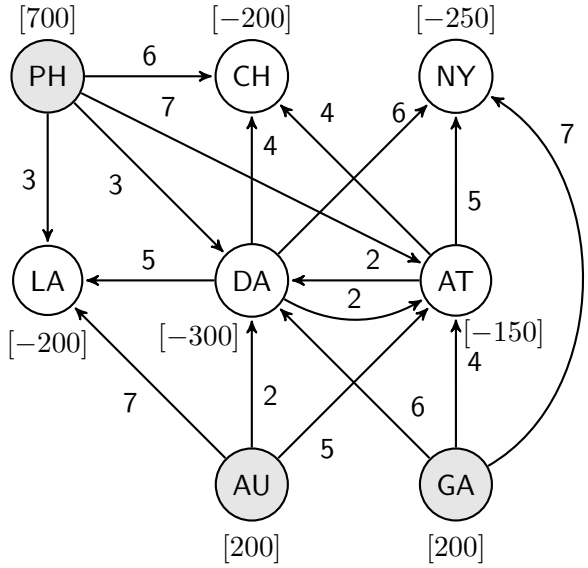
3 Modelització amb GLPK. Opcions avançades del llenguatge GNU MathProg

GNU MathProg és el llenguatge que utilitzem per a descriure els problemes de programació lineal en GLPK i que després el programa glpsol interpreta. Aquest llenguatge admet moltes més opcions de les que hem vist en les dues pràctiques anteriors. En aquesta pràctica veurem algunes de les possibilitats més avançades que ofereix el llenguatge GNU MathProg.

A la carpeta de documentació `doc` que es crea amb la instal·lació del paquet GLPK hi trobareu el manual `gmpl.pdf` que conté les especificacions i opcions en versió més completa del que veurem a classe. També és molt interessant la carpeta d'exemples, entre els quals hi ha el sudoku que explicarem. Finalment, a la carpeta `doc` hi trobareu el manual `glpk.pdf` i `glpk-java.pdf` de documentació de les biblioteques de C i java, respectivament, que es poden cridar des d'un programa fet per l'usuari, sense passar pel programa glpsol.

3.1 Declaració de conjunts com a subconjunts d'altres conjunts i una alternativa per a donar valors als paràmetres-matriu

L'enginyer de logística d'una empresa té el següent problema: ha de subministrar materials als clients de diferents ciutats (Los Angeles, Dallas, Chicago, New York i Atlanta) des dels magatzems que estan a altres localitats (Phoenix, Austin i Gainesville). Les rutes i els costos de transport són els que indica el graf següent, havent-hi dues ciutats destí que també actuen de node:



En els vèrtexs del graf dirigit també s'indica, entre claudàtors, el nombre d'existències a cada magatzem (en positiu) i la demanda a cada destí (en negatiu). A més, en cada arc hi pot haver un flux de com a màxim 200 unitats. Volem calcular l'estratègia de distribució que optimitzi costos.

Les variables d'aquest problema són els fluxos en cada un dels arcs. Aquests arcs venen determinats per una parella ordenada de vèrtexs (ciutats), les quals seran també importants a l'hora de d'especificar les restriccions. En cada ciutat hi tenim una restricció:

$$\text{material que surt de la ciutat} - \text{material que arriba a la ciutat} = \text{requeriment de la ciutat}, \quad (1)$$

on el requeriment és un número positiu si a la ciutat hi ha un magatzem (indica existències), un número negatiu si la ciutat és destí (indica demanda) i zero si la ciutat només és un node de distribució(també anomenat node de pas).

Si x_{kj} denota la quantitat que circula per l'arc que va des de la ciutat k a la ciutat j, i r_i indica el requeriment a la ciutat i amb el conveni dels signes que hem fixat, l'equació (**??**) a la ciutat i és

$$\sum_{\substack{j \text{ tal que} \\ (i,j) \text{ és arc}}} x_{ij} - \sum_{\substack{k \text{ tal que} \\ (k,i) \text{ és arc}}} x_{ki} = r_i$$

