Universidade Estadual do Ceará
Aluno: Janaína Ribeiro dos Santos **Matrícula:** 1612643
Aluno: Francisco Matheus Fernandes Freitas **Matrícula:** 1607881
Estruturas de Dados I
Prof. Marcos Negreiros
Semestre: 2022/1 – AC04

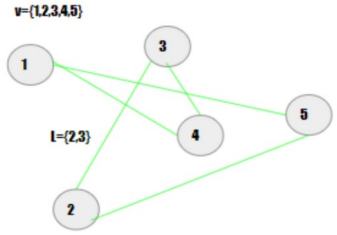
Parte I: Grafos

1. Defina e exemplifique com uma figura:

a. Grafo

Solução:

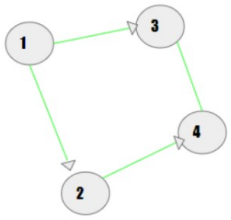
É um conjunto formado por vértices e direção entre este, sendo determinado por G(v,l) onde V é o conjunto de vértices e L o conjunto de ligação, ou arestas, entre os vértices.



b. Grafo Misto

Solução:

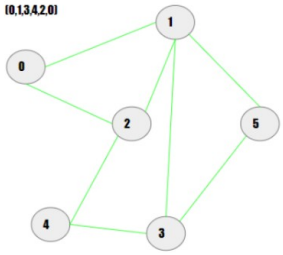
São grafos que possuem ligações orientadas: <1,3>; <1,2>;<2,4>, não orientadas: (3,4), além de serem paralelas permitindo loops.



c. Cadeia em um Grafo Não Orientado

Solução:

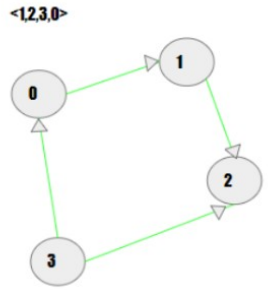
Uma cadeia é uma sequência de ligações com origem em V_0 e término em um determinado V_k, possuindo ligações não orientadas que são iguais aos elos.



d. Cadeia em um Grafo Orientado

Solução:

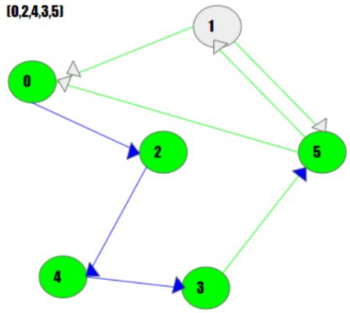
Uma cadeia é uma sequência de ligações com origem em V_0 e término em um determinado V_k sendo adjacentes entre si, possuindo ligações orientadas.



e. Caminho – Simples

Solução:

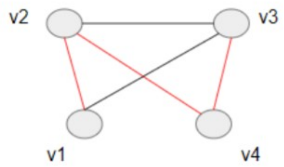
Um caminho simples em grafos é uma rota aciclica, no caso desse caminho não existem vértices repetidos.



f. Caminho – Elementar

Solução:

Um caminho é elementar quando não passa duas vezes pelo mesmo vértice.

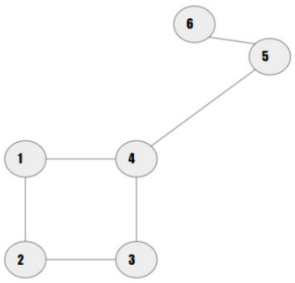


v1, v2, v4,v3

g. Grafo Conexo

Solução:

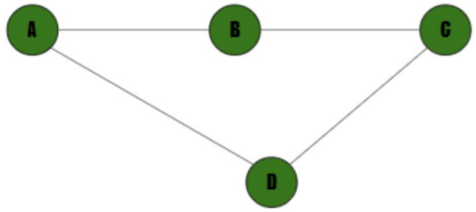
É conexo quando todo vértice do grafo está ligado, formando uma única componente.



h. Grafo f-Conexo

Solução:

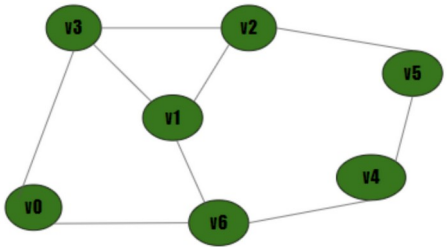
É f-Conexo, ou fortemente conexo, quando é possível alcançar de um vértice dos demais do grafo, e dos demais vértices.



i. Matriz de Adjacência

Solução:

Matriz de Adjacência é uma matriz formada por zeros e uns, onde os vértices são adjacentes sendo caracterizada como simétrica possuindo valores reais.

