OR

ASSIGNMENT 1

Title : The Transportation Problem

• Problem Statement :

Milk in a milk shed area is collected on 3 routes A, B, C There are 4 chilling centers P, Q, R, S where milk is kept before transporting it to a milk center. Each route is able to supply on an avg. one thousand litres of milk per day. The supply of milk on routes A, B, C are 150, 160, 90 thousand litres resp. The cost of transporting thousand litres of milk from each route to each chilling centers differs according to dist. Cost in Rs is shown in table below.

Routes	Chilling centers			
	P	Q	R	S
A	16	18	21	12
B	17	19	14	13
C	32	11	15	10

Minimize the total transportation cost.

Objective :

To understand the implementation of various transportation problem methods like North-West corner, least cost, Vogel's Approx. method.

S/W and H/W requirements

Visual Studio Code

2GB RAM, 500GB HDD, Windows OS, i5 processor.

Outcomes:
After completion of this assignment of various methods the students will be able to calculate the minimum transportation cost

Theory:

North-West Corner Method.

This is a method adopted to compute the initial cost of feasible solution of the transportation problem. The name is given to this method because the basic variables are selected from extreme left corner.

~~Algorithm~~ Code:

```
def north_west_corner(supply, demand):
    supply_copy = supply.copy()
    demand_copy = demand.copy()
    j = 0
    i = 0
    bfs = []
    while len(bfs) < leng(supply) + len(demand) -1:
        s = supply_copy[i]
        d = demand_copy[j]
        v = min(s,d)
        supply_copy[i] -= v.
        demand_copy[j] -= v
        bfs.append((i,j), v))
```

if supply_copy[i] == 0 and i < len(supply)-1:
    i += 1
elif demand_copy[j] == 0 and j < len(demand)-1:
    j += 1
return bfs

Least Cost Method:

The least cost Method is another method used to obtain the initial feasible solution for the transportation problem. Here, the allocation begins with the cell which has the minimum cost. The lower cost cells are chosen over the higher cost cell with the objective to have the least cost of transportation.

Algorithm:

Step 1: Select the cell having minimum unit cost cij and allocate as much as possible ie. min(si, dj)

Step 2: a. Subtract this minimum value from supply si and demand dj.
b. If the supply si is 0, then cross that row and if the demand dj is 0 then cross the column.
c. If min unit cost cell is not unique, then select the cell where maximum allocation can be possible.

Step 3 : Repeat this steps for all uncrossed rows and columns until all supply and demand values are zero.

Vogel's Approximation Method.

VAM is an iterative procedure calculated to find out the initial feasible solution of the transportation problem like least cost Method, here also the shipping cost is taken into consideration, but in relative sense.

Algorithm :

Step 1 : Find the cells having smallest and next to smallest cost in each row and write the difference along the side of the table in row penalty.

Step 2 : Find the cells having smallest cost in each column and write the diff along the side of the table in each column penalty.

Step 3 : Select the row or column with the maximum penalty and find cell that has least cost in selected row or column. Allocate as much as possible in this cell.

Step 4 : Adjust the supply and demand and cross-out the satisfied row or column.

Page No.
Date

Step 5: Repeat this steps unit all supply and demand values are 0.

Conclusion:

We studied and implemented the north-west corner, least cost and Vogel's approximation method for calculating the transportation cost. Where VAM gave the least cost among the 3 costs for transportation.

Code :

Northwest-corner Method

```
class NorthWestCornerMethod
{
   public static void main(String args[])
   {
      int [][]arr = {{16,18,21,12},{17,19,14,13},{32,11,15,10},};
      int []supply = {150,160,90};
      int []demand = {140,120,90,50};
      int [][]allocation = new int[3][4];
      int row = 3;
      int col = 4;

      System.out.println("Input matrix is ");
      for(int i=0;i<3;i++)
      {
         for(int j=0;j<4;j++)
         {
            System.out.print(arr[i][j]+" ");
         }
         System.out.println();
      }
      int i_idx=0,j_idx=0;
      for(int j=0;j<4;j++)
      {
         while(demand[j]!=0)
         {
            for(int i=0;i<3;i++)
            {
               if(supply[i]>0 && demand[j]>0)
               {
                  i_idx = i;
                  j_idx = j;
                  break;
               }
            }
            if(demand[j_idx] > supply[i_idx])
            {
               demand[j_idx] = demand[j_idx] - supply[i_idx];
               allocation[i_idx][j_idx] = supply[i_idx];
               supply[i_idx] = 0;
            }
            else if(supply[i_idx] > demand[j_idx])
```

```
            {
               supply[i_idx] = supply[i_idx] - demand[j_idx];
               allocation[i_idx][j_idx] = demand[j_idx];
               demand[j_idx] = 0;
            }
            else if(supply[i_idx] == demand[j_idx])
            {
               allocation[i_idx][j_idx] = supply[i_idx];
               supply[i_idx] = 0;
               demand[j_idx] = 0;
            }
         }
      }

      System.out.println("Output matrix is ");
      for(int i=0;i<3;i++)
      {
         for(int j=0;j<4;j++)
         {
            System.out.print(allocation[i][j]+" ");
         }
         System.out.println();
      }

      int cost = 0;
      int value = 0;

      for(int i=0;i<row;i++)
      {
         for(int j=0;j<col;j++)
         {
            if(allocation[i][j] > 0)
            {
               value = arr[i][j]*allocation[i][j];
               cost += value;
            }
         }
      }

      System.out.println("Transportation cost is " + cost);

   }
}
```

