Resolução Exame de IA – Época Normal (Moodle) 2018/19

Pergunta **1**

1) Pesquisa de Soluções (5 Valores)

Um puzzle é constituído um tabuleiro com três peças vermelhas (“V”), três peças brancas (“B”) e um espaço vazio (“x”). O objetivo final consiste em trocar a posição das peças vermelhas e brancas de modo a que as peças vermelhas fiquem na zona direita e as brancas na zona esquerda. O estado inicial típico do puzzle e o estado final desejado são apresentados em seguida:

Existem três métodos legais de movimentar uma peça, cada um com um custo associado, ou seja:

- m1) Movimentar uma peça para uma casa livre adjacente, custo 1;
- m2) Saltar sobre uma peça (de qualquer cor) para uma casa livre, custo 2;
- m3) Saltar sobre duas peças (de quaisquer cores) para uma casa livre, custo 2.

As peças vermelhas só se podem movimentar da esquerda para a direita e as peças brancas no sentido oposto (i.e. da direita para a esquerda). Pretende-se encontrar a solução com menor custo.

1.1) Formule o problema como um problema de pesquisa, indicando o modo de representação do estado, estado inicial, teste objetivo, operadores (nome, pré-condições e efeitos) e função de custo.

A representação de um estado poderá ser um vetor/array com 7 entradas, em que cada índice representa uma das 7 casas do tabuleiro e cujo valor nesse índice representa a peça que se localiza nessa posição do tabuleiro. Cada índice poderá, portanto, ter os valores **V** (peça vermelha), **x** (ausência de peça, casa vazia) ou **B** (peça branca).

O estado inicial será [V, V, V, x, B, B, B].

O teste objetivo será [B, B, B, x, V, V, V].

Os operadores poderão ser os seguintes:

- Nome: m1V

Pre-condições: peça ao qual é aplicado é Vermelha (valor V) e a casa imediatamente à sua direita (seja i o índice da peça, a casa i+1) deve estar vazia (valor x), i deve ser menor do que 7

Efeitos: O índice i passa a ter o valor x e o índice i + 1 passa a ter o valor V, tem um impacto de custo 1 na solução final

- Nome: m1B

Pre-condições: peça ao qual é aplicado é Branca (valor B) e a casa imediatamente à sua esquerda (seja i o índice da peça, a casa i-1) deve estar vazia (valor x), i deve ser maior do que 0

Efeitos: O índice i passa a ter o valor x e o índice i - 1 passa a ter o valor B, tem um impacto de custo 1 na solução final

- Nome: m2V

Pre-condições: peça ao qual é aplicado é Vermelha (valor V) e a casa de índice i + 2 (seja i o índice da peça) deve estar vazia (valor x), i deve ser menor do que 6

Efeitos: O índice i passa a ter o valor x e o índice i + 2 passa a ter o valor V, tem um impacto de custo 2 na solução final

- Nome: m2B

Pre-condições: peça ao qual é aplicado é Branca (valor B) e a casa de índice i -2 (seja i o índice da peça) deve estar vazia (valor x), i deve ser maior do que 1

Efeitos: O índice i passa a ter o valor x e o índice i - 2 passa a ter o valor B, tem um impacto de custo 2 na solução final

- Nome: m3V

Pre-condições: peça ao qual é aplicado é Vermelha (valor V) e a casa de índice i +3 (seja i o índice da peça) deve estar vazia(valor x), i deve ser menor do que 5

Efeitos: O índice i passa a ter o valor x e o índice i + 3 passa a ter o valor V, tem um impacto de custo 3 na solução final

- Nome: m3B

Pre-condições: peça ao qual é aplicado é Branca (valor B) e a casa de índice i - 3 (seja i o índice da peça) deve estar vazia (valor x), i deve ser maior do que 2

Efeito: O índice i passa a ter o valor x e o índice i -3 passa a ter o valor B, tem um impacto de custo 3 na solução final

A função de custo, a qual queremos minimizar, tem o valor igual à soma do custo de todos os movimentos efetuados para chegar ao estado objetivo (o custo de aplicação de cada operador foi previamente mencionado nos efeitos dos mesmos).

Pergunta **2**

1.2) Indique se as seguintes heurísticas para o algoritmo A* são admissíveis, justificando devidamente:

a) h1 = 0

b) h2 = 5

c) h3 = soma do número de Bs nas células 5, 6 e 7 com o número de Vs nas células 1, 2 e 3.

