Computer Applications in Operation Research

PROGRAM OF LINEARE PROGRAMMING PROBLEMS

```
/* SOLVE A LINEAR PROGRAMMING PROBLEM BY THE REGULAR
SIMPLEX METHOD */
 /*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^ */

#include <stdio.h>;
#include <conio.h>;
#include <math.h> ;
#include <float.h>;

#define MAX    10
#define YES  1
#define NO   0
#define M     100

void clear      (void)                  ;
void screen     (void)                  ;
void subscr     (void)                  ;
void simplex    (void)                          ;
void accept     (void)                  ;
void validity   (void)                  ;
void display    (void)                  ;
void check      (void)                  ;
void modify1    (void)                          ;
void modify2    (void)                          ;
void print      (void)                  ;
void standard   (void)                          ;
void draw       (void)                  ;
void fill      (void)                   ;
void check_opt  (void)                          ;
void solve      (void)                  ;
void value      (void)                  ;
void feas_check  (void)                         ;
void bound_check (void)                         ;
void algorithm   (void)                         ;

typedef int   array_int   [MAX]          ;
typedef char  array_char  [MAX]          ;
typedef float array_float [MAX]          ;
typedef float multi_array [MAX][MAX]     ;

array_float cof, rhs, tem, min, bas, sol, del      ;
array_int   rel, index, init                ;
```

```
array_char  res                                  ;
multi_array tab, bas_mat, slack, art                        ;
char        ch, ch1                                  ;
float       val, temp                                ;
int         n, c, r, i, j, k, l, v, a_v, tem_c_s     ;
int         end_flag, valid, res_flag, flag1, flag2  ;
int         c_x, c_a, c_s, prow, pcol, opt, page ,linc   ;

void main()
 {
 algorithm();
 ch = 'y';
 while ( (ch == 'y') || (ch == 'Y') )
  {
  screen();
  accept();
  display();
  simplex();
  }
 clrscr();
 }

void simplex()
 {
 standard();
 clear();
 screen();
 subscr();
 gotoxy(30,6);
 printf("INITIAL SIMPLEX TABLEAU ");
 draw();
 fill();
 check_opt();
 while ( opt != 1)
  {
  solve();
  if ( flag1 == 0 )
   {
   gotoxy(2,23);
   printf("  There exists an unbounded solution to the given problem.");
   goto exit;
   }
  else
   check_opt();
  if ( flag2 == 0 )
   {
```

```
    gotoxy(2,23);
    printf("  There exists an infeasible solution to the given problem.");
    goto exit;
    }
  }
 val = 0;
 for ( i = 0 ; i < n ; i++ )
  val = val + rhs[i]*bas[i];
 value();
 exit :
        gotoxy(2,24);
        clreol();
        gotoxy(79,24);
        printf("%c",186);
        gotoxy(2,24);
        printf("  Do you want to solve another problem ? [y/n]");
        ch = getch();
 }

void clear()
 {
 gotoxy(2,24);
 printf("  Press any key to continue .... ");
 getch();
 }

void screen()                    /* Draw the screen */
 {
 clrscr();
 gotoxy(1,1);
 printf("%c",201);
 for ( i = 2 ; i < 80 ; i++ )
  {
  gotoxy(i, 1);
  printf("%c",205);
  gotoxy(i, 5);
  printf("%c",205);
  gotoxy(i, 25);
  printf("%c",205);
  }
 gotoxy(79,1);
 printf("%c",187);
 gotoxy(1,3);
 printf("          TO SOLVE A L.P.P BY THE REGULAR SIMPLEX
METHOD\n");
```

```
 printf("
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n ");
 gotoxy(1,5);
 printf("%c",204);
 gotoxy(79,5);
 printf("%c",185);
 for ( i = 2 ; i < 25 ; i++ )
  {
  gotoxy(1,i);
  printf("%c",186);
  gotoxy(79,i);
  printf("%c",186);
  }
 gotoxy(1,25);
 printf("%c",200);
 gotoxy(79,25);
 printf("%c",188);
}

void subscr()
 {
 for ( i = 2 ; i < 79 ; i++ )
  {
  gotoxy(i,7);
  printf("%c",196);
  }
 }

void algorithm()
 {
 screen();
 gotoxy(3,6);
 printf("Basic steps of solving an L.P.P by the Regular Simplex method :-");
 gotoxy(3,7);

printf("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ");
 gotoxy(3,9);
 printf("Step 1: Standardization. ");
 gotoxy(3,10);
 printf("     (i) Convert the objective to a maximise function if not.");
 gotoxy(3,11);
 printf("     (ii) Check if all the rhs' are +ve, else make them.");
 gotoxy(3,12);
 printf("     (iii) Convert the constraints to equalities as follows");
 gotoxy(3,13);
```

```
    printf("          For a '=' constraint, add a artificial variable.");
    gotoxy(3,14);
    printf("          For a '<=' constraint, add a slack variable. ");
    gotoxy(3,15);
    printf("          For a '>=' constraint, subtract a surplus variable. ");
    gotoxy(3,16);
    printf("          and add an artificial variable.");
    gotoxy(3,17);
    printf("          The coefficients of the slack/surplus variables will be 0 and");
    gotoxy(3,18);
    printf("      those of the artificial variables will be -M, where M is a very");
    gotoxy(3,19);
    printf("      large value.");
    gotoxy(3,20);
    printf("Step 2: Construct the starting Simplex table.");
    gotoxy(3,21);
    printf("Step 3: Compute Net Evaluation Row by the formula,");
    gotoxy(3,22);
    printf("        delj = Zj - Cj = Cb.Xj - Cj");
    clear();
    screen();
    gotoxy(3,6);
    printf("(Algorithm continued....)");
    gotoxy(3,8);
    printf("Step 4: Optimality test.");
    gotoxy(3,9);
    printf("      (i) If all delj >= 0, the solution under test is optimal.");
    gotoxy(3,10);
    printf("          Alternate solutions exist if delj for any non-basic variables =0");
    gotoxy(3,11);
    printf("      (ii) If atleast 1 delj < 0, proceed to step 5.");
    gotoxy(3,12);
    printf("      (iii) If all elements of the pivotal column <= 0, Unbounded solution");
    gotoxy(3,13);
    printf("Step 5: Determination of incoming vector.");
    gotoxy(3,14);
    printf("        Vector corresponding to most -ve delj.");

    gotoxy(3,15);
    printf("      Determination of outgoing vector.");
    gotoxy(3,16);
    printf("        Vector corresponding to the minimum ratio for Xb/(pivotal
column)");
    gotoxy(3,17);
    printf("Step 6: Use row reductions to simplify the Simplex table.");
    gotoxy(3,18);
```

```
 printf("Step 7: Repeat steps 4 to 6 till optimal solution is obtained.");
 clear();
 }

void accept()                                   /* Accept all the */
 {                                            /* parameters of  */
 gotoxy(3,6);                                 /* the problem    */
 printf(" Is the linear programming problem a  ");
 gotoxy(5,8);
 printf(" A> Maximization problem  ");
 gotoxy(5,9);
 printf(" I> Minimization problem  ");
 gotoxy(3,11);
 printf(" (Press 'a' for a maximization problem ");
 gotoxy(3,12);
 printf("    and 'i' for a minimization one ) ");
 valid = NO;
 while ( valid != YES )
  {
  gotoxy(45, 12);
  ch1 = getche();
  if ( (ch1 == 'a') || (ch1 == 'i') || (ch1 == 'A') || (ch1 == 'I') )
   valid = YES;
  else
   {
   gotoxy(2,24);
   printf("  Wrong key !  Press the right one < 'a' or 'i' >");
   }
 }
screen();
valid = NO;
while ( valid != YES )
 {
 gotoxy(3,6);
 printf("  Number of variables used :  ");
 gotoxy(32,6);
 validity();
 scanf("%d",&v);
 a_v = v;
 if ( v > 0 )
  {
  for ( i = 2 ; i < 40 ; i++ )
      {
      gotoxy(i,24);
      printf(" ");
      }
```

```
  valid = YES;
  }

 else
 {
  gotoxy(2,24);
  printf("  Number of variables cannot be <= 0 ! ");
  }
 }
gotoxy(3,8);
printf("  Type 'y' against variables which are unrestricted ");
gotoxy(3,9);
printf("  and type 'n' against variables which are restricted. ");
k = 0;
for ( c = 0 ; c < v ; c++ )
 {
 if ( page == 1)
  k = 0;
 else
  k++;
 line = 11 + k;
 gotoxy(3, 11+k);
 printf("  Is variable x%d unrestricted ? [y/n] ",c+1);
 valid = NO;
 page = 0;
 while ( valid != YES )
  {
  gotoxy(45,11+k);
  res[c] = getche();
  if ( (res[c] == 'y') || (res[c] == 'Y') || (res[c] == 'n') || (res[c] == 'N'))
      {
      valid = YES;
      if ( (res[c] == 'y') || (res[c] == 'Y') )
       res_flag = YES;
      for ( j = 2 ; j < 70 ; j++ )
       {
       gotoxy(j,24);
       printf(" ");
       }
      }
  else
      {
      gotoxy(2,24);
      printf("  Wrong key !  Press the right one < 'y' or 'n' >");
      }
  if ( line >= 22 )
```

```
        {
        page = 1;
        gotoxy(3,24);
        printf("  Press any key for the next page ....");
        ch = getch();
        screen();
        gotoxy(3,8);
        printf("  Type 'y' against variables which are unrestricted ");
        gotoxy(3,9);
        printf("  and type 'n' against variables which are restricted. ");
        }
    }
  }

 for ( i = 0 ; i < MAX ; i++ )
  for ( j = 0 ; j < MAX ; j++ )
   {
   tab[i][j] = 0 ;
   slack[i][j] = 0;
   art[i][j] = 0;
   }
 screen();
 page = 0;
 gotoxy(3,6);
 printf("  Please enter the coefficients of the objective function : ");
 k = 0;
 for (c = 0 ; c < v ; c++ )
  {
  if ( page == 1 )
   {
   page = 0;
   k = 0;
   }
  else
   k++;
  gotoxy(3,8+k);
  line = 8+k;
  printf("  Coeff of x%d : ",c+1);
  validity();
  scanf("%f",&cof[c]);
  if ( line>= 21 )
   {
   page = 1;
   gotoxy(3,24);
   printf("  Press any key for the next page ....");
   ch = getch();
```

```
   screen();
   gotoxy(3,6);
   printf("  Please enter the coefficients of the objective function : ");
   }
  }
gotoxy(3, 22);
printf("  How many constraints ? ");
valid = NO;

 while ( valid != YES )
  {
  gotoxy(30, 22);
  validity();
  scanf("%d",&n);
  if ( n > 0 )
   {
   for ( i = 2 ; i < 40 ; i++ )
       {
       gotoxy(i,24);
       printf(" ");
       }
   valid = YES;
   }

  else
   {
   gotoxy(2,24);
   printf("  Number of constraints cannot be <= 0 ! ");
   }
  }
for ( k = 0 ; k < n ; k++ )
  {
  screen();
  gotoxy(3,6);
  printf("  Enter the coefficients of the variables of the %dth constraint : ",k+1);
  c = 0;
  for ( j = 0 ; j < v ; j++ )
   {
   if ( page == 1 )
       {
       c = 0;
       page = 0;
       }
   else
       c++;
   gotoxy(3, 8+c);
```

```
        line = 8+c;
        printf("  Coeff of x%d : ",j+1);
        validity();
        scanf("%f",&tab[k][j]);
        if ( line >= 21 )
            {
            page = 1;
            gotoxy(3,24);
            printf("  Press any key for the next page ....");
            ch = getch();
            screen();
            gotoxy(3,6);
            printf("  Please enter the coefficients of the objective function : ");
            }
      }
     if ( v > 5 )
      {
      screen();
      gotoxy(3, 8);
      }
     else
      gotoxy(3,10+c);
     printf("  Enter the RHS of the %dth constraint : ",k+1);
     validity();
     scanf("%f",&rhs[k]);
     if ( v > 5 )
      gotoxy(3,10);
     else
      gotoxy(3, 12+c);
     printf("  The relational operator used in the %dth constraint : ",k+1);
     if ( v > 5 )
      gotoxy(3,12);
     else
     gotoxy(3,13+c);
     printf("  ( If the relation is '>=', enter '0' ) ");
     if ( v > 5 )
      gotoxy(3,13);
     else

     gotoxy(3,14+c);
     printf("  ( If the relation is '<=', enter '1' ) ");
     if ( v > 5 )
      gotoxy(3,14);
     else
      gotoxy(3,15+c);
     printf("  ( If the relation is '=' , enter '2' ) ");
```

```
valid = NO;
while ( valid != YES )
 {
 if ( v > 5 )
     gotoxy(60,10);
 else
     gotoxy(60,12+c);
 validity();
 scanf("%d",&rel[k]);
 if ( (rel[k] >= 0) && (rel[k] <= 2) )
     {
     for ( j = 2 ; j < 75 ; j++ )
      {
      gotoxy(j,24);
      printf(" ");
      }
     valid = YES;
     }
  else
     {
     gotoxy(2,24);
     printf(" Wrong key ! Press the right key < '0' , '1' or '2' >");
     }
 }
 gotoxy(2,24);
 printf("  Press any key to continue ....");
 ch = getch();
 }
}

void validity()                    /* Checks if <esc>, <enter> */
 {                              /* <tab>,<backspace> have   */
 int key;                         /* beenn pressed          */

 do
  {
  key = bioskey(1);

  if ( (key == 283) || (key == 7181) || (key == 19712) || (key == 19200) )
   {
       key = bioskey(0);
   key = 0;
   }
  }
  while (key == 0);
 }
```

```
void display()                  /* Display the L.P.P */
 {
 screen();
 gotoxy(3,6);
 printf("  The given linear programming problem is...");
 if ( (ch1 == 'a') || (ch1 == 'A') )
  {
  gotoxy(3,8);
  printf("  MAXIMIZE  P = ");
  print();
  }
 else
 if ( (ch1 == 'i') || (ch1 == 'I') )
  {
  gotoxy(3,8);
  printf("  MINIMIZE  P = ");
  print();
  }
 gotoxy(2,24);
 printf("  Press any key to continue ....");
 ch = getch();
 }

void print()                    /* Print the L.P.P */
 {
 c = 5;
 j = 0;
 k = -1;
 r = 9;
 while ( j < v )
  {
  if ( c >= 60 )
   {
   r++;
   k = 0;
   }
  else
   k++;
  c = 20 + (10*k);
  gotoxy(c,r);
  printf(" %g",cof[j]);
  printf(".x%d ",j+1);
  if ( j != (v-1) )
    printf(" + ");
  j++;
```

```
 }
r += 3;
gotoxy(3,r);
printf("  subject to the constraints ... ");
for ( k = 0 ; k < n ; k++ )
 {
 r++;
 j = 0;
 l = -1;
 c = 5;

 while ( j < v )
  {
  if ( c >= 60 )
       {
       r++;
       l = 0;
       }
  else
       l++;
  c = 5 + (10*l);
  gotoxy(c,r);
  printf(" %g",tab[k][j]);
  printf(".x%d ",j+1);
  if ( j != (v-1) )
       printf(" + ");
  j++;
  }
if ( rel[k] == 0 )
  {
  gotoxy(7+c,r);
  printf(" >= ");
  }
else
if ( rel[k] == 1 )
  {
  gotoxy(7+c,r);
  printf(" <= ");
  }
else
if ( rel[k] == 2 )
  {
  gotoxy(7+c,r);
  printf(" = ");
  }
gotoxy(10+c,r);
```

```
    printf(" %g",rhs[k]);
    }
  line = r;
  if ( (line>= 15 ) && ( v > 2 ) )
   {
   gotoxy(2,24);
   printf("Press any key for next page ....");
   getch();
   screen();
   r = 8;
   gotoxy(3, r);
   }
  else
   {
   r += 2;
   gotoxy(3,r);
   }
  printf("   where");

  if ( res_flag == YES )
   {
   for ( i = 0 ; i < v ; i++ )
    {
    r++;
    if ( res[i] == 'n' )
        {
        gotoxy(3, r);
        printf("    x%d >= 0",i+1);
        }
    else
    if ( res[i] == 'y' )
        {
        gotoxy(3, r);
        printf("    x%d is unrestricted",i+1);
        }
    }
   }
 else
 if ( res_flag == NO )
  {
  for ( i = 0 ; i < v ; i++ )
   {
   r++;
   gotoxy(3, r);
   printf("    x%d >= 0",i+1);
   }
```

```
    }
  }

void standard()
  {
  valid = NO;                              /* If any of the RHS' is */
  for ( i = 0 ; i < n ; i++ )              /* negative , multiply the */
    {                                      /* entire constraint by -1 */
    if ( rhs[i] < 0 )
      {
      valid = YES;
      for ( j = 0 ; j < v ; j++ )
          tab[i][j] = -tab[i][j];
      rhs[i] = -rhs[i];
      if ( rel[i] == 0 )
          rel[i] = 1;
      else
      if ( rel[i] == 1 )
          rel[i] = 0;
      }
    }
  for ( i = 0 ; i < a_v ; i++ )            /* If any of the variables */
    {                                      /* is unrestricted in sign */
    if ( res[i] == 'y' )                   /* reform the constraints */
      {                                    /* accordingly */
      cof[v] = -cof[i];
      for ( j = 0 ; j < n ; j++ )
          tab[j][v] = -tab[j][i];
      v++;
      }
    }

c_x = v - a_v;
if ( ( ch1 == 'i' ) || ( ch1 == 'I' ) )    /* If the objective is to */
  for ( i = 0 ; i < v ; i++ )              /* minimise, convert the */
    cof[i] = - cof[i];                     /* objective function to */
                                           /* a maximisation problem */
tem_c_s = 0;
for ( i = 0 ; i < n ; i++ )
  if ( (rel[i] == 0) || (rel[i] == 2) )
    tem_c_s++;
screen();
c_a = 0 ;
c_s = 0 ;
for ( i = 0 ; i < n ; i++ )
  {
```

```
 if ( rel[i] == 0 )
  {
  slack[i][c_s] = -1;
  c_s++;
  art[i][c_a] = 1;
  bas[i] = -M;
  index[i] = v+tem_c_s+c_a;
  c_a++;
  }
 else
 if ( rel[i] == 1 )
  {
  slack[i][c_s] = 1;
  bas[i] = 0;
  index[i] = v+c_s;
  c_s++;
  }
 else
 if ( rel[i] == 2 )
  {
  art[i][c_a] = 1;
  bas[i] = -M;
  index[i] = v+tem_c_s+c_a;
  c_a++;
  }
 }
gotoxy(3,6);
printf(" The given L.P. problem is reduced to the standard form ....");
gotoxy(3,8);
printf(" MAXIMIZE P = ");

c = 20;
j = 0;
k = -1;
r = 9;
while ( j < v )
 {
 if ( c >= 60 )
  {
  r++;
  k = 0;
  }

 else
  k++;
 c = 20 + (10*k);
```

```
 gotoxy(c,r);
 printf(" %g",cof[j]);
 printf(".x%d ",j+1);
 if ( (c_s != 0) || (c_a != 0) )
   printf(" + ");
 j++;
 }
j = 0;
while ( j < c_s )      //this loop is to print standard objective fn/..
 {
 cof[v+j] = 0;
 if ( c >= 60 )
  {
  r++;
  k = 0;
  c = 20;
  }
 else
  k++;
 c = 20 + (10*k);
 gotoxy(c,r);
 printf(" %g",cof[v+j]);          //for slack variables
 printf(".s%d ",j+1);
 if ( j != (c_s-1) || (c_a != 0 ) )
   printf(" + ");
 j++;
 }

j = 0;
while ( j < c_a )
 {
 cof[v+c_s+j] = -M;
 if ( c >= 60 )
  {
  r++;
  k = 0;
  c = 20;
  }
 else
  k++;
 c = 20 + (10*k);
 gotoxy(c,r);
 printf(" %g",cof[v+c_s+j]);
 printf(".a%d ",j+1);
 if ( j != (c_a-1) )
   printf(" + ");
```

```
    j++;
    }

  for ( i = 0 ; i < n ; i++ )
   {
   k = 0;
   for ( j = v ; j < v+c_s ; j++ )
    {
    tab[i][j] = slack[i][k];
    k++;
    }
   }
  for ( i = 0 ; i < n ; i++ )
   {
   k = 0;
   for ( j = v+c_s ; j < v+c_s+c_a ; j++ )
    {
    tab[i][j] = art[i][k];
    k++;
    }
   }
 r += 2;
 gotoxy(3,r);
 printf("   subject to the constraints ....");
 k = 0;
 r += 2;
 for ( i = 0 ; i < n ; i++ )
  {
  r++;
  j = 0;
  k = -1;
  while ( j < v )
   {
   if ( c >= 60 )
       {
       r++;
       k = 0;
       }
   else
       k++;
   c = 5 + (10*k);
   gotoxy(c,r);
   printf(" %g",tab[i][j]);
   printf(".x%d ",j+1);
   printf(" + ");
   j++;
```

```
    }
  j = 0;
  while ( j < c_s )
   {
   if ( c >= 60 )
        {
        r++;
        k = 0;
        }
   else
        k++;
   c = 5 + (10*k);

   gotoxy(c,r);
   printf(" %g",tab[i][v+j]);
   printf(".s%d ",j+1);
   if ( j != c_s)
        printf(" + ");
   j++;
   }
  j = 0;
  while ( j < c_a )
   {
   if ( c >= 60 )
        {
        r++;
        k = 0;
        }
   else
        k++;
   c = 5 + (10*k);
   gotoxy(c,r);
   printf(" %g",tab[i][v+c_s+j]);
   printf(".a%d ",j+1);
   if ( j != c_a-1)
        printf(" + ");
   j++;
   }
  c = c + 5;
  gotoxy(c,r);
  printf(" = ");
  c = c + 2;
  gotoxy(c,r);
  printf(" %g",rhs[i]);
  }
line = r;
```

```
if ( (line >= 15 ) && ( v > 2 ) )
 {
 gotoxy(2,24);
 printf("Press any key for next page ....");
 getch();
 screen();
 r = 8;
 gotoxy(3, r);
 }
else
 {
 r += 3;
 gotoxy(3,r);
 }
printf("  where ");
for ( i = 1 ; i <= v ; i++ )
 {
 r++;
 gotoxy(3,r);
 printf("  x%d >= 0",i);
 }
}

void draw()                          /* To draw the Simplex table */
 {
 gotoxy(15,9);
 printf("C[j]");
 for ( i = 5 ; i < 75 ; i++ )
  {
  gotoxy(i,10);
  printf("%c",196);
  }
 for ( i = 11 ; i < 13 + (2*n) ; i++ )
  {
  gotoxy(5,i);
  printf("%c",179);
  }
 gotoxy(5,10);
 printf("%c",218);
 for ( i = 11 ; i < 13 + (2*n) ; i++ )
  {
  gotoxy(10,i);
  printf("%c",179);
  }
 gotoxy(10,10);
 printf("%c",194);
```

```
for ( i = 11 ; i < 13 + (2*n) ; i++ )
 {
 gotoxy(15,i);
 printf("%c",179);
 }
gotoxy(15,10);
printf("%c",194);
for ( i = 11 ; i < 13 + (2*n) ; i++ )
 {
 gotoxy(20,i);
 printf("%c",179);
 }
gotoxy(20,10);
printf("%c",194);
for ( i = 11 ; i < 13 + (2*n) ; i++ )
 {
 gotoxy(75,i);
 printf("%c",179);
 }
gotoxy(75,10);
printf("%c",191);
for ( i = 11 ; i < 13 + (2*n) ; i++ )
 {
 gotoxy(70,i);
 printf("%c",179);
 }
gotoxy(70,10);
printf("%c",194);
r = 13 + (2*n);
for ( i = 5 ; i < 75 ; i++ )
 {
 gotoxy(i,r);
 printf("%c",196);
 }

r = 13 + (2*n);
gotoxy(5,r);
printf("%c",192);
gotoxy(10,r);
printf("%c",193);
gotoxy(15,r);
printf("%c",193);
gotoxy(20,r);
printf("%c",193);
gotoxy(75,r);
printf("%c",217);
```

```
 gotoxy(70,r);
 printf("%c",193);
 gotoxy(20,10);
 printf("%c",197);
 gotoxy(70,10);
 printf("%c",197);
 gotoxy(20,r);
 printf("%c",197);
 gotoxy(70,r);
 printf("%c",197);
 gotoxy(16,r+1);
 printf("%c[j]",30);
 for ( i = 5 ; i < 75 ; i++ )
  {
  gotoxy(i,12);
  printf("%c",196);
  }
 gotoxy(5,12);
 printf("%c",195);
 gotoxy(10,12);
 printf("%c",197);
 gotoxy(15,12);
 printf("%c",197);
 gotoxy(20,12);
 printf("%c",197);
 gotoxy(75,12);
 printf("%c",180);
 gotoxy(70,12);
 printf("%c",197);
 gotoxy(6,11);
 printf("C[b]");
 gotoxy(12,11);
 printf("BAS");
 gotoxy(17,11);
 printf("RHS");
 gotoxy(72,11);
 printf("MIN");
 for ( i = 20 ; i < 70 ; i++ )
  {
  gotoxy(i,8);
  printf("%c",196);
  }
r = 15 + (2*n);

for ( i = 20 ; i < 70 ; i++ )
 {
```

```
    gotoxy(i,r);
    printf("%c",196);
    }
  gotoxy(20,9);
  printf("%c",179);
  gotoxy(70,9);
  printf("%c",179);
  gotoxy(20,r-1);
  printf("%c",179);
  gotoxy(70,r-1);
  printf("%c",179);
  gotoxy(20,8);
  printf("%c",218);
  gotoxy(70,8);
  printf("%c",191);
  gotoxy(20,r);
  printf("%c",192);
  gotoxy(70,r);
  printf("%c",217);
  for ( i = 5 ; i < 15 ; i++ )
    {
    gotoxy(i,r);
    printf("%c",196);
    }
  gotoxy(5,r-1);
  printf("%c",179);
  gotoxy(15,r-1);
  printf("%c",179);
  gotoxy(5,r-2);
  printf("%c",195);
  gotoxy(15,r-2);
  printf("%c",197);
  gotoxy(5,r);
  printf("%c",192);
  gotoxy(15,r);
  printf("%c",217);
  }

void calc_del()                    /* To compute the del[j] row */
  {
  for ( i = 0 ; i < v+c_s+c_a ; i++ )
    {
    temp = 0;
    for ( j = 0 ; j < n ; j++ )
      temp = temp + tab[j][i] * bas[j];
    del[i] = temp - cof[i] ;
```

```
  }
 }

void fill()                              /* To fill all the elements */
 {                                /* in the Simplex table    */
 calc_del();
 r = 13 + (2*n);
 k = -1;
 c = 23;
 for ( i = 0 ; i < v ; i++ )
  {
  gotoxy(c,9);
  printf("%.1f ",cof[i]);
  gotoxy(c,11);
  printf("x%d",i+1);
  gotoxy(c,r+1);
  printf("%.1f",del[i]);
  c = c + 7;
  }
 for ( i = 0 ; i < c_s ; i++ )
  {
  gotoxy(c,9);
  printf("%.1f ",cof[v+i]);
  gotoxy(c,11);
  printf("s%d",i+1);
  gotoxy(c,r+1);
  printf("%.1f",del[v+i]);
  c = c + 7;
  }
 for ( i = 0 ; i < c_a ; i++ )
  {
  gotoxy(c,9);
  printf("-M ",cof[v+c_s+i]);
  gotoxy(c,11);
  printf("a%d",i+1);
  gotoxy(c,r+1);
  printf("%.1f",del[v+c_s+i]);
  c = c + 7;
  }
 for ( i = 0 ; i < n ; i++ )
  {
  r = 13 + (2*i);
  if ( bas[i] == -M )
   {
   gotoxy(6,r);
   printf("-M");
```

```
    }
   else
    {
    gotoxy(6,r);
    printf("%.1f",bas[i]);
    }
   if ( index[i] < v )
    {
    gotoxy(11,r);
    printf(" x%d",index[i]+1);
    }

   else
   if ( (index[i] >= v) && (index[i] < v+c_s) )
    {
    gotoxy(11,r);
    printf(" s%d",(index[i] - v + 1));
    }
   else
   if ( (index[i] >= v+c_s) && (index[i] < v+c_s+c_a) )
    {
    gotoxy(11,r);
    printf(" a%d",(index[i] - (v+c_s) + 1));
    }
   gotoxy(16,r);
   printf("%.1f",rhs[i]);
   for ( k = 0 ; k < v+c_s+c_a ; k++ )
    {
    c = 24 + (7*k);
    gotoxy(c,r);
    printf("%.1f",tab[i][k]);
    }
   gotoxy(71,r);
   printf("%.1f",min[i]);
   }
  gotoxy(6,r+3);
  val = 0;
  for ( i = 0 ; i < n ; i++ )
     val = val + rhs[i]*bas[i];
  printf("Z = %.1f",val);
  }

void solve()                    /* Actual Simplex procedure */
  {
  temp = 0;
  for ( j = 0 ; j < v+c_s+c_a ; j++ )
```

```
 {
 if ( del[j] < temp )
  {
  pcol = j ;
  temp = del[pcol];
  }
 }
bound_check();
if ( flag1 == 1 )
 {
 for ( j = 0 ; j < n ; j++ )
  if ( tab[j][pcol] != 0 )
      min[j] = rhs[j] / tab[j][pcol];
  else
      min[j] = M;
 temp = M;
 for ( i = 0 ; i < n ; i++ )
  {
  if ( (min[i] >= 0) && (tab[i][pcol] > 0) )
      {
      if (min[i] < temp)
       {
       prow = i ;
       temp = min[i];
       }

      else
      if (min[i] == temp)
       {
       if ((i > prow) && (c_a > 0) )
          prow = i;
       else
       if ((i > prow) && (bas[prow] < M))
        prow = i;
       }
      }
  }
 bas[prow] = cof[pcol];
 index[prow] = pcol;
 i = 0;
 temp = tab[prow][pcol];
 while ( i < v+c_s+c_a )
  {
  tab[prow][i] = tab[prow][i]/temp;
  i++ ;
  }
```

```
rhs[prow] = min[prow];
for ( i = 0 ; i < n ; i++ )
 if ( i != prow          )
     {
     temp = tab[i][pcol];
     if ( ( tab[i][pcol] - tab[prow][pcol]*temp ) == 0 )
       {
       j = 0;
       while ( j < v+c_s+c_a )
        {
        tab[i][j] = tab[i][j] - tab[prow][j]*temp;
        j++;
        }
       rhs[i] = rhs[i] - rhs[prow]*temp;
       }
     else
     if ( ( tab[i][pcol] + tab[prow][pcol]*temp ) == 0 )
       {
       j = 0;
       while ( j < n+c_s+c_a )
        {
        tab[i][j] = tab[i][j] + tab[prow][j]*temp;
        j++;
        }
       rhs[i] = rhs[i] + rhs[prow]*temp;
       }
     }
feas_check();
if ( flag2 == 1 )
 {
 for ( j = 0 ; j < v+c_s+c_a ; j++ )
     {
     temp = 0;
     for ( i = 0 ; i < n ; i++ )
      temp = temp + tab[i][j] * bas[i];
     del[j] = temp - cof[j] ;
     }

 val = 0;
 for ( i = 0 ; i < n ; i++ )
     val = val + rhs[i]*bas[i];
 getch();
 screen();
 subscr();
 gotoxy(30,6);
 printf("SIMPLEX TABLEAU ");
```

```
    draw();
    fill();
    }
   }
  }

void check_opt()                    /* Check for optimality */
 {
 opt = 1;
 for ( i = 0 ; i < v+c_s+c_a ; i++ )
  if ( del[i] < 0 )
    opt = 0 ;
 }

void value()                        /* Print the solution */
 {
 gotoxy(3,6);
 printf("              OPTIMAL SIMPLEX TABLEAU ");
 gotoxy(2,24);
 clrcol();
 gotoxy(79,24);
 printf("%c",186);
 gotoxy(2,24);
 printf(" Press any key to continue ....");
 getch();
 screen();
 gotoxy(3,6);
 printf(" Solution :-");
 gotoxy(3,7);
 printf(" ===========");
 gotoxy(3,8);
 if ( val >= M/2 )
  printf(" There exists a non-existing ( pseudo_optimum ) feasible solution ");
 gotoxy(3,9);
 printf(" The optimal solution to the given problem is given by : ");
 if ( (ch1 == 'a') || (ch1 == 'A') )
  {
  gotoxy(3,10);
  printf(" Max Z = %.2f",val);
  }
 else
 if ( (ch1 == 'i') || (ch1 == 'I') )     /* If the original problem */
  {                                  /* was a minimisation one  */
  val = -val;                           /* negate the value obtained */
  gotoxy(3,10);
  printf(" Min Z = %.2f",val);
```

```
    }

  gotoxy(3,12);
  printf(" when");
  for ( i = 0 ; i < v ; i++ )
   {
   res_flag = NO;                    /* In case there are no    */
   for ( j = 0 ; j < n ; j++ )       /* unrestricted variables  */
    {                              /* print the values of the */
    if ( index[j] == i)              /* as it is                */
        {
        k = j;
        res_flag = 1;
        }
    }
   if (res_flag)
    {                              /* In case all or few      */
    gotoxy(3,14+i);                  /* variables are unrestric- */
    printf(" x%d = %.2f ",i+1,rhs[k]);   /* evaluate them accordingly */
    sol[i] = rhs[k];
    }
   else
    {
    gotoxy(3,14+i);
    printf(" x%d = 0",i+1);
    sol[i] = 0;
    }
  }
 if ( c_x > 0 )
  {
  r = 15 + j;
  gotoxy(3,r+1);
  printf(" where");
  r++;
  for ( i = 0 ; i < a_v ; i++ )
   {
   if ( res[i] == 'y' )
       {
       r = r + 1 + i;
       gotoxy(3,r+i);
       printf(" x%d = %.2g ",i+1,sol[i]-sol[v-i]);
       }
   }
  }
}
```

```
void bound_check()                    /* Checks for an unbounded */
 {                                  /* solution. If all the    */
 flag1 = 0;                            /* elements of the pivotal */
 for ( i = 0 ; i < n ; i++ )              /* column are negative     */
  if ( tab[i][pcol] > 0 )                 /* then the solution to    */
    flag1 = 1;                         /* the given problem is    */
 }                                  /* unbounded               */

void feas_check()                        /* If at any instance any  */
 {                                  /* of the RHS' is/are      */
 flag2 = 1;                            /* negative, then there    */
 for ( i = 0 ; i < n ; i++ )              /* exists an infeasible    */
  if ( rhs[i] < 0 )                    /* solution to the problem */
    flag2 = 0;
 }
```

