Branch & Bound method

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

int a[10][10],visited[10],n,cost=0;

void get()
{
  int i,j;
  printf("\n\nEnter Number of Cities: ");
  scanf("%d",&n);
  printf("\nEnter Cost Matrix: \n");
  for( i=0;i<n;i++)
  {
    printf("\n Enter Elements of Row # : %d\n",i+1);
    for( j=0;j<n;j++)
    scanf("%d",&a[i][j]);
    visited[i]=0;
  }

  printf("\n\nThe Cost Matrix is:\n");
  for( i=0;i<n;i++)
  {
    printf("\n\n");
    for( j=0;j<n;j++)
      printf("\t%d",a[i][j]);
  }
}

void mincost(int city)
{
  int i,ncity;
  visited[city]=1;
  printf("%d ===> ",city+1);
  ncity=least(city);

  if(ncity==999)
  {
    ncity=0;
    printf("%d",ncity+1);
    cost+=a[city][ncity];
    return;
  }

  mincost(ncity);
}

int least(int c)
{
  int i,nc=999;
  int min=999,kmin;
  for(i=0;i<n;i++)
  {
    if((a[c][i]!=0)&&(visited[i]==0))
    {
      if(a[c][i]<min)
      {
        min=a[i][0]+a[c][i];
        kmin=a[c][i];
        nc=i;
      }
    }
  }

  if(min!=999)
    cost+=kmin;
  return nc;
}

void put()
{
  printf("\n\nMinimum cost:");
  printf("%d",cost);
}

int main()
{
  get();
  printf("\n\nThe Path is:\n\n");
  mincost(0);
  put();
  getch();
}
```

Cutting-plane method

```
#include <iostream>
#include <cmath>
using namespace std;

int d[10]={0};
int dmap[10]={-1};
float mat[10][10], b[10], temp[10][10], constants[10];
float ans[10][10], z[10];
int R, C;

int duald[10]={0};
int dualdmap[10]={-1};
float dualmat[10][10], dualb[10], dualtemp[10][10], dualconstants[10];
float dualans[10][10], dualz[10];
int dualR, dualC;
int main()
{
    int in_var, eqn, var;
    int flag = 0;int incos=0;
    string inequality;
    cout << "Enter no of variables\n";
    cin >> in_var;
    var= in_var;
    cout << "Enter no. of equations\n";
    cin >> eqn;
    for(int i = 0 ; i < eqn ; i++)
    {
        cout << "Enter coefficients and constant term of equation no " << i+1 << " seperated by spaces:\n";
        for(int j = 0 ; j <= in_var ; j++)
        {
            cin >> mat[i][j];
        }

    }

    for(int i=0;i < eqn;i++)
    {
        for(int j=0;j<=in_var;j++)
        {
            temp[i][j] = mat[i][j];
        }
    }

    cout << "Enter the objective function to be maximised\n";
    for(int i=0;i<=in_var;i++)
    {
        int a=0;
        cin >> a;
        if(a>0)flag=1;
        temp[eqn][i] = -a;
    }

    cout << "\nInitial simplex table\n ";
    for(int i=0;i <= eqn;i++)
    {
        for(int j=0;j<=in_var;j++)
        {
            cout << " " << temp[i][j];
        }
        cout << endl;
    }
    int dummy=1;
    cout << "\n Flag is " << flag;
    while(flag){

        float min = 10000; int prod=0;
        int minpos = -1;
        for(int i=0;i<in_var;i++)
        {
            if(temp[eqn][i] < 0)prod=1;
        }
        if(!prod){
            cout << "Prod is " << prod;
            flag=0;
            break;
        }

        for(int i=0;i<in_var;i++)
        {
            if(temp[eqn][i] <= min)
            {
                min = temp[eqn][i];
                minpos = i;
            }
        }

        cout << "\n Value of minpos is " << minpos << "  and incoming variable is  " << in_var << endl;
        int pivot = -1;
        float minp = 10000;
        for(int i=0;i< eqn;i++)
        {
            if(1.0*temp[i][in_var]/temp[i][minpos] < minp && 1.0*temp[i][in_var]/temp[i][minpos]>0)
            {
                minp = 1.0*temp[i][in_var]/temp[i][minpos];
                pivot = i;
            }
        }

        if(pivot==-1){
            flag=0;
            incos = 1;
            break;
        }
        cout << "\n Pivot : " << temp[pivot][minpos] << " at pos " << pivot;
        cout << "\n Most negative element " << min;

        d[minpos] = 1;
        dmap[minpos] = pivot;
        float p = temp[pivot][minpos];

        for(int i=0;i<=eqn;i++)
        {
            for(int j=0;j<=in_var;j++)
            {
                if(i==pivot || j==minpos)continue;
                temp[i][j] = temp[i][j] - 1.0*temp[pivot][j]*temp[i][minpos]/p;
            }
        }

        for(int i=0;i<=eqn; i++)
        {
            if(i==pivot)continue;
            temp[i][minpos] = -temp[i][minpos]/p;
        }
        for(int j=0;j<=in_var;j++)
        {
            if(j==minpos)continue;
            temp[pivot][j] = temp[pivot][j]/p;
        }
        temp[pivot][minpos] = 1.0/temp[pivot][minpos];



        cout << "\n Simplex table\n ";
        for(int i=0;i <= eqn;i++)
        {
            for(int j=0;j<=in_var;j++)
            {
                cout <<temp[i][j] << " ";
            }
            cout << endl;
        }
    }

    if(incos)
    {
        cout << "\n\n The given system of equations inconsistent  ";
        return 0;
    }

    for(int j=0;j<in_var;j++)
    {
        if(temp[eqn][j]==0)
        {
            cout << "\n\n However infinite solutions exist for this case ";
            break;
        }
    }

    bool flagdual = true;
    while(flagdual){

        bool ifc=true;
        for(int i=0;i < eqn;i++)
        {
            if(fabs(floor(temp[i][in_var])-temp[i][in_var])>1e-5){
                ifc = false;break;
            }
```

