Algoritmos e Estrutura de Dados

Árvores AVL

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Faculdade de Engenharia

Agenda

Exercício

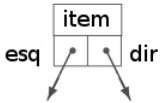
Vamos implementar a estrutura de dados Árvore AVL.

Árvore AVL

Implementação

Uma Árvore AVL é formada pela estrutura **nodo**, que contém três campos:

- Um ponteiro **esq**, que indica o filho da esquerda daquele nodo.
- Um ponteiro **dir**, que indica o filho da direita daquele nodo.
- Um campo **item** do tipo **int**, que é o tipo de dado a ser armazenado no nodo da árvore.



Árvore AVL

Estrutura Nodo

Segue o pseudo-código referente à estrutura nodo:

Algoritmo 1: Nodo

```
1 início
2     registro {
3         Inteiro: item;
4         Inteiro: altura;
5         Ponteiro Nodo: esq;
6         Ponteiro Nodo: dir;
7     } Nodo;
```

Árvore AVL

Estrutura Nodo

Na linguagem C, a implementação fica conforme o código a seguir:

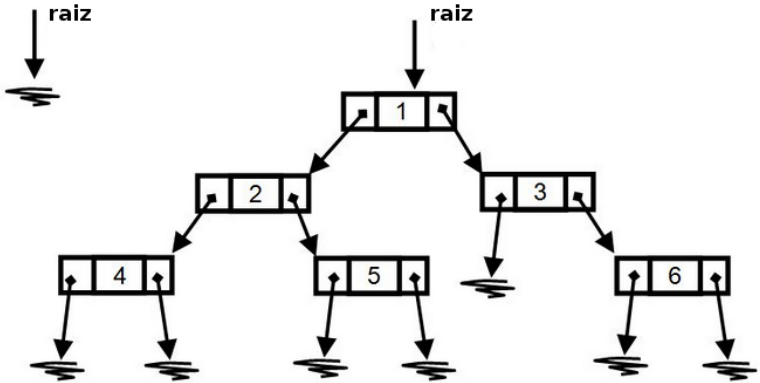
```c
typedef struct nodo_t
{
    int item;
    int altura;
    struct nodo_t *esq;
    struct nodo_t *dir;
} nodo;
```

Árvore AVL

Estrutura AVL

A estrutura **Árvore AVL** trata-se de um ponteiro do tipo **nodo**:

- O ponteiro **raiz** aponta para o nodo raiz da árvore.
- Se a árvore está vazia, o ponteiro **raiz** aponta para NULL.



Algoritmo 2: Arvore

```
1 início
2     Tipo Nodo* Arvore;
```

Na linguagem C, a implementação fica conforme o código a seguir:

```c
typedef nodo* ArvAVL;
```

Árvore AVL

Inserção

Agora, vamos implementar dois algoritmos básicos:

- **CriaÁrvore**: cria uma árvore vazia.
- **ArvoreVazia**: verifica se uma árvore está vazia.

CriaÁrvore | ÁrvoreVazia

Pseudo-Código

Algoritmo 3: CriaÁrvore

Saída: Ponteiro *r* para raiz.

```
1 início
2     r ← NULL
```

Algoritmo 4: ÁrvoreVazia

Entrada: Ponteiro *r* para raiz.
Saída: V ou F

```
1 início
2     retorna (r = NULL)
```

CriaÁrvoreAVL | ÁrvoreVazia

Código-Fonte

Na linguagem C, a implementação fica conforme o código a seguir:

```
1  ArvAVL* cria_arvore_avl() {
2      ArvAVL* raiz = (ArvAVL*) malloc(sizeof(ArvAVL));
3      if(raiz != NULL)
4          *raiz = NULL;
5      return raiz;
6  }
7
8  int arvore_vazia(nodo *r) {
9      return r == NULL;
10 }
```

Altura Nodo | Fator Balanceamento | Maior

Introdução

Agora, vamos implementar dois algoritmos básicos:

- **AlturaNodo**: retorna a altura de um nodo.
- **FatorBalanceamento**: retorna o fator de balaceamento de um nodo.
- **Maior**: retorna compara dois elementos, retornando o maior.

Altura Nodo

Pseudo-Código

Algoritmo 5: AlturaNodo

Entrada: Ponteiro para o nodo n.
Saída: Altura do nodo n.

```
1 início
2     se (n = NULL) então
3         retorna ( -1 )
4     senão
5         retorna n.altura
```

Adaptado de [Backes 2016]

Altura Nodo

Código-Fonte

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 int altura_nodo(nodo* no){
2     if(no == NULL)
3         return -1;
4     else
5     return no->altura;
6 }
```

Fator de Balanceamento

Pseudo-Código

Algoritmo 6: FatorBalanceamento

Entrada: Ponteiro para o nodo *n*.