OUTPUT

```
-----------------------------------------------------------------------+

         TO SOLVE A L.P.P BY THE REGULAR SIMPLEX METHOD
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

-----------------------------------------------------------------------|
 Basic steps of solving an L.P.P by the Regular Simplex method :-
 -------------------------------------------------------------

Step 1: Standardization.
     (i) Convert the objective to a maximise function if not.
     (ii) Check if all the rhs' are +ve, else make them.
     (iii) Convert the constraints to equalities as follows
          For a '=' constraint, add a artificial variable.
          For a '<=' constraint, add a slack variable.
          For a '>=' constraint, subtract a surplus variable.
          and add an artificial variable.
          The coefficients of the slack/surplus variables will be 0 and
     those of the artificial variables will be -M, where M is a very
     large value.
Step 2: Construct the starting Simplex table.
Step 3: Compute Net Evaluation Row by the formula,
      delj = Zj - Cj = Cb.Xj - Cj

 Press any key to continue ....
--------------------------------------------------------------------------------+

        TO SOLVE A L.P.P BY THE REGULAR SIMPLEX METHOD
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

-------------------------------------------------------------------------|
(Algorithm continued....)

Step 4: Optimality test.
     (i) If all delj >= 0, the solution under test is optimal.
        Alternate solutions exist if delj for any non-basic variables =0
     (ii) If atleast 1 delj < 0, proceed to step 5.
     (iii) If all elements of the pivotal column <= 0, Unbounded solution
Step 5: Determination of incoming vector.
      Vector corresponding to most -ve delj.
   Determination of outgoing vector.
      Vector corresponding to the minimum ratio for Xb/(pivotal column)
Step 6: Use row reductions to simplify the Simplex table.
Step 7: Repeat steps 4 to 6 till optimal solution is obtained.
```

```
Press any key to continue ....
-------------------------------------------------------------------------+-------------------------
-------------------------------------------------------------------------

        TO SOLVE A L.P.P BY THE REGULAR SIMPLEX METHOD
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

-------------------------------------------------------------------------
 Is the linear programming problem a

  A> Maximization problem
  I> Minimization problem

 (Press 'a' for a maximization problem
   and 'i' for a minimization one )


        TO SOLVE A L.P.P BY THE REGULAR SIMPLEX METHOD
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

-------------------------------------------------------------------------
 Number of variables used : 3


     Please enter the coefficients of the objective function :


 Coeff of x1 : 2
 Coeff of x2 : 3
 Coeff of x3 : 8


How many constraints ?  3
Enter the coefficients of the variables of the 1th constraint :


 Coeff of x1 : 6
 Coeff of x2 : 8
 Coeff of x3 : 4

Enter the RHS of the 1th constraint : 14
The relational operator used in the 1th constraint :  1
( If the relation is '>=', enter '0' )
( If the relation is '<=', enter '1' )
( If the relation is '=' , enter '2' )

Press any key to continue ....
```

```
Enter the coefficients of the variables of the 2th constraint :


Coeff of x1 : 4
Coeff of x2 : 7
Coeff of x3 : 3

Enter the RHS of the 2th constraint : 30

The relational operator used in the 2th constraint :  1
( If the relation is '>=', enter '0' )
( If the relation is '<=', enter '1' )
( If the relation is '=' , enter '2' )

Press any key to continue ....
Enter the coefficients of the variables of the 3th constraint :


Coeff of x1 : 3
Coeff of x2 : 6
Coeff of x3 : 8

Enter the RHS of the 3th constraint : 18

The relational operator used in the 3th constraint :  1
( If the relation is '>=', enter '0' )
( If the relation is '<=', enter '1' )
( If the relation is '=' , enter '2' )
Press any key to continue ....
The given linear programming problem is...

MAXIMIZE  P =
        2.x1  +  3.x2  +  8.x3


subject to the constraints ...
 6.x1  +  8.x2  +  4.x3  <= 14
 4.x1  +  7.x2  +  3.x3  <= 30
 3.x1  +  6.x2  +  8.x3  <= 18

where
 x1 >= 0
 x2 >= 0
 x3 >= 0
```

The given L.P. problem is reduced to the standard form ....

MAXIMIZE P =
2.x1 + 3.x2 + 8.x3 + 0.s1 + 0.s2 +
0.s3

subject to the constraints ....

6.x1 + 8.x2 + 4.x3 + 1.s1 + 0.s2 + 0.s3 = 14

4.x1 + 7.x2 + 3.x3 + 0.s1 + 1.s2 + 0.s3 = 30

3.x1 + 6.x2 + 8.x3 + 0.s1 + 0.s2 + 1.s3 = 18

Press any key for next page ....
where
x1 >= 0
x2 >= 0
x3 >= 0

INITIAL SIMPLEX TABLEAU

C[j]		2.0	3.0	8.0	0.0	0.0	0.0	
C[b]	BAS	RHS	x1	x2	x3	s1	s2	s3	MIN
0.0	s1	14.0	6.0	8.0	4.0	1.0	0.0	0.0	0.0
0.0	s2	30.0	4.0	7.0	3.0	0.0	1.0	0.0	0.0
0.0	s3	18.0	3.0	6.0	8.0	0.0	0.0	1.0	0.0
Z = 0.0		-[j]	-2.0	-3.0	-8.0	0.0	0.0	0.0	

OPTIMAL SIMPLEX TABLEAU

```
---------------------------------------------------------------------
          +-----------------------------------------------+
        C[j] | 2.0   3.0   8.0   0.0   0.0   0.0
+-----------+-----------------------------------------------+----+
|C[b]| BAS| RHS| x1    x2    x3    s1    s2    s3          MIN|
+----+----+----+-----------------------------------------------+----|
|0.0   | s1  5.0|  4.5   5.0   0.0   1.0   0.0   -0.5      |3.5 |
|
|0.0   | s2  23.2  2.9   4.8   0.0   0.0   1.0   -0.4      |10.0|
|
|8.0   | x3   2.2   0.4   0.8   1.0   0.0   0.0   0.1      |2.2 |
|  ------------------------------
|Z = 18.0 |-[j]| 1.0   3.0   0.0   0.0   0.0   1.0        |
+---------+   +-----------------------------------------------+
```

Press any key to continue ....

