Programação I

Árvores Binarias

Estrela Ferreira Cruz

1

Objetivos da aula

Objetivos da aula:

Estruturas de Dados não Lineares (árvores)

Árvores m-árias:

- Apresentação do conceito de árvore;
- Constituição de uma árvore;
- Grau, altura, tamanho e nível de uma árvore.

Árvores binárias:

- Árvore binária de pesquisa;
- Árvore binária balanceada;

Algoritmos de travessias da árvore: VLR, LVR e LRV.

2

2

Árvores

Como é constituída uma árvore?

3

3

Árvores

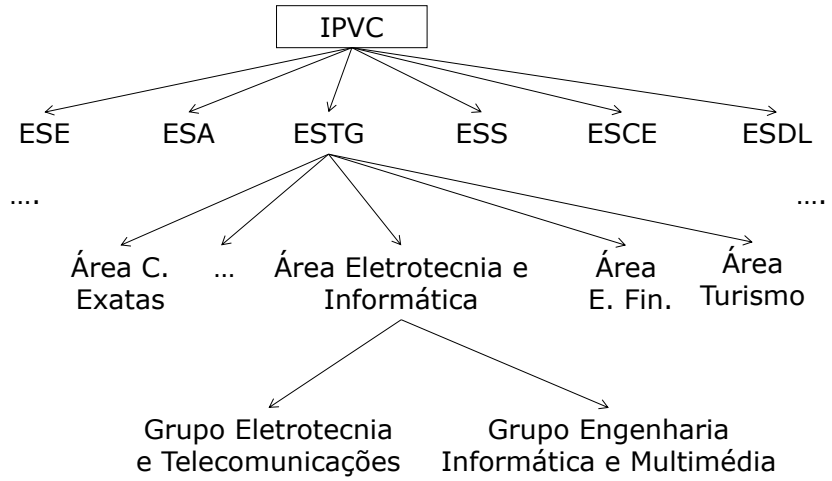
Estruturas de dados não lineares: árvores e grafos

- As estruturas de dados não lineares permitem representar informação não linear na forma hierárquica (árvores) e na forma de rede (grafos).
- Uma árvore é uma estrutura de dados não linear utilizada na representação de estruturas hierárquicas de informação, tais como organogramas, árvores genealógicas, entre outras hierarquias e taxonomias de informação.
- Uma árvore é constituída por nós (elementos que contêm informação) e arcos (liga dois nós).
- Um exemplo de uma árvore pode ser visto no slide seguinte onde se representa parte da estrutura hierárquica interna do IPVC (escolas, áreas científicas e grupos disciplinares).

4

4

Árvores



5

5

Árvores

Uma estrutura em **árvore** é um **conjunto de nós** com as seguintes características:

- raiz,
- **nós internos** ou ramificados,
- **nós folha** ou terminais.

Um **nó folha** ou **terminal** é um nó que **não tem descendentes**.

Um **nó interno** é um nó que não é folha.

Definição **recursiva de árvores**:

Uma **estrutura em árvore** é um conjunto finito de nós, tendo em conta os seguintes pressupostos:

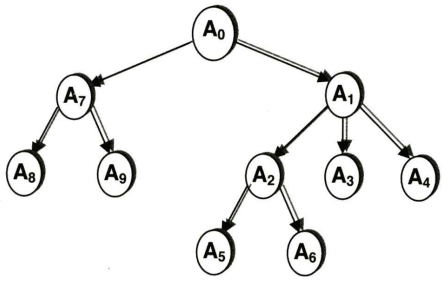
- Existe um **nó raiz** no topo da hierarquia
- Os restantes nós representam um número **finito de subárvores disjuntas** A1, A2, A3 ... An (n>=0). **Cada Ai é designado por subárvore do nó raiz.**

6

6

Árvores

- Na figura seguinte temos:
- Nó A0 – raiz da árvore
- As subárvores da raiz A0 estão representadas por S1 e S2, com a seguinte constituição:
- S1={A1,A2,A3,A4,A5,A6}
- S2={A7,A8,A9}
- Nós folhas={A8,A9,A5,A6,A3, A4}
- Nós internos ={A7,A0,A1,A2}



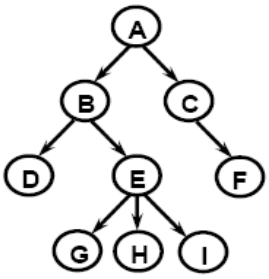
7

7

Árvores

Caminhos em árvores:

- Se um nó R é a RAIZ da árvore, então todo o nó, exceto R, tem exatamente uma ligação para ele a partir da raiz. Ou seja, existe um caminho único da raiz R para cada nó.
- O número de ligações que têm de ser seguidas até um determinado nó, é o comprimento do caminho.
- Se P é o pai de A então A é o filho de P.
- Um arco um dois nós. Se Pai é o nó superior de um arco então Filho é o nó inferior desse arco.



