

Projeto 4 – Fundamentos de programação (2017) – 12 pontos

“Mini Akinator”

Instruções:

- Trabalho desenvolvido em grupos com no máximo 5 integrantes
- O nome do arquivo deve ser $N_1-N_2-...-N_n.cpp$, sendo N_i o nome do i-ésimo membro do grupo (Exemplo: AndréMatos-CarlosVilagran.cpp).
- Não envie os arquivos executáveis (.exe).
- Envie o arquivo compactado/código fonte para luan.soares.o@gmail.com.
- Data limite para entrega: dia 27/11/2017 às 23:59 (Devido ao fim do ano letivo, esta data **NÃO PODERÁ SER ADIADA**)

1 - Cenário

1.1 - Akinator

Akinator é um jogo da internet. Consiste num gênio virtual que é capaz de adivinhar a personagem em que o jogador está pensando, seja ela real ou não, através de perguntas sobre suas características. O jogo está disponível para plataformas Android (gratuito), IOS (pago), Windows 10 Mobile (Pago) e através do site pt.akinator.com.



Figura 1: Akinator

1.2 - Árvores Binárias de Decisão

Uma árvore binária de decisão (ABD) é uma forma de árvore binária na qual os nós da árvore contêm “questões” que podem ter uma resposta binária: “falsa” ou “verdadeira”, “sim” ou “não”, “afirmativo” ou “negativo”, etc.

As folhas de uma ABD contêm as decisões. Um uso comum de ABDs é para identificação. Por exemplo, manuais de equipamentos ou guia de resolução de problemas on-line, que perguntam uma série de questões como: “Você já verificou se há energia?”, e dependendo de como você responde a questão, direciona você a uma ação a ser tomada, ou mais questões para refinar qual é o problema. Um tipo de ABD também é usada em jogos de adivinhação de números, onde o programa pergunta questões como “o número é menor que 8?”.

Um dos usos mais comuns de ABDs é em taxonomia, onde várias características físicas são usadas para identificar um objeto (um organismo, um mineral, uma doença, etc). Nestas aplicações, é importante aprender novos objetos que não se encaixam no conjunto já existente.

Um exemplo de execução de um programa

```
Iniciando...
Não foi possível encontrar um objeto, informe um objeto:
porco
Executar novamente?
s
Iniciando ...
O objeto é um porco?
n
Então qual é?
cão
Informe uma questão para a qual a resposta sim significa 'cão' e a não
significa 'porco'
Tem pelagem?
Executar novamente?
s
Iniciando ...
O objeto: Tem pelagem?
n
É um porco?
n
Então o que é?
homem
Informe uma questão para a qual a resposta sim significa 'humano' e a
não significa 'porco'
É bípede?
Executar novamente?
s
Iniciando ...
```

O objeto: Tem pelagem?
s
É um cão?
n
Então o que é?
gato
Informe uma questão para a qual a resposta sim significa 'gato' e a não
significa 'cão'
Ronrona?
Executar novamente?
n
Gravando árvore de decisão...

A árvore criada pela execução do programa pode ser vista na Figura 2.

