算法设计与分析第三章上机作业

班级：2018211313 班　　学号：2018211366　　姓名：蒋潇逸

版本：1.0

更新：*November 12, 2021*

本文档是算法设计与分析第三章上机作业，具体代码详见附录。

目录

1 最长公共子序列

1.1 题目描述

若给定序列 $X = \{x_1, x_2, ..., x_m\}$，则另一序列 $Z = \{z_1, z_2, ..., z_k\}$ 是 X 的子序列是指存在一个严格递增下标序列 $\{i_1, i_2, ..., i_k\}$ 使得对于所有 $j = 1, 2, ..., k$ 有：$z_j = x_i$

1.2 实验步骤以及要求

利用“附件 1. 最长公共子序列输入数据-2020”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

1.3 代码实现

根据最优子结构性质，建立子问题最优值的递归关系。$c[i][j]$ 序列 $X(i)$ 和 $Y(j)$ 的最长公共子序列的长度

1. $X(i) = x_1, x_2, ..., x_i; Y(j) = y_1, y_2, ..., y_j$ 当 $i = 0$ 或 $j = 0$ 时，即其中 1 个序列为空，则空序列是 X_i 和 Y_j 的最长公共子序列。故此时 $C[i][j] = 0$。

2. 其它情况下，由最优子结构性质可建立递归关系如下：

$$c[i][j] = \begin{cases} 0 & i = 0, 或 j = 0 \\ c[i-1][j-1] + 1 & i, j > 0; x_i = y_j \\ \max\{c[i][j-1], c[i-1][j]\} & i, j > 0; x_i \neq y_j \end{cases}$$

1.4 结果分析

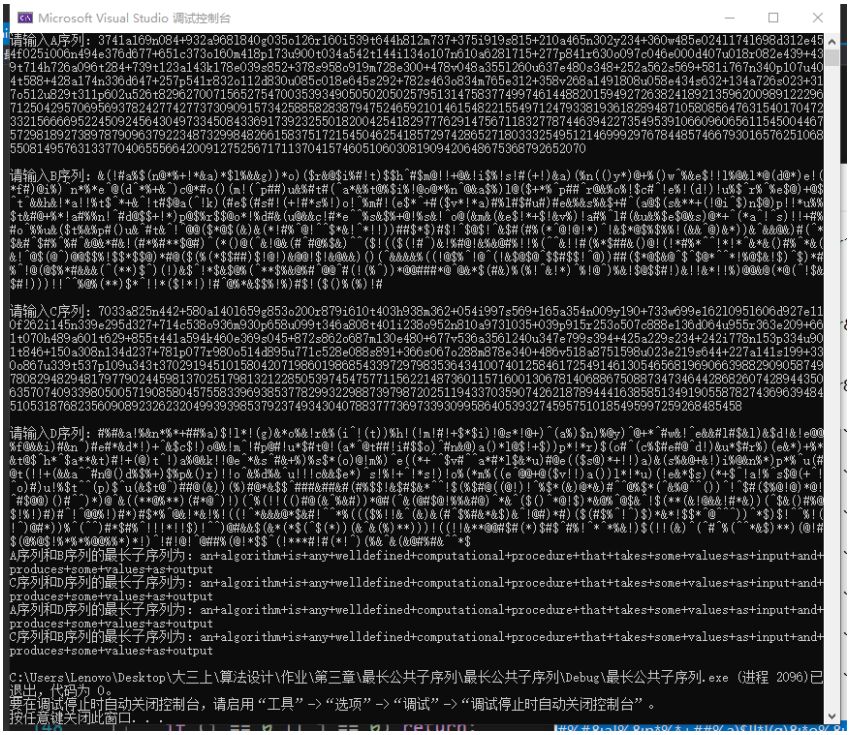


图 1: 运行结果

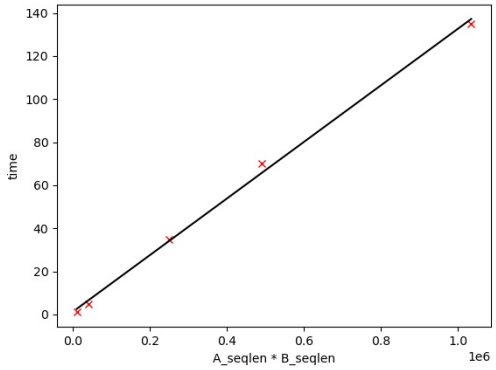
四组字符串的最长公共子序列均为 $an + algorithm + is + any + welldefined + computational + procedure + that + takes + some + values + as + input + and + produces + some + values + as + output$ 符合预期。

1.5 拟合算法时间复杂度

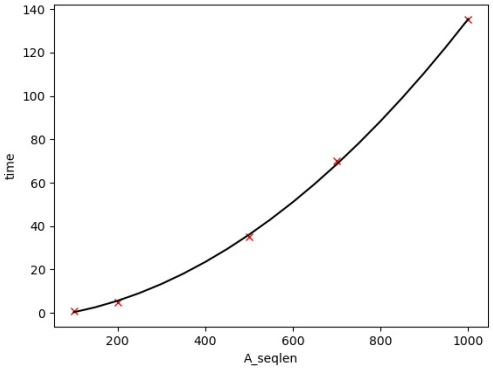
输入对不同长度的字符串，程序运行时间如下表所示

表 1: 运行时间

字符串 A 长度	字符串 B	两字符串长度相乘	时间
100	100	10000	1ms
200	200	40000	5ms
500	500	250000	35ms
700	700	490000	70ms
1032	1003	1035096	145ms



(a) m*n 拟合结果



(b) n 的拟合结果

由于当用 m*n 与时间拟合时，是个线性的函数，当用 n 与实践拟合时 (m,n 的数量差不多的情况下)，是个二次函数，故可知时间复杂度为 $O(m*n)$。

1.6 算法改进

在算法 *lcsLength* 和 *lcs* 中，可进一步将数组 b 省去：数组元素 c[i][j] 的值仅由 c[i-1][j-1],c[i-1][j] 和 c[i][j-1] 这 3 个数组元素的值所确定。对于给定的数组元素 c[i][j]，可以不借助于数组 b 而仅借助于 c 本身在 $O(1)$ 时间内确定 c[i][j] 的值是由 c[i-1][j-1],c[i-1][j] 和 c[i][j-1] 中哪一个值所确定的。

i 行某一列元素或者第 i-1 行某一列元素基础上的，如果不需要求解最长公共子序列是什么，只需要求解其长度，那么我们可以建立一个 2*n 的滚动数组代替 n*m 的数组来实现空间优化，这种方法只需要用到 cur 层和 pre 层的数据，通过 cur 和 pre 的不断交换实现 m*n 的数组功能。

```
cin >> s1 >> s2;
len1 = s1.size();
len2 = s2.size();
cur = 1;
pre = 0;
for (i = 0; i <= len2; i++)dp[0][i] = 0;
dp[1][0] = 0;
for (i = 1; i <= len1; i++)
{
    for (j = 1; j <= len2; j++)
    {
        if (s1[i - 1] == s2[j - 1])
            dp[cur][j] = dp[pre][j - 1] + 1;
        else
            dp[cur][j] = max(dp[cur][j - 1], dp[pre][j]);
    }
    swap(cur, pre);
}
cout << dp[pre][len2] << endl;
```