Solution :-

The optimal solution to the given problem is given by :
Max Z = 18.00

when

x1 = 0
x2 = 0
x3 = 2.25

Do you want to solve another problem ? [y/n]

SPECIAL NOTE ABOUT THE LINEAR PROGRAMMING PROBLEMS

This Program is free from all the errors. and can be easily execute in the computer.

We are using modular approach in this program.

This program can accept maximum 10 coefficients of objective functions and constraint equations.

This program is applicable for both the minimization or maximization problems.

This program contains simplex method, big M method but not graphical solution. In the next edition of this book, we will also provide graphical solution with linear programming solutions.

To understand this program we have given output (from example).

Your suggestions or comments is very necessary for modifying this Program.

—AUTHOR

TRANSPORTATION PROBLEMS IN C

Transportation problems can be easily solve by many methods which are given in the transportation chapter of our book.

Generally we find the transportation cost by using vogel's approximation method and we check the optimality by Modi method.

So we have made two C Programs ont he basis of VAM Method and MODI Method.

Here we are giving two programs of transportation problem

1) VAM Program

2) OPTIMAL Program

In optimal program, it is very difficult to check the optimality for all transportation problems. Because loop forms many times until the solution is optimal.

So, it is very difficult to construct the loop in C Program but we have tried to make better program for find the optimal solution.

In optimal program, we are using manual operator approach for making the loops. We choose the operator @ for value θ.

User choose the minimum value θ from table. And then we goto for N allocations for loop.

For N allocations, user should decide the co-ordinates for making the loop. Here we can't indicate arrow mark in loop. But we can understand loop. (help the output example)

Table is given below for choosing the coordinates according to situations of problem this coordinate is based on array system. (C lang).

(0, 0)	(9, 1)	(0, 2)	(0, 3)	.....	.....
(1, 0)	(1, 1)	(1, 2)	(1, 3)	.....	.....
(2, 0)	(2, 1)	(2, 2)	(2, 3)	.....	.....
(3, 0)	(3, 1)	(3, 2)	(3, 3)	.....	.....
.....	.....	.....	.....	.....	.....

For example if you want to enter 1st coordinate you should type the value
1
1
then you go to second cordinate. (please help the output example).

And at the last..... your suggestions is very necessary for modifying these programs.