Para a heurística **hx** ser admissível, nunca pode sobrestimar o custo da solução ótima a partir do estado em questão (h(n) tem que er menor ou igual do que caminho_ótimo(n), a partir de n).

a) h1 = 0 é sempre admissível, pois nunca sobrestima o valor da solução ótima (tem valor demasiado baixo!)

b) h2 = 5 não é admissível, por exemplo seja o estado atual for [B, B, B, V, x, V, V], em que resta apenas um movimento de custo 1 para chegar ao estado objetivo (mover a peça no índice central 1 casa para a direita), a heurística com valor 5 sobrestima o custo do caminho ótimo a partir desse estado (5 não é menor ou igual do que 1)

c) h3 é admissível, pois nunca vai sobrestimar o custo da solução ótima! No limite da posição inicial, a heurística tem valor de 6, inferior ao custo da solução ótima a partir do estado inicial (são necessários, pelo menos, mais do que 6 movimentos!!). No limite de uma posição final,

como [B,B,B,V,x,V,V] ou [B,B,B,V,V,V,x] também não sobrestima, pois a heurística tem para ambos estes casos valor 0 e o custo para o ótimo é, respetivamente, 1 e 3.

Pergunta **3**

1.3) Defina, justificando, uma heurística melhor que lhe permita aplicar o Algoritmo A* de modo eficiente à resolução do problema. Indique também o seu modo de cálculo a partir da representação definida em 1.1) apresentando o respetivo pseudo-código.

O custo de efetuar um qualquer movimento nunca ultrapassa distância absoluta que percorre ao efetuar esse movimento (custo de mover para casa livre adjacente = 1, distância = 1, custo de saltar sobre uma peça para casa livre = 2, distância = 2 e custo de salto sobre duas peças para casa livre = 3, distância = 3). Por este motivo, o somatório de max(0, distância à casa objetivo mais próxima) nunca vai sobrestimar o custo da solução ótima. A distância à casa objetivo mais próxima é, para as peças vermelhas, 4-posição_peça_vermelha (peça vermelha em i=0 tem distância de 4) e, para as peças brancas, posição_peça_branca - 2 (peça branca em i = 6 tem distância de 4).

Por exemplo, a heurística poderia ser (em pseudocódigo/linguagem C) a seguinte:

```
heuristica(estado) {
    h = 0;
    for (i = 0; i < 7; i++) {
        if(estado[i] = V)  h += max(0, 4 - i);
        else if (estado[i] = B) h += max(0, i - 2);
    }
    return h;
}
```

Pergunta **4**

1.4) Supondo o seguinte estado inicial **[B B V B V V x]** apresente a sequência de nós gerados nas árvores de pesquisa criadas, até encontrar a solução do problema, utilizando:

c1) Pesquisa em Profundidade

c2) Pesquisa em Largura

Nota: Suponha que, na expansão de um nó, são gerados primeiro os movimentos m3 (saltos duplos), depois m2 (saltos) e finalmente m1. Dentro de cada tipo de movimento, suponha que são gerados, primeiro, os movimentos das peças V e depois das B. Em caso de empate suponha que são expandidos, primeiro, os nós mais próximos da raiz da árvore e em caso ainda de empate, os nós mais à esquerda.

Profundidade

[B B V B V V x]

[B B V B x V V]

[B B V B V x V]

(usando [B B V B x V V])

[B B x B V V V]

(usando [B B x B V V V])

[B B B x V V V] -> estado objetivo

Largura:

[B B V B V V x]

[B B V B x V V]

[B B V B V x V]

(usando [B B V B x V V])

[B B x B V V V]

(usando [B B V B V x V])

[B B x B V V V]

[B B V B x V V]

(usando [B B x B V V V])

[B B B x V V V] -> estado objetivo

Pergunta **5**

2) Otimização de Rotas de Visitas a Clientes (5 valores)

Suponha o seguinte problema de otimização em que o objetivo é definir a melhor rota para visitar todos os clientes uma única vez, saindo da Fábrica 1 e concluindo o trajeto na Fábrica 2. Como simplificação suponha que é possível viajar em linha reta entre dois pontos no mapa.

2.1) Formule este problema como um problema de otimização, definindo uma representação para a definição do problema (posições das fábricas e clientes), para a solução do problema e definindo a função de avaliação a utilizar e respetivo pseudo-código.

Definição enquanto problema de otimização:

Representação de uma solução: lista de clientes visitados (tamanho 6 portanto). Por exemplo: [2, 4, 3, 1, 5, 6] significa que partiu da fábrica 1, visitou os clientes 2 4 3 1 5 e 6, e regressou à fábrica 2. (solução indicada na figura).

