

算法设计与分析实验报告

学院：电信学院

姓名：贾根龙

班级：计算机54

学号:2150500126

实验一 线性时间查找中位数

1. 问题描述

编写程序，实现先线性时间内选择n个元素的中位数算法，并对不同的n，测试平均时间效率。

1. 问题分析

线性时间选择问题分属于第二章内容，可采用递归与分治策略。可以类似于快速排序算法，找到一个基准，但与快速排序不同。例如：第一次以数组首元素为基准把数组划分，使得首元素a[p]左侧元素均小于它，右侧元素均大于它。记此时a[p]下标为i。设数组长度为n，则若i=n，则a[p]即为中位数；若i<n,则中位数定在a[p]的右侧出现；若i>n，则中位数定在a[p]左侧出现。故再对a[p]左侧或右侧元素按上述方法进行递归查找。这种形况下，平均时间复杂度为O(n)。

1. 算法程序

#include <stdio.h>

void exch (int \*array, int i, int j)

{

if (i!=j)

{

int w;

w=array[i];

array[i]=array[j];

array[j]=w;

}

}

double select\_middle (int \*array, int beg, int end, int n)

{

if (n == 1)

return array[1]; //数组只有一个元素

int i = beg, j;

for (j=i+1; j<=end; ++j)

if (array[j]<=array[beg])

{

++i;

exch (array, i, j);

}

exch (array, beg, i); //对数组首元素进行划分，用i记录首元素位置

if (i < n/2)

return select\_middle (array, i+1, end, n);

else if (i > n/2)

return select\_middle (array, beg, i-1, n);

else

{

if (n%2)

return array[i];

else

{

int j, m=array[0];

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

if (array[j] > m)

m=array[j];

return (double)(array[i]+m)/2;

}

}

}

int main()

{

int a[100];

int c=0;

char s;

printf("请输入数组元素（长度小于100）：\n");

for(int i=0;i<100&&s!='\n';i++)

{

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

s=getchar();

c++;

}

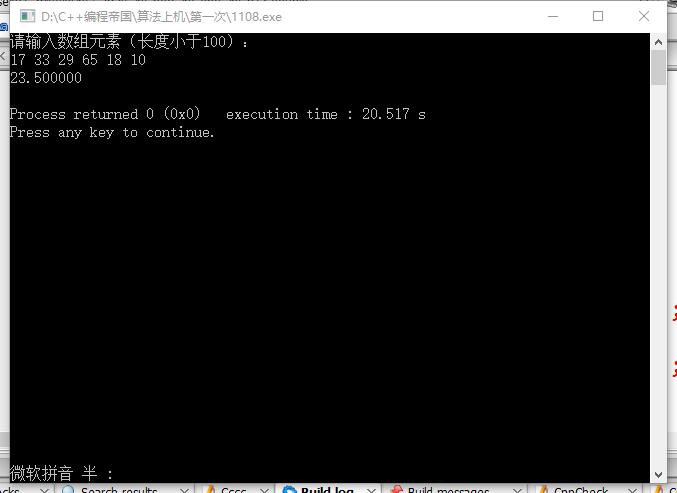
printf("%lf\n",select\_middle(a,0,c-1,c));

return 0;

}

程序主体式找到数组首元素位置，并用i记录该位置。然后判断i与n/2的大小关系进行递归运算。在主函数中，用字符s判断当输入“回车”时结束输入，并用c记录数组长度，算法实现较为简单。

1. 运行结果



实验二 0-1背包问题

1. 问题描述

给定n种物品和一背包。物品i的重量是Wi，其价值是Vi，背包容量为C。问应该如何选择装入背包的物品，使得装入背包的物品总价值最大。

1. 问题分析

0-1背包问题可用动态规划来解决，找出递归关系即可。设背包问题最优值为m(i,j),即背包容量为j，可选择物品为i,i+1,…,n时0-1背包问题的最优值为m(i,j)。则当0=<j<Wi时，m(i,j)=m(i+1,j),因为此时背包无法装下第i件物品；当j>=Wi时，m(i,j)=max{m(i+1,j),m(i+1,j-Wi)+Vi},即选第i件物品和不选第i件物品两者中较大的作为m(i,j)的取值。初始条件是当j>Wn时，m(i,j)=Vn；当0=<j<Wn时，m(i,j)=0。即此时只有第件物品，若第n件物品重量小于背包容量，则放入，否则不放入。

