lab2实验报告

学号：PB20061338 姓名： 柯志伟

1 实验设备和环境

平台：windows
编程语言：C与C++

2 实验内容及要求

2.1 实验内容

实验2.1: 求矩阵链乘最优方案

1．n个矩阵链乘，求最优链乘方案，使链乘过程中乘法运算次数最少。
2．n的取值5，10，15，20，25，矩阵大小见2_1_input.txt。
3．求最优链乘方案及最少乘法运算次数，记录运行时间，画出曲线分析。
仿照P214 图15-5，打印n=5时的结果并截图

实验2.2: 求最长公共子序列

1．给定两个序列X、Y，求出这两个序列的最长公共子序列（某一个即可）。
2．X，Y序列由A、B、C、D四种字符构成,序列长度分别取10、15、20、25、30，见2_2_input.txt。
3．打印最长公共子序列，记录运行时间，画出曲线分析

2.2 实验要求

1．完成实验2.1和实验2.2,并使用给定不同规模数据分析程序运行的渐进时间复杂度
2．撰写实验报告,包含实验设备和环境、实验内容及要求、方法和步骤、结果与分析(比较实际复杂度和理论复杂度是否相同，给出分析)

2.3 方法和步骤

矩阵链乘

1. 获取所有输入数据

```cpp
ifstream fin("E:\\Savefiles\\Algorithm\\lab2\\lab2\\ex1\\input\\2_1_input.txt",ios::in);

LARGE_INTEGER   StartingTime, EndingTime, ElapsedMicroseconds;
LARGE_INTEGER   Frequency;

// 准备数据
vector<int> lens;
vector<vector<long long> > mats;
vector<double> costs;
vector<string> results;
vector<long long> chain_times;
int count = 0;

while(! fin.eof()) {
    int tmp_int1;
    int tmp_int2;
    fin >> tmp_int1;
    count ++;
    lens.push_back(tmp_int1);
    vector<long long> tmp_vec;
    while(tmp_int1 >= 0) {
        fin >> tmp_int2;
        tmp_vec.push_back(tmp_int2);
        tmp_int1-- ;
    }
    mats.push_back(tmp_vec);
}
```

2. 实现矩阵链乘算法

```cpp
void matrix_chain_order(vector<long long>&p,vector<vector<long long> >&m,vector<vector<int> >&s){
    int n = p.size() -1;
    for(int i=0;i<n;i++){
        m[i][i] = 0;
    }
    for(int l=2;l<=n;l++){
        for(int i=0;i<n-l+1;i++){
            int j = i+l-1;
            m[i][j] = Inifity;
            for(int k=i;k<=j-1;k++){
                long long q = m[i][k] + m[k+1][j] +p[i]*p[k+1]*p[j+1];
                if (q<0)
                    cout << " error";
                if (q< m[i][j] ){
                    m[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
    }
}
void print_optimal_parens(vector<vector<int> >&s,int i,int j, string& result){
    if (i==j){
        result.push_back('A');
        result = result + std::to_string(i+1);
    }
    else {
        result.push_back('(');
        print_optimal_parens(s,i,s[i][j], result);
        print_optimal_parens(s,s[i][j]+1,j, result);
        result.push_back(')');
    }
}
```

