
PARADIGMAS DE PROGRAMAÇÃO III

LESI – 2º ANO

2003 – 2004

ANEXO ÀS NOTAS TEÓRICAS

GRAFOS

Prof. F. Mário Martins

GRAFOS

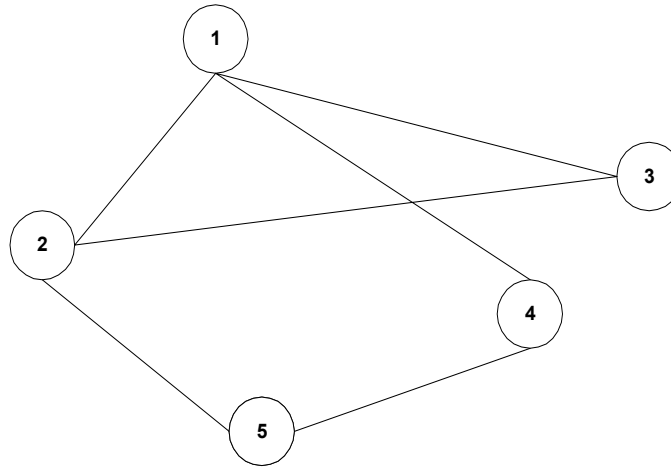
1.- Definições.

GRAFOS são, formalmente, um par $\mathbf{G} = (\mathbf{N}, \mathbf{A})$ em que \mathbf{N} é um conjunto de **nodos** (“nodes”) (*ou pontos ou vértices*), tendo definida, e representando, uma qualquer **relação binária** entre *pares* dos seus nodos, o que define \mathbf{A} como sendo um **conjunto de pares de nodos** relacionados entre si por tal relação, conjunto esse designado, genericamente, pelo conjunto das **ligações** (“edges”).

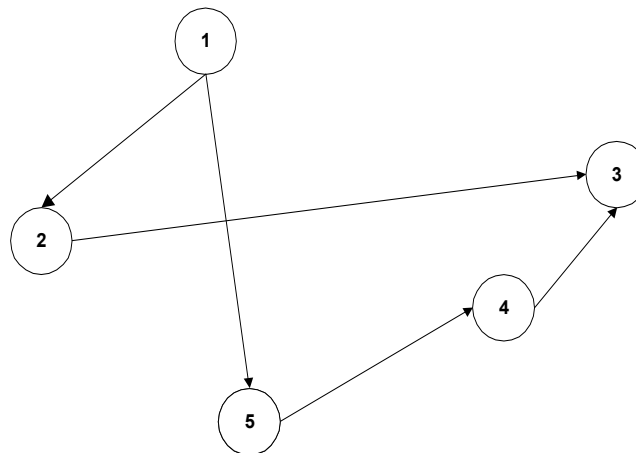
Os GRAFOS possuem uma representação diagramática muito característica, que consiste na apresentação dos **nodos como círculos** e nas **relações entre pares de nodos como segmentos ou setas**, conforme, respectivamente, não exista ou exista **ordem** nos relacionamentos entre tais pares.

Quando tais pares de nodos não possuem uma ordem, ou seja, quando o par (a, b) é semanticamente igual ao par (b, a), o grafo diz-se **não orientado** e as relações são representadas por **ligações ou linhas (“edges” ou “lines”)**. Quando existe ordem em tais pares e o par (a, b) é semanticamente diferente do par (b, a), então o grafo diz-se **orientado** (“directed graph” ou “digraph”) e as relações são representadas por **setas (“arcs”)**. Embora o conjunto de nodos possa sempre ser infinito, apenas lidaremos com grafos com **número finito de nodos**.