Output:

```
Input matrix is
17 19 14 13
32 11 15 10
Output matrix is
140 10 0 0
0 110 50 0
0 0 40 50
Transportation cost is 6310
```

Least Cost Method:

```
import static java.util.Arrays.stream;

class LeastCostMethod
{
   public static void main(String args[])
   {
      int [][]arr = {{16,18,21,12},{17,19,14,13},{32,11,15,10},};
      int []supply = {150,160,90};
      int []demand = {140,120,90,50};
      int [][]allocation = new int[3][4];
      int row = 3;
      int col = 4;
      boolean []DoneRow = {false,false,false};
      boolean []DoneCol = {false,false,false,false};
      System.out.println("Input matrix is ");
      for(int i=0;i<3;i++)
      {
         for(int j=0;j<4;j++)
         {
            System.out.print(arr[i][j]+" ");
         }
         System.out.println();
      }

      int [][]arrcopy = new int[arr.length][];
      for(int i=0;i<arrcopy.length;i++)
      {
         arrcopy[i] = new int[arr[i].length];
         for(int j=0;j<arrcopy[i].length;j++)
         {
            arrcopy[i][j] = arr[i][j];
         }
```

```
}
/*System.out.println("Copied array is :");
for(int i=0;i<3;i++)
{
   for(int j=0;j<4;j++)
   {
      System.out.print(arrcopy[i][j]+" ");
   }
   System.out.println();
}*/

int i_idx=0;
int j_idx=0;

int supplyLeft = stream(supply).sum();
//System.out.println("Supply Left :"+supplyLeft);
while(supplyLeft>0)
{
   int value = 100;
   for(int i=0;i<row;i++)
   {
      for(int j=0;j<col;j++)
      {
         if(value>arrcopy[i][j])
         {
            value = arrcopy[i][j];
            //System.out.println(value);
            i_idx = i;
            j_idx = j;
         }

      }
   }
   /*System.out.println("Lowest Value : "+arrcopy[i_idx][j_idx]);
   arrcopy[i_idx][j_idx]=100;
   System.out.println("Copied array is :");
   for(int i=0;i<3;i++)
   {
      for(int j=0;j<4;j++)
      {
         System.out.print(arrcopy[i][j]+" ");
      }
      System.out.println();
   }*/
```

```
if(demand[j_idx] > supply[i_idx])
{
   demand[j_idx] = demand[j_idx] - supply[i_idx];
   allocation[i_idx][j_idx] = supply[i_idx];
   supplyLeft = supplyLeft - supply[i_idx];
   supply[i_idx] = 0;
}
else if(supply[i_idx] > demand[j_idx])
{
   supply[i_idx] = supply[i_idx] - demand[j_idx];
   allocation[i_idx][j_idx] = demand[j_idx];
   supplyLeft = supplyLeft - demand[j_idx];
   demand[j_idx] = 0;
}
else if(supply[i_idx] == demand[j_idx])
{
   allocation[i_idx][j_idx] = supply[i_idx];
   supplyLeft = supplyLeft - supply[i_idx];
   supply[i_idx] = 0;
   demand[j_idx] = 0;
}

/*System.out.println("Output matrix is ");
for(int i=0;i<3;i++)
{
   for(int j=0;j<4;j++)
   {
      System.out.print(allocation[i][j]+" ");
   }
   System.out.println();
}
System.out.println("Supply Left :"+supplyLeft);
System.out.println("Supply :");
for(int i=0;i<row;i++)
{
   System.out.print(supply[i]+ " ");
}
System.out.println("Demand :");
for(int i=0;i<col;i++)
{
   System.out.print(demand[i]+ " ");
}*/

int toRemoveRow  = 5;
//System.out.println("Supply :");
```

```
    for(int i=0;i<row;i++)
    {
        if(DoneRow[i] == false && supply[i]==0)
        {
            toRemoveRow = i;
            DoneRow[i]=true;
        }
    }
    if(toRemoveRow != 5)
    {
        for(int i=0;i<col;i++)
        {
            arrcopy[toRemoveRow][i] = 100;
        }
    }

    int toRemoveCol = 5;
    //System.out.println("Demand :");
    for(int i=0;i<col;i++)
    {
        if(DoneCol[i] == false && demand[i]==0)
        {
            toRemoveCol = i;
            DoneCol[i] = true;
        }
    }
    if(toRemoveCol != 5)
    {
        for(int i=0;i<row;i++)
        {
            arrcopy[i][toRemoveCol] = 100;
        }
    }

}



/*for(int i=0;i<row;i++)
{
    System.out.println(supply[i]);
}

for(int i=0;i<col;i++)
{
```

```
            System.out.println(demand[i]);
        }*/
        System.out.println("Output matrix is ");
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<4;j++)
            {
                System.out.print(allocation[i][j]+" ");
            }
            System.out.println();
        }

        int cost = 0;
        int value1 = 0;

        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                if(allocation[i][j] > 0)
                {
                    value1 = arr[i][j]*allocation[i][j];
                    cost += value1;
                }
            }
        }

        System.out.println("Transportation cost is " + cost);

    }
}
```

