Лабораторная робота №3

*из курса "****Системное программное обеспечение - 2****"*

***Выполнил:***

*Банин Сергей Александрович*

***Группа*** *ИВ-02*

***Киев - 2013 р***

2. Скласти програму реалізації угорського методу для неоднорідної обчислювальної системи.

**Экспонированная вершина** – это вершина, которая не участвует в текущем паросочетании. Т.е. либо «незадействованный разработчик», либо «свободная задача».  
**Альтернирующая цепь** – это цепь, ребра которой попеременно лежат или не лежат в паросочетании. (…- владение технологией – назначенная задача – владение технологией – назначенная задача — …)  
**Альтернирующее дерево** – дерево, состоящее из альтернирующих цепей  
**Аугментальная цепь** – это такая альтернирующая цепь, начальная и конечная вершины которой экспонированы. Вот как называется то, что мы и ищем! =)  
**Аугментальное дерево** – соответственно дерево, в котором хотя бы одна из веток – это аугментальная цепь.

## Угорський метод.

Завдання планування зводиться до пошуку максимального паросполучення в зваженому дводольному графі, треба знайти максимальне паросполучення, щоб сума призначень була мінімальна. Угорський алгоритм складається з наступних кроків:

1) У кожному стовпці матриці знаходимо мінімальний елемент і віднімаємо від кожного елемента цього стовпця.

2) У кожному рядку матриці знаходимо мінімальний елемент і віднімаємо від кожного елемента стовпця.

3) У кожному рядку і стовпці повинен виявитися як мінімум один «0». З цієї матриці забираємо все «0» і замінюємо на «1».

4) Знаходимо максимальне паросполучення. Якщо ми отримаємо досконалий варіант, то перенесемо його на першу матрицю, отримаємо мінімальну суму і перейдемо на пункт 16 алгоритму). Якщо ні, виконуємо наступний пункт.

5) На цій матриці відзначаємо нулі, які увійшли до розв’язку з шагу 4 і ті, які не ввійшли (закреслюємо). Відзначаємо рядки, у яких є закреслювані «0».

6) Відмічаємо ті стовпці, в яких є закреслювані нулі відзначених рядків.

7) Відмічаємо рядок, що містить відзначений «0», який містить рядок, відзначений в попередньому кроці.

8) Відмічаємо стовпець, в якому є закреслюваний «0», який містить рядок, відзначений в попередньому кроці.

9) Відмічаємо пунктиром помічені стовпці і непомічені рядки.

10) Виписуємо елементи, через які проходять пунктирні лінії і серед них шукаємо мінімальний елемент.

11) Віднімаємо цей мінімальний елемент від тих стовпців матриці через які не проходять пунктирні лінії.

12) Додаємо цей елемент до тих рядків, через які проходять пунктирні лінії.

13) Об’єднувані одиниці переносимо на початкову матрицю.

14) Переходимо к пункту 4

15) Кінець розв’язку.

// hungarian.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include<stdio.h>

#include<conio.h>

#define max 20

int \_tmain(int argc, \_TCHAR\* argv[])

{int tr,size,m,av,i,j,r,c,diff,l=0,rz[max],cz[max],rc[max],cc[max];

int a[max][max],t[max][max],s[max][max],minr[max],minc[max],ct[max][max],minnew;

int cross[max][max],alloccount=0,minrow,mincol;

int nrz[max],ncz[max],nrc[max],ncc[max],nl=0;

int fzr[max][max],fzc[max][max];

int fz[max][max],k,flag=0,alloc[max][max];

int ffz[max][max],fffz[max][max];

int cost[max][max],costm=0;

//clrscr();

//inputting array

printf("Enter the size of matrix\n enter no of rows\n");

scanf("%d",&r);

printf("enter no of column\n");

scanf("%d",&c);

//inputting array

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

{

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

{ printf("Enter element a[%d][%d] - ",i,j);

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

}

}

if(r!=c)

{

if(r>c)

{

diff=r-c;

while(diff!=0)

{

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

{

a[i][c]=0;

}

c++;diff--;

}

}

if(r<c)

{

diff=c-r;

while(diff!=0)

{

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

{

a[r][i]=0;

}

r++;diff--;

}

}

}

if(r==c)

{

size=r;

}

//displaying array

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

{

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

{

printf(" %d ",a[i][j]);

}

printf("\n");

}

//making cost matrix

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

{

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

{

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

}

}

//initializing all matrices

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

{

rz[i]=cz[i]=rc[i]=cc[i]=0;

nrz[i]=ncz[i]=nrc[i]=ncc[i]=0;

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

{

fzr[i][j]=fzc[i][j]=fz[i][j]=ffz[i][j]=fffz[i][j]=0;

cross[i][j]=-1,alloc[i][j]=0;

}

}

//calculating min row

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

{

minr[i]=a[i][0];

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

{

if(minr[i]>a[i][j])

