

Concurso Nacional de Programação Lógica e Funcional
CNPLf'2005 – Parte da tarde (4 horas; 5 problemas)

<http://www.estig.ipb.pt/cnplf05>

Departamento de Informática e Comunicações
ESTIG – Instituto Politécnico de Bragança

14 de Maio de 2005
14h00m – 18h00m

Conteúdo

Organização	2
Problema T1: Intervalos Temporais	3
Problema T2: Sopa de Letras Hexagonal	5
Problema T3: Numogramas	7
Problema T4: Cavaleiro da Ordem de Malta	9
Problema T5: O seu a seu dono	12

Organização

Comissão Científica do CNPLf'05

- Delfim Torres – delfim@mat.ua.pt
- Miguel Filgueiras – mig@ncc.up.pt
- Amilcar Cardoso – amilcar@dei.uc.pt
- Salvador Abreu – spa@di.uevora.pt
- Jaime Brito Ramos – jabr@math.ist.utl.pt
- Maria Antónia Lopes – mal@di.fc.ul.pt
- Pedro Guerreiro – pg@di.fct.unl.pt
- José Julio Alferes – jja@di.fct.unl.pt
- Rui Mendes – azuki@di.uminho.pt
- Luis Reis – lpreis@fe.up.pt
- Simão Melo de Sousa – desousa@di.ubi.pt
- João Paulo Azevedo – pja@di.uminho.pt
- Pedro Rangel Henriques – prh@di.uminho.pt
- Maria João Varanda – mjoao@ipb.pt

Comissão Organizadora

- Maria João Varanda – mjoao@ipb.pt
- Pedro Henriques – prh@di.uminho.pt
- Henrique Cardoso – hlc@ipb.pt
- Rui Pedro Lopes – rlopes@ipb.pt
- André Coelho – ei7677@alunos.ipb.pt
- Eduardo Teixeira – a13677@alunos.ipb.pt
- Pedro Sousa – ei12260@alunos.ipb.pt

Apoio Técnico

- Fernando Soares – fsoares@ipb.pt

Secretariado

- Fernanda Maçorano – macorano@ipb.pt

INTERVALOS TEMPORAIS

Introdução

Nos anos 80, James Allen propôs uma abordagem à representação e ao raciocínio temporais baseada em intervalos temporais, definidos como segmentos da linha temporal limitados por dois pontos: (t_1, t_2) , com $t_1 < t_2$.

Um dos aspectos da proposta de Allen foi a adopção de um conjunto de 13 relações temporais binárias entre intervalos, mutuamente exclusivas (ver Figura 1).

Relação	Exemplo	Relação	Exemplo
α before β		α after β	
α equal β			
α meets β		α met_by β	
α overlaps β		α overlapped_by β	
α during β		α contains β	
α starts β		α started_by β	
α finishes β		α finished_by β	

Figura 1: Relações temporais binárias; α e β são intervalos temporais

Allen demonstrou que para todo o par de intervalos α e β definidos na linha temporal, existe uma e uma só relação r de entre as treze tal que $\alpha r \beta$.

Por exemplo, a relação entre o intervalo $\alpha = (2, 5)$ e o intervalo $\beta = (3, 8)$ é “overlaps”, ou seja, $(2, 5)$ overlaps $(3, 8)$.

Vamos considerar um modelo discreto em que os tempos são representados por dois inteiros $H:M$, onde $H \in [0, 23]$ e $M \in [0, 59]$, significando respectivamente Horas e Minutos, com circularidade nas horas, dado que estamos a ignorar a representação dos dias. Por exemplo, as seguintes relações são verdadeiras:

- (15:30, 17:30) before (20:12, 0:34)
- (23:20, 1:35) meets (1:35, 4:03)
- (1:35, 4:03) met_by (23:20, 1:35)
- (23:20, 1:35) during (22:12, 4:03)
- (22:12, 4:03) contains (23:20, 1:35)
- (22:12, 1:35) overlaps (23:20, 4:03)

Tarefa

A sua tarefa consiste em escrever um programa que seja capaz de determinar qual a relação temporal existente entre dois intervalos dados.