图 3: 最长子序列空间复杂度优化

2 最长递减子序列

2.1 题目描述

利用最长公共子序列求解下列最长递减子序列问题：

- 给定由 n 个整数 $a_1, a_2, ..., a_n$ 构成的序列，在这个序列中随意删除一些元素后可得到一个子序列 $a_i, a_j, a_k, ..., a_m$，其中 $1 \leq i \leq m \leq n$，并且 $a_i \leq a_j \leq a_k \leq ... \leq a_m$，则称序列 $a_i, a_j, a_k, ..., a_m$ 为原序列的一个递减子序列，长度最长的递减子序列即为原序列的最长递减子序列
- 例如，序列 $\{1, 7, 2, 3, 6, 5\}$，它的一个最长递减子序列为 $\{7, 6, 5\}$

2.2 实验步骤以及要求

要求：

- 利用“附件 2. 最大子段和输入数据-序列 1”、“附件 2. 最大子段和输入数据-序列 2”，求这两个序列中的最长递减子序列

2.3 代码实现

对于动态规划问题，往往存在递推解决方法，这个问题也不例外。要求长度为 i 的序列的 $A_i\{a_1, a_2, , a_i\}$ 最长递减子序列，需要先求出序列 $A_{i-1}\{a_1, a_2, , a_{i-1}\}$ 中以各元素 $(a_1, a_2, , a_{i-1})$ 作为最大元素的最长递减序列，然后把所有这些递减序列与 a_i 比较，如果某个长度为 m 序列的末尾元素 $a_j(j < i)$ 比 a_i 要大，则将元素 a_i 加入这个递减子序列，得到一个新的长度为 $m + 1$ 的新序列，否则其长度不变，将处理后的所有 i 个序列的长度进行比较，其中最长的序列就是所求的最长递增子序列。

2.4 结果分析

图 4: 序列 1 运行结果

图 5: 序列 2 运行结果

2.5 拟合算法时间复杂度

对于不同长度的序列 N，程序运行的时间如下表所示

表 2: 运行时间

序列长度	运行时间
369	1ms
1107	4ms
2214	11ms
4428	46ms

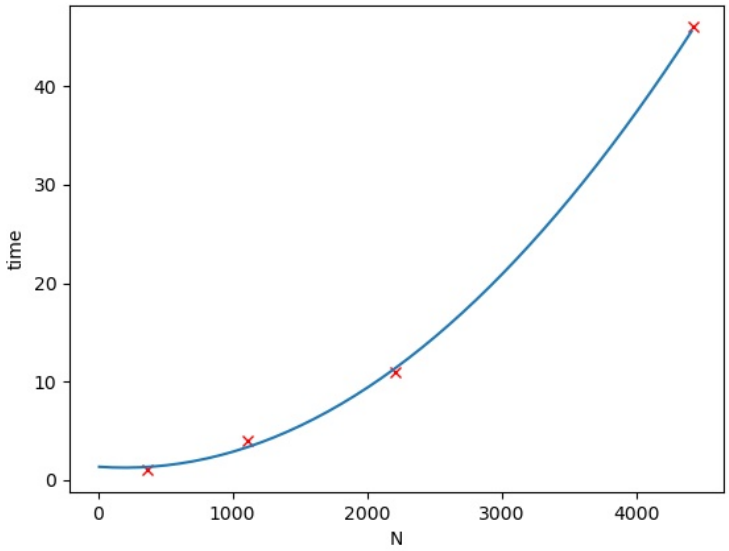


图 6: 拟合结果

从拟合结果可以看出该算法的时间复杂度是 $O(n^2)$

2.6 算法改进

定义一个最长子序列数组 $Array$，以及当前长度 Len，从头到尾维护数组 a

- $a[i] \leq Array[i]$（当前元素小于等于子序列结尾元素），则 a[i] 进入子序列：$Array[++Len] = a[i]$
- $a[i] > Array[i]$，这时对 Array 进行维护，把 Array 中比 a[i] 小的第一个元素替换成 a[i]（替换第一个小的原因是维护 Array 的递减性），这样可以降低后面元素进入子序列的门槛

为了降低算法复杂度，因为 Array 是降序序列，因此可以使用二分法将该算法的时间复杂度从 $O(n^2)$ 降为 $O(nlogn)$, 优化代码如图所示：

```
int LIS(int a[])
{
    int Cnt = 0;
    Array[0] = a[0];
    for (int i = 1; i < n; i++)
    {
        if (a[i] > Array[Cnt])
            Array[++Cnt] = a[i];
        else
        {
            int Index = lower_bound(Array, Array + Cnt + 1, a[i]) - Array;
            Array[Index] = a[i];
        }
    }
    return Cnt + 1;   //实际长度需+1
}
```

3 最大子段和

3.1 题目描述

给定 n 个整数组成的序列 $a_1, a_2, ..., a_n$，求该序列形如 $\sum_{k=i}^{j} a_k, 1 \le i, j \le n$ 的子段和的最大值

3.2 实验步骤以及要求

针对“附件 2. 最大子段和输入数据-序列 1”“附件 2. 最大子段和输入数据-序列 2”中给出的序列 1、序列 2, 分别计算其最大子段和。

- 序列 1：长度在 300-400 之间，由 (-100,100) 内的数字组成
- 序列 2：长度在 100-200 之间，由 (-50,50) 内的数字组成
  - 指出最大子段和在原序列中的位置
  - 给出最大子段和具体值

3.3 代码实现

- 若记 $b[j] = max(a[i] + a[i+1] + ... + a[j])$, 其中 $1 \le i \le j$, 并且 $1 \le j \le n$。则所求的最大子段和为 $max \quad b[j], 1 \le j \le n$
- 由 $b[j]$ 的定义可易知，当 $b[j-1] > 0$ 时 $b[j] = b[j-1] + a[j]$，否则 $b[j] = a[j]$。故 $b[j]$ 的动态规划递归式为: $b[j] = max(b[j-1] + a[j], a[j]), 1 \le j \le n$

3.4 结果分析

图 7: 序列 1 运行结果

图 8: 序列 2 运行结果

3.5 拟合算法时间复杂度

由于该算法的时间复杂度是 $O(n)$, 利用 clock 函数计时时，时间变化不明显却需要较大规模的数据才能让时间发生变化，故此处不再模拟运行该算法的运行时间。

4 凸多边形最优三角剖分

4.1 题目描述

- 多边形的三角剖分, 将多边形分割成互不相交的三角形的弦的集合 T 三角形顶点：凸多边形顶点，三角形边：多边形边、弦。
- 最优三角剖分, 给定凸多边形 P，以及定义在由多边形的边和弦组成的三角形上的权函数 w（如欧氏距离），要求：确定该凸多边形的三角剖分，使得即该三角剖分中诸三角形上权之和为最小——最优值

4.2 实验步骤以及要求

- 算法 1, 利用：教科书上 $O(n^3)$ 算法
  - 附件 3-1.21 个基站凸多边形数据
  - 附件 3-2.29 个基站凸多边形数据
- 算法 2, 采用近似的复杂度为 $O(n^2)$ 的剖分算法，实现三角剖分, 对子问题 $\{v(i-1), v(i), ..., v(k), ...v(j)\}$，从 $K(\leq 3)$ 个固定位置选断点 $v(k)$，比较这 K 个断点的目标值，从中选取最优断点。
  - 做图表示最优、次优三角剖分结果，可以手绘
  - 计算最优、次优三角剖分对应的目标值——边长弦长总和