GRAFO NÃO ORIENTADO

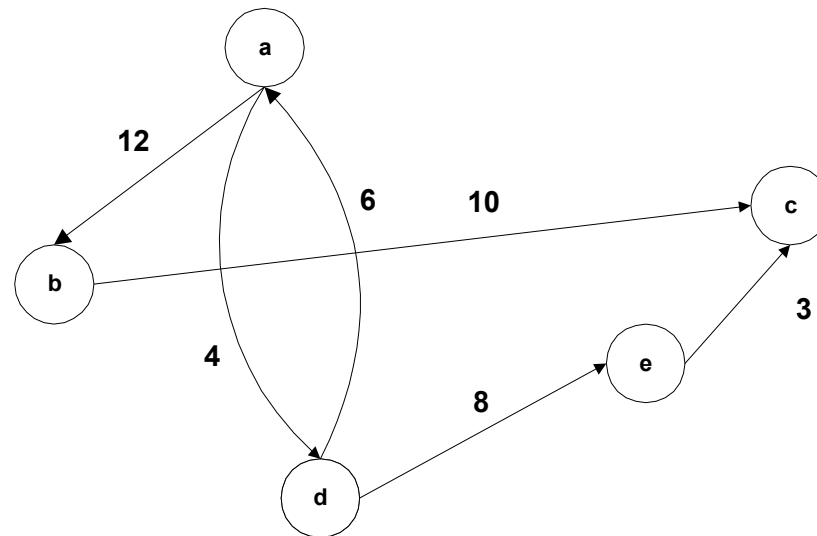


GRAFO ORIENTADO



Os **Grafos Orientados Etiquetados** (“labelled”) possuem informação adicional associada a cada um dos seus arcos, informação essa que pode ser dos mais variados tipos e com a mais diversa estrutura. No exemplo associam-se valores inteiros que representam custos de travessia de tais arcos durante um percurso pelo grafo (o que quer que tais **custos** representem semanticamente).

GRAFO ORIENTADO ETIQUETADO



Para além de muitas outras propriedades, os grafos que neste contexto iremos abordar caracterizam-se por serem:

- **Orientados (ou não);**
- **Acíclicos (ou não);**
- **Etiquetados (ou não);**

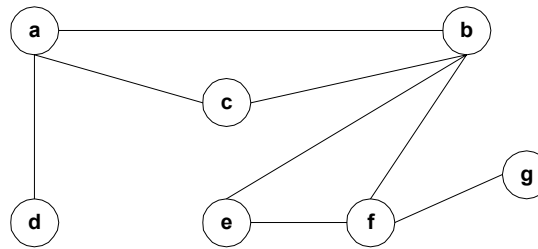
Uma matriz bidimensional poderia ser construída visando caracterizar as propriedades dos vários tipos de grafos, que são, de facto, muito usados em Informática e noutras áreas científicas.

Os GRAFOS, cuja teoria é complexa, importante, mas é sempre colocada em último lugar ou considerada dispensável em Engenharia Informática e Ciências da Computação, são uma das estruturas de dados mais importantes em Computação, quer pela sua generalidade (de facto, listas e árvores são casos particulares de grafos) quer pelo grande número de áreas de aplicação (cf. produção industrial, investigação operacional, inteligência artificial, teoria dos autómatos, etc.).

Apresentaremos em seguida **diferentes representações de grafos em Prolog** bem como um conjunto de predicados que correspondem às mais usuais e interessantes operações sobre grafos, em particular predicados e algoritmos que têm directamente a ver com problemas genéricos de determinação de múltiplas soluções em contextos em que a informação se encontra relacionada entre si através de relações binárias estáticas e dinâmicas que configuram usuais problemas de grafos e de inteligência artificial.

2.- Representações de Grafos em Prolog.

Considere-se o grafo seguinte:



Uma primeira hipótese de representação deste grafo consiste em **representar como factos Prolog as ligações existentes entre nodos**, cf.

```
ligacao(a, b). ligacao(a, c).  
ligacao(a, d). ligacao(b, c).  
ligacao(b, e). ligacao(b, f).  
ligacao(e, f). ligacao(f, g).
```

Se pretendêssemos representar o facto de que as ligações são bi-direccionais, ou seja, que o grafo não é orientado (sendo as ligações comutativas), poderíamos fazer o seguinte:

- a) *acrescentar 8 novos factos `ligacao/2`*;
- b) *definir a regra `ligacao(X,Y):- ligacao(Y,X)`*. (como sabemos tal seria muito mau por questões que têm a ver com recursividade infinita !!);

c) definir a regra

```
ha_caminho(X, Y) :- ligacao(X,Y) ; ligacao(Y,X).
```

d) finalmente, e cf. se aconselhou anteriormente, definir

```
ha_caminho(X, Y) :- ligacao(X,Y), !.  
ha_caminho(X, Y) :- ligacao(Y,X).
```

Outra representação possível para este grafo seria a que consiste em representá-lo como uma estrutura formada pela lista de nodos e a lista de ligações, cf.

```
grafo([a,b,c,d,e,f,g], [ligacao(a, b), ligacao(a,c), ...])
```

Se o grafo fosse etiquetado, então poderíamos associar a cada ligação o respectivo peso, cf.

```
ligacao(a, b, 30).
```

ou mesmo informação mais complexa, cf.

```
ligacao(a, b, info(distancia(20), custo(100), ...)).
```

sendo sempre possível separar as ligações das etiquetas caso tal seja julgado mais conveniente, mas relacionando tal informação através de *chaves/códigos*, cf., por exemplo

```
ligacao(1, a, b,).  
info(1, distancia(20), custo(100), ...)).
```

Uma outra representação mais compacta poderia ser:

```
grafo([a,b,c,d,e,f,g], [ligacao(a, b, 20),  
                        ligacao(a, c, 15), ...])
```