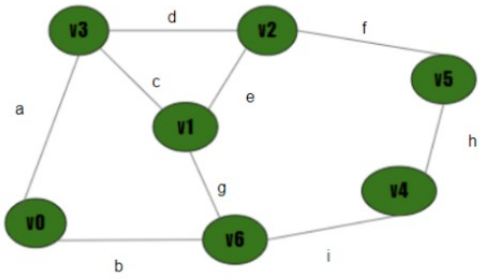


	v0	v1	v2	v3	v4	v5	v6
v0	0	0	0	1	0	0	1
v1	0	0	1	1	0	0	1
v2	0	1	0	1	0	1	0
v3	1	1	1	0	0	0	0
v4	0	0	0	0	0	1	1
v5	0	0	1	0	1	0	0
v6	1	1	0	0	1	0	0

j. Matriz de Incidência

Solução:

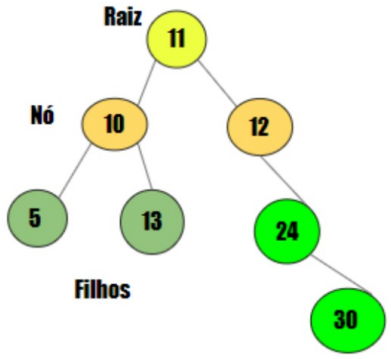
A matriz de incidência associa as arestas do grafo com os seus vértices.



	a	b	c	d	e	f	g	h	i
v0	1	1	0	0	0	0	0	0	0
v1	0	0	1	0	1	0	1	0	0
v2	0	0	0	1	1	1	0	0	0
v3	1	0	1	1	0	0	0	0	0
v4	0	0	0	0	0	0	0	1	1
v5	0	0	0	0	0	1	0	1	0
v6	0	1	0	0	0	0	1	0	1

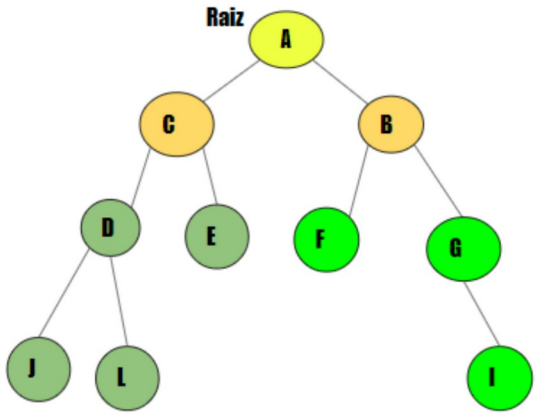
k. Árvore
Solução:

Árvore é uma estrutura de dados que possui uma raiz com nós para esquerda e direita, sendo inserido os valores de forma hierárquica, podendo ser ou não binária, possuindo elementos subordinados aos nós internos, ou sub-árvores, chamados de filhos. Possui operações de busca, remoção e inserção.

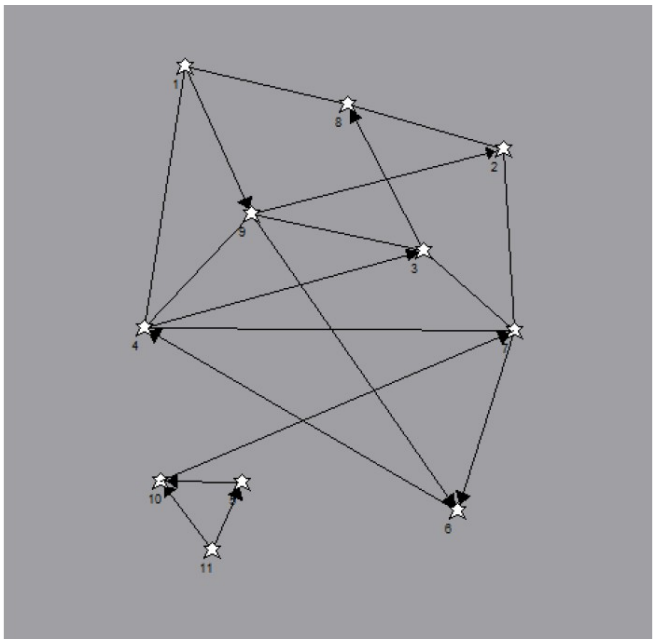


l. Árvore enraizada
Solução:

Uma árvore enraizada possui um conjunto finito de nós, sendo o elemento raiz não nulo. A raiz não possui nenhum pai, os nós que estão ligados a raiz possuem nível 1 na hierarquia.