```
1 início
2     retorna abs(AlturaNodo(n.esq) - AlturaNodo(n.dir))
```

Fator de Balanceamento

Código-Fonte

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 int fator_balanceamento_nodo(nodo* no){
2     return labs(altura_nodo(no->esq) - altura_nodo(no->dir));
3 }
```

Maior

Código-Fonte

Na linguagem C, a implementação da função **Maior** fica conforme o código a seguir:

```
1 int maior(int x, int y){
2     if(x > y)
3         return x;
4     else
5         return y;
6 }
```

Rotação LL

Pseudo-Código

Algoritmo 7: RotaçãoLL

Entrada: Ponteiro para o nodo desbalanceado A.

```
1 início
2     B ← A.esq
3     A.esq ← B.dir
4     B.dir ← A
5     A.altura ← maior(AlturaNodo(A.esq), AlturaNodo(A.dir)) + 1
6     B.altura ← maior(AlturaNodo(B.esq), A.altura) + 1
7     A ← B
```

Adaptado de [Backes 2016]

Rotação LL

Código-Fonte

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void rotacao_LL(ArvAVL *A){
2     nodo *B;
3     B = (*A)->esq;
4     (*A)->esq = B->dir;
5     B->dir = *A;
6     (*A)->altura = maior(altura_nodo((*A)->esq),altura_nodo((*A)->dir)) + 1;
7     B->altura = maior(altura_nodo(B->esq),(*A)->altura) + 1;
8     *A = B;
9 }
```

Rotação RR

Pseudo-Código

Algoritmo 8: RotaçãoRR

Entrada: Ponteiro para o nodo desbalanceado A.

```
1 início
2     B ← A.dir
3     A.dir ← B.esq
4     B.esq ← A
5     A.altura ← maior(AlturaNodo(A.esq), AlturaNodo(A.dir)) + 1
6     B.altura ← maior(AlturaNodo(B.dir), A.altura) + 1
7     A ← B
```

Adaptado de [Backes 2016]

Rotação RR

Código-Fonte

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void rotacao_RR(ArvAVL *A){
2     nodo *B;
3     B = (*A)->dir;
4     (*A)->dir = B->esq;
5     B->esq = (*A);
6     (*A)->altura = maior(altura_nodo((*A)->esq),altura_nodo((*A)->dir)) + 1;
7     B->altura = maior(altura_nodo(B->dir),(*A)->altura) + 1;
8     (*A) = B;
9 }
```

Rotação RL

Algoritmo 9: RotaçãoRL

Entrada: Ponteiro para a raiz r.

```
1 início
2     RotaçãoLL(r.dir)
3     RotaçãoRR(r)
```

Adaptado de [Backes 2016]

Rotação RL

Código-Fonte

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void rotacao_RL(ArvAVL *A){
2     rotacao_LL(&(*A)->dir);
3     rotacao_RR(A);
4 }
```

Rotação LR

Algoritmo 10: RotaçãoLR

Entrada: Ponteiro para a raiz r.

```
1 início
2     RotaçãoRR(r.esq)
3     RotaçãoLL(r)
```

Adaptado de [Backes 2016]

Rotação LR

Código-Fonte

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void rotacao_LR(ArvAVL *A){
2     rotacao_RR(&(*A)->esq);
3     rotacao_LL(A);
4 }
```

Inserir

Pseudocódigo

Algoritmo 11: InserirAVL

Entrada: Ponteiro para a raiz r, item x.
Saída: V ou F

```
1 início
2     se ( r = NULL) então
3         novo ← ALOCA_NODO()
4         novo.item ← x
5         novo.altura ← 0
6         novo.esq ← NULL
7         novo.dir ← NULL
8         r ← novo
9         retorna Verdadeiro
```

Inserir

Pseudocódigo InserirAVL

Algoritmo 12: InserirAVL (continuação)

```
10 início
11     atual ← r
12     se (x < atual.item) então
13         se (InserirAVL(atual.esq, x)) então
14             se (FatorBalanceamento(atual) ≥ 2) então
15                 se (x < r.esq.item) então
16                     RotacaoLL(r)
17                 senão
18                     RotacaoLR(r)
19     senão
20         se (x > atual.item) então
21             se (InserirAVL(atual.dir, x)) então
22                 se (FatorBalanceamento(atual) ≥ 2) então
23                     se (r.dir.item < x) então
24                         RotacaoRR(r)
25                     senão
26                         RotacaoRL(r)
27         senão
28             retorna Falso
29     atual.alt ← maior(AlturaNodo(atual.esq), AlturaNodo(atual.dir)) + 1
```