A més els arcs tenen una capacitat de transport limitada (no més de 200 unitats per ruta):

$$x_{ij} \leq 200, \quad \text{per a tot arc } (i,j).$$

Una manera de resoldre-ho en GLPK és declarant primer el conjunt de ciutats i després declarant el conjunt de rutes (arcs) com a subconjunt del conjunt de parelles de ciutats (producte cartesià de ciutats per ciutats).

```
set Ciutats;
set Rutes in  {Ciutats, Ciutats};

# en el paràmetre demanda_oferta escriurem l'oferta en positiu i la demanda
    en negatiu.
param demanda_oferta{Ciutats};
param cost_transport{Rutes};

var x{Rutes}>=0;

minimize costos: sum{(i,j) in Rutes} x[i,j]*cost_transport[i,j];


subject to
maxim_flux{(i,j) in Rutes}: x[i,j]<=200;

equilibri{j in Ciutats}: sum{(j,i) in Rutes} x[j,i] - sum{(i,j) in Rutes} x
   [i,j] = demanda_oferta[j];


data;

set Ciutats:= dal ny chi pho atl la aus gain;
set Rutes:= (pho,*) chi dal la atl (dal,*) la chi ny atl (atl,*) dal chi ny
    (aus,*) la dal atl (gain,*) dal atl ny;

param demanda_oferta:=
dal -300
ny -250
chi -200
```

```
28 pho 700
29 atl -150
30 la -200
31 aus 200
32 gain 200;
33
34 param cost_transport:=
35 [pho,*] chi 6 dal 3 la 3 atl 7
36 [dal,*] la 5 chi 4 atl 2 ny 6
37 [atl,*] dal 2 chi 4 ny 5
38 [aus,*] la 7 dal 2 atl 5
39 [gain,*] dal 6 atl 4 ny 7;
```

En la línia 2 també podem declarar el conjunt de `Rutes`, com a subconjunt del producte cartesià doble del conjunt `Ciutats`, usant la següent sintaxi alternativa: `set Rutes within Ciutats cross Ciutats;`

Aquesta declaració del conjunt `Rutes` ens permetrà denotar les arestes com a parelles de ciutats, per exemple com `(pho,dal)`.

Fixem-nos ara com introduïm el conjunt `Rutes` en l'apartat de dades, a la línia 22:
`set Rutes:= (pho,*) chi  dal la atl (dal,*) la chi ny atl...`
Escrivim de manera prou curta totes les parelles (rutes) que surten de `pho`, després totes les que comencen per `dal`, etc.

De forma molt similar donem valor al paràmetre `cost_transport` a les línies 34–39.

3.2 Utilització de paràmetres amb valors per defecte

El problema anterior també el podem resoldre declarant variables indexades a totes les parelles de ciutats i després imposant que el flux sigui nul en les parelles que no són rutes possibles del problema. No declararíem el conjunt `Rutes` i la declaració de variables seria `var{Ciutats,Ciutats};` Per a no haver d'introduir costos de transport entre parelles de ciutats que després no usarem podem utilitzar l'opció `default` (valor per defecte) en la declaració del paràmetre:

```
param costos_transport{Ciutats,Ciutats}, default 0;
```

Aquesta opció fa que si, per a una parella de ciutats, no li assignem cap altre valor al paràmetre, el sistema li donarà el valor 0. Amb l'opció `default 0` a la declaració del paràmetre podem donar valors al paràmetre `costos_transport` en el bloc de dades exactament igual com ho fèiem en les línies 34–39 del codi de la secció anterior.

Per a modelitzar el fet que les rutes inexistents tinguin flux 0 es pot aprofitar la limitació de capacitat, és a dir, posar capacitat 200 a les rutes existents i 0 a les inexistents. Podem aprofitar també l'opció default en la declaració del nou paràmetre `traficmaxim`:

```
param traficmaxim{Ciutats,Ciutats}, default 0;
```

i després donar-li valors en el bloc de dades com hem fet en amb el paràmetre `cost_transport`.

Naturalment hauríem de modificar el 200 de la línia 14 del codi anterior per `traficmaxim[i,j]`.

3.3 Utilització de condicions lògiques en la definició dels paràmetres

La proposta alternativa que acabem d'explicar per a resoldre el problema de transport de la l'exemple de **??** tenia l'inconvenient que havíem de declarar el nou paràmetre–matriu `traficmaxim` i després omplir-lo al bloc de dades. Aquest paràmetre de fet només prenia dos valors:

`traficmaxim[i,j]=200` si (i,j) és ruta possible i `traficmaxim[i,j]=0` si no ho és.

El paràmetre `costos_transport` valia 0 (el defecte) en les parelles (i,j) que eren rutes impossibles. Podem aprofitar aquest fet per a escriure els dos casos anteriors:

`traficmaxim[i,j]=0` si `costos_transport[i,j]=0`

`traficmaxim[i,j]=200` en cas contrari.