2. Represente o grafo a seguir na estrutura de dados mais adequada, de modo que seja possível editá-lo, e avalie a sua complexidade em espaço. Verifique a conectividade e a f-conectividade do grafo.



Solução:

Primeiro foi necessário identificar as adjacências e incidências do grafo, de modo a identificar um possível padrão aplicável a uma estrutura de maneira mais amigável. Usei o método explicado em sala para montar as duas matrizes que serviriam para a análise.

Matriz de Incidência

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	+-1	0	0	0	+-1	1	0	0
2	0	0	0	0	0	0	+-1	+-1	-1	0	0
3	0	0	0	-1	0	0	+-1	1	+-1	0	0
4	+-1	0	1	0	0	-1	+-1	0	+-1	0	0
5	0	0	0	0	0	0	0	0	0	1	-1
6	0	0	0	1	0	0	-1	0	-1	0	0
7	0	+-1	+-1	+-1	0	1	0	0	0	-1	0
8	+-1	+-1	-1	0	0	0	0	0	0	0	0
9	-	1	+-1	+-1	0	1	0	0	0	0	0
10	0	0	0	0	-1	0	1	0	0	0	-1
11	0	0	0	0	1	0	0	0	0	1	0

Ao analisar as adjacências, é possível usarmos uma estrutura estudada anteriormente, um vetor estático de listas encadeadas simples. Ilustramos na matriz de incidências abaixo:

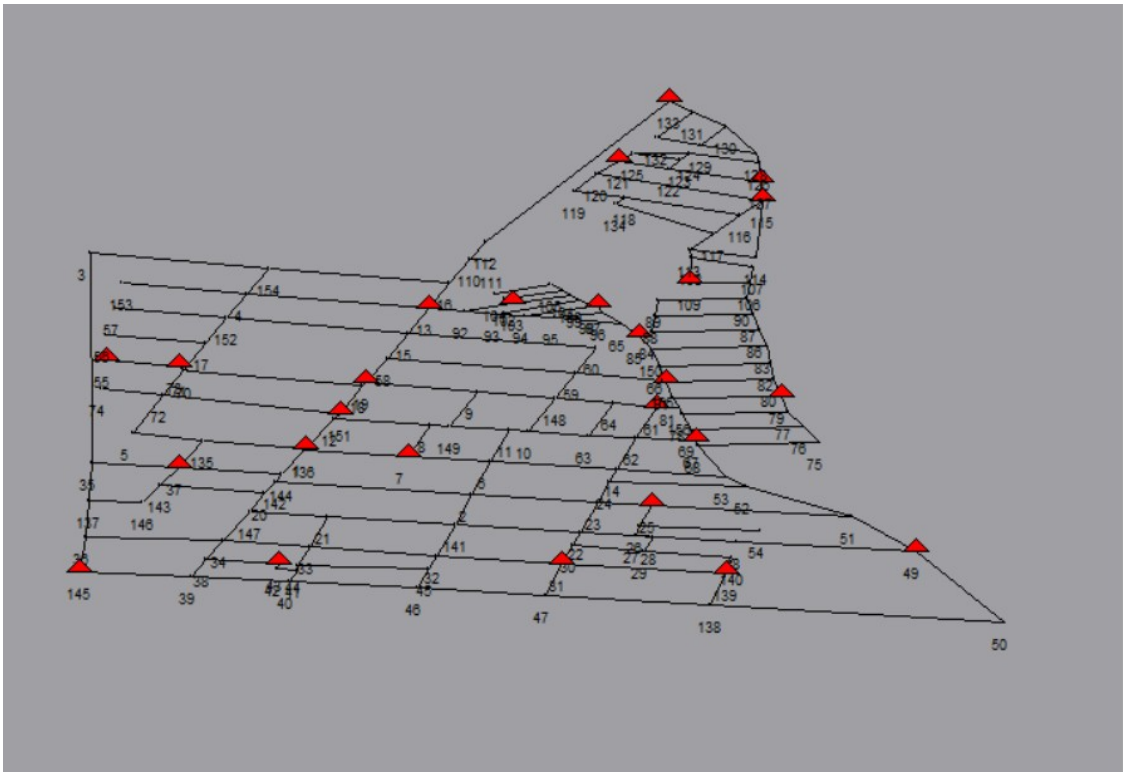
Desse modo, podemos editar de maneira mais fácil as possíveis incidências.