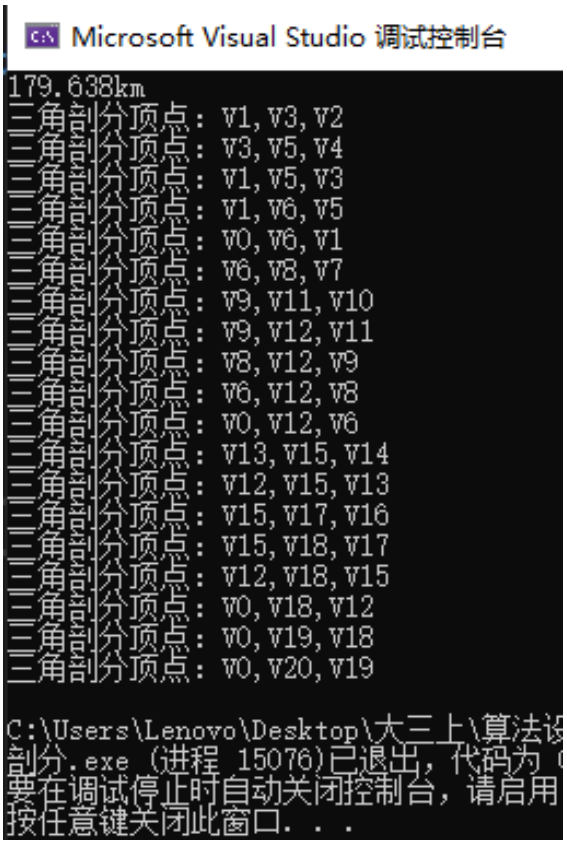
4.3 代码实现

$t[i][j]$ 的值利用最优子结构性质递归地计算:

- 当 $j-i \geq 1$ 时，凸子多边形 $P\{v_{i-1}, v_i, ..., v_j\}$ 至少有 $j-i+2 \geq 3$ 个顶点
- 选取最优顶点 v_k,，用顶点 v_k, 进行子问题划分
- $[i][j]$ 的值为三部分相加：$t[i][k]+t[k+1][j]+$ 三角形 $v_{i-1}v_kv_j$ 权值, 其中 $i \leq k \leq j-1$

为了降低该算法的复杂度，对子问题 $\{v_{i-1}, v_i, ..., v_k, ...v_j\}$，从 $K(\leq 3)$ 个固定位置选断点 v_k，比较这 K 个断点的目标值，从中选取最优断点。即在第三层循环中，计数判断，到第三层循环进行到第四次时，利用 break 函数退出循环，此方法虽然降低了时间复杂度，但是求出的解可能不是最优的。

