****

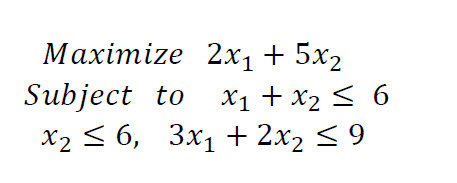
**LAB EXERCISE 10**

Name: Jayannthan P T

Dept: CSE ‘A’

Roll No.: 205001049



1. Solve the optimization function using graphical method
   1. 
   2. **Code:**

#include <stdio.h>

#include <conio.h>

#define INFINITY 999

#define N 3

#define M 6

**void** calctemp(**float** **\***temp, **float** a[N][M], **float** c[M], **int** basic[N])

{

**int** i, j;

    for (i = 0; i < M; i++)

    {

        temp[i] = 0;

        for (j = 0; j < N; j++)

            temp[i] = temp[i] + c[basic[j]] \* a[j][i];

        temp[i] = temp[i] - c[i];

    }

}

**void** displayframe(**float** c[M])

{

    printf("\t\tc[j]\t");

    printf("%g\t%g\t%g\t%g\t%g\t%g\n", c[0], c[1], c[2], c[3], c[4], c[5]);

    printf("\nc[B]\tB\tb\ta1\ta2\ta3\ta4\ta5\ta6\n");

}

**void** minimum(**float** **\***arr, **int** **\***arrminpos, **int** n)

{

**int** i;

**float** arrmin;

    arrmin = arr[0];

    \*arrminpos = 0;

    for (i = 0; i < n; i++)

        if (arr[i] < arrmin)

        {

            arrmin = arr[i];

            \*arrminpos = i;

        }

    printf("\n%d\n", \*arrminpos);

}

**void** display(**float** c[N], **float** b[N], **float** a[N][M], **int** basic[N])

{

**int** i, j;

    displayframe(c);

    for (i = 0; i < N; i++)

    {

        printf("\n%.4g\tX%d\t%.4g\t", c[basic[i]], basic[i] + 1, b[i]);

        for (j = 0; j < M; j++)

            printf("%.4g\t", a[i][j]);

        printf("\n");

    }

}

**int** main()

{

**float** temp[M] = {{0}, {0}, {0}, {0}, {0}, {0}};

**float** c[M] = {0};

**float** a[N][M] = {0};

**float** b[N] = {0};

**int** tempminpos; */\* Stores the minimum valued position of {Zj-Cj} i.e.*

*coming in variable \*/*

**float** miniratio[N]; */\* Stores the value of the ratio b[i]/a[i][j] \*/*

**int** miniratiominpos; */\* Stores the minimum valued position of b[i]/a[i][j]*

*i.e. going out variable \*/*

**float** key; */\* Stores the key element \*/*

**int** gooutcol; */\* Stores the column number which goes out \*/*

**float** z; */\* Stores the value of the objective function \*/*

**float** x[M]; */\* Stores the value of the variables \*/*

**int** i, j; */\* Loop variables \*/*

**int** basic[N]; */\* Stores the basic variable \*/*

**int** nonbasic[N]; */\* Stores the non-basic variable \*/*

**int** flag = 0; */\* Terminating variable \*/*

    for (i = 0; i < N; i++)

    {

        basic[i] = (i + N);

        nonbasic[i] = i;

    }

    printf("\nMax z = c1x1 + c2x2 + c3x3\n");

    printf("\na11x1 + a12x2 + a13x3 <= b1\n");

    printf("\na21x1 + a22x2 + a23x3 <= b2\n");

    printf("\na31x1 + a31x2 + a32x3 <= b3\n");

    printf("\nEnter values of ci's\n");

    for (i = 0; i < N; i++)

    {

        printf("\nEnter c[%d]\t", i + 1);

        scanf("%f", &c[i]);

    }

    printf("\nEnter values of ai's\n");

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

        {

            printf("\nEnter a[%d][%d]\t", i + 1, j + 1);

            scanf("%f", &a[i][j]);

        }

    }

    printf("\nEnter values of bi's\n");

    for (i = 0; i < N; i++)

    {

        printf("\nEnter b[%d]\t", i + 1);

        scanf("%f", &b[i]);