1. 算法程序

#include<stdio.h>

int V[200][200];//前i个物品装入容量为j的背包中获得的最大价值

int max(int a,int b)

{

if(a>=b)

return a;

else return b;

}

int KnapSack(int n,int w[],int v[],int x[],int C)

{

int i,j;

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

V[i][0]=0;

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

V[0][j]=0;

for(i=0;i<=n-1;i++)

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

if(j<w[i])

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

else

V[i][j]=max(V[i-1][j],V[i-1][j-w[i]]+v[i]);

j=C;

for(i=n-1;i>=0;i--)

{

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

{

x[i]=1;

j=j-w[i];

}

else

x[i]=0;

}

printf("选中的物品是:\n");

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

printf("%d ",x[i]);

printf("\n");

return V[n-1][C];

}

int main()

{

int s;//获得的最大价值

int w[15];//物品的重量

int v[15];//物品的价值

int x[15];//物品的选取状态

int n,i;

int C;//背包最大容量

n=5;

printf("请输入背包的最大容量:\n");

scanf("%d",&C);

printf("输入物品数:\n");

scanf("%d",&n);

printf("请分别输入物品的重量:\n");

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

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

printf("请分别输入物品的价值:\n");

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

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

s=KnapSack(n,w,v,x,C);

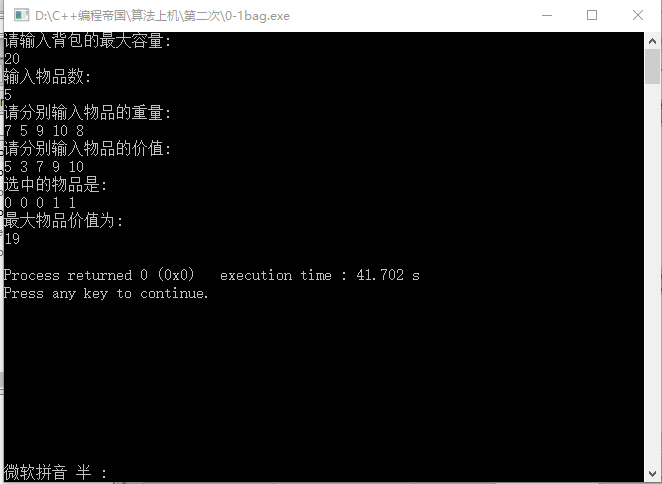
printf("最大物品价值为:\n");

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

return 0;

}

1. 运行结果



运行结果中，0代表不装入该背包，1代表装入背包。

实验三 多机制调度问题

1. 问题描述

设有n个独立的作业{1,2,…,n},由m台相同的机器进行加工处理。作业i所需要的处理时间为ti。约定，每个作业均可在任何一台机器上加工处理，但未完工前不允许中断处理，作业不能拆分为更小的子作业。给出一种作业调度方案，使所给的n个作业在尽可能短的时间内由m台相同的机器加工完成。

1. 问题分析

该问题可运用贪心算法简单解决，采用最长处理作业优先的贪心选择策略设计。即先处理时间长的优先处理。则需要一个排序算法对数组元素进行排序。当n<=m时，总处理时间固定为这n个作业中处理时间最长的用时；当n>m时就优先处理时间长的作业。

1. 算法程序

#include<stdio.h>

#include<string.h>

#include<iostream>

#include<algorithm>

using namespace std;

bool comp(int x,int y)

{

return x>y;

}

int main()