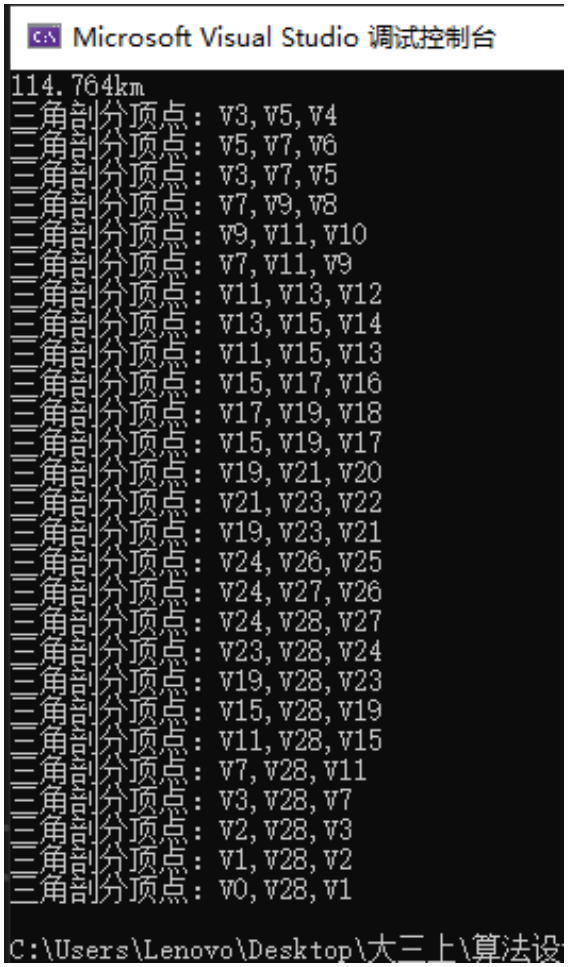
4.4 结果分析



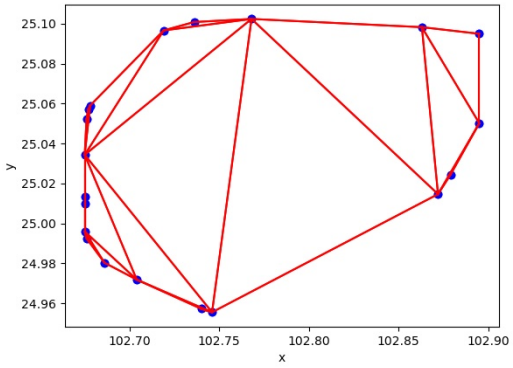
(a) 21 凸多边形最优三角剖分

(b) 21 凸多边形次优三角剖分

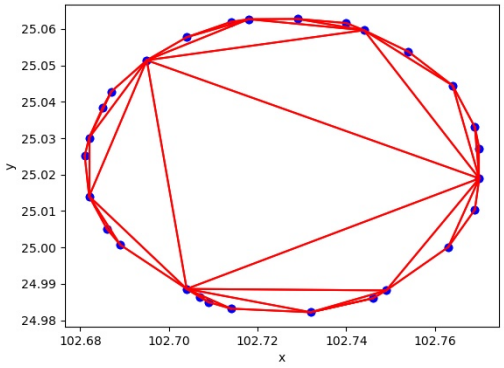
(a) 29 凸多边形最优三角剖分



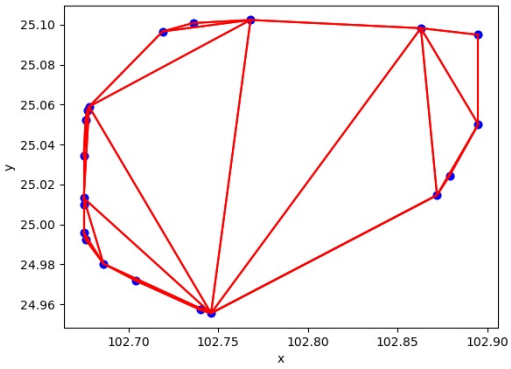
(b) 29 凸多边形次优三角剖分



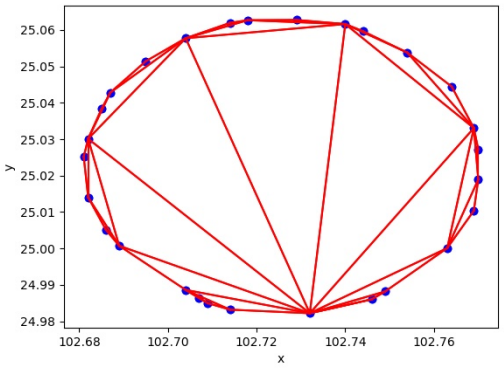
(a) 21 凸多边形最优三角剖分



(b) 29 凸多边形最优三角剖分



(c) 21 凸多边形次优三角剖分



(d) 29 凸多边形次优三角剖分

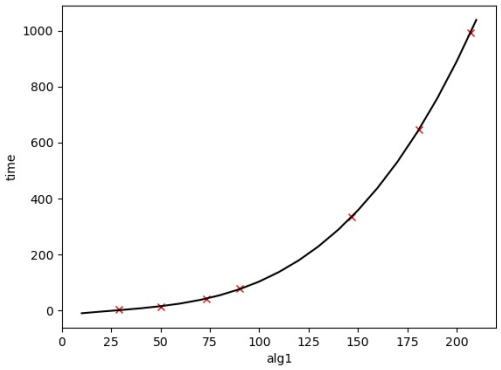
4.5 拟合算法时间复杂度

表 3: 运行时间

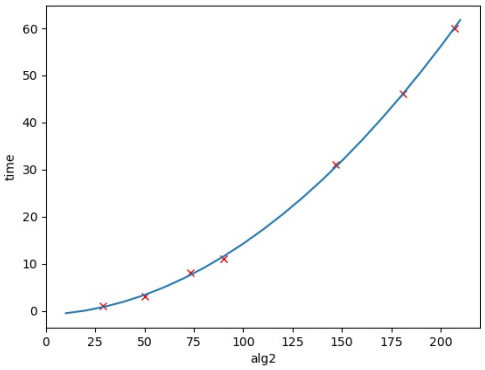
多边形顶点个数	算法 1 时间 $O(n^3)$	算法 2 时间 $O(n^3)$
29	3ms	1ms
50	14ms	3ms
73	42ms	8ms
90	79ms	11ms
147	336ms	31ms
181	647ms	46ms
207	992ms	60ms

用 n^3 拟合算法 1 的时间复杂度，效果良好；用 n^2 拟合算法 1 的时间复杂度，效果良好。说明他们的时间复杂度分别是 $O(n^2)$ 和 $O(n^3)$。虽然算法 1 的时间复杂度高，但是算法 1 求出的是最优解，虽然算法 2 时间复杂度低，但是算法 2 求出的是次优解，这

也体现了问题解的准确性与时间复杂度之间的权衡。



(a) 算法 1 拟合结果



(b) 算法 2 拟合结果

5 0-1 背包问题

5.1 题目描述

给定 n 种物品 $\{1,2,3,...,n\}$ 和一背包。物品 i 的重量是 w_i，其价值为 v_i，背包的容量为 C。问：应如何选择装入背包的物品，使得装入背包中物品在其总重量不超过背包容量 C 的前提下，背包中物品的总价值最大?

5.2 实验步骤以及要求

- 利用“附件 4. 背包问题输入数据”给出的 2 组背包数据（背包容量、物品重量、物品价值），计算最优物品装载方案
- 要求给出最优方案中，1. 各个物品是否被放入 2. 物品放入后的背包总重量、总价值

5.3 代码实现

$$m(i,j)=\begin{cases}\max\limits_{x_i\in(0,1)}\{m(i+1,j),m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

考察子问题中的物品 $\{i,i+1,i+2,...,n-1,n\}$

- 如果第 i 种物品重量 w_i 大于容量 j，第 i 种物品无法放入，可以不考虑，x_i=0, 向后考虑其它物品，子问题 $< i, j >$ 缩减为规模更小的 $< i+1, j >$
- 如果第 i 种物品重量 w_i 小于容量 j，第 i 种物品可以考虑放入，分为 2 种情况：
  - 将第 i 种物品放入，对总价值的贡献为 v_i，剩余容量为 $j - w_i$，子问题 $< i, j >$ 缩减为规模更小的 $< i+1, j-w_i >$
  - 第 i 种物品不放入，背包容量仍为 j，子问题 $< i, j >$ 缩减为规模更小的 $< i+1, j >$

5.4 结果分析

图 13: 51 个物品运行结果

图 14: 101 个物品运行结果

5.5 拟合算法时间复杂度

该问题的算法时间复杂度与最长子序列类似，这里不再画图，即当用背包容量 C* 物品个数 N 与时间拟合时，是个线性的函数，故可知时间复杂度为 $O(C * N)$。

5.6 算法改进

对于状态转移方程，考虑到每个 m[i][W] 都仅和 m[i-1][W] 或 m[i-1][W-Wi] 有关，那么可以通过利用滚动数组的方式，将 m[N][C] 所需要空间压缩至 m[2][C]，从而将空间复杂度降为 O(C)。举个例子，我们将 m[0][Wi] 状态存在二维表的第 0 行，接着推

出 m[1][Wi] 对应的值，并将 m[1][Wi] 的状态存放于二维表的第 1 行。再接着推出 m[2][Wi] 的状态时，m[0][Wi] 的状态已经没有保存意义了，因此直接将 m[2][Wi] 的状态保存到二维表的第 0 行，并以此类推，使数组空间得到循环利用。具体代码如下：

```
int GetMaxValue() {
    for (int w = 0; w <= Capacity; w++)
        selected[0][w] = 0;   //表示容量为w的背包为空时，其价值为0
    for (int i = 1; i <= N; i++)
        selected[i % 2][0] = 0;      //表示容量为0的背包，其价值为0
    for(int i=1;i<=N;i++)
        for (int w = 1; w <= Capacity; w++) {
            if (comodity[i].weight > w)
                selected[i % 2][w] = selected[(i - 1) % 2][w];
            else
                selected[i % 2][w] = Max(selected[(i - 1) % 2][w], \
                    comodity[i].value + selected[(i - 1)%2][w - comodity[i].weight]);
        }
    return selected[1][Capacity];
}
```

6 实验总结

通过本章的学习和上机作业的巩固，我对于动态规划算法有了更深入的理解和认识。在我看来，动态规划是通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。动态规划常常适用于有重叠子问题和最优子结构性质的问题。它的主要思想就是：若要解一个给定问题，我们需要解其不同部分（即子问题），再合并子问题的解以得出原问题的解。通常许多子问题非常相似，为此动态规划法试图仅仅解决每个子问题一次，从而减少计算量：一旦某个给定子问题的解已经算出，则将其记忆化存储，以便下次需要同一个子问题解之时直接查表。动态规划最重要是写出递归表达式，根据递归表达式来写程序。总之，这次实验虽然较难，但是对于我来说收获是非常大的。

A 最长公共子序列

```
#include <iostream>
#include <string>
using namespace std;
void LCSLength(int m, int n, string x, string y, int** c, int** b);
void LCS(int i, int j, string x, int** b);
int main()
{
    int a_len, b_len, c_len, d_len, ** c, ** b;
    string A, B, C, D;
    cout << "请输入A序列: ";
    cin >> A;
    cout << endl;
    cout << "请输入B序列: ";
    cin >> B;
    cout << endl;
    cout << "请输入C序列: ";
    cin >> C;
    cout << endl;
    cout << "请输入D序列: ";
    cin >> D;
    a_len = A.length();
    b_len = B.length();
    c_len = C.length();
    d_len = D.length();

    b = new int* [a_len + 1];
    c = new int* [a_len + 1];
    for (int i = 0; i < a_len + 1; i++)
    {
        b[i] = new int[b_len + 1];
        c[i] = new int[b_len + 1];
    }
    cout << "A序列和B序列的最长子序列为: ";
    LCSLength(a_len, b_len, A, B, c, b);
    LCS(a_len, b_len, A, b);
    cout << endl;

