Vítor Paixão Damasceno

Prática 13
Laboratório de AEDS

Belo Horizonte, Brasil

2024

1 Introdução

Neste experimento, foi desenvolvido um programa em Processing que simula algoritmos de busca em árvores binárias, sendo eles o algoritmo e Pré-Ordem, Em-Ordem e Pós-Ordem.

1.1 Descrição do Problema

Devemos, à partir da classe parcialmente implementada ArvoreBinaria:

- Implementar o método insereRec() à partir do pseudocódigo fornecido.
- Implementar a classe No e três métodos referentes aos três algoritmos de busca.
- Imprimir os graficamente os resultados de cada algoritmo.

1.2 Objetivos da Prática

O principal objetivo deste experimento é demonstrar o funcionamento dos algoritmos de busca em árvores binárias. No processo, espera-se compreender o funcionamento geral da estrutura de dados árvore e implementações práticas de métodos recursivos de busca.

2 Desenvolvimento

Primeiramente, o método insereRec() foi construído a partir do pseudocódigo fornecido.

```
1  private No insereRec(No raiz, int valor) {
2          if(raiz == null){
3              raiz = new No(valor);
4              return raiz;
5          }
6
7          if (valor < raiz.valor)
8              if (raiz.esq == null)
9                  raiz.esq = new No(valor);
10             else
11                 insereRec(raiz.esq, valor);
12         else if (valor > raiz.valor)
13             if (raiz.dir == null)
14                 raiz.dir = new No(valor);
15             else
16                 insereRec(raiz.dir, valor);
17
18         return raiz;
```

```
19    }
```

Em seguida, a classe No foi implementada junto de seus três métodos de busca. os algoritmos de busca foram feitos à partir dos slides da aula de árvores e grafos.

```
1   class No {
2     int valor;
3     No esq;
4     No dir;
5
6     No(int valor) {
7       this.valor = valor;
8       esq = null;
9       dir = null;
10    }
11
12    void preOrdem(){
13        imprimeNo();
14        if (esq != null) esq.preOrdem();
15        if (dir != null) dir.preOrdem();
16    }
17
18     void emOrdem(){
19        if (esq != null) esq.emOrdem();
20        imprimeNo();
21        if (dir != null) dir.emOrdem();
22    }
23
24     void posOrdem(){
25        if (esq != null) esq.posOrdem();
26        if (dir != null) dir.posOrdem();
27        imprimeNo();
28    }
29
30    void imprimeNo(){
31      int x = 40;
32      int y = height - 40;
33      fill(#00ACFF);
34      ellipse(x * (indice + 5), y, 30, 30);
35      fill(0);
36      textAlign(CENTER, CENTER);
37      text(this.valor, x * (indice + 5), y);
38      indice++;
```

```
39    }
40  }
```

Como observado, o método imprimeNo() foi a abordagem escolhida para a representação gráfica do resultado dos algoritmos. Ela consiste na impressão de cada nó um após o outro na parte inferior da tela. Essa impressão sequencial é comandada por uma varíavel global índice, já que a função é chamada dentro de uma recursão e portanto é impossível controlar este índice por meio de um laço for convencional.

Para que o usuário pudesse escolher, em tempo de execução, qual dos três algoritmos de busca será impresso na tela, foram implementados botões e seus respectivos sprites. São cinco botões: um para encerrar a simulação, um para sortear outra arvore binária e os outros três referentes aos seus respectivos algoritmos de busca.

Para detectar o clique do usuário nos botões, adotou-se a seguinte função

```
1  void menu(){
2    int tamanho = (height/15 + width/15)/2;
3    for(int i = 0; i < 5; i++)
4      image(botao[i], i * tamanho, 0, tamanho, tamanho);
5
6    if(click && mouseY >= 0 && mouseY <= tamanho){
7      if(mouseX > 0 * tamanho && mouseX < 1 * tamanho){
8        exit();
9      }
10     if(mouseX > 1 * tamanho && mouseX < 2 * tamanho){
11       setup();
12       click = false;
13     }
14     if(mouseX > 2 * tamanho && mouseX < 3 * tamanho){
15       modo = 1;
16       click = false;
17     }
18     if(mouseX > 3 * tamanho && mouseX < 4 * tamanho){
19       modo = 2;
20       click = false;
21     }
22     if(mouseX > 4 * tamanho && mouseX < 5 * tamanho){
23       modo = 3;
24       click = false;
25     }
26   }
27 }
```

Figura 1 – Sprites dos botões

3 Resultados

Dessa forma, as funções setup() e draw() do programa ficaram da seguinte forma:

```
void setup(){
    size(600, 600);
    strokeWeight(2);
    background(#A8E5EA);

    modo = 0;
    indice = 0;

    botao[0] = loadImage("button1.png");
    botao[1] = loadImage("button2.png");
    botao[2] = loadImage("button3.png");
    botao[3] = loadImage("button4.png");
    botao[4] = loadImage("button5.png");

    newTree();
}
```

```
void draw(){
  menu();

  arv.mostrar();

 if(modo != 0){
    fill(255);
    rect(0, height - 80, width - 1, 79);
    switch(modo){
      case 1:
      arv.raiz.preOrdem();
      textSize(30);
      text("PR   ORDEM", 90, height - 40);
      break;

      case 2:
```

