**算法第八次作业编程练习报告**

姓名：吴静 学号：2113285 班级：信息安全二班

##### 编程练习1——最长子序列问题

1.源码部分

#include<iostream>

#include<time.h>

using namespace std;

int m;

int C[50][50] = { 0 };

string rand\_str(const int len)

{

string str;

//循环向字符串中添加随机生成的字符

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

{

char c = 'a' + rand() % 26;

str.push\_back(c);

//push\_back()是string类尾插函数,这里插入随机字符c

}

return str;

}

void badd(int used[],int n) {

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

if (used[i] != 1) {

used[i] = 1;

return;

}

else

used[i] = 0;

}

return;

}

void LCS(int n1, int n2,string str1,string str2) {

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

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

if (str1[i] == str2[j])

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

else

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

}

}

}

int main() {

clock\_t start1, finish1, start2, finish2;

cout << "请输入序列长度：" << endl;

cin >> m;

string str1;

string str2;

cout << "随机产生的字符串为：" << endl;

str1 = rand\_str(m);

str2 = rand\_str(m);

cout << str1 << endl << str2 << endl;

str1 = '0' + rand\_str(m);

str2 = '0' + rand\_str(m);

//cout << "请输入两字符串：" << endl;

//cin >> str1 >> str2;

//str1 = '0' + str1;

//str2 = '0' + str2;

cout << "动态规划法算出的最大子序列中元素为：" << endl;

start2 = clock();

LCS(m, m, str1, str2);

cout << C[m][m] << endl;

finish2 = clock();

cout << "动态规划法耗时：" << finish2 - start2 << "s" << endl;

int maxstring = 0;

int used[30] = { 0 };

start1 = clock();

string ss;

for (int i = 0; i < pow(2, str1.length() - 1);i++) {

badd(used, str1.length() - 1);

int temp = 0;

string s;

for (int j = 0; j < str1.length(); j++)

if (used[j] == 1)

s += str1[j];

int ptr = 1;

int total = 0;

for (int j = 0; j < s.length(); j++) {

while (ptr < str2.length())

{

if (s[j] == str2[ptr]) {

ptr++;

if (j == s.length() - 1)

total = 1;

break;

}

ptr++;

}

}

if (total==1) {

if (maxstring < (int)s.length())

ss = s;

maxstring = max(maxstring, (int)s.length());

}

}

cout << "穷举法算出的最大子序列为：" << ss << endl;

finish1 = clock();

cout << "穷举法算出的最大子序列中元素为：" << endl;

cout << maxstring << endl;

cout << "穷举法耗时：" << finish1 - start1 << "s" << endl;

}

2.说明部分

（1）字符子序列：字符串中不一定连续但先后顺序一致的n个字符，即可以去掉字符串中的部分字符，但不可改变其前后顺序。

（2）穷举法：

* 找出A字符串中所有可能的子序列（2n个）

此时可利用1和0来表示“有该元素“和”没有该元素“，用used数组来表示是否有该元素，模拟二进制加法，代码实现如下：

void badd(int used[],int n) {

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

if (used[i] != 1) {

used[i] = 1;

return;

}

else

used[i] = 0;

}

return;

}

* 对于A的每一个子序列，判断其是否为B的一个子序列

此时可以对于A的每一个子序列，用指针的方式对B进行一个一个查找，如果B按顺序查找含有A的子序列的所有元素，则他进入考察范围内，所有考察范围内元素个数最多的即为最长子序列，关键代码实现如下：

for (int i = 0; i < pow(2, str1.length() - 1);i++) {

badd(used, str1.length() - 1);

int temp = 0;

string s;

for (int j = 0; j < str1.length(); j++)

if (used[j] == 1)

s += str1[j];

int ptr = 1;

int total = 0;

for (int j = 0; j < s.length(); j++) {

while (ptr < str2.length())

{

if (s[j] == str2[ptr]) {

ptr++;

if (j == s.length() - 1)

total = 1;

break;

}

ptr++;

}

}

if (total==1) {

if (maxstring < (int)s.length())

ss = s;

maxstring = max(maxstring, (int)s.length());