Output:

```
Input matrix is
16 18 21 12
17 19 14 13
32 11 15 10
Output matrix is
140 10 0 0
0 70 90 0
0 40 0 50
Transportation cost is 5950
```

Vogel’s Approximation Method:

```
#include<iostream>
#include<stdio.h>
#include<conio.h>
#include<iomanip>
#include<stdlib.h>
#define MAX 5
using namespace std;
enum boolean{FALSE,TRUE};
class voggelsmethod{
   int data[MAX][MAX];
   int requered[MAX];
   int capacity[MAX];
   int allocation[MAX][MAX];
   int no_of_rows,no_of_columns,no_of_allocation;
   public:
      lcmethod(){
         for(int i=0;i<MAX;i++){
            capacity[i]=0;
            requered[i]=0;
            for(int j=0;j<MAX;j++){
               data[i][j]=0;
               allocation[i][j]=0;
            }
         }
         no_of_rows=no_of_columns=no_of_allocation=0;
      }
      void setColumn(int no){no_of_columns=no;};
      void setRow(int no){no_of_rows=no;}
      void getData();
      void getCapacity();
      void getRequiredValue();
      void makeAllocation();
      boolean checkValue(int [],int);
      int getMinVal(int [],int);
      int getTotalMinVal(int [],int,int);
      int getMinValsPos(int,int [],int);
      void display();
      int getPanalty(int [],int);
};
int voggelsmethod::getPanalty(int array[],int no){
   int i,j,temp;
   for(i=0;i<no;i++)
      for(j=i+1;j<no;j++)
```

```
            if(array[i]>array[j]){
                temp=array[i];
                array[i]=array[j];
                array[j]=temp;
            }
    return array[1]-array[0];
}
int voggelsmethod::getMinVal(int array[],int no){
    int min=array[0];
    for(int i=0;i<no;i++)
        if(array[i]<min)
            min=array[i];
    return min;
}
int voggelsmethod::getMinValsPos(int value,int temp_data[],int no){
    int k=0;
    for(int i=0;i<no;i++)
        if(temp_data[i]==value)
            return i;
    return -1;
}
int voggelsmethod::getTotalMinVal(int array[],int n,int value){
    int no=0;
    for(int i=0;i<n;i++)
        if(array[i]==value)
                no++;
    return no;
}
boolean voggelsmethod::checkValue(int arr[],int no){
    for(int i=0;i<no;i++)
        if(arr[i]!=0)
            return FALSE;
    return TRUE;
}
void arrayCopy(int start,int end,int array1[],int start1,int array2[]){
    for(int i=start,j=start1;i<end;i++,j++)
        array2[j]=array1[i];
}
int getTotal(int array[],int no){
    int sum=0;
    for(int i=0;i<no;i++)
        sum+=array[i];
    return sum;
}
void copy2DArray(int startRow,int startCol,int endRow,int endCol,int
```

```
array[][MAX],int start1Row,int start1Col,int ans[][MAX]){
   for(int i=startRow,k=start1Row;i<endRow;i++,k++)
      for(int j=startCol,l=start1Col;j<endCol;j++,l++)
         ans[k][l]=array[i][j];
}
int getMaxVal(int array[MAX],int no){
   int max=0;
   for(int i=0;i<no;i++)
      if(array[i]>max)
         max=array[i];
   return max;
}
int getMaxValPos(int array[MAX],int no,int value){
   for(int i=0;i<no;i++)
      if(value==array[i])
         return i;
   return -1;
}
void voggelsmethod::makeAllocation(){
   int i=0,j=0,min,total_min;
   int temp_requered[MAX]={0};
   int temp_capacity[MAX]={0};
   int temp_data[MAX][MAX]={0};
   int position[MAX]={0};
   int dataPos[MAX]={0};