Cada cliente e cada fábrica tem uma posição no plano (x,y) associada e o custo de ir de um cliente ou fábrica para outro cliente ou fábrica é dado como a distância cartesiana entre as suas 2 posições.

O custo de uma solução é dado pela soma das distâncias percorridas (desde que sai da fábrica 1 e visita cada cliente até que chega à fábrica 2). Pretende-se minimizar este custo.

A função de avaliação é portanto dada como:

f(x):-

```
custo = 0
for(i = 1; i <= clientesVisitados.size; i++)
```

```
    if(i = 1)
        custo += dist(fab1, clientesVisitados[i])
        custo += dist(clientesVisitados[i], clientesVisitados[i+1]
    elseif(i = clientesVisitados.size)
         custo += dist(clientesVisitados[i], fab2)
     else
         custo += dist(clientesVisitados[i], clientesVisitados[i+1])
  return custo
```

Pergunta **6**

2.2) Defina uma função de vizinhança simples que lhe permita aplicar o algoritmo Hill-Climbing na resolução deste problema e apresente o respetivo pseudo-código.

Uma função de vizinhança simples é uma que troca qualquer 2 elementos da lista de clientes visitados.

```
v(clientesVisitados):-
    a = nrAleatorio(1, 6) // 6 = tamanho da lista de clientesVisitados
    b = nrAleatorio(1, 6) diferente de a
    novoClientesVisitados = clientesVisitados
    novoClientesVisitados[a] = clientesVisitados[b]
    novoClientesVisitados[b] = clientesVisitados[a]
    return novoClientesVisitados
```

Pergunta **7**

2.3) Apresente o código completo necessário para a aplicação do Hill-climbing na resolução deste problema, em pseudo-código ou numa linguagem de programação à sua escolha. Pode usar a variação que considerar melhor do algoritmo mas deve indicar qual a versão do hill-climbing utilizada. O código deve gerar uma solução inicial aleatória e utilizar como critério de paragem, 100 iterações seguidas sem melhoria da solução.

Variante Hill-Climbing clássico. Javascript em runtime:

```
function hill_climbing(fabricas, clientes){
// Escolha estado inicial.
let estado_atual = [];
for(let i = 0; i < 6; i++) {
let randomNum = [];
while(true){
randomNum = Math.round(Math.random()* 5);
if(estado_atual.includes(randomNum))
continue;
```

```
estado_atual.push(randomNum);
break;
}
}
// Variante.
for(let i = 0; i < 100; i++) {
let novoEstado = new_random_state(estado_atual);
if(eval(novoEstado, fabricas, clientes) < eval(estado_atual, fabricas, clientes)) { // MELHOR.
estado_atual = novoEstado;
i = 0;
}
}
// Solução.
console.log("Solução com custo " + eval(estado_atual, fabricas, clientes));
for(let i = 0; i < 6; i++) {
console.log("Cliente " + (i+1) + ": " + estado_atual[i]);
}
}
```

Pergunta **8**

2.4) Utilize o código construído para resolver o problema para as seguintes instâncias, em que são indicadas as posições (x,y) das duas fábricas e dos seis clientes, indicando para cada uma qual a solução final obtida e o custo respetivo.

Problema 1) Fabricas: (1,1), (10,10) ; Clientes: (8,8), (4,4), (3,3), (7,7), (2,2), (9,9)

Problema 2) Fabricas: (1,1), (10,10) ; Clientes: (2,2), (2,8), (6,6), (1,6), (10,5), (5,8)

Problema 3) Fabricas: (1,1), (5,2) ; Clientes: (2,8), (1,4), (6,6), (6,1), (1,6), (5,8)