```

```
38      for (int i = 0; i < a_len + 1; i++)      //释放矩阵空间并将指针置
            空
39      {
40          delete[] b[i];
41          delete[] c[i];
42          b[i] = NULL;
43          c[i] = NULL;
44      }
45      delete[] b;
46      delete[] c;
47      b = NULL;
48      c = NULL;
49
50      b = new int* [c_len + 1];
51      c = new int* [c_len + 1];
52      for (int i = 0; i < c_len + 1; i++)
53      {
54          b[i] = new int[d_len + 1];
55          c[i] = new int[d_len + 1];
56      }
57      cout << "C序列和D序列的最长子序列为：";
58      LCSLength(c_len, d_len, C, D, c, b);
59      LCS(c_len, d_len, C, b);
60      cout << endl;
61
62      for (int i = 0; i < c_len + 1; i++)      //释放矩阵空间并将指针置
            空
63      {
64          delete[] b[i];
65          delete[] c[i];
66          b[i] = NULL;
67          c[i] = NULL;
68      }
69      delete[] b;
70      delete[] c;
71      b = NULL;
72      c = NULL;
73
74      b = new int* [a_len + 1];
```

```
75      c = new int* [a_len + 1];
76      for (int i = 0; i < a_len + 1; i++)
77      {
78          b[i] = new int[d_len + 1];
79          c[i] = new int[d_len + 1];
80      }
81      cout << "A序列和D序列的最长子序列为: ";
82      LCSLength(a_len, d_len, A, D, c, b);
83      LCS(a_len, d_len, A, b);
84      cout << endl;
85
86      for (int i = 0; i < a_len + 1; i++)      //释放矩阵空间并将指针置
            空
87      {
88          delete[] b[i];
89          delete[] c[i];
90          b[i] = NULL;
91          c[i] = NULL;
92      }
93      delete[] b;
94      delete[] c;
95      b = NULL;
96      c = NULL;
97
98      b = new int* [c_len + 1];
99      c = new int* [c_len + 1];
100     for (int i = 0; i < c_len + 1; i++)
101     {
102         b[i] = new int[b_len + 1];
103         c[i] = new int[b_len + 1];
104     }
105     cout << "C序列和B序列的最长子序列为: ";
106     LCSLength(c_len, b_len, C, B, c, b);
107     LCS(c_len, b_len, C, b);
108     cout << endl;
109
110     for (int i = 0; i < c_len + 1; i++)      //释放矩阵空间并将指针置
            空
111     {
```

```
        delete[] b[i];
        delete[] c[i];
        b[i] = NULL;
        c[i] = NULL;
    }
    delete[] b;
    delete[] c;
    b = NULL;
    c = NULL;
    return 0;
}
void LCSLength(int m, int n, string x, string y, int** c, int** b)
{
    int i, j;
    for (i = 1; i <= m; i++) c[i][0] = 0;    //初始化，Y[j]为空时
    for (i = 1; i <= n; i++) c[0][i] = 0;     //初始化，  X[i]为空时

    for (i = 1; i <= m; i++)              //两重循环，自下而上，
                                                //计算子问题{X(i
                                                   ), Y(j)}
        for (j = 1; j <= n; j++) {

            if (x[i] == y[j]) {                              //情况1
                c[i][j] = c[i - 1][j - 1] + 1;
                b[i][j] = 1;
            }
            else if (c[i - 1][j] >= c[i][j - 1]) {    //情况2
                c[i][j] = c[i - 1][j]; b[i][j] = 2;
            }
            else {
                c[i][j] = c[i][j - 1];             //情况3
                b[i][j] = 3;
            }
        }
}
void LCS(int i, int j, string x, int** b)
{
    if (i == 0 || j == 0) return;
    if (b[i][j] == 1) { LCS(i - 1, j - 1, x, b); cout << x[i]; }
```

```
150     /*第1种情况下，X(i)和Y(j)的最长公共子序列由X(i-1)和
151       Y(j-1)的解LCS(i-1, j-1, x, b)，加上位于最后的x[i]
152       组成*/
153     else if (b[i][j] == 2) LCS(i - 1, j, x, b);
154     else LCS(i, j - 1, x, b);      //其它2种情况下，原问题解
155                                                    //等于子问
                                                         题解
156 }
```

B 最长公共子序列，空间复杂度优化

```
1  #include<iostream>
2  #include<string>
3  #include<algorithm>
4  using namespace std;
5  int main()
6  {
7      ios::sync_with_stdio(false);
8      cin.tie(0);
9      int N, cur, pre;
10     string s1, s2;
11     int dp[2][1001];
12     unsigned len1, len2, i, j;
13     cin >> N;
14     while (N--)
15     {
16         cin >> s1 >> s2;
17         len1 = s1.size();
18         len2 = s2.size();
19         cur = 1;
20         pre = 0;
21         for (i = 0; i <= len2; i++)dp[0][i] = 0;
22         dp[1][0] = 0;
23         for (i = 1; i <= len1; i++)
24         {
25             for (j = 1; j <= len2; j++)
26             {
```

```
                if (s1[i - 1] == s2[j - 1])
                    dp[cur][j] = dp[pre][j - 1] + 1;
                else
                    dp[cur][j] = max(dp[cur][j - 1], dp[pre][j]);
            }
            swap(cur, pre);
        }
        cout << dp[pre][len2] << endl;
    }
    return 0;
}
```

C 最长递减子序列

```
#include <iostream>
#include <cstring>
constexpr auto MAX_NUM = 1000;
using namespace std;

int Fun(int aIn[], int pTable[], int nLen)
{
    int nMaxLen = 0;
    for (int i = nLen - 1; i >= 0; --i) {
        int nMax = 0;
        for (int j = i + 1; j < nLen; ++j) {
            if (aIn[j] <= aIn[i]) {
                nMax = nMax < pTable[j] ? pTable[j] : nMax;
            }
        }
        pTable[i] = 1 + nMax;
        nMaxLen = nMaxLen < pTable[i] ? pTable[i] : nMaxLen;
    }

    return nMaxLen;
}

void PrintMaxSubsequence(int aIn[], int pTable[], int nMaxLen, int
```

```
    nLen)
{
    for (int i = 0, j = 0; i < nLen; ++i) {
        if (pTable[i] == nMaxLen) {
            cout << aIn[i] << " ";
            nMaxLen--;
        }
    }
    cout << endl;
}

int main()
{
    int aIn[MAX_NUM], nLen = 0;
    for (; cin >> aIn[nLen];)
    {
        nLen++;
    }
    int* pTable = new int[nLen];
    memset(pTable, 0, nLen * sizeof(int));
    int nMaxLen = Fun(aIn, pTable, nLen);
    cout << "最长子序列长度为:  ";
    cout << nMaxLen << endl;
    cout << "具体子序列为:  ";
    PrintMaxSubsequence(aIn, pTable, nMaxLen, nLen);
    delete[] pTable;
    return 0;
}
```