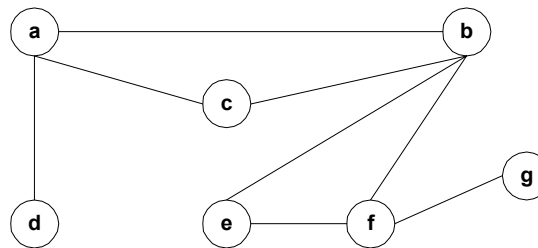
Finalmente, poderemos sempre representar o grafo como sendo uma lista de pares onde a cada nodo se associa uma lista de todos os seus nodos adjacentes, ou seja, aos quais se liga directamente (eventualmente com etiquetas), cf.

```
grafo([a/[b,c,d],b/[c,e,f], ...]).
```

```
grafo([a/[b/20,c/15,d/23],b/[c/12,e/5,f/8], ...]).
```

Estas várias representações possíveis serão mais ou menos efectivas consoante o tipo de problema a resolver, devendo-se sempre procurar um equilíbrio entre legibilidade da representação e eficiência da sua utilização em Prolog.

Consideremos de novo o grafo original



representado na BC usando os seguintes factos e regras:

```
ligacao(a, b). ligacao(a, c).  
ligacao(a, d). ligacao(b, c).  
ligacao(b, e). ligacao(b, f).  
ligacao(e, f). ligacao(f, g).
```

```
ha_caminho(X, Y) :- ligacao(X, Y), !.  
ha_caminho(X, Y) :- ligacao(Y, X).
```

Vamos, com base nesta representação, definir para já predicados que permitam responder às questões essenciais sobre os grafos assim representados, designadamente:

- 1.- *Quais os nodos acessíveis a partir de um nodo X ?*
- 2.- *Existe algum caminho que ligue dois nodos dados ?*
- 3.- *Quais são todos os caminhos entre dois nodos ?*

Outras questões interessantes poderiam ser colocadas relacionadas com a teoria geral dos grafos, como por exemplo, como percorrer um grafo completamente passando 1 e 1 só vez por cada nodo, como percorrer um grafo com o menor esforço possível, etc.

De momento porém, estas são as questões que nos interessam analisar, compreender e responder usando o paradigma em estudo.

Considere-se que a BC contém a seguinte representação do grafo e as seguintes regras e predicados:

```
ligacao(a, b).  
ligacao(a, c).  
ligacao(a, d).  
ligacao(b, c).  
ligacao(b, e).  
ligacao(b, f).  
ligacao(e, f).  
ligacao(f, g).
```

```
ligaDirecto(X, Y) :- ligacao(X, Y). /* sem CUT para ser usado em findall, etc. */  
ligaDirecto(X, Y) :- ligacao(Y, X).
```

```
/* ----- PREDICADOS ----- */
```

```
/* nodos directamente acessíveis do nodo X, ou seja, adjacentes */
```

```
acessiveisDe(X, Ln) :- findall(Y, ligaDirecto(X, Y), Ln), !.  
acessiveisDe(_, []).
```

```
/* predicado que determina se há ou não caminho entre dois nodos A e B */
```

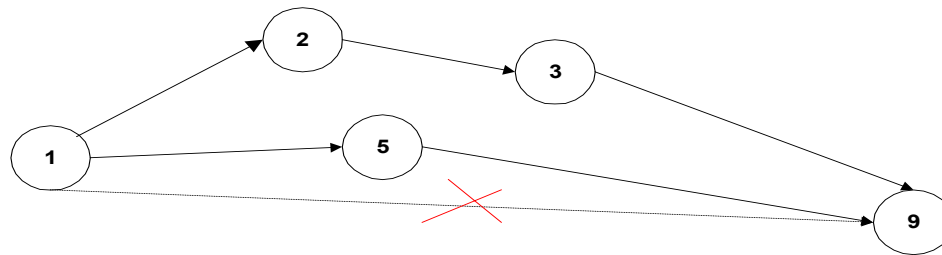
```
ha_caminho(A, B) :- ligaDirecto(A, B), !.  
ha_caminho(A, B) :- ligaDirecto(A, X), ha_caminho(X, B).
```

/* determinação de um dos possíveis caminhos entre A e B */

caminho(A, B, Cam) :-
 travessia(A, B, **[A]**, Cam).

travessia(A, B, Visitados, [B | Visitados]) :-
 ligaDirecto(A, B).

travessia(A, B, Visitados, Cam) :-
 ligaDirecto(A, C),
 C \== B, /* evita recursividade com B */
 \+ member(C, Visitados), /* evita ciclos */
 travessia(C, B, [C | Visitados], Cam).