3. 运行程序,衡量程序运行时间,保存结果

```
    for(int repeat = 0; repeat < 10000; repeat++) {
        for(int i=0; i< count; i++) {
            m.clear();
            s.clear();
            m.resize(lens[i]);
            s.resize(lens[i]);
            for(int j=0;j< lens[i]; j++) {
                m[j].resize(lens[i]);
                s[j].resize(lens[i]);
            }

            if(repeat == 0) {
                QueryPerformanceFrequency(&Frequency);
                QueryPerformanceCounter(&StartingTime);
                matrix_chain_order(mats[i], m, s);
                QueryPerformanceCounter(&EndingTime);
                ElapsedMicroseconds.QuadPart = EndingTime.QuadPart - StartingTime.QuadPart;
                double time_cost = (ElapsedMicroseconds.QuadPart * 1000.0) / Frequency.QuadPart;
                costs.push_back(time_cost);
                string result;
                print_optimal_parens(s, 0, lens[i]-1,result);
                results.push_back(result);
                chain_times.push_back(m[0][lens[i]-1]);
            }
            else {
                QueryPerformanceFrequency(&Frequency);
                QueryPerformanceCounter(&StartingTime);
                matrix_chain_order(mats[i], m, s);
                QueryPerformanceCounter(&EndingTime);
                ElapsedMicroseconds.QuadPart = EndingTime.QuadPart - StartingTime.QuadPart;
                double time_cost = (ElapsedMicroseconds.QuadPart * 1000.0) / Frequency.QuadPart;
                costs[i] = (costs[i] + time_cost) / 2.0 ;
            }
        }
    }
```

程序输出结果

```
e:\Savefiles\Algorithm\lab2\ex1\src\mcm.exe
Minimum chain times: 154865959097238
Scheme: (A1((A2A3)A4)A5))
matrix scale: 5 time costs: 0.000489026
Minimum chain times: 42524697503391
Scheme: ((A1A2)(((((((A3A4)A5)A6)A7)A8)A9)A10))
matrix scale: 10 time costs: 0.00303599
Minimum chain times: 5400945319618
Scheme: ((((((((((A1A2)A3)A4)A5)A6)A7)A8)A9)A10)A11)A12)A13)A14)A15)
matrix scale: 15 time costs: 0.00942401
Minimum chain times: 319329979644400
Scheme: ((A1(A2(A3(A4(A5(A6(A7(A8(A9(A10(A11(A12(A13(A14A15)))))))))))))))(((((A16A17)A18)A19)A20))
matrix scale: 20 time costs: 0.0196768
Minimum chain times: 574911761218280
Scheme: ((A1(A2(A3(A4(A5(A6(A7(A8(A9(A10A11)))))))))))(((((((((((A12A13)A14)A15)A16)A17)A18)A19)A20)A21)A22)A23)A24)A25
))
matrix scale: 25 time costs: 0.037236
```

规模为5时运行结果

```
154865959097238
128049683226820 138766801119366
74062781976714 105723424955724 183439291324068
15903764653528 43981152513978 119490227350806 120958281818244
0 0 0 0 0


0
0 3
0 2 3
0 1 2 3
```

最长公共子序列

1. 获取所有输入数据

```
ifstream fin;
fin.open("E:\\Savefiles\\Algorithm\\lab2\\ex2\\input\\2_2_input.txt", ios::in);

int count = 0;

vector<int> lens;
vector<string> strs;
vector<string> common_strs;

LARGE_INTEGER   StartingTime, EndingTime, ElapsedMicroseconds;
LARGE_INTEGER   Frequency;


vector<double> costs;

// 准备数据
while(! fin.eof()) {
    count ++;
    int tmp_int;
    fin >> tmp_int;
    lens.push_back(tmp_int);
    string tmp_str;
    fin >> tmp_str;
    strs.push_back(tmp_str);
    fin >> tmp_str;
    strs.push_back(tmp_str);
}

// 计算最长公共子序列并衡量时间性能
```

