实验3 分治算法实现

姓名：叶倩琳

班级：软工1706

完成时间：2019/04/03

**问题一**

1. **问题描述**

最长公共子序列问题（不记录b数组）

1. **代码实现**

#include <iostream>

#include<cstring>

using namespace std;

/\* 求最长公共子序列（没有保存数组b） \*/

#define MAXN 1000

char c[MAXN][MAXN];

//求最大长度

int LCS(int lenx,int leny,char\* x,char\* y){

int i,j;

//边界条件

for(i=0;i<=lenx;i++)

c[i][0]=0;

for(i=0;i<=leny;i++)

c[0][i]=0;

//递归求最优解

for(i=1;i<=lenx;i++){

for(j=1;j<=leny;j++){

if(x[i-1]==y[j-1]){

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];

}

}

return c[lenx][leny];

}

//递归打印公共子序列

void PrintLCS(int i,int j,char\* x,char\* y) {

if(i==0 || j==0)

return;

if(x[i-1]==y[j-1]){

PrintLCS(i-1,j-1,x,y);

cout<<x[i-1]; //若两个数相等，则打印该数

}

else if(c[i-1][j]>=c[i][j-1])

PrintLCS(i-1,j,x,y);

else

PrintLCS(i,j-1,x,y);

}

int main(int argc, char\*\* argv) {

//char\*x1="ABCBDAB",\*y1="BDCABA",\*x2="zhejiang university of technology",\*y2="zhejiang university city college";

char x[MAXN],y[MAXN];

int lenx,leny;

cout<<" 请输入第一个字符串序列x：";

while(gets(x)){ //获取输入的字符串序列

cout<<" 请输入第二个字符串序列y：";

gets(y);

lenx=strlen(x),leny=strlen(y);

cout<<x<<" 和 "<<y<<" 的长公共子序列长度为："<<LCS(lenx,leny,x,y)<<endl;

cout<<x<<" 和 "<<y<<" 的最长公共子序列是： ";

PrintLCS(lenx,leny,x,y);

cout<<endl;

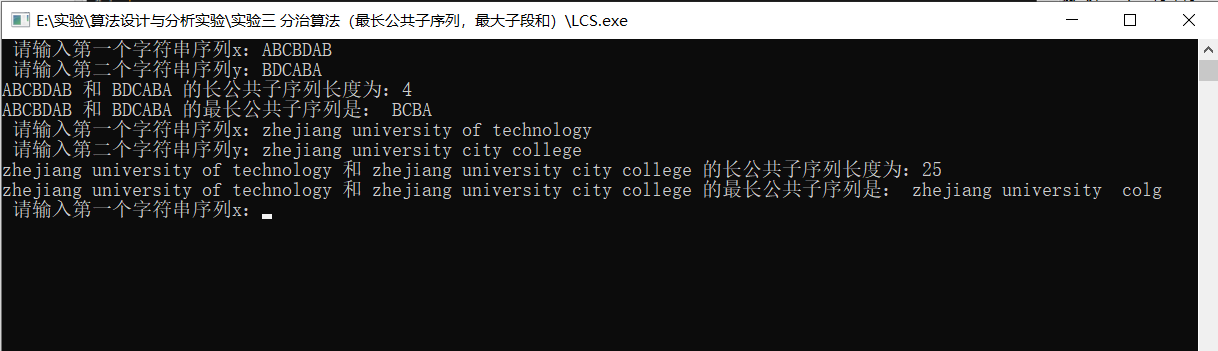
cout<<" 请输入第一个字符串序列x：";

}

return 0;

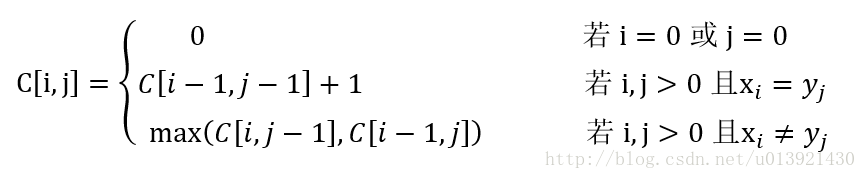
}

1. **输出结果**



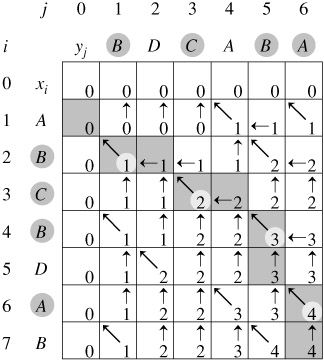
1. **性能分析**

代码中存在两个for循环的嵌套，所以计算LCS的过程中时间复杂度是Ο（m\*n），通过下面的m×n的表格可以直观地得到该算法的空间复杂度也是Ο（m\*n）。



我们可以通过表b来记录数组C是由上述三种情况的哪一种，也可以不用数组b，因为C[i，j]的值只取决于于三项：C[i-1，j]，C[i，j-1]，C[i-1，j-1]。这样就为内存节约了一个Ο（m\*n）空间。但是计算LCS的复杂度并未减少，因为下列表C的行列依旧是m和n，所以LCS的空间复杂度依然是Ο（m\*n）。