{

int ncases,n,m,i,a[10010],ans[101];

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

memset(ans,0,sizeof(ans));

printf("请输入作业数和机器数：\n");

scanf("%d %d",&n,&m);

printf("请输入每个作业在机器上的加工时间：\n");

for(i=0;i<=n-1;i++)

{

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

}

sort(a,a+n,comp);

for(i=0;i<=n-1;i++)

{

\*min\_element(ans,ans+m)=\*min\_element(ans,ans+m)+a[i];

}

printf("最短加工时间：\n");

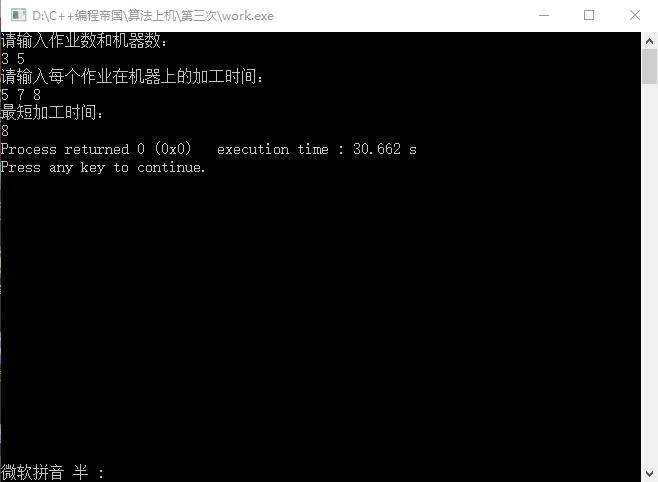
printf("%d",\*max\_element(ans,ans+m));

return 0;

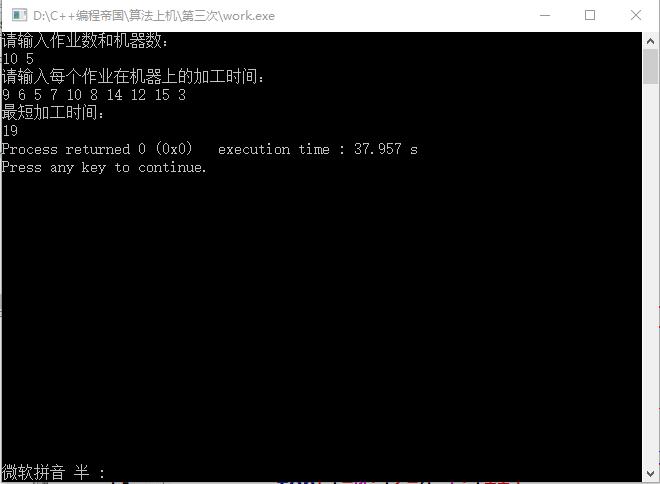
}

其中，memset()函数是将数组元素初始化，本算法是将数组a和ans初始化为0。函数\*min\_element()和\*max\_element()是找出数组中的最小值和最大值。

1. 运行结果



当作业数小于机器数时，最短加工时间即为最长加工时间。



实验四 旅行商问题

1. 问题描述

某售货员要到若干城市去推销商品，已知各城市之间的路程（或旅费）。他要选定一条从驻地出发，经过每个城市一次，最后回到驻地的路线，使总的路程（或总路费）最小。

1. 问题分析

路线是一个带权图。图中各边的路费为正数。图的一条周游路线是包括V中的每一个顶点在内的一条回路。周游路线的费用是这条路线上所有的费用之和。旅行售货员问题的解空间是一颗排列树，从树的根结点到任一的叶结点的路径定义了图的一条周游路线。旅行售货员问题要在图G中找出费用最小的周游路线。运用回溯法可轻松解决该问题。在递归算法BackTrack中，当i=n时，当前扩展结点是排列树的叶结点的父节点。此时算法检测图G是否存在一条从顶点x[n-1]到顶点x[n]的边和一条从顶点x[n]到顶点1的边。如果这两条边都存在，则找到一条旅行售货员回路。此时，算法还需判断这条回路是否优于当前最优回路的费用bestc。如果是，则必须更新当前最优值bestc和当前最优解bestx。