VAM PROGRAM IN C

VAM PROGRAM IN C

```
/* SOLVE AN TRANSPORTATION PROBLEM USING VOGEL'S
APPROXIMATION METHOD */
/*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^ */
//#include<iostream.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <float.h>
#define MAX 10
#define YES 1
#define NO  0
#define M   10000
typedef int array[MAX][MAX];
typedef int list[MAX];
void problem   (void);
void algorithm  (void);
void trans1();
void proc();
void validity   (void);
void screen     (void);
void draw       (void);
void outline    (void);
void accept     (void);
void display    (array);
void initfeas   (void);
void accept_cl  (void);
void modify     (void);
void fill       (array);
void estimate   (void);
void check      (void);
void clear      (void);
void assign     (void);

char  ch,ch1;
int tcost=0;
int   i, j, k, l, m, row, col, taken, r_count, c_count,nos,nod;
int   min, min1, min2, max, total_cost;
int dcost[10][10],basic[10][10],sup1[10],dem1[10];
int   valid, flag, taken_flag, flg, mod_flag, chk_flag;
int   mark, fact, t_mark, t_fact, total_supply, total_demand;
array cost, asgn, temp;
list  supply, demand, t_supply, t_demand, r_pen, c_pen, r_index, c_index;
```

```
main()
 {
problem();
 algorithm();
 ch = 'y';
 while ( (ch == 'y') || (ch == 'Y') )
  {
  screen();
  accept();
  display(cost);
  check();
  if ( chk_flag == 1 )
  display(cost);
  modify();
  if ( mod_flag )
  display(cost);
  clrscr();
  trans1();
  proc();
  clreol();
  gotoxy(2,24);
  printf("Do you want to continue with another problem ? [y/n] ");
  ch = getche();
  };
  clrscr();
 return;
 }

void screen()
 {
 clrscr();
 gotoxy(1,1);
 printf("%c",201);
 for ( i = 2 ; i < 80 ; i++ )
  {
  gotoxy(i, 1);
  printf("%c",205);
  gotoxy(i, 5);
  printf("%c",205);
  gotoxy(i, 25);
  printf("%c",205);
 }
gotoxy(79,1);
printf("%c",187);
gotoxy(1,3);
```

```
 printf("      TO FIND THE INITIAL FEASIBLE SOLUTION OF A
TRANSPORTATION PROBLEM ");
 gotoxy(1,4);
 printf("                USING VOGEL'S APPROXIMATION METHOD\n");
 gotoxy(1,5);
 printf("%c",204);
 gotoxy(79,5);
 printf("%c",185);
 for ( i = 2 ; i < 25 ; i++ )
  {
  gotoxy(1,i);
  printf("%c",186);
  gotoxy(79,i);
  printf("%c",186);
  }
 gotoxy(1,25);
 printf("%c",200);
 gotoxy(79,25);
 printf("%c",188);
 }

void clear()
 {
 gotoxy(2,24);
 printf("Press any key to continue....");
 getch();
 }

void problem()
 {
 screen();
 gotoxy(3,6);
 printf("Problem Definition :-");
 gotoxy(3,7);
 printf("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ");
 gotoxy(3,8);
 printf("   In the assignment problem, the objective is to transport goods from
more");
 gotoxy(3,9);
 printf("than one centre, called 'origins', to more than one places, called ");
 gotoxy(3,10);
 printf("'destinations', in such manner that the cost of transportation is
minimum.");
 gotoxy(3,11);
 printf("   The availability of the various origins and the requirements of");
 gotoxy(3,12);
```

```
printf("destinations are finite and constitute of limited resources.");
gotoxy(3,14);
printf("Mathematically, the transportation problem can be stated as :");
gotoxy(3,16);
printf("    Let there be 'm' origins, ith origin possessing 'Ai' units of a certain");
gotoxy(3,17);
printf("product and 'n' destinations, jth destination requiring 'Bj' units of the ");
gotoxy(3,18);
printf("product.");
gotoxy(3,19);
printf("    Let 'Cij' be the cost of shipping 1 unit from the ith origin to the jth");
gotoxy(3,20);
printf("destination.");
gotoxy(3,21);
printf("    Let 'Xij' be the amount to be shipped from the ith origin to the jth");
gotoxy(3,22);
printf("destination.");
clear();
screen();
gotoxy(3,6);
printf("Minimise the total cost");
gotoxy(3,7);
printf("    Z = ä ä Cij.Xij , for i = 1,2....n, j = 1,2....n ");
gotoxy(3,9);
printf("subject to the restrictions of the form :");
gotoxy(3,11);
printf("Availability constraint :");
gotoxy(3,12);
printf("    ä Xij = Ai for i = 1,2....m, j = 1,2....n");
gotoxy(3,14);
printf("Requirement constraint  :");
gotoxy(3,15);

printf("    ä Xij = Bj for i = 1,2....m, j = 1,2....n");
gotoxy(3,17);
printf("Total supply = Total demand (i.e.)");
gotoxy(3,18);
printf("    ä Ai = ä Bj for i = i,2....m, j = 1,2....n");
clear();
screen();
gotoxy(3,6);
printf("Tabular ");
gotoxy(3,7);
printf("  Representation :");
mark = 5;
fact = 5;
```

```
outline();
row = 11 + fact*2;
gotoxy(24,row);
printf("%c",218);
for ( i = 1 ; i <= mark*5 ; i++ )
 {
 gotoxy(24+i,row);
 printf("%c",196);
 gotoxy(24+i,row+2);
 printf("%c",196);
 }
gotoxy(24+mark*5,row); printf("%c",191);
gotoxy(24,row+1); printf("%c",179);
gotoxy(24+mark*5,row+1); printf("%c",179);
gotoxy(24,row+2); printf("%c",192);
gotoxy(24+mark*5,row+2); printf("%c",217);
col = 29 + mark*5;
gotoxy(col,9);
printf("%c",218);
for ( i = 1 ; i <= fact*2 ; i++ )
 {
 gotoxy(col,9+i);
 printf("%c",179);
 gotoxy(col+5,9+i);
 printf("%c",179);
 }
for ( i = 1 ; i < 5 ; i++ )
 {
 gotoxy(col+i,9);
 printf("%c",196);
 row = 9 + fact*2;
 gotoxy(col+i,row);
 printf("%c",196);
 }
gotoxy(34+mark*5,9); printf("%c",191);
gotoxy(34+mark*5,row); printf("%c",217);
gotoxy(29+mark*5,row); printf("%c",192);
gotoxy(22,10); printf("1");
gotoxy(22,12); printf(":");
gotoxy(22,14); printf("i");
gotoxy(22,16); printf(":");
gotoxy(22,18); printf("m");

gotoxy(26,8);  printf("1");
gotoxy(31,8);  printf("...");
gotoxy(36,8);  printf("j");
```

```
gotoxy(41,8);  printf("...");
gotoxy(46,8);  printf("n");
gotoxy(26,10); printf("C11");
gotoxy(26,12); printf(":");
gotoxy(26,14); printf("Ci1");
gotoxy(26,16); printf(":");
gotoxy(26,18); printf("Cm1");
gotoxy(31,10); printf("...");
gotoxy(31,14); printf("...");
gotoxy(31,18); printf("...");
gotoxy(36,10); printf("C1j");
gotoxy(36,12); printf(":");
gotoxy(36,14); printf("Cij");
gotoxy(36,16); printf(":");
gotoxy(36,18); printf("Cmj");
gotoxy(41,10); printf("...");
gotoxy(41,14); printf("...");
gotoxy(41,18); printf("...");
gotoxy(46,10); printf("C1n");
gotoxy(46,12); printf(":");
gotoxy(46,14); printf("Cin");
gotoxy(46,16); printf(":");
gotoxy(46,18); printf("Cmn");
row = 12 + fact * 2;
gotoxy(25,row); printf("B1");
gotoxy(30,row); printf("...");
gotoxy(35,row); printf("Bj");
gotoxy(40,row); printf("...");
gotoxy(45,row); printf("Bn");
col = 30 + mark * 5;
gotoxy(col,10);  printf("A1");
gotoxy(col,12);  printf(":");
gotoxy(col,14);  printf("Ai");
gotoxy(col,16);  printf(":");
gotoxy(col,18);  printf("Am");
gotoxy(col,row); printf(" ä Ai = ã Bj ");
clear();
}

void algorithm()
 {
 screen();
 gotoxy(3,6);
 printf("Steps for finding the Initial BFS using Vogel's Approximation Method :-");
 gotoxy(3,7);
```

```
printf("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ");
 gotoxy(3,9);
 printf("Step 1. For each row of the transportation table identify the smallest
and");
 gotoxy(3,10);
 printf("      next to smallest cost. Determine the difference between them");
 gotoxy(3,11);
 printf("      (penalty) for each row. Similarly, compute the penalty for each");
 gotoxy(3,12);

 printf("      column.");
 gotoxy(3,13);
 printf("Step 2. Identify the row or column with the largest penalty among all
the");
 gotoxy(3,14);
 printf("      rows and columns. If a tie occurs, use any arbitrary tie breaking");
 gotoxy(3,15);
 printf("      choice. Let the largest penalty correspond to ith row and let Cij be");
 gotoxy(3,16);
 printf("      the smallest cost in the ith row. Allocate the largest possible");
 gotoxy(3,17);
 printf("      amount Xij = min(Ai, Abj) in cell (i, j) and cross out the ith row");
 gotoxy(3,18);
 printf("      or the jth column.");
 gotoxy(3,19);
 printf("Step 3. Repeat steps 1 & 2 for the reduced transportation problem.
Repeat");
 gotoxy(3,20);
 printf("      the procedure until all the requirements are satisfied.");
 clear();
 }

void accept()
 {
 gotoxy(3,6);
 printf(" Is the transportation problem a  ");
 gotoxy(5,8);
 printf(" A> Maximization problem  ");
 gotoxy(5,9);
 printf(" I> Minimization problem  ");
 gotoxy(3,11);
 printf(" (Press 'a' for a maximization problem ");
 gotoxy(3,12);
 printf("   and 'i' for a minimization one ) ");
```

```
valid = NO;
while ( valid != YES )
 {
 gotoxy(45, 12);
 ch1 = getche();
 if ( (ch1 == 'a') || (ch1 == 'i') || (ch1 == 'A') || (ch1 == 'I') )
  valid = YES;
 else
  {
  gotoxy(2,24);
  clreol();
  gotoxy(79,24);
  printf("%c",186);
  gotoxy(2,24);
  printf("  Wrong key !  Press the right one < 'a' or 'i' >");
  }
 }
screen();
valid = NO;
while ( valid != YES )
 {
 gotoxy(3,6);
 printf("  Number of factories (rows) :  ");
 gotoxy(35,6);
 validity();
 scanf("%d",&fact);

 if ( fact > 0 )
  {
  for ( i = 2 ; i < 40 ; i++ )
      {
      gotoxy(i,24);
      printf(" ");
      }
  valid = YES;
  }
 else
  {
  gotoxy(2,24);
  clreol();
  gotoxy(79,24);
  printf("%c",186);
  gotoxy(2,24);
  printf("  Number of factories cannot be <= 0 ! ");
  row = 6;
  col = 35;
```

```
    for ( i = 0; i < 10 ; i++ )
        {
        gotoxy(col+i,row);
        printf(" ");
        }
    }
  }
 valid = NO;
 while ( valid != YES )
  {
  gotoxy(3,8);
  printf("  Number of markets (columns) :  ");
  gotoxy(35,8);
  validity();
  scanf("%d",&mark);
  if ( mark > 0 )
   {
   for ( i = 2 ; i < 40 ; i++ )
       {
       gotoxy(i,24);
       printf(" ");
       }
   valid = YES;
   }
  else
   {
   gotoxy(2,24);
   clreol();
   gotoxy(79,24);
   printf("%c",186);
   gotoxy(2,24);
   printf("  Number of markets cannot be <= 0 ! ");
   row = 8;
   col = 35;
   for ( i = 0; i < 10 ; i++ )
       {
       gotoxy(col+i,row);
       printf(" ");
       }
   }
 }

for ( i = 0 ; i < MAX ; i++ )
 for ( j = 0 ; j < MAX ; j++ )
  cost[i][j] = 0;
screen();
```

```
 draw();
 accept_el();
 }

void validity()                          /* To check if the key */
 {                                       /* pressed is <enter>, */
 int key;                                /* <backspace>, <esc>  */
 do                                      /* or <tab>            */
  {
  key = bioskey(1);
  if ( (key == 283) || (key == 7181) || (key == 19712) || (key == 19200) )
   {
   key = bioskey(0);
   key = 0;
   }
  }
  while (key == 0);
 }

void outline()
 {
 col = 0;
 for (i = 0; i < 4 ; i++ )
  {
  gotoxy(20 + i,7);
  printf("%c",205);
  row = 9 + fact*2;
  gotoxy(20 + i,row);
  printf("%c",205);
  gotoxy(20 + i,9);
  printf("%c",196);
  }
 for (i = 0 ; i < mark ; i++ )
  for ( j = 0 ; j < 5 ; j++ )
   {
   col++;
   gotoxy(24 + col,7);
   printf("%c",205);
   row = 9 + fact*2;
   gotoxy(24 + col,row);
   printf("%c",205);
   gotoxy(24 + col,9);
   printf("%c",196);
   }
 for (i = 0; i < 2 ; i++ )
  {
```

```
  gotoxy(20,7+i);
  printf("%c",186);
  col = 24 + mark*5;
  gotoxy(col,7+i);
  printf("%c",186);
  gotoxy(24,7+i);
  printf("%c",179);
  }

row = 0;
for (i = 0 ; i < fact ; i++ )
 for ( j = 0 ; j < 2 ; j++ )
  {
  row++;
  gotoxy(20,9+row);
  printf("%c",186);
  col = 24 + mark*5;
  gotoxy(col,9+row);
  printf("%c",186);
  gotoxy(24,9+row);
  printf("%c",179);
  }
gotoxy(20,7);
printf("%c",201);
col = 24 + mark*5;
gotoxy(col,7);
printf("%c",187);
row = 9 + fact*2;
gotoxy(col,row);
printf("%c",188);
gotoxy(20,row);
printf("%c",200);
gotoxy(24,7);
printf("%c",209);
gotoxy(24,row);
printf("%c",207);
gotoxy(20,9);
printf("%c",199);
gotoxy(col,9);
printf("%c",182);
gotoxy(24,9);
printf("%c",197);
gotoxy(26,6);
printf("MARKETS");
row = 9;
if ( fact >= 3 )
```

```
 {
 gotoxy(19,row+1);
 printf("S");
 gotoxy(19,row+2);
 printf("U");
 gotoxy(19,row+3);
 printf("P");
 gotoxy(19,row+4);
 printf("P");
 gotoxy(19,row+5);
 printf("L");
 gotoxy(19,row+6);
 printf("Y");
 }
else
if ( fact < 3 )
 {
 gotoxy(18,row+1);
 printf("SU");
 gotoxy(18,row+2);
 printf("PP");
 gotoxy(18,row+3);
 printf("LY");
 }

 row = 12 + (fact * 2);
 gotoxy(5,row);
 printf("MARKET");
 row = 9 + (fact * 2) + 4;
 gotoxy(5,row);
 printf("REQUIREMENT");
 col = 24 + (mark * 5) + 4;
 gotoxy(col,7);
 printf("FACTORY");
 col = 24 + (mark * 5) + 4;
 gotoxy(col,8);
 printf("CAPACITY");
 }

void draw()
 {
 outline();
 row = 9;
 col = 29 + (mark * 5);
 for ( i = 0 ; i < fact ; i++ )
  for ( j = 0 ; j < 2 ; j++ )
```

```
    {
    row++;
    gotoxy(col,row);
    printf("%c",179);
    gotoxy(col+5,row);
    printf("%c",179);
    }
  for ( j = 0 ; j < 5 ; j++ )
   {
   col++;
   gotoxy(col,9);
   printf("%c",196);
   row = 9 + ( fact * 2 );
   gotoxy(col,row);
   printf("%c",196);
   }
  row = 9;
  for ( i = 0 ; i < fact-1 ; i++ )
   {
   row = row + 2;
   col = 29 + (mark * 5);
   gotoxy(col,row);
   printf("%c",195);
   for ( k = 0 ; k < 5 ; k++ )
    {
    col++;
    gotoxy(col,row);
    printf("%c",196);
    }
   gotoxy(col,row);
   printf("%c",180);
   }
  col = 24;
  for ( i = 0 ; i < mark-1 ; i++ )
   {
   row = 9;
   col = col + 5;
   gotoxy(col,row);
   printf("%c",194);

   for ( j = 0 ; j < fact ; j++ )
    for ( k = 0 ; k < 2 ; k++ )
        {
        row++;
        gotoxy(col,row);
        printf("%c",179);
```

```
        }
   gotoxy(col,row);
   printf("%c",207);
   }
 row = 9;
 for ( i = 0 ; i < fact-1 ; i++ )
  {
  col = 24;
  row = row + 2;
  gotoxy(col,row);
  printf("%c",195);
  for ( j = 0 ; j < mark ; j++ )
   {
   for ( k = 0 ; k < 5 ; k++ )
        {
        col++;
        gotoxy(col,row);
        printf("%c",196);
        }
   gotoxy(col,row);
   printf("%c",197);
   }
  gotoxy(col,row);
  printf("%c",182);
  }
 col = 21;
 for ( i = 0 ; i < mark ; i++ )
  {
  col = col + 5;
  gotoxy(col, 8);
  printf("%d",i+1);
  }
 row = 8;
 for ( i = 0 ; i < fact ; i++ )
  {
  row = row + 2;
  gotoxy(22, row);
  printf("%d",i+1);
  }
 row = 11 + (fact * 2);
 col = 24;
 for ( i = 0 ; i < mark ; i++ )
  for ( j = 0 ; j < 5 ; j++ )
   {
   col++;
   gotoxy(col,row);
```

```
        printf("%c",196);
        gotoxy(col,row+2);
        printf("%c",196);
        }

      for ( j = 0 ; j < 2 ; j++ )
       {
       row++;
       gotoxy(24,row);
       printf("%c",179);
       col = 24 + ( mark * 5 );
       gotoxy(col,row);
       printf("%c",179);
       }
      col = 24;
      for ( i = 0 ; i < mark-1 ; i++ )
       {
       row = 11 + (fact * 2);
       col = col + 5;
       gotoxy(col,row);
       printf("%c",194);
       for ( k = 0 ; k < 2 ; k++ )
        {
        row++;
        gotoxy(col,row);
        printf("%c",179);
        }
       gotoxy(col,row);
       printf("%c",193);
       }
     row = 11 + fact * 2;
     col = 24 + mark * 5;
     gotoxy(24,row);
     printf("%c",218);
     gotoxy(col,row);
     printf("%c",191);
     gotoxy(col,row+2);
     printf("%c",217);
     gotoxy(24,row+2);
     printf("%c",192);
     col = 29 + mark*5;
     gotoxy(col,9);
     printf("%c",218);
     gotoxy(col+5,9);
     printf("%c",191);
     row = 9 + fact*2;
```

```
gotoxy(col+5,row);
printf("%c",217);
gotoxy(col,row);
printf("%c",192);
}

void accept_el()                        /* Accept all criteria */
{                                       /* of the given problem */
flag = 0;
gotoxy(2,24);
clreol();
gotoxy(79,24);
printf("%c",186);
gotoxy(2,24);
printf("Enter all the pre-assigned costs in the corresponding cells.");
row = 8;

for ( i = 0 ; i <fact ; i++ )
 {
 row = row + 2;
 col = 20;
 for ( j = 0 ; j <mark ; j++ )
  {
  col = col + 5;
  gotoxy(col,row);
  validity();
  scanf("%d",&cost[i+1][j+1]);
  }
 }
row = 9 + (fact * 2) + 3;
col = 20;
total_supply = 0;
// total_demand=0;
gotoxy(2,24);
clreol();
gotoxy(79,24);
printf("%c",186);
gotoxy(2,24);
printf("Enter the maximum supply to each of the markets.");
for ( i = 0 ; i <mark ; i++ )
 {
 col = col + 5;
 gotoxy(col,row);
 validity();
 scanf("%d",&supply[i+1]);
 //total_supply = total_supply + supply[i+1];
```

```
// scanf("%d",&demand[i+1]);
 // total_demand = total_demand + demand[i+1];

 }
col = 24 + (mark * 5) + 6;
row = 8;
total_demand = 0;
//   total_supply=0;
gotoxy(2,24);
clreol();
gotoxy(79,24);
printf("%c",186);
gotoxy(2,24);
printf("Enter the maximum requirement from each of the factories.");
for ( i =0 ; i <fact : i++ )
 {
 row = row + 2;
 gotoxy(col,row);
 validity();
 scanf("%d",&demand[i+1]);
 // total_demand = total_demand + demand[i+1];
 // scanf("%d",&supply[i+1]);
 // total_supply = total_supply + supply[i+1];
  }
}

void  check()                    /* If the objective is to maximise then  */
 {                               /* negate all the pre-assigned costs and */
 chk_flag = 0;                /* solve as a minimisation problem */
 if ( (ch1 == 'a') || (ch1 == 'A') )
  {
  chk_flag = 1;                  /* Store the original costs in a */
  i = 0;                         /* temporary matrix          */
  while ( i < fact )
   {
   j = 0;

   while ( j < mark )
       {
       temp[i][j] = cost[i][j];
       cost[i][j] = -cost[i][j];
       j++;
       }
    i++;
    }
  }
```

```
        }

 void modify()
  {
   int ts=0,td=0;
   for(i=0;i<mark;i++)
   {
    ts=ts+supply[i+1];
   }
   for(i=0;i<fact;i++)
   {
    td=td+demand[i+1];
   }
   mod_flag=0;
   if(ts!=td)
   {
          mod_flag=1;
          if(ts<td)
          {
                mark++;