下图是第一个测试数据的Print输出过程:



1. **实验体会**

1）在本题求解中，我在第二个测试数据的调试方面花费了一些时间，我原先用char x[MAXN]存储第一个字符串，再用cin>>x得到，这导致当输入的字符串中含空格时一直达不到预期的效果。经过查阅测试，我把cin>>x改成了gets(x)，最终解决了这个BUG。

2）动态规划常用于求解最优化问题，它适用于那些子问题相互重叠的情况，即不同的子问题具有公共的子问题。

3）我大致比较了分治法和动态规划的区别:分治法会多次求解同一子问题;但动态规划只对每个子子问题求解一次，并将其保存在一个表中，从而避免了不必要的重复计算。设计一个问题的动态规划算法主要有四步：

    （1）      找出最优解的性质，刻画其结构特征；(子问题的最优解也是父问题的最优解)

    （2）      递归的定义最优解的值;(LCS共有三种情况)

    （3）      以自底向上的方式计算出最优值；(自上到下，自左到右完善整个C表)

    （4）      根据计算最优解时得到的信息，构造一个最优解。(将该数据与该数据在表中位置的左边，左上，上边位置的数据的值的比较，倒推出前一种状态，以此类推)

**问题二**

**1、问题描述**

最大子段和问题，比较三重循环，分治法，动态规划算法的效果。

**2、代码实现**

#include <iostream>

#include<cstring>

using namespace std;

/\* 求最长子段和（连续的） \*/

//穷举法

int maxInterval\_all(int len,int\*a){

int i,j,k,ans=0,sum=0;

int start=0,end=0;

//三重循环查找最长字段和

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

for(j=i;j<len;j++){

sum=0;

for(k=i;k<=j;k++){

sum+=a[k];

if(sum>ans){

start=i;

end=j;

ans=sum;

}

}

}

}

return ans;

}

//分治

int maxInterval\_div(int\*a,int left,int right){

if(left==right)

return a[left]>0?a[left]:0;

int mid=(left+right)/2;

int leftMaxInterval=maxInterval\_div(a,left,mid); //左边区域的最大子段和

int rightMaxInterval=maxInterval\_div(a,mid+1,right); //右边区域的最大子段和

int i,j;

int sum=0,left\_max=0,right\_max=0;

for(i=mid;i>=left;i--){

sum+=a[i];

if(sum>left\_max)

left\_max=sum;

}

sum=0;

for(i=mid+1;i<right;i++){

sum+=a[i];

if(sum>right\_max)

right\_max=sum;

}

//将左右两边的最大子段和相加，分别与左、右的最大子段和比较

int ans=left\_max+right\_max;

if(ans<leftMaxInterval)

ans=leftMaxInterval;

if(ans<rightMaxInterval)

ans=rightMaxInterval;

return ans;

}

//动态规划：考虑以i为终点（即必须包含a[i]）

int maxInterval(int len,int\*a){

int i,b=0,sum=0;

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

if(b>0) //若以i-1为终点的最大和b>0

//以i为终点的最大和 = 以i-1为终点的最大和b + a[i]

b+=a[i];

else //若以i-1为终点的最大和b<=0

//因为要最大，前面以i-1为终点的最大和b是负数，

//必然不能使其更大，所以令以i为终点的最大和 b = a[i]

b=a[i];

if(b>sum)

sum=b;

}

return sum;

}

int main(int argc, char\*\* argv) {

int a[]={-2,11,-4,13,-5,-2};

cout<<" 数组a为：";

int i,len;

len=6;

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

cout<<a[i]<<" ";

cout<<endl;

cout<<" 穷举法求最长子序列："<<maxInterval\_all(len,a)<<endl;

