**贪心法**

**1610300c02 夏鹏程**

**总分: 52 / 60**

编程题总分: 52 / 60

7-1答案正确 得分: 6 / 6

同时有n个任务需要计算机后台处理，但是每次只能处理一个任务，其他任务处于等待。系统总体等待时间是每个任务的等待时间之和。如何安排任务的处理顺序使得系统总体等待时间最小。

**输入格式:**

第一行输入整数n，代表任务的个数，1<n<100。第二行输入n个整数，代表每个任务需要处理的时间，每个任务处理的时间都小于100。

**输出格式:**

输出最小的总体等待时间。

**输入样例:**

在这里给出一组输入。例如：

2

3 5

**输出样例:**

在这里给出相应的输出。例如：

3

#include <iostream>

#include <string>

#include <cstring>

#include <stdio.h>

#include <cmath>

#include <algorithm>

using namespace std;

int a[10000];

int d[10000];

int main()

{ int n;

int m=1;

while(cin>>n)

{ memset(d,0,sizeof(d));

memset(a,0,sizeof(a));

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

{ cin>>a[i];

}

sort(a,a+n);

int sum=0;

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

{ sort(d,d+m);

sum+=d[0];

d[0]+=a[j];

}

cout<<sum<<endl;

}

return 0; }

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案正确 | 3 ms | 376KB |
| 1 | 答案正确 | 3 ms | 384KB |
| 2 | 答案正确 | 3 ms | 376KB |

7-2答案正确 得分: 8 / 8

给定一个正整数，去掉其中的k位数，剩下的数字按原先次序排列，如何让生成的数最小。

**输入格式:**

输入一个长度不超过10的正整数n，和要去掉的位数k。

**输出格式:**

输出n在去掉k位数后的最小生成数。

**输入样例:**

在这里给出一组输入。例如：

251 1

**输出样例:**

在这里给出相应的输出。例如：

21

#include <iostream>

#include<string.h>

using namespace std;

const int Limit\_Size = 21;

char num[ Limit\_Size ];

int S, len;

void init( )

{ cin >> num >> S;

len = strlen( num ); }

void work( )

{ int i, j, k;

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

{ for ( j = 0; j < len - 1; j++ )

if ( num[ j ] > num[ j + 1 ] )

{ for ( k = j; k < len - 1; k++ )

num[ k ] = num[ k + 1 ];

break;

} len--; } }

void print( )

{ int i;

if ( len <= 0 )

cout << 0;

else

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

cout << num[ i ];

cout << endl;

}

int main( )

{ init( );

work( );

print( );

return 0; }

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

7-3部分正确 得分: 3 / 8

一辆加满油的汽车欲行驶到m公里外的终点，旅途中有若干加油站，并且每个加油站距离起点的公里数已知，汽车加满油后可行驶n公里。如何用最少的加油次数到达目的地。

**输入格式:**

第一行输入终点公里数m，和汽车加满油可行驶的公里数n，以及加油站的个数k。第二行输入k个整数分别代表从起点到终点的k个加油站距离起点的公里数。

**输出格式:**

输出最少的加油次数。

**输入样例:**

在这里给出一组输入。例如：

10 6 3

2 5 9

**输出样例:**

在这里给出相应的输出。例如：

1

#include <iostream>

using namespace std;

void greedy(int d[],int n,int k) {

int num = 0;

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

if(d[i] > n) {

cout<<"no solution"<<endl;

return; } }

for(int i = 0,s = 0;i <= k;i++) {

s += d[i];

if(s > n) {

num++;

s = d[i]; } }

cout<<num<<endl;

}

int main() {

int i,n,k;

int d[1000];

cin>>n>>k;

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

cin>>i;

greedy(d,n,k); }

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案错误 | 2 ms | 248KB |
| 1 | 答案正确 | 2 ms | 384KB |
| 2 | 答案错误 | 2 ms | 252KB |

7-4部分正确 得分: 5 / 8

有n场活动需要举行，而每个会场在一个时段内只能接待一场活动，如果要把n场活动全部举行，最少需要几个会场。

**输入格式:**

第一行输入活动的个数n，n<100。第二行依次输入n场活动的起始时间（整数）。第三行依次输入n场活动的结束时间（整数）。

**输出格式:**

输出最少需要的会场个数。

**输入样例:**

在这里给出一组输入。例如：

3

2 3 0

4 5 2

**输出样例:**

在这里给出相应的输出。例如：

2

#include <stdio.h>

#include <algorithm>

using namespace std;

struct active{ int start,end,flag; } act[100];

bool cmp(struct active a,struct active b)