                supply[mark]=td-ts;
          }
          if(td<ts)
          {
                fact++;
                demand[fact]=ts-td;
          }
   }
}

void display(array temp)
 {
 screen();
 draw();
 if (flag)
  {
  gotoxy(3,24);
  printf("INITIAL BASIC FEASIBLE SOLUTION");
  }
 fill(temp);
 }

void fill(array temp)
{
 row = 8;
 for ( i = 0; i < fact ; i++ )
```

```
    {
    row = row + 2;
    col = 20;
    for ( j = 0 ; j < mark ; j++ )
     {
     col = col + 5;
     gotoxy(col,row);
     printf("%d",temp[i+1][j+1]);
     }
   }

  row = 9 + (fact * 2) + 3;
  col = 20;
  for ( i = 0 ; i < mark ; i++ )
   {
   col = col + 5;
   gotoxy(col,row);
   printf("%d",supply[i+1]);
   }
  col = 24 + (mark * 5) + 6;
  row = 8;
  for ( i = 0 ; i < fact ; i++ )
   {
   row = row + 2;
   gotoxy(col,row);
   printf("%d",demand[i+1]);
   }
  if (!flag)
   {
   gotoxy(2,24);
   clreol();
   gotoxy(79,24);
   printf("%c",186);
   }
  gotoxy(50,24);
  printf("Press any key to continue....");
  getch();
  }

void initfeas()
 {
 flag = 1;
 for ( i = 0 ; i < fact ; i++ )   /* All cells of the assignment  */
  for ( j = 0 ; j < mark ; j++ )        /* matrix are given value zero  */
       asgn[i][j] = 0;
 for ( j = 0 ; j < mark ; j++ )/* A temporary supply list is   */
```

```
 t_supply[j] = supply[j];            /* created                    */
for ( j = 0 ; j < fact ; j++ )  /* A temporary demand list is  */
 t_demand[j] = demand[j];                  /* created                    */
t_mark = mark;              /* t_mark & t_fact are dummy variables */
t_fact = fact;          /* which indicate the number of rows and*/
                            /* columns of the matrix being processed*/

while ( (t_mark > 0) && (t_fact > 0) )
 {
 for ( i = 0 ; i < mark ; i++ )        /* c_pen has a list of the column */
  c_pen[i] = 0;                        /* penalties                  */
 for ( i = 0 ; i < fact ; i++ ) /* r_pen has a list of the row   */
  r_pen[i] = 0;                        /* penalties                  */

 for ( i = 0 ; i < mark ; i++ )     /* Find the penalty of all columns*/
  {                                    /* of the current matrix      */
  m = i;
  taken_flag = 0;              /* c_index is a list of all the   */
  for ( l = 0 ; l < c_count ; l++ )    /* columns in which no more cells */
      if ( m == c_index[l] )           /* can be assigned values     */
       taken_flag = 1;           /* c_count indicates the number   */

  if ( !taken_flag )           /* of elements in c_index     */
      {
      min1 = M;
      for ( j = 0 ; j < fact ; j++ )  /* min1 indicates the first      */
       {                               /* minimum of the current column */
       m = j;
       taken_flag = 0;              /* r_index is a list of all the */
       for ( l = 0 ; l < r_count ; l++ ) /* rows in which no more cells */
        {
        if ( j == r_index[l] )          /* can be assigned values     */
         taken_flag = 1;                /* r_count indicates the number */
        if ( !taken_flag )              /* of elements in r_index     */
         if ( cost[j][i] < min1 )
              {
              min1 = cost[j][i];
              taken = j;
              }
        }
       }
      min2 = M;                         /* min2 indicates the second*/
      for ( j = 0 ; j < fact ; j++ )  /* minimum of the current column */
       {
       m = j;
       taken_flag = 0;
```

```
        for ( l = 0 ; l < r_count ; l++ )
         {
         if ( j == r_index[l] )
          taken_flag = 1;
         if ( !taken_flag )
          if ( (cost[j][i] < min2) && (cost[j][i] >= min1) && (j != taken) )
              min2 = cost[j][i];
         }
        c_pen[i] = abs(min2 - min1); /* Penalty for the particular */
        }                            /* column is found           */
       }
 }
for ( i = 0 ; i < fact ; i++ )     /* Find the penalty of all rows */
 {                                 /* of the current matrix        */
 taken_flag = 0;
 m = i;
 for ( l = 0 ; l < r_count ; l++ )
     if ( m == r_index[l] )
      taken_flag = 1;
 if ( !taken_flag )
     {
     min1 = M;                     /* min1 indicates the first    */
     for ( j = 0 ; j < mark ; j++ ) /* minimum of the current row  */
      {
      taken_flag = 0;
      m = j;
      for ( l = 0 ; l < c_count ; l++ )
       if ( m == c_index[l] )
         taken_flag = 1;
      if ( !taken_flag )
       if ( cost[i][j] < min1 )
         {
         min1 = cost[i][j];
         taken = j;
         }
       }

      min2 = M;                     /* min2 indicates the second*/
      for ( j = 0 ; j < mark ; j++ ) /* minimum of the current row */
       {
       taken_flag = 0;
       m = j;
       for ( l = 0 ; l < c_count ; l++ )
        if ( m == c_index[l] )
          taken_flag = 1;
       if ( !taken_flag )
```

```
        {
        if ( (cost[i][j] < min2) && (cost[i][j] >= min1) && (j != taken) )
          min2 = cost[i][j];
        }
       }
      r_pen[i] = abs(min2 - min1);          /* Penalty for the particular */
      }                              /* row is found       */
 }
max = 0;                             /* Maximum of all penalties is */
flg = 0;                                    /* found by scanning thro' all */
for ( i = 0 ; i < mark ; i++ )       /* the column penalties and then */
 if ( c_pen[i] > max )                      /* the row penalties          */
      {
      max = c_pen[i];
      col = i ;
      flg = 1;
      }
for ( i = 0 ; i < fact ; i++ )
 if ( r_pen[i] > max )
      {
      max = r_pen[i];
      row = i ;
      flg = 2;
      }
if ( flg == 1 )                             /* If the maximum is a column */
 {                                          /* penalty then find the      */
 min = M;                                   /* minimum in that column     */
 for ( k = 0 ; k < fact ; k++ )
      {
      taken_flag = 0;
      m = k;
      for ( l = 0 ; l < r_count ; l++ )
       if ( m == r_index[l] )
         taken_flag = 1;
      if ( !taken_flag )
       if ( cost[k][col] < min )
         {
         min = cost[k][col];
         row = k;                           /* Row indicates the row in   */
         }                                  /* which the minimum value was */
      }                                     /* found                      */
 }
else
if ( flg == 2 )                             /* If the maximum is a column */
 {                                          /* penalty then find the      */
 min = M;
```

```
                        /* minimum in that column    */

    for ( k = 0 ; k < mark ; k++ )
        {
        taken_flag = 0;
        m = k;
        for ( l = 0 ; l < c_count ; l++ )
         if ( m == c_index[l] )
           taken_flag = 1;
        if ( !taken_flag )
         if ( cost[row][k] < min )
           {
           min = cost[row][k];
           col = k;                    /* Column indicates the column in  */
           }                           /* which the minimum value was */
        }                              /* found                       */
    }
   if ( t_supply[col] <= t_demand[row] )   /* If supply <= demand for */
    {                                      /* that cell, then assign */
    asgn[row][col] = t_supply[col];        /* the supply value       */
    t_demand[row] = t_demand[row] - t_supply[col];    /* Reduce demand */
    t_supply[col] = 0;                      /* accordingly   */
    c_index[c_count] = col;              /* Include that column in */
    c_count++;                             /* c_index. Reduce the no.*/
    t_mark--;                              /* of columns by one in the */
    }                                      /* next matrix to be      */
   else                                 /* processed.             */
   if ( t_supply[col] > t_demand[row] )
    {                           /* If supply > demand for */
    asgn[row][col] = t_demand[row];          /* that cell, then assign */
    t_supply[col] = t_supply[col] - t_demand[row];  /* the demand value. */
    t_demand[row] = 0;                     /* Reduce supply accordingly */
    r_index[r_count] = row;                /* Include that row in    */
    r_count++;                             /* r_index. Reduce the no. */
    t_fact--;                              /* of rows by one in the  */
    }                                      /* next matrix to be      */
   }                                       /* processed.             */
}

void estimate()                            /* Estimate the total   */
{                                          /* cost by finding the  */
total_cost = 0;                            /* summation of the     */
 if ( chk_flag == 0 )                      /* product of the assigned */
  {                                        /* value to each cell and */
  for ( i = 0 ; i < fact ; i++ )        /* the corresponding pre- */
   for ( j = 0 ; j < mark ; j++ )          /* assigned cost.       */
```

```
        total_cost = total_cost + ( cost[i][j] * asgn[i][j] );
  }
 else
 if ( chk_flag == 1 )
   {
   for ( i = 0 ; i < fact ; i++ )
    for ( j = 0 ; j < mark ; j++ )
        total_cost = total_cost + ( temp[i][j] * asgn[i][j] );
   }
 }

void assign()
 {
 screen();
 gotoxy(3,6);
 printf("Initial Basic Feasible Solution :-");
 gotoxy(3,7);
 printf("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ");
 k = 0;
 for ( i = 0 ; i < fact ; i++ )
  for ( j = 0 ; j < mark ; j++ )
   if ( asgn[i][j] != 0 )
        {
        gotoxy(7,9+k);
        printf("Factory %d ships %d units of goods to Market
%d.",i+1,asgn[i][j],j+1);
        k++;
        }
  gotoxy(7, 11+k);
  if ( (ch1 == 'a') || (ch1 == 'A') )
   printf("The maximum profit is estimated to be %d.",total_cost);
  else
  if ( (ch1 == 'i') || (ch1 == 'I') )
   printf("The minimum cost is estimated to be %d.",total_cost);
  }
 void trans1()
 {
 int i,j;
 nos=fact;
 nod=mark;
 printf("\n");
 for(i=1;i<=nos;i++)
        {
                for(j=1;j<=nod;j++)
                {
                        dcost[i][j]=cost[i][j];
```

```
                }
        }
for(i=1;i<=nos+nos;i++)
{
 sup1[i]=demand[i];
}
for(i=1;i<=nod+nod;i++)
{
 dem1[i]=supply[i];
}
}
void proc()
{
 int count=0,i=1,j=1,l,flag=1,index,RMax=0,CMax=0,Rarray[' '],Carray[' '];
 int temp,Rdiff[' '],Cdiff[' '],Rindex,Cindex,RMin=0,CMin=0,row,col;
 int x,y,diff;
        while(flag)
        {
                int flag1=0;
                for(l=1;l<=nod;l++)
                if(dem1[l]==0)
                 flag1=1;
                else
                {
                 flag1=0;
                 break;
                }
                if(flag1)
                 flag=0;
                else
                {

                  RMax=0;
                  CMax=0;

                  for(row=1;row<=nos;row++)
                  {
                        for(col=1;col<=nod;col++)
                        {
                          if(dem1[col]!=0&&sup1[row]!=0)
                                Rarray[col]=dcost[row][col];
                          else
                                Rarray[col]=32767;
                        }
                        //sort array in ascending
```

```
    for(x=1;x<=nod;x++)

    for(y=x;y<=nod;y++)
    if(Rarray[x]>Rarray[y])
    {
            temp=Rarray[x];
            Rarray[x]=Rarray[y];
            Rarray[y]=temp;
    }

    diff=Rarray[2]-Rarray[1];
    Rdiff[row]=diff;
}//end for loop

for(col=1;col<=nod;col++)
{
    for(row=1;row<=nos;row++)
    {
      if(sup1[row]!=0&&dem1[col]!=0)
        Carray[row]=dcost[row][col];
      else
        Carray[row]=32767;
    }
    //sort array in ascending
    for(x=1;x<=nos;x++)

    for(y=x;y<=nos;y++)
    if(Carray[x]>Carray[y])
    {
            temp=Carray[x];
            Carray[x]=Carray[y];
            Carray[y]=temp;
    }

    diff=Carray[2]-Carray[1];
    Cdiff[col]=diff;
}//end for loop


//finding Max Vogel number

for(row=1;row<=nos;row++)
if(RMax<Rdiff[row])
{
  RMax=Rdiff[row];
  Rindex=row;
```

```
}
for(col=1;col<=nod;col++)
if(CMax<Cdiff[col])
{
  CMax=Cdiff[col];
  Cindex=col;
}
if(RMax<=CMax)
{
  CMin=32767;
  for(row=1;row<=nos;row++)
  {
      if(sup1[row]!=0&&dem1[Cindex]!=0)
      if(CMin>cost[row][Cindex])
      {
              CMin=cost[row][Cindex];
              i=row;
      }
  }
  j=Cindex;
}
else
{
  RMin=32767;
  for(col=1;col<=nod;col++)
  {
      if(dem1[col]!=0&&sup1[Rindex]!=0)
      if(RMin>cost[Rindex][col])
      {
              RMin=cost[Rindex][col];
              j=col;
      }
  }
  i=Rindex;
}
//allocating amount at selected cell
if(sup1[i]<=dem1[j])
{
      basic[i][j]=sup1[i];
      dem1[j]-=sup1[i];
      sup1[i]=0;
}
else
{
      basic[i][j]=dem1[j];
      sup1[i]-=dem1[j];
```

```
                    dem1[j]=0;
                }
            }
}//end of while
            printf("\n Basic Cells:\n");
            for(i=1;i<=nos;i++)
            for(j=1;j<=nod;j++)
            {
             tcost+=basic[i][j]*cost[i][j];
             if(basic[i][j]!=0)
             {
              //cout<<endl<<'x'<<i<<j<<':'<<basic[i][j];
              printf("\n x:%d%d ':'%d",i,j,basic[i][j]);
              count=count+1;
             }
            }
            getch();
            display(basic);
            if(count==(nos+nod-1))
             { gotoxy(3,20);
              printf("Solution is Non-Degenerate");}
            else
             { gotoxy(3,20);
              printf("Solution is Degenerate");    }
        // return(tcost);
        gotoxy(3,22);
        printf("Minimum Cost = %d",tcost);
        getch();

}


/*void proc()
{
int count=0,tcost=0,i=1,j=1,l,flag=1,index,RMax=0,CMax=0,Rarray[' '],Carray[' '];
int temp,Rdiff[' '],Cdiff[' '],Rindex,Cindex,RMin=0,CMin=0,row,col;
int x,y,diff;
      while(flag)
      {

            int flag1=0;
            for(l=1;l<=nod;l++)
            if(demand[l]==0)
             flag1=1;
            else
            {
```

```
 flag1=0;
 break;
}
if(flag1)
 flag=0;
else
{

 RMax=0;
 CMax=0;

 for(row=1;row<=nos;row++)
 {
      for(col=1;col<=nod;col++)
      {
        if(demand[col]!=0&&supply[row]!=0)
              Rarray[col]=dcost[row][col];
        else
              Rarray[col]=32767;
      }
      //sort array in ascending
      for(x=1;x<=nod;x++)

      for(y=x;y<=nod;y++)
      if(Rarray[x]>Rarray[y])
      {
            temp=Rarray[x];
            Rarray[x]=Rarray[y];
            Rarray[y]=temp;
      }

      diff=Rarray[2]-Rarray[1];
      Rdiff[row]=diff;
 }//end for loop

 for(col=1;col<=nod;col++)
 {
      for(row=1;row<=nos;row++)
      {
        if(supply[row]!=0&&demand[col]!=0)
          Carray[row]=dcost[row][col];
        else
          Carray[row]=32767;
      }
      //sort array in ascending
      for(x=1;x<=nos;x++)
```

```
    for(y=x;y<=nos;y++)
    if(Carray[x]>Carray[y])
    {
            temp=Carray[x];
            Carray[x]=Carray[y];
            Carray[y]=temp;
    }

    diff=Carray[2]-Carray[1];
    Cdiff[col]=diff;
}//end for loop


//finding Max Vogel number

for(row=1;row<=nos;row++)
if(RMax<Rdiff[row])
{
  RMax=Rdiff[row];
  Rindex=row;
}
for(col=1;col<=nod;col++)
if(CMax<Cdiff[col])
{
  CMax=Cdiff[col];
  Cindex=col;
}
if(RMax<=CMax)
{
  CMin=32767;
  for(row=1;row<=nos;row++)
  {
     if(supply[row]!=0&&demand[Cindex]!=0)
     if(CMin>cost[row][Cindex])
     {
            CMin=cost[row][Cindex];
            i=row;
     }
  }
  j=Cindex;
}
else
{
  RMin=32767;
  for(col=1;col<=nod;col++)
```

```
                    {
                        if(demand[col]!=0&&supply[Rindex]!=0)
                        if(RMin>cost[Rindex][col])
                        {
                                RMin=cost[Rindex][col];
                                j=col;
                        }
                    }
                    i=Rindex;
                }
                //allocating amount at selected cell
                if(supply[i]<=demand[j])
                {
                     basic[i][j]=supply[i];
                     demand[j]-=supply[i];
                     supply[i]=0;
                }
                else
                {
                     basic[i][j]=demand[j];
                     supply[i]-=demand[j];
                     demand[j]=0;
                }
            }
}//end of while
            printf("\n Basic Cells:\n");
            for(i=1;i<=nos;i++)
            for(j=1;j<=nod;j++)
            {
             tcost+=basic[i][j]*cost[i][j];
             if(basic[i][j]!=0)
             {
              //cout<<endl<<'x'<<i<<j<<':'<<basic[i][j];
              printf("\n'x'%d%d ':'%d",basic[i][j]);
              count=count+1;
             }
            }
            if(count==(nos+nod-1))
              printf("\n Solution is Non-Degenerate");
            else
              printf("\n Solution is Degenerate");
        //  return(tcost);
        printf("\n Minimum Cost  = %d",tcost);


} */
```

