big_m_menu.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int get_slack(int ies[], int m)
{
   int slack = 0;
   for(int i=0; i<m; i++)
   {
      if(ies[i] == -1)
         slack++;
   }
   return slack;
}

int get_surplus(int ies[], int m)
{
   int surplus = 0;
   for(int i=0; i<m; i++)
   {
      if(ies[i] == 1)
         surplus++;
   }
   return surplus;
}

int get_artificial(int ies[], int m)
{
   int slack = get_slack(ies, m);
   return m - slack;
}

void initialize_table(float table[][100], float a[][100], float b[], float c[], int ies[], int m, int n)
{
   float cbv[100];
   for(int i=0; i<m; i++)
   {
      if(ies[i] >= 0)
         cbv[i] = -100000;
      else
         cbv[i] = 0;
   }
   int surplus = get_surplus(ies, m);
   for(int i=0; i<m; i++)
   {
      int curr_surplus = 0;
      for(int j=0; j<n; j++)
         table[i][j] = a[i][j];
      for(int j=n; j<n+surplus; j++)
```

```
            table[i][j] = 0;
        if(ies[i] == 1)
            table[i][n+curr_surplus++] = -1;
        table[i][n+surplus] = b[i];
    }
    for(int i=0; i<=n+surplus; i++)
    {
        float inner_product = 0;
        for(int j=0; j<m; j++)
            inner_product += table[j][i]*cbv[j];
        if(i>=n) table[m][i] = inner_product;
        else table[m][i] = inner_product - c[i];
    }
}

void copy_2d_array(float src[][100], float dest[][100], int m, int n)
{
    for(int i=0; i<m; i++)
    {
        for(int j=0; j<n; j++)
            dest[i][j] = src[i][j];
    }
}

void print_line(int len)
{
    for(int i=0; i<len; i++)
        printf("-");
    printf("\n");
}

void print_simplex_table(float arr[][100], int bv[], int nbv[], int m, int n)
{
    printf("\n");
    printf("         ");
    print_line(10*(n+1)+1);

    printf("         |");
    for(int i=0; i<n; i++)
    {
        printf("    X%d   |", nbv[i]+1);
    }
    printf("    Xb   |\n");

    print_line(10*(n+2));

    for(int i=0; i<=m; i++)
    {
        if(i!=m)
            printf("    X%d   |", bv[i]+1);
        else
            printf("         |");
```

```
        for(int j=0; j<=n; j++)
            printf(" %+7.2f |", arr[i][j]);
        printf("\n");
        if(i!=m)
            print_line(10*(n+2));
        else
        {
            printf("         ");
            print_line(10*(n+1)+1);
        }
    }
    printf("\n");
}

void print_solution(float table[][100], int bv[], int nbv[], int m,
int n, int surplus)
{
    float sol_array[100];
    printf("\nThe solution is: \n");
    for(int i=0; i<n+surplus; i++)
    {
        if(nbv[i] >= n) continue;
        sol_array[nbv[i]] = 0;
    }
    for(int i=0; i<m; i++)
    {
        if(bv[i] >= n) continue;
        sol_array[bv[i]] = table[i][n+surplus];
    }
    for(int i=0; i<n; i++)
        printf("x%d = %f, ", i+1, sol_array[i]);
    if (table[m][n+surplus]<0)
        printf("\nOptimal value is: %f\n\n", -1.0*table[m][n+surplus]);
    else
        printf("\nOptimal value is: %f\n\n", table[m][n+surplus]);
}

int is_nearly_equal(float a, float b, float abs_err_th)
{
    int flag = 0;
    float err = fabs(a-b);
    if(err <= abs_err_th)
        flag = 1;
    return flag;
}

int get_pivot_h(float table[][100], int m, int n)
{
    float min = 0;
    int pivot_h = -1;
    for(int i=0; i<n; i++)
    {
```

```
        float value = table[m][i];
        if(value > 0 || is_nearly_equal(value, 0, pow(2, -6))) continue;
        if(value < min || pivot_h == -1)
        {
            pivot_h = i;
            min = table[m][i];
        }
    }
    return pivot_h;
}

int get_pivot_v(float table[][100], int pivot_h, int m, int n)
{
    float min = 0;
    int pivot_v = -1;
    for(int i=0; i<m; i++)
    {
        float ai = table[i][pivot_h];
        float ratio = table[i][n]/ai;
        if(ai < 0 || is_nearly_equal(ai, 0, pow(2, -6))) continue;
        if(ratio < min || pivot_v == -1)
        {
            pivot_v = i;
            min = ratio;
        }
    }
    return pivot_v;
}

void next_iteration(float table[][100], int bv[], int nbv[], int pivot_h, int pivot_v, int m, int n)
{
    float new_table[100][100];
    float pivot = table[pivot_v][pivot_h];

    int temp = bv[pivot_v];
    bv[pivot_v] = nbv[pivot_h];
    nbv[pivot_h] = temp;

    new_table[pivot_v][pivot_h] = 1.0/pivot;

    for(int i=0; i<=n; i++)
    {
        if(i == pivot_h) continue;
        new_table[pivot_v][i] = table[pivot_v][i]/pivot;
    }

    for(int i=0; i<=m; i++)
    {
        if(i == pivot_v) continue;
        new_table[i][pivot_h] = -table[i][pivot_h]/pivot;
    }
```

