实验2、《贪心算法实验》

**一、实验目的**

1. 了解贪心算法思想

2. 掌握贪心法典型问题，如背包问题、作业调度问题等。

**二、实验内容**

1. 编写一个简单的程序，实现单源最短路径问题。

2. 编写一段程序，实现找零。

【问题描述】当前有面值分别为2角5分，1角，5分，1分的硬币，请给出找n分钱的最佳方案（要求找出的硬币数目最少）。

3. 编写程序实现多机调度问题

【问题描述】要求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台机器加工处理完成。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。作业不能拆分成更小的子作业。

**三、算法思想分析**

1.初始化将源点设计为红点集，其余点设计为蓝点，重复选择蓝点集中与源点路径最短的点加入红点集，更新剩余的蓝点集路径，直至蓝点集为空或者只剩下没有连通的点，那么源点到其余所有点的最短路径就出来了。

2.找零问题是典型的贪心问题，但是并不代表所有的找零都能用贪心算法找到最优解。只有满足贪心选择性质的找零才能找到最优解，本题满足贪心选择性质，直接先一直选面值最大的硬币，再一次减小即可。

3.先对作业按时长进行重排序，再依次找目前用时最短的机器安排工作并加上对应时长，最后总时长为机器中用时最长的那个时长。

**四、实验过程分析**

1.单源最短路径的算法思想并不难，但是在实际编码过程中还是有很多小问题需要注意，首先，一定要新建数组存储路径变化，因为后面计算路径时会用到原数组，如果直接在原数组上更改后面就找不到原数据了，那么就会出现偏差。其次就是建议先写个伪代码，判断的if-else语句比较多，容易搞混，在代码中一定要及时备注，某些代码的功能是什么，不然再次看代码时需要思考很久甚至忘记。

2.找零问题直接用while循环或者不断取余取模即可解决。

3.作业调度问题大致分为三步，一是排序，二是不断找最短时长的机器安排作业，三是找最长时间为作业完成时间。

**五、算法源代码及用户屏幕**

1.（1）算法源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

单源最短路径问题。

codeblocks C++

2018.10.26

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{ //原始带权有向图

int a[5][5] = {{0,10,-1,30,100},

{-1,0,50,-1,-1},

{-1,-1,0,-1,10},

{-1,-1,20,0,60},

{-1,-1,-1,-1,0}};

//单源最短路径图

int b[5][5] = {{0,10,-1,30,100},

{0,0,0,0,0},

{0,0,0,0,0},

{0,0,0,0,0},

{0,0,0,0,0}};

//创建数组红蓝点集

int red[5] = {0,-1,-1,-1,-1};

int red\_N = 0;

int blueToRed = 0;

int blue[5] = {-1, 1, 2, 3, 4};

int blue\_N = 0;

for(int i=0; i<5-1; i++){

//从蓝点集中选出与源点最短路径的蓝点

//初始化随机蓝点

int blue\_min;

for(int j=0; j<5; j++){

if(blue[j]>=0)

blue\_min = j;

}

for(int j=0; j<5; j++){

if(blue[j]>0 && b[i][j]>=0 && b[i][j]<b[i][blue\_min]){

//blue[j]>0表示j是蓝点，a[i][j]>0表示i,j两点连通

blue\_min = j;

}

}

//将蓝点集中与源点最短路径的点加入到红点集

red[++red\_N] = blue\_min;

//将该点从蓝点集中去除

blue[blue\_min] = -1;

//更改加入后的数据

for(int j=0; j<5; j++){

//判断j点是否为蓝点

if(blue[j]>=0){ //j是蓝点

if(a[blue\_min][j]>0){ //新加入的蓝点与j点连通

if(b[i][j]<0){ //上一步i,j两点不连通

b[i+1][j] = b[i][blue\_min]+a[blue\_min][j];

}

//上一步i,j两点连通且加起来路径比原来短

else if(b[i][blue\_min] + a[blue\_min][j] < b[i][j]){

b[i+1][j] = b[i][blue\_min]+a[blue\_min][j];

}

else{ //上一步i,j两点连通但加起来路径比原来长

b[i+1][j] = b[i][blue\_min]+a[blue\_min][j];

}

}

else{ //新加入蓝点与j点不连通

b[i+1][j] = b[i][j];

}

}

else{ //j不是蓝点

b[i+1][j] = b[i][j];

}

}

}

//输出原始带权有向图的邻接矩阵图

cout<<"Original:"<<endl;