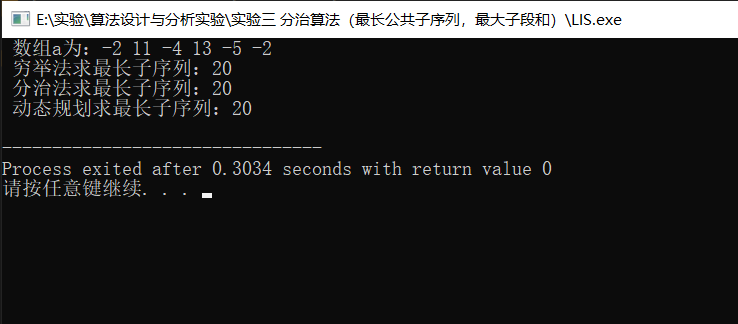
cout<<" 分治法求最长子序列："<<maxInterval\_div(a,0,len)<<endl;

cout<<" 动态规划求最长子序列："<<maxInterval(len,a)<<endl;

return 0;

}

**3、输出结果**



**4、性能分析**

三重循环的时间复杂度为 O(n^3) ，也可改进为二重循环 O（n^2）；分治算法 T(n)=2T(n/2)+O(n)==>时间复杂度为 O（nlogn）；动态规划算法只有一个for循环，时间复杂度为 O（n）。因此，动态规划算法最为优越。

**5、实验体会**

1）我大致分析了一下本题的动态规划算法，可以用数组 dp[i] 保留为以 i 结尾的最大子段和，对于 dp[i] 而言只有两种情况：若dp[i - 1] > 0, 则dp[i] = dp[i - 1] + a[i]，否则dp[i] = a[i]，最后求出dp数组中的最大值。

2）结合书中的分析，可省略数组dp，只需要两个变量：一个保存当前以i-1为终点（即必须包含a[i]）的所有子区间中的最大和(b)，一个保存最大和(sum)。具体分析如下：

若以i-1为终点的最大和b>0，显然，（以i为终点的最大和） = （以i-1为终点的最大和b） + a[i]; 若以i-1为终点的最大和b<=0 ,因为要最大，前面以i-1为终点的最大和b是负数，必然不能使其更大，所以令以i为终点的最大和 b = a[i]。最后，若以i为终点的最大和(b>sum），更新最大和sum，输出sum。

3）在分析的过程中我经常与问题一中的LCS问题搞混。LCS求最长公共子序列是不连续的；而LIC求最大子段和是连续的区间。

**问题三**

**1、问题描述**

最大子段和问题，比较三重循环，分治法，动态规划算法的效果。

问：历史上何时太阳黑子迎来了最大爆发？

**2、代码实现**

#include <iostream>

#include<cstring>

#include<fstream>

#include<sstream>

#include<ctime>

#include<vector>

#include<stdlib.h>

using namespace std;

/\* 求最长子段和（连续的）-----历史上何时太阳黑子迎来了最大爆发（文件操作） \*/

#define MAXN 1000

//穷举法(更新最长字段的开始和结束位置)

int maxInterval\_all(int len,int\*a,int& start,int& end){

int i,j,k,ans=0,sum=0;

start=0,end=0;

//三重循环查找最长字段和

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

for(j=i;j<len;j++){

sum=0;

for(k=i;k<=j;k++){

sum+=a[k];

if(sum>ans){

start=i;

end=j;

ans=sum;

}

}

}

}

return ans;

}

//分治

int maxInterval\_div(int\*a,int left,int right){

if(left==right)

return a[left]>0?a[left]:0;

int mid=(left+right)/2;

int leftMaxInterval=maxInterval\_div(a,left,mid); //左边区域的最大子段和

int rightMaxInterval=maxInterval\_div(a,mid+1,right); //右边区域的最大子段和

int i,j;

int sum=0,left\_max=0,right\_max=0;

for(i=mid;i>=left;i--){

sum+=a[i];

if(sum>left\_max)

left\_max=sum;

}

sum=0;

for(i=mid+1;i<right;i++){

sum+=a[i];

if(sum>right\_max)

right\_max=sum;

}

//将左右两边的最大子段和相加，分别与左、右的最大子段和比较

int ans=left\_max+right\_max;

if(ans<leftMaxInterval)

ans=leftMaxInterval;

if(ans<rightMaxInterval)

ans=rightMaxInterval;

return ans;

}

//动态规划：考虑以i为终点（即必须包含a[i]）

int maxInterval(int len,int\*a){

int i,b=0,sum=0;

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

if(b>0) //若以i-1为终点的最大和b>0

//以i为终点的最大和 = 以i-1为终点的最大和b + a[i]

b+=a[i];

else //若以i-1为终点的最大和b<=0

//因为要最大，前面以i-1为终点的最大和b是负数，

//必然不能使其更大，所以令以i为终点的最大和 b = a[i]

b=a[i];

//若b>sum，更新sum

if(b>sum)

sum=b;

}

return sum;

}

/\*文件操作：

ofstream: 写操作（输出）的文件类 (由ostream引申而来)

ifstream: 读操作（输入）的文件类(由istream引申而来)

fstream: 可同时读写操作的文件类 (由iostream引申而来)\*/

int main(int argc, char\*\* argv) {

ifstream file("E:\\实验\\算法设计与分析实验\\实验三 分治算法（最长公共子序列，最大子段和）\\太阳黑子处理后数据.txt");

if(!file.is\_open()){

cerr<<"文件打开失败！"<<endl;

return 0;

}

vector<string>v;

string s;

int a[MAXN]; //数组a存放太阳黑子数

int len=0;

while(getline(file,s)){ //从文件中读取每一行 （每一行有三组数据：year sunspot variation）

v.push\_back(s); //存入向量中

}

vector<string>::iterator it;

for(it=v.begin();it!=v.end();it++){

//istringstream可以创建一个对象，绑定一行字符串，然后以空格为分隔符把该行分隔

istringstream is(\*it);

string str;

int temp=0;

while(is>>str){

if(temp==2){

double x=atof(str.c\_str()); //把字符串的数字转为数字型的数值

//double x = stod(str);

a[len++]=x;

}

temp++;

}

}

len--; //数组a长度

int start,end;

clock\_t cstart,cend;

cstart=clock();

int sum1=maxInterval\_all(len,a,start,end);

cend=clock();

int t1=cend-cstart;

cstart=clock();

int sum2=maxInterval\_div(a,0,len);

cend=clock();

int t2=cend-cstart;

cstart=clock();

int sum3=maxInterval(len,a);

cend=clock();

int t3=cend-cstart;

cout<<" 太阳黑子最大爆发的年份是："<<v.at(end)<<endl;

cout<<" 三重循环用时："<<t1<<" ms"<<endl;

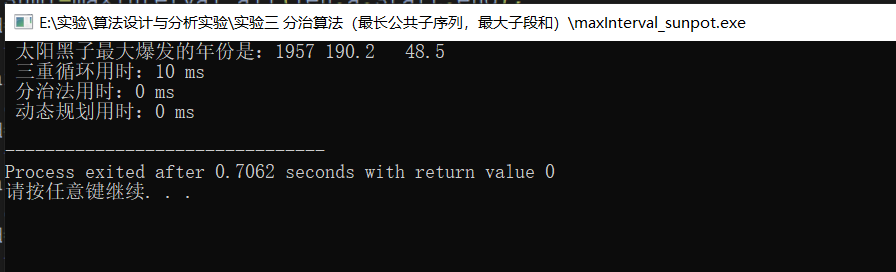
cout<<" 分治法用时："<<t2<<" ms"<<endl;

cout<<" 动态规划用时："<<t3<<" ms"<<endl;

return 0;

}

**3、输出结果**



**4、性能分析**

三重循环的时间复杂度为 O(n^3) 最慢；分治算法 T(n)=2T(n/2)+O(n)==>时间复杂度为 O（nlogn）；动态规划算法只有一个for循环，时间复杂度为 O（n）。从输出时间来看，分治与动态规划较优。

**5、实验体会**

1）在本题中，我在处理文件和字符串读取操作这两方面遇到了一点困难。以下是我的解决过程。

2）我回想了一下大一关于文件的操作，用 ifstream 打开文件,通过 is\_open() 判断该文件是否被打开，再用 getlin(file,s) 读取每一行的字符串并保存入向量v中。

3）对文件中数据的操作，我用vector保存数据，遍历该向量时在分别读取每一行的三个数据时遇到了困难。经查阅资料，发现 istringstream 是C++里面的一种输入输出控制类，它可以创建一个对象is，然后is对象就可以绑定一行字符串，然后通过 is>>str 以空格为分隔符把该行分隔开来。

4）因为分割开后的数据仍然是字符串，因此需要用 double x=atof(str.c\_str()); 或 double x = stod(str); 或 int x=atoi(str.c\_str()); 把字符串的数字转为数字型的数值，存入数组a中，其中 c\_str() 方法是将string转为char\*的字符串。我在引用的时候还因为忘记加头文件#include<stdlib.h>而报错。

5）总的来说，这道题让我温习了文件处理和字符串操作，受益良多。