2. 实现最长公共子序列算法

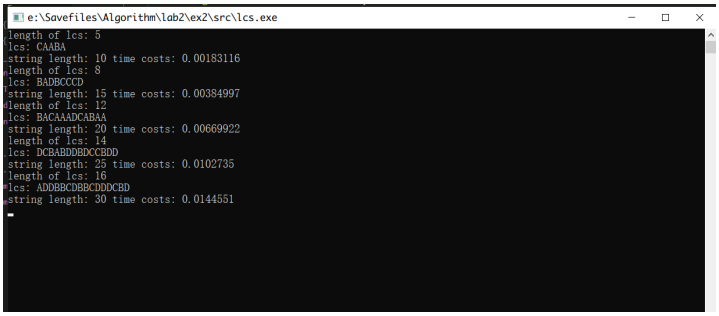
```
void lcs_length(string& X, string& Y,vector<vector<int> >&c, vector<vector<char> >&b){
    int m = X.size();
    int n = Y.size();
    for(int i=1;i<=m;i++)       c[i][0] = 0;
    for(int j=0;j<=n;j++)       c[0][j] = 0;
    for(int i=1;i<= m;i++){
        for(int j=1;j<= n;j++){
            if (X[i-1] == Y[j-1]){
                c[i][j] = c[i-1][j-1]+1;
                b[i][j] = 'b';
            }
            else if(c[i-1][j] >= c[i][j-1]){
                c[i][j] = c[i-1][j];
                b[i][j] = 'u';
            }
            else {
                c[i][j] = c[i][j-1];
                b[i][j] = 'l';
            }
        }
    }
}
void print_lcs(vector<vector<char> >&b, string& X, int i,int j, string& commonstr){
    if(i==0 || j==0){
        return ;
    }
    if (b[i][j] == 'b'){
        print_lcs(b,X,i-1,j-1, commonstr);
        commonstr.push_back(X[i-1]);
    }
    else if(b[i][j] == 'u'){
        print_lcs(b,X,i-1,j, commonstr);
    }
    else {
        print_lcs(b,X,i,j-1, commonstr);
```

3. 运行程序,衡量程序运行时间,保存结果

```cpp
for(int repeat = 0; repeat < 10000 ;repeat++) {
    for(int i=0; i< count; i++){
        c.clear();
        b.clear();
        int m = lens[i];
        int n = lens[i];
        c.resize(m+1);
        b.resize(m+1);

        for(int j=0;j<=m;j++) {
            c[j].resize(n+1);
            b[j].resize(n+1);
        }
        if (repeat == 0) {
            QueryPerformanceFrequency(&Frequency);
            QueryPerformanceCounter(&StartingTime);
            lcs_length(strs[2*i],strs[2*i+1],c,b);
            QueryPerformanceCounter(&EndingTime);
            ElapsedMicroseconds.QuadPart = EndingTime.QuadPart - StartingTime.QuadPart;
            double time_cost = (ElapsedMicroseconds.QuadPart * 1000.0) / Frequency.QuadPart;
            costs.push_back(time_cost);
            string commonstr;
            print_lcs(b,strs[2*i],m,n, commonstr);
            common_strs.push_back(commonstr);
        }
        else {
            QueryPerformanceFrequency(&Frequency);
            QueryPerformanceCounter(&StartingTime);
            lcs_length(strs[2*i],strs[2*i+1],c,b);
            QueryPerformanceCounter(&EndingTime);
            ElapsedMicroseconds.QuadPart = EndingTime.QuadPart - StartingTime.QuadPart;
            double time_cost = (ElapsedMicroseconds.QuadPart * 1000.0) / Frequency.QuadPart;
            costs[i] = (costs[i] + time_cost) / 2.0 ;
        }
```