Vetor de listas

1	4	8	9	
2	7	8		
3	7	8	9	
4	1	3	7	9
5	10			
6	4			
7	2	3	4	6
8	1	2		
9	2	3	4	6
10	7			
11	5	10		

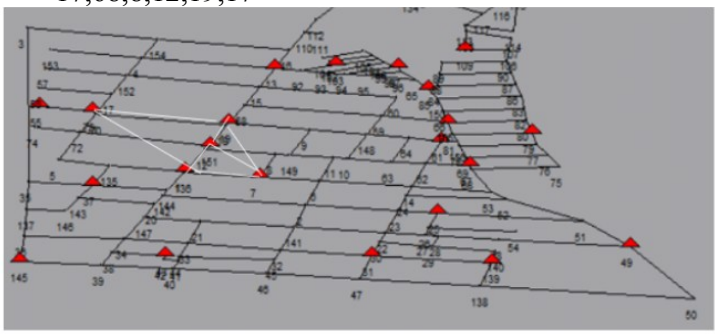
Sua complexidade em espaço seria relativo ao número de vértice para o vetor estático de listas, e as listas teriam no máximo o número de vértices do grafo, variando com relação a incidência. Se trata de um grafo conexo, porém não f. Conexo pois o vértice 11 não é alcançável por nenhum outro vértice.

3. No grafo da figura abaixo – representando a rede de ruas do bairro Ancuri de Fortaleza/CE, represente os sub-grafos solução indicados.



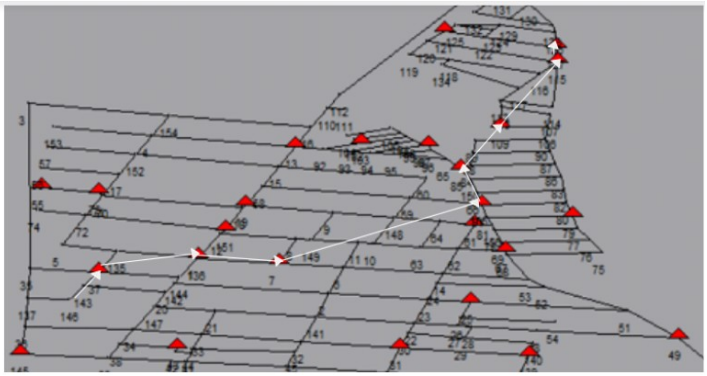
a. Um circuito hamiltoniano entre os vértices vermelhos sobre o grafo
Solução:

<17,68,8,12,19,17>

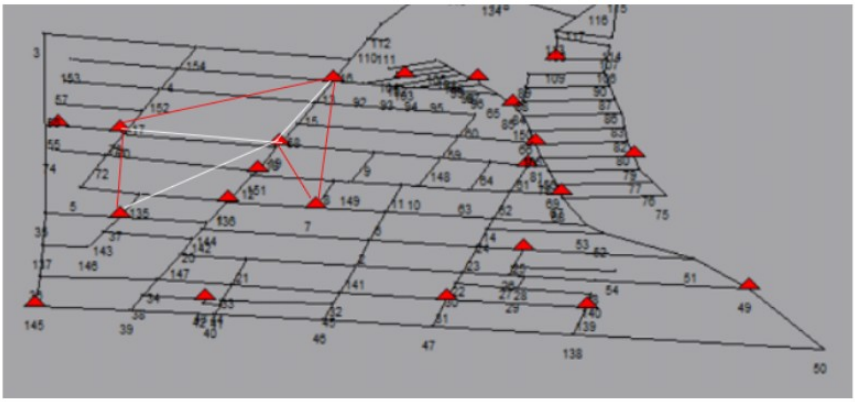


b. Um caminho simples do vértice 143 ao vértice 137
Solução:

