

PRA – Projeto de Arquivos

Introdução a árvores binárias

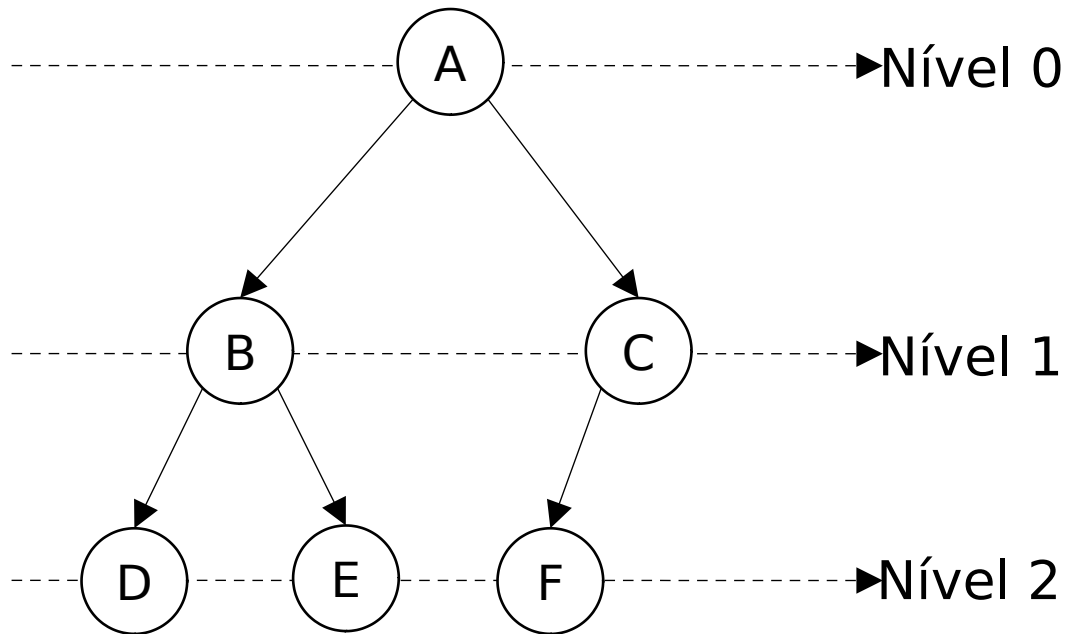
Prof. Allan Rodrigo Leite

Introdução

- Estruturas de dados baseadas em lista representam dados de maneira sequencial
- Árvores são estruturas adequadas para representação hierárquica
 - Estruturas de árvores possui uma definição recursiva
 - Cada nó da árvore forma uma subárvore
- Uma árvore é formada por
 - Um conjunto de nós
 - Um dos nós é denominado *raiz*
 - Cada nó pode ter múltiplos *filhos*
 - Os nós que não possuem filhos são chamados de *folhas*

