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

struct Edge{
    int v1;
    int v2;
    int cost;
    struct Edge *next;
};
typedef struct Edge Edge;

typedef struct{
    int cnt;
    int *bitmap;
    int nearcnt;
    Edge *nearHeap;
}Tree;

void pushHeap(Edge heap[], Edge item, int *heapcnt);
Edge *popHeap(Edge heap[], int *heapcnt);

Edge *Sollin(Tree *, int);
void dfs(int s, Edge *mst, Tree* forest);
int* order;
int numNode;

int main(void)
{
    int i, j, c;

    scanf("%d", &numNode);

    int *matrix = malloc(numNode * numNode * sizeof(int));
    order = malloc(numNode * numNode * sizeof(int));
    Tree *forest = malloc(numNode * sizeof(Tree));

    for(i=0; i<numNode*numNode; i++)
        matrix[i] = 0;

    while(scanf("%d", &i) != EOF)
    {
        scanf("%d", &j);
        scanf("%d", &c);
        matrix[i*numNode+j] = matrix[j*numNode+i] = c;
        order[i*numNode+j] = order[j*numNode+i] = i;
    }
    Edge item;
    for(i=0; i<numNode; i++)
    {
        forest[i].cnt = 1;
        forest[i].bitmap = malloc(numNode * sizeof(int));
        for(j=0; j<numNode; j++)
            forest[i].bitmap[j] = 0;
        forest[i].bitmap[i] = 1;

        forest[i].nearcnt = 0;
        forest[i].nearHeap = malloc((numNode) * sizeof(Edge)); //at most numNode-cnt+1
```

```
        elements (from 1)
        for(j=0; j<numNode; j++)
            if(matrix[i*numNode+j] != 0)
            {
                item.v1 = i;
                item.v2 = j;
                item.cost = matrix[i*numNode+j];
                pushHeap(forest[i].nearHeap, item, &(forest[i].nearcnt));
            }
    }
    free(matrix);

    Edge *mst = Sollin(forest, numNode);
    for(i=0; i<numNode; i++)
    {
        if(forest[i].cnt != 0)
        {
            forest[i].cnt=0;
            free(forest[i].bitmap);
            free(forest[i].nearHeap);
        }
    }
    dfs(0, mst, forest);    //In DFS, consider "forest" as "visited" array

    printf("\n");
    free(forest);
    return 0;
}

void pushHeap(Edge heap[], Edge item, int *heapcnt)
{
    int i = ++(*heapcnt);
    while(i!=1 && item.cost < heap[i/2].cost)
    {
        heap[i] = heap[i/2];
        i/=2;
    }
    heap[i] = item;
}

Edge *popHeap(Edge heap[], int *heapcnt)
{
    int parent, child;
    Edge *item, temp;
    item = malloc(sizeof(Edge));

    *item = heap[1];
    temp = heap[(*heapcnt)--];
    parent = 1;
    child = 2;
    while(child <= *heapcnt)
    {
        if(child < *heapcnt && heap[child].cost > heap[child+1].cost)
            child++;
        if(temp.cost <= heap[child].cost)   break;
        heap[parent] = heap[child];
        parent = child;
```

```
        child *= 2;
    }
    heap[parent] = temp;
    return item;
}

Edge * Sollin(Tree * forest, int nodecnt)
{
    int i, j, k, heapcnt, v1Tree, v2Tree;
    Edge *min=NULL, *mst=NULL, *stage=NULL, *tmp=NULL;
    int v1[200], v2[200], cost[200], printed[200], cnt = 0, iteration = 1;
    while(forest[0].cnt != nodecnt)
    {
        for(i=0; i<nodecnt; i++)
            if(forest[i].cnt != 0)
            {
                min = popHeap(forest[i].nearHeap, &(forest[i].nearcnt));
                if(!stage)
                    stage = min;
                if(!mst)
                {
                    mst = min;
                    mst->next = NULL;
                }
                else
                {
                    tmp = mst;
                    while(1)
                    {
                        if(tmp->v1 == min->v2 && tmp->v2 == min->v1)    break;
                        if(!tmp->next)
                        {
                            tmp->next = min;
                            tmp = tmp->next;
                            tmp->next = NULL;
                            break;
                        }
                        tmp = tmp->next;
                    }
                }
            }

