Universidade Federal de Goiás
Goiânia 06 de Junho de 2018
Edson Júnior Frota Silva
Engenharia de Computação
Estrutura de Dados II

Matrícula: 201515412
Prof°(a): Renata Dutra

Árvores de Pesquisa Binária

13.1 Crie o arquivo `arv.h`, com os tipos e funções para árvores definidos nesse capítulo (exceto a função `emnivel()`, que depende do tipo `Fila`), e use esse arquivo num programa que cria e exibe a árvore da Figura 13.5.

```c
#include<stdio.h>
#include<stdlib.h>
#define fmt "%d "
typedef int Item;
typedef struct arv
{
    struct arv *esq;
    Item item;
    struct arv *dir;
} *Arv;

Arv arv(Arv e, Item x, Arv d)
{
    Arv n = (struct arv * )malloc(sizeof(struct arv));
    n -> esq = e;
    n -> item = x;
    n -> dir = d;
    return n;
}

void EmOrdem(Arv A)
{
    if (A == NULL) return;
    EmOrdem(A -> esq);
    printf(fmt , A -> item);
    EmOrdem(A -> dir);
}

void PreOrdem(Arv A)
{
    if (A == NULL) return;
    printf(fmt, A -> item);
    PreOrdem(A -> esq);
    PreOrdem(A -> dir);
}

void PosOrdem(Arv A)
{
    if (A == NULL) return;
    PosOrdem(A -> esq);
    PosOrdem(A -> dir);
    printf(fmt, A -> item);
}


void Destroi(Arv *A)
{
    if (*A == NULL) return;
    Destroi (&(*A) -> esq);
    Destroi (&(*A) -> dir);
    free(*A);
    *A = NULL;
}

void Ins(Item x, Arv *A)
{
    if(*A == NULL) *A = arv(NULL, x, NULL);
    else if(x <= (*A) -> item) Ins(x,&(*A) -> esq);
    else Ins(x, &(*A) -> dir);
```

```c
}

int Busca(Item x, Arv A)
{
    if (A == NULL) return 0;
    if(x == A -> item)  return 1;
    if(x <= A -> item) return Busca(x, A -> esq);
    else return Busca(x, A -> dir);
}

Item RemMax(Arv *A)
{
    if (*A == NULL) abort();
    while((*A) -> dir != NULL) A = &(*A) -> dir;
    Arv n = *A;
    Item x = n -> item;
    *A = n -> esq;
    free(n);
    return x;
}

void Rem(Item x, Arv *A)
{
    if (*A == NULL) return;
    if (x == (*A)->item)
    {
        Arv n = *A;
        if(n->esq == NULL) *A = n->dir;
        else if( n->dir == NULL) *A = n->esq;
        else n->item = RemMax(&n -> esq);
        if(n != *A) free(n);
    }
    else if(x <= (*A)->item) Rem(x, &(*A)->esq);
    else Rem(x,&(*A)->dir);
}


int NumNos(Arv *A)
{
    if (*A == NULL) return 0;
    Arv n = *A;
    int num = 0;

}

int main()
{
    Arv R = arv(arv(arv(NULL, 4, NULL),2,arv(NULL,5,NULL)),1,arv(NULL,3,arv(NULL,6,NULL)));
    EmOrdem(R);
}
```

13.2 Crie a função `nos(A)`, que devolve o total de *nós* na árvore binária `A`.

```c
#include<stdio.h>
typedef struct No_A
{
    int valor;
    struct No_A *esquerda;
    struct No_A *direita;
} No_A;

No_A* raiz = NULL;
int y = 0;

int inserir_elemento( int numero )
{
    No_A* novo = ( No_A *)malloc(sizeof(No_A));
    if( novo == NULL )
        return (0);

    novo -> valor = numero;
    novo -> esquerda = novo -> direita = NULL;

    if( raiz == NULL )
    {
```

```
        raiz = novo;
        return (1);
    }

    No_A* pai = NULL;
    No_A* x = raiz;
    while( x != NULL )
    {
        pai = x;
        if( x -> valor > numero )
        {
            x = x -> esquerda;
        }

        else
        {
            x = x -> direita;
        }
    }
    if( pai -> valor > numero )
    {
        pai -> esquerda = novo;
    }
    else
    {
        pai -> direita = novo;
    }
    return (1);
}

void nos( No_A *raiz )
{
    y = y + 1;
}

void ordem(No_A *x)
{
    if( x != NULL)
    {

        ordem( x -> esquerda );
        nos( raiz );
        ordem( x -> direita );

    }
}
int main()
{
    int x;
    for(x = 4; x < 10; x++)
    {
        inserir_elemento( x+1 );
    }

    ordem( raiz );

    printf("\n A arvore possui um total de %i nos.\n", y);

    return(0);
}
```