travessia(1, 9, [1], Cam) ~~→~~ **[1, 9]**

travessia(1, 9, [1], Cam)

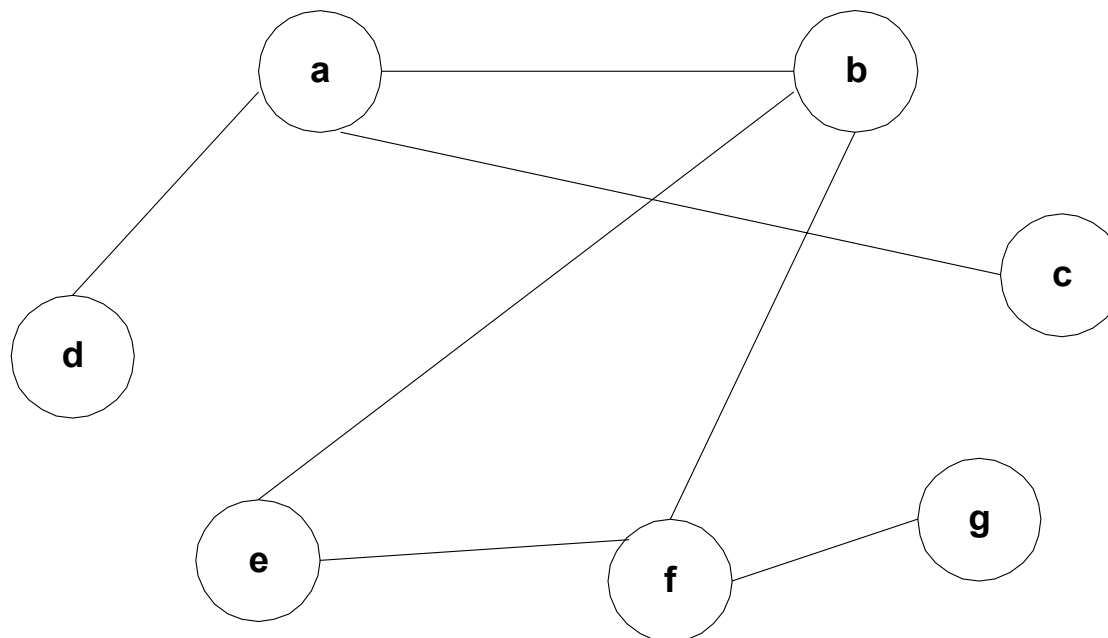
ligaDirecto(1,2)	→	[9, 3, 2, 1]
ligaDirecto(1,5)	→	[9, 5, 1]
.....		

```
/* determinação de todos os caminhos entre A e B */
```

```
caminhos(A, B, Lc) :- setof(Cam, caminho(A, B, Cam), Lc), !.  
caminhos(_, _, []).
```

```
/*-----*/
```

