Khemis Miliana University
Faculty of Science and Technology
Mathematics and Computer Science Department

جامعة جيلالي بونعامة– خميس مليانة
كلية العلوم والتكنولوجيا
قسم الرياضيات و الاعلام الالي

Module : Operations Research 1

Responsible: Dr. I. Ait Abderrahim

Tutorial sheet 5

Problem: Assignment problem

Task: **Hungarian Method**

Solution: cpp 1

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<iomanip.h>
#define MAX 50
enum boolean{FALSE,TRUE};

class HungarianMethod{
    int data[MAX][MAX];
    int allocation[MAX][MAX];
    int no_of_rows,no_of_columns;
    int bal_stat;

    public:
        HungarianMethod(){
            int i,j;
            for(i=0;i<MAX;i++){
                for(j=0;j<MAX;j++){
                    data[i][j]=0;
                    allocation[i][j]=0;
                }
            }
            no_of_rows=no_of_columns=bal_stat=0;
        }
        void setRow(int no){no_of_rows=no;}
        void setColumn(int no){no_of_columns=no;}
        void getData();
        void makeAllocation();
        void display();
        void rowMinima(int [][MAX],int,int);
        void columnMinima(int [][MAX],int,int);
        boolean checkValue(int,int,int [][MAX]);
};
void HungarianMethod::getData(){
    int i,j;
    cout<<"enter cost Metrix :\n";
    for(i=0;i<no_of_rows;i++){
        cout<<"enter "<<i+1<<" row :";
        for(j=0;j<no_of_columns;j++)
            cin>>data[i][j];
    }
}
void copyArray(int startRow,int startCol,int endRow,int endCol,int temp[][MAX],int
start1row,int start1col,int ans[][MAX]){
    int i,j,k,l;
    for(i=startRow,k=start1row;i<endRow;i++,k++)
        for(j=startCol,l=start1col;j<endCol;j++,l++)
            ans[k][l]=temp[i][j];
}
int getMinVal(int temp[],int no){
    int min=temp[0];
    for(int i=0;i<no;i++)
        if(min>temp[i])
            min=temp[i];
```

Module : Operations Research 1

Responsible: Dr. I. Ait Abderrahim

```
    return min;
}
int getPosition(int temp[],int no,int value){
    for(int i=0;i<no;i++)
        if(temp[i]==value)
            return i;
    return -1;
}
int countVal(int temp[],int no,int value){
    int i,sum=0;
    for(i=0;i<no;i++)
        if(temp[i]==value)
            sum++;
    return sum;
}
void HungarianMethod::rowMinima(int temp[][MAX],int row,int col){
    int i,j,min;
    for(i=0;i<row;i++){
        min=9999;
        for(j=0;j<col;j++)
            if(min>temp[i][j])
                min=temp[i][j];
        for(j=0;j<col;j++)
            temp[i][j]-=min;
    }
}
void HungarianMethod::columnMinima(int temp[][MAX],int row,int col){
    int i,j,min;
    for(i=0;i<row;i++){
        min=9999;
        for(j=0;j<col;j++)
            if(min>temp[j][i])
                min=temp[j][i];
        for(j=0;j<col;j++)
            temp[j][i]-=min;
    }
}
boolean HungarianMethod::checkValue(int row,int col,int temp[][MAX]){
    int i,j;
    for(i=0;i<row;i++)
        for(j=0;j<col;j++)
            if(temp[i][j]==0)
                return TRUE;
    return FALSE;

}
void HungarianMethod::makeAllocation(){
    int temp_data[MAX][MAX]={0};
    int i,j;
    if(no_of_rows>no_of_columns){
        for(i=0;i<no_of_rows;i++)
            data[i][no_of_columns]=0;
        no_of_columns++;
        bal_stat=1;
    }else if(no_of_rows<no_of_columns){
        for(i=0;i<no_of_columns;i++)
            data[no_of_rows][i]=0;
        no_of_rows++;
        bal_stat=2;
    }
    copyArray(0,0,no_of_rows,no_of_columns,data,0,0,temp_data);
    rowMinima(temp_data,no_of_rows,no_of_columns);
    columnMinima(temp_data,no_of_rows,no_of_columns);
```

Module : Operations Research 1

Responsible: Dr. I. Ait Abderrahim