13.3 Crie a função `folhas(A)`, que devolve o total de *folhas* na árvore binária `A`.

```
#include<stdio.h>

typedef struct No_A
{
    int valor;
    struct No_A *esquerda;
    struct No_A *direita;
} No_A;

No_A* raiz = NULL;
```

```c
int y = 0;

int inserir_elemento( int num )
{
    No_A* novo = ( No_A * )malloc(sizeof( No_A ));
    if( novo == NULL )
        return (0);

    novo -> valor = num;
    novo -> esquerda = novo -> direita = NULL;

    if( raiz == NULL )
    {
        raiz = novo;
        return (1);
    }

    No_A* pai = NULL;
    No_A* x = raiz;
    while( x != NULL )
    {
        pai = x;
        if( x -> valor > num )
        {
            x = x -> esquerda;
        }


        else
        {
            x = x->direita;
        }
    }
    if( pai -> valor > num )
    {
        pai -> esquerda = novo;
    }
    else
    {
        pai -> direita = novo;
    }

    return (1);

}

int Total_de_Folhas( No_A *x )
{
    if( x == NULL )
    {
        return (0);
    }

    if( ( x->esquerda == NULL) && (x->direita == NULL ) )
    {
        return (1);
    }
    return ( Total_de_Folhas( x->esquerda ) + Total_de_Folhas( x -> direita ));
}

int main()
{

    int x;
    for( x = 4; x < 10; x++ )
    {

        inserir_elemento( x+1 );

    }

    int y = Total_de_Folhas( raiz );

    printf("\n Total de folhas da arvore: %i.\n", y);

    return (0);

 }
```

13.4 Crie a função `altura(A)`, que devolve a *altura* da árvore binária `A`.

```c
#include<stdio.h>
typedef struct No_A
{
    int valor;
    struct No_A *esquerda;
    struct No_A *direita;
} No_A;

No_A* raiz = NULL;
int y = 0;

int inserir_elemento( int numero )
{
    No_A* novo = ( No_A *)malloc(sizeof(No_A));
    if( novo == NULL )
        return (0);

    novo -> valor = numero;
    novo -> esquerda = novo -> direita = NULL;

    if( raiz == NULL )
    {
        raiz = novo;
        return (1);
    }

    No_A* pai = NULL;
    No_A* x = raiz;
    while( x != NULL )
    {
        pai = x;
        if( x -> valor > numero )
        {
            x = x -> esquerda;
        }

        else
        {
            x = x -> direita;
        }
    }
    if( pai -> valor > numero )
    {
        pai -> esquerda = novo;
    }
    else
    {
        pai -> direita = novo;
    }
    return (1);
}

int Maior(int a, int b)
{
    if(a > b)
    {
        return (a);
    }
    else
    {
        return b;
    }

}

int Altura_Arvore(No_A *p)
{
    if((p == NULL) || (p->esquerda == NULL && p->direita == NULL))
    {
        return (0);
    }
    else
    {
        return (1 + maior(altura(p->esquerda), altura(p->direita)));
    }
}
```

```
77
78  int main()
79  {
80
81      int x;
82      for(x = 4; x < 10; x++)
83      {
84
85          inserir( x + 1 );
86
87      }
88
89      y = Altura_Arvore( raiz );;
90
91      printf("\n Altura da arvore: %i.\n", y);
92
93  }
```