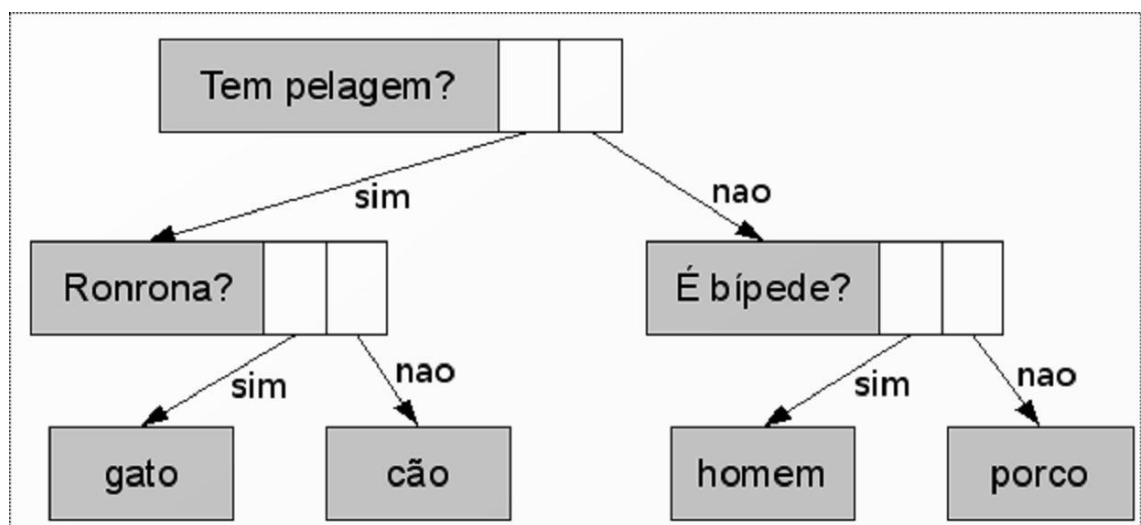


Figura 2: Árvore binária de decisão gerada pela execução do sistema.

2 - Objetivo do trabalho:

Desenvolva um programa em linguagem C que possibilite criar automaticamente árvores binárias de decisão conforme o exemplo de execução acima. O programa deverá gravar a árvore em um arquivo .dot, usando a linguagem DOT, uma linguagem muito simples para definir grafos (e também árvore)¹.

Além disso, toda vez que o programa for executado, caso exista um arquivo ".dot", o mesmo deverá ser carregado em memória, populando a árvore de decisão. Quando o carregamento for finalizado, o sistema deverá

¹ Ver detalhes em:

http://en.wikipedia.org/wiki/DOT_language

<http://www.graphviz.org/Documentation/dotguide.pdf>

Gerar figura a partir de ".dot": <http://www.webgraphviz.com/>

estar no mesmo estado de quando ele foi desligado. Em outras palavras, o sistema deve manter aquilo que aprendeu com as execuções anteriores.

Para a árvore gerada pelo exemplo anterior o arquivo “.dot” ficaria assim:

```
digraph graphname
{
    pelagem -> ronrona [label="sim"];
    ronrona -> gato [label="sim"];
    pelagem -> bipede [label="nao"];
    bipede -> homem [label="sim"];
    ronrona -> cao [label="nao"];
    bipede -> porco [label="nao"];
}
```

Poderá também ficar assim:

```
digraph graphname
{
    a [label="Tem pelagem?"];
    b [label="Ronrona?"];
    c [label="gato"];
    d [label="E bipede?"];
    e [label="cao"];
    f [label="homem"];
    g [label="porco"];
    a -> b [label="sim"];
    b -> c [label="sim"];
    a -> d [label="nao"];
    d -> f [label="sim"];
    b -> e [label="nao"];
    d -> g [label="nao"];
}
```

Outro exemplo, um código assim:

```
digraph arvore{
    "e mamifero"->"e bipede?"[label= sim];
    "e bipede?"->"tem pelagem?"[label= sim];
    "tem pelagem?"->"Tem orelha grande"[label= sim];
    "Tem orelha grande"->"Voa?"[label= sim];
    "Voa?"->"Morcego"[label= sim];
    "e mamifero"->"tem escamas?"[label= nao];
    "tem escamas?"->"rasteja?"[label= sim];
    "rasteja?"->"reptil"[label= sim];
    "tem escamas?"->"voa?"[label= nao];
    "voa?"->"gaviao"[label= sim];
    "e bipede?"->"porco"[label= nao];
    "tem pelagem?"->"humano"[label= nao];
    "Tem orelha grande"->"macaco"[label= nao];
    "Voa?"->"Coelho"[label= nao];
    "rasteja?"->"peixe"[label= nao];
    "voa?"->"galinha"[label= nao];
}
```

Ou assim:

```
digraph arvore{
  a [label = "e mamifero"];
  b [label = "e bipede?"];
  c [label = "tem pelagem?"];
  d [label = "Tem orelha grande"];
  e [label = "Voa?"];
  f [label = "tem escamas?"];
  g [label = "rasteja?"];
  h [label = "morcego"];
  i [label = "gaviao"];
  j [label = "porco"];
  k [label = "humano"];
  l [label = "coelho"];
  m [label = "peixe"];
  n [label = "galinha"];
  o [label = "macaco"];
  p [label = "Voa?"];
  q [label = "Reptil"];
  a -> b [label = "Sim"];
  a -> f [label = "Nao"];
  b -> c [label = "Sim"];
  b -> j [label = "Nao"];
  c -> d [label = "Sim"];
  c -> k [label = "Nao"];
  d -> e [label = "Sim"];
  d -> o [label = "Nao"];
  e -> h [label = "Sim"];
  e -> l [label = "Nao"];
  f -> g [label = "Sim"];
  f -> p [label = "Nao"];
  p -> i [label = "Sim"];
  p -> n [label = "Nao"];
  g -> q [label = "Sim"];
  g -> m [label = "Nao"];
}
```