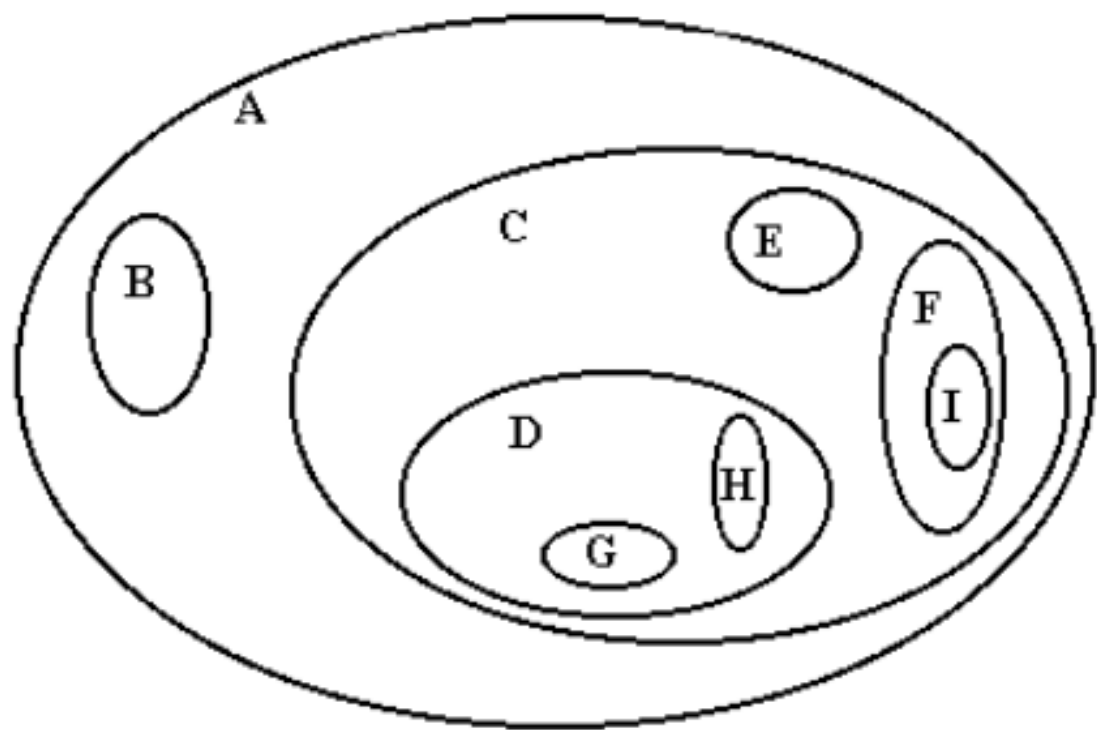
Introdução

- Estrutura de uma árvore
 - Nós representam os vértices da árvore
 - Arestas conectam dois vértices
 - Níveis da árvore

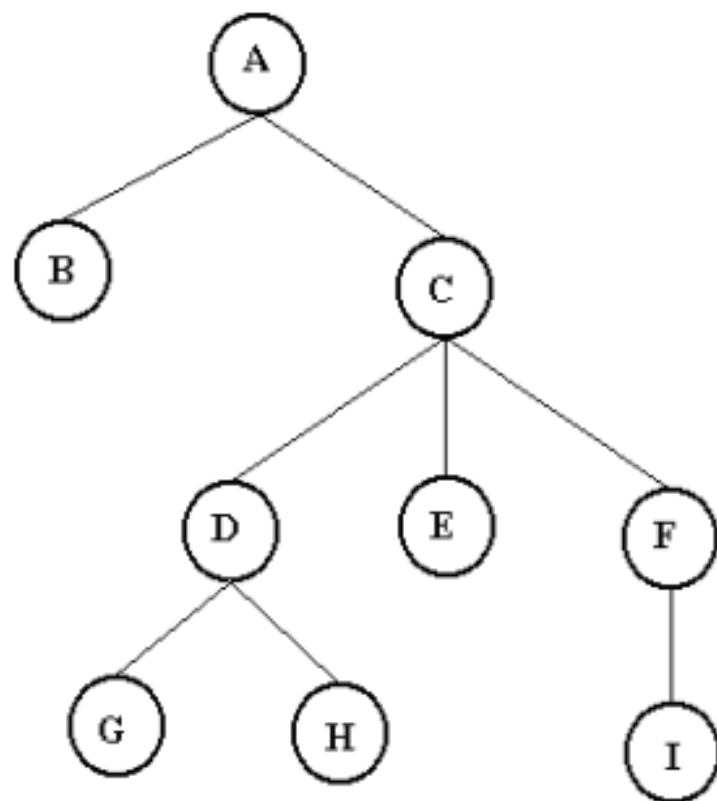


Introdução

Diagrama de Inclusão

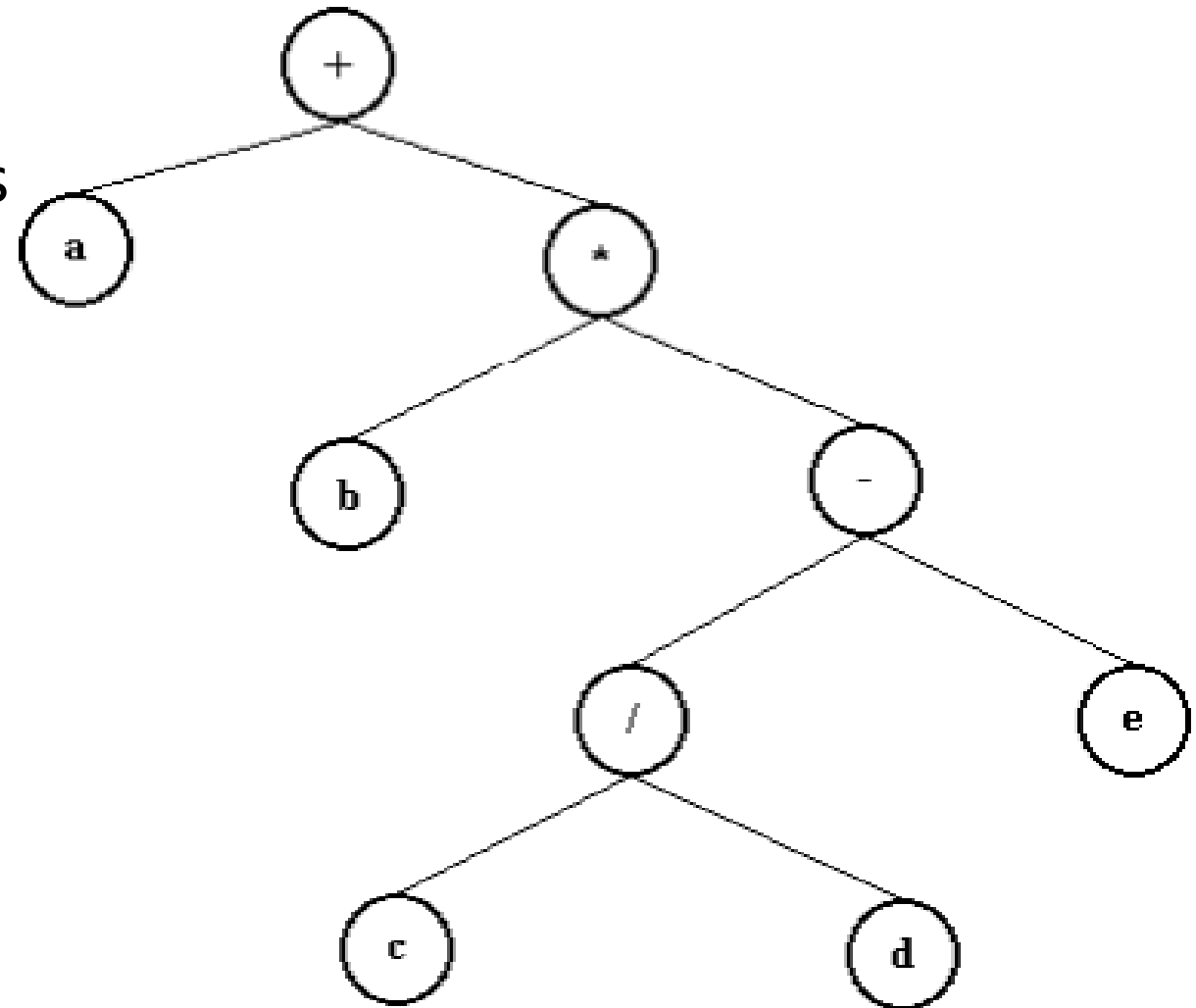


Representação Hierárquica



Introdução

- Exemplos
 - Representação de expressões aritméticas
(a + (b * (c / d - e)))



Introdução

- Conceitos

- Cada nó pode possuir um único nó pai e uma coleção de zero ou mais nós filhos
- Nó raiz não possui nó pai
- Nós folhas possuem uma coleção vazia de filhos
- Grau de saída representa a quantidade de filhos de um nó
- Grau de uma árvore representa a maior quantidade de filhos entre todos os nós
- Uma floresta é uma coleção de árvores

Introdução

- Caminho

- É a uma sequência de nós distintos ligados pela relação de pai e filho
- Existe apenas um caminho da raiz para qualquer nó
- O comprimento de um caminho refere-se ao número de nós contidos

- Profundidade

- Comprimento do maior caminho da raiz até o nó folha mais distante
- Profundidade é uma medida de altura da árvore

- Largura

- Maior número de filhos dos nós pertencentes ao mesmo nível

Introdução

- Árvore cheia
 - Árvore de grau D onde todos os nós possuem o mesmo número de filhos, exceto os nós folha
- Árvore cheia de grau 2
 - Pode ser implementadas sequencialmente por vetor
 - Chamadas de árvores binárias
 - Exemplo com expressões aritméticas
 - Nós folhas representam os operandos (números)
 - Nós internos representam os operadores