Apresentam-se em seguida alguns exemplos da execução deste programa Prolog para o seguinte grafo:



```
SWI-Prolog (version 3.3.0)
% grafos_ex1 compiled 0.00 sec, 2,136 bytes

Yes
?- ligacao(a, Y).

Y = b ;
Y = c ;
Y = d ;

No
?- ligacao(X, Y).

X = a
Y = b ;

X = a
Y = c ;

X = a
Y = d ;

X = b
Y = c ;

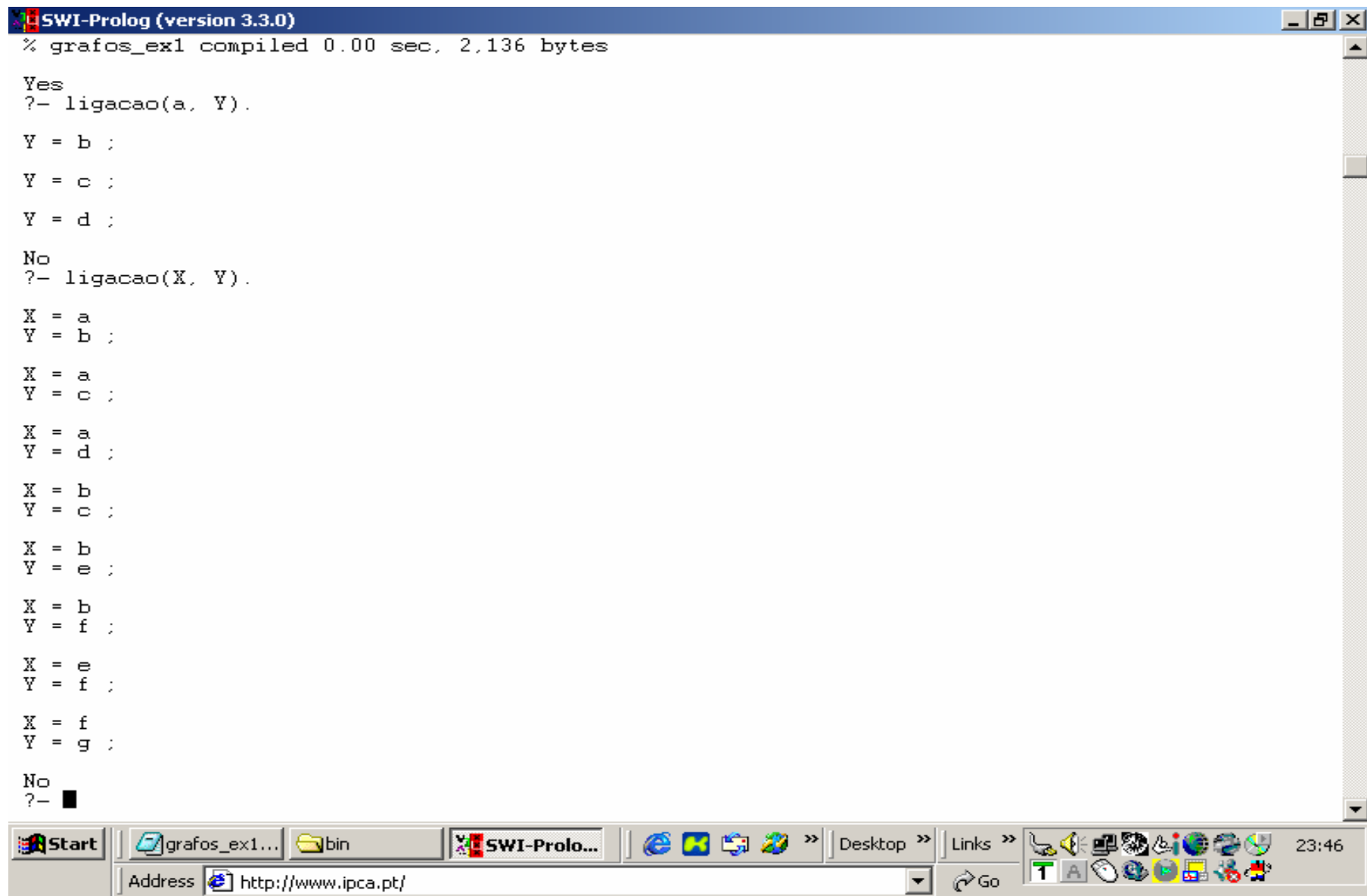
X = b
Y = e ;

X = b
Y = f ;

X = e
Y = f ;

X = f
Y = g ;

No
?-
```



```
SWI-Prolog (version 3.3.0)
Welcome to SWI-Prolog (Version 3.3.0)
