# 2111033\_H8\_Q1

姓名：艾明旭 学号：2111033 专业：信息安全

1. 枚举法和DP方法的时间对比分析

**1.源代码：**

#include<iostream>

#include<stack>

#include<queue>

#include<vector>

#include<cmath>

#include<Windows.h>

using namespace std;

#define m 5//序列规模，可更改

void Decimal2Binary(int n, stack<int>& BinaryResult) {//十进制转二进制的函数，用一个stack储存结果，用于枚举法

//例：4的二进制表示为100，但是按照取余法获得的结果在栈中保存的顺序从栈底到栈顶为001

if (!n) {//若n本身就为0.直接压入0后返回

BinaryResult.push(0);

return;

}

while (n) {//只要n不为0，就继续除

BinaryResult.push(n % 2);//将余数压入栈中

n = n / 2;

}

return;

}

bool isSubsequence(string S, queue<char> s) {//判断一个队列s中储存的序列是否为S的subsequence，用于枚举法

int j = 0;

while (!s.empty()) {//只要此时s队列不为空，就继续循环

if (j == m)//若此时已经遍历到S的末尾，但s中还有元素未匹配，则不为子序列

return false;

if (S[j] == s.front())//若子序列此时开头元素出现，就弹出队头元素，从下一个开始判断

s.pop();

j++;

}

return true;

}

void LongestCommonSubsequenceProblem\_EnumerationMethod(string A,string B) {//EnumerationMethod解决最长公共子序列问题（Brute-Force）

stack<int> BinaryResult;//用于产生二进制，决定选择子序列元素

int maxL = -1;//保存最大序列长度

queue<char> maxSub;//保存最大序列长度此时对应的序列

for (int i = 0; i < pow(2, m); i++) {//从[0,2^m），依次将十进制转为二进制数，对应所选择的子序列元素

//使用右对齐的匹配策略

//比如对于xyz，产生随机二进制数为101，即对应xz;产生随机二进制数10，即y

Decimal2Binary(i, BinaryResult);

int j = m-BinaryResult.size();//为了符合右对齐

queue<char> subA;//由于判断是否为子序列需要从头元素判断，符合FIFO的特性，因此采用队列形式存储

while (!BinaryResult.empty()) {

if (BinaryResult.top())//若此时栈顶的元素为1，说明选择j位置的A元素

subA.push(A[j]);

BinaryResult.pop();

j++;

}

int size = subA.size();

if (isSubsequence(B, subA)&&size>maxL) {//若subA为B的子序列，且长度大于最大子序列

maxL = size;

maxSub = subA;

}

}

while (!maxSub.empty()) {

cout << maxSub.front();

maxSub.pop();

}

cout << endl;

return;

}

void LongestCommonSubsequenceProblem\_DynamicProgramming(string A,string B) {//DynamicProgramming解决最长公共子序列问题

int\*\* C = new int\* [m + 1];//二维动态数组C[i,j],含义为A中前i个元素和B中前j个元素的最大子序列长度，共有[0,m]共m+1行

vector<char> maxSub[m+1][m+1];//保存每个C[i][j]位置的最大子序列

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

C[i] = new int[m + 1];//每行有m+1个元素即m+1列

for (int i = 0; i <= m; i++) {//当j=0时，A部分中只有为空集时才为B部分的子集，即最大子序列长度为0

C[i][0] = 0;

vector<char> v;//并用一个空的vector代表空集填充

maxSub[i][0] = v;

}

for (int j = 0; j <= m; j++) {//当i=0时，A中部分为空集，一定为B部分的子集，但最大子序列长度只能为0

C[0][j] = 0;

vector<char> v;//并用一个空的vector代表空集填充

maxSub[0][j] = v;

}

for (int i = 1; i <= m; i++) {

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

if (A[i - 1] == B[j - 1]) {//由于A和B的string类型从[0,m)。因此对应下标应为m-1

C[i][j] = C[i - 1][j - 1] + 1;

vector<char> v = maxSub[i - 1][j - 1];

v.push\_back(A[i - 1]);//在maxSub[i - 1][j - 1]这个位置的子序列基础上，再push进一个此时的A[i-1]或B[j-1]

maxSub[i][j] = v;

}

else {

C[i][j] = max(C[i][j - 1], C[i - 1][j]);

if (C[i][j - 1] > C[i - 1][j])

maxSub[i][j] = maxSub[i][j - 1];

else

maxSub[i][j] = maxSub[i - 1][j];