```
    int min,pos,count;
    int tempCol[MAX]={0};
    while(checkValue(no_of_rows,no_of_columns,temp_data)){
        for(i=0;i<no_of_rows;i++){
            count=countVal(temp_data[i],no_of_columns,0);
            if(count==1){
                pos=getPosition(temp_data[i],no_of_columns,0);
                allocation[i][pos]=data[i][pos];
                for(j=0;j<no_of_rows;j++)
                    if(temp_data[j][pos]==0)
                        temp_data[j][pos]=9999;
            }
        }
        for(i=0;i<no_of_rows;i++){
            for(j=0;j<no_of_columns;j++)
                tempCol[j]=temp_data[j][i];
            count=countVal(tempCol,no_of_rows,0);
            if(count==1){
                pos=getPosition(tempCol,no_of_rows,0);
                allocation[i][pos]=data[i][pos];
                for(j=0;j<no_of_columns;j++)
                    if(temp_data[pos][j]==0)
                        temp_data[pos][j]=9999;
            }

        }
    }
}
void HungarianMethod::display(){
    int i,j;
    cout<<"\nGiven Cost Metrix :\n";
    for(i=0;i<no_of_rows;i++)
        cout<<"\t"<<char(65+i);
    cout<<endl;
    for(i=0;i<no_of_rows;i++){
        cout<<i+1;
        for(j=0;j<no_of_columns;j++)
            cout<<"\t"<<data[i][j];
        cout<<endl;
    }
    if(bal_stat!=0){
        cout<<"\n\nhere the give cost metrix is not squar Matrix\n";
        cout<<"so this is a unbalance problem and as a solution";
        cout<<"\n we have add an extra "<<((bal_stat==1)?"column":"row")<<" with 0
value in each\n";
    }
    cout<<"\n\nOpportunity Matrix :\n";
    rowMinima(data,no_of_rows,no_of_columns);
    columnMinima(data,no_of_rows,no_of_columns);
    for(i=0;i<no_of_rows;i++){
        for(j=0;j<no_of_columns;j++)
            cout<<"\t"<<data[i][j];
        cout<<endl;
    }
    int sum=0;
    cout<<"\n\nJobs\t:\tMachine\t:\tCost\n";
    for(i=0;i<no_of_rows;i++)
        for(j=0;j<no_of_columns;j++)
            if(allocation[i][j]!=0){
                cout<<i+1<<"\t:\t"<<char(65+j)<<"\t:\t"<<allocation[i][j];
                sum+=allocation[i][j];
                cout<<endl;
            }
```

Module : Operations Research 1

Responsible: Dr. I. Ait Abderrahim

```
    cout<<"\nTotal Assignment Cost = "<<sum<<" RS.";
}
void main(){
    clrscr();
    HungarianMethod hm;
    int row,col;

    cout<<"enter no of row :";
    cin>>row;
    cout<<"enter no of column :";
    cin>>col;

    hm.setRow(row);
    hm.setColumn(col);
    hm.getData();
    clrscr();
    hm.makeAllocation();
    hm.display();
    getch();
}
```

Solution: Python

```
import numpy as np
from scipy.optimize import linear_sum_assignment

def hungarian_algorithm(cost_matrix):
    """
    This function implements the Hungarian algorithm to solve the assignment
problem.

    Parameters:
    cost_matrix (numpy.ndarray): The cost matrix representing the assignment
problem.

    Returns:
    tuple: A tuple containing two numpy arrays. The first array represents the row
indices and the second array represents the column indices of the assigned tasks.
    """
    try:
        # Check if the cost matrix is a 2-dimensional numpy array
        if not isinstance(cost_matrix, np.ndarray) or cost_matrix.ndim != 2:
            raise TypeError("The cost matrix must be a 2-dimensional numpy array")

        # Apply the Hungarian algorithm to find the optimal assignment
        row_indices, col_indices = linear_sum_assignment(cost_matrix)

        return row_indices, col_indices
    except Exception as e:
        # Log the error
        print(f"Error: {e}")
        return None, None

# Test the code
cost_matrix = np.array([[4, 2, 8], [3, 5, 6], [7, 1, 9]])
row_indices, col_indices = hungarian_algorithm(cost_matrix)
print("Optimal Assignment:")
for i in range(len(row_indices)):
    print(f"Task {row_indices[i]+1} -> Worker {col_indices[i]+1}")
```

Correct answer: