Concurso Nacional de Programação Lógica e Funcional CeNPLf'06

Faculdade de Engenharia da Universidade do Porto

5–7 de Maio de 2006

Parte da tarde

(4 horas – 6 problemas)

Problemas

6 de Maio de 2006
15h00m – 19h00m

T7: A estranha pirâmide do Faraó RomZé II

Introdução

Diz a lenda que existiu, já na fase de decadência da Faraónia, um misterioso faraó, RomZé II, que era divinamente alto e teimoso e que gostava do passatempo de mostrar aos comuns dos mortais o quão longe estariam da sua divinal estatura. Como muitos dos outros faraós, outro passatempo era mandar construir pirâmides. Claro que RomZé II queria que a sua próxima pirâmide fosse a mais alta e inatingível de todas: mais alta em altura total; e inatingível porque só mesmos os deuses (como ele, claro!) conseguiriam chegar ao topo.

O faraó mandou vir de todo o país os melhores arquitectos, engenheiros civis e sacerdotes (claro, estes são sempre necessários!) para realizar o seu sonho. Aos arquitectos, pediu que desenhassem uma pirâmide vistosa e seguindo os princípios do urbanismo da época, aos engenheiros civis, pediu para organizar todo o processo de construção (materiais e escravos), e aos sacerdotes que dissessem as medidas divinais. Estes últimos estudaram bem as dimensões do faraó RomZé II (pés, pernas, cabelos), e, junto com os astros decidiram que:

- A pirâmide terá que ter uma altura total (do ponto mais alto até à base) determinada, **H**;
- A pirâmide terá que ter na base um número de blocos fixo: **T**. Os blocos terão forma de paralelipípedo.
- O desnível dos degraus aumenta em cada nível, da base até ao topo.
- Para cada bloco de um nível superior, com altura ao solo **h**, existirá sempre um número constante, **N**, de blocos que o suportam. Cada um destes blocos terá exactamente uma altura ao solo igual a $\frac{h}{N+5}$.
- Cada bloco suportará unicamente um bloco do nível acima.
- Todas as alturas ao solo encontradas serão números inteiros.
- A altura do nível mais baixo é unitária (1).

Por exemplo, na figura 1, temos uma pirâmide com altura **H**=729 e **T**=64 blocos na base. Pode-se ver também que tem 4 níveis e que o valor de **N** é 4, pois cada bloco tem 4 outros blocos a suportá-lo (excepto para os da base). A sequência de alturas é então a seguinte:

