**动态规划法**

**1610300c02 夏鹏程**

**总分: 44 / 50**

编程题总分: 44 / 50

7-1答案正确 得分: 8 / 8

给出2个字符串，输出2字符串的最长公共子串。

**输入格式:**

输入2个字符串，不可包含空格。

**输出格式:**

输出2个字符串的最长公共子串。若没有公共子串，则输出“NULL”

**输入样例1:**

abcding

sbcdiofng

**输出样例1:**

bcdi

**输入样例2:**

understanding

fw

**输出样例2:**

NULL

#include <stdio.h>

#include<stdlib.h>

#include<string.h>

void main()

{ char s1[50];

gets(s1);

char s2[50];

gets(s2);

int count[50] = { 0 };

int i = 0;

int k = 0;

int max = 0;

int m = 0;

char \*ps[50] = { 0 };

for (char \*pc1 = s1; \*pc1 != '\0'; pc1++)

{ for (char \*pc2 = s2; \*pc2 != '\0'; pc2++)

{ char \*temp1 = pc1;

char \*temp2 = pc2;

if ((\*temp1 == \*temp2) && (\*temp2 != '\0'))

{ ps[i] = temp1;//保留地址

while ((\*temp1 == \*temp2) && (\*temp2 != '\0'))

{ temp1++;

temp2++;

count[i]++; }

i++; } //printf("NULL");break; } }

for (int j = 0; j < 50; j++)

{ if (max < count[j])

{ k = j;

max = count[j]; } }

//printf("%d\n", max);

while (m < max)

{ printf("%c", \*ps[k]);

ps[k]++;

m++; }

if(m==0)

printf("NULL");

printf("\n");

system("pause"); }

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案正确 | 2 ms | 128KB |
| 1 | 答案正确 | 1 ms | 128KB |
| 2 | 答案正确 | 2 ms | 128KB |
| 3 | 答案正确 | 2 ms | 136KB |

7-2答案正确 得分: 8 / 8

给定*K*个整数组成的序列{ *N*​1​​, *N*​2​​, ..., *N*​*K*​​ }，“连续子列”被定义为{ *N*​*i*​​, *N*​*i*+1​​, ..., *N*​*j*​​ }，其中 1≤*i*≤*j*≤*K*。“最大子列和”则被定义为所有连续子列元素的和中最大者。例如给定序列{ -2, 11, -4, 13, -5, -2 }，其连续子列{ 11, -4, 13 }有最大的和20。现要求你编写程序，计算给定整数序列的最大子列和。

本题旨在测试各种不同的算法在各种数据情况下的表现。各组测试数据特点如下：

* 数据1：与样例等价，测试基本正确性；
* 数据2：102个随机整数；
* 数据3：103个随机整数；
* 数据4：104个随机整数；
* 数据5：105个随机整数；

**输入格式:**

输入第1行给出正整数*K* (≤100000)；第2行给出*K*个整数，其间以空格分隔。

**输出格式:**

在一行中输出最大子列和。如果序列中所有整数皆为负数，则输出0。

**输入样例:**

6

-2 11 -4 13 -5 -2

**输出样例:**

20

#include<stdio.h>

int main(){

int result=0,thissum=0,N,digit;

scanf("%d",&N);

for(int i=0;i<N;i++){ scanf("%d",&digit);

thissum+=digit;

if(thissum>result)

result=thissum;

if(thissum<0) thissum=0; }

printf("%d",result);

return 0; }

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案正确 | 2 ms | 128KB |
| 1 | 答案正确 | 2 ms | 160KB |
| 2 | 答案正确 | 2 ms | 256KB |
| 3 | 答案正确 | 3 ms | 128KB |
| 4 | 答案正确 | 19 ms | 156KB |

7-3部分正确 得分: 6 / 8

有一个背包的最大能承受的重量是 M ，有 n 个物品，每个物品有各自的重量和价值，计算在不超出背包最大承重限制下，背包中物品最大价值可以是多少？

**输入格式:**

第一行输入背包的最大承重量 M 和物品的个数 n (1<M<500，1<n<20)，第二行输入1至n个物品的重量，第三行输入1至n个物品的价值。

**输出格式:**

输出最大可能的价值。

**输入样例:**

在这里给出一组输入。例如：

30 3

5 10 20

50 60 140

**输出样例:**

在这里给出相应的输出。例如：

200

#include <iostream>

using namespace std;

int knapsack(int \*W, int \*V, int \*res, int n, int C)