Se optar pela Programação Lógica, implemente o programa através de um predicado `reltemp/3`, em que os dois primeiros argumentos devem estar inicialmente instanciados com intervalos e o terceiro argumento deverá, no final, ficar instanciado com a respectiva relação temporal.

Se optar pela Programação Funcional, desenvolva uma função `reltemp/2`, em que os dois argumentos são intervalos e o resultado a respectiva relação temporal.

Os Dados

Os dados do problema são dois intervalos temporais no formato $(T1, T2)$. Os tempos representam-se no formato H:M, tal como foi dito atrás.

Os Resultados

O resultado pretendido é o nome da relação entre o primeiro e o segundo intervalos dados, de acordo com o quadro da Figura 1.

Exemplo (Prog. Lógica)

```
?- reltemp((2:30,3:13), (2:30,3:42), Rel).  
    Rel = starts
```

Exemplo (Prog. Funcional)

```
> reltemp (2:30,3:13) (2:30,3:42)  
> starts
```

SOPA DE LETRAS HEXAGONAL

Introdução

As sopas de letras sempre foram um divertimento muito apreciado pelas pessoas que gostam de jogos de palavras. Aqui temos uma variante interessante: a sopa de letras hexagonal. Neste caso, o objectivo é descobrir palavras válidas (i.é, existentes num dado dicionário) escondidas no hexágono. As palavras podem ser descobertas seguindo qualquer um dos seis sentidos possíveis a partir de uma casa do tabuleiro. Esses sentidos correspondem às seis linhas de casas contíguas (na mesma direcção) que partem de cada uma das seis arestas da casa considerada.

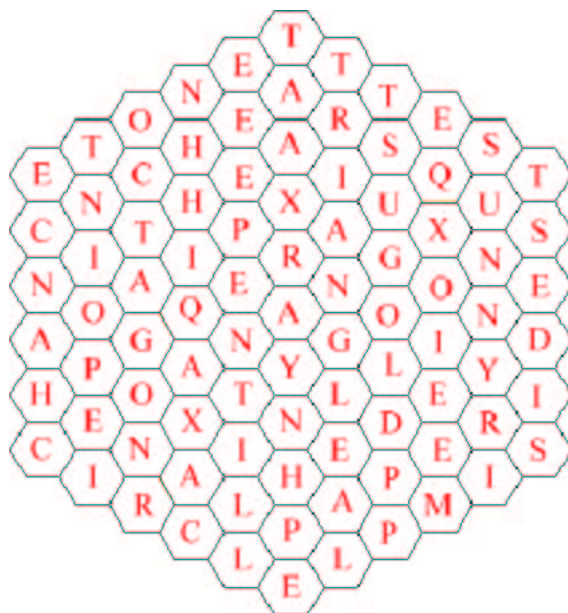


Figura 1: Sopa de letras hexagonal

O Alberto gosta bastante destes problemas mas no fim da organização das jornadas de informática sente-se muito cansado. Com a preguiça que tem neste momento pediu-lhe a si que o resolvesse. Como você adora programação, decidiu utilizar os seus conhecimentos resolvendo este problema automaticamente.

Tarefa

O programa deverá receber como parâmetros dois argumentos: o primeiro consiste no hexágono; e o segundo contém a lista de palavras do dicionário a considerar. O objectivo é descobrir quais das palavras pertencentes ao dicionário se encontram escondidas na sopa de letras hexagonal.

No caso de preferir uma linguagem de programação em lógica, deverá implementar o predicado `sopa/3` em que os 2 primeiros argumentos representam o hexágono e o dicionário e, como de costume, o último retorna o resultado.

Caso prefira as linguagens funcionais, desenvolva a função equivalente `sopa/2` que recebe os argumentos definidos acima e devolve o resultado.

Os Dados

Os argumentos do programa consistem numa estrutura contendo o hexágono e outra contendo o dicionário. O dicionário é simplesmente uma lista de palavras. A definição do hexágono utiliza uma lista de listas, em que cada uma dessas listas representa as letras de uma linha de casas hexagonais. O varrimento é efectuado sempre no mesmo sentido. Por exemplo, no caso da figura 1, o hexágono poderia ser representado por:

```
[[e,t,o,n,e,t],[c,n,c,h,e,a,t],...]
```

Os Resultados

O resultado da invocação do predicado ou da função `sopa` é devolvido numa lista que contém as palavras encontradas no hexágono.

Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido:

```
?- sopa([[a,c,a],[l,a,n,r],[o,o,t,o,o],[r,a,a,r],[u,m,m]],
        [arpao, basco, carta, toma, marco, ola, casto,
         cao, tosco, anta, um, aro, mata], Res)
R = [ola, cao, anta, um, aro, mata]
```

Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
> sopa [[a,c,a],[l,a,n,r],[o,o,t,o,o],[r,a,a,r],[u,m,m]]
      [arpao, basco, carta, toma, marco, ola, casto,
       cao, tosco, anta, um, aro, mata]
> [ola, cao, anta, um, aro, mata]
```

NUMOGRAMAS

Introdução

Todos vós certamente já fizeram Numogramas ou, pelo menos, sabem o que são. Os Numogramas são uma espécie de palavras cruzadas, que costumam aparecer em jornais e revistas, mas com números e, nos mais difíceis, também com operações. Dito de outra forma, na sua versão mais difícil são puzzles, como o que se mostra abaixo, onde se preenchem os espaços em branco com algarismos ou operações, por forma a que todas as igualdades (quer na horizontal quer na vertical) sejam verdadeiras.

	+	2	//	2	=	
		+		-		
	*		//	2	=	6
-		//		+		
1	+	5		2	=	3
=		=		=		
7		1				

Tarefa

A sua tarefa consiste em escrever um programa que resolva este problema, ou seja, que dado um Numograma como o da figura, devolva, para cada um dos espaços em branco, qual o algarismo ou operação a colocar nesse espaço para que todas as igualdades, quer na horizontal quer na vertical, se verifiquem. Havendo várias soluções possíveis para um puzzle, basta que o seu programa devolva uma (qualquer) delas.

Se optar pela Programação Lógica, implemente o programa através de um predicado `numo/2` em que o primeiro argumento contém os dados (i.e. a definição do Numograma) e o segundo devolve o resultado, conforme descrito abaixo. Se escolher a Programação Funcional, desenvolva uma função `numo/1` que recebe como argumento os dados (a definição do Numograma) descritos abaixo e devolve o resultado.

Os Dados

O Numograma é dado como input, quer para a programação em lógica quer funcional, sob a forma de uma lista, em que cada elemento corresponde a uma das linhas do puzzle. Uma linha vai ser também uma lista com os elementos que a constituem, onde os quadrados a preto têm a constante `p`, os quadrados a preencher têm a constante `b` e as restantes posições têm ou um algarismo ou uma operação. As operações possíveis são: soma, subtração, multiplicação e divisão inteira.

Por exemplo, para o Numograma da figura, o input deveria ser a lista:

```
[ [b, +, 2, //, 2,'=', b],
  [b, p, +, p, -, p, p],
  [b, *, b, //, 2,'=', 6],
  [-, p, //, p, +, p, p],
  [1, +, 5, b, 2,'=', 3],
  ['=', p,'=', p,'=', p, p],
  [7, p, 1, p, b, p, p] ]
```

Os Resultados

O resultado deverá ser uma lista de ternos (**Linha,Coluna,Valor**), com um terno destes para cada posição **Linha/Coluna** que tenha um espaço em branco no puzzle, e onde **Valor** deverá conter o algarismo ou operação a colocar nessa posição por forma a satisfazer todas as igualdades conforme descrito acima.

Exemplo (Prog. Lógica)

Abaixo apresenta-se a chamada ao predicado `numo/2` e uma solução relativamente ao Numograma da figura acima.