$h_3 = 729$

$h_2 = \frac{729}{4+5} = 81$

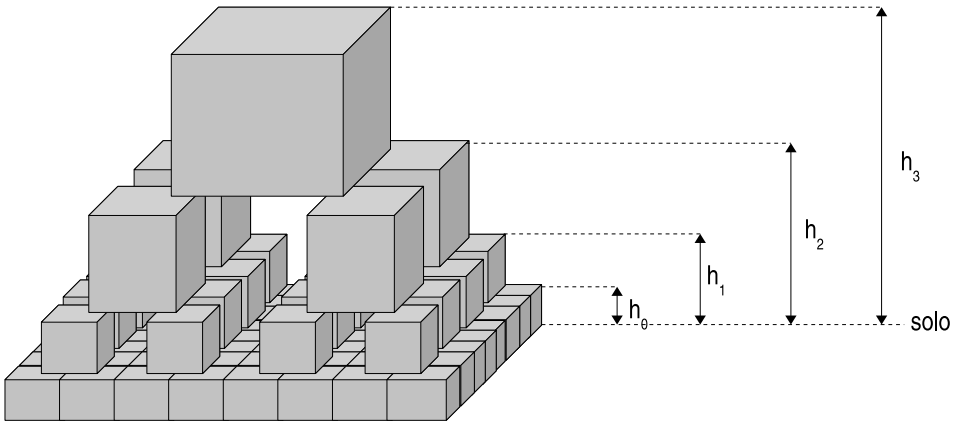

Figura 1: A pirâmide pensada pelos arquitectos do faraó

$h_1 = \frac{81}{4+5} = 9$

$h_0 = \frac{9}{4+5} = 1$

O faraó ordenou que se lançasse aos crocodilos do Rio Nilo toda a equipa caso a pirâmide não fosse contruída o mais depressa possível e respeitando as normas descritas. O problema é descobrir quantos níveis (da base ao topo) serão necessários no total para construir a dita pirâmide. Em pânico, os sacerdotes conseguiram pedir aos deuses que enviassem uma equipa concorrente do CeNPLf'2006 para lhes dar a solução... Caso o consigam resolver, terão pronta uma máquina do tempo para viajar para o tempo de RomZé II... mas não se esqueçam da escova de dentes!

Tarefa

A sua tarefa consiste em escrever um programa que receba a altura total, H, bem como o número de blocos, T, que existirá na base. Deverá devolver o número de níveis necessários

Se optar pela Programação em Lógica, implemente o programa através de um predicado `romze/3` em que os 2 primeiros argumentos são os dados e o último terá o resultado.

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

Os Dados

Assuma que a altura máxima, **H**, não excederá 100000 e que o número de blocos na base, **T**, não excederá os 10000. Ambos serão sempre inteiros positivos superiores a 1.

Pode assumir que cada caso de teste tem sempre uma e uma só solução.

Os Resultados

O resultado deverá ser um número inteiro, indicando o número de níveis que a pirâmide terá.

1 Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido:

```
?- romze(1024, 729, L).

L = 3;
no

?- romze(100000, 3125, L).

L = 6;
no
```

2 Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
> romze 729 64

4

> romze 11 6

2
```

T8: Código de Gray-Askores

Introdução

Código de Gray

Um código de Gray é um código cíclico binário de distância unitária. Por outras palavras, é uma sequência de 2^n números binários de n bits, tal que entre cada dois números consecutivos apenas 1 bit muda. Como é um código cíclico, considera-se que o último número da sequência precede o primeiro.
Por exemplo, um código de Gray para $n = 3$ é:

000 001 011 010 110 111 101 100

Estes códigos são usados há muito tempo em codificadores mecânicos, dado que uma pequena mudança de posição nunca afecta mais do que um bit, o que limita muito os efeitos de erros de desalinhamento. Se se usasse um código binário convencional, um pequeno erro na fronteira entre dois números (p.ex., 3 e 4, codificados como 011 e 100) poderia resultar em transições bruscas na descodificação (p.ex., $3 \rightarrow 7 \rightarrow 5 \rightarrow 4$ em vez de $3 \rightarrow 4$). Estes códigos têm também sido muito utilizados em Comunicações[1].

Código de Gray para RGB

Zekit Askores, investigador ilustre e famoso, propôs recentemente uma variante do Código de Gray que promete revolucionar a indústria. A diferença está em que este código não é formado por números binários, mas sim por palavras formadas por letras r, g e b.
Conhecido como Código de Gray-Askores, é também cíclico e de distância unitária. Define-se como uma sequência de 3^c palavras de c letras do conjunto $\{r, g, b\}$ tal que entre cada duas palavras consecutivas apenas uma letra muda.
Por exemplo, um código de Gray-Askores para $c = 2$ é:

rr rg rb gb gr gg bg bb br

É fácil verificar que para um mesmo c existem vários códigos diferentes. Por exemplo, um outro código para $c = 2$ é o seguinte:

rr rg rb gb bb bg gg gr br

A Figura 1 ilustra a aplicação deste código a um codificador circular.

[1] A primeira utilização conhecida de um código de Gray deve-se ao engenheiro francês Émile Baudot, num telégrafo demonstrado em 1878. Foram patenteados apenas em 1953, por Frank Gray, investigador da Bell Labs.

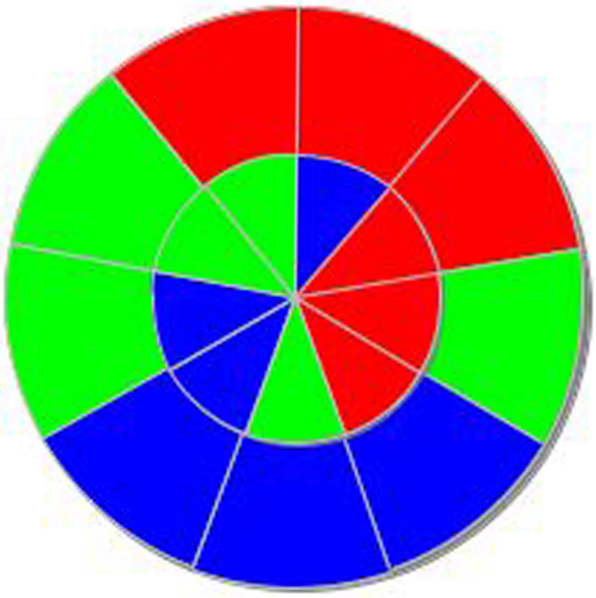

Figura 2: Codificador circular

Tarefa

A sua tarefa consiste em escrever um programa que produza um Código de Gray-Askores para um dado número de letras de cada palavra.

Se optar pela Programação Lógica, implemente o programa através de um predicado `gray/2` em que o primeiro argumento é o número de letras de cada palavra e o segundo devolve o resultado.
Se escolher a Programação Funcional, desenvolva uma função `gray/1` que recebe como argumento o número de letras de cada palavra e devolve o resultado.

Os Dados

Assuma que o número c de letras que constituem cada palavra do código que se pretende produzir é um inteiro positivo igual ou superior a 1 e inferior a 10.

Os Resultados

O resultado será apresentado sob a forma de uma lista de palavras, cada uma das quais será representada por uma lista de c letras do conjunto $\{r, g, b\}$.

1º Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido:

```
?- gray(3, C).

C = [[r, r, r], [r, r, g], [r, r, b], [r, g, b], [r, b, b], [r, b, g],
     [r, g, g], [r, g, r], [r, b, r], [g, b, r], [b, b, r], [b, g, r],
     [g, g, r], [g, g, g], [b, g, g], [b, b, g], [g, b, g], [g, b, b],
     [b, b, b], [b, g, b], [g, g, b], [g, r, b], [b, r, b], [b, r, g],
     [g, r, g], [g, r, r], [b, r, r]]
```

2º Exemplo (Prog. Funcional)

Mostra-se em seguida um outro exemplo ilustrativo do programa pretendido:

```
> gray 4

[[r, r, r, r], [r, r, r, g], [r, r, r, b], [r, r, g, b], [r, r, b, b],
 [r, r, b, g], [r, r, g, g], [r, r, g, r], [r, r, b, r], [r, g, b, r],
 [r, b, b, r], [r, b, g, r], [r, g, g, r], [r, g, g, g], [r, b, g, g],
 [r, b, b, g], [r, g, b, g], [r, g, b, b], [r, b, b, b], [r, b, g, b],
 [r, g, g, b], [r, g, r, b], [r, b, r, b], [r, b, r, g], [r, g, r, g],
 [r, g, r, r], [r, b, r, r], [g, b, r, r], [b, b, r, r], [b, g, r, r],
 [g, g, r, r], [g, g, r, g], [b, g, r, g], [b, b, r, g], [g, b, r, g],
 [g, b, r, b], [b, b, r, b], [b, g, r, b], [g, g, r, b], [g, g, g, b],
 [b, g, g, b], [b, b, g, b], [g, b, g, b], [g, b, b, b], [b, b, b, b],
 [b, g, b, b], [g, g, b, b], [g, g, b, g], [b, g, b, g], [b, b, b, g],
 [g, b, b, g], [g, b, g, g], [b, b, g, g], [b, g, g, g], [g, g, g, g],
 [g, g, g, r], [b, g, g, r], [b, b, g, r], [g, b, g, r], [g, b, b, r],
 [b, b, b, r], [b, g, b, r], [g, g, b, r], [g, r, b, r], [b, r, b, r],
 [b, r, g, r], [g, r, g, r], [g, r, g, g], [b, r, g, g], [b, r, b, g],
 [g, r, b, g], [g, r, b, b], [b, r, b, b], [b, r, g, b], [g, r, g, b],
 [g, r, r, b], [b, r, r, b], [b, r, r, g], [g, r, r, g], [g, r, r, r],
 [b, r, r, r]]
```

T9: Derivadas de um polinómio num ponto

Introdução

Os polinómios são funções da classe C^∞, isto é, são funções contínuas e diferenciáveis com continuidade o número de vezes que se desejar.

Tarefa

A sua tarefa consiste em escrever um programa que dado um polinómio

$$p(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n \,,$$

calcule o valor da k-ésima derivada de $p(x)$ num dado ponto z, isto é, devolva o valor $p^{(k)}(z)$.
Se optar pela Programação Lógica, implemente o programa através de um predicado `valor/4` em que o primeiro argumento é a lista $[a_0, a_1, \ldots, a_n]$ dos coeficientes do polinómio $p(x)$; o segundo argumento um inteiro não negativo k; o terceiro um número real z; e o último trás o resultado $p^{(k)}(z)$.
Se escolher a Programação Funcional, desenvolva uma função `valor/3` que recebe como argumento os 3 dados indicados e devolve o resultado.

Os Dados

Não existem mais dados além dos já indicados.

Os Resultados

O resultado de invocar o predicado ou a função `valor` é o valor da k-ésima derivada de $p(x)$ no ponto z.

1º Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido (o valor da segunda derivada de $p(x) = 3x^2 + 5$ no ponto 0 é 6):

```
?- valor([3,0,5],2,0,V).

V = 6
```

2º Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido (o valor da terceira derivada de $6x^5 - x^3 + 4x^2$ no ponto 1 é 354):

```
> valor([6,0,-1,4,0,0],3,1);

354
```

T10: No Bom Caminho

Introdução

Os gráficos vectoriais são uma forma de representar informação gráfica [sic] que pode manter estrutura e em certos casos pode ter vantagens em termos de compacidade, quando comparada com uma representação "raster." Algumas linguagens de programação dispõem dum dispositivo adequado para produzir gráficos vectoriais: a **tartaruga**, inicialmente popularizada pela linguagem *Logo*.

Tarefa

A sua tarefa consiste em escrever um programa que, dada uma sequência de instruções de desenho vectorial, produza uma sequência equivalente mas, **sempre que possível, mais curta**.

- Para preservar a declaratividade dos gráficos e da linguagem, as instruções permitidas só podem ser as seguintes:
  - `sobe`, `desce`: levanta e baixa, respectivamente, a "caneta" que a tartaruga segura.
  - `anda`: faz a tartaruga avançar 1 passo.
  - `vira`: a tartaruga vira 90 graus no sentido dos ponteiros dum relógio.
- Assume-se que o desenho é efectuado num quadrado de 20x20 e que a tartaruga se encontra inicialmente no canto inferior esquerdo, virada para cima e com a caneta no ar ("em cima.")
- Sempre que a tartaruga "bater numa parede," é como se a instrução `anda` tivesse sido ignorada.

Se optar pela Programação Lógica, implemente o programa através de um predicado `caminha/2` em que o primeiro argumento é uma lista (instanciada) e o último será unificado com o resultado. Se escolher a Programação Funcional, desenvolva uma função `caminha` que recebe como argumento uma lista de instruções e devolve como resultado outra lista, equivalente à primeira mas mais simples.

Os Dados

Os dados do problema, passados como argumento ao programa a desenvolver, são simplesmente a lista de instruções a simplificar.

Os Resultados

O resultado de invocar o predicado ou a função `caminha` é simplesmente uma lista semelhante à de entrada mas simplificada.
A qualidade da simplificação servirá para desempatar os programas correctos.

1º Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido:

```
?- caminha([anda, vira, vira, vira, vira, anda, desce, anda], X).

