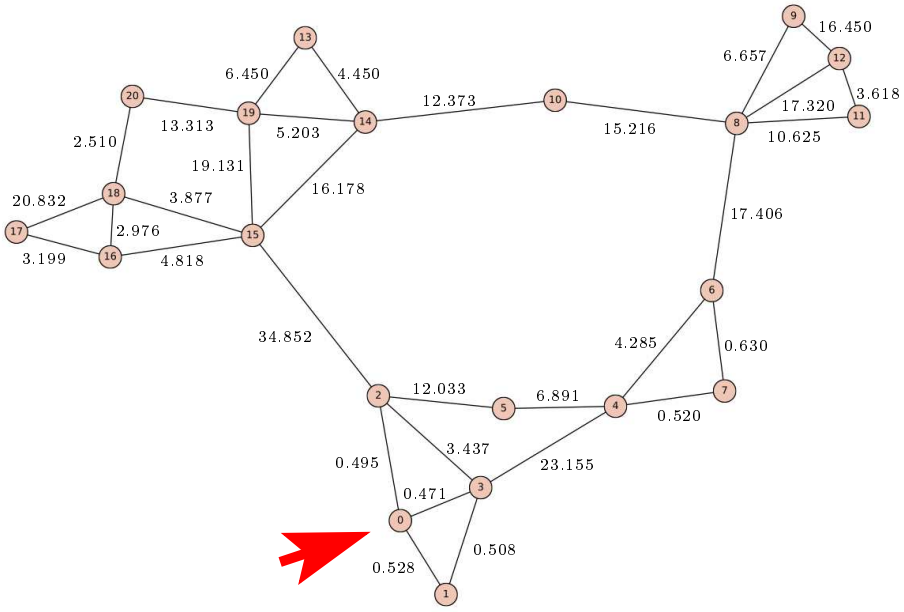
Matemàtica computacional i analítica de dades
Algorítmia i combinatòria en grafs ...
Curs 2020–21

8 Cerca de camins òptims. Algoritme Dijkstra.

L'esquema següent mostra l'esquema d'una xarxa d'ordinadors (fictícia, però no impossible) on en cada aresta s'ha marcat el temps que necessita un senyal per arribar d'un ordinador a l'altre (temps de resposta)



En el fitxer `InfoXarxa.txt` hi ha el temps de resposta de cada una de les connexions (directes) d'aquests ordinadors. Podeu veure com les primeres línies

```
0:1,0.528;2,0.495;3,0.471
1:0,0.528;3,0.508
```

mostren que el node 0 està connectat amb l'1, el 2 i el 3; i que aquestes connexions tenen un temps de resposta de 0.528, 0.495 i 0.471 respectivament; per al node 1 les connexions s'estableixen amb el 0 i el 3 amb temps 0.528 (simètric) i 0.508. El codi del fitxer `Pr08.c` permet carregar aquestes dades i, per a comprovar que tot ha funcionat, les mostra en el terminal.

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

#define MAXARST 5

typedef struct{
    int numnode;
    double temps;
}infoaresta;

typedef struct{
    int narst;
    infoaresta arestes[MAXARST];
}Connode;
```

```
15
16 int main()
17 {
18     FILE *dades;
19     int nvert=0,c,i,j,node;
20
21     if((dades=fopen("InfoXarxa.txt","r"))==NULL)
22     {
23         printf("\nNo s'ha accedit al fitxer de dades\n");
24         return 1;
25     }
26     while((c=fgetc(dades))!=EOF)
27     {
28         if(c=='\n') nvert++;
29     }
30     rewind(dades);
31     printf("%d nodes\n",nvert);
32     printf("-----------------------------------------------\n");
33     Connode infnodes[nvert];
34     for (i=0; i<nvert; i++)
35     {
36         fscanf(dades,"%i",&node);
37         j=0;
38         while(fgetc(dades)!='\n'){
39             fscanf(dades,"%i",&infnodes[node].arestes[j].numnode);
40             fgetc(dades);
41             fscanf(dades,"%lf",&infnodes[node].arestes[j].temps);
42             j++;
43         }
44         infnodes[node].narst=j;
45     }
46     fclose(dades);
47     for (i=0; i<nvert; i++)
48     {
49         printf("Connexions del node %i:\n",i);
50         for (j=0; j<infnodes[i].narst; j++)
51         {
52             printf("\tNode %2i\t\tTemps %6.3lf\n",infnodes[i].arestes[j].
    numnode,
53                     infnodes[i].arestes[j].temps);
54         }
55         printf("-----------------------------------------------\n");
56     }
57     printf("FI\n");
58     return 0;
59 }
```

8.1 Exercici

Afegiu com a les dues primeres línies del programa i en format comentari els vostres `Nom`, `Cognom` i `NIU`.

El nom dels programes que contenen el codi ha de ser de la forma `PrXExY.c`, on `X` fa referència a la pràctica i `Y` a l'exercici. Per exemple, el nom del programa de l'apartat següent hauria de ser `Pr8Ex811.c`.

Exercici 8.1.1: Obteniu, mitjançant l'algoritme Dijkstra, el camí més ràpid per a connectar l'ordinador 0 amb cada un dels altres. Repasseu, abans de començar a escriure el codi, com funciona aquest algoritme.

Si el graf a estudiar és molt gran i l'objectiu és el temps a un node concret, és millor treballar amb una cua prioritària que conté els nodes dels que ja tenim un temps

calculat però encara no sabem si és el temps mínim, i aturar-se quan analitzem el node al que volem arribar.

En aquest cas, com que el graf és petit i volem calcular el temps per a tots els nodes, mantindrem una llista de nodes pendents amb una estimació del recorregut corresponent a cada un d'ells. A cada pas, treurem d'aquesta llista el node amb recorregut més curt, actualitzant l'estimació del recorregut per a tots els nodes que estan connectats amb ell i no hagin sortit de la llista de nodes pendents.

Una possiblitat és utilitzar una estructura del tipus:

```
typedef struct{
    double temps;       // el temps que triguem si venim pel node
    anterior
    int pendent;        // si el node encara està pedent o no
    unsigned anterior; // número de node anterior
}EstatDij;
```

I guardar a una variable un vector `LaCua` de tipus `EstatDij` amb tants elements com nodes tingui el graf. Per exemple, abans de començar inicialitzarem el nodel pel que comencem (el `0`):

```
LaCua[0].pendent=1;
LaCua[0].temps=0.0;
```

I la resta de nodes `i>0`:

```
LaCua[i].pendent=1;
LaCua[i].temps=LF_INFINIT; // LF_INFINIT seria un temps superior a tots
```

Comencem una iteració on a cada pas:

- Busca el node que estigui `pendent==1` i tingui un valor de `temps` més baix (el primer cas serà `i=0`) i posar el camp `LaCua[i].pendent=0`.
- Ha de mirar les connexions `j` del node `i` i, per cada `j` tal que `LaCua[j].pendent==1` (en el primer cas encara tots!) mirar quan es triga en arribar: la suma del que es triga en arribar a `i` (que ja la tenim) i el temps que tenim a la informació del graf que es triga en anar de `i` a `j`.

En el nostre cas, a la primera iteració actualitzarem les variables:

```
LaCua[0].pendent=0;
LaCua[1].temps=0.0+0.528;
LaCua[1].anterior=0;
LaCua[2].temps=0.0+0.495;
LaCua[2].anterior=0;
LaCua[3].temps=0.0+0.471;
LaCua[3].anterior=0;
```

A la segona iteració triarem el node amb `pendent==1` (per tant ja no pot ser el node `0`) que tingui el `temps` més baix, obtenint que hem d'analitzar `i=3` ...

En el cas general, la iteració acabaria quan no quedés cap node amb `pendent==1` amb `temps` finit, però en aquest cas, com que el graf és connex, podeu acabar quan s'hagin passat tots els nodes a `pendent=0`.

Si mostrem la informació del node cada cop que el seleccionem per passar-lo de `pendent==1` a `pendent==0`, quedaria un llistat com:

```
Node  0  Temps: 0.000 Cami optim: 0 0
Node  3  Temps: 0.471 Cami optim: 0 3
Node  2  Temps: 0.495 Cami optim: 0 2
Node  1  Temps: 0.528 Cami optim: 0 1
Node  5  Temps:12.528 Cami optim: 0 2 5
Node  4  Temps:19.419 Cami optim: 0 2 5 4
Node  7  Temps:19.939 Cami optim: 0 2 5 4 7
Node  6  Temps:20.569 Cami optim: 0 2 5 4 7 6
Node 15  Temps:35.347 Cami optim: 0 2 15
Node  8  Temps:37.975 Cami optim: 0 2 5 4 7 6 8
Node 18  Temps:39.224 Cami optim: 0 2 15 18
Node 16  Temps:40.165 Cami optim: 0 2 15 16
Node 20  Temps:41.734 Cami optim: 0 2 15 18 20
Node 17  Temps:43.364 Cami optim: 0 2 15 16 17
Node  9  Temps:44.632 Cami optim: 0 2 5 4 7 6 8 9
Node 11  Temps:48.600 Cami optim: 0 2 5 4 7 6 8 11
Node 14  Temps:51.525 Cami optim: 0 2 15 14
Node 12  Temps:52.218 Cami optim: 0 2 5 4 7 6 8 11 12
Node 10  Temps:53.191 Cami optim: 0 2 5 4 7 6 8 10
Node 19  Temps:54.478 Cami optim: 0 2 15 19
Node 13  Temps:55.975 Cami optim: 0 2 15 14 13
```

No intenteu ser el més eficients possible des del principi. Convé que comenceu amb una versió on sigui fàcil distingir tots el passos i, un cop tingueu una versió funcional, mireu d'optimitzar el programa respecte el nombre d'operacions que ha de realitzar i de la memòria que pot necessitar. Penseu que per a un graf petit com el de l'exemple no hi ha d'haver cap problema per molt ineficient que sigui el programa.

Instruccions finals

Quan acabeu la pràctica, feu el lliurament dels fitxers de codi (que tenen els noms de la forma `Pr8ExY.c` segons el que hem indicat anteriorment) a través del Campus Virtual des de l'apartat de lliuraments de l'assignatura. Recordeu que al principi de cada fitxer hi ha d'haver el vostre nom i NIU.