}

}

（3）动态规划法

设两个字符串A，B长度分别为n，m，令C[i,j]表示A的前i个字符和B的前j个字符的最长公共子序列。

首先，C[i,0]和C[0,j]均为0（进行初始化）

其次，明确当ai=bi且i,j均大于0时，C[i,j]=C[i-1,j-1]+1；

同时，当ai!=bi且i,j均大于0时,C[i,j]=max{C[i,j-1],C[i-1,j]}，即前面位数的最长子序列。

综上，

0 i==0||j==0

C[i,j]= C[i-1,j-1]+1 i>0&&j>0&&ai==bj

max{C[i,j-1],C[i-1,j]} i>0&&j>0&&ai!=bi

该过程关键代码实现：

void LCS(int n1, int n2,string str1,string str2) {

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

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

if (str1[i] == str2[j])

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

else

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

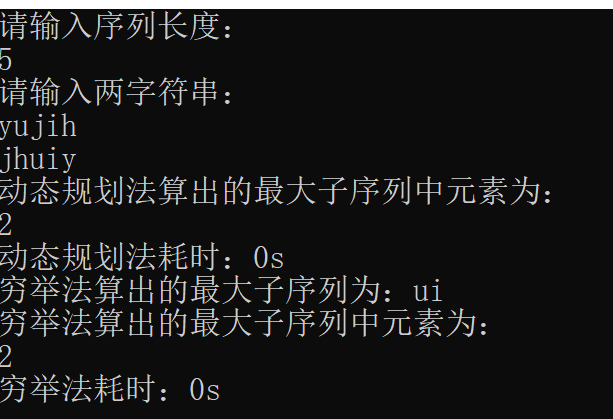
}