X = [anda, anda, desce, anda]
```

2º Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
> (caminha '(sobe vira vira anda anda vira vira desce
           anda vira vira vira vira anda desce anda))

(desce anda anda anda)
```

T11: oBIamrpaelhrfaemietnot

Introdução

O João passa os dias no bar da faculdade a jogar às cartas com os colegas em vez de ir às aulas. Graças a isso, já quase consegue baralhar as cartas como um profissional: num baralhamento perfeito, o baralho é dividido exactamente ao meio e as cartas das duas metades são intercaladas, começando com uma carta da metade inferior; quando o João baralha, acontece por vezes ficarem duas cartas consecutivas trocadas.

Por exemplo, numerando as cartas de `0` a `51`, depois de um baralhamento perfeito, a ordem das cartas é:

```
26 0 27 1 28 2 29 3 30 4 31 5 ... 50 24 51 25
```

No caso caso do João, o resultado pode ser:

```
26 0 27 1 28 2 29 30 3 4 31 5 ... 50 24 51 25
```

em que as cartas `3` e `30` ficaram trocadas.

Como qualquer bom profissional, o João sabe que não se deve baralhar demais e nunca baralha mais de dez vezes as cartas.

Tarefa

A sua tarefa consiste em escrever um programa que receba a lista com as cartas baralhadas e determine quantas vezes é que o João baralhou as cartas e quais os erros que cometeu. Note que o João comete no máximo um erro de cada vez que baralha, mas pode não cometer erro nenhum. Os erros devem ser identificados através do número do baralhamento em que ocorreram e da posição da primeira carta que foi trocada. Assume-se que nenhuma carta é afectada por mais do que um erro.

Se optar pela Programação em Lógica, implemente o programa através de um predicado `baralhado/3` em que o primeiro argumento é a lista das cartas (uma permutação dos naturais de `0` a `51`), o segundo argumento devolve o número de vezes que o baralho foi baralhado (entre `0` e `10`) e o terceiro argumento devolve uma lista de pares `(B,P)` em que `B` é o número do baralhamento e `P` é a posição da primeira carta trocada.

Se escolher a Programação Funcional, desenvolva uma função `baralhado` que receba como argumento a lista das cartas (uma permutação dos naturais de `0` a `51`) e devolva um par em que o primeiro elemento é o número de vezes que o baralho foi baralhado (entre `0` e `10`) e o segundo elemento é uma lista de pares `(B P)` em que `B` é o número do baralhamento e `P` é a posição da primeira carta trocada.

1º Exemplo (Prog. Lógica)

Seguem-se exemplos ilustrativos do programa pretendido:

```
?- baralhado([26, 0, 27, 1, 28, 2, 29, 30, 3, 4, 31, 5, 32, 6, 33,
              7, 34, 8, 35, 9, 36, 10, 37, 11, 38, 12, 39, 13, 40,
              14, 41, 15, 42, 16, 43, 17, 44, 18, 45, 19, 46, 20,
              47, 21, 48, 22, 49, 23, 50, 24, 51, 25],S,L).

S = 1
L = [(1, 8)]

?- baralhado([19, 39, 6, 26, 46, 13, 33, 0, 20, 40, 7, 27, 47, 14,
              34, 1, 21, 41, 8, 28, 48, 15, 35, 2, 22, 42, 9, 29,
              49, 16, 36, 3, 23, 43, 10, 30, 50, 17, 37, 4, 24, 44,
              11, 31, 51, 18, 38, 5, 25, 45, 12, 32],S,L).

S = 3
L = []
```

2º Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
> baralhado [49 26 43 40 37 34 31 28 25 22 19 16 13 10 7 4 1
             51 48 45 42 39 36 33 24 27 30 21 18 15 12 9 6 3 0
             50 47 44 41 38 35 32 29 46 23 20 17 2 11 8 5 14]