Aquest fet ens permet definir el paràmetre directament assignant-li els valors segons aquestes dues condicions. En la seva declaració escrivim:

```
param traficmaxim{i in Ciutats,j in Ciutats}:=
                                    if costos_transport[i,j] = 0 then 0 else 200;
```

I ja no cal donar-li valors en el bloc de dades.

3.4 Veure per pantalla la solució en un format triat

La funció `printf` del llenguatge C es pot utilitzar també en GLPK per veure la solució per pantalla o també per detectar errors en el resultat. Les opcions són les mateixes que en el C:

per imprimir per pantalla el paràmetre p que conté una cadena de caràcters

```
printf "el paràmetre val %s", p;
```

per imprimir una variable x que pren un valor enter:

```
printf "la variable x val %d", x;
```

per imprimir un número real a amb tres decimals i després saltar de línia:

```
printf "número amb decimals %.3f \n", a;
```

si volem imprimir dues variables x i y una entera i l'altra real posem:

```
printf "x = %d y=  %.3f ", x,y;
```

si volem imprimir per pantalla la solució que ha calculat GLPK, cal que escrivim la funció `solve;` abans de `printf` — (tot això ha d'anar abans del bloc de dades).

Per exemple, el codi que podríem posar en el primer exemple per veure per pantalla només aquelles rutes que s'utilitzen a la solució optimal i quina quantitat s'hi transporta:

```
1 ...
2 equilibri{j in Ciutats}: sum{(j,i) in Rutes} x[j,i] - sum{(i,j) in Rutes} x
     [i,j] = demanda_oferta[j];
3
4 solve;
5 for{(i,j) in Rutes: x[i,j]!=0} printf "\n de %s a %s ---> %d", i,j, x[i,j];
6 printf "\n\n";
7
8 data;
9 ...
```

La sortida per pantalla és la següent:

```
OPTIMAL LP SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (137084 bytes)

 de pho a chi ---> 200
 de pho a dal ---> 200
 de pho a la ---> 200
 de pho a atl ---> 100
 de dal a ny ---> 50
 de dal a atl ---> 50
 de aus a dal ---> 200
 de gain a ny ---> 200
```

3.5 Utilització de condicions lògiques en els conjunts d'índexs

A vegades en algunes fórmules ens convé fer variar els índexs (en un sumatori, per exemple) en un conjunt prèviament declarat, però amb alguna condició addicional. Fixem-nos, per exemple, en l'exercici 6 de la pràctica 2, en què s'havia de calcular el nombre de treballadors en cada torn en una planta química. Cada dia tenia tres torns. Els treballadors durant tot un mes en feien un durant quatre dies de la setmana consecutius. Les restriccions venien donades pel mínim nombre de treballadors que necessitava la planta cada torn i cada dia de la setmana:

	dll	dt	dc	dj	dv	ds	dg
nit	5	3	2	4	3	2	2
matí	7	8	9	5	7	2	5
tarda	9	10	10	7	11	2	2

```
/* VERSIO COMPLICADA */
set torns;
set dies:=1..7;
param necessitats{torns,dies};

var x{torns,dies}>=0,integer; #nombre de treballadors el torn i dia

minimize z:sum{i in torns, j in dies} x[i,j];

subject to
r_NecessitatTorn{j in torns, k in dies}: sum{i in dies: (k<=i+3 and i<=k)
    or k<=i+3-7} x[j,i] >= necessitats[j,k];

data;
set torns:=nit mati tarda;

param necessitats:
        1  2   3   4   5   6   7:=
nit     5  3   2   4   3   2   2
mati    7  8   9   5   7   2   5
tarda   9  10  10  7   11  2   2;

end;
```

La línia 11 del codi GLPK anterior conté totes les restriccions. Per satisfer el requisit de mínim nombre de treballadors en el torn j (matí, tarda o nit) i dia k (de 1 a 7) que ens dóna la taula, cal tenir en compte que un treballador que fa quatre dies seguits en el torn j començant el dia i treballa els dies $i, i+1, i+2, i+3$. Per tant, perquè el dia k estigui treballant cal que

$$i \leq k \leq i+3 \quad \text{o} \quad k \leq i+3-7 \text{ (si ens passem de 7 i tornem a començar setmana)}$$

Per al torn j i dia k hem de sumar tots els treballadors que fan torn j i comencen dia i satisfent les condicions anteriors:
`sum{i in dies: (k<=i+3 and i<=k) or k<=i+3-7} x[j,i] >= necessitats[j,k]`