```
?- numo( [ [b, +, 2, //, 2,'=', b],
           [b, p, +, p, -, p, p],
           [b, *, b, //, 2,'=', 6],
           [-, p, //, p, +, p, p],
           [1, +, 5, b, 2,'=', 3],
           ['=', p,'=', p,'=', p, p],
           [7, p, 1, p, b, p, p] ], Res).
```

```
Res = [ (1, 1, 2), (1, 7, 2), (2, 1, *), (3, 1, 4), (3, 3, 3),
        (5, 4, //), (7, 5, 2)]
```

Exemplo (Prog. Funcional)

```
> numo [ [b, +, 2, //, 2,'=', b],
         [b, p, +, p, -, p, p],
         [b, *, b, //, 2,'=', 6],
         [-, p, //, p, +, p, p],
         [1, +, 5, b, 2,'=', 3],
         ['=', p,'=', p,'=', p, p],
         [7, p, 1, p, b, p, p] ]
```

```
> [ (1, 1, 2), (1, 7, 2), (2, 1, *), (3, 1, 4), (3, 3, 3),
    (5, 4, //), (7, 5, 2)]
```

CAVALEIRO DA ORDEM DE MALTA

Introdução

Ser Cavaleiro da Ordem de Malta (CM) é uma *honra superior* que sempre fascinou a nobreza europeia (a portuguesa incluída). Ser CM significa, entre muitas outras incumbências, organizar e apoiar peregrinações a locais sagrados como Roma, Jerusalém, ou Santiago de Compostela, . . .

Para que um *nobre*¹, do sexo masculino, possa propor-se a Cavaleiro de Malta é necessário, entre outros requisitos de carácter e de riqueza própria (que serão atestados pelos seus *padri-nhos*²), provar uma de duas coisas:

- que tem ascendência nobre, por *varonia legítima* (pelo lado paterno e fruto de união legal (por casamento)), há pelo menos 150 anos;
- que tem nas últimas 4 gerações pelo menos 2 antepassados com título de nobreza (*barão, conde, marquês* ou *duque*).

Para verificar a ascendência nobre de um indivíduo, ou a existência de títulos de nobreza, recorre-se à sua *Árvore Genealógica* (AG)—uma árvore binária em que cada indivíduo (representado por um código) está associado à AG do seu pai (à esquerda) e à AG da sua mãe (à direita), ou **vazio** caso esse antecessor seja desconhecido ou não seja nobre—e ao *Livro de Nobreza* (LN)—uma colecção de registos em que cada ficha contém dados sobre indivíduos nobres: código de identificação, nome, ano de nascimento, tipo de filiação (Legítimo, ou Ilegítimo), e o título de nobreza (*barão, conde, marquês, duque, ou nenhum*).

Para medir a ascendência nobre há pelo menos 150 anos toma-se como referência a data de nascimento do indivíduo e a data de nascimento de cada antecessor nobre (que conste na AG). Quanto à contagem das gerações, a primeira é a dos pais do indivíduo.

Tarefa

A sua tarefa consiste em escrever um programa que, dada a *árvore genealógica* de um nobre e um *livro de nobreza* verifique se o indivíduo na raiz da árvore cumpre as condições para se propor a Cavaleiro da Ordem de Malta.

Se optar pela Programação Lógica, implemente o programa através de um predicado `maltes/2` em que os 2 argumentos são os dados e que sucede se e só se o resultado da verificação for positivo.

Se escolher a Programação Funcional, desenvolva uma função `maltes/2` que recebe como argumento os 2 dados e devolve o resultado.

¹Um indivíduo que por algum dos seus *costados* (algum dos seus *ascendentes*, ou *antecessores*) tem direito ao uso de *brasão*.

²Dois Cavaleiros de Malta há mais de 5 anos.

Os Dados

Os 2 dados do problema, passados como argumentos ao programa a desenvolver, são:

- a *Árvore Genealógica* do indivíduo que se quer propor, representada através do functor `ag/3` que contém o código do indivíduo na raiz e as árvores genealógicas do pai e da mãe, ou `vazio`;
- o *Livro de Nobreza*, representado como uma lista ordenada por ordem alfabética dos códigos dos indivíduos. Cada indivíduo será representado através de um functor `ind/4` que contém o código, o ano de nascimento, o tipo de filiação (l,i) e o título de nobreza (b,c,m,d,nil).

Os Resultados

O resultado de invocar o predicado ou função `maltes` é um valor de verdade (valor do tipo `bool`) que indica se o nobre na raiz da árvore satisfaz uma das 2 condições para poder ser um CM.

Exemplo (Prog. Lógica)