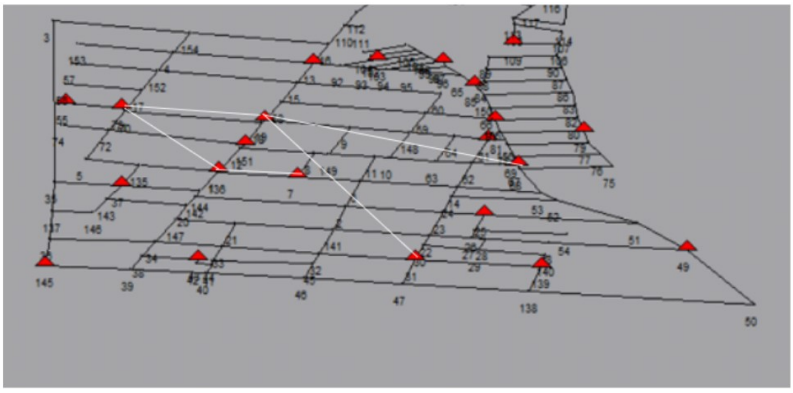
<143, 135, 12, 8, 150, 65, 113, 137>



c. Um caminho hamiltoniano entre os vértices vermelhos
Solução:
<135,17,16,8,68>



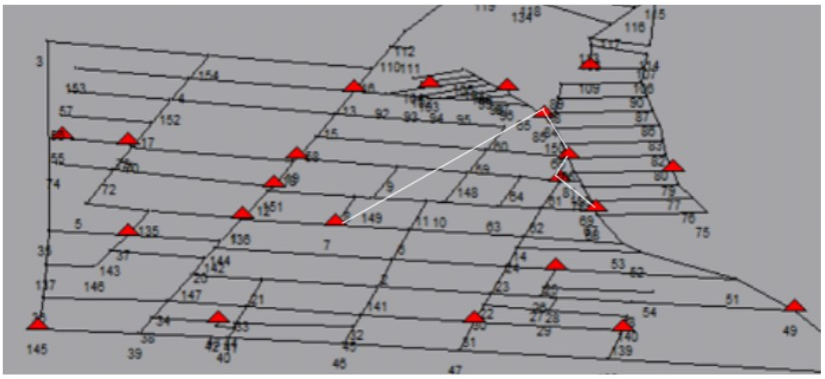
d. Uma árvore conectando os vértices vermelhos do grafo composta apenas de caminhos simples entre pares de vértices
Solução:
<17,12,8,68,22,78> em in-ordem



e. Uma árvore conectando os vértices vermelhos compostos de caminhos elementares entre os pares de vértices.

Solução:

(88,8,150,66,78) em in-ordem



Parte II: Árvores Binárias

4. Mostre e descreva

a. O CAD Árvore Binária.

Solução:

CAD Node que será usado na Tree. Servirá para encapsular os conteúdos que entrarão na estrutura:

```
public class Node
{
    public int Key { get; set; } // Chave que servirá para organizar a tree
    public Object Obj { get; set; } // Conteúdo que será guardado nesse node
    public Node Left { get; set; } // Node à esquerda
    public Node Right { get; set; } // Node à direita
    public Node(int key, Object obj) // Construtor do Node
    {
        Key = key;
        Obj = obj;
        Left = null;
        Right = null;
    }
    public override string ToString() // Método para mostrar melhor o node
    {
        return $"Id: {Key}, Conteúdo: {Obj}";
    }
}
```

CAD Tree, tratando a árvore como um objeto em si:

```
namespace Codigos.Estruturas;
public class Tree
{
    public Node Root { get; set; } // Váriavel que servirá como base para a estrutura
    public Tree(){} // Construtor da Tree

    public ~Tree(){} // Destrutor da Tree

    public bool Insert(int id, Object obj); // Método que insere nodes na tree, se possivel.

    public bool Search(int id); // Método que responde true se encontrar um node com o id passado.

    public bool Delete(int id); /* Fará a deleção do node que possuir o id passado
                                , caso contrario, um false é retornado.*/

    public void Show(char Mode); /* Método que mostra toda a tree, de acordo com o modo
                                passado no parametro: 'C' = crescente, 'D' = decrescente, 'L' = Layer */
}
```

Temos todas as funções principais de uma estrutura Tree.

b. Defina formalmente o que é uma árvore binária.

Solução:

Se trata de uma árvore que é uma estrutura de dados que permite a organização da informação de tal forma que todo elemento da estrutura possa ser encontrado no menor tempo possível, dependendo da necessidade de sua disponibilidade.

Árvore binária é uma estrutura de busca, onde cada nó tem no máximo dois filhos, e apenas um nó pai. Não podendo assumir índices repetidos, do contrário não será uma estrutura de busca.

c. Discuta a complexidade de caso médio e de pior caso dos métodos de inserção, busca e deleção de uma árvore binária.
Solução:

Para o caso médio assumimos que uma parte da entrada de dados esteja não ordenada, nesse caso os métodos da árvore teriam complexidade próxima ($\approx$) do $\log_2 n$, onde n é o número de itens do conjunto de dados. Isso porque a árvore teria pouca degeneração, mantendo suas características.