    }

    while (flag == 0)

    {

        z = 0;

        calctemp(temp, a, c, basic);

        printf("\n");

        minimum(temp, &tempminpos, M);

        display(c, b, a, basic);

        printf("\nZj-Cj\t\t\t");

        for (i = 0; i < M; i++)

            printf("%.4g\t", temp[i]);

        printf("\n\n");

        for (i = 0; i < N; i++)

        {

            x[basic[i]] = b[i];

            x[nonbasic[i]] = 0;

            printf("x[%d]=%g\n", basic[i] + 1, b[i]);

        }

        for (i = 0; i < N; i++)

            z = z + c[i] \* x[i];

        printf("Max(z) = %g", z);

        for (i = 0; i < N; i++)

        {

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

            {

                miniratio[i] = INFINITY;

                continue;

            }

            if (a[i][tempminpos] < 0)

            {

                miniratio[i] = INFINITY;

                continue;

            }

            miniratio[i] = b[i] / a[i][tempminpos];

        }

        minimum(miniratio, &miniratiominpos, N);

        for (i = 0; i < N; i++)

            if (miniratiominpos == i)

                gooutcol = basic[i];

        printf("\nComing in variable = X%d\t", tempminpos + 1);

        printf("Going out variable = X%d\n", gooutcol + 1);

        basic[miniratiominpos] = tempminpos;

        nonbasic[tempminpos] = gooutcol;

        key = a[miniratiominpos][tempminpos];

        b[miniratiominpos] = b[miniratiominpos] / key;

        for (i = 0; i < M; i++)

            a[miniratiominpos][i] = a[miniratiominpos][i] / key;

        for (i = 0; i < N; i++)

        {

            if (miniratiominpos == i)

                continue;

            key = a[i][tempminpos];

            for (j = 0; j < M; j++)

            {

                a[i][j] = a[i][j] - a[miniratiominpos][j] \* key;

            }

            b[i] = b[i] - b[miniratiominpos] \* key;

        }

        getch();

        for (i = 0; i < M; i++)

        {

            flag = 1;

            if (temp[i] < 0)

            {

                flag = 0;

                break;

            }

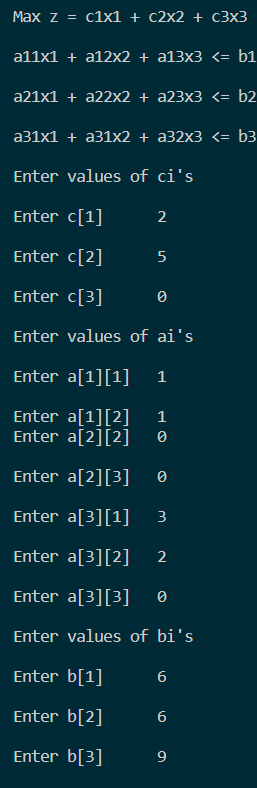
        }

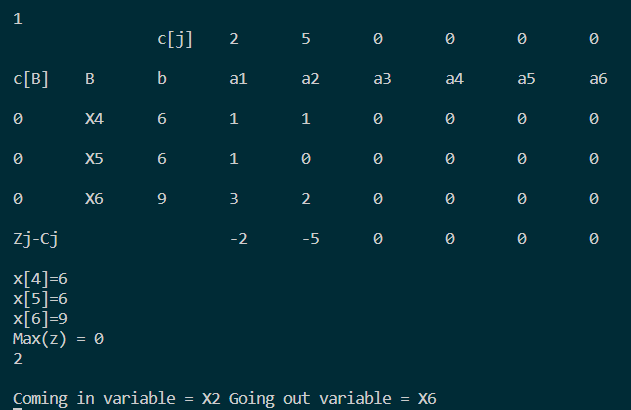
    }

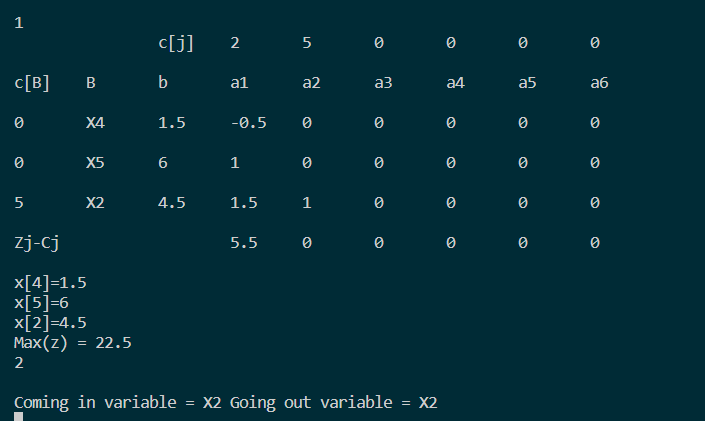
    return 0;