D 最大子段和

```
#include <iostream>
#include <vector>
using namespace std;
int MaxSum(int n, vector<int> a);
int begin_seq = 0, end_seq = 0;
int main()
```

```
{
    vector<int> arr;
    int len = 0, temp;
    cout << "请输入序列: " << endl;
    arr.push_back(0);
    while (cin >> temp)
    {
        arr.push_back(temp);
        len++;
    }
    cout << "该序列的最大子段和为: " << MaxSum(len, arr) << endl;
    cout << "该子段开始位置为: " << begin_seq << "   结束位置为: " <<
        end_seq << endl;
    return 0;
}
int MaxSum(int n, vector<int> a)
{
    int sum = 0, b = 0, start_temp = 0;
    for (int i = 1; i <= n; i++)
    {
        b += a[i];
        if (b > sum)
        {
            sum = b;
            begin_seq = start_temp;
            end_seq = i;
        }
        if (b < 0)
        {
            b = 0;
            start_temp = i + 1;
        }
    }
    return sum;
}
```

E 凸多边形最优三角剖分 O(n3)

```cpp
#include <cstdio>
#include <cstdlib>
#define _USE_MATH_DEFINES
#include <math.h>
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>

constexpr auto INF = 100000000;
constexpr auto N = 100;
const double EARTH_RADIUS = 6378.137;//赤道半径km
using namespace std;
struct Point
{
    int ENODEBID;
    double LONGITUDE;
    double LATITUDE;
}point[N];

string Trim(string& str)
{
    //str.find_first_not_of(" \t\r\n"),在字符串str中从索引0开始，返回首次不匹配"\t\r\n"的位置
    str.erase(0, str.find_first_not_of(" \t\r\n"));
    str.erase(str.find_last_not_of(" \t\r\n") + 1);
    return str;
}

double rad(double LatOrLon)
{
    return LatOrLon * M_PI / 180.0;
}

double getCircumference(int point_num)
{
    double s = 0;
    for (int i = 0; i < point_num - 1; i++)
    {
```

```
39          double lat1 = point[i].LATITUDE, lng1 = point[i].LONGITUDE;
40          double lat2 = point[i + 1].LATITUDE, lng2 = point[i + 1].
               LONGITUDE;
41          double radLat1 = rad(lat1);
42          double radLat2 = rad(lat2);
43          double radlng1 = rad(lng1);
44          double radlng2 = rad(lng2);
45          //利用正弦余弦公式求距离
46          s += EARTH_RADIUS * acos(cos(radLat1) * cos(radLat2) * cos(
               radlng1 - radlng2) + sin(radLat1) * sin(radLat2));
47      }
48      return s;
49  }
50
51  double GetDistance2(int v_1, int v_2, int v_3)
52  {
53      //double lng1, double lat1, double lng2, double lat2
54      //将对应的经纬度转化为弧度
55      double lat1 = point[v_1].LATITUDE, lng1 = point[v_1].LONGITUDE;
56      double lat2 = point[v_2].LATITUDE, lng2 = point[v_2].LONGITUDE;
57      double lat3 = point[v_3].LATITUDE, lng3 = point[v_3].LONGITUDE;
58      double radLat1 = rad(lat1);
59      double radLat2 = rad(lat2);
60      double radLat3 = rad(lat3);
61      double radlng1 = rad(lng1);
62      double radlng2 = rad(lng2);
63      double radlng3 = rad(lng3);
64      //利用正弦余弦公式求距离
65      double s = 0;
66      s += EARTH_RADIUS * acos(cos(radLat1) * cos(radLat2) * cos(
           radlng1 - radlng2) + sin(radLat1) * sin(radLat2));
67      s += EARTH_RADIUS * acos(cos(radLat1) * cos(radLat3) * cos(
           radlng1 - radlng3) + sin(radLat1) * sin(radLat3));
68      s += EARTH_RADIUS * acos(cos(radLat3) * cos(radLat2) * cos(
           radlng3 - radlng2) + sin(radLat3) * sin(radLat2));
69      //s = s * 1000;
70      return s;
71  }
72
```

```
73  double MinWeightTriangulation(int n, double** t, int** s)
74  {
75
76      for (int i = 1; i <= n; i++)    t[i][i] = 0;
77
78      for (int r = 2; r <= n; r++)
79      {
80          for (int i = 1; i <= n - r + 1; i++)
81          {
82              int j = i + r - 1;
83
84              t[i][j] = t[i + 1][j] + GetDistance2(i - 1, i, j);
85              s[i][j] = i;
86
87              for (int k = i + 1; k < i + r - 1; k++)
88              {
89                  double u = t[i][k] + t[k + 1][j] + GetDistance2(i -
                        1, k, j);
90                  if (u < t[i][j])
91                  {
92                      t[i][j] = u;
93                      s[i][j] = k;
94                  }
95              }
96          }
97      }
98      return t[1][n - 1];
99  }
100
101 void TackBack(int i, int j, int** s)
102 {
103     if (i == j)
104         return;
105     else
106     {
107         TackBack(i, s[i][j], s);
108         TackBack(s[i][j] + 1, j, s);
109         cout << "三角剖分顶点: V" << i - 1 << ",V" << j << ",V" << s[
                i][j] << endl;
```

```
110         //cout << "[" << i - 1 << "," << j << "," << s[i][j] << "],";
111     }
112 }
113 
114 
115 int main()
116 {
117     ifstream fin("C:\\Users\\Lenovo\\Desktop\\data_3.csv"); //打开文件流操作
118     string line;
119     int num = 0;
120     getline(fin, line);
121     while (getline(fin, line))   //整行读取，换行符“\n”区分，遇到文件尾标志eof终止读取
122     {
123         //cout << "原始字符串: " << line << endl; //整行输出
124         istringstream sin(line); //将整行字符串line读入到字符串流istringstream中
125         vector<string> fields; //声明一个字符串向量
126         string field;
127         while (getline(sin, field, ',')) //将字符串流sin中的字符读入到field字符串中，以逗号为分隔符
128         {
129             fields.push_back(field); //将刚刚读取的字符串添加到向量fields中
130         }
131         if (fields[0] != "" && fields[1] != "" && fields[2] != "")
132         {
133             point[num].ENODEBID = atoi(Trim(fields[0]).c_str()); //清除掉向量fields中第一个元素的无效字符，并赋值给变量name
134             point[num].LONGITUDE = atof(Trim(fields[1]).c_str());
135             point[num].LATITUDE = atof(Trim(fields[2]).c_str());
136             num++;
137         }
138     }
139     double** t;
140     int** s;
141     t = new double* [(long long)num + 1];
142     s = new int* [(long long)num + 1];
```

```
143     for (int i = 0; i < num + 1; i++)
144     {
145         t[i] = new double[(long long)num + 1];
146         s[i] = new int[(long long)num + 1];
147     }
148     cout << (MinWeightTriangulation(num, t, s) - getCircumference(num
            )) / 2 + getCircumference(num) << "km" << endl;
149     TackBack(1, --num, s);
150     return 0;
151 }
```