3.6 Utilització de condicions lògiques per a donar les restriccions. El sudoku (tret dels exemples de la instal·lació de GLPK)

```
/* Volem resoldre el sudoku següent:
   +-------+-------+-------+
   | 5 3 . | . 7 . | . . . |
   | 6 . . | 1 9 5 | . . . |
   | . 9 8 | . . . | . 6 . |
   +-------+-------+-------+
   | 8 . . | . 6 . | . . 3 |
   | 4 . . | 8 . 3 | . . 1 |
   | 7 . . | . 2 . | . . 6 |
   +-------+-------+-------+
   | . 6 . | . . . | 2 8 . |
   | . . . | 4 1 9 | . . 5 |
   | . . . | . 8 . | . 7 9 |
   +-------+-------+-------+   */

param givens{1..9, 1..9}, integer, >= 0, <= 9, default 0;
/* els números que ja estan col·locats  */

var x{i in 1..9, j in 1..9, k in 1..9}, binary;
/* x[i,j,k] = 1 vol dir que la casella [i,j] té assignat el número k */

s.t. fa{i in 1..9, j in 1..9, k in 1..9: givens[i,j] != 0}:
     x[i,j,k] = (if givens[i,j] = k then 1 else 0);
/* assignació dels números predefinits donats al paràmetre "givens" */

s.t. fb{i in 1..9, j in 1..9}: sum{k in 1..9} x[i,j,k] = 1;
/* restricció que cada cel·la té assignat un sol número */

s.t. fc{i in 1..9, k in 1..9}: sum{j in 1..9} x[i,j,k] = 1;
/* restricció que cada número ha d'aparèixer un cop a cada fila  */

s.t. fd{j in 1..9, k in 1..9}: sum{i in 1..9} x[i,j,k] = 1;
/* restricció que cada número ha d'aparèixer un cop a cada columna  */

s.t. fe{I in 1..9 by 3, J in 1..9 by 3, k in 1..9}:
     sum{i in I..I+2, j in J..J+2} x[i,j,k] = 1;
/* cada número ha de sortir un cop a cada quadrat 3 x 3 */

solve;

```

```
41 /* Les instruccions a continuació són per treure per pantalla la solució
      amb format */
42 for {i in 1..9}
43 {  for {0..0: i = 1 or i = 4 or i = 7}
44       printf " +-------+-------+-------+\n";
45    for {j in 1..9}
46    {  for {0..0: j = 1 or j = 4 or j = 7} printf(" |");
47       printf " %d", sum{k in 1..9} x[i,j,k] * k;
48       for {0..0: j = 9} printf(" |\n");
49    }
50    for {0..0: i = 9}
51       printf " +-------+-------+-------+\n";
52 }
53
54 data;
55 /* Dades de l'exemple */
56
57 param givens : 1 2 3 4 5 6 7 8 9 :=
58            1   5 3 . . 7 . . . .
59            2   6 . . 1 9 5 . . .
60            3   . 9 8 . . . . 6 .
61            4   8 . . . 6 . . . 3
62            5   4 . . 8 . 3 . . 1
63            6   7 . . . 2 . . . 6
64            7   . 6 . . . . . 2 8 .
65            8   . . . 4 1 9 . . 5
66            9   . . . . 8 . . 7 9 ;
67 end;
```

Fixem-nos en com en la línia 16 definim el paràmetre `givens`: ha de ser enter entre 0 i 9, i l'opció `default 0` vol dir que si en el bloc de dades no donem cap valor a una component, aquesta valdrà zero.

Després en les línies 57–66 introduïm els valors del paràmetre, com fèiem habitualment amb els paràmetres-matriu, però amb `.` en algunes components. Això provoca que el sistema els hi doni el valor per defecte, que hem fixat en zero. És a dir, que el GLPK farà els càlculs amb la matriu que hem posat al bloc de dades substituint els punts per zeros.

Les línies 22–23 utilitzen condicions lògiques tant en el conjunt d'índexs

```
s.t. fa{i in 1..9, j in 1..9, k in 1..9: givens[i,j] != 0}:
```

com en el terme independent de les restriccions:

```
x[i,j,k] = (if givens[i,j] = k then 1 else 0);
```

Les línies 42–52 són per a treure per pantalla el sudoku resolt en un format determinat.