Para o pior caso, assumimos que a entrada já se encontrada ordenada, logo nossa árvore ficaria muito degenerada, aumentando a complexidade de seus métodos para $O(n)$, visto que nesse caso, nossa árvore seria basicamente uma lista encadeada.

d. Por quê a deleção pode não funcionar numa árvore binária que não é de busca?
Solução:

A árvore não sendo binária pode ter a ocorrência de valores repetidos, logo pode existir problemas na deleção de um valor caso o mesmo esteja duplicado dentro da mesma. Ao se procurar pelo valor indicado, não será confiavel que o primeiro encontrado é o que eu estaria realmente solicitando.

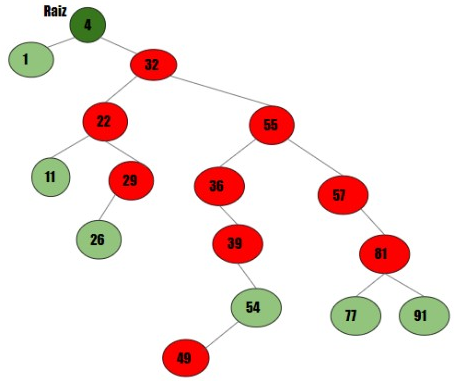
e. Qual a complexidade da altura de uma árvore binária cheia? E de uma árvore d-ária cheia?
Solução:

Para uma árvore binária cheia, temos que o número de itens aumenta de acordo com a seguinte função: $2^{(n+1)}-1$, desse jeito para calcular sua altura teremos complexidade de $\log_2 n$.

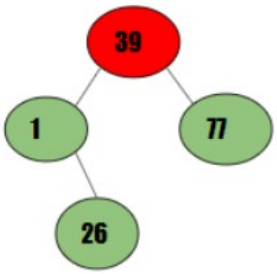
Para uma árvore d-ária cheia, teríamos que calcular como ele cresceria. Para isso fazemos uso da fórmula da PA: $\frac{a_1(q^{(n+1)}-1)}{q-1}$. Para uma árvore terciaria teríamos: $\frac{3^{(n+1)}-1}{2}$. Sua complexidade quanto à altura seria então $\log\frac{3^n-1}{2}$.

5. Mostre como fica a árvores binárias depois de acontecer os seguintes movimentos nessa ordem:

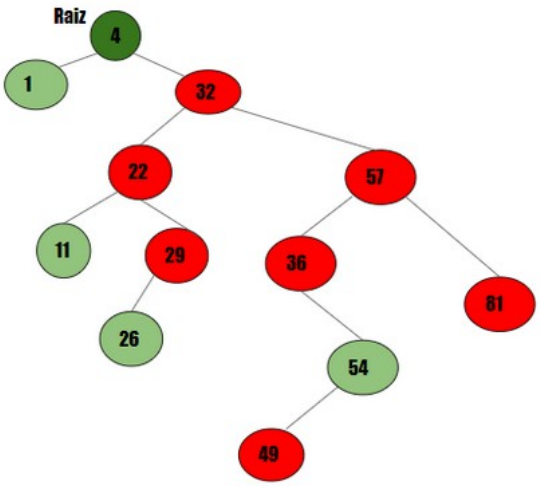
a. Inserção S = (4, 32, 22, 55, 36, 1, 57, 29, 49, 39, 81, 91, 26, 54, 77, 91, 11, 32, 54)
Solução:



b. Busca B = (39, 92, 1, 77, 26)
Solução:



c. Deleção D = (77, 55, 91, 39)
Solução:

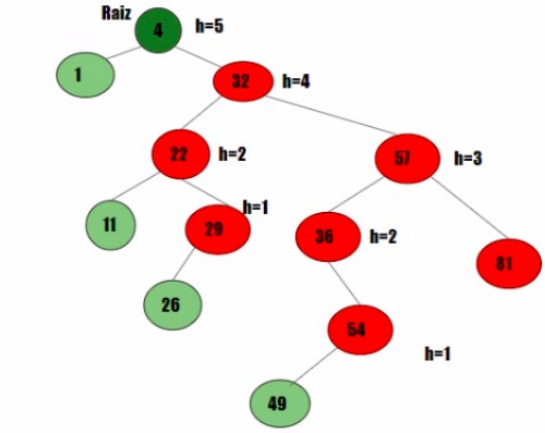


d. Mostre a árvore binária em pré-ordem, pós-ordem e in-ordem.
Solução:

pré-ordem: 4,1,32,22,11,29,26,55,36,39,54,49,57,81,77,91
pós-ordem: 1,11,26,29,22,49,54,39.36,77,91,81,57,55,32,4
in-ordem: 1,4,11,22,26,29,32,36,39,49,54,55,57,77,81,91

e. Apresente a altura dos nós internos, o número de níveis da árvore e o nó gerador da árvore final após a deleção.
Solução:

Começando pela altura, considerando h = altura, temos:



Considerando que a contagem de níveis começa em 0, a árvore possui 7 níveis, onde o nível zero começa na raiz.