{

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

}

}

}

//subtracting min row from respective row

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

{

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

{

a[i][j]=a[i][j]-minr[i];

}

}

//calculating min column

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

{

minc[i]=a[0][i];

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

{

if(minc[i]>a[j][i])

{

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

}

}

}

//subtracting min column from respective column

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

{

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

{

a[j][i]=a[j][i]-minc[i];

}

}

//displaying reduced matrix

printf("\nreduced matrix\n");

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

{

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

{

printf(" %d ",a[i][j]);

}

printf("\n");

}

//copying reduced matrix

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

{

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

{

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

//t[i][j] is row column reduced matrix

}

}

//calculating row zero i.e. no of zeros in row

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

{

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

{

if(a[i][j]==0)

{

rz[i]=rz[i]+1;

}

}

}

tr=size;

if(size%2>0)

{

av=(size+1)/2;

}

else

{

av=size/2;

}

while(tr>=av)

{

//striking lines in row

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

{

if(rz[i]==tr)

{

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

{

a[i][j]=-1;

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

}

}

}

tr--;

}

//calculating colomn zero

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

{

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

{

if(a[i][j]==0)

{

cz[j]=cz[j]+1;

}

}

}

//striking lines in coloumn

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

{

if(cz[j]>=1)

{

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

{

a[i][j]=-1;

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

}

}

}

//calculating no of rows and columns crossed

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

{

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

{

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

{

rc[i]=rc[i]+1;

}

}

}

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

{

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

{

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

{

cc[i]=cc[i]+1;

}

}

}

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

{

if(rc[i]==size)

{

l=l+1;

}

}

if(l!=size)

{

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

{

if(cc[i]==size)

{

l=l+1;

}

}

}

//cross matrix

printf("cross matrix \n");

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

{

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

{

printf(" %d ",cross[i][j]);

}

printf("\n");

}

printf("\nno of lines striked = %d ",l);

if(l==size)

{

printf("\nwe are done\n");

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

{

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

{

if(t[i][j]==0)

{

nrz[i]=nrz[i]+1;//new row zero

}

}

}

minrow=nrz[0];

printf("nrz matrix\n");

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

{

printf(" %d \n",nrz[i]);

if(minrow>nrz[i])

{

minrow=nrz[i];

}

//printf("min row =%d",minrow);

}

while(alloccount!=size)

{

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

{

if(nrz[i]==minrow)

{

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

{

if(t[i][j]==0)

{

alloc[i][j]=1;

alloccount=alloccount+1;

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

{

t[i][k]=-1;

t[k][j]=-1;

}

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

{

nrz[i]=0;

}

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

{

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

{

if(t[i][j]==0)

{

nrz[i]=nrz[i]+1;//new row zero

}

}

}

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

{

if(nrz[i]==0)

{

nrz[i]=100;

}

}

//minrow=nrz[0];

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

{

if(minrow>nrz[i])

{

minrow=nrz[i];

}

}

}

}

}

}

}

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

{

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

{

if(alloc[i][j]==1)

{

printf("\ntask %d is allocated to processor %d \n",i+1,j+1);

costm=costm+cost[i][j];

}

}

}

if(l==size&&alloccount==size)

{

printf("we have done it! \n");

printf("cost = %d \n",costm);

flag=1;

}

}

if(l!=size&&flag==0)//if no of line striked is not equal to the size

{

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

{

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

{

printf("%d",a[i][j]);

}

}

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

{

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

{

if(a[i][j]>0)

{

minnew=a[i][j];

}

}

}

//finding new min

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

{

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

{

if(a[i][j]>0&&minnew>a[i][j])

minnew=a[i][j];

}

}

//subtracting new min from reduced

printf("\nmin new = %d\n",minnew);

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

{

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

{

if(a[i][j]>0) //mistake in this part

a[i][j]=a[i][j]-minnew;

}

}

//copying reduced part in original reduced

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

{

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

{

if(a[i][j]>=0)

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

}

}

//adding min new to cross point

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

{

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

{

if(cross[i][j]==1)

{

t[i][j]=t[i][j]+minnew;

}

}

}

//final reduced

printf("final reduced\n");

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

{

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

{

printf(" %d ",t[i][j]);

}

printf("\n");

}

for(i=0;i<size;i++)//saving final reduced

{

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

{

s[i][j]=t[i][j];

}

}

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

{

nrz[i]=0,ncz[i]=0;rz[i]=0,cz[i]=0;//resseting row zero

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

{

cross[i][j]=-1;

}

}

//calculating colomn zero

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

{

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

{

if(a[i][j]==0)

{

cz[j]=cz[j]+1;

}

}

}

tr=size;

if(size%2>0)

{

av=(size+1 )/2;

}

else

{

av=size/2;

}

while(tr>=av)

{

//striking lines in col

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

{

if(cz[j]==tr)

{

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

{

a[i][j]=-1;

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

}

}

}

tr--;