3.7 Exercicis

1. Una empresa d'electrodomèstics està pensant en construir vàries fàbriques per proveir cinc ciutats que tenen una demanda anual estimada, en milers d'unitats, donada per:

ciutats	C_1	C_2	C_3	C_4	C_5
demanda	5	4	3	5	6

   S'ha establert tres possibles localitzacions, L_1, L_2 i L_3, per a les fàbriques, totes elles amb les mateixes característiques. La producció màxima de cada fàbrica seria de 15 milers d'unitats anuals i el cost anual d'amortització de la inversió seria igual a 10 milions d'euros. En la taula següent es recull el benefici aconseguit venent cada unitat de producte produït a la fàbrica F_i a la ciutat C_j, en milers d'euros.

	C_1	C_2	C_3	C_4	C_5
L_1	0.5	0.7	0.8	0.8	0.9
L_2	0.9	0.7	0.9	0.9	0.7
L_3	1	0.8	1	0.8	0.9

   Plantegeu el problema de programació lineal per a decidir en quina o quines localitzacions s'obren fàbriques per a maximitzar el benefici.

2. Una empresa de Califòrnia té actualment un magatzem a cadascuna de les ciutats següents: Baltimore (A), Cheyenne (B), Salt Lake City (C), Memphis (D) i Wichita (E). Aquests magatzems subministren a clients de diverses àrees, que podem considerar localitzades a les ciutats de Atlanta (1), Boston (2), Chicago (3), Denver (4), Omaha (5) i Portland (6).

   L'empresa té la sensació que hi ha massa magatzems i que es poden reduir una mica els costos totals si se'n tanquen un o més, tot i l'increment que podria suposar en els costos de transport. Les dades rellevants queden resumides a la taula següent:

Cost per tona i mes

Ciutat del client / Magatzem	**1**	**2**	**3**	**4**	**5**	**6**	Cap.	Fix
A	1675	500	685	1630	1160	2800	12	7650
B	1460	1940	970	200	495	1200	18	3500
C	1925	2400	1425	500	950	900	21	3500
D	480	1355	643	1045	665	2321	16	4100
E	922	1646	700	508	411	1797	25	2200
Demanda	10	8	12	6	7	11		

   El *cost per tona i mes* és la matriu que dóna el cost mensual de transportar una tona de material des de cada magatzem a cada ciutat receptora. *Cap.* és la "capacitat del magatzem", és a dir la quantitat màxima de tones que pot servir en un mes. *Fix* és el cost fix mensual de tenir obert el magatzem, independentment de la quantitat que serveixi. Finalment, la *demanda* és la quantitat mensual que cal servir a cada ciutat receptora.

   (a) Quin és el cost total actual? (Trobeu la distribució òptima, mantenint la situació actual de magatzems, tots oberts).

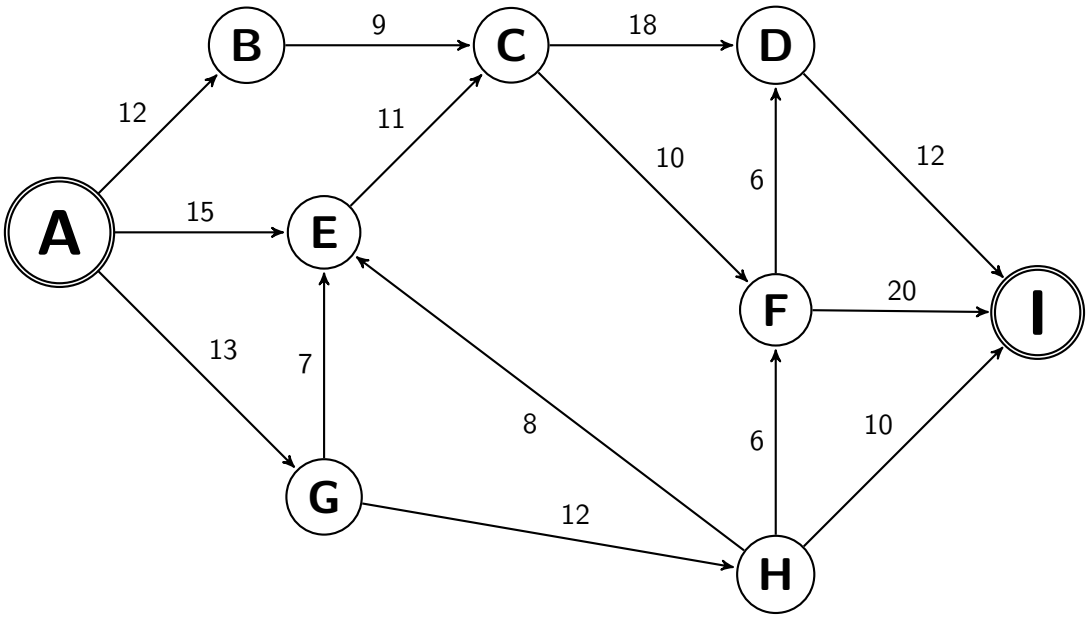
(b) Suposem ara que no és obligatori mantenir tots els magatzems oberts. Trobeu les localitzacions i les distribucions òptimes.