Código javascript:

```
hill_climbing([[1,1], [10,10]], [[8,8], [4,4], [3,3], [7,7], [2,2], [9, 9]]);

hill_climbing([[1,1], [10,10]], [[2,2], [2,8], [6,6], [1,6], [10,5], [5, 8]]);

hill_climbing([[1,1], [5,2]], [[2,8], [1,4], [6,6], [6,1], [1,6], [5, 8]]);


function hill_climbing(fabricas, clientes){
    let estado_atual = [];
    for(let i = 0; i < 6; i++) {
        let randomNum = [];
        while(true){
            randomNum = Math.round(Math.random()* 5);
            if(estado_atual.includes(randomNum))  continue;
            estado_atual.push(randomNum);
            break;
        }
    }
```

```
    for(let i = 0; i < 100; i++) {
        let novoEstado = new_random_state(estado_atual);
        if(eval(novoEstado, fabricas, clientes) < eval(estado_atual, fabricas, clientes)) {
            estado_atual = novoEstado;
            i = 0;
        }
    }
    console.log("Solução com custo " + eval(estado_atual, fabricas, clientes));
    for(let i = 0; i < 6; i++) {
        console.log("Cliente " + (i+1) + ": " + estado_atual[i]);
    }
}
function new_random_state(clientesVisitados) {
    let a = randomNum = Math.round(Math.random()* 5);
    let b;
    while(true) {
        b = Math.round(Math.random()* 5);
        if(a == b)
            continue;
        break;
    }
    let novaCopia = [];
    for(let i = 0; i < clientesVisitados.length; i++)
        novaCopia.push(clientesVisitados[i]);
    novaCopia[a] = clientesVisitados[b];
    novaCopia[b] = clientesVisitados[a];
    return novaCopia;
}
function eval(clientesVisitados, fabricas, clientes) {
    let custo = 0;
    for(let i = 0; i < clientesVisitados.length; i++) {
        if(i == 0){
            custo += dist(fabricas[0], clientes[clientesVisitados[0]]);
            custo += dist(clientes[clientesVisitados[0]], clientes[clientesVisitados[1]]);
        }
        else if(i == clientesVisitados.length - 1) {
```

```
            custo += dist(fabricas[1], clientes[clientesVisitados[i]]);
        }
        else custo += dist(clientes[clientesVisitados[i]], clientes[clientesVisitados[i+1]]);
    }
    return custo;
}
function dist(a, b){  return Math.sqrt((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1]));  }
```

Cada cliente esta na gama [0, 5], i.e., o cliente 0 na implementação de javascript corresponde ao cliente numero 1 do problema.
1 - Solução com custo 12.727922061357859
Cliente 1: 4
Cliente 2: 2
Cliente 3: 1
Cliente 4: 3
Cliente 5: 0
Cliente 6: 5
2- Solução com custo 22.132560768607995
Cliente 1: 0
Cliente 2: 3
Cliente 3: 1
Cliente 4: 5
Cliente 5: 2
Cliente 6: 4
3- Solução com custo 18.886349517372675
Cliente 1: 1
Cliente 2: 4
Cliente 3: 0
Cliente 4: 5
Cliente 5: 2
Cliente 6: 3

Pergunta **9**

Acha possivel criar um agente simples reflexo (agente reativo simples) para jogar o jogo do galo com garantia que este agente nunca perde neste jogo? Justifique explicando como implementaria o agente simples reflexo para este problema.

Sim, porque o jogo do galo é um problema com um espaço de estados reduzido. Por isso, um agente simples reflexo seria uma forma rápida de resolver o problema.

De forma a implementar este problema teria numa estrutura de dados a melhor jogada a partir do estado atual. O estado atual seria constituído pelas posições das peças no tabuleiro e também a peça que identifica o agente (X ou O).

Pergunta **10**

Considerando um espaço de estados em que todas as soluções têm custos diferentes, a estratégia de pesquisa do custo uniforme encontra sempre a mesma solução encontrada pelo A* usando uma heurística admissível? Justifique adequadamente.

O algoritmo de pesquisa uniforme é completo e garante solução ótima. O algoritmo A* é completo e garante encontrar solução ótima se a heurística h utilizada no cálculo de f = g + h for admissível (que é o caso em questão!). Como todas as soluções têm custos diferentes, só existe uma solução ótima. Como ambos os algoritmos irão, sob as condições mencionadas acima, encontrar a solução ótima, então a pesquisa de custo uniforme irá encontrar sempre a mesma solução (ótima!) que o algoritmo A*.

Pergunta **11**

Supondo a seguinte árvore de pesquisa em que cada arco apresenta o custo do operador correspondente, indique justificando, qual o nó expandido em seguida utilizando cada um dos seguintes métodos:

a) Pesquisa Largura

b) Pesquisa em Profundidade

c) Pesquisa de Custo Uniforme

d) Pesquisa Gulosa

e) Pesquisa A*

Resposta: C E D C G

a) Assumindo que o último nó expandido foi o nó B, o nó seguinte será o C (pois é o próximo nó no nível 2, e BFS explora todos os nós nível a nível)

b) Assumindo que o último nó expandido foi o nó B, o nó seguinte será o E (pois é o filho mais à esquerda do nó B), DFS explora sempre primeiro em profundidade

c) O nó D, pois é a nó-folha da árvore com menor valor g (g = 3), pesquisa custo uniforme pesquisa primeiro nos com menor valor g

d) o nó C, pois pesquisa gulosa explora sempre primeiro os nós com menor valor de heurística, C tem h=1

e) o nó G, pois pesquisa A* explora sempre primeiro os nós com menor valor f = g + h, nó G tem f = g + h = 8 + 2 = 10

Pergunta **12**

Na aplicação do algoritmo minimax com cortes alfa-beta, explique que papel pode ter a ordenação dos nós gerados e avaliados pela função de avaliação.