}

* 1. **Output:**

****

****

****



1. Implement 8 Queens backtracking algorithm
   1. **Code:**

#include <iostream>

using **namespace** std;

#define N 8

**void** boardPrint(**int** board[N][N])

{

    for (**int** i = 0; i < N; i++)

    {

        for (**int** j = 0; j < N; j++)

        {

            cout << board[i][j] << " ";

        }

        cout << endl;

    }

}

**bool** isValid(**int** board[N][N], **int** row, **int** col)

{

    for (**int** i = 0; i < col; i++)

        if (board[row][i])

        {

            return false;

        }

    for (**int** i = row, j = col; i >= 0 && j >= 0; i--, j--)

    {

        if (board[i][j])

        {

            return false;

        }

    }

    for (**int** i = row, j = col; j >= 0 && i < N; i++, j--)

    {

        if (board[i][j])

        {

            return false;

        }

    }

    return true;

}

**bool** solveNQn(**int** board[N][N], **int** col)

{

    if (col >= N)

    {

        return true;

    }

    for (**int** i = 0; i < N; i++)

    {

        if (isValid(board, i, col))

        {

            board[i][col] = 1;

            if (solveNQn(board, col + 1))

            {

                return true;

            }

            board[i][col] = 0;

        }

    }

    return false;

}

**bool** Solutions()

{

**int** board[N][N];

    for (**int** i = 0; i < N; i++)

        for (**int** j = 0; j < N; j++)

        {

            board[i][j] = 0;

        }

    if (solveNQn(board, 0) == false)

    {

        cout << "Solution does not exist";

        return false;

    }

    boardPrint(board);

    return true;

}

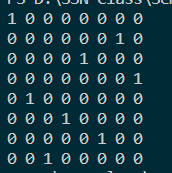
**int** main()

{

    Solutions();

}

* 1. **Output:**

****



1. To Implement Floyd’s Algorithm for all pair shortest path using DP
   1. **Code:**

#include <stdio.h>

#include <conio.h>

#define INFINITY 999

#define N 3

#define M 6

**void** calctemp(**float** **\***temp, **float** a[N][M], **float** c[M], **int** basic[N])

{

**int** i, j;

    for (i = 0; i < M; i++)

    {

        temp[i] = 0;

        for (j = 0; j < N; j++)

            temp[i] = temp[i] + c[basic[j]] \* a[j][i];

        temp[i] = temp[i] - c[i];

    }

}

**void** displayframe(**float** c[M])

{

    printf("\t\tc[j]\t");

    printf("%g\t%g\t%g\t%g\t%g\t%g\n", c[0], c[1], c[2], c[3], c[4], c[5]);

    printf("\nc[B]\tB\tb\ta1\ta2\ta3\ta4\ta5\ta6\n");

}

**void** minimum(**float** **\***arr, **int** **\***arrminpos, **int** n)

{

**int** i;

**float** arrmin;

    arrmin = arr[0];

    \*arrminpos = 0;

    for (i = 0; i < n; i++)

        if (arr[i] < arrmin)

        {

            arrmin = arr[i];

            \*arrminpos = i;

        }

    printf("\n%d\n", \*arrminpos);

}

**void** display(**float** c[N], **float** b[N], **float** a[N][M], **int** basic[N])

{

**int** i, j;

    displayframe(c);

    for (i = 0; i < N; i++)

    {

        printf("\n%.4g\tX%d\t%.4g\t", c[basic[i]], basic[i] + 1, b[i]);

        for (j = 0; j < M; j++)

            printf("%.4g\t", a[i][j]);

        printf("\n");