Após as deleções, o nó gerador continua sendo o nó 4.

6. Mostre a implementação do método de inserção numa árvore binária.
Solução:
Para o método de inserção em uma árvore binária, podemos fazer da seguinte maneira em C.

Primeiro o struct de um Nó, e
já definindo Pont como um ponteiro de No.

```c
typedef struct aux
{
    Key key;
    /*dados podem vir aqui*/
    struct aux *left, *right;
} No;

typedef No *Pont;
```

Implementação recursiva de inserção na árvore binária

```c
Pont insert(Pont root, Pont no)
{
    if (root == NULL)
        return (no);
    if (root->key < no->key)
    {
        root->right = insert(root->right, no);
    }
    else
    {
        root->left = insert(root->left, no);
    }
    return root;
}
```

7. Mostre a implementação do método de deleção numa árvore binária com substituição.
Solução:

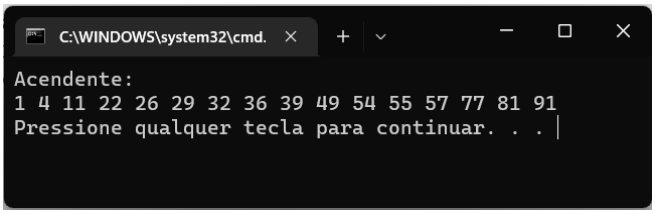
```python
def remover(self, valor, NoArvore=Raiz):
        if NoArvore == Raiz:
            NoArvore = self.raiz
        if NoArvore is None:
            return NoArvore
        if valor < NoArvore.dado:
            NoArvore.esquerda = self.remover(valor, NoArvore.esquerda)
        elif valor > NoArvore.dado:
            NoArvore.direita = self.remover(valor, NoArvore.direita)
        else:
            if NoArvore.esquerda is None:
                return NoArvore.direita
            elif NoArvore.direita is None:
                return NoArvore.esquerda
            else:
                substituir = self.NoArvore.direita
                NoArvore.dado = substituir
                NoArvore.direita = self.remover(substituir, NoArvore.direita)
        return NoArvore
```

8. Apresente o método e o resultado que permite mostrar a ordenação ascendente dos dados inseridos numa árvore binária (resultante de 5).
Solução:
O método usado para mostrar a ordenação ascendente dos dados foi feito usando a recursividade, e está na linguagem C abaixo:

```c
void showC(Pont root)
{
    if (root ≠ NULL)
    {
        showC(root→left);
        printf("%i ", root→key);
        showC(root→right);
    }
}
```

O método sempre testa se a root corrente não é nula, indicando que pode ser uma folha ou um nó interno. Em seguida é chamado seu filho esquerdo, que se for nula nada acontece, mas do contrário a recursividade entra em ação.

Ao chegar na extrema esquerda, ele exibe seu conteúdo e vai para a sua direita, repetindo os passos de tentar encontrar sempre algo mais à direita. No fim, teremos o seguinte resultado para a entrada de dados da questão 5, assumindo que está árvore binária não permite índices repetidos:



9. Discuta as diferenças entre o método de deleção por substituição e o método de deleção por reinserção. Qual a complexidade média de ambos?
Solução:

A diferença central se encontra no modo de deleção, que ao ser por reinserção será guardado os nós esquerdo e direito, enquanto que o nó raiz atual é deletado. Feito isso, um dos ponteiros será tratado como nó gerador da arvore, enquanto que o outro será inserido de maneira igual como no método de inserção já mostrado anteirormente.

Sua complexidade pode ser encontrada como a complexidade do método de busca mais a complexidade do método de inserção de um único nó. Logo temos $O(n^2)$.

10. Uma calculadora eletrônica deve ser implementada considerando que qualquer expressão aritmética possa ser representada. Mostre uma implementação desta funcionalidade usando árvores binárias. Considere o exemplo para ilustrar sua implementação: F(x) = 3*Log(2*sqrt(x)) – (x-6) / Log (4 x^0.5). Apresente os resultados para x=4, x=21, x=-12,7.