   int sum_of_cap,sum_of_req;
   sum_of_cap=getTotal(capacity,no_of_rows);
   sum_of_req=getTotal(requered,no_of_columns);
   if(sum_of_cap!=sum_of_req){
      if(sum_of_cap>sum_of_req){
         for(j=0;j<no_of_rows;j++)
            data[j][no_of_columns]=0;
         requered[no_of_columns]=sum_of_cap-sum_of_req;
         no_of_columns++;
      }
      else{
         for(j=0;j<no_of_columns;j++)
            data[no_of_rows][j]=0;
         capacity[no_of_rows]=sum_of_req-sum_of_cap;
         no_of_rows++;
      }
   }
   i=j=0;
   arrayCopy(0,no_of_rows,capacity,0,temp_capacity);
   arrayCopy(0,no_of_columns,requered,0,temp_requered);
```

```
    copy2DArray(0,0,no_of_rows,no_of_columns,data,0,0,temp_data);
    int rowPanalty[MAX]={0};
    int colPanalty[MAX]={0};
    int panaltyData[MAX]={0},n=0;
    while(!checkValue(temp_capacity,no_of_rows) ||
!checkValue(temp_requered,no_of_columns)){

        for(i=0;i<no_of_rows;i++){
            arrayCopy(0,no_of_columns,temp_data[i],0,panaltyData);
            if(temp_capacity[i]!=0)
                rowPanalty[i]=getPanalty(panaltyData,no_of_columns);
            else
                rowPanalty[i]=0;
        }
        for(i=0;i<no_of_columns;i++){
            for(j=0;j<no_of_rows;j++)
                panaltyData[j]=temp_data[j][i];
            if(requered[i]!=0)
                colPanalty[i]=getPanalty(panaltyData,no_of_rows);
            else
                colPanalty[i]=0;
        }
        int maxRowPanalty=getMaxVal(rowPanalty,no_of_rows);
        int maxColPanalty=getMaxVal(colPanalty,no_of_columns);
        int maxPanRow[MAX]={0};
        int maxPanCol[MAX]={0};
        if(maxRowPanalty>maxColPanalty){
            i=getMaxValPos(rowPanalty,no_of_rows,maxRowPanalty);
            for(j=0;j<no_of_columns;j++)
                maxPanRow[j]=temp_data[i][j];
            min=getMinVal(maxPanRow,no_of_columns);
            j=getMinValsPos(min,maxPanRow,no_of_columns);
        }
        else{
            j=getMaxValPos(colPanalty,no_of_columns,maxColPanalty);
            for(i=0;i<no_of_rows;i++)
                maxPanCol[i]=temp_data[i][j];
            min=getMinVal(maxPanCol,no_of_rows);
            i=getMinValsPos(min,maxPanCol,no_of_rows);
        }

        if(temp_capacity[i]>temp_requered[j]){
            allocation[i][j]=temp_requered[j];
            for(int k=0;k<no_of_rows;k++)
                temp_data[k][j]=9999;
```

```
            temp_capacity[i]-=temp_requered[j];
            temp_requered[j]=0;
        }
        else if(temp_capacity[i]<temp_requered[j]){
            allocation[i][j]=temp_capacity[i];
            for(int k=0;k<no_of_columns;k++)
                temp_data[i][k]=9999;
            temp_requered[j]-=temp_capacity[i];
            temp_capacity[i]=0;
        }
        else{
            int k;
            allocation[i][j]=temp_capacity[i];
            for(k=0;k<no_of_rows;k++)
                temp_data[k][j]=9999;
            for(k=0;k<no_of_columns;k++)
                temp_data[i][k]=9999;
            temp_requered[j]=temp_capacity[i]=0;
        }
        n++;
    }
    no_of_allocation=n;
}
void voggelsmethod::getCapacity(){
    cout<<"enter capacity for each source : \n";