程序输出结果



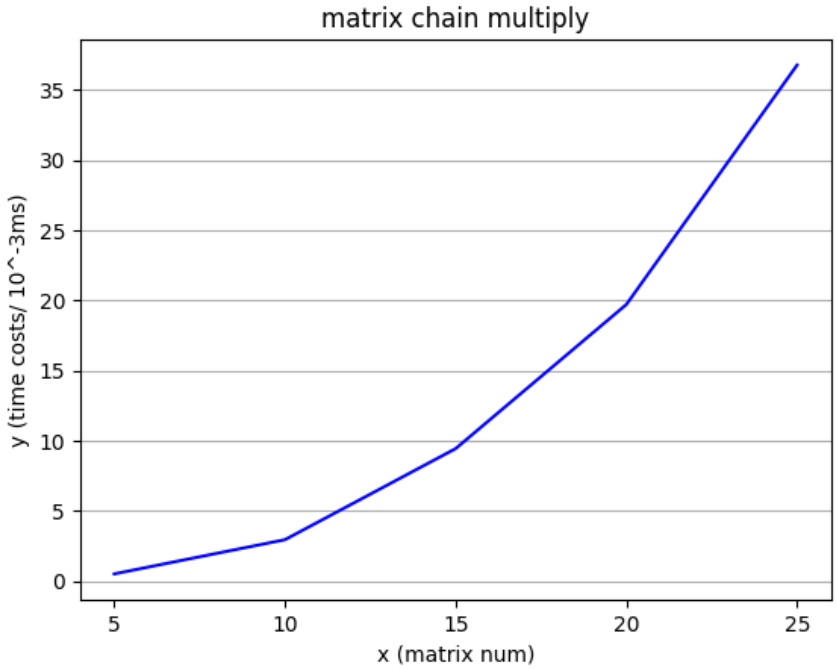
2.4 结果与分析

矩阵链乘

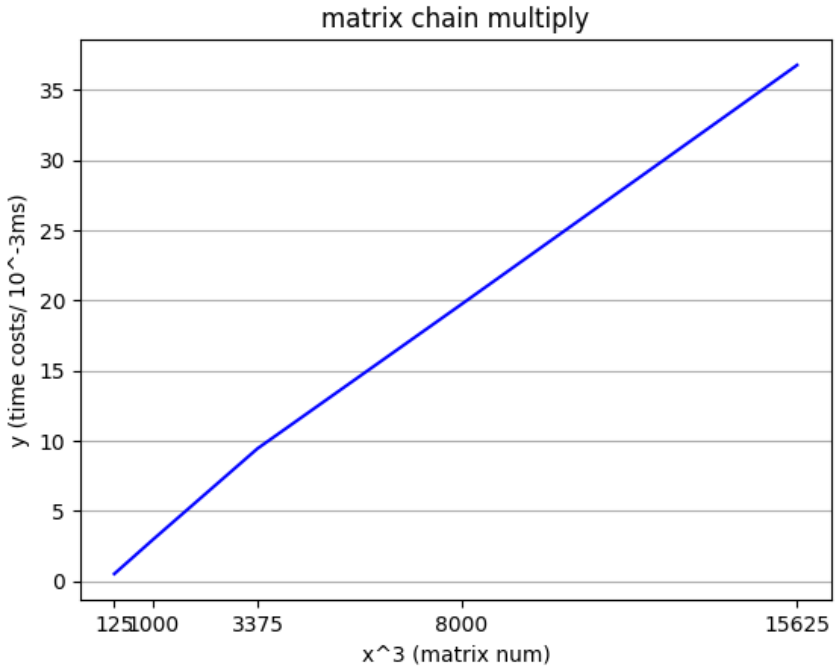
运行时间：

数据规模(矩阵数)	运行时间(ms)
5	0.000489026
10	0.00303599
15	0.00942401
20	0.0196768
25	0.037236

时间复杂度趋势图：


matrix chain multiply
y (time costs/ 10^-3ms)
x (matrix num)
0
5
10
15
20
25
30
35
5
10
15
20
25

使用n^3分析：


matrix chain multiply
y (time costs/ 10^-3ms)
x^3 (matrix num)
0
5
10
15
20
25
30
35
125
1000
3375
8000
15625