Solução:
Primeiro definimos o nó da nossa árvore.

```
// Classe base para representar um nó da árvore binária
public class BinaryTreeNode
{
    public string Value { get; set; }
    public BinaryTreeNode Left { get; set; }
    public BinaryTreeNode Right { get; set; }

    public BinaryTreeNode(string value)
    {
        Value = value;
        Left = null;
        Right = null;
    }
}
```

Feito isso, podemos definir a árvore da calculadora da seguinte maneira: (Ficou muito grande, por isso a implementação está na próxima página.)

Os resultados foram:

```
matheus@matheus-hpnotebook:~/workspace/CSharpCodes/Tree
[matheus@matheus-hpnotebook Tree]$ dotnet run

Entre com o valor de x:
  (Para sair, digite 0)
4
O resultado é: 2,6666666666666665

Entre com o valor de x:
  (Para sair, digite 0)
21
O resultado é: -1,2896375561638524

Entre com o valor de x:
  (Para sair, digite 0)
-12.27
O resultado é: NaN
```

```
public class BinaryTree
{
    public BinaryTreeNode Root { get; set; }

    public BinaryTree(string expression)
    {
        // Cria a árvore binária a partir da expressão fornecida
        Root = CreateTree(expression);
    }

    // Método auxiliar para criar a árvore binária recursivamente
    private BinaryTreeNode CreateTree(string expression)
    {
        // Verifica se a expressão está vazia
        if (string.IsNullOrEmpty(expression))
        {
            return null;
        }

        // Remove os espaços em branco da expressão
        expression = expression.Replace(" ", "");

        // Verifica se a expressão é um número ou uma operação
        bool isNumber = double.TryParse(expression, out double result);
        if (isNumber)
        {
            // Cria um nó com o número e retorna
            return new BinaryTreeNode(expression);
        }
        else
        {
            // A expressão é uma operação, então precisamos encontrar a posição da operação
            int operatorIndex = GetOperatorIndex(expression);

            // Cria os nós filhos recursivamente
            BinaryTreeNode left = CreateTree(expression.Substring(0, operatorIndex));
            BinaryTreeNode right = CreateTree(expression.Substring(operatorIndex + 1));

            // Cria o nó pai com a operação e os nós filhos
            return new BinaryTreeNode(expression[operatorIndex].ToString())
            {
                Left = left,
                Right = right
            };
        }
    }

    // Método auxiliar para encontrar a posição da operação na expressão
    private int GetOperatorIndex(string expression)
    {
        // Começa com a operação de maior precedência
        int index = GetOperatorIndex(expression, new[] { "*", "/" , "log", "^"});
        if (index != -1)
        {
            return index;
        }

        // Se não houver operações de multiplicação ou divisão, procura por operações de adição e subtração
        index = GetOperatorIndex(expression, new[] { "+", "-" });
        if (index != -1)
        {
            return index;
        }

        // Se não houver operações, retorna -1
        return -1;
    }

    // Método auxiliar para encontrar a posição da primeira operação especificada na expressão
    private int GetOperatorIndex(string expression, string[] operators)
    {
        int index = -1;
        int minIndex = int.MaxValue;

        // Procura por cada operação especificada
        foreach (string op in operators)
        {
            index = expression.IndexOf(op);
            if (index != -1 && index < minIndex)
            {
                minIndex = index;
            }
        }

        return minIndex;
    }

    // Método para avaliar a expressão na árvore binária e retornar o resultado
    public double Evaluate()
    {
        return EvaluateNode(Root);
    }

    // Método auxiliar para avaliar um nó recursivamente
    private double EvaluateNode(BinaryTreeNode node)
    {
        // Verifica se o nó é uma folha (número)
        if (node.Left == null && node.Right == null)
        {
            return double.Parse(node.Value);
        }
        else
        {
            // O nó é uma operação, então avalia os nós filhos
            double left = EvaluateNode(node.Left);
            double right = EvaluateNode(node.Right);

            // Realiza a operação e retorna o resultado
            switch (node.Value)
            {
                case "+":
                    return left + right;
                case "-":
                    return left - right;
                case "*":
                    return left * right;
                case "/":
                    return left / right;
                case "log":
                    return Log(left, right);
                case "sqrt":
                    return Sqrt(left, right);
                case "^":
                    return Pow(left, right);

                default:
                    throw new InvalidOperationException("Operação inválida.");
            }
        }
    }
```