OUTPUT:-

```
   TO FIND THE INITIAL FEASIBLE SOLUTION OF A TRANSPORTATION
PROBLEM
              USING VOGEL'S APPROXIMATION METHOD
-------------------------------------------------------------------------------
Problem Definition :-
---------------------
   In the assignment problem, the objective is to transport goods from more
than one centre, called 'origins', to more than one places, called
'destinations', in such manner that the cost of transportation is minimum.
   The availability of the various origins and the requirements of
destinations are finite and constitute of limited resources.

Mathematically, the transportation problem can be stated as :

   Let there be 'm' origins, ith origin possessing 'Ai' units of a certain
product and 'n' destinations, jth destination requiring 'Bj' units of the
product.
   Let 'Cij' be the cost of shipping 1 unit from the ith origin to the jth
destination.
   Let 'Xij' be the amount to be shipped from the ith origin to the jth
destination.

Press any key to continue....

   TO FIND THE INITIAL FEASIBLE SOLUTION OF A TRANSPORTATION
PROBLEM
              USING VOGEL'S APPROXIMATION METHOD
-------------------------------------------------------------------------------
Minimise the total cost
   Z = _ _ Cij.Xij , for i = 1,2....n, j = 1,2....n

subject to the restrictions of the form :

Availability constraint :
   _ Xij = Ai for i = 1,2....m, j = 1,2....n

Requirement constraint :
   _ Xij = Bj for i = 1,2....m, j = 1,2....n

Total supply = Total demand (i.e.)
   _ Ai = _ Bj for i = i,2....m, j = 1,2....n
```