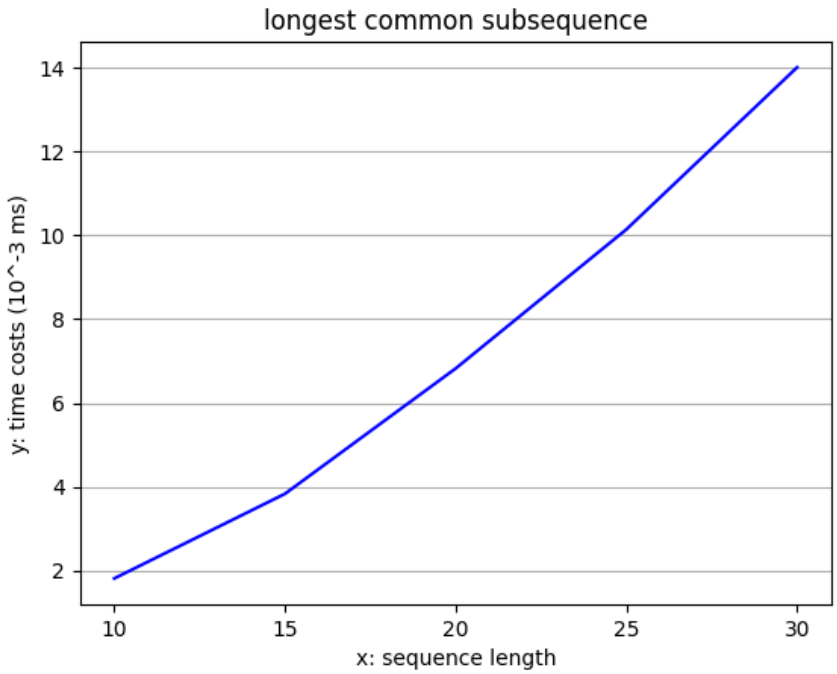
实验分析：分析矩阵链乘动态规划算法程序可知,程序设计三重循环,总的时间复杂度为O(n^3),通过实验结果,算法运行时间随输入数据规模的增大呈非线性变化, 当使用n^3进行分析作图时,效果较好,符合矩阵链乘动态规划算法O(n^3)的时间复杂度

最长公共子序列

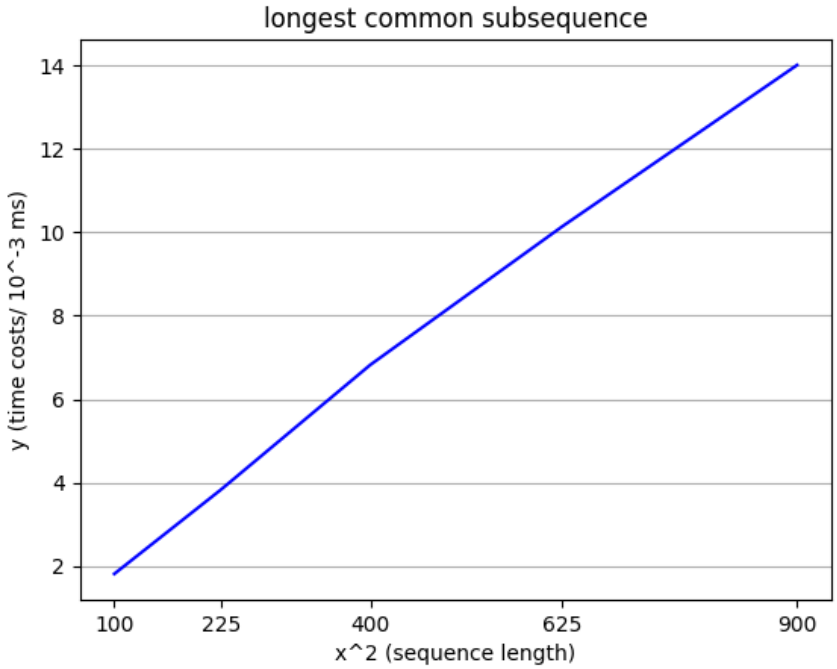
运行时间：

数据规模(序列长)	运行时间(ms)
10	0.00183116
12	0.00384997
20	0.00669922
25	0.0102735
30	0.0144551

时间复杂度趋势图：



使用n^2分析：



实验分析：分析最长公共子序列动态规划算法可知,程序包含两重循环,总的时间复杂度为O(mn),其中m,n分别为两个串的长度, 本次实验中m=n,故为O(n^2),通过实验结果,算法运行时间随输入数据规模的增大呈非线性变化,当使用n^2进行分析作图时,效果较好,符合最长公共子序列动态规划算法O(n^2)的时间复杂度