13.5 Crie a função `tem(A,x)`, que informa se a árvore binária `A` tem o item `x`.

```
1   #include<stdio.h>
2   #include<stdlib.h>
3   #include<locale.h>
4
5   typedef struct No{
6
7       int valor;
8       struct No *dir;
9       struct No *esq;
10
11  }No;
12
13  struct No *root = NULL;
14
15  int inserir(int num){
16
17      No* novo = (No *)malloc(sizeof(No));
18      if(novo == NULL)
19          return 0;
20
21      novo->valor = num;
22      novo->esq = novo->dir = NULL;
23
24      if(root == NULL){
25
26          root = novo;
27          return 1;
28
29      }
30
31      No* pai = NULL;
32      No* p = root;
33
34      while(p != NULL)
35          {
36              pai = p;
37              if(p->valor > num)
38                      p = p->esq;
39
40                  else
41                      p = p->dir;
42          }
43          if(pai->valor > num)
44              pai->esq = novo;
45          else
46              pai->dir = novo;
47
48          return 1;
49
50  }
51
52  No* buscar(int n)
53  {
54      No* p = root;
55      while(p != NULL)
56      {
```

```c
        if(p->valor == n)
            return p;

        if(p->valor > n)
            p = p->esq;
        else
            p = p->dir;
    }

    return NULL;
}

int main(){

    setlocale(LC_ALL, "Portuguese");
    int op, n, n1;

    do{

    printf("\nOpções: \n");
    printf("  [1] Inserir\n");
    printf("  [2] Buscar um elemento da lista\n");
    printf("  [3] Sair\n");
    scanf("%d", &op);
    system("cls");

    switch(op){

        case 1: printf("OPÇÃO 1\n");
            printf("Digite um número: ");
            scanf("%d", &n);
            inserir(n);
            system("cls");
            break;

        case 2: printf("OPÇÃO 2\n");
            printf("Digite o número a ser buscado: ");
            scanf("%d", &n1);
            if(buscar(n) == NULL)
                        printf("Número NÃO foi encontrado na árvore\n");
                    else
                        printf("Número foi encontrado na árvore\n");
            system("Cls");
            break;


    }

    }while(op != 3 );


    return 0;
}
```

13.6 Uma árvore `A` é *estritamente binária* se cada nó em `A` é uma folha ou tem dois filhos. Crie a função `eb(A)`, que informa a árvore `A` é estritamente binária.

```c
#include<stdio.h>
#include<stdlib.h>
#include<locale.h>

typedef struct No{

    int valor;
    struct No *esq;
    struct No *dir;

}No;

struct No *root = NULL;
```

```c
int aux = 0;

int inserir(int num){

    No* novo = (No *)malloc(sizeof(No));
    if(novo == NULL)
        return 0;

    novo->valor = num;
    novo->esq = novo->dir = NULL;

    if(root == NULL){

        root = novo;
        return 1;

    }

    No* pai = NULL;
    No* p = root;

    while(p != NULL)
        {
            pai = p;
            if(p->valor > num)
                    p = p->esq;

                else
                    p = p->dir;
        }
        if(pai->valor > num)
            pai->esq = novo;
        else
            pai->dir = novo;

        return 1;

}

void noFolha(){

    if(root == NULL)
        printf("A árvore está vazia\n");
    else
        folha(root);
}

void folha(No *p){

    if(p != NULL){

        folha(p->esq);
        if(p->esq == NULL)
            aux++;

        folha(p->dir);

        if(p->dir == NULL)
            aux++;

    }

}

int main(){

    setlocale(LC_ALL, "Portuguese");
    int op, n;
    do{

    printf("\nOpções: \n");
    printf("  [1] Inserir\n");
    printf("  [2] Quantidade de nós folha na árvore\n");
    printf("  [3] Sair\n");
    scanf("%d", &op);
    system("cls");

    switch(op){

        case 1: printf("OPÇÃO 1\n");
```

```
            printf("Digite um número: ");
            scanf("%d", &n);
            inserir(n);
            system("cls");
            break;

        case 2:
                noFolha();
                printf("%d", aux);
                aux = 0;
                break;

    }

    }while(op != 3 );


    return 0;
}
```