```
    for(int i=0; i<=m; i++)
    {
        for(int j=0; j<=n; j++)
        {
            if(i == pivot_v || j == pivot_h) continue;
            float p = pivot, s = table[i][j];
            float q = table[i][pivot_h], r = table[pivot_v][j];
            new_table[i][j] = (p*s - q*r)/p;
        }
    }

    copy_2d_array(new_table, table, m+1, n+1);
}

void simplex(float a[][100], float b[], float c[], int ies[], int m, int n, int query_iteration, int choice)
{
    float table[100][100];
    int bv[100], nbv[100];

    int iterations = 1, status = -1;
    float ratio = 0;

    initialize_table(table, a, b, c, ies, m, n);

    int surplus = get_surplus(ies, m);

    for(int i=0; i<m; i++)
        bv[i] = n+surplus+i;
    for(int i=0; i<n + surplus; i++)
        nbv[i] = i;
    int flag=1;
    while(flag)
    {
        int pos=0, neg=0, zero=0;
        int pivot_h, pivot_v;
        for(int i=0; i<n+surplus; i++)
        {
            if(is_nearly_equal(table[m][i], 0, pow(2, -6))) zero++;
            else if(table[m][i] < 0) neg++;
            else pos++;
        }
        if(neg == 0)
        {
            for(int j=0; j<m; j++)
            {
                int index = bv[j] - n - surplus;
                if(index >= 0 && ies[index] != -1)
                {
                    status = 3;
                    flag = 0;
                    break;
                }
```

```
    }
    if(flag != 0)
    {
        if(zero > 0)
        {
            status = 1;
            flag = 0;
        }
        else
        {
            status = 0;
            flag = 0;
        }
    }
}
else
{
    pivot_h = get_pivot_h(table, m, n+surplus);
    if(pivot_h < 0)
    {
        printf("Error occurred!\n");
        exit(-1);
    }
    int pos_col=0, neg_col=0, zero_col=0;
    for(int i=0; i<m; i++)
    {
        if(is_nearly_equal(table[i][pivot_h], 0, pow(2, -6))) zero_col++;
        else if(table[i][pivot_h] < 0) neg_col++;
        else pos_col++;
    }
    if(pos_col == 0)
    {
        status = 2;
        flag = 0;
        continue;
    }
    pivot_v = get_pivot_v(table, pivot_h, m, n+surplus);
    if(pivot_v < 0)
    {
        printf("Error occurred!\n");
        exit(-1);
    }
    if(query_iteration == 1)
        flag = 0;
    else
    {
        next_iteration(table, bv, nbv, pivot_h, pivot_v, m, n+surplus);
        iterations++;
    }
}
if(query_iteration == iterations || flag == 0)
{
```

```
            flag = 0;
            if (choice==1)
               printf("Not yet supported\n");
            else if (choice==2)
               printf("%d\n", iterations);
            else if (choice==3)
            {
               for(int i=0; i<n; i++)
                  printf("X%d, ", nbv[i]+1);
               printf("\n%f\n", table[m][n+surplus]);
            }
            else if (choice==4)
            {
               for(int i=0; i<m; i++)
                  printf("X%d, ", bv[i]+1);
               float min_ratio = table[pivot_v][n+surplus] / table[pivot_v][pivot_h];
               printf("min_ratio = %f\n", min_ratio);
            }
            else if (choice==5)
               print_simplex_table(table, bv, nbv, m, n+surplus);
            else if (choice==6)
            {
               if(status == 0)
               {
                  print_solution(table, bv, nbv, m, n, surplus);
                  printf("Unique solution\n");
               }
               else if(status == 1)
               {
                  print_solution(table, bv, nbv, m, n, surplus);
                  printf("Alternate solution exists\n");
               }
               else if(status == 2)
               {
                  printf("Unbounded Solution detected\n");
                  printf("column %d has all elements 0 or negative\n", pivot_h+1);
               }
               else
                  printf("Problem is Infeasible\n");
            }
         }
      }
   }


void main()
{
   int m,n;
   float c[100], b[100], a[100][100];
   int ies[100];

   printf("Enter number of equations (n) : ");
```

```
    scanf("%d", &m);
    printf("Enter number of variables (m) : ");
    scanf("%d", &n);
    printf("\nEnter the constraint equation:\n");
    for(int i=0; i<m; i++)
    {
        printf("(Coefficients of constraint equation - %d)\n",i+1);
        for(int j=0; j<n; j++)
        {
            printf("Input for matrix A's equation %d coefficient of x%d = ", i+1,j+1);
            scanf("%f", &a[i][j]);
        }
        printf("\nType of equation \n -1 : Ax<=B \n 0 : Ax=B \n 1 : Ax>=B \n Enter your option : ");
        scanf("%d", &ies[i]);
        printf("Input for matrix B's constant for equation %d : ", i+1);
        scanf("%f", &b[i]);
        printf("\n");
    }