Na aplicação do algorítmo minimax com alfa-beta prunning, a ordenação dos nós tem um forte impacto no número de cortes executados pelo algorítmo alfa-beta.

Seja **b** o fator de ramificação médio da árvore e **d** a sua profundidade:

Se a ordenação dos nós estiver na pior forma possível, não vai ser possível realizar quaisquer cortes alfa beta, sendo necessário visitar todos os nós da árvore (visitar b^d nós).

No entanto, se a ordenação dos nós gerados for feita na sua forma ótima, isto é, na forma em que o número de cortes alfa-beta é maximo, o número de nós visitados será muito menor, sendo apenas necessário visitar b^(d/2) nós, isto é, apenas cerca de metade da profundidade.

Se a ordenação for feita de forma aleatória, o número de nós visitados poderá ser menor do que no pior caso.

Pergunta **13**

Aplique o algoritmo minimax com cortes alfa-beta à seguinte àrvore, supondo que joga primeiro MAX, depois MIN e novamente MAX, indicando:

a) Qual o valor final dos Nós A, B, C e D (Nota: dado que são usados cortes alfa-beta o valor pode ser um número, ex: 8 ou uma condição ex: >=20 ou <=8)?

b) Quais os nós folha (de entre L a Y) que não chegam a ser avaliados pelo algoritmo minimax com cortes alfa-beta?

Assumindo que o Max é o primeiro a jogar,

a) A = 8 ; B = 3 ; C <= 1 ; D = 8

b) Não chegam a ser avaliados os nós O, Q, T, U e Y

Pergunta **14**

A determinada altura, na aplicação do algoritmo de otimização “Arrefecimento Simulado”, a função de avaliação indica que o estado atual vale 20. Gerou-se um estado sucessor com função de avaliação de valor 18. A temperatura atual é de 0.9. Calcule a probabilidade de o estado gerado ser aceite nos seguintes casos: a) O problema a resolver é um problema de maximização da função de avaliação. b) O problema a resolver é um problema de minimização da função de avaliação. Justifique.

No caso **a**, em que se trata de um problema de maximização, o estado vizinho gerado, que apresenta valor inferior ao atual, é pior do que o valor atual. Por esse motivo, a probabilidade de ele ser aceite será igual a e^(delta(E)/temperatura):

delta(E) = 18 - 20 = -2

probabilidade de novo estado ser aceite = e^(-2/0.9) = 0.1084 = 10.84%

Esta probabilidade é tanto maior quanto maior for a temperatura atual e tanto menor quanto maior for a diferença de valores entre o estado atual e o estado pior.

No caso **b**, em que se trata de um problema de minimização, o estado vizinho gerado, que apresenta valor inferior ao atual, é melhor do que o atual. Por esse motivo, ele é aceite, transitando o estado atual para esse novo estado.

Pergunta **15**

Caracterize o conceito de *overfitting* e indique que cuidados devem ser tidos em conta na aplicação de redes neuronais de modo a evitar este problema.

O conceito de *overfitting* surge quando o modelo que foi induzido se ajusta demasiado (sobreajuste) ao conjunto que foi utilizado para treino. No caso das redes neuronais, para estas serem capazes de generalizar com base nos inputs, deve ser utilizado um número de exemplos de treino elevado, na ordem de 10 vezes o número de graus de liberdade da rede neuronal. Este elevado número de exemplos de treino vai permitir forte diversidade no conjunto de treino e, consequentemente, diminuir o grau de *overfitting* na rede neuronal.

Pergunta **16**

Explique em que consistem e para que servem os métodos Holdout e Cross-Validation, distinguindo-os devidamente. Identifique um exemplo em que seja mais indicado utilizar Holdout e outro em que seja mais indicado utilizar Cross-Validation.

Os métodos Holdout e Cross-Validation servem para, no contexto de avaliação de modelos, efetuar a estimação da taxa de erro.