Gera uma árvore como a da Figura 3:

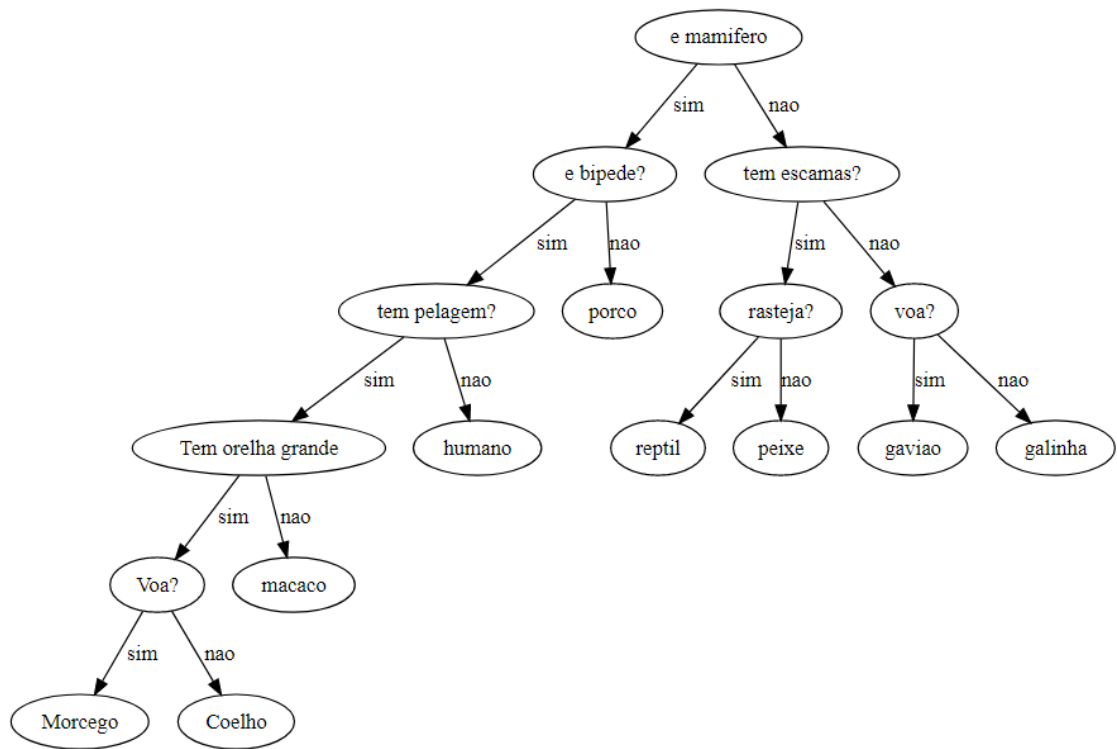


Figura 3: Exemplo de árvore gerada pelo código do arquivo “.dot”.

3 – Desafio (2 pontos extras)

É possível que, de acordo com as entradas do usuário, a árvore binária gerada pelo programa acabe ficando desbalanceada. Em outras palavras, é possível que haja muito mais galhos do lado esquerdo do que do direito, por exemplo (Como na Figura 3). Para evitar este tipo de situação, existe na computação um tipo especial de árvore conhecida como **árvore balanceada**.

Uma árvore é considerada balanceada se, e somente se, para cada um de seus nós, as alturas das sub-árvores a esquerda e a direita forem iguais ou se diferirem em apenas uma unidade².

Aplique o conceito de árvore balanceada em seu sistema, para evitar a o excesso de galhos em apenas um lado da árvore de decisão.

² Mais detalhes em: https://pt.wikipedia.org/wiki/%C3%81rvore_AVL
http://wiki.icmc.usp.br/images/f/fa/%C3%81rvores_AVL.pdf