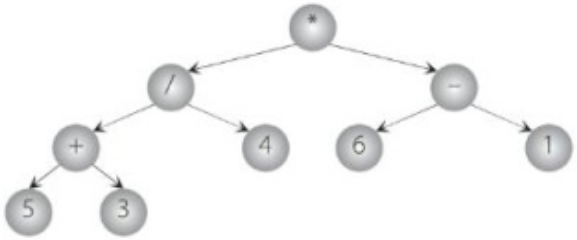
13.7 Duas árvores binárias `A` e `B` são *iguais* se elas têm a mesma forma e os mesmos itens. Crie a função `igual(A,B)`, que informa se `A` é igual a `B`.

```
#include<stdio.h>
#include<stdlib.h>
#include<locale.h>

typedef struct No{

    int valor;
    struct No *dir;
    struct No *esq;

}No;

struct No *root = NULL;
struct No *root1 = NULL;

int inserir1(int num){

    No* novo = (No *)malloc(sizeof(No));
    if(novo == NULL)
        return 0;

    novo->valor = num;
    novo->esq = novo->dir = NULL;

    if(root == NULL){

        root = novo;
        return 1;

    }

    No* pai = NULL;
    No* p = root;

    while(p != NULL)
        {
            pai = p;
            if(p->valor > num)
                    p = p->esq;

                else
                    p = p->dir;
        }
        if(pai->valor > num)
            pai->esq = novo;
        else
            pai->dir = novo;

        return 1;
```

```
 50
 51  }
 52
 53  int inserir2(int num){
 54
 55      No* novo1 = (No *)malloc(sizeof(No));
 56      if(novo1 == NULL)
 57          return 0;
 58
 59      novo1->valor = num;
 60      novo1->esq = novo1->dir = NULL;
 61
 62      if(root1 == NULL){
 63
 64          root1 = novo1;
 65          return 1;
 66
 67      }
 68
 69      No* pai = NULL;
 70      No* p = root1;
 71
 72      while(p != NULL)
 73          {
 74              pai = p;
 75              if(p->valor > num)
 76                      p = p->esq;
 77
 78                  else
 79                      p = p->dir;
 80          }
 81          if(pai->valor > num)
 82              pai->esq = novo1;
 83          else
 84              pai->dir = novo1;
 85
 86          return 1;
 87
 88  }
 89
 90  int verificarIgualdade(No *p, No *q){
 91
 92      if( p == NULL && q == NULL)
 93          return 1;
 94      if( p == NULL || q == NULL)
 95          return 0;
 96      if(( p->valor == q->valor) && verificarIgualdade( p->esq, q->esq ) && verificarIgualdade(
p->dir, q->dir ));
 97          return 1;
 98  }
 99
100
101  int main(){
102
103      setlocale(LC_ALL, "Portuguese");
104      int op, n, n1;
105
106      do{
107
108      printf("\nOpções: \n");
109      printf("  [1] Inserir elemento\n");
110      printf("  [2] Verificar se as duas árvores são iguais\n");
111      printf("  [3] Sair\n");
112      scanf("%d", &op);
113      system("cls");
114
115      switch(op){
116
117          case 1: printf("OPÇÃO 1\n");
118              printf("Digite um número para a PRIMEIRA árovre: ");
119              scanf("%d", &n);
120              inserir1(n);
121              printf("Digite um número para a SEGUNDA árvore: ");
122              scanf("%d", &n1);
123              inserir2(n1);
124              system("cls");
125              break;
126
127          case 2: printf("OPÇÃO 2\n");
128
```

```
129             verificarIgualdade(root, root1);
130
131             system("cls");
132             break;
133
134
135     }
136
137     }while(op != 3 );
138     return 0;
139 }
```