当i<n时，当前扩展结点位于排列树的第i-1层。图G中存在从顶点x[i-1]到顶点x[i]的边时，x[1:i]构成图G的一条路径，且当x[1:i]的费用小于当前最优值时算法进入排列树的第i层，否则，将剪去相应子树。算法中用变量cc记录当前路径x[1:i]的费用。

1. 算法程序

#include<iostream>

using namespace std;

#define NUM 10

int cc; //当前费用

int bestc; //当前最优费用

int a[NUM][NUM]; //图G的邻接矩阵

int x[NUM]; //当前解

int bestx[NUM]; //最优解,顺序输出即可

int n;

void input()

{

int i,j;

cout<<"输入城市的数量：";

cin>>n;

cout<<"输入大于0有边，输入小于0无边：\n";

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

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

{

cout<<"请输入第"<<i<<"个城市到第"<<j<<"个城市所需的路费：";

cin>>a[i][j];

a[j][i]=a[i][j];

}

}

void BackTrack(int i) //t从2开始

{

if(i==n) //解空间树完成

{

if(a[x[n-1]][x[n]]>0&&a[x[n]][x[1]]>0) //有路

{

if(bestc<0||bestc>cc+a[x[n-1]][x[n]]+a[x[n]][x[1]]) //当前路径更小，则更新bestc

{

int j;

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

{

bestx[j]=x[j];

bestc=cc+a[x[n-1]][x[n]]+a[x[n]][x[1]];

}

}

}

}

else //继续构建解空间树

{

int j;

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

{

if(a[x[i-1]][x[j]]>0) //有路

if(bestc<0||bestc>cc+a[x[i-1]][x[j]]+a[x[j]][x[1]]) //用路径更短的一条路径来替换

{

int temp;

cc+=a[x[i-1]][x[j]];

temp=x[i];

x[i]=x[j];

x[j]=temp;

BackTrack(i+1);

temp=x[i];

x[i]=x[j];

x[j]=temp;

cc-=a[x[i-1]][x[j]];

}

}

}

}

int tsp()

{

//初始化

int i;

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

x[i]=i;

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

{

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

{

a[k][j]=-1;

}

}

cc = 0;

bestc=-1;

input();

BackTrack(2);

return bestc;

}

void output()

{

int i;

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

cout<<bestx[i]<<" ";

}

int main()

{

cout<<"走"<<n<<"个城市最少的路费是:"<<tsp();

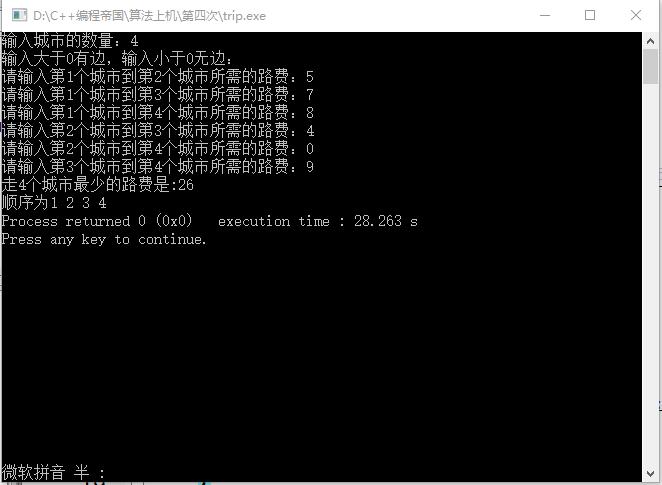
cout<<"\n顺序为";

output();

return 0;

}

1. 实验结果



输入0代表两个城市没有边经过。

如果所有排列树走到叶子结点的父结点若没有回到驻地的边，此时函数输出-1：