}

//calculating row zero i.e. no of zeros in row

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

{

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

{

if(a[i][j]==0)

{

rz[i]=rz[i]+1;

}

}

}

//striking lines in row

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

{

if(rz[j]>=1)

{

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

{

a[i][j]=-1;

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

}

}

}

//cros mat

printf("new cross");

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

{

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

{

printf("%d",cross[i][j]);

}

printf("\n");

}

//calculating new row zero

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

{

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

{

if(t[i][j]==0)

{

nrz[i]=nrz[i]+1;

}

}

}

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

{

if(nrz[i]>=2)

{

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

{

t[i][j]=-1;

}

}

}

//restoring t[i][j]

for(i=0;i<size;i++)//saving final reduced

{

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

{

t[i][j]=s[i][j];

}

}

//calculating new colomn zero

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

{

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

{

if(t[j][i]==0)

{

ncz[j]=ncz[j]+1;

}

}

}

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

{

if(ncz[j]>=1)

{

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

{

t[i][j]=-1;

}

}

}

//calculating no of rows and columns crossed in new matrix

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

{

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

{

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

{

nrc[i]=nrc[i]+1;

}

}

}

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

{

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

{

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

{

ncc[i]=ncc[i]+1;

}

}

}

//counting no of lines striked

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

{

if(nrc[i]==size&&ncc[i]==size)

{

nl=nl+1;

}

}

printf("\nno of lines striked = %d",nl);

if(nl==size)

{

printf("now we are done \n");

}

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

{

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

{

if(t[i][j]==0)

{

nrz[i]=nrz[i]+1;//new row zero

}

}

}

minrow=nrz[0];

printf("nrz matrix\n");

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

{

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

if(minrow>nrz[i])

{

minrow=nrz[i];

}

}

printf("minrow = %d",minrow);

alloccount=0;

while(alloccount!=size&&flag!=1)

{

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

{

if(nrz[i]==minrow&&flag!=1)

{

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

{

if(t[i][j]==0)

{

alloc[i][j]=1;

alloccount=alloccount+1;

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

{

t[i][k]=-1;

t[k][j]=-1;

}

}

}

if(j==size)

{

flag=1;

break;

}

}

/\*else

{

for(p=0;p<size;p++)

{

for(q=0;q<size;q++)

{

if(t[p][q]==0)

{

nrz[p]=nrz[p]+1;//new row zero

}

}

}

minrow=nrz[0];

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

{

if(minrow>nrz[i])

{

minrow=nrz[i];

}

}

}\*/

}

}

if(flag==1)//it means row allocation failed

{

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

{

ncz[i]=0;

}

//restoring t[i][j]

for(i=0;i<size;i++)//saving final reduced

{

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

{

t[i][j]=s[i][j];

}

}

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

{

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

{

if(t[j][i]==0)

{

ncz[i]=ncz[i]+1;//new row zero

}

}

}

mincol=ncz[0];

//printf("ncz matrix\n");

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

{

//printf(" %d ",ncz[i]);

if(mincol>ncz[i])

{

mincol=ncz[i];

}

}

//printf("mincol = %d",mincol);

alloccount=0;

while(alloccount!=size)

{

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

{

if(ncz[i]==mincol)

{

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

{

if(t[j][i]==0)

{

alloc[j][i]=1;

alloccount=alloccount+1;

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

{

t[j][k]=-1;

t[k][i]=-1;

}

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

{

ncz[i]=0;

}

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

{

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

{

if(t[j][i]==0)

{

ncz[i]=ncz[i]+1;//new row zero

}

}

}

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

{

if(ncz[i]==0)

{

ncz[i]=100;

}

}

mincol=ncz[0];

//printf("ncz matrix\n");

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

{

//printf(" %d ",ncz[i]);

if(mincol>ncz[i])

{

mincol=ncz[i];

}

}

//printf("mincol = %d",mincol);

}

}

}

}

}

}

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

{

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

{

if(alloc[i][j]==1)

{

printf("\ntask %d is allocated to processor %d \n",i+1,j+1);

costm=costm+cost[i][j];

}

}

}

}

if(alloccount==size)

{

printf("we have done it! \n");

printf("cost = %d \n",costm);

}

getch();

return 0;

}

Enter the size of matrix

enter no of rows

3

enter no of column

3

Enter element a[0][0] - 5

Enter element a[0][1] - 6

Enter element a[0][2] - 4

Enter element a[1][0] - 6

Enter element a[1][1] - 56

Enter element a[1][2] - 3

Enter element a[2][0] - 12

Enter element a[2][1] - 234

Enter element a[2][2] - 56

5 6 4

6 56 3

12 234 56

reduced matrix

1 0 0

3 51 0

0 220 44

cross matrix

1 0 1

0 -1 0

0 -1 0

no of lines striked = 3

task 1 is allocated to processor 2

task 2 is allocated to processor 3

task 3 is allocated to processor 1

we have done it!

cost = 21

we have done it!

cost = 21