{ return a.end<b.end; }

int main()

{ int n;

int count=0;//代表当前被安排的活动

int sum=0;//所需要会场数目

scanf("%d",&n);

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

{ scanf("%d %d",&act[i].start,&act[i].end);

act[i].flag=0; }

sort(act,act+n,cmp);

while(count<n)

{ int temp=0;

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

{ if(act[i].start>=temp && act[i].flag==0)

{ temp=act[i].end;

act[i].flag=1;

count++; } }//会场所有活动安排完成

sum++;

}

printf("%d\n",sum);

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 | 256KB |
| 2 | 答案错误 | 2 ms | 128KB |

7-5答案正确 得分: 10 / 10

假设有*N*项物品，大小分别为*s*​1​​、*s*​2​​、…、*s*​*i*​​、…、*s*​*N*​​，其中*s*​*i*​​为满足1≤*s*​*i*​​≤100的整数。要把这些物品装入到容量为100的一批箱子（序号1-*N*）中。装箱方法是：对每项物品, 顺序扫描箱子，把该物品放入足以能够容下它的第一个箱子中。请写一个程序模拟这种装箱过程，并输出每个物品所在的箱子序号，以及放置全部物品所需的箱子数目。

**输入格式：**

输入第一行给出物品个数*N*（≤1000）；第二行给出*N*个正整数*s*​*i*​​（1≤*s*​*i*​​≤100，表示第*i*项物品的大小）。

**输出格式：**

按照输入顺序输出每个物品的大小及其所在的箱子序号，每个物品占1行，最后一行输出所需的箱子数目。

**输入样例：**

8

60 70 80 90 30 40 10 20

**输出样例：**

60 1

70 2

80 3

90 4

30 1

40 5

10 1

20 2

5

#include <stdio.h> //时间：2018年4月20日09:57:19

#include <string.h> //思路：关键是确定每一个数组元素（物品）放入哪个箱子（容器）中？从第2个物品开始，

#include <math.h> //循环n-1次，检测这n-1个物品，与前面物品之和是否满足其和小于箱子的容量，如果小于100，则找到其应该放在

#include <stdlib.h> //箱子的位置。

#include <ctype.h>

#define N 1000

int main()

{ int n, i, j, flag, max;

int s[N]; //输入数据

int d[N]; //临时数组（用于动态处理箱子中的物品）

int pos[N];

scanf("%d", &n);

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

{ pos[i] = -1; }

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

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

d[i] = s[i]; }

pos[0] = 0; //初始化第一个箱子的编号为0

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

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

{ flag = 0;

if (d[i] + d[j] <= 100) //寻找相同的容器

{ d[j] = d[j] + d[i];

d[i] = 0;

pos[i] = j; //记录d[i]中第i个元素放入的容器的编号

flag = 1;

break; } }

if (flag == 0)

{ pos[i] = i; } }

max = 0;

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

{ if (pos[i]>max)

max = pos[i]; }

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

{ printf("%d %d\n", s[i], pos[i] + 1) }

printf("%d\n", max + 1);

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 | 128KB |
| 2 | 答案正确 | 2 ms | 132KB |
| 3 | 答案正确 | 2 ms | 196KB |

7-6答案正确 得分: 10 / 10

有了一张自驾旅游路线图，你会知道城市间的高速公路长度、以及该公路要收取的过路费。现在需要你写一个程序，帮助前来咨询的游客找一条出发地和目的地之间的最短路径。如果有若干条路径都是最短的，那么需要输出最便宜的一条路径。

**输入格式:**

输入说明：输入数据的第1行给出4个正整数*N*、*M*、*S*、*D*，其中*N*（2≤*N*≤500）是城市的个数，顺便假设城市的编号为0~(*N*−1)；*M*是高速公路的条数；*S*是出发地的城市编号；*D*是目的地的城市编号。随后的*M*行中，每行给出一条高速公路的信息，分别是：城市1、城市2、高速公路长度、收费额，中间用空格分开，数字均为整数且不超过500。输入保证解的存在。