```
?- maltes( ag( 1, ag( 2a, ag(3a, ag(4a, ag(5a, ag(6a, ag(7a, ag(8a, vazio, vazio),
                                     vazio), vazio), vazio), vazio),
          ag(5b, ag(6b, vazio, vazio), vazio)),
          vazio),
          ag(3b, vazio,
          ag(4d, ag(5g, ag(6c, ag(7b, ag(8b, vazio, vazio),
                                     vazio), vazio), vazio),
          ag(5h, vazio, ag(6d, vazio, vazio) ))),
          ag( 2b, ag(3c, ag(4e, ag(5i, vazio, vazio),
          ag(5j, vazio, vazio)),
          ag(4f, vazio, vazio)),
          ag(3d, vazio,
          ag(4h, ag(5o, ag(6e, ag(7c, ag(8c, vazio, vazio),
                                     vazio), vazio), vazio),
          ag(5p, vazio, vazio) )) ),
          [ ind(1, 1975,1,nil), ind(2a,1955,1,nil), ind(2b,1955,1,nil),
            ind(3a,1925,1,nil), ind(3b,1928,1,nil), ind(3c,1930,1,nil), ind(3d,1935,1,nil),
            ind(4a,1900,1,b), ind(4d,1905,1,nil), ind(4e,1903,1,nil), ind(4f,1900,i,nil),
            ind(4h,1915,1,nil),
            ind(5a,1875,1,nil), ind(5b,1878,i,nil), ind(5g,1880,1,nil), ind(5h,1885,1,nil),
            ind(5i,1875,1,nil), ind(5j,1877,1,nil), ind(5o,1880,1,nil), ind(5p,1890,1,nil),
            ind(6a,1850,1,nil), ind(6b,1844,1,nil), ind(6c,1840,i,nil),
            ind(6d,1865,1,nil), ind(6e,1850,1,nil),
            ind(7a,1830,1,nil), ind(7b,1810,1,c), ind(7c,1825,1,d),
            ind(8a,1800,1,m), ind(8b,1796,1,nil), ind(8c,1805,1,nil) ] ).
yes
```

Neste exemplo, o predicado sucede porque se verifica a primeira condição.

Exemplo (Prog. Funcional)

Neste caso a função retorna verdadeiro porque se verifica a segunda condição.