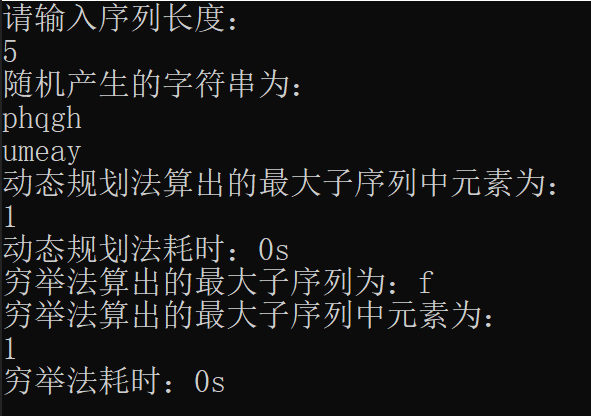
}

}

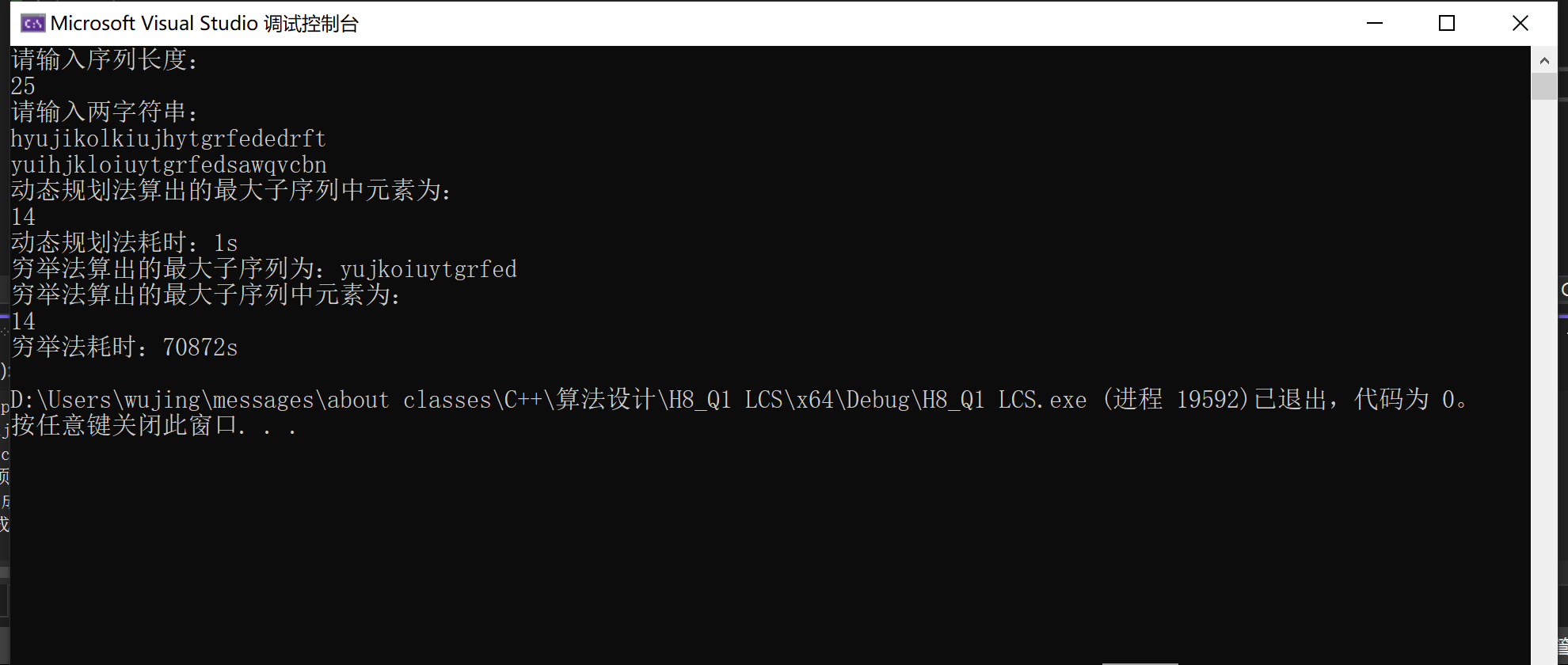
3.运行示例

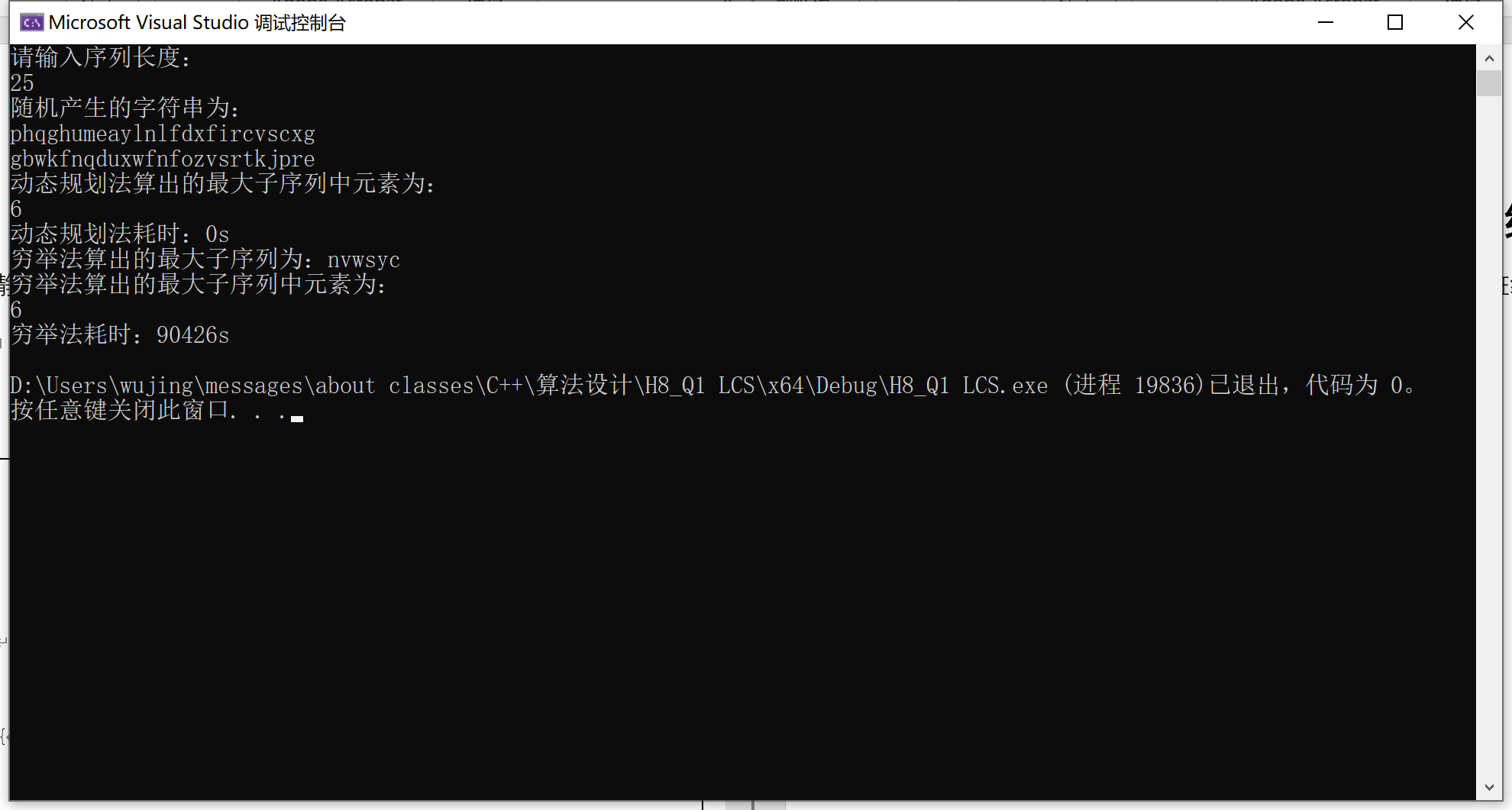
（1）序列长度为5时



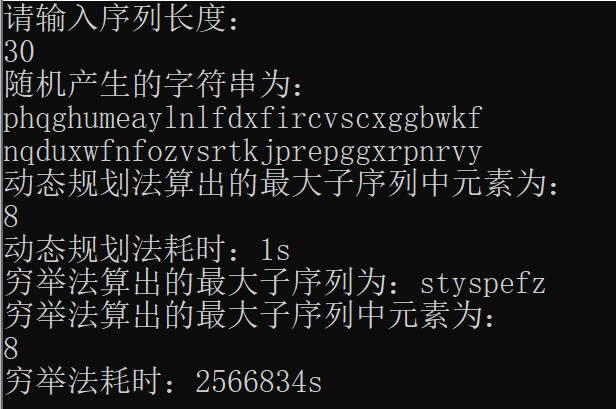


（2）序列长度为25时





（3）序列长度为30时



可以看出，随着序列长度的增加，动态规划法的耗时区别不大，但是穷举法的耗时随指数级别增加。

##### 编程练习2——高性能计算机任务分配问题（二次动态规划）

1.源码部分

#include<iostream>

#include<fstream>

#include<math.h>

using namespace std;

int nA, nB, p;

int ta[10] = { 0 }, tb[10] = { 0 }, ka[10] = { 0 }, kb[10] = { 0 };

int onenode[50][50][50] = { 0 };

int nodes[50][50][50] = { 0 };

void anode() {//onenode[z][i][j]为第z个节点完成i个A任务和j个B任务所要使用的最短时间

for (int z = 1; z <= p; z++) {

//节点序号为1-p

int P[30][30][2] = { 0 };

//节点i完成任务(a,b)所花的最短时间(最后完成的一项子任务是A类子任务)

//节点i完成任务(a,b)所花的最短时间(最后完成的一项子任务是B类子任务)

memset(P, 0x3F, sizeof(P));//初始化为最大值

P[0][0][0] = 0;

P[0][0][1] = 0;

for (int i = 0; i <= nA; i++) {

//总共可能有0-nA个A类任务

for (int j = 0; j <= nB; j++) {

//总共可能有0-nB个B类任务

for (int w = 1; w <= i; w++)

P[i][j][0] = min((P[i - w][j][1] + ta[z] + ka[z] \* w \* w), P[i][j][0]);

//Pi(a,b,0)=min(1<=w<=a){Pi(a-w,b,1)+ta+ka\*w\*w}

for (int w = 1; w <= j; w++)

P[i][j][1] = min((P[i][j - w][0] + tb[z] + kb[z] \* w \* w), P[i][j][1]);

//Pi(a,b,1)=min(1<=w<=a){Pi(a,b-w,0)+tb+kb\*w\*w}

onenode[z][i][j] = min(P[i][j][0], P[i][j][1]);

//节点i完成所分配的任务时,最后所完成的子任务不是A类子任务就是B类子任务.