        while(stage)
        {
            v1[cnt] = stage->v1;
            v2[cnt] = stage->v2;
            cost[cnt] = stage->cost;
            printed[cnt] = 0;
            cnt++;
            v1Tree = v2Tree = -1;
            for(i=0; v1Tree==-1 || v2Tree==-1; i++)
            {
                if(forest[i].bitmap[stage->v1])
                    v1Tree = i;
                if(forest[i].bitmap[stage->v2])
                    v2Tree = i;
            }
```

```
        if(v1Tree > v2Tree)
        {
            j = v1Tree;
            v1Tree = v2Tree;
            v2Tree = j;
        }
        forest[v1Tree].cnt += forest[v2Tree].cnt;
        forest[v2Tree].cnt = 0;
        heapcnt = 0;
        tmp = malloc((nodecnt-forest[v1Tree].cnt+1)*sizeof(Edge));

        for(i=0; i<nodecnt; i++)
        {
            forest[v1Tree].bitmap[i] = (forest[v1Tree].bitmap[i] | forest[v2Tree].bitmap[
            i]);
            forest[v2Tree].bitmap[i] = 0;

            if(!forest[v1Tree].bitmap[i])
            {
                for(j=1; j<= forest[v1Tree].nearcnt; j++)
                    if(i == forest[v1Tree].nearHeap[j].v2)
                        break;
                for(k=1; k<= forest[v2Tree].nearcnt; k++)
                    if(i == forest[v2Tree].nearHeap[k].v2)
                        break;

                if(k <= forest[v2Tree].nearcnt && j <= forest[v1Tree].nearcnt)
                    if(forest[v2Tree].nearHeap[k].cost <= forest[v1Tree].nearHeap[j].cost)
                        pushHeap(tmp, forest[v2Tree].nearHeap[k], &heapcnt);
                    else
                        pushHeap(tmp, forest[v1Tree].nearHeap[j], &heapcnt);
                else if(k <= forest[v2Tree].nearcnt)
                    pushHeap(tmp, forest[v2Tree].nearHeap[k], &heapcnt);
                else if(j <= forest[v1Tree].nearcnt)
                    pushHeap(tmp, forest[v1Tree].nearHeap[j], &heapcnt);
            }
        }

        free(forest[v1Tree].nearHeap);
        free(forest[v2Tree].nearHeap);

        forest[v1Tree].nearcnt = heapcnt;
        forest[v1Tree].nearHeap = tmp;

        stage = stage->next;
    }
    printf("iteration %d: ", iteration);
    for(j = 0; j < cnt; j++){
        int min, index;
        for(i = 0; i < cnt; i++){
            if(printed[i] == 0){
                min = cost[i], index = i;
                break;
            }
        }
        for(i = 0; i < cnt; i++){
```

```
                if(printed[i] == 0)
                if(cost[i] < min){
                    min = cost[i];
                    index = i;
                }
            }
            if(order[v1[index]*numNode+v2[index]] == v1[index])
                printf("(%d,%d,%d)", v1[index], v2[index], cost[index]);
            else
                printf("(%d,%d,%d)", v2[index], v1[index], cost[index]);
            printed[index] = 1;
        }
        iteration++;
        cnt = 0;
        printf("\n");
    }


    return mst;
}
void dfs(int s, Edge *mst, Tree *forest)
{
    printf("%d ", s);
    forest[s].cnt = 1;
    Edge *tmp = mst;
    for(;tmp; tmp=tmp->next)
    {

        if(tmp->v1 == s && forest[tmp->v2].cnt == 0)
            dfs(tmp->v2, mst, forest);
        if(tmp->v2 == s && forest[tmp->v1].cnt == 0)
            dfs(tmp->v1, mst, forest);
    }
}
```