Copyright (c) 1993-1999 University of Amsterdam. All rights reserved.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [grafos_ex1].
% grafos_ex1 compiled 0.00 sec, 2,836 bytes

Yes
?- acessiveisDe(f, L).

L = [g, b, e]

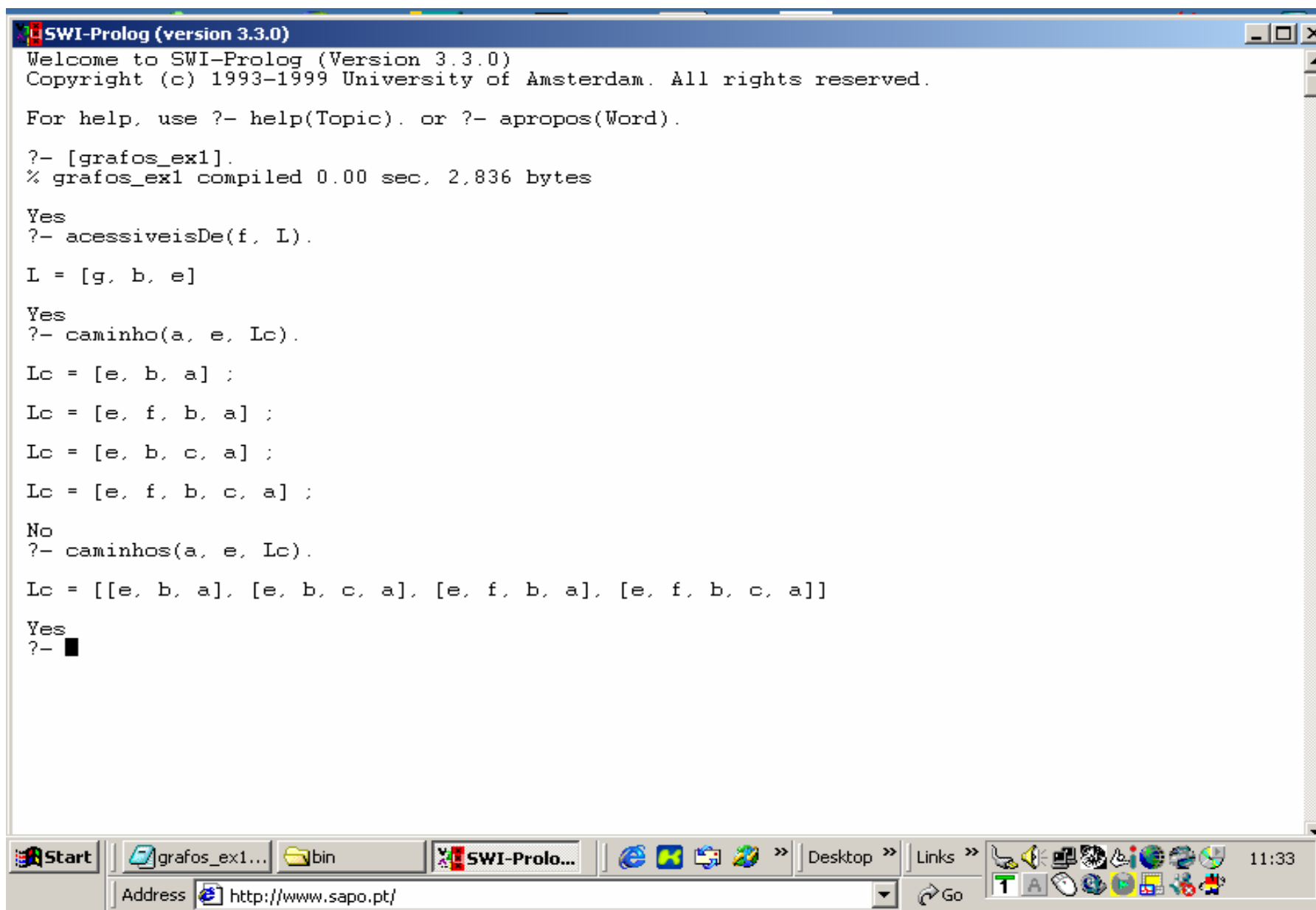
Yes
?- caminho(a, e, Lc).

Lc = [e, b, a] ;
Lc = [e, f, b, a] ;
Lc = [e, b, c, a] ;
Lc = [e, f, b, c, a] ;

No
?- caminhos(a, e, Lc).

Lc = [[e, b, a], [e, b, c, a], [e, f, b, a], [e, f, b, c, a]]

Yes
?-
```