(c) Suposem que restringim la quantitat màxima de magatzems a 4, 3, 2 o 1. Trobeu en cada cas els costos totals òptims si és possible i comenteu els resultats.

3. El dibuix mostra la xarxa de vols setmanals que una companyia aèria té pactats amb el consorci d'aeroports i que connecten les ciutats A amb I.

a) Quin és el màxim nombre de vols que la companyia pot oferir entre les dues ciutats?

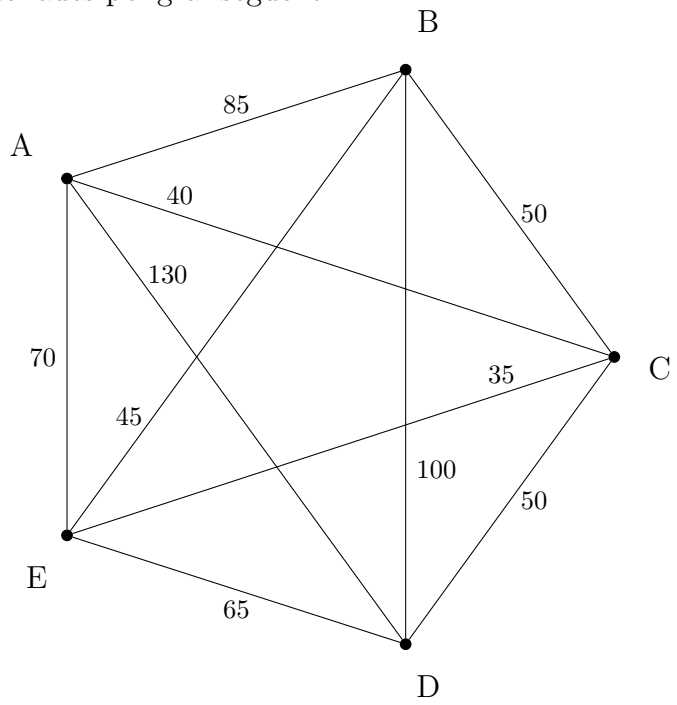
b) Suposeu que un dels aeroports de les ciutats de connexió s'ha de tancar per obres. Quin tancament afectaria menys a la companyia?



Suggeriments de programació:

- Haureu de distingir el node d'origen i el node de destí de la xarxa. Per això podeu declarar dos paràmetres no numèrics que guardaran el nom d'aquests nodes. En la declaració heu de posar:
  `param nomdelnode symbolic ;`
- La restricció d'equilibri (o conservació) de flux s'ha d'imposar a cadascun dels nodes de la xarxa, excepte per als nodes d'origen i de destí. Aquest conjunt de nodes de pas el podeu escriure:
  `NODES diff {nomdelnode1, nomdelnode2}`

4. Trobeu la ruta més curta per a un viatjant que viu a la ciutat A i ha de visitar les ciutats B, C, D i E (només una vegada) i tornar a casa seva. Les distàncies de les diferents rutes entre les ciutats vénen donades pel graf següent:



5. Resoleu el problema de saber si és possible col·locar 8 reines en un tauler d'escacs sense que es matin, amb la condició que hi hagi una reina en la casella de la fila i i la columna j.

6. Practiqueu la següent manera d'escriure, en el bloc de dades, les dades d'un conjunt i d'un paràmetre(definit sobre el conjunt) alhora:
`param : nomdelconjunt : nomdelparametre :=`
En l'exemple de la xarxa de l'apartat 3.1, podem definir el conjunt de *Rutes* i el paràmetre *cost_transport* alhora:

```
34 param: Rutes: cost_transport:=
35 pho chi 6
36 pho dal 3
37 pho la 3
38 pho atl 7
39 dal la 5
40 ...
```