    }

}

**int** main()

{

**float** temp[M] = {{0}, {0}, {0}, {0}, {0}, {0}};

**float** c[M] = {0};

**float** a[N][M] = {0};

**float** b[N] = {0};

**int** tempminpos; */\* Stores the minimum valued position of {Zj-Cj} i.e.*

*coming in variable \*/*

**float** miniratio[N]; */\* Stores the value of the ratio b[i]/a[i][j] \*/*

**int** miniratiominpos; */\* Stores the minimum valued position of b[i]/a[i][j]*

*i.e. going out variable \*/*

**float** key; */\* Stores the key element \*/*

**int** gooutcol; */\* Stores the column number which goes out \*/*

**float** z; */\* Stores the value of the objective function \*/*

**float** x[M]; */\* Stores the value of the variables \*/*

**int** i, j; */\* Loop variables \*/*

**int** basic[N]; */\* Stores the basic variable \*/*

**int** nonbasic[N]; */\* Stores the non-basic variable \*/*

**int** flag = 0; */\* Terminating variable \*/*

    for (i = 0; i < N; i++)

    {

        basic[i] = (i + N);

        nonbasic[i] = i;

    }

    printf("\nMax z = c1x1 + c2x2 + c3x3\n");

    printf("\na11x1 + a12x2 + a13x3 <= b1\n");

    printf("\na21x1 + a22x2 + a23x3 <= b2\n");

    printf("\na31x1 + a31x2 + a32x3 <= b3\n");

    printf("\nEnter values of ci's\n");

    for (i = 0; i < N; i++)

    {

        printf("\nEnter c[%d]\t", i + 1);

        scanf("%f", &c[i]);

    }

    printf("\nEnter values of ai's\n");

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

        {

            printf("\nEnter a[%d][%d]\t", i + 1, j + 1);

            scanf("%f", &a[i][j]);

        }

    }

    printf("\nEnter values of bi's\n");

    for (i = 0; i < N; i++)

    {

        printf("\nEnter b[%d]\t", i + 1);

        scanf("%f", &b[i]);

    }

    while (flag == 0)

    {

        z = 0;

        calctemp(temp, a, c, basic);

        printf("\n");

        minimum(temp, &tempminpos, M);

        display(c, b, a, basic);

        printf("\nZj-Cj\t\t\t");

        for (i = 0; i < M; i++)

            printf("%.4g\t", temp[i]);

        printf("\n\n");

        for (i = 0; i < N; i++)

        {

            x[basic[i]] = b[i];

            x[nonbasic[i]] = 0;

            printf("x[%d]=%g\n", basic[i] + 1, b[i]);

        }

        for (i = 0; i < N; i++)

            z = z + c[i] \* x[i];

        printf("Max(z) = %g", z);

        for (i = 0; i < N; i++)

        {

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

            {

                miniratio[i] = INFINITY;

                continue;

            }

            if (a[i][tempminpos] < 0)

            {

                miniratio[i] = INFINITY;

                continue;

            }

            miniratio[i] = b[i] / a[i][tempminpos];

        }

        minimum(miniratio, &miniratiominpos, N);

        for (i = 0; i < N; i++)

            if (miniratiominpos == i)

                gooutcol = basic[i];

        printf("\nComing in variable = X%d\t", tempminpos + 1);

        printf("Going out variable = X%d\n", gooutcol + 1);

        basic[miniratiominpos] = tempminpos;

        nonbasic[tempminpos] = gooutcol;

        key = a[miniratiominpos][tempminpos];

        b[miniratiominpos] = b[miniratiominpos] / key;

        for (i = 0; i < M; i++)

            a[miniratiominpos][i] = a[miniratiominpos][i] / key;

        for (i = 0; i < N; i++)

        {

            if (miniratiominpos == i)

                continue;

            key = a[i][tempminpos];

            for (j = 0; j < M; j++)

            {

                a[i][j] = a[i][j] - a[miniratiominpos][j] \* key;

            }

            b[i] = b[i] - b[miniratiominpos] \* key;

        }

        getch();

        for (i = 0; i < M; i++)

        {

            flag = 1;

            if (temp[i] < 0)

            {

                flag = 0;

                break;

            }

        }

    }

    return 0;

}

* 1. **Output:**