    for(int i=0;i<no_of_rows;i++){
        cout<<"s"<<i+1<<" : ";
        cin>>capacity[i];
    }
}
void voggelsmethod::getRequiredValue(){
    cout<<"enter required unit value for each destination : \n";
    for(int i=0;i<no_of_columns;i++){
        cout<<"d"<<i+1<<" : ";
        cin>>requered[i];

    }
}
void voggelsmethod::display(){
    int i;
    cout<<"\ngiven data :\n";
    cout<<setw(9);
    for(i=0;i<no_of_columns;i++)
        cout<<"D"<<i+1<<setw(4);
    cout<<setw(5)<<"cap"<<endl<<setw(0);
```

```
    for(i=0;i<no_of_rows;i++){
        cout<<setw(3)<<"S"<<i+1;
        for(int j=0;j<no_of_columns;j++)
            cout<<setw(5)<<data[i][j];
        cout<<setw(5)<<capacity[i]<<endl;
    }
    cout<<setw(4)<<"req";
    for(i=0;i<no_of_columns;i++)
        cout<<setw(5)<<requered[i];

    cout<<"\n\n after allocation :\n";
    for(i=0;i<no_of_rows;i++){
        for(int j=0;j<no_of_columns;j++){
            if(allocation[i][j]<200)
                cout<<setw(5)<<data[i][j]<<"*"<<setw(2)<<allocation[i][j];
            else
                cout<<setw(8)<<data[i][j];
        }
        cout<<endl;
    }
    int k=0,sum=0;
    for(i=0;i<no_of_rows;i++){
        for(int j=0;j<no_of_columns;j++){
            if(allocation[i][j]<200 && allocation[i][j]>0){
                cout<<"("<<data[i][j]<<" * "<<allocation[i][j]<<")";
                if(k<no_of_allocation-1){
                    cout<<"+";
                    k++;
                }
                sum+=data[i][j]*allocation[i][j];
            }
        }
    }
    cout<<"\nanswer : "<<sum;
    if((no_of_rows+no_of_columns-1)==no_of_allocation){
        cout<<"\nhere "<<no_of_rows<<"+"<<no_of_columns<<"-1
="<<no_of_allocation<<" no. of allocations";
        cout<<"\n so this problem is non-degenarated solution";
    }
    else{
        cout<<"\nhere "<<no_of_rows<<"+"<<no_of_columns<<"-1
!="<<no_of_allocation<<"no of allocations";
        cout<<"\n so this problem is degenarated solution";
    }
}
```

```
void voggelsmethod::getData(){
   cout<<"enter source to destination data:"<<endl;
   for(int i=0;i<no_of_rows;i++){
      cout<<"enter "<<i<<"th row : ";
      for(int j=0;j<no_of_columns;j++){
         cin>>data[i][j];
      }
   }
}
int main(){

   voggelsmethod v;
   int r,c;
   cout<<"enter no of Rows : ";
   cin>>r;
   cout<<"enter no of columns : ";
   cin>>c;

   v.setColumn(c);
   v.setRow(r);

   v.getData();
   v.getCapacity();
   v.getRequiredValue();
   v.makeAllocation();

   v.display();
   return 0;
}
```

Output:

```
enter no of Rows : 3
enter no of columns : 4
enter source to destination data:
enter 0th row : 16
18
21
12
enter 1th row : 17
19
14
13
enter 2th row : 32
11
```

15
10
enter capacity for each source :
s1 : 150
s2 : 160
s3 : 90
enter required unit value for each destination :
d1 : 140
d2 : 120
d3 : 90
d4 : 50

given data :

	D1	D2	D3	D4	cap
S1	16	18	21	12	150
S2	17	19	14	13	160
S3	32	11	15	10	90
req	140	120	90	50	

after allocation :

16*100	18	21	12*50
17*40	19*30	14*90	13
32	11*90	15	10

(16 * 100)+(12 * 50)+(17 * 40)+(19 * 30)+(14 * 90)+(11 * 90)
answer : 5700
here 3+4-1 =6 no. of allocations
so this problem is non-degenarated solution.