{ int value = 0;

int \*\*f = new int\*[n];

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

{ f[i] = new int[C+1]; }

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

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

f[i][j] = 0;

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

{ f[i][0] = 0; }

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

{ f[0][i] = (i < W[0])?0:V[0]; }

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

{ for(int y = 1; y <= C; y++)

{ if(y >= W[i])

{ f[i][y] = (f[i-1][y] > (f[i-1][y-W[i]] + V[i]))?f[i-1][y]:(f[i-1][y-W[i]] + V[i]);

} else { f[i][y] = f[i-1][y]; } } }

/\*for(int i = 0; i < n; i++)

{ for(int j = 0; j < C+1;j++)

cout << f[i][j] << " ";

cout << endl; }\*/

value = f[n-1][C];

int j = n-1;

int y = C;

while(j)

{ if(f[j][y] == (f[j-1][y-W[j]] + V[j]))

{ res[j] = 1;

y = y - W[j]; }

j--; }

if(f[0][y])

{ res[0] = 1; }

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

{ delete f[i];

f[i] = 0; }

delete [] f;

f = 0;

return value; }

void test1()

{ int n, C;

while(cin >> C>> n)

{ int \*W = new int[n];

int \*V = new int[n];

int \*res = new int[n];

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

{ res[i] = 0; }

int w = 0, v = 0;//i = 0;

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

{ cin>>w;

W[j]=w; }

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

{ cin>>v;

V[k]=v; }

/\*while(i < n)

{ cin >> w >> v;

W[i] = w;

V[i] = v;

i++; }\*/

int value = knapsack(W, V, res, n, C);

cout << value << endl;

//for(int i = 0; i < n; i++)

//cout << res[i] << " ";

//cout << endl;

delete W;

delete V;

delete res; } }

int main()

{ test1();

return 0; }

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案正确 | 2 ms | 248KB |
| 1 | 答案错误 | 2 ms | 248KB |
| 2 | 答案正确 | 2 ms | 248KB |
| 3 | 答案正确 | 2 ms | 252KB |

7-4答案正确 得分: 8 / 8

用 n 种不同币值的硬币凑出 m 元，最少需要多少硬币。

**输入格式:**

第一行输入需要凑的钱数 m 和硬币的种类 n （0<m<100，0<n<10），第二行输入 n 种硬币的具体币值，假设硬币供应量无限多。

**输出格式:**

输出最少需要的硬币个数

**输入样例:**

在这里给出一组输入。例如：

6 3

1 3 4

**输出样例:**

在这里给出相应的输出。例如：

2

#include<iostream>

using namespace std;

//money需要找零的钱

//coin可用的硬币

//硬币种类

void FindMin(int money,int \*coin, int n)

{

int \*coinNum=new int[money+1]();//存储1...money找零最少需要的硬币的个数

int \*coinValue=new int[money+1]();//最后加入的硬币，方便后面输出是哪几个硬币

coinNum[0]=0;

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

{

int minNum=i;//i面值钱，需要最少硬币个数

int usedMoney=0;//这次找零，在原来的基础上需要的硬币

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

{

if(i>=coin[j])//找零的钱大于这个硬币的面值

{

//if(coinNum[i-coin[j]]+1<=minNum)//所需硬币个数减少了

/\*

上面的判断语句有问题，在更新时，需要判断i-coin[j]是否能找的开，如果找不开，就不需要更新。

多谢zywscq 指正

\*/

if(coinNum[i-coin[j]]+1<=minNum&&(i==coin[j]||coinValue[i-coin[j]]!=0))//所需硬币个数减少了

{

minNum=coinNum[i-coin[j]]+1;//更新

usedMoney=coin[j];//更新

}

}

}

coinNum[i]=minNum;

coinValue[i]=usedMoney;

}

//输出结果

if(coinValue[money]==0)

cout<<"找不开零钱"<<endl;

else

{

cout<<coinNum[money]<<endl; //最少硬币数

//cout<<"硬币分别为:";

//while(money>0)

//{

//cout<<coinValue[money]<<",";