Árvores binárias

- Representação em C

```
typedef struct {  
    struct No* pai;           //ponteiro para o nó pai  
    struct No* esquerda;     //ponteiro para o nó filho a esquerda  
    struct No* direita;      //ponteiro para o nó filho a direita  
    float v;                  //conteúdo de um nó arbitrário da árvore  
} No;  
  
typedef struct arvore {  
    No* raiz;  
} Arvore;
```

Árvores binárias

- Operações básicas
 - Criar uma estrutura de árvore
 - Inserir um novo elemento no árvore
 - Remover o elemento da árvore
 - Verificar se a árvore está vazia
 - Liberar memória da árvore
 - Percorrer os caminhos da árvore

Árvores binárias

- Criar uma árvore

```
Arvore* cria() {  
    Arvore *arvore;  
    arvore = malloc(sizeof(Arvore));  
    arvore->raiz = NULL;  
  
    return arvore;  
}
```

- Verifica se a árvore está vazia

```
int vazia(Arvore* arvore) {  
    return (arvore->raiz == NULL);  
}
```

Árvores binárias

- Inserir um elemento na árvore

```
No* adiciona(Arvore* arvore, No* pai, float valor) {  
    No *no = malloc(sizeof(No));  
  
    no->pai = pai;  
    no->esquerda = NULL;  
    no->direita = NULL;  
    no->valor = valor;  
  
    if (pai == NULL)  
        arvore->raiz = no;  
  
    return no;  
}
```

Árvores binárias

- Remover um elemento na árvore

```
void remove(Arvore* arvore, No* no) {  
    if (no->esquerda != NULL)  
        remove(arvore, no->esquerda);  
  
    if (no->direita != NULL)  
        remove(arvore, no->direita);  
  
    if (no->pai == NULL) {  
        arvore->raiz = NULL;  
    } else {  
        if (no->pai->esquerda == no) {  
            no->pai->esquerda = NULL;  
        } else {  
            no->pai->direita = NULL;  
        }  
    }  
  
    free(no);  
}
```

Árvores binárias

- Percorrer a árvore

```
void percorrer(No* no) {  
    if (no != NULL) {  
        printf("%f", no->valor);  
  
        percorrer(no->esquerda);  
        percorrer(no->direita);  
    }  
}
```

Árvores binárias

- Busca em árvore
 - Estratégia para visitar os nós de uma árvore
- Busca em profundidade
 - Elementos são visitados intercalando os níveis da árvore
 - O resultado da busca a partir da raiz forma múltiplos caminhos
- Busca em largura
 - Elementos são visitados no mesmo nível da árvore
 - Em seguida é realizada a mesma busca no próximo nível da árvore
 - Precisa ser conhecido quais nós pertencem a cada nível

Árvores binárias

- Busca em profundidade
 - A busca ocorre da esquerda para a direita a cada subárvore
 - Estratégias para visitar os nós: *pre-order*, *pre-order* e *pos-order*
- *Pre-order*
 - Raiz, esquerda, direita
- *In-order*
 - Esquerda, raiz e direita
- *Pos-order*
 - Esquerda, direita e raiz

Árvores binárias

- Árvores ordenadas
 - Os nós são dispostos de forma hierárquica e ordenados
 - O nó da esquerda possui um conteúdo menor que o nó atual
 - O nó da direita possui um conteúdo maior que o nó atual
- Altura e largura máxima são as dimensões da árvore
 - O esforço computacional para percorrer uma árvore é proporcional a altura e largura

Árvores binárias

- Balanceamento de árvores
 - Árvore degenerada
 - Ocorre quando cada nó possui uma única subárvore associada
 - Neste caso a árvore representa uma estrutura linear
 - Percorrer nesta árvore corresponde a complexidade N
 - Árvore cheia
 - Possui um balanceamento perfeito
 - Todos os nós possuem dois filhos ou são folhas
 - Percorrer nesta árvore corresponde a complexidade $\log N$

Árvores binárias genéricas

- Estrutura de uma árvore binária genérica

```
typedef struct {  
    struct No* esquerda; //ponteiro para o filho a esquerda  
    struct No* direita;  //ponteiro para o filho a direita  
    void *v;              //conteúdo genérico do nó  
} No;
```

```
typedef struct arvore {  
    No* raiz;              //raiz da árvore  
    int tamanho;           //tamanho do conteúdo em bytes  
} Arvore;
```

Árvores binárias genéricas

- Como tornar genérica a operação de pesquisa?
- Estratégia baseada em *callback*
 - Permite separação entre
 - Função que percorre os elementos da estrutura de dados
 - Operação realizada sobre cada elemento visitado na iteração
 - Função para percorrer é genérica
 - A parte específica é o que será feito a cada elemento visitado
 - Operação de *callback*

Árvores binárias genéricas

- Implementação ocorre por meio de ponteiro de funções
 - Nome de uma função representa o endereço da função
- Exemplo
 - Assinatura da função
`void callback(void* v);`
 - Declaração do ponteiro para armazenar o endereço da função
`void *(cb)(void*);`

onde `cb` é ponteiro para funções com mesma assinatura que `callback`.

Árvores binárias genéricas

- Recomendações sobre o uso de *callbacks*
 - Evitar uso de variáveis globais
 - Problemas de concorrência
 - Dificuldade para compreensão do código
- Passar parâmetros genéricos, se for o caso
 - Pode ter, além do ponteiro do elemento atual, um ponteiro para dados
- Isolar o comportamento genérico específico do *callback*
 - Reduz o acoplamento entre o cliente e a implementação de lista genérica

Árvores binárias genéricas

- Operação de percorrer genérica com suporte a *callback*

```
void percorrer(No* no, void (callback)(void*)) {  
    if (no != NULL) {  
        callback(no->v);  
  
        percorrer(no->esquerda);  
        percorrer(no->direita);  
    }  
}
```

Implementação de listas genéricas

- Operação de percorrer sendo chamada pelo cliente

```
void meuCb(void* par) {  
    float *v = (float*) par;  
    printf("%f\n", v);  
}
```

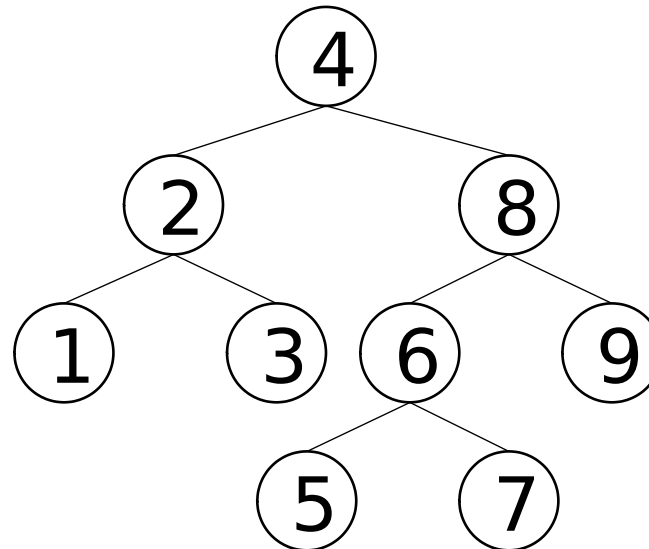
```
int main(int argc, char *argv[]){  
    Arvore *a = cria();  
    //Insere dados na lista...  
    percorre(a, meuCb); //Passando a função como parâmetro  
}
```


Exercícios

1. Implemente um algoritmo que visite todos os nós de uma árvore binária com uma busca em profundidade *in-order*, *pre-order* e *pos-order*

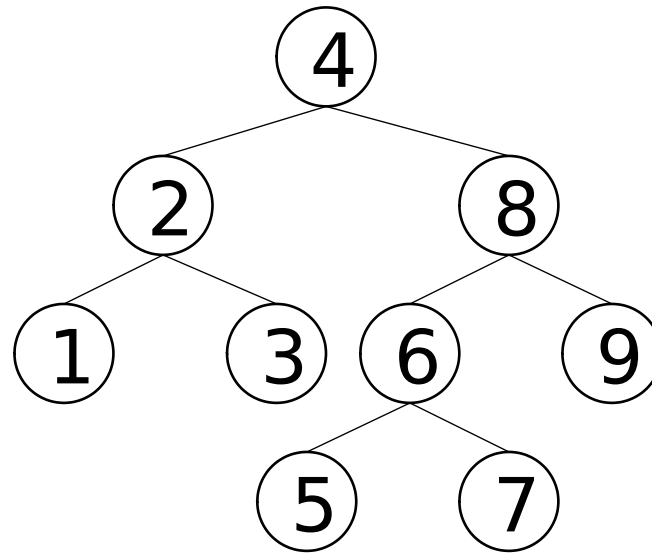
- Exemplo, para a árvore abaixo a saída deve ser:

- *Pre-order*: 4, 2, 1, 3, 8, 6, 5, 7, 9
- *In-order*: 1, 2, 3, 4, 5, 6, 7, 8, 9
- *Pos-order*: 1, 3, 2, 5, 7, 6, 9, 8, 4



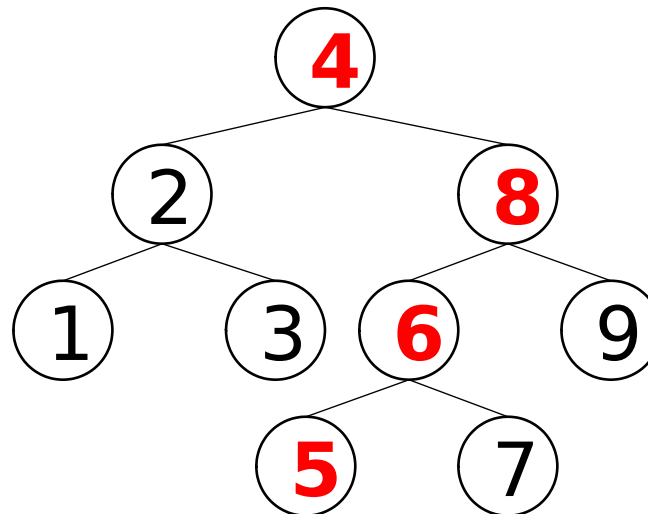
Exercícios

2. Implemente um algoritmo que visite todos os nós de uma árvore binária com uma busca em largura
- Exemplo, para a árvore abaixo a saída deve ser: 4,2,8,1,3,6,9,5,7



Exercícios

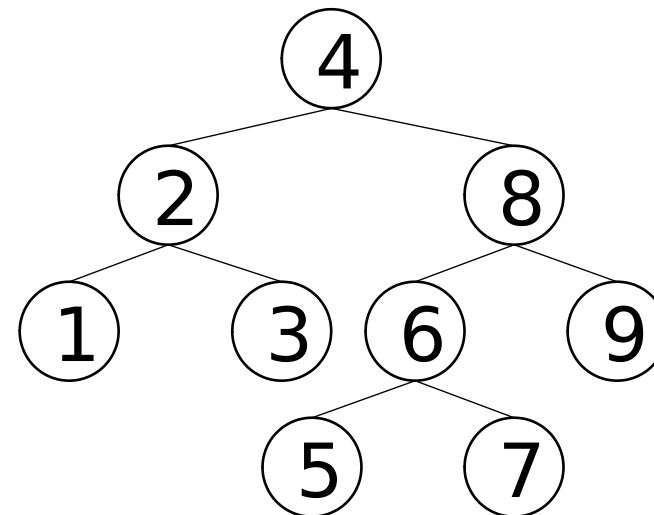
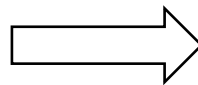
3. Dada uma árvore binária, retorne o número de comparações feitas em uma pesquisa binária para localizar um nó dada uma chave de acesso.
- Os nós desta árvore devem respeitar as seguintes regras:
 - O valor de cada nó deve ser inteiro
 - O nó filho à esquerda possui um valor menor ou igual ao valor de seu nó pai
 - O nó filho à direita possui um valor maior do que o valor de seu nó pai
 - Exemplo: dada a árvore abaixo e a chave de pesquisa 5, a saída esperada é 4



Exercícios

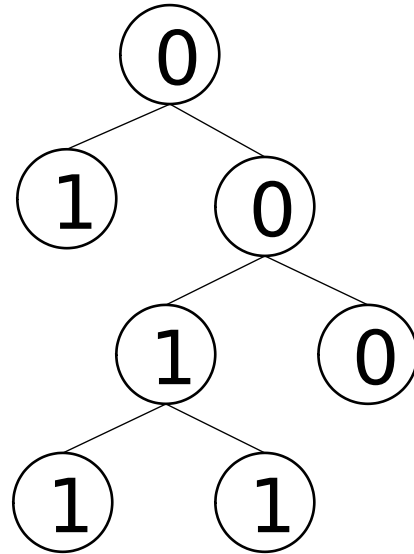
4. Implemente uma função para adicionar nós em uma árvore binária que respeite as regras abaixo necessárias para uma pesquisa binária.
- O valor de cada nó deve ser inteiro
 - O nó filho à esquerda possui um valor menor ou igual ao valor de seu nó pai
 - O nó filho à direita possui um valor maior do que o valor de seu nó pai
 - Exemplo: deve ser gerada a árvore abaixo ao adicionar 4,2,1,3,8,9,6,5,7.

```
Arvore *a = cria();  
adiciona(a,4);  
adiciona(a,2);  
adiciona(a,1);  
adiciona(a,3);  
adiciona(a,8);  
adiciona(a,9);  
adiciona(a,6);  
adiciona(a,5);  
adiciona(a,7);
```



Exercícios

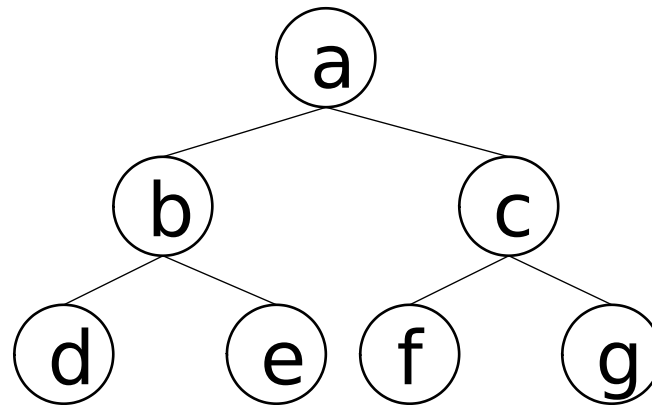
5. Uma árvore *unival* (valor universal) é uma árvore na qual todos os nós abaixo dela possuem o mesmo valor. Dada a raiz de uma árvore binária, conte o número de subárvores *unival*
- Exemplo, a árvore abaixo possui 5 subárvores *unival*



Exercícios

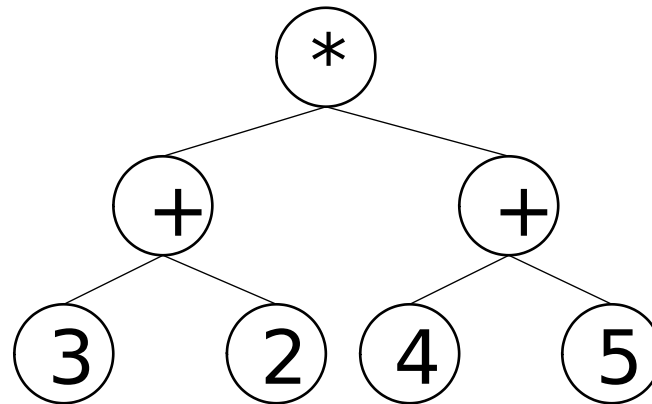
6. Dadas duas listas de nós de uma mesma árvore binária geradas em *pre-order* e *in-order*, escreva um programa capaz de reconstruir a árvore.

- Exemplo, *pre-order*: [a,b,d,e,c,f,g] e *in-order*: [d,b,e,a,f,c,g], a saída esperada é:



Exercícios

7. Suponha uma expressão aritmética representada como uma árvore binária. Cada folha é um número inteiro e cada nó interno representa uma operação, como +, -, * ou /. Dada a raiz da árvore, escreva um programa que avalia a expressão.
- Exemplo, dada a árvore abaixo a saída é 45 pois $(3 + 2) * (4 + 5)$



PRA – Projeto de Arquivos

Introdução a árvores binárias

Prof. Allan Rodrigo Leite