```
                }
                if(ifc){flagdual=false;break;}
                float xbmax=-1; int xbpos;
                for(int i=0;i < eqn;i++)
                {
                        for(int j=0;j<=in_var;j++)
                        {
                                dualtemp[i][j] = temp[i][j];
                                if(j==in_var && fabs(floor(temp[i][j])-temp[i][j])>1e-5){
                                        if((temp[i][j]-floor(temp[i][j])) > xbmax){
                                                xbmax = temp[i][j]-floor(temp[i][j]);
                                                xbpos = i;
                                        }
                                }
                        }
                }
                cout << "\nvalue of xbmax: " << xbmax << endl;
                if((int)xbmax==-1 || fabs(xbmax-1)<1e-5){cout << "\n Integer solution already
calculated : ";return 0;}
                cout << "\n value of xbpos : " << xbpos;
                for(int j=0;j<=in_var;j++){
                        dualtemp[eqn][j] = floor(temp[xbpos][j]) - temp[xbpos][j];
                        dualtemp[eqn+1][j] = temp[eqn][j];
                }
                eqn = eqn +1;

                cout << "\nPrinting in-between table: \n";
                for(int i=0;i <= eqn;i++)
                {
                        for(int j=0;j<=in_var;j++)
                        {
                                cout << dualtemp[i][j] << " " ;
                        }
                        cout << endl;
                }

                float min = 10000;int minpos = -1;
                        int pivot = -1;

                        for(int i=0;i< eqn;i++)
                        {
                                if(dualtemp[i][in_var] < min){
                                        min = dualtemp[i][in_var];
                                        pivot = i;
                                }
                        }

                cout << "\n Value of pivot: " << pivot << endl;
                        float minp = 10000;
                for(int j=0;j<in_var;j++){
                        float f = 1.0*dualtemp[eqn][j]/dualtemp[pivot][j];
                        cout << "\n Value f : " << fabs(f) << " ";
```

```
            if(fabs(f) < minp){minp = fabs(f);minpos = j;cout << " \n Minpos selected as:
" << j << endl;}
        }
        float p = dualtemp[pivot][minpos];
        cout << "\n Value of minpos and p: " << minpos << " " << p << endl;
        for(int i=0;i<=eqn;i++)
            {
                for(int j=0;j<=in_var;j++)
                {
                    if(i==pivot || j==minpos)continue;
                    dualtemp[i][j] = dualtemp[i][j] - 1.0*dualtemp[pivot]
[j]*dualtemp[i][minpos]/p;
                }
            }

        for(int i=0;i<=eqn; i++)
            {
                if(i==pivot)continue;
                dualtemp[i][minpos] = -dualtemp[i][minpos]/p;
            }
            for(int j=0;j<=in_var;j++)
            {
                if(j==minpos)continue;
                dualtemp[pivot][j] = dualtemp[pivot][j]/p;
            }
            dualtemp[pivot][minpos] = 1.0/dualtemp[pivot][minpos];

        for(int i=0;i <= eqn;i++)
            {
                for(int j=0;j<=in_var;j++)
                {
                    cout <<dualtemp[i][j] << " ";
                    temp[i][j] = dualtemp[i][j];
                }
                cout << endl;
            }
    }
    return 0;
}
```

Output

```
Enter no of variables
2
Enter no. of equations
2
Enter coefficients and constant term of equation no 1 seperated by spaces:
6 5 25
Enter coefficients and constant term of equation no 2 seperated by spaces:
1 3 10
Enter the objective function to be maximised
2 3 0
```

```
Initial simplex table
  6 5 25
 1 3 10
 -2 -3 0

 Flag is 1
 Value of minpos is 1  and incoming variable is  2

 Pivot : 3 at pos 1
 Most negative element -3
 Simplex table
 4.33333 -1.66667 8.33333
0.333333 0.333333 3.33333
-1 1 10

 Value of minpos is 0  and incoming variable is  2

 Pivot : 4.33333 at pos 0
 Most negative element -1
 Simplex table
 0.230769 -0.384615 1.92308
-0.0769231 0.461538 2.69231
0.230769 0.615385 11.9231
Prod is 0
value of xbmax: 0.923077

 value of xbpos : 0
Printing in-between table:
0.230769 -0.384615 1.92308
-0.0769231 0.461538 2.69231
-0.230769 -0.615385 -0.923077
0.230769 0.615385 11.9231

 Value of pivot: 2

 Value f : 1
 Minpos selected as: 0

 Value f : 1
 Value of minpos and p: 0 -0.230769
1 -1 1
-0.333333 0.666667 3
-4.33333 2.66667 4
1 0 11

value of xbmax: 1
```