Hungarian.cpp

```cpp
//
// Created by juan on 15/04/17.
//

#define N 25 //max number of vertices in one part
#define INF 100000000 //just infinity

#include <iostream>
#include <string.h>
#include <iomanip>

using namespace std;

int cost[N][N]; //cost matrix
int n, max_match; //n workers and n jobs
int lx[N], ly[N]; //labels of X and Y parts
int xy[N]; //xy[x] - vertex that is matched with x,
int yx[N]; //yx[y] - vertex that is matched with y
bool S[N], T[N]; //sets S and T in algorithm
int slack[N]; //as in the algorithm description
int slackx[N]; //slackx[y] such a vertex, that l(slackx[y]) + l(y) - w(slackx[y],y) = slack[y]
int previo[N]; //array for memorizing alternating paths


void init_labels() {
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
            lx[x] = max(lx[x], cost[x][y]);
}

void update_labels() {
    int x, y, delta = INF; //init delta as infinity
    for (y = 0; y < n; y++) //calculate delta using slack
        if (!T[y])
            delta = min(delta, slack[y]);
    for (x = 0; x < n; x++) //update X labels
        if (S[x]) lx[x] -= delta;
    for (y = 0; y < n; y++) //update Y labels
        if (T[y]) ly[y] += delta;
    for (y = 0; y < n; y++) //update slack array
        if (!T[y])
            slack[y] -= delta;
}

void add_to_tree(int x, int prevx)
//x - current vertex,prevx - vertex from X before x in the alternating path,
//so we add edges (prevx, xy[x]), (xy[x], x)
{
    S[x] = true; //add x to S
    previo[x] = prevx; //we need this when augmenting
    for (int y = 0; y < n; y++) //update slacks, because we add new vertex to S
        if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
            slack[y] = lx[x] + ly[y] - cost[x][y];
            slackx[y] = x;
        }
}

void augment() //main function of the algorithm
{
    if (max_match == n) return; //check wether matching is already perfect
    int x, y, root = 0; //just counters and root vertex
```

```
    int q[N], wr = 0, rd = 0; //q - queue for bfs, wr,rd - write and read
    //pos in queue
    memset(S, false, sizeof(S)); //init set S
    memset(T, false, sizeof(T)); //init set T
    memset(previo, -1, sizeof(previo)); //init set previo - for the alternating tree
    for (x = 0; x < n; x++) //finding root of the tree
        if (xy[x] == -1) {
            q[wr++] = root = x;
            previo[x] = -2;
            S[x] = true;
            break;
        }

    for (y = 0; y < n; y++) //initializing slack array
    {
        slack[y] = lx[root] + ly[y] - cost[root][y];
        slackx[y] = root;
    }

    //second part of augment() function
    while (true) //main cycle
    {
        while (rd < wr) //building tree with bfs cycle
        {
            x = q[rd++]; //current vertex from X part
            for (y = 0; y < n; y++) //iterate through all edges in equality graph
                if (cost[x][y] == lx[x] + ly[y] && !T[y]) {
                    if (yx[y] == -1) break; //an exposed vertex in Y found, so
                    //augmenting path exists!
                    T[y] = true; //else just add y to T,
                    q[wr++] = yx[y]; //add vertex yx[y], which is matched
                    //with y, to the queue
                    add_to_tree(yx[y], x); //add edges (x,y) and (y,yx[y]) to the tree
                }
            if (y < n) break; //augmenting path found!
        }
        if (y < n) break; //augmenting path found!

        update_labels(); //augmenting path not found, so improve labeling
        wr = rd = 0;
        for (y = 0; y < n; y++)
            //in this cycle we add edges that were added to the equality graph as a
            //result of improving the labeling, we add edge (slackx[y], y) to the tree if
            //and only if !T[y] && slack[y] == 0, also with this edge we add another one
            //(y, yx[y]) or augment the matching, if y was exposed
            if (!T[y] && slack[y] == 0)
            {
                if (yx[y] == -1) //exposed vertex in Y found - augmenting path exists!
                {
                    x = slackx[y];
                    break;
                }
                else
                {
                    T[y] = true; //else just add y to T,
                    if (!S[yx[y]])
                    {
                        q[wr++] = yx[y]; //add vertex yx[y], which is matched with
                        //y, to the queue
                        add_to_tree(yx[y], slackx[y]); //and add edges (x,y) and (y,
                        //yx[y]) to the tree
                    }
                }
            }
        if (y < n) break; //augmenting path found!
    }
```

```
    if (y < n) //we found augmenting path!
    {
        max_match++; //increment matching
        //in this cycle we inverse edges along augmenting path
        for (int cx = x, cy = y, ty; cx != -2; cx = previo[cx], cy = ty)
        {
            ty = xy[cx];
            yx[cy] = cx;
            xy[cx] = cy;
        }
        augment(); //recall function, go to step 1 of the algorithm
    }
}//end of augment() function

double hungarian() {
    double ret = 1; //weight of the optimal matching
    max_match = 0; //number of vertices in current matching
    memset(xy, -1, sizeof(xy));
    memset(yx, -1, sizeof(yx));
    init_labels(); //step 0
    augment(); //steps 1-3
    for (int x = 0; x < n; x++) //forming answer there
        ret *= 1.0*cost[x][xy[x]]/100;
    return ret;
}

int main(){
    cin.tie(0);
    ios_base::sync_with_stdio(0);
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> cost[i][j];
    double ans = hungarian() * 100;
    cout << fixed << setprecision(6) << ans;
}
```