F 凸多边形最优三角剖分 O(n2)

```
1  #include <cstdio>
2  #include <cstdlib>
3  #define _USE_MATH_DEFINES
4  #include <math.h>
5  #include <iostream>
6  #include <vector>
7  #include <fstream>
8  #include <sstream>
9
10 constexpr auto INF = 100000000;
11 constexpr auto N = 100;
12 const double EARTH_RADIUS = 6378.137;//赤道半径km
13 using namespace std;
14 struct Point
15 {
16     int ENODEBID;
17     double LONGITUDE;
18     double LATITUDE;
19 }point[N];
20
21 string Trim(string& str)
22 {
23     //str.find_first_not_of(" \t\r\n"),在字符串str中从索引0开始，返回
          首次不匹配"\t\r\n"的位置
```

```
    str.erase(0, str.find_first_not_of(" \t\r\n"));
    str.erase(str.find_last_not_of(" \t\r\n") + 1);
    return str;
}

double rad(double LatOrLon)
{
    return LatOrLon * M_PI / 180.0;
}

double getCircumference(int point_num)
{
    double s = 0;
    for (int i = 0; i < point_num - 1; i++)
    {
        double lat1 = point[i].LATITUDE, lng1 = point[i].LONGITUDE;
        double lat2 = point[i + 1].LATITUDE, lng2 = point[i + 1].
           LONGITUDE;
        double radLat1 = rad(lat1);
        double radLat2 = rad(lat2);
        double radlng1 = rad(lng1);
        double radlng2 = rad(lng2);
        //利用正弦余弦公式求距离
        s += EARTH_RADIUS * acos(cos(radLat1) * cos(radLat2) * cos(
           radlng1 - radlng2) + sin(radLat1) * sin(radLat2));
    }
    return s;
}

double GetDistance2(int v_1, int v_2, int v_3)
{
    //double lng1, double lat1, double lng2, double lat2
    //将对应的经纬度转化为弧度
    double lat1 = point[v_1].LATITUDE, lng1 = point[v_1].LONGITUDE;
    double lat2 = point[v_2].LATITUDE, lng2 = point[v_2].LONGITUDE;
    double lat3 = point[v_3].LATITUDE, lng3 = point[v_3].LONGITUDE;
    double radLat1 = rad(lat1);
    double radLat2 = rad(lat2);
    double radLat3 = rad(lat3);
```

```
61      double radlng1 = rad(lng1);
62      double radlng2 = rad(lng2);
63      double radlng3 = rad(lng3);
64      //利用正弦余弦公式求距离
65      double s = 0;
66      s += EARTH_RADIUS * acos(cos(radLat1) * cos(radLat2) * cos(
            radlng1 - radlng2) + sin(radLat1) * sin(radLat2));
67      s += EARTH_RADIUS * acos(cos(radLat1) * cos(radLat3) * cos(
            radlng1 - radlng3) + sin(radLat1) * sin(radLat3));
68      s += EARTH_RADIUS * acos(cos(radLat3) * cos(radLat2) * cos(
            radlng3 - radlng2) + sin(radLat3) * sin(radLat2));
69      //s = s * 1000;
70      return s;
71  }
72
73  double MinWeightTriangulation(int n, double** t, int** s)
74  {
75
76      for (int i = 1; i <= n; i++)    t[i][i] = 0;
77
78      for (int r = 2; r <= n; r++)
79      {
80          for (int i = 1; i <= n - r + 1; i++)
81          {
82              int j = i + r - 1;
83
84              t[i][j] = t[i + 1][j] + GetDistance2(i - 1, i, j);
85              s[i][j] = i;
86              int count = 0;
87              for (int k = i + 1; k < i + r - 1; k++)
88              {
89                  count++;
90                  double u = t[i][k] + t[k + 1][j] + GetDistance2(i -
                        1, k, j);
91                  if (u < t[i][j])
92                  {
93                      t[i][j] = u;
94                      s[i][j] = k;
95                  }
```

```
                if (count == 3)
                    break;
            }
        }
    }
    return t[1][n - 1];
}

void TackBack(int i, int j, int** s)
{
    if (i == j)
        return;
    else
    {
        TackBack(i, s[i][j], s);
        TackBack(s[i][j] + 1, j, s);
        cout << "三角剖分顶点: V" << i - 1 << ",V" << j << ",V" << s[
           i][j] << endl;
        //cout << "[" << i - 1 << "," << j << "," << s[i][j] << "],";
    }
}


int main()
{
    ifstream fin("C:\\Users\\Lenovo\\Desktop\\data_3.csv"); //打开文
       件流操作
    string line;
    int num = 0;
    getline(fin, line);
    while (getline(fin, line))   //整行读取, 换行符“\n”区分, 遇到文
       件尾标志eof终止读取
    {
        //cout << "原始字符串: " << line << endl; //整行输出
        istringstream sin(line); //将整行字符串line读入到字符串流
           istringstream中
        vector<string> fields; //声明一个字符串向量
        string field;
        while (getline(sin, field, ',')) //将字符串流sin中的字符读入
```

```
            到field字符串中，以逗号为分隔符
131         {
132             fields.push_back(field); //将刚刚读取的字符串添加到向量
                    fields中
133         }
134         if (fields[0] != "" && fields[1] != "" && fields[2] != "")
135         {
136             point[num].ENODEBID = atoi(Trim(fields[0]).c_str()); //清
                    除掉向量fields中第一个元素的无效字符，并赋值给变量name
137             point[num].LONGITUDE = atof(Trim(fields[1]).c_str());
138             point[num].LATITUDE = atof(Trim(fields[2]).c_str());
139             num++;
140         }
141     }
142     double** t;
143     int** s;
144     t = new double* [(long long)num + 1];
145     s = new int* [(long long)num + 1];
146     for (int i = 0; i < num + 1; i++)
147     {
148         t[i] = new double[(long long)num + 1];
149         s[i] = new int[(long long)num + 1];
150     }
151     cout << (MinWeightTriangulation(num, t, s) - getCircumference(num
            )) / 2 + getCircumference(num) << "km" << endl;
152     TackBack(1, --num, s);
153     return 0;
154 }
```

