cout << "The LCS is: ";
    LCS(size_b, size_c, str[1], str[2]);
    cout << endl << endl;

    return 0;
}
```

C LCS_3.cpp

```cpp
/**
 * @file LCS_3.cpp
 * @author zhang ziliang (ziliangzhang@bupt.edu.cn)
 * @brief 将c数组改为一维数组，减少空间复杂度
 * @date 2023-11-30
 */
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>

using namespace std;

const int MAX_SIZE = 2000;

int c[MAX_SIZE];
int tmp[MAX_SIZE]; // 记录c[i-1][j-1]

// 求最长公共子序列的长度
void LCSLength(int m, int n, char *x, char *y)
{
    // 初始化，若不初始化，上一次调用LCSLength写入c数组的值会影响本次调用
    memset(c, 0, sizeof(c));

    int i, j;
    for (i = 1; i <= m; i++)
    {
        for (j = 1; j <= n; j++)
        {
            c[j] = max(c[j - 1], c[j]);
            if (x[i] == y[j])
                c[j] = tmp[j - 1] + 1;
        }
        for (j = 1; j <= n; j++)
            tmp[j] = c[j];
    }
}

int main()
{
    char str[4][MAX_SIZE];

    // 读入数据
    ifstream file("DATA/LCS_data.txt", ios::in);
    if (!file.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    string line;
    int i = 0;
    while (getline(file, line))
```

```cpp
    {
        if (line.size() > 10)
        {
            strcpy(str[i] + 1, line.c_str());
            i++;
        }
    }

    // 求最长公共子序列
    int size_a = strlen(str[0] + 1), size_b = strlen(str[1] + 1), size_c = strlen(str[2] + 1), size_d = strlen(str[3] + 1);

    LCSLength(size_a, size_b, str[0], str[1]);
    cout << "=======A-B=======" << endl;
    cout << "The length of LCS is: " << c[size_b] << endl;

    LCSLength(size_c, size_d, str[2], str[3]);
    cout << "=======C-D=======" << endl;
    cout << "The length of LCS is: " << c[size_d] << endl;

    LCSLength(size_a, size_d, str[0], str[3]);
    cout << "=======A-D=======" << endl;
    cout << "The length of LCS is: " << c[size_d] << endl;

    LCSLength(size_b, size_c, str[1], str[2]);
    cout << "=======B-C=======" << endl;
    cout << "The length of LCS is: " << c[size_c] << endl;

    return 0;
}
```

D LDS_1.cpp

```cpp
/**
 * @file LDS_1.cpp
 * @author zhang ziliang (ziliangzhang@bupt.edu.cn)
 * @brief 利用最长公共子序列求解最长递减子序列问题
 * @date 2023-11-30
 */
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;
```

```cpp

const int MAX_SIZE = 2000;
const int INF = 0x3f3f3f3f;

int c[MAX_SIZE][MAX_SIZE];

// 求最长公共子序列的长度
void LCSLength(int m, int n, const vector<int> &x, const vector<int> &y)
{
    int i, j;
    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++)
        {
            if (x[i] == y[j])
                c[i][j] = c[i - 1][j - 1] + 1;
            else if (c[i - 1][j] >= c[i][j - 1])
                c[i][j] = c[i - 1][j];
            else
                c[i][j] = c[i][j - 1];
        }
}

// 输出最长公共子序列
void LCS(int i, int j, const vector<int> &x, const vector<int> &y)
{
    if (i == 0 || j == 0)
        return;
    if (x[i] == y[j])
    {
        LCS(i - 1, j - 1, x, y);
        cout << " ";
    }
    else if (c[i][j] == c[i - 1][j])
        LCS(i - 1, j, x, y);
    else
        LCS(i, j - 1, x, y);
}

int main()
{
    vector<int> a, b;
    a.push_back(INF);
    b.push_back(INF);

    // 读入数据
    ifstream file("DATA/LDS_data1.txt", ios::in);
    if (!file.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    string line;
    while (getline(file, line) && line != "")
        a.push_back(stoi(line));

    ifstream file2("DATA/LDS_data2.txt", ios::in);
    if (!file2.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    while (getline(file2, line))
        b.push_back(stoi(line));

    // 将原序列降序排列
    vector<int> a_t = a, b_t = b;
    sort(a_t.begin(), a_t.end(), [](int a, int b)
         { return a > b; });
    sort(b_t.begin(), b_t.end(), [](int a, int b)
         { return a > b; });

    // 求最长递减子序列
    LCSLength(a.size() - 1, a_t.size() - 1, a, a_t);
    cout << "The length of LDS of sequence1: " << c[a.size() - 1][a_t.size() - 1]
        << endl;
    cout << "The LDS of sequence1: ";
    LCS(a.size() - 1, a_t.size() - 1, a, a_t);
    cout << endl;
    LCSLength(b.size() - 1, b_t.size() - 1, b, b_t);
    cout << "The length of LDS of sequence2: " << c[b.size() - 1][b_t.size() - 1]
        << endl;
    cout << "The LDS of sequence2: ";
    LCS(b.size() - 1, b_t.size() - 1, b, b_t);

    return 0;
}
```

E LDS_2.cpp

```
/**
 * @file LDS_2.cpp
 * @author zhang ziliang (ziliangzhang@bupt.edu.cn)
 * @brief 最长递减子序列问题的O(nlogn)算法
```