Press any key to continue....

TO FIND THE INITIAL FEASIBLE SOLUTION OF A TRANSPORTATION PROBLEM

USING VOGEL'S APPROXIMATION METHOD

Tabular Representation

```
                MARKETS
        +-----------------------+ FACTORY
        | | 1   ... j   ... n | CAPACITY
        |-+-------------------| +---+
        S| 1  C11 ... C1j ... C1n| |A1 |
        U|
        P|: :     :     :    :
        P|
        L| i  Ci1 ... Cij ... Cin| |Ai |
        Y|
        |: :     :     :    :
        |
        | m | Cm1 ... Cmj ... Cmn| |Am |
        +-------------------------+ +---+

        MARKET      B1  ... Bj  ... Bn    _Ai = _Bj
 REQUIREMENT
```

Press any key to continue....

TO FIND THE INITIAL FEASIBLE SOLUTION OF A TRANSPORTATION PROBLEM

USING VOGEL'S APPROXIMATION METHOD

Steps for finding the Initial BFS using Vogel's Approximation Method :-

---

Step 1. For each row of the transportation table identify the smallest and next to smallest cost. Determine the difference between them (penalty) for each row. Similarly, compute the penalty for each column.

Step 2. Identify the row or column with the largest penalty among all the rows and columns. If a tie occurs, use any arbitrary tie breaking choice. Let the largest penalty correspond to ith row and let Cij be the smallest cost in the ith row. Allocate the largest possible amount Xij = min(Ai, Abj) in cell (i, j) and cross out the ith row or the jth column.

Step 3. Repeat steps 1 & 2 for the reduced transportation problem. Repeat

the procedure until all the requirements are satisfied.

Press any key to continue....

TO FIND THE INITIAL FEASIBLE SOLUTION OF A TRANSPORTATION PROBLEM
USING VOGEL'S APPROXIMATION METHOD

Is the transportation problem a

A> Maximization problem
I> Minimization problem

(Press 'a' for a maximization problem
and 'i' for a minimization one)

Number of factories (rows) : 3

Number of markets (columns) : 4

	MARKETS				FACTORY CAPACITY
	1	2	3	4	
1	19	30	50	10	7
2	70	30	40	60	9
3	40	8	70	20	18

MARKET REQUIREMENT

5	8	7	14

Enter the maximum requirement from each of the factories.

	MARKETS				FACTORY CAPACITY
	1	2	3	4	
1	19	30	50	10	7
2	70	30	40	60	9
3	40	8	70	20	18

MARKET REQUIREMENT

5	8	7	14

Basic Cells:

x:11 ':'5
x:14 ':'2
x:23 ':'7
x:24 ':'2
x:32 ':'8
x:34 ':'10

MARKETS

	1	2	3	4
1	5	0	0	2
2	0	0	7	2
3	0	8	0	10

FACTORY CAPACITY

7
9
18

MARKET REQUIREMENT

5	8	7	14

Press any key to continue....!

Solution is Non-Degenerate

Transportation Cost = 779

SPECIAL NOTE ABOUT THE TRANSPORTATION PROBLEM

This Program is free from all the errors. and can be easily execute in the computer.

We are using modular approach in this program.

This program contains maximum seven (7) rows and maximum (7) columns for suitability.

When you put the values in table according to problem, you should press the ENTER key for substituting the values row by row.

We have defined all the variables in the both programs.

By VAM Program we find the initial basic feasible solution of transportation problem and for check the optimality we are giving the optimal program in C Lang.

This program is applicable for both the minimization for maximization and balanced or unbalanced problems.

Your suggestions or comments is very necessary for modifying this Program.