}

}

}

vector<char> resultSub = maxSub[m][m];//对应C[m][m]的最长子序列

for (int i = 0; i < C[m][m]; i++) {

cout << resultSub[i];

}

cout << endl;

return;

}

int main() {

string A, B;

cin >> A >> B;

LARGE\_INTEGER t1, t2, tc;

QueryPerformanceFrequency(&tc);

QueryPerformanceCounter(&t1);

LongestCommonSubsequenceProblem\_EnumerationMethod(A,B);//枚举法解决最长公共子序列问题

LongestCommonSubsequenceProblem\_DynamicProgramming(A,B);//动态规划法解决最长公共子序列问题

QueryPerformanceCounter(&t2);

double time = (double)(t2.QuadPart - t1.QuadPart) / (double)tc.QuadPart;

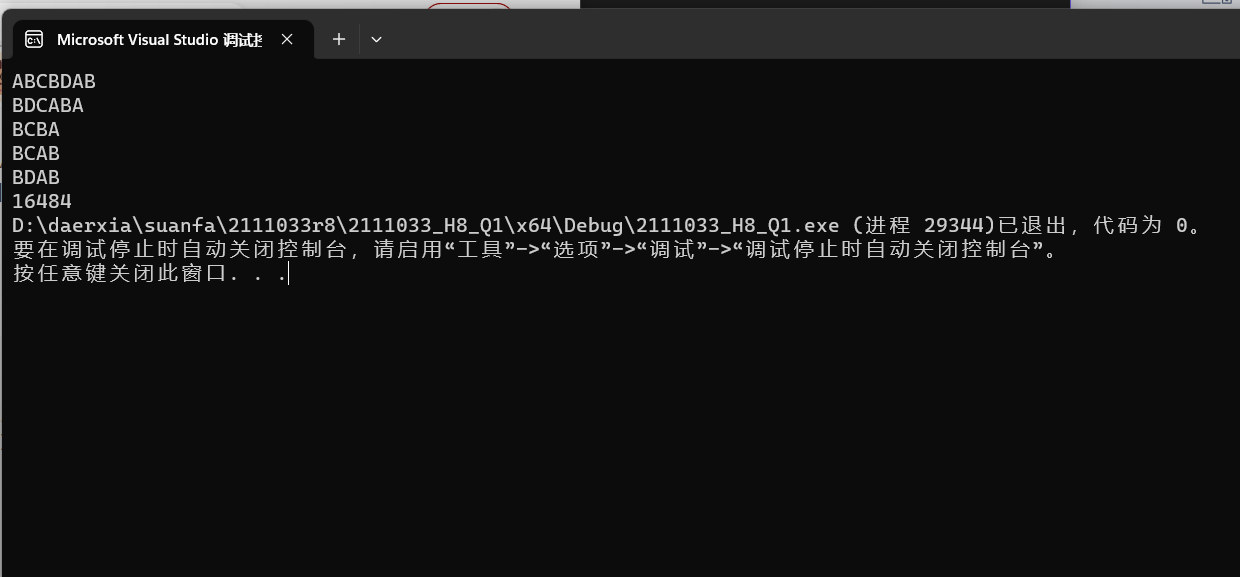
cout << "time = " << time << "s" << endl; //输出时间（单位：ｓ）

return 0;