[9 [(3 4) (7 12) (8 39)]]
```

T12: Whitoff

Introdução

O Whitoff é um jogo para dois jogadores, que é jogado num tabuleiro quadrado e consiste em movimentar uma peça um qualquer número de casas na horizontal, vertical ou diagonal mas sempre para cima e para a esquerda. Inicialmente, um jogador coloca a peça e o outro movimenta-a. Seguidamente, os jogadores vão movimentando a peça alternadamente. O jogador que colocar peça no canto superior esquerdo é o vencedor. A figura 3 mostra as direcções admitidas.

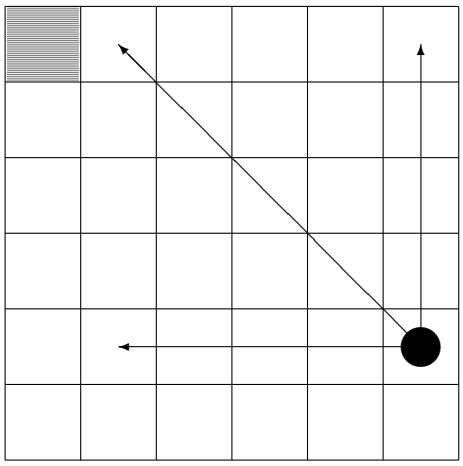

Figura 3: Movimentos possíveis no Whitoff

Este jogo, tal como muitos outros, possui uma estratégia vencedora. Quem souber qual é consegue sempre ganhar desde que jogue primeiro. Existe um conjunto de casas que dão sempre a vitória ao jogador que lá colocou a peça. Por exemplo, a casa $(1, 1)$ é uma dessas casas: se o jogador colocar lá a peça o adversário perdeu o jogo. Da mesma forma, a casa $(2, 3)$ também é uma casa *vencedora*[2].

Tarefa

Pretende-se que desenvolva um programa que escreva todas as casas *vencedoras* num tabuleiro de $n \times n$. Este programa deverá receber como argumento a dimensão do tabuleiro e devolver a lista das casas *vencedoras*. Cuidado que este problema é bastante simples mas pretende-se uma implementação que funcione bem mesmo com tabuleiros de grande dimensão (calcule o resultado em no máximo 20 segundos para um tabuleiro de 50×50).

Se optar pela Programação Lógica, implemente o programa através de um predicado `whitoff/2` em que o primeiro argumento é o tamanho do tabuleiro e o último será instânciado com a lista das posições *vencedoras*.

[2]Descubra porquê.

Se escolher a Programação Funcional, desenvolva a função
`whitoff :: Integer -> [(Integer, Integer)]`
que recebe como argumento o tamanho do tabuleiro e devolve a lista de posições *vencedoras*.

Os Dados

Os dados deste problema são a dimensão do tabuleiro.

Os Resultados

O resultado da invocação da função ou predicado `whitoff` é uma lista de pares com as coordenadas das posições *vencedoras*.

Exemplo

A sintaxe deste exemplo depende obviamente da linguagem de programação utilizada. Segue-se dois exemplos, um que demorou 8 segundos e outro que demorou 13 segundos num pentium IV a 3.4 GHz.

```
?- whitoff(45, Res).

Res = [(1,1),(2,3),(3,2),(4,6),(6,4),(5,8),(8,5),(7,11),(11,7),
       (9,14),(14,9),(10,16),(16,10),(12,19),(19,12),(13,21),(21,13),
       (15,24),(24,15),(17,27),(27,17),(18,29),(29,18),(20,32),(32,20),
       (22,35),(35,22),(23,37),(37,23),(25,40),(40,25),(26,42),(42,26),
       (28,45),(45,28)]

> whitoff 50

[(1,1),(2,3),(3,2),(4,6),(6,4),(5,8),(8,5),(7,11),(11,7),(9,14),
 (14,9),(10,16), (16,10),(12,19),(19,12),(13,21),(21,13),(15,24),
 (24,15),(17,27),(27,17),(18,29), (29,18),(20,32),(32,20),(22,35),
 (35,22),(23,37),(37,23),(25,40),(40,25),(26,42), (42,26),(28,45),
 (45,28),(30,48),(48,30),(31,50),(50,31)]
```