13.8 Uma expressão aritmética pode ser representada por uma árvore binária cuja raiz é uma operação e cujas subárvores são operandos. Por exemplo, a expressão `((5+3)/4)*(6-1)` pode ser representada como na figura a seguir. Crie a função `valor(A)`, que avalia uma expressão aritmética representada por uma árvore binária `A` (cujos nós guardam números inteiros).




```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<locale.h>
4
5  typedef struct No{
6      int num;
7      struct No *esq;
8      struct No *dir;
9  }No;
10
11 No* root = NULL;
12 int aux = 0;
13
14 int inserir(int n){
15     No* novo = (No *)malloc(sizeof(No));
16     if(novo == NULL)
17         return 0;
18
19     novo->num = n;
20     novo->esq = novo->dir = NULL;
21
22     if(root == NULL){
23         root = novo;
24         return 1;
25     }
26
27     No* pai = NULL;
28     No* p = root;
29     while(p != NULL){
30             pai = p;
31             if(p->num > n)
32                     p = p->esq;
33
34                 else
35                     p = p->dir;
36         }
37         if(pai->num > n)
```

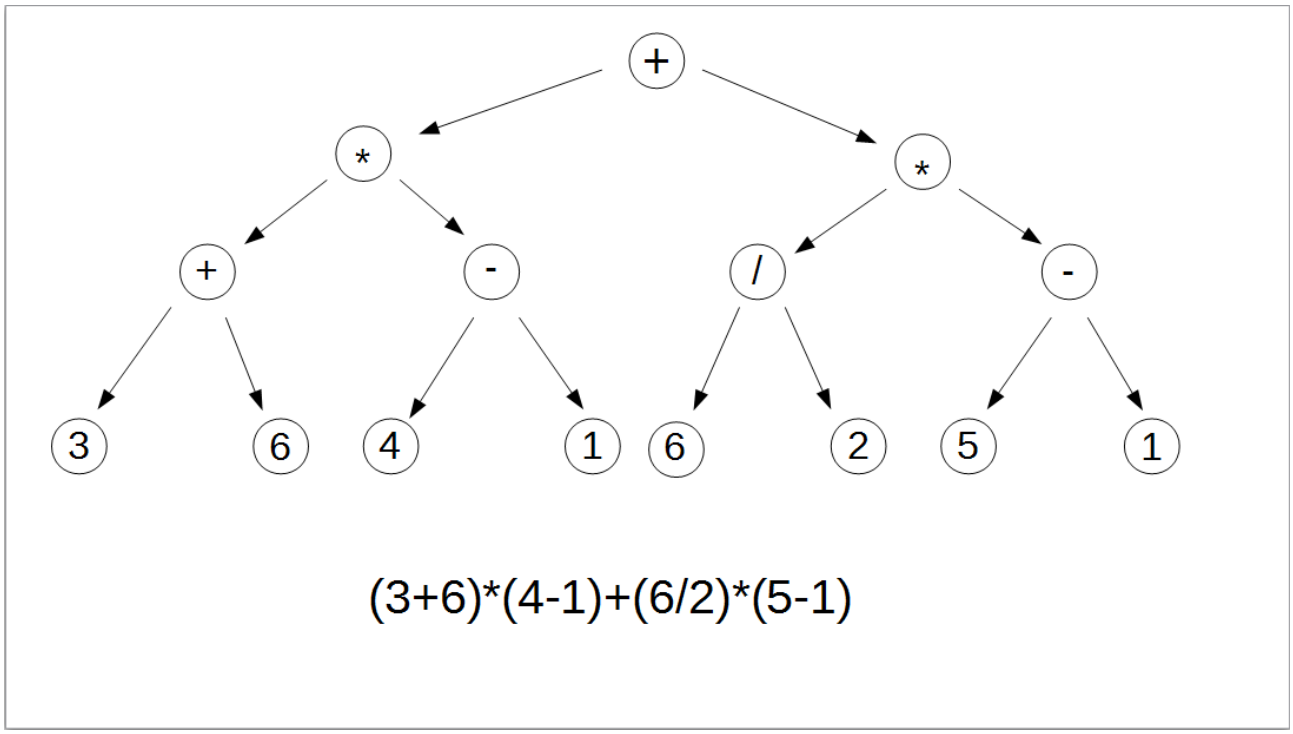
```
38              pai->esq = novo;
39          else
40              pai->dir = novo;
41
42          return 1;
43
44  }
45
46  void valor(No *p){
47      int resul;
48
49      if(p != NULL){
50
51          valor(p->esq);
52          resul = resul + p->num;
53          valor(p->dir);
54
55      }
56      printf("Resultado: %d", resul);
57  }
58
59  int main(){
60
62
63      inserir(*);
64      inserir(/);
65      inserir(+);
66      inserir(5);
67      inserir(3);
68      inserir(4);
69      inserir(-);
70      inserir(6);
71      inserir(1);
72
73      valor(root);
74
75  }
```