The image shows a Windows XP desktop environment. In the foreground, there is a terminal window titled "SWI-Prolog (version 3.3.0)". The terminal displays the following text:

```
Welcome to SWI-Prolog (Version 3.3.0)
Copyright (c) 1993-1999 University of Amsterdam. All rights reserved.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [grafos_ex1].
% grafos_ex1 compiled 0.01 sec, 2,484 bytes

Yes
?- caminho(a, e, Cam).

Cam = [e, b, a] ;
Cam = [e, f, b, a] ;
Cam = [e, b, c, a] ;
Cam = [e, f, b, c, a] ;

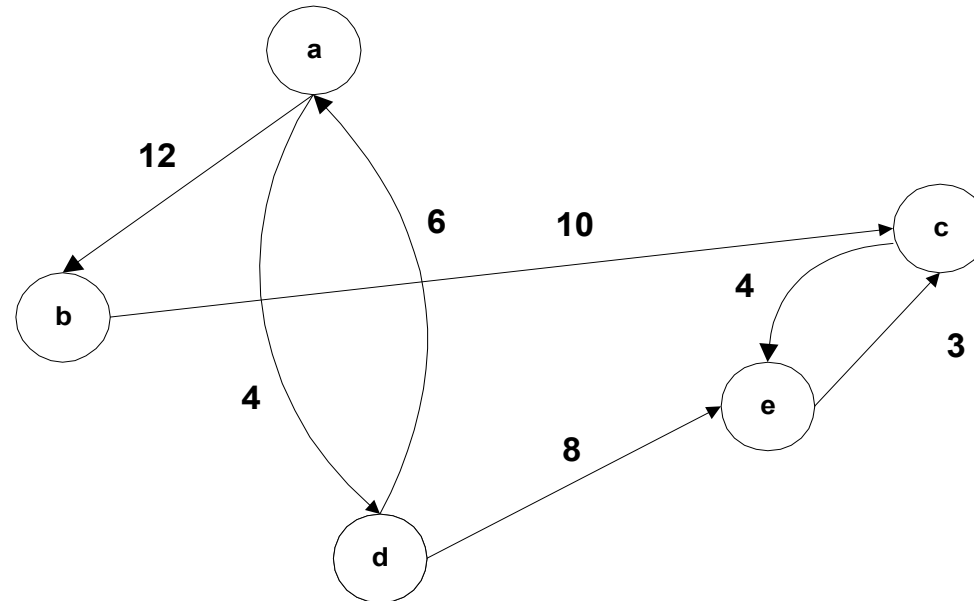
No
?- caminhos(a, e, Lc).

Lc = [[e, b, a], [e, b, c, a], [e, f, b, a], [e, f, b, c, a]] ;
Lc = [] ;

No
?-
```