—AUTHOR

TRANSPORTATION PROBLEMS

TRANSPORTATION PROBLEMS

OPTIMAL PROGRAM IN "C"

```
//LPP Transportation problem using VAM with optimality.//
#include<process.h>
#include<stdio.h>
#include<conio.h>
        int TC,demand[10],supply[10],nod,nos;
                int stones[10][10];
                int cij[10][10];
                int c[10][10];
                int u[10],v[10];
                int rhs[10];
                int basic[10][10];
                int cost[10][10];
                int dcost[10][10];
                void finduv();
                void  opti();
                void Demand();
                void Supply();
                void Cost();
                void Display();
                void Check();
                int  VAM_Process();
             //  int k;


/*int TRANSPORTATION::basic[10][10];
int TRANSPORTATION::cost[10][10];
int TRANSPORTATION::dcost[10][10];*/

void main()
{
        clrscr();
        printf("\n\nINPUT NO. OF FACTORY SUPPLY/AVIALABILITY :\t");
    scanf("%d",&nos);
        printf("\n\nINPUT NO. OF DEMAND/REQUIREMENT :\t");
    scanf("%d",&nod);
        Supply();
        Demand();
        Cost();
        getch();
        clrscr();
        Display();
        Check();
        TC=VAM_Process();
```

```
        opti();
    //printf("\nTRANSPORTATION COST (TC) = %d",TC);
        getch();
}

int VAM_Process()
{
        int count=0,tcost=0,i=1,j=1,flag=1,index,RMax=0,CMax=0,Rarray['
'],Carray[' '];
        int temp,Rdiff[' '],Cdiff[' '],Rindex,Cindex,RMin=0,CMin=0;
        int row,col,diff,l,x,y;
        int k;
        while(flag)
        {

                int flag1=0;
                for(l=1;l<=nod;l++)
                if(demand[l]==0)
                 flag1=1;
                else
                {
                 flag1=0;
                 break;
                }
                if(flag1)
                 flag=0;
                else
                {

                  RMax=0;
                  CMax=0;

                  for(row=1;row<=nos;row++)
                  {
                        for(col=1;col<=nod;col++)
                        {
                          if(demand[col]!=0&&supply[row]!=0)
                                Rarray[col]=dcost[row][col];
                          else
                                Rarray[col]=32767;
                        }
                        //sort array in ascending
                        for( x=1;x<=nod;x++)
                        for( y=x;y<=nod;y++)
                        if(Rarray[x]>Rarray[y])
                        {
```

```
                temp=Rarray[x];
                Rarray[x]=Rarray[y];
                Rarray[y]=temp;
        }

        diff=Rarray[2]-Rarray[1];
        Rdiff[row]=diff;
 }//end for loop

 for( col=1;col<=nod;col++)
 {
        for(row=1;row<=nos;row++)
        {
          if(supply[row]!=0&&demand[col]!=0)
             Carray[row]=dcost[row][col];
          else
             Carray[row]=32767;
        }
        //sort array in ascending
        for( x=1;x<=nos;x++)
        for( y=x;y<=nos;y++)
        if(Carray[x]>Carray[y])
        {
                temp=Carray[x];
                Carray[x]=Carray[y];
                Carray[y]=temp;
        }

        diff=Carray[2]-Carray[1];
       Cdiff[col]=diff;
 }//end for loop


 //finding Max Vogel number
 for(row=1;row<=nos;row++)
 if(RMax<Rdiff[row])
 {
   RMax=Rdiff[row];
   Rindex=row;
 }
 for(col=1;col<=nod;col++)
 if(CMax<Cdiff[col])
 {
   CMax=Cdiff[col];
   Cindex=col;
 }
```

```
                    if(RMax<=CMax)
                    {
                      CMin=32767;
                      for(row=1;row<=nos;row++)
                      {
                          if(supply[row]!=0&&demand[Cindex]!=0)
                          if(CMin>cost[row][Cindex])
                          {
                                  CMin=cost[row][Cindex];
                                  i=row;
                          }
                      }
                      j=Cindex;
                    }
                    else
                    {
                      RMin=32767;
                      for(col=1;col<=nod;col++)
                      {
                          if(demand[col]!=0&&supply[Rindex]!=0)
                          if(RMin>cost[Rindex][col])
                          {
                                  RMin=cost[Rindex][col];
                                  j=col;
                          }
                      }
                      i=Rindex;
                    }
                    //allocating amount at selected cell
                    if(supply[i]<=demand[j])
                    {
                          basic[i][j]=supply[i];
                          demand[j]-=supply[i];
                          supply[i]=0;
                    }
                    else
                    {
                          basic[i][j]=demand[j];
                          supply[i]-=demand[j];
                          demand[j]=0;
                    }
                }
}//end of while

          //   cout<<"\n Basic Cells:\n";
               k=1;
```

```
                printf("\n*****************************\n");
                printf("\n BASIC CELLS :\n");
                for(i=1;i<=nos;i++)
                for(j=1;j<=nod;j++)
                {
                 tcost+=basic[i][j]*cost[i][j];
                 if(basic[i][j]!=0)
                 {

                  printf("\n x%d%d  :  %d",i,j,basic[i][j]);
                  stones[i][j]=basic[i][j];
                  count=count+1;
                  c[i-1][j-1]=cost[i][j];
                 }
                 if(basic[i][j]==0)
                  cij[i][j]=cost[i][j];
                }

            if(count==(nos+nod-1))
                // cout<<"\n Solution is Non-Degenerate";
                 printf("\n Solution is Non-Degenerate:");
               else
               //  cout<<"\n Solution is Degenerate";
                 printf("\n Solution is Degenerate");
                TC=tcost;
               return(tcost);
}


void Check()
{
 int balance,i;
       int dem=0,sup=0;
       for(i=1;i<=nos;i++)
         sup+=supply[i];

       for(i=1;i<=nod;i++)
         dem+=demand[i];

       if(dem==sup)
       // cout<<"\n Demand and Supply are equal";
        printf("\n Demand and Supply are equal");
       else
       {
        //cout<<"\n Demand and Supply are unequal";
        printf("\n Demand and Supply are unequal");
```

```
            getch();

            if(sup<dem)
            {
                  balance=dem-sup;
                  nos+=1;
                  supply[nos]=balance;
            }
            else
            if(dem<sup)
            {
                  balance=sup-dem;
                  nod+=1;
                  demand[nod]=balance;
            }
            //cout<<endl<<"Balanced Tranportation Problem";
            printf("\nBalanced Tranportation Problem");
            Display();
          }
}

void Display()
{
 int i,j;
       //cout<<"\n Given problem:\n";
       printf("\n Given problem:\n");
       for(i=1;i<=nos;i++)
       {
             for(j=1;j<=nod;j++)
             {
                   //cout<<setw(5)<<cost[i][j];
                   printf("\t%d",cost[i][j]);
                   dcost[i][j]=cost[i][j];
             }
          //cout<<setw(5)<<supply[i]<<endl;
          printf("\t%d\n",supply[i]);
       }
  //   cout<<endl;
       printf("\n");
       for(i=1;i<=nod;i++)
          //  cout<<setw(5)<<demand[i];
            printf("\t%d",demand[i]);

  getch();
}
```

```
void Demand()
{
 int i;
        for(i=1;i<=nod;i++)
        {
                //cout<<"\nInput D"<<i<<':';
                printf("\nINPUT demand D%d : ",i);
             //  cin>>demand[i];
             scanf("%d",&demand[i]);
        }
}
void Supply()
{
 int i;
        for(i=1;i<=nos;i++)
        {
          //     cout<<"\n Input S"<<i<<':';
           // cin>>supply[i];
            printf("\n Input supply   S%d :",i);
            scanf("%d",&supply[i]);
        }
}
void Cost()
{
int i,j;
        for(i=1;i<=nos;i++)
        for(j=1;j<=nod;j++)
        {
                //cout<<"\n Input C"<<i<<j<<':';
                //cin>>cost[i][j];
                printf("\n Input C%d%d :",i,j);
                scanf("%d",&cost[i][j]);
        }
}

void opti()
{

 int i,j,k,p,dee[10][10],uv[10][10],dummys[10][10],nall,vali[10],valj[10],z;
 char ch,asc;
 int bigsma,sma,r1,c1,neg=0,tea,p1,q1,dumsto[10][10],tcost,count;
 do {
  neg=0;
  count=0;
  bigsma=0,sma=0,r1=0,c1=0;
 getch();
```

```
clrscr();
printf("\n*******************************\n");
            printf("\nBASIC CELLS MATRIX\n");
            for(i=1;i<=nos;i++)
        {   for(j=1;j<=nod;j++)
                    {
                    printf("%d\t",stones[i][j]);
                    }
                    printf("\n");
            }
            getch();
            printf("\n*******************************\n");
            printf("\nUNOCCUPIED CELLS VALUES  (cij) \n");

           for(i=1;i<=nos;i++)
            {   for(j=1;j<=nod;j++)
                    {
                    printf("%d\t",cij[i][j]);
                    }
                    printf("\n");
            }
            getch();

            finduv();
           //  getch();
            clrscr();
            printf("\nUi & Vi values are\n");
            printf("\n*******************************\n");
                for(i=0;i<nos;i++)
                        printf("u[%d]=%d\n",i+1,u[i]);
                printf("\n");
                for(j=0;j<nod;j++)
                        printf("v[%d]=%d\n",j+1,v[j]);

                printf("\n*************************\n");
                getch();
             // printf("\n UV CELLS :\n");
              for(i=0;i<nos;i++)
              {   for(j=0;j<nod;j++)
                        {
                        if(cij[i+1][j+1]!=0)
                        {
                                uv[i][j]=u[i]+v[j];
                            //printf("%d\t",uv[i][j]);
                        }
                        else
```

```
                {
                 uv[i][j]=0;
                }
            }
        }

    printf("Ui + Vj  for  EMPTY CELLS\n");
for(i=0;i<nos;i++)
{   for(j=0;j<nod;j++)
            {
             printf("%d\t",uv[i][j]);
            }
    printf("\n");
}

    for(i=0;i<nos;i++)
    {
            for(j=0;j<nod;j++)
            {
              if(cij[i+1][j+1]!=0)
                        dee[i][j]=cij[i+1][j+1]-uv[i][j];
              else
                        dee[i][j]=0;
            }
    }
    printf("\n************************************\n");
    printf("\n Cell Evaluation for Empty cells ( Dij )\n");

    for(i=0;i<nos;i++)
    {
            for(j=0;j<nod;j++)
            {
              printf("%d\t",dee[i][j]);
            }
            printf("\n");
    }
    bigsma=0;
    for(i=0;i<nos;i++)
    {
            for(j=0;j<nod;j++)
            {
             if(dee[i][j]<0)
              {     sma=dee[i][j];
                    if (bigsma>sma)
                    {
                     bigsma=sma;
```

```
                                r1=i;
                                c1=j;
                               }
                             neg=1;
                            }

                          }
                    }
              //    asc=233;
                    if(neg==0)
                    {
                     printf("\n Since All Cells values are positive, hence Solution is
Optimal \n");
                     printf("\n MINIMUM /OPTIMAL TRANSPORTATION COST
(TC) IS   = %d",TC);
                     getch();
                     break;
                    // exit(0);
                    }
                    else
                    {    clrscr();
                         printf("\nSince Cell values are negative  and highest negative
value is %d found at %d%d index\n\n",bigsma,r1,c1);
                        tcost=0;
                       for(i=0;i<nos;i++)
                        { for(j=0;j<nod;j++)
                            {
                             if(r1==i&&c1==j)
                                   dummys[i][j]=bigsma;
                             else
                                dummys[i][j]=stones[i+1][j+1];
                              printf("%d\t",dummys[i][j]);
                            }
                            printf("\n");
                        }
                           for(i=0;i<nos;i++)
                        { for(j=0;j<nod;j++)
                            {
                             dumsto[i][j]=9999;
                            }
                        }
                        printf("\n Enter N Allocation to form loop and its indexes \n");
                        scanf("%d",&nall);
                        for(i=0;i<nall;i++)
                            {
                             printf("Enter %d coordinates\n",i+1);
```

```
                scanf("%d%d",&vali[i],&valj[i]);
               }
             for(i=0;i<nall;i++)
               {
                printf("\n%d coordinates",i+1);
                printf("\n%d\t%d",vali[i],valj[i]);
               }
                getch();
                clrscr();
               printf("\n************* LOOPINGS *********\n");

                for(j=0;j<nall;j++)
                  {
                       p1=vali[j];
                       q1=valj[j];
                       dumsto[p1][q1]=dummys[p1][q1];
                   }

            for(i=0;i<nos;i++)
             {
               for(j=0;j<nod;j++)
                {
                 printf("%d\t",dumsto[i][j]);
                }
               printf("\n");
             }

                printf("\n ******** LOOPINGS AFTER INSERTING
TEETA VLAUE*********\n");
                printf("Enter the value of @ \n");
                scanf("%d",&tea);
               // z=0;

                for(j=0;j<nall;j++)
                  {
                       p1=vali[j];
                       q1=valj[j];
          //           dummystones[p][q]=tea;
                       if(j==0)
                        dumsto[p1][q1]=tea;
                       else
                        dumsto[p1][q1]=dumsto[p1][q1]+tea;
                       tea=tea*-(1);
                  }
          printf("\n********\n");
            for(i=0;i<nos;i++)
```

```
        {
            for(j=0;j<nod;j++)
            {
              printf("%d\t",dumsto[i][j]);
            }
            printf("\n");
        }

       for(i=0;i<nos;i++)
        {
            for(j=0;j<nod;j++)
            {
             if (dumsto[i][j]!=9999)
                stones[i+1][j+1]=dumsto[i][j];
            }
            printf("\n");
        }
       clrscr();
       printf("\n**************************stones\n");
      for(i=0;i<nos;i++)
        {
            for(j=0;j<nod;j++)
            {
              printf("%d\t",stones[i+1][j+1]);
            }
            printf("\n");
        }

        getch();

      for(i=0;i<nos;i++)
         for(j=0;j<nod;j++)
              cij[i][j]=0;

for(i=1;i<=nos;i++)
{
    for(j=1;j<=nod;j++)
     {
          tcost+=stones[i][j]*cost[i][j];
           if(stones[i][j]!=0)
           {
                  count=count+1;
                c[i-1][j-1]=cost[i][j];
           }
           if(stones[i][j]==0)
              cij[i][j]=cost[i][j];
```

```
                }
            }
                getch();
                printf("\n*****************************(cij)\n");
                 for(i=0;i<nos;i++)
                    {
                       for(j=0;j<nod;j++)
                       {
                         printf("%d\t",cij[i+1][j+1]);
                       }
                       printf("\n");
                    }


              //if(count==(nos+nod-1))
                  // cout<<"\n Solution is Non-Degenerate";
              //     printf("\n Solution is Non-Degenerate : %d",tcost);
                  //else
                  // cout<<"\n Solution is Degenerate";
              //    printf("\n Solution is Degenerate : %d ",tcost);
          //      return(tcost);
          TC=tcost;
//        opti();

     } // end of else block
    }while(1);


  getch();

  }  // End of opti function.

void finduv()
        {
//      static int u1[10],v1[10];
        int i,m,j,k=0,l=0,con,pos[10],pos1[10],count=0,big=0,crow[10],index;
        clrscr();

              //clrscr();
           printf("\n**********************\n");
            for(i=0;i<nos;i++)
            {  for(j=0;j<nod;j++)
                     {
                     if(c[i][j]!=0)
                     {
```

```
                    count++;
                }
                }
                crow[i]=count;
                if(big<=crow[i])
                {
                big=crow[i];
                index=i;
                }
            //  printf("index = %d\n",index);
        }


//      printf("%d%d\n",nos,nod);
        getch();

    con=index;
    for(j=0;j<nod;j++)
        if(c[con][j]!=0)        {
            v[j]=c[con][j]-u[con];
            pos[k]=j;
//          printf("v[%d]=%d\n",j+1,v[j]);
//          printf("pos[%d]=%d",k,pos[k]);
            k++;
        }
//  m=0;
//  printf("%d",k);
    for(m=0;m<k;m++)  {
        j=pos[m];
        for(i=0;i<nos;i++)
            if(c[i][j]!=0)   {
                u[i]=c[i][j]-v[j];
                pos1[l]=i;
//              printf("u[%d]=%d\n",i+1,u[i]);
//              printf("pos[%d]=%d",l,pos1[l]);
                l++;
            }
    }
//              printf("\n%d\t%d",m,j);
    i=0;
    for(m=0;m<l;m++)   {
        i=pos1[m];
        for(j=0;j<nod;j++)
            if(c[i][j]!=0)   {
                v[j]=c[i][j]-u[i];
                pos[k]=j;
```

```
//                              printf("v[%d]=%d\n",j+1,v[j]);


//                      printf("pos[%d]=%d",k,pos[k]);
                                k++;
                        }
        }
    /*for(i=0;i<nos;i++)
                printf("u[%d]=%d\n",i+1,u[i]);
        printf("\n");
        for(j=0;j<nod;j++)
                printf("v[%d]=%d\n",j+1,v[j]);*/

        getch();

}
```

OUTPUT :

INPUT NO. OF FACTORY SUPPLY/AVIALABILITY : 3

INPUT NO. OF DEMAND/REQUIREMENT : 4

Input supply S1 :7

Input supply S2 :9

Input supply S3 :18

INPUT demand D1 : 5

INPUT demand D2 : 8

INPUT demand D3 : 7

INPUT demand D4 : 14

Input C11 :19

Input C12 :30

Input C13 :50

Input C14 :10

Input C21 :70

Input C22 :30

Input C23 :40

Input C24 :60

Input C31 :40

Input C32 :8

Input C33 :70

Input C34 :20

Given problem:

```
19   30   50   10   7
70   30   40   60   9
40   8    70   20   18

5    8    7    14
```

Demand and Supply are equal

****

BASIC CELLS :

x11 : 5
x14 : 2
x23 : 7
x24 : 2
x32 : 8
x34 : 10

Solution is Non-Degenerate:

****

BASIC CELLS MATRIX

```
5    0    0    2
0    0    7    2
0    8    0    10
```

****

UNOCCUPIED CELLS VALUES (cij)

```
0    30   50   0
70   30   0    0
40   0    70   0
```

****

Ui & Vi values are

****

u[1]= -10
u[2]=40
u[3]=0

```
********** LOOPINGS *********
9999   9999   9999   9999
9999   -18    9999    2
9999    8     9999    10

******** LOOPINGS AFTER INSERTING TEETA VALUE*********
Enter the value of (a)   2


*****************************stones
5     0     0     2
0     2     7     0
0     6     0     12


*****************************(cij)
 0    30    50    0
70    0     0    60
40    0    70    0


*****************************

BASIC CELLS MATRIX
5     0     0     2
0     2     7     0
0     6     0     12


*****************************

UNOCCUPIED CELLS VALUES  (cij)
0     30    50    0
70    0     0     60
40    0     70    0


**********************
```

```
Ui & Vi values are

******************************
u[1]= -10
u[2]=40
u[3]=0

v[1]=29
v[2]=8
v[3]=0
v[4]=20

*************************


*************************
Ui + Vj  for  EMPTY CELLS
0     -2     -10     0
69     0      0      60
29     0      0      0

***********************************

 Cell Evaluation for Empty cells ( Dij )
0     32     60     0
1     0       0     0
11    0      70     0

Since All Cells values are positive, hence Solution is  Optimal

MINIMUM /OPTIMAL TRANSPORTATION COST (TC) IS   =  743
```