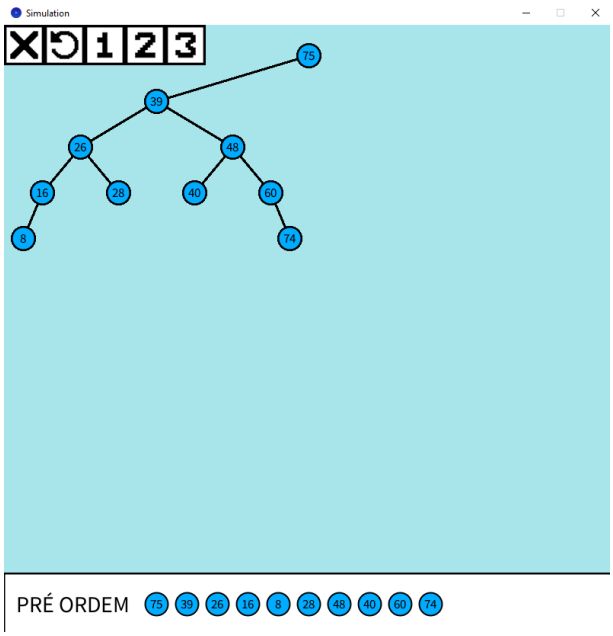

Figura 2 – Algoritmo de pré-ordem

```
17          arv.raiz.emOrdem();
18          textSize(30);
19          text(" EM ORDEM", 90, height - 40);
20          break;
21
22          case 3:
23          arv.raiz.posOrdem();
24             textSize(30);
25          text("P  S  ORDEM", 90, height - 40);
26          break;
27      }
28      indice = 0;
29   }
30  }
```

.

4 Conclusão

Dessa forma, os objetivos propostos pela prática foram devidamente concluídos. Para isso, não houveram dificuldades significativas para realização da prática, e os objetivos de compreender melhor os algoritmos de busca em arvores binárias foi satisfatoriamente alcançado.

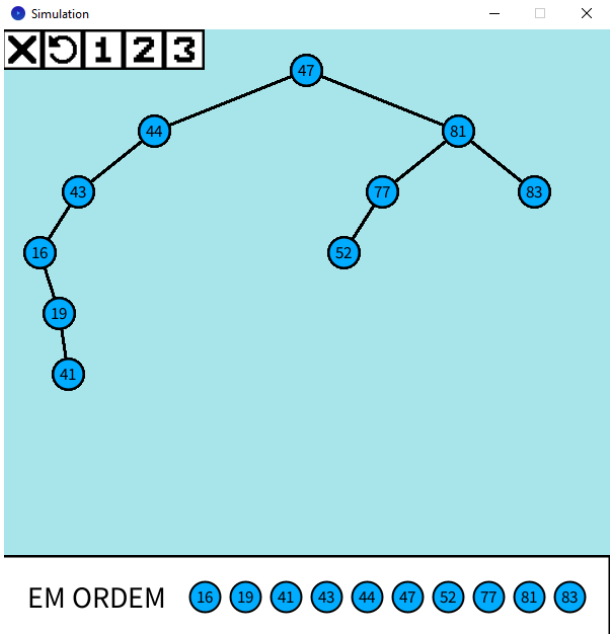

Figura 3 – Algoritmo de em-ordemn

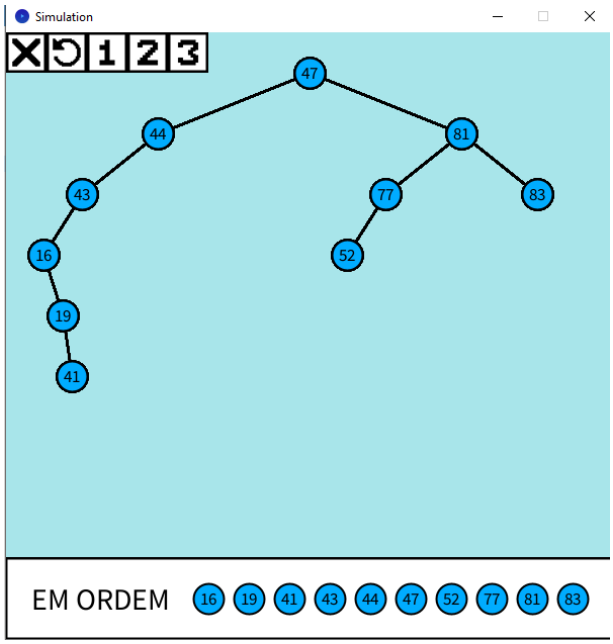

Figura 4 – Algoritmo de pós-ordem