Below the terminal window, a web browser window is visible. The address bar shows "http://www.ipca.pt/". The taskbar at the bottom includes the Start button, several open application windows (including "grafos_ex1...", "bin", and "SWI-Prolo..."), and a system tray with various icons and a clock showing "0:17".

Considere-se agora o seguinte grafo orientado e etiquetado.



que se irá representar por factos do tipo,

`ligacao(a, b, 12).`

Onde o terceiro parâmetro representa o **peso ou custo**, qualquer que seja a medida (cf. km, dias, preço, etc.) da travessia directa entre tais nodos.

Vejamos agora para estes grafos, as definições dos predicados mais gerais e mais importantes, usando uma nova BC e um conjunto de regras e predicados adequados.


```
/* GRAFO ORIENTADO COM CUSTOS */
```

```
ligacao(a, b, 12).  
ligacao(a, d, 4).  
ligacao(d, a, 6).  
ligacao(b, c, 10).  
ligacao(d, e, 8).  
ligacao(e, c, 3).  
ligacao(c, e, 4).
```

```
/* predicado que determina se há ou não caminho entre os nodos A e B */
```

```
ha_caminho(A, B) :- ligacao(A, B, _), !.  
ha_caminho(A, B) :- ligacao(A, X, _),  
                    ha_caminho(X, B).
```

```
/* predicado travessia/4 que realiza a travessia do grafo registrando os nodos  
percorridos, e que garante que não existem ciclos nas soluções */
```

```
travessia(A, B, Visitados, [B|Visitados]) :-  
    ligacao(A, B, _).
```

```
travessia(A, B, Visitados, Cam) :-  
    ligacao(A, C, _),  
    C \== B,  
    \+ member(C, Visitados),  
    travessia(C, B, [C|Visitados], Cam).
```

```
/* predicado que determina, aleatoriamente, um caminho entre A e B */
```

```
caminho(A, B, Cam) :- travessia(A, B, [A], Cam).
```

```
/* determinação de todos os caminhos entre dois nodos A e B */
```

```
caminhos(A, B, Lc) :-  
    setof(Cam, caminho(A, B, Cam), Lc), !.  
caminhos(_, _, []).
```

```
/* predicado que determina aleatoriamente um caminho entre A e B, dando como  
resultado tal caminho e o respectivo custo */
```

```
caminhoCusto(A, B, Cam, Custo) :-  
    travessiaCusto(A, B, [A], Cam, Custo).
```

```
travessiaCusto(A, B, Visitados,  
    [B|Visitados], Custo1) :- ligacao(A, B, Custo1).
```

```
travessiaCusto(A, B, Visitados, Cam, Custo) :-  
    ligacao(A, C, Custo2),  
    C \== B,  
    \+ member(C, Visitados),  
    travessiaCusto(C, B, [C|Visitados], Cam, CustoResto),  
    Custo is Custo2 + CustoResto.
```

```
/* determinação de todos os caminhos entre dois nodos A e B
   e respectivos Custos */
```

```
caminhosCusto(A, B, Lc) :-
    setof(Cam/Custo, caminhoCusto(A, B, Cam,
        Custo), Lc), !.
caminhosCusto(_, _, []).
```

```
/* predicado que calcula a lista de todos os nodos do grafo */
```

```
nodos(Ln) :- setof(N, (X^C^ligacao(X, N, C);
    Y^C^ligacao(N, Y, C)), Ln), !.
nodos([]).
```

```
SWI-Prolog (version 3.3.0)
Welcome to SWI-Prolog (Version 3.3.0)
Copyright (c) 1993-1999 University of Amsterdam. All rights reserved.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [grafos_ex2].
% grafos_ex2 compiled 0.01 sec, 3,612 bytes

Yes
?- caminho(a, e, Cam).

Cam = [e, c, b, a] ;
Cam = [e, d, a] ;

No
?- caminhoCusto(a, e, Cam, Custo).

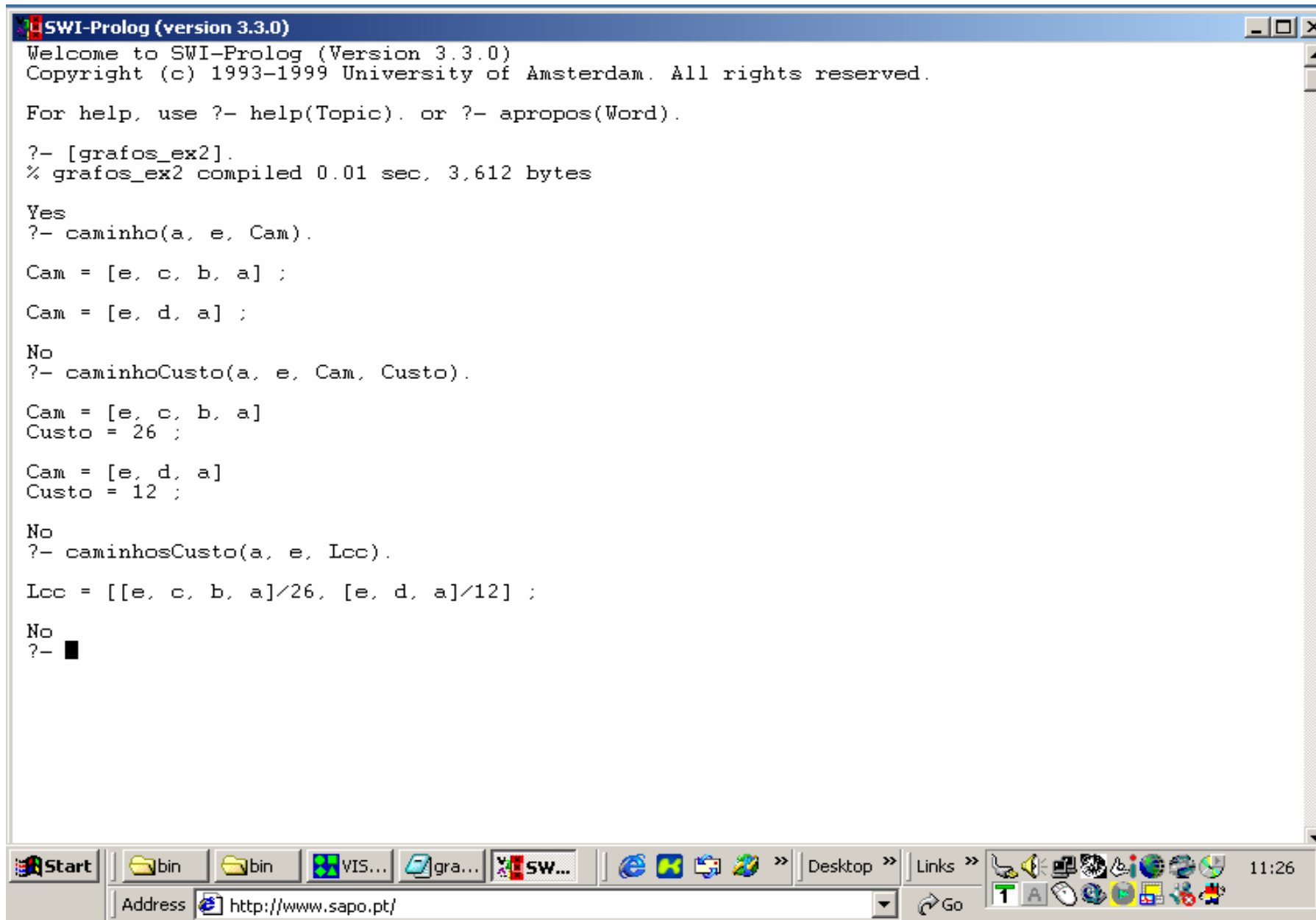
Cam = [e, c, b, a]
Custo = 26 ;

Cam = [e, d, a]
Custo = 12 ;

No
?- caminhosCusto(a, e, Lcc).

Lcc = [[e, c, b, a]/26, [e, d, a]/12] ;

No
?-
```



Outros problemas envolvendo grafos poderiam ser considerados e são mesmo muito importantes de serem considerados. Porém, no contexto desta disciplina, estes são os problemas fundamentais a tratar usando teoria de grafos e usando a linguagem Prolog.

Todos os problemas que possa vir a ser necessário resolver, quer nos trabalhos práticos quer na avaliação teórica desta disciplina, são baseados nestes predicados, sendo, quando muito, casos particulares dos mesmos.

Finalmente, a teoria de grafos é muito mais rica e envolve problemas muito mais complexos e interessantes (cf. problema do “carteiro”, travessias diversas, etc.), aqui não abordados.

Prof. F. Mário Martins