Adaptado de [Backes 2016]

Inserir

Código-Fonte - Parte 1

Na linguagem C, a implementação fica conforme o código a seguir:

```
1  int insere_ArvAVL(ArvAVL *raiz, int valor){
2      int res;
3      if (*raiz == NULL){
4          nodo *novo;
5          novo = (nodo*)malloc(sizeof(nodo));
6          if(novo == NULL)
7              return 0;
8
9          novo->item = valor;
10         novo->altura = 0;
11         novo->esq = NULL;
12         novo->dir = NULL;
13         *raiz = novo;
14         return 1;
15     }
16     nodo *atual = *raiz;
17     /** Continua no proximo slide **/
```

Inserir

Código-Fonte - Parte 2

```
16      nodo *atual = *raiz;
17      /** Continuacao do slide anterior **/
18      if (valor < atual->item){
19          if ((res = insere_ArvAVL(&(atual->esq), valor)) == 1){
20              if (fator_balanceamento_nodo(atual) >= 2){
21                  if (valor < (*raiz)->esq->item ){
22                      rotacao_LL(raiz);
23                  } else {
24                      rotacao_LR(raiz);
25                  }
26              }
27          }
28      } else {
29          if (valor > atual->item){
30          /** Continua no proximo slide **/
```

Inserção

Código-Fonte - Parte 3

```
29          if (valor > atual->item){
30              /** Continuacao do slide anterior **/
31              if ((res = insere_ArvAVL(&(atual->dir), valor)) == 1){
32                  if (fator_balanceamento_nodo(atual) >= 2){
33                      if((*raiz)->dir->item < valor) {
34                          rotacao_RR(raiz);
35                      } else {
36                          rotacao_RL(raiz);
37                      }
38                  }
39              }
40          } else {
41              printf("Valor duplicado!!\n");
42              return 0;
43          }
44      }
45      atual->altura=maior(altura_nodo(atual->esq),altura_nodo(atual->dir))+1;
46      return res;
47  }
```

Remover

Algoritmo ProcuraMaior

Algoritmo 13: ProcuraMaior

Entrada: Ponteiro para o nodo **atual**.
Saída: Ponteiro para o maior valor a partir do nodo **atual**.

```
1 início
2     n1 ← atual
3     n2 ← atual.dir
4     enquanto (n2≠ NULL) faça
5         n1 ← n2
6         n2 ← n2.dir
7     retorna n1
```

Adaptado de [Backes 2016]

Remover

Algoritmo RemoverAVL

Algoritmo 14: RemoverAVL

Entrada: Ponteiro para a raiz r, item x.
Saída: V ou F

```
1  início
2      se (r = NULL) então
3          retorna Falso
4      se (x < r.item) então
5          se (RemoverAVL(r.esq, x)) então
6              se (FatorBalanceamento(r) ≥ 2) então
7                  se (AlturaNodo(r.dir.esq) ≤ AlturaNodo(r.dir.dir)) então
8                      RotacaoRR(r)
9                  senão
10                     RotacaoRL(r)
11     se (x > r.item ) então
12         se (RemoverAVL(r.dir, x)) então
13             se (FatorBalanceamento(r) ≥ 2) então
14                 se (AlturaNodo(r.esq.dir) ≤ AlturaNodo(r.esq.esq)) então
15                     RotacaoLL(r)
16                 senão
17                     RotacaoLR(r)
```

Adaptado de [Backes 2016]

Remover

Algoritmo RemoverAVL

Algoritmo 15: RemoverAVL (Continuação)

```
18 início
19     se (r.item = x) então
20         se (r.esq = NULL) ou (r.dir = NULL) então
21             removerNodo ← r
22             se (r.esq ≠ NULL) então
23                 r ← r.esq
24             senão
25                 r ← r.dir
26             DESALOCA_NODO(removerNodo)
27         senão
28             antecessor ← ProcuraMaior(r.dir)
29             r.item ↔ antecessor.item // Troca os valores.
30             RemoverAVL(r.dir, antecessor.item)
31             se (FatorBalanceamento(r) ≥ 2) então
32                 se AlturaNodo(r.esq.dir) ≤ AlturaNodo(r.esq.esq) então
33                     RotaçãoLL(r)
34                 senão
35                     RotaçãoLR(r)
36             se (r ≠ NULL) então
37                 r.altura ← maior(AlturaNodo(r.esq), AlturaNodo(r.dir)) + 1
38             retorna Verdadeiro
39         r.altura ← maior(AlturaNodo(r.esq), AlturaNodo(r.dir)) + 1
40         retorna Verdadeiro
```

Adaptado de [Backes 2016]

Remover

Código-Fonte - Parte 1

Na linguagem C, a implementação fica conforme o código a seguir:

```
1  int remove_ArvAVL(ArvAVL *raiz, int valor){
2    if(*raiz == NULL){
3        printf("Valor nao existe!!\n");
4        return 0;
5    }
6    int res;
7    if(valor < (*raiz)->item){
8        if((res = remove_ArvAVL(&(*raiz)->esq,valor)) == 1){
9              if(fator_balanceamento_nodo(*raiz) >= 2){
10                 if(altura_nodo((*raiz)->dir->esq) <= altura_nodo((*raiz)->dir->dir))
11                     rotacao_RR(raiz);
12                 else
13                     rotacao_RL(raiz);
14             }
15       }
16   }
17     /** Continua no proximo slide **/
```

Remover

Código-Fonte - Parte 2

```
17    /** Continuacao do slide anterior **/
18  if((*raiz)->item < valor){
19      if((res = remove_ArvAVL(&(*raiz)->dir, valor)) == 1){
20              if(fator_balanceamento_nodo(*raiz) >= 2){
21                  if(altura_nodo((*raiz)->esq->dir) <= altura_nodo((*raiz)->esq->esq) )
22                      rotacao_LL(raiz);
23                  else
24                      rotacao_LR(raiz);
25              }
26      }
27  }
28  if((*raiz)->item == valor){
29      if(((*raiz)->esq == NULL || (*raiz)->dir == NULL)){
30              nodo *oldNode = (*raiz);
31              if((*raiz)->esq != NULL)
32                  *raiz = (*raiz)->esq;
33              else
34                  *raiz = (*raiz)->dir;
35              free(oldNode);
36        } else {
37              /** Continua no proximo slide **/
```

Remover

Código-Fonte - Parte 3

```
36          } else {
37              /** Continuacao do slide anterior **/
38              nodo* temp = procura_menor((*raiz)->dir);
39              (*raiz)->item = temp->item;
40              remove_ArvAVL(&(*raiz)->dir, (*raiz)->item);
41              if(fator_balanceamento_nodo(*raiz) >= 2){
42                  if(altura_nodo((*raiz)->esq->dir) <= altura_nodo((*raiz)->esq->esq))
43                      rotacao_LL(raiz);
44                  else
45                      rotacao_LR(raiz);
46              }
47          }
48          if (*raiz != NULL)
49              (*raiz)->altura = maior(altura_nodo((*raiz)->esq), altura_nodo((*raiz)->dir)) + 1;
50          return 1;
51      }
52      (*raiz)->altura = maior(altura_nodo((*raiz)->esq), altura_nodo((*raiz)->dir)) + 1;
53      return res;
54  }
```

Função Main

Introdução

Por fim, falta apenas criar uma **função Main** para manipular nossa estrutura de dados Árvore AVL.

Função Main

Código-Fonte Apagar Árvore

```
1  int main() {
2      int rand_max = 10; int vaux = 0;
3      srand(42);
4      ArvAVL *arvore_avl;
5      arvore_avl = cria_arvore_avl();
6      for (int i = 0; i < 20; i++) {
7          insere_ArvAVL(arvore_avl, rand() % rand_max);
8      }
9      printf("\nPercurso Pre-Ordem\n");
10     pre_ordem(arvore_avl);
11     printf("\nPercurso Em-Ordem\n");
12     em_ordem(arvore_avl);
13     printf("\nPercurso Pos-Ordem\n");
14     pos_ordem(arvore_avl);
15     vaux = rand() % rand_max;
16     printf("\nRemover valor %d\n", vaux);
17     remove_ArvAVL(arvore_avl, vaux);
18
19     apagar_ArvAVL(arvore_avl);
20     return 0;
21 }
```

Referências

Bibliografia Básica

- Bibliografia Básica


BACKES, A. *Estrutura de Dados Descomplicada - Em Linguagem C*. 1. ed. São Paulo: Elsevier, 2016.

- Material Complementar
  - Código-fonte e listas de exercícios - Material disponível on-line
- Vídeo-aulas
  - Canal do youtube do prof. André Backes.
  - Vídeo-aula: Estrutura de dados - Univesp
- Animação
  - Árvore AVL - USFCA
  - Árvore AVL - CS - Armstrong

Algoritmos e Estrutura de Dados

Árvores AVL

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Faculdade de Engenharia