```cpp
 * @date 2023-11-30
 */

/*
长度为i的递减子序列，只需要保存下结尾数字最大的一个
因为结尾数字最大，能够保证后序子序列能够全部探测到
*/
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

using namespace std;

const int N = 1000;
const int INF = 0x3f3f3f3f;

int f[N]; // f[i]存储长度为i的递减子序列中最大结尾数字
// f[N]一定是单调递减的

void LDS(const vector<int> &a)
{
    /*
    找到大于等于a[i]的最小的f[l]
    f[l + 1]一定小于a[i]
    用a[i]去替换f[l + 1]
    */
    int len = 0;
    f[0] = 2 * 1e9;
    for (int i = 1; i <= a.size() - 1; i++)
    {
        int l = 0, r = len;
        while (l < r)
        {
            int mid = l + r + 1 >> 1;
            if (f[mid] >= a[i])
                l = mid;
            else
                r = mid - 1;
        }
        f[l + 1] = a[i];
        if (l + 1 > len)
            len = l + 1;
    }
    cout << len << endl;
}

```

```
int main()
{
    vector<int> a, b;
    a.push_back(INF);
    b.push_back(INF);

    // 读入数据
    ifstream file("DATA/LDS_data1.txt", ios::in);
    if (!file.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    string line;
    while (getline(file, line) && line != "")
        a.push_back(stoi(line));

    ifstream file2("DATA/LDS_data2.txt", ios::in);
    if (!file2.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    while (getline(file2, line))
        b.push_back(stoi(line));

    // 计算LDS
    cout << "The length of LDS of sequence1: ";
    LDS(a);
    cout << "The length of LDS of sequence2: ";
    LDS(b);

    return 0;
}
```

F MSS.cpp

```
/**
 * @file MSS.cpp
 * @author zhang ziliang (ziliangzhang@bupt.edu.cn)
 * @brief 最大子段和算法
 * @date 2023-12-01
 */
#include <iostream>
#include <fstream>
#include <string>
```

```cpp

using namespace std;

const int MAX_SIZE = 1000;
const int INF = 0x3f3f3f3f;

int best_i, best_j; // 最大子段的起始位置和结束位置

// 求解最大子段和
int MaxSum(int n, int *a)
{
    int max_sum = -INF, sum = 0; // 最大子段和，当前子段和
    int cur_i, cur_j;            // 当前子段的起始位置和结束位置
    for (int i = 1; i <= n; i++)
    {
        if (sum > 0)
        {
            cur_j = i;
            sum += a[i];
        }
        else
        {
            cur_i = i;
            cur_j = i;
            sum = a[i];
        }
        if (sum > max_sum)
        {
            best_i = cur_i;
            best_j = cur_j;
            max_sum = sum;
        }
    }
    return max_sum;
}

int main()
{
    int a[MAX_SIZE], b[MAX_SIZE];
    int n = 0, m = 0;

    // 读入数据
    ifstream file("DATA/MSS_data1.txt", ios::in);
    if (!file.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    string line;
    while (getline(file, line) && line != "")
        a[++n] = stoi(line);

    ifstream file2("DATA/MSS_data2.txt", ios::in);
    if (!file2.is_open())
    {
        cerr << "Failed to open file." << endl;
        return 1;
    }
    while (getline(file2, line))
        b[++m] = stoi(line);

    // 求解最大子段和
    int max_sum = MaxSum(n, a);
    cout << "=====Sequence1=====" << endl;
    cout << "Best i = " << best_i << ", best j = " << best_j << endl;
    cout << "Max subsequence sum of sequence1 is: " << max_sum << endl
         << endl;
    max_sum = MaxSum(m, b);
    cout << "=====Sequence2=====" << endl;
    cout << "Best i = " << best_i << ", best j = " << best_j << endl;
    cout << "Max subsequence sum of sequence2 is: " << max_sum << endl
         << endl;

    // 验证结果
    cout << "Verifying..." << endl;
    max_sum = -INF;
    for (int i = 1; i <= n; i++)
    {
        int sum = 0;
        for (int j = i; j <= n; j++)
        {
            sum += a[j];
            if (sum > max_sum)
            {
                max_sum = sum;
                best_i = i;
                best_j = j;
            }
        }
    }
    cout << "=====Sequence1=====" << endl;
    cout << "Best i = " << best_i << ", best j = " << best_j << endl;
    cout << "Max subsequence sum of sequence1 is: " << max_sum << endl
         << endl;

    max_sum = -INF;
    for (int i = 1; i <= m; i++)
    {
        int sum = 0;
        for (int j = i; j <= m; j++)
        {
            sum += b[j];
            if (sum > max_sum)
            {
                max_sum = sum;
                best_i = i;
                best_j = j;
            }
        }
    }
    cout << "=====Sequence2=====" << endl;
    cout << "Best i = " << best_i << ", best j = " << best_j << endl;
    cout << "Max subsequence sum of sequence2 is: " << max_sum << endl;

    return 0;
}
```