}

}

}

}

void manynodes() {

memset(nodes, 0x3f, sizeof(nodes));

for (int i = 0; i <= nA; i++) {

for (int j = 0; j <= nB; j++) {

nodes[1][i][j] = onenode[1][i][j];

}

}

//nodes[a][i][j]：将任务分配给前a个节点，完成i个A任务和j个B任务所要的最短时间

for (int a = 2; a <= p; a++) {

//节点个数从2开始，一直到p（总共p个节点）

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

//总共可能有0-nA个A类任务

for (int j = 0; j <= nB; j++)

//总共可能有0-nB个B类任务

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

for (int l = 0; l <= j; l++)

nodes[a][i][j] = min(nodes[a][i][j], max(nodes[a - 1][i - k][j - l], onenode[a][k][l]));

//分配给第a个节点的任务是(k,l)

//前a-1个节点的任务是(a-k,j-l)，所要的最短时间为nodes[a-1][i-k][j-l]

//节点是并行完成任务的，所以最短时间即为所有节点完成任务的最短时间max{nodes[a-1][i-k][j-l],onenode[a][k][l]}

//而我们要求的是这些最短时间中的最短时间，即min{nodes[a][i][j],max{...}}

}

}

int main() {

ifstream infile(".//hpc.in.txt", ios::in);

if (!infile)

cout << "打开失败！" << endl;

int x = 3;

int xx[3];//存nA，nB,p三个变量

while (x) {

infile >> xx[3 - x];

x--;

}

nA = xx[0];

nB = xx[1];

p = xx[2];//依次赋值

for (int i = 1; i <= p; i++)

infile >> ta[i] >> tb[i] >> ka[i] >> kb[i];

infile.close();

anode();

manynodes();

ofstream outfile("hpc.out.txt");

outfile << nodes[p][nA][nB];

outfile.close();

cout << "已成功将答案写入文件hpc.out";

return 0;

}

2.说明部分

（1）对于文件的输入输出问题，采用fstream头文件下的ifstream和ofstream两个类，具体实现如下：

ifstream infile(".//hpc.in.txt", ios::in);

if (!infile)

cout << "打开失败！" << endl;

int x = 3;

int xx[3];//存nA，nB,p三个变量

while (x) {

infile >> xx[3 - x];

x--;

}

nA = xx[0];

nB = xx[1];

p = xx[2];//依次赋值

for (int i = 1; i <= p; i++)

infile >> ta[i] >> tb[i] >> ka[i] >> kb[i];

infile.close();

ofstream outfile("hpc.out.txt");

outfile << nodes[p][nA][nB];

outfile.close();

（2）思路：第一次动态规划旨在处理出每一个节点执行i个A任务和j个B任务的最短时间，第二次动态规划旨在探究如何分配任务使得完成全部任务的时间最短。

（3）对于第一次动态规划，用fi(a,b)表示第i个节点执行i个A任务和j个B任务的最短时间，这里根据题意又可以细分为两种情况：最后完成的一项子任务是A类任务，用Pi(a,b,0)表示；最后完成的一项子任务是B类任务，用Pi(a,b,1)表示，由于节点i完成所分配任务时，最后完成的子任务不是A类任务就是B类任务，所以，要求第i个节点执行i个A任务和j个B任务的最短时间，即为min{Pi(a,b,0),Pi(a,b,1)}，即

fi(a,b)= min{Pi(a,b,0),Pi(a,b,1)}

首先，如果A类任务和B类任务个数为0，则不需要耗费时间即可完成这些任务，即

P[0][0][0] = 0;

P[0][0][1] = 0;

（初始化）

其次，注意到题目所给条件：

* 从其他状态转入A类状态或者B类状态时要耗费启动时间ta或者tb
* 节点i连续处理x个A类子任务，则对应的执行时间为: t=kiA\*x2
* 节点i连续处理x个B类子任务，则对应的执行时间为: t=kiB\*x