//输出第一行

cout<<setw(3)<<" ";

for(int i=0; i<5; i++)

cout<<setw(3)<<i;

cout<<endl;

for(int i=0; i<5; i++){

//输出行号点数

cout<<setw(3)<<i;

for(int j=0; j<5; j++){

cout<<setw(3)<<a[i][j];

}

cout<<endl;

}

cout<<endl;

//输出最终结果

cout<<"Changed:"<<endl;

//输出第一行

cout<<setw(3)<<" ";

for(int i=0; i<5; i++)

cout<<setw(3)<<i;

cout<<endl;

for(int i=0; i<5; i++){

//输出行号点数

cout<<setw(3)<<i;

for(int j=0; j<5; j++){

cout<<setw(3)<<b[i][j];

}

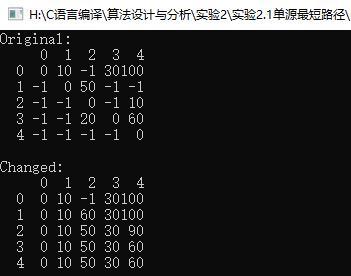
cout<<endl;

}

return 0;

}

（2）用户屏幕



2.（1）算法源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

实现面值分别为2角5分，1角，5分，1分的硬币找零

codeblocks C++

2018.10.26

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

int main()

{

int number;

int remain; //记录剩下的数额

cout<<"Please enter the amount of change you want (in cents)：";

cin>>number;

//建立a数组保存已有面值硬币，单位为分

double a[4]={25,10,5,1};

//建立b数组保存找回每个硬币的数量

double b[4]={0,0,0,0};

for(int i=0;i<4;i++){

int x = 0; //记录硬币数量

while(number>=a[i]){

x++;

number -= a[i];

}

b[i] = x;

}

cout<<"25 cents: "<<b[0]<<endl;

cout<<"10 cents: "<<b[1]<<endl;

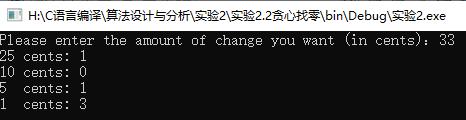
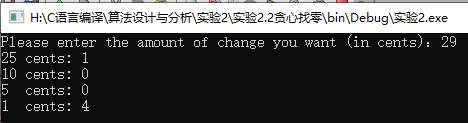
cout<<"5 cents: "<<b[2]<<endl;

cout<<"1 cents: "<<b[3]<<endl;

return 0;

}

（2）用户屏幕



3.（1）算法源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

多机调度问题。

codeblocks C++

2018.10.26

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

int shortestTime(int b[],int machines);

int longestTime(int b[],int machines);

void quicksort(int a[],int left,int right);

int \_partition(int a[],int left,int right);

int main()

{

//用户输入作业、机器数量，每个作业完成时长

int n;

cout<<"Please enter the number of jobs：";

cin>>n;

int a[n];

cout<<"Please enter the completion time for each job："<<endl;

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

cin>>a[i];

}

//对作业进行重排序

quicksort(a,0,n);

int machine;

cout<<"Please enter the number of machines：";

cin>>machine;

int b[machine];

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

b[i] = 0;

int shortT;

//找到目前最短时长的机器并安排作业

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

shortT = shortestTime(b,machine);

b[shortT] += a[i];

}

int longT;

longT = longestTime(b,machine);

cout<<"Total processing time："<<b[longT]<<endl;

return 0;

}

//快速排序算法

void quicksort(int a[],int left,int right){

int q;

if(left<right){

q=\_partition(a,left,right); //分解

quicksort(a,left,q); //排序

quicksort(a,q+1,right);

}

}

int \_partition(int a[],int left,int right){

//p,q分别代表left，right

int p,q;

p=left;

q=right;

//将关键数据暂时存储到变量中，空出a[p]的位置

int s = a[p];

while(p<q){

//从右端开始找，直到找到一个数小于s,然后赋值给a[p]

while(p<q && a[q]>=s)

q--;

if(p<q)

a[p]=a[q];

//从左端开始找，直到找到一个数大于s，然后将值赋给a[q]

while(p<q && a[p]<=s)

p++;

if(p<q)

a[q]=a[p];

}

a[p]=s;

return p;

}

//寻找最短时间

int shortestTime(int b[],int machines){

int flag = 0;

int small = b[flag];

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

if(b[i]<small){

small = b[i];

flag = i;

}

}

return flag;

}

//寻找最长时间

int longestTime(int b[],int machines){

int flag = 0;

int longest = b[flag];

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

if(b[i]>longest){

longest = b[i];

flag = i;

}

}

return flag;

}

（2）用户屏幕