//money-=coinValue[money];

//}

}

delete []coinNum;

delete []coinValue;

}

int main()

{

int Money;

int type;

cin>>Money>>type;

int coin[10];

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

cin>>coin[i];

FindMin(Money,coin,type);

return 0; }

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案正确 | 3 ms | 248KB |
| 1 | 答案正确 | 3 ms | 248KB |
| 2 | 答案正确 | 3 ms | 248KB |
| 3 | 答案正确 | 3 ms | 252KB |

7-5部分正确 得分: 4 / 8

跳房子是小朋友玩的游戏。地面上画出一连串格子，每个格子里有一个整数，小朋友从外面跳入格子，并继续往前跳，直到跳出所有格子。每次跳跃的规则是，可以跳入下一格或下下格或下下下格。怎么跳能让落脚格子里的数的累加和最小。

**输入格式:**

第一行输入格子数 n (1<n<100)，第二行输入从起点处到终点处每个格子里的数，该数小于10。

**输出格式:**

输出最小累加和。

**输入样例:**

在这里给出一组输入。例如：

4

3 3 2 5

**输出样例:**

在这里给出相应的输出。例如：

2

#include <iostream>

#include <cstdio>

using namespace std;

int main()

{

int n;

int a[6005];

int step[6005];

scanf("%d",&n);

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

scanf("%d",&a[i]);

step[i]=i;

}

for(int k=0;k<n;k++){

for(int j=1;j<=a[k];j++){

if(step[k]+1<step[k+j]){

step[k+j]=step[k]+1; } } }

printf("%d",step[n-1]+1);

return 0; }

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案正确 | 2 ms | 252KB |
| 1 | 答案正确 | 2 ms | 252KB |
| 2 | 答案错误 | 2 ms | 256KB |
| 3 | 答案错误 | 2 ms | 248KB |

7-6答案正确 得分: 10 / 10

在一个m行n列方格矩阵中，每一个方格内摆放着价值不等的宝贝（价值可正可负），让小明感到好奇的是，从左上角到达右下角的所有可能路线中，能捡到宝贝的价值总和最大是多少？而且这种达到最大值的路线 又有多少条？【注意：只能从一个格子向下或向右走到相邻格子，并且走到的格子宝贝一定会被捡起。】

**输入格式:**

第一行为整数m，n（均不大于100），下一行开始会有一个m行n列的整数方阵，对应方格矩阵中的宝贝价值（这些值的绝对值都不超过500）。

**输出格式:**

单独一行输出2个整数，分别为能捡到宝贝价值总和的最大值和达到最大值的路线数量，2个整数间隔一个空格。

**输入样例:**

在这里给出一组输入。例如：

4 5

2 -1 6 -2 9

-3 2 5 -5 1

5 8 3 -2 4

5 2 8 -4 7

**输出样例:**

对应的输出为：

26 3

#include <iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int r,c;

int m[104][104];

int val[104][104];

int ans[104][104];

int main()

{

// freopen("a.txt","r",stdin);

memset(ans,0,sizeof ans);

memset(m,0,sizeof m);

memset(val,0,sizeof val);

scanf("%d %d",&r,&c);

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

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

scanf("%d",&m[i][j]);

int total = 0;

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

{ total += m[1][i];

val[1][i] = total;

ans[1][i] = 1;

}

total = 0;

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

{ total += m[i][1];

val[i][1] = total;

ans[i][1] = 1; }

for( int i = 2 ; i <= r ; i++ )

for( int j = 2 ; j<= c ; j++ )

{ val[i][j] = max( val[i][j-1],val[i-1][j] )+m[i][j];

if( val[i][j-1] == val[i-1][j] )

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

else if( val[i][j-1] > val[i-1][j] )

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

else ans[i][j] = ans[i-1][j]; }

cout<<val[r][c]<<" "<<ans[r][c]<<endl;

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

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

cout<<val[i][j]<<" ";

printf("\n");

}\*/

return 0; }

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案正确 | 3 ms | 380KB |
| 1 | 答案正确 | 3 ms | 412KB |
| 2 | 答案正确 | 3 ms | 512KB |
| 3 | 答案正确 | 3 ms | 380KB |
| 4 | 答案正确 | 4 ms | 412KB |