对于Pi(a,b,0)，由于他最后处理的任务是A类任务，所以可以假设他最后连续处理了w个A类任务，于是总的处理时间等于处理a-w个A类任务和b个B类任务加上从B到A的启动时间和连续处理w个A类任务的执行时间的最小值，即：

Pi(a,b,0)=min(1<=w<=a){Pi(a-w,b,1)+tiA+kiA\*w2}

同理

Pi(a,b,1)=min(1<=w<=b){Pi(a,b-w,0)+tiB+kiB\*w2}

具体代码实现如下：

void anode() {//onenode[z][i][j]为第z个节点完成i个A任务和j个B任务所要使用的最短时间

for (int z = 1; z <= p; z++) {

//节点序号为1-p

int P[30][30][2] = { 0 };

//节点i完成任务(a,b)所花的最短时间(最后完成的一项子任务是A类子任务)

//节点i完成任务(a,b)所花的最短时间(最后完成的一项子任务是B类子任务)

memset(P, 0x3F, sizeof(P));//初始化为最大值

P[0][0][0] = 0;

P[0][0][1] = 0;

for (int i = 0; i <= nA; i++) {

//总共可能有0-nA个A类任务

for (int j = 0; j <= nB; j++) {

//总共可能有0-nB个B类任务

for (int w = 1; w <= i; w++)

P[i][j][0] = min((P[i - w][j][1] + ta[z] + ka[z] \* w \* w), P[i][j][0]);

//Pi(a,b,0)=min(1<=w<=a){Pi(a-w,b,1)+ta+ka\*w\*w}

for (int w = 1; w <= j; w++)

P[i][j][1] = min((P[i][j - w][0] + tb[z] + kb[z] \* w \* w), P[i][j][1]);

//Pi(a,b,1)=min(1<=w<=a){Pi(a,b-w,0)+tb+kb\*w\*w}

onenode[z][i][j] = min(P[i][j][0], P[i][j][1]);

//节点i完成所分配的任务时,最后所完成的子任务不是A类子任务就是B类子任务.

}

}

}

}

（4）对于第二次动态规划，要求解的问题为“有p个节点可以并行地完成任务，如何分配该（nA，nB）个任务使得全部任务的完成时间最短。

思路：设Ci(a,b)表示为将任务(a,b)分配给前i个节点，假设分配i个节点，第i个节点分配了w1个A类任务，w2个B类任务，则前i-1个节点应该分配a-w1个A类任务和b-w2个B类任务，由于这些节点是并行完成任务，所以所用的总时间应该是这两个部分的最长时间，即max{Ci-1(a-w1,b-w2),fi(w1,w2)}，由于所求的是最短时间，所以

Ci(a,b)=min{max{Ci-1(a-w1,b-w2),fi(w1,w2)}}

具体代码实现如下：

void manynodes() {

memset(nodes, 0x3f, sizeof(nodes));

for (int i = 0; i <= nA; i++) {

for (int j = 0; j <= nB; j++) {

nodes[1][i][j] = onenode[1][i][j];

}

}

//nodes[a][i][j]：将任务分配给前a个节点，完成i个A任务和j个B任务所要的最短时间

for (int a = 2; a <= p; a++) {

//节点个数从2开始，一直到p（总共p个节点）

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

//总共可能有0-nA个A类任务

for (int j = 0; j <= nB; j++)

//总共可能有0-nB个B类任务

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

for (int l = 0; l <= j; l++)

nodes[a][i][j] = min(nodes[a][i][j], max(nodes[a - 1][i - k][j - l], onenode[a][k][l]));

//分配给第a个节点的任务是(k,l)

//前a-1个节点的任务是(a-k,j-l)，所要的最短时间为nodes[a-1][i-k][j-l]

//节点是并行完成任务的，所以最短时间即为所有节点完成任务的最短时间max{nodes[a-1][i-k][j-l],onenode[a][k][l]}

//而我们要求的是这些最短时间中的最短时间，即min{nodes[a][i][j],max{...}}

}

}

3.运行示例