13.9 Crie a função `exibe_dec(A)`, que exibe os itens de uma árvore de busca binária em ordem decrescente.

```
1   #include<stdio.h>
2   #include<stdlib.h>
3
4   typedef struct No{
5
6       int valor;
7       struct No *esq;
8       struct No *dir;
9
10  }No;
11
12  struct No *root = NULL;
13
14  int inserir(int num){
15
16      No* novo = (No *)malloc(sizeof(No));
17      if(novo == NULL)
18          return 0;
19
20      novo->valor = num;
21      novo->esq = novo->dir = NULL;
22
23      if(root == NULL){
24
25          root = novo;
26          return 1;
27
28      }
29
30      No* pai = NULL;
31      No* p = root;
32
```

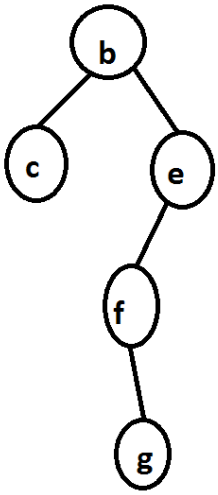
```c
    while(p != NULL)
        {
            pai = p;
            if(p->valor > num)
                    p = p->esq;

                else
                    p = p->dir;
        }
        if(pai->valor > num)
            pai->esq = novo;
        else
            pai->dir = novo;

        return 1;

}

void mostrar(){

    if(root == NULL)
        printf("A árvore está vazia\n");
    else
        ordemDecrescente(root);
}

void ordemDecrescente(No *p){

    if(p != NULL){

        ordemDecrescente(p->dir);
        printf("%d->", p->valor);
        ordemDecrescente(p->esq);


    }

}

int main(){

    setlocale(LC_ALL, "Portuguese");
    int op, n;

    do{

    printf("\nOpções: \n");
    printf("  [1] Inserir\n");
    printf("  [2] Exibir elementos da árvore em ordem decrescente: \n");
    printf("  [3] Sair\n");
    scanf("%d", &op);
    system("cls");

    switch(op){

        case 1:

            printf("OPÇÃO 1\n");
            printf("Digite um número: ");
            scanf("%d", &n);
            inserir(n);
            system("cls");
            break;

        case 2:

            printf("OPÇÃO 2\n");
            mostrar();
            break;

    }

    }while(op != 3 );


    return 0;

 }
```

10. **(Valor 10) Faça uma representação gráfica de uma árvore binária que represente a expressão aritmética (3+6)*(4-1)+ (6/2)*(5-1).**



11. Dado o seguinte programa, faça a representação gráfica da árvore criada.

```
#include <stdio.h>
struct arv {
char info;
struct arv* esq;
struct arv* dir;};
typedef struct arv Arv;
Arv* inicializa(void){
return NULL;}
Arv* cria(char c, Arv* sae, Arv* sad){
Arv* p=(Arv*)malloc(sizeof(Arv));
p->info = c;
p->esq = sae;
p->dir = sad;
return p;}
int vazia(Arv* a){
return a==NULL;}
void imprime (Arv* a){
if (!vazia(a)){
printf("%c ", a->info); /* mostra raiz */
imprime(a->esq); /* mostra sae */
imprime(a->dir); /* mostra sad */ }
}
int main(){
Arv* a1= cria('e',inicializa(),inicializa());
Arv* a2= cria('c',inicializa(),a1);
Arv* a3= cria('f',inicializa(),inicializa());
Arv* a4= cria('g',inicializa(),inicializa());
Arv* a5= cria('d',a3,a4);
Arv* a = cria('b',a2,a5 );
imprime(a);
}
```



UFG
Edson Júnior Frota Silva
Engenharia de Computação