```
// EP 2 - AED II - Tarde - 2018 - Profº Xavier
// Nome: Bryan Munekata
// NUSP: 9911444

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

typedef struct reg {
   int vertice;
   float peso;
   struct reg * prox;
} Reg;

typedef struct reg2 {
   int vertice;
   int cor; // 0 p/ branco, 1 p/ cinza, 2 p/ preto
   float d;
   int pi;
   struct reg2 * prox;
} Reg2;

int v;
Reg2 resultado[10000];
int tempo = 0;

// Fila dos vértices
int fila[10000];

int enqueue(int v) {
   int tmp = fila[0];
   int n = 0;
   while (tmp != -1) {
       tmp = fila[++n];
   }
   fila[n] = v;

   return 0;
}

int dequeue() {
   int dequeue_number = fila[0];

   for (int i = 0; i < 10000; i++) {
       fila[i] = fila[i + 1];
   }

   return dequeue_number;
}
```

```
int bfs(Reg lista_adj[], int vertice) {
   for (int i = 0; i < v; i++) {
       if (i != vertice) {
           resultado[i].cor = 0;
           resultado[i].d = -1;
           resultado[i].pi = -1;
       }
   }

   resultado[vertice].cor = 1;
   resultado[vertice].d = 0;
   resultado[vertice].pi = -1;

   enqueue(resultado[vertice].vertice);
   while (fila[0] != -1) {
       int u = dequeue();

       Reg * tmp = &lista_adj[u - 1];

       while (tmp->prox != NULL) {
           tmp = tmp->prox;
           if (resultado[tmp->vertice].cor == 0) {
               resultado[tmp->vertice].cor = 1;
               resultado[tmp->vertice].d = resultado[u - 1].d + tmp->peso;
               resultado[tmp->vertice].pi = u;
               enqueue(resultado[tmp->vertice].vertice);
           }
       }

       resultado[u - 1].cor = 2;
   }

   // Imprime resultados
   for (int i = 0; i < v; i++) {
       printf("%0.2f ", resultado[i].d);
   }

   printf("\n");

   return 0;
}

int main(int argc, char *argv[]) {
   //Abre txt
   FILE * fp = fopen(argv[1], "r");

   //Le a primeira linha e armazena o numero de vertices no 'v'
   fscanf(fp, "%i", &v);
```

```
//Monta lista_adj de adjacencia
Reg lista_adj[v - 1];

for (int i = 0; i < v; i++) {
    lista_adj[i].vertice = i + 1;
    lista_adj[i].prox = NULL;
}

// Monta lista_adj de resultado
for (int i = 0; i < v; i++) {
    resultado[i].vertice = i + 1;
    resultado[i].cor = 0;
    resultado[i].d = -1;
    resultado[i].pi = -1;
    resultado[i].prox = NULL;
}

int origem, destino;
float peso;

//Le linha por linha e insere vertices na lista_adj de adjacencia
while (fscanf(fp, "%i %i %f", &origem, &destino, &peso) != EOF) {
    Reg * no_vertice = (Reg *) malloc((int)sizeof(Reg));

    no_vertice->vertice = destino;
    no_vertice->peso = peso;
    no_vertice->prox = NULL;

    Reg * tmp = &lista_adj[origem];

    while (tmp->prox != NULL) {
        tmp = tmp->prox;
    }

    tmp->prox = no_vertice;

    //insere vertices no sentido contrario da lista_adj de adjacencia
    Reg * no_vertice_invertido = (Reg *) malloc((int)sizeof(Reg));

    no_vertice_invertido->vertice = origem;
    no_vertice_invertido->peso = peso;
    no_vertice_invertido->prox = NULL;

    Reg * tmp1 = &lista_adj[destino];

    while (tmp1->prox != NULL) {
        tmp1 = tmp1->prox;
    }
```

```
        tmp1->prox = no_vertice_invertido;
    }

    fclose(fp);

    // Preenche o vetor da fila com -1 (NULL)
    for (int i = 0; i < 10000; i++) {
        fila[i] = -1;
    }

    // Executa BFS em cada vértice
    for (int i = 0; i < v; i++) {
        bfs(lista_adj, i);
    }

    return 0;
}
```

input1.txt (passado no primeiro argumento ao chamar o script):

```
7
0 1 1.1
0 2 2.2
0 3 3.3
1 4 0.1
1 5 0.2
1 6 0.3
```

output (saída na tela):

```
0.00 1.10 2.20 3.30 1.20 1.30 1.40
1.10 0.00 3.30 4.40 0.10 0.20 0.30
2.20 3.30 0.00 5.50 3.40 3.50 3.60
3.30 4.40 5.50 0.00 4.50 4.60 4.70
1.20 0.10 3.40 4.50 0.00 0.30 0.40
1.30 0.20 3.50 4.60 0.30 0.00 0.50
1.40 0.30 3.60 4.70 0.40 0.50 0.00
```