O método Holdout, utilizado quando há muitos exemplos (> 1000 exemplos, ordem dos milhares), consiste em dividir 2/3 dos dados existentes para a fase de treino e os restantes 1/3 para a fase de teste.

O método Cross-Validation, utilizado para conjuntos de tamanho intermédio (aproximadamente 1000 exemplos), consistem em dividir os dados em **k** partições disjuntas. Ao utilizar k-fold cross validation, o conjunto de dados é divido em k partições (k-folds) e em cada experiência (run), usa k-1 folds como conjunto de treino e a restante (1 fold, atual) como conjunto de teste (iterativamente). Ao utilizar leave-one-out cross validation, um caso específico de k-fold em que k = N, usa, em cada iteração-experiência, N-1 exemplos para conjunto de treino e o exemplo restante para conjunto de teste.

Por exemplo, num caso em que temos 15000 exemplos devermos utilizar Holdout (devido ao elevado número de exemplos, cross-validation seria computacionalmente muito pesado) e num caso em que temos cerca de 800 exemplos deveremos utilizar cross-validation.

Pergunta **17**

Considere a seguinte matriz de confusão que traduz os resultados de uma marca de teste de gravidez.

Resultado do Teste Estado	Teste Positivo	Teste Negativo
Grávida	45	5
Não grávida	20	30

Identifique os valores dos Verdadeiros Positivos (VP); Falsos Negativos (FN); Falsos Positivos (FP) e Verdadeiros Negativos (VN).
Calcule a taxa de acerto; a precisão; a sensibilidade e a medida F.

Verdadeiro Positivo = Grávida e deu teste positivo; VP=45
Verdadeiro Negativo = Não-Grávida e deu teste negativo; VN=30
Falso Positivo = Não-Grávida e deu teste positivo; FP=20
Falso Negativo = Grávida e deu teste negativo; FN=5

Taxa de Acerto = (VP+VN) / (VP+VN+FP+FN) = (45+30)/(45+30+20+5) = 0.75 = 75%
(Percentagem de Acerto do Teste de Gravidez – acertou em 75% dos casos)
Esta medida não funciona bem em conjuntos não balanceados)

Precisão = VP / (VP+FP) = 45 / (45 + 20) = 0.692 = 69.2%
(Percentagem de Casos realmente Corretos de entre os 65 Testes Positivos)

Sensibilidade = VP / (VP+FN) = 45 / (45 + 5) = 0.9 = 90%
(Percentagem de casos Corretos detetada pelo Teste de Gravidez de entre todas as Grávidas)

Medida F = 2TP / (2TP + FP + FN) = (2*45) / (2*45 + 5 + 20) = 0.783 = 78.3%
(Média harmónica da precisão e da sensibilidade)

Pergunta **18**

Construa uma DCG que permita validar sintática e semanticamente frases do tipo [X, subiu/desceu, de, N1, valor(es) para, N2, valor(es)]. Por exemplo, a frase “Ana subiu de 14 valores para 12 valores” possui erro semântico, e a frase “Rui desceu de 5 valor para 1 valores” possui erro sintático.

Simple, but incomplete solution:

```
frase --> sintagma_nominal, sintagma_verbal.
sintagma_nominal --> nome.
nome --> ['Ana'].
nome --> ['Rui'].
sintagma_verbal --> verbo(N1, N2), proposição, numero(N1), valor(N1), proposição,
numero(N2), valor(N2).
verbo(subir) -->['subiu'], {N1 > N2 ; write('erro semantico')}.
verbo(descer) --> ['descer'], {N1 < N2 ; write('erro semantico')}.
proposição --> ['de'].
proposição -->  ['para'].
valor(N1) --> ['valor'], {N1 = 1 ; write('erro sintatico')}.
valor(N1) --> ['valores'], {N1 \= 1 ; write('erro sintatico')}.
numero(12) --> ['12'].
numero(14) --> ['14'].
numero(5) --> ['5'].
numero(1) --> ['1'].
```

```
Frase --> SN, SV
SN --> Nome
Nome --> [Rui, Ana, Pedro, Maria, ...]
SV --> Verbo(n1, n2), De, Valor(n1), PalavraValor(n1), Para, Valor(n2), PalavraValor(n2)
Verbo: n1 > n2 -> subir
Verbo: n1 < n2 -> descer
Valor --> [0,1,2,3,...,18,19,20]
PalavraValor(x) --> x = 1 -> valor
PalavraValor(x) --> x != 1 -> valores
De --> [de]
Para --> [para]
```