}

**PS：时间测试采用QueryPerformanceCounter()函数**

**首先对长度不同的序列进行一定的测试**



1. 序列规模为5——
2. 产生随机样例：（大小写字母+数字）

在Python的Jupter Notebook中,使用random和string两个包随机生成规模大小为5输入样例A字符串，结果如上图所示：**5RgtV**。

由于若继续随机生成序列，有子序列可能性较低，因此人为规定另一个输入样例B字符串为：**q5gKV**。

二者最长公共子序列应为**5gV**。

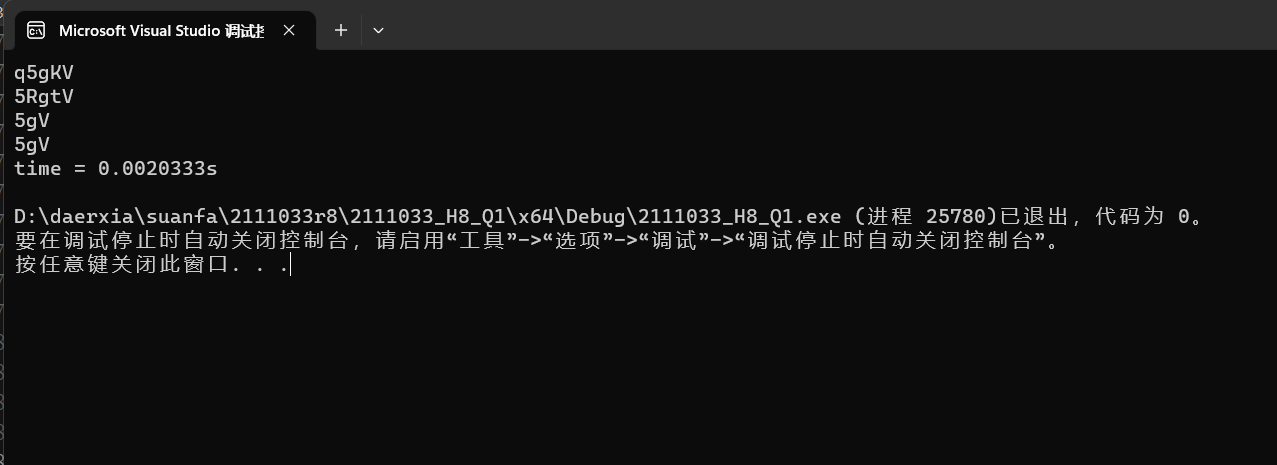
1. 枚举法void LongestCommonSubsequenceProblem\_EnumerationMethod(string A,string B) 测试：



**所用时为0.0005997s。**

**此时最长子序列结果与预期一致，校验了枚举法的正确性。**

1. DP法void LongestCommonSubsequenceProblem\_DynamicProgramming(string A,string B) 测试：



**所用时为0.0020333s。**

**同时结果也和枚举法一样，校验了DP方法的正确性。**

1. **字符串规模为5测试时间对比：**

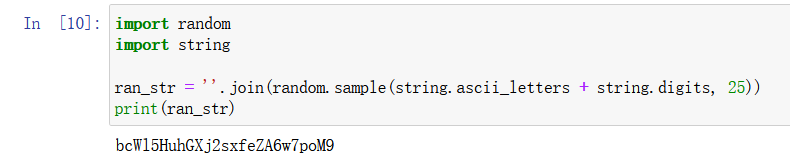
枚举法测试时间为：0.0005997s

DP法测试时间为：0.0020333s

**可以看到在规模较小时，DP法相对于枚举法只能节省较少的时间，差异并不明显，但已经能提升一定的时间性能了。**

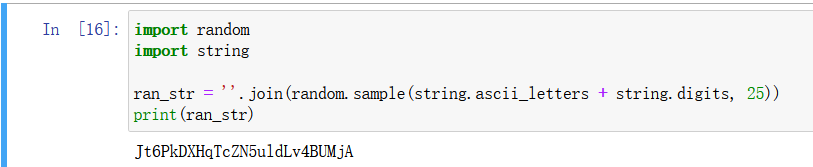
1. 序列规模为25——

（1）产生随机样例：（大小写字母+数字）



在Python的Jupter Notebook中,使用random和string两个包随机生成规模大小为25输入样例A字符串，结果如上图所示：

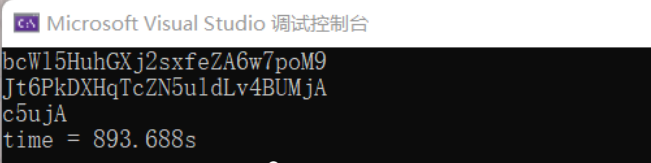
**bcWl5HuhGXj2sxfeZA6w7poM9**。



同理，随机生成规模为25的另一个输入样例B字符串，结果如上图所示：**Jt6PkDXHqTcZN5uldLv4BUMjA**。

二者最长公共子序列应为：**c5ujA**。

1. 枚举法void LongestCommonSubsequenceProblem\_EnumerationMethod(string A,string B) 测试：



**所用时为893.688s。**

**结果再次与预期一致，再次校验了枚举法的正确性。**

（3）DP法void LongestCommonSubsequenceProblem\_DynamicProgramming(string A,string B) 测试：



**所用时为0.0012543s。**

**同时结果也与枚举法一致，再次校验了DP方法的正确性。**

1. **字符串规模为25测试时间对比：**

枚举法测试时间为：893.688s

DP法测试时间为：0.0012543s

**可以看到在规模较大时，DP方法相比于枚举法的时间性能有指数级别的提升，因此更加体现了DP动态规划对于减小时间复杂方面的重要性，运行效率提升非常明显。**