```
> maltes ag( 1, ag( 2a, ag(3a, ag(4a, ag(5a, vazio,
                                ag(6a, ag(7a, vazio,
                                    ag(8a, vazio, vazio) )),
                                vazio),
                                ag(5b, ag(6b, vazio, vazio), vazio)),
            vazio),
            ag(3b, vazio,
                ag(4d, ag(5g, ag(6c, ag(7b, ag(8b, vazio, vazio),
                                        vazio), vazio), vazio),
                    ag(5h, vazio, ag(6d, vazio, vazio) )))),
            ag( 2b, ag(3c, ag(4e, ag(5i, vazio, vazio),
                            ag(5j, vazio, vazio)),
                ag(4f, vazio, vazio)),
            ag(3d, vazio,
                ag(4h, ag(5o, ag(6e, ag(7c, ag(8c, vazio, vazio),
                                        vazio), vazio), vazio),
                    ag(5p, vazio, vazio) )))
)
[ ind(1, 1975,1,nil), ind(2a,1955,1,nil), ind(2b,1955,1,nil),
  ind(3a,1925,1,nil), ind(3b,1928,1,nil), ind(3c,1930,1,nil), ind(3d,1935,1,nil),
  ind(4a,1900,i,nil), ind(4d,1905,1,nil), ind(4e,1903,1,b), ind(4f,1900,i,d),
  ind(4h,1915,1,nil),
  ind(5a,1875,1,c), ind(5b,1878,i,nil), ind(5g,1880,1,nil), ind(5h,1885,1,nil),
  ind(5i,1875,1,nil), ind(5j,1877,1,nil), ind(5o,1880,1,nil), ind(5p,1890,1,m),
  ind(6a,1850,1,nil), ind(6b,1844,1,nil), ind(6c,1840,i,nil),
  ind(6d,1865,1,nil), ind(6e,1850,1,nil),
  ind(7a,1830,1,nil), ind(7b,1810,1,c), ind(7c,1825,1,d),
  ind(8a,1800,1,m), ind(8b,1796,1,nil), ind(8c,1805,1,nil) ]
> true
```

O SEU A SEU DONO

Introdução

A Miss Teresa, que é professora do ensino básico, costuma dar uma lembrança a cada um dos seus alunos quando estes terminam o 4o ano. Para isso ela começa por elaborar uma lista de lembranças com pelo menos uma lembrança por cada menino. Depois escolhe uma lembrança diferente para cada menino.

Tendo constatado que os meninos muitas vezes ficam a cobiçar as prendas uns dos outros, a Miss Teresa decidiu este ano introduzir uma alteração que permita levar em conta os gostos dos meninos. Forneceu então a cada menino a lista das lembranças, tendo assinalado a lembrança que tinha escolhido para ele. Cada menino preencheu a sua eventual preferência por outras lembranças, indicando-as por ordem decrescente de preferência.

A Miss Teresa estabeleceu ainda uma ordem total entre os meninos tendo em conta o seu aproveitamento ao longo do ano (não há portanto dois meninos com o mesmo aproveitamento).

Tarefa

A sua tarefa consiste em escrever um programa lógico ou funcional que permita à Miss Teresa determinar que lembrança deve dar a que menino, de forma a que:

- nenhum menino fique sem lembrança – não sendo possível atribuir uma lembrança que o menino prefira, o menino fica com a lembrança atribuída inicialmente pela Miss;
- nenhum menino fica com uma prenda x **diferente da inicialmente estabelecida**, se outro menino com melhor aproveitamento preferir essa mesma prenda x relativamente à que lhe foi atribuída.

Exemplo: Suponha que a turma tem 4 meninos ordenados da seguinte forma: $A < B < C < D$ (A é o que teve o melhor aproveitamento) e a Miss tem uma lista com 5 prendas, designadas por 1,2,3,4,5. Suponha ainda que:

- ao menino A, a Miss atribuiu a lembrança 2 mas ele preferia a 4 e mais ainda a 3
- ao menino B, a Miss atribuiu a lembrança 3 e ele não prefere nenhuma outra
- ao menino C, a Miss atribuiu a lembrança 1 mas ele preferia a 5 e mais ainda a 4
- ao menino D, a Miss atribuiu a lembrança 4 mas ele preferia a 5

Nesta situação, a atribuição A-4 B-3 C-1 D-5 não é boa já que não respeita as condições anteriores. Isto acontece porque o menino D tem uma prenda diferente da inicialmente estabelecida pela Miss (a 5) e esta é preferida por um menino com melhor aproveitamento: o menino C prefere a 5 à 1! A boa atribuição, i.e., a que está de acordo as condições anteriores, é A-2 B-3 C-5 D-4. Isto porque o único menino que não tem a prenda inicialmente estabelecida pela Miss é o C, mas a prenda 5 não é preferida por nenhum dos meninos com melhor aproveitamento.

Se optar pela Programação Lógica, implemente o programa através de um predicado `lembrancas/3` em que os 2 primeiros argumentos são os dados e o último trás o resultado. Se escolher a Programação Funcional, desenvolva uma função `lembrancas/2` que recebe como argumento os 2 dados e devolve o resultado.

Os Dados

Os 2 dados do problema, passados como argumentos ao programa a desenvolver, são:

- o numero de lembranças (pelo menos tantas quantos meninos)
- uma lista de listas de preferências de cada um dos meninos – cada uma destas listas tem as lembranças que o menino prefere por ordem decrescente de preferência, sendo a última (e eventualmente a única) lembrança de cada uma destas listas a escolha inicial da professora

onde meninos e lembranças são representados por números consecutivos a começar em 1, sendo que a numeração dos meninos reflecte o seu aproveitamento (o primeiro menino é o que teve melhor aproveitamento).

Os Resultados

O resultado de invocar o predicado ou a função `lembrancas` é uma lista representando a prenda a atribuir a cada um dos meninos – a posição i da lista tem a lembrança atribuída ao menino i .

Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido:

```
?-lembrancas(5, [[3,4,2], [3], [4,5,1], [5,4]], L).  
L = [2,3,5,4]
```

Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
> lembrancas 5 [[3,4,2], [3], [4,5,1], [5,4]]  
> [2,3,5,4]
```