G 凸多边形最优三角剖分画图

```
1 import matplotlib.pyplot as plt
2 import csv
3
4 point_x = []
5 point_y = []
6 plt.figure('Line fig')
```

```
ax = plt.gca()
# 设置x轴、y轴名称
ax.set_xlabel('x')
ax.set_ylabel('y')
with open("C:\\Users\\Lenovo\\Desktop\\data_2.csv", "r", encoding='utf-8', ) as csvfile:
    reader = csv.reader(csvfile)
    num = 0
    for line in reader:
        if 1 <= num <= 21:
        # if 1 <= num <= 29:
            point_x.append(line[1])
            point_y.append(line[2])
        num = num + 1
# line = [[1, 3, 2], [3, 5, 4], [1, 5, 3], [1, 6, 5],
#         [0, 6, 1], [6, 8, 7], [9, 11, 10], [9, 12, 11], [8, 12, 9],
#         [6, 12, 8], [0, 12, 6], [13, 15, 14], [12, 15, 13], [15, 17, 16],
#         [15, 18, 17], [12, 18, 15], [0, 18, 12], [0, 19, 18], [0, 20, 19]]
line = [[1, 3, 2], [3, 5, 4], [1, 5, 3], [6, 8, 7], [6, 9, 8], [9, 11, 10], [9, 12, 11], [9, 13, 12], [13, 15, 14],
        [15, 17, 16], [13, 17, 15], [18, 20, 19], [17, 20, 18], [13, 20, 17], [9, 20, 13], [6, 20, 9], [5, 20, 6],
        [1, 20, 5], [0, 20, 1]]
# line = [[3, 5, 4], [2, 5, 3], [5, 7, 6], [5, 8, 7], [8, 10, 9], [5, 10, 8], [10, 12, 11], [10, 13, 12], [13, 15, 14],
#         [13, 16, 15], [10, 16, 13], [5, 16, 10], [17, 19, 18], [16, 19, 17], [19, 21, 20], [16, 21, 19], [21, 23, 22],
#         [21, 24, 23], [16, 24, 21], [5, 24, 16], [2, 24, 5], [24, 26, 25], [24, 27, 26], [24, 28, 27], [2, 28, 24],
#         [1, 28, 2], [0, 28, 1]]
# line = [[3, 5, 4], [5, 7, 6], [3, 7, 5], [7, 9, 8], [9, 11, 10], [7, 11, 9], [11, 13, 12], [13, 15, 14], [11, 15, 13],
#         [15, 17, 16], [17, 19, 18], [15, 19, 17], [19, 21, 20], [21, 23, 22], [19, 23, 21], [24, 26, 25], [24, 27, 26],
#         [24, 28, 27], [23, 28, 24], [19, 28, 23], [15, 28, 19], [11, 28, 15], [7, 28, 11], [3, 28, 7], [2, 28, 3],
#         [1, 28, 2], [0, 28, 1]]
```

```
point_xtp = [[]]
point_ytp = [[]]
for i in range(len(point_x) - 1):
    point_xtp.append([])
    point_xtp[i].append(float(point_x[i]))
    point_xtp[i].append(float(point_x[i + 1]))
    point_ytp.append([])
    point_ytp[i].append(float(point_y[i]))
    point_ytp[i].append(float(point_y[i + 1]))
print(point_xtp)
print(point_ytp)
for i in range(len(point_xtp) - 1):
    plt.plot(point_xtp[i], point_ytp[i], color='r')
    plt.scatter(point_xtp[i], point_ytp[i], color='b')
for i in range(len(line)):
    pointX = [float(point_x[line[i][0]]), float(point_x[line[i][1]])]
    pointY = [float(point_y[line[i][0]]), float(point_y[line[i][1]])]
    plt.plot(pointX, pointY, color='r')
    pointX = [float(point_x[line[i][1]]), float(point_x[line[i][2]])]
    pointY = [float(point_y[line[i][1]]), float(point_y[line[i][2]])]
    plt.plot(pointX, pointY, color='r')
    pointX = [float(point_x[line[i][2]]), float(point_x[line[i][0]])]
    pointY = [float(point_y[line[i][2]]), float(point_y[line[i][0]])]
    plt.plot(pointX, pointY, color='r')
plt.savefig("C:\\Users\\Lenovo\\Desktop\\21_n2.jpg")
plt.show()
```

H 背包问题

```
#include <iostream>
#include <string>
#include <algorithm>
#include <stack>
using namespace std;
void Knapsack(int v[], int w[], int c, int n, int** m);
int main()
{
```

```
    int** m, * value, * weight, bag_capacity, item_num;
    cout << "请输入背包容量: ";
    cin >> bag_capacity;
    cout << endl;
    cout << "请输入物品数量: ";
    cin >> item_num;
    cout << endl;
    m = new int* [item_num + 1];
    for (int i = 0; i < item_num + 1; i++)
    {
        m[i] = new int[bag_capacity + 1];
    }
    value = new int[item_num + 1];
    weight = new int[item_num + 1];
    cout << "请输入物品重量: ";
    for (int i = 0; i < item_num; i++)
    {
        cin >> weight[i + 1];
    }
    cout << endl;
    cout << "请输入物品价值: ";
    for (int i = 0; i < item_num; i++)
    {
        cin >> value[i + 1];
    }
    cout << endl;
    Knapsack(value, weight, bag_capacity, item_num, m);
    cout << "背包能装入物品的最大价值为: " << m[1][bag_capacity] <<
       endl;
    int left_weight = 0;
    int test_value = 0, test_weight = 0;
    cout << "背包中放入物品: ";
    for (int i = 1, j = bag_capacity; i < item_num; i++)//输出放入方
       案
    {
        if (m[i][j] == m[i + 1][j - weight[i]] + value[i])
        {
            cout << i << " ";
            test_value += value[i];
```

```
            test_weight += weight[i];
            j = j - weight[i];
        }
        left_weight = j;
    }
    if (m[item_num][left_weight] >= weight[item_num])
    {
        test_value += value[item_num];
        test_weight += weight[item_num];
        cout << item_num << " ";
    }
    cout << endl;
    cout << "此时背包的重量为: " << "  " << test_weight;
    delete[]value;
    value = NULL;
    delete[]weight;
    weight = NULL;
    for (int i = 0; i < item_num + 1; i++)       //释放矩阵空间并将指
        针置空
    {
        delete[] m[i];
        m[i] = NULL;
    }
    delete[] m;
    m = NULL;
    return 0;
}

void Knapsack(int v[], int w[], int c, int n, int** m)
{
    int JMax = w[n] - 1;

    for (int j = 0; j <= JMax; j++) m[n][j] = 0;
    for (int j = w[n]; j <= c; j++) m[n][j] = v[n];

    for (int i = n - 1; i >= 1; i--)
    {

        JMax = w[i] - 1;
```

```

        for (int j = 0; j <= JMax; j++) m[i][j] = m[i + 1][j];

        for (int j = w[i]; j <= c; j++)
            m[i][j] = max(m[i + 1][j], m[i + 1][j - w[i]] + v[i]);
    }
}
```

I 画拟合曲线

```
# import matplotlib.pyplot as plt
# import numpy as np
# x = [2000, 5000, 10000, 15000, 20000, 30000]
# y = [0.4, 1.6, 2, 4, 5.6, 8.6]
# plt.scatter(x, y)
# plt.show()
# parameter = np.polyfit(x, y, 1)
# p = np.poly1d(parameter)
# plt.scatter(x, y)
# plt.plot(x, p(x), color='g')
# plt.show()
# print(p)
# coding=utf-8
import pylab
import numpy as np
from scipy.optimize import curve_fit
import math

##使用curve_fit

import numpy as np
import matplotlib.pyplot as plt
from numpy import polyfit, poly1d


# 自定义函数 e指数形式
def func(x, a, b, c):
```

```
28      return a * x * np.log(x) + b * x + c
29
30
31  # 定义x、y散点坐标
32  # x = [10000, 40000, 250000, 490000, 1035096]
33  x_ = []
34  for i in range(10, 220, 10):
35      x_.append(int(i))
36  print(x_)
37  x_ = np.array(x_)
38  x = [29, 50, 73, 90, 147, 181, 207]
39  x = np.array(x)
40  num = [1, 3, 8, 11, 31, 46, 60]
41  y = np.array(num)
42
43  coeff = polyfit(x, y, 2)
44  print(coeff)
45  p = plt.plot(x, y, 'rx')
46  p = plt.plot(x_, coeff[0] * x_ * x_ + coeff[1] * x_ + coeff[2])
47  plt.xlabel('alg2')
48  plt.ylabel('time')
49  # p = plt.plot(x, y, 'b--')
50  plt.savefig("C:\\Users\\Lenovo\\Desktop\\32.jpg")
51  plt.show()
52  # plt.savefig("C:\\Users\\Lenovo\\Desktop\\remer.jpg")
53  # print(a * x * np.log(x) + b)
```