**输出格式:**

在一行里输出路径的长度和收费总额，数字间以空格分隔，输出结尾不能有多余空格。

**输入样例:**

4 5 0 3

0 1 1 20

1 3 2 30

0 3 4 10

0 2 2 20

2 3 1 20

**输出样例:**

3 40

#include<iostream>

using namespace std;

int mcost[501],visited[501],i,j,s,e,a,b,c,d;

int cost[501][501],g[501][501],dis[501],n,m;

void min(int s,int x,int e)

{ nt i,j;

dis[s]=0,visited[s]=1;

for(i=1;i<x-1;i++)

{ int min=501;

for(j=0;j<x;j++)

{ if(visited[j]==0 && dis[j]<min)

{ min=dis[j];

s=j; } }

visited[s]=1;

for(j=0;j<x;j++)

{ if(visited[j]==0 && min+g[s][j]<dis[j])

{ dis[j]=min+g[s][j];

mcost[j]=mcost[s]+cost[s][j]; }

else if(visited[j]==0 && min+g[s][j]==dis[j] && mcost[j]>mcost[s]+cost[s][j])

mcost[j]=mcost[s]+cost[s][j]; } } }

int main()

{ cin>>n>>m>>s>>e;

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

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

{ g[i][j]=g[j][i]=501;

cost[i][j]=cost[j][i]=501; } }

while(m--)

{ cin>>a>>b>>c>>d;

g[a][b]=g[b][a]=c;

cost[a][b]=cost[b][a]=d; }

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

{ dis[i]=g[s][i];

mcost[i]=cost[s][i]; }

min(s,n,e);

cout<<dis[e]<<" "<<mcost[e]<<endl;

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 | 296KB |
| 2 | 答案正确 | 7 ms | 2168KB |
| 3 | 答案正确 | 251 ms | 2256KB |

7-7答案正确 得分: 10 / 10

农夫要修理牧场的一段栅栏，他测量了栅栏，发现需要*N*块木头，每块木头长度为整数*L*​*i*​​个长度单位，于是他购买了一条很长的、能锯成*N*块的木头，即该木头的长度是*L*​*i*​​的总和。

但是农夫自己没有锯子，请人锯木的酬金跟这段木头的长度成正比。为简单起见，不妨就设酬金等于所锯木头的长度。例如，要将长度为20的木头锯成长度为8、7和5的三段，第一次锯木头花费20，将木头锯成12和8；第二次锯木头花费12，将长度为12的木头锯成7和5，总花费为32。如果第一次将木头锯成15和5，则第二次锯木头花费15，总花费为35（大于32）。

请编写程序帮助农夫计算将木头锯成*N*块的最少花费。

**输入格式:**

输入首先给出正整数*N*（≤10​4​​），表示要将木头锯成*N*块。第二行给出*N*个正整数（≤50），表示每段木块的长度。

**输出格式:**

输出一个整数，即将木头锯成*N*块的最少花费。

**输入样例:**

8

4 5 1 2 1 3 1 1

**输出样例:**

49

#include<iostream>

#include<vector>

using namespace std;

void Sum(vector<int> &arr)

{

int total=0;//总花费

while(arr.size()>1)

{

int min1=arr[0];//权值最小的两个值

int pos1=0,pos2=0;//当前最小结点的下标

int tmp=0;//保存当前两个最小结点的权值之和

for(int i=1; i<arr.size(); i++)

{

if(arr[i]<min1)

{

min1=arr[i];

pos1=i;

}

}

tmp+=min1;

arr.erase(arr.begin()+pos1);

int min2=arr[0];//删除一个数之后的第一个元素

for(int i=1; i<arr.size(); i++)

{

if(arr[i]<min2)

{

min2=arr[i];

pos2=i;

}

}

tmp+=min2;

arr.erase(arr.begin()+pos2);

total+=tmp;

arr.push\_back(tmp);

}

cout<<total;

}

int main()

{

int N,W;

cin>>N;

vector<int> weight;

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

{

cin>>W;

weight.push\_back(W);

}

Sum(weight);

return 0;

}

| **测试点** | **结果** | **耗时** | **内存** |
| --- | --- | --- | --- |
| 0 | 答案正确 | 2 ms | 248KB |
| 1 | 答案正确 | 2 ms | 384KB |
| 2 | 答案正确 | 130 ms | 404KB |
| 3 | 答案正确 | 124 ms | 404KB |