    printf("\nEnter the coefficients of the maximizing equation (enter -ve coefficients for minimising
function) :\n");
    for(int i=0; i<n; i++)
    {
        printf("\nInput for objective function's coefficient of x%d : ",i+1);
        scanf("%f", &c[i]);
    }


    int flag = 1;
    int I;
    while(flag)
    {
        printf("\n\n(1) List of all BFS\n(2) Number of Iterations to solve the problem\n(3) List of all
Non-basic variables along with net evaluations in ith iteration\n(4) List of Basic variables along
with min ratios in ith iteration\n(5) Simplex table of ith iteration\n(6) Optimal solution\n(7)
Quit\n\nEnter your choice: ");
        int ch;
        scanf("%d", &ch);
        if (ch==1)
            simplex(a, b, c, ies, m, n, -1, 1);
        else if (ch==2)
            simplex(a, b, c, ies, m, n, -1, 2);
        else if (ch==3)
        {
            printf("Enter i: ");
            scanf("%d", &I);
            simplex(a, b, c, ies, m, n, I, 3);
        }
        else if (ch==4)
        {
            printf("Enter i: ");
            scanf("%d", &I);
```

```
            simplex(a, b, c, ies, m, n, I, 4);
        }
        else if (ch==5)
        {
            printf("Enter i: ");
            scanf("%d", &I);
            simplex(a, b, c, ies, m, n, I, 5);
        }
        else if (ch==6)
            simplex(a, b, c, ies, m, n, -1, 6);
        else if (ch==7)
            flag = 0;
        else
        {
            printf("Invalid choice\n");
            flag = 0;
        }
    }
}
```

OUTPUT

```
Enter number of equations (n) : 2
Enter number of variables (m) : 3

Enter the constraint equation:
(Coefficients of constraint equation - 1)
Input for matrix A's equation 1 coefficient of x1 = 1
Input for matrix A's equation 1 coefficient of x2 = 1
Input for matrix A's equation 1 coefficient of x3 = 0

Type of equation
 -1 : Ax<=B
 0 : Ax=B
 1 : Ax>=B
 Enter your option : 1
Input for matrix B's constant for equation 1 : 2

(Coefficients of constraint equation - 2)
Input for matrix A's equation 2 coefficient of x1 = 2
Input for matrix A's equation 2 coefficient of x2 = 0
Input for matrix A's equation 2 coefficient of x3 = 1

Type of equation
 -1 : Ax<=B
 0 : Ax=B
 1 : Ax>=B
 Enter your option : -1
Input for matrix B's constant for equation 2 : 5


Enter the coefficients of the maximizing equation (enter -ve coefficients for minimising function) :
```

Input for objective function's coefficient of x1 : -4

Input for objective function's coefficient of x2 : -8

Input for objective function's coefficient of x3 : -3

(1) List of all BFS
(2) Number of Iterations to solve the problem
(3) List of all Non-basic variables along with net evaluations in ith iteration
(4) List of Basic variables along with min ratios in ith iteration
(5) Simplex table of ith iteration
(6) Optimal solution
(7) Quit

Enter your choice: 6

The solution is:
x1 = 2.000000, x2 = 0.000000, x3 = 0.000000,
Optimal value is: 8.000000

Unique solution

(1) List of all BFS
(2) Number of Iterations to solve the problem
(3) List of all Non-basic variables along with net evaluations in ith iteration
(4) List of Basic variables along with min ratios in ith iteration
(5) Simplex table of ith iteration
(6) Optimal solution
(7) Quit

Enter your choice: 5
Enter i: 1

```
        ------------------------------------------------
        |   X1   |   X2   |   X3   |   X4   |   Xb   |
---------------------------------------------------------------
   X5   |  +1.00 |  +1.00 |  +0.00 |  -1.00 |  +2.00 |
---------------------------------------------------------------
   X6   |  +2.00 |  +0.00 |  +1.00 |  +0.00 |  +5.00 |
---------------------------------------------------------------
        | -99996.00 | -99992.00 |  +3.00 | +100000.00 | -200000.00 |
        ------------------------------------------------
```

(1) List of all BFS
(2) Number of Iterations to solve the problem
(3) List of all Non-basic variables along with net evaluations in ith iteration
(4) List of Basic variables along with min ratios in ith iteration

(5) Simplex table of ith iteration
(6) Optimal solution
(7) Quit

Enter your choice: 3
Enter i: 1
X1, X2, X3,
-200000.000000

(1) List of all BFS
(2) Number of Iterations to solve the problem
(3) List of all Non-basic variables along with net evaluations in ith iteration
(4) List of Basic variables along with min ratios in ith iteration
(5) Simplex table of ith iteration
(6) Optimal solution
(7) Quit

Enter your choice: 2
2

(1) List of all BFS
(2) Number of Iterations to solve the problem
(3) List of all Non-basic variables along with net evaluations in ith iteration
(4) List of Basic variables along with min ratios in ith iteration
(5) Simplex table of ith iteration
(6) Optimal solution
(7) Quit

Enter your choice: 7