8

8

Árvores

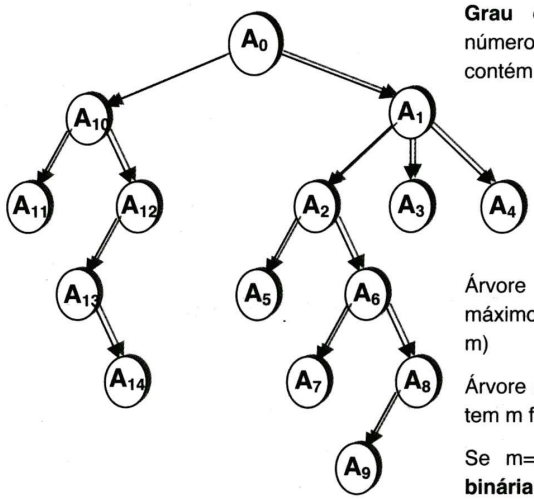
Alguns conceitos:

- O GRAU de um nó, representa o número de subárvores que o nó contém.
- O GRAU da árvore é o maior grau existente entre todos os nós.
- O nível de um nó P é o número de nós do caminho de P até à raiz da árvore.
- O nível do nó raiz de uma árvore é igual a 1.
- A altura de um nó P é o comprimento do caminho mais longo entre esse nó e as folhas da árvore (incluindo o próprio).
- A altura de uma árvore é a altura do seu nó raiz.
- Tamanho de um nó – número de descendentes desse nó mais ele próprio
- Tamanho da árvore – número de nós da árvore.
- Profundidade de um nó – comprimento do caminho da raiz até esse nó.

9

9

Árvores



Grau de um nodo representa o número de subárvores que o nodo contém.

O grau de um nodo folha é 0

O grau do nodo A_0 é 2

O grau do nodo A_1 é 3

O grau do nodo A_{13} é 1

Árvore **m-ária** – cada nodo tem no máximo m filhos (número de filhos <= m)

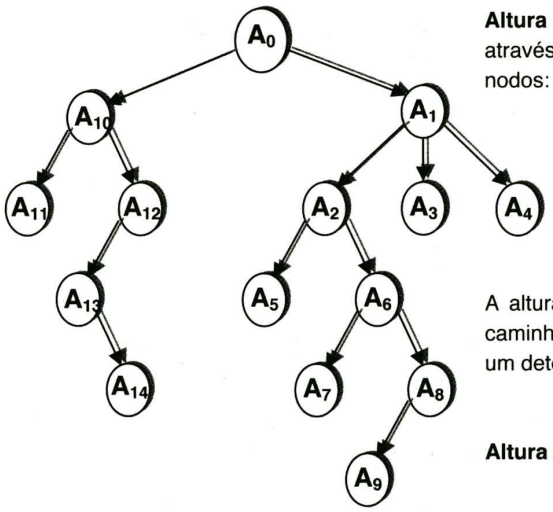
Árvore **m-ária completa** – cada nodo tem m filhos (número de filhos = m)

Se m=2 então temos uma **árvore binária**

10

10

Árvores



Altura de uma árvore **A** define-se através do maior nível dos seus nodos:

Se A está vazia, então a sua altura é igual a zero.

Se A não está vazia, então a sua altura é igual ao nível máximo dos seus nodos.

A altura de uma árvore representa o caminho mais longo do nodo raiz até um determinado nodo (folha).

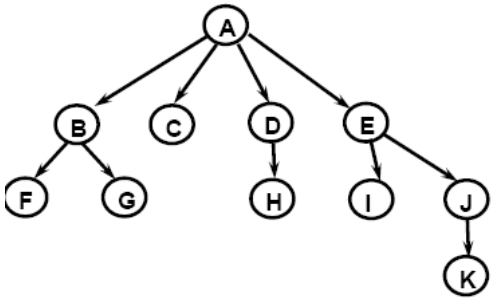
Altura = 6

11

11

Árvores

- Exercício: Qual a altura, profundidade e tamanho dos nós da árvore seguinte?



Nó	Altura	Profund.	Tam.
A	4	0	11
B	2	1	3
C	1	1	1
D	2	1	2
E	3	1	4
F	1	2	1
G	1	2	1
H	1	2	1
I	1	2	1
J	2	2	2
K	1	3	1

12

12

Árvores

Uma **árvore binária** é um tipo especial de árvore m-ária.

Cada nodo de uma árvore binária tem, no máximo, dois filhos.

Cada filho é designado por filho **direito** e filho **esquerdo**.

Cada nodo pode não ter filhos, sendo designado por **folha** ou terminal.

Uma **árvore binária** R é um conjunto finito de nodos com as seguintes características:

- Se R é um conjunto vazio, então a árvore está vazia.
- Tem um nodo **raiz (R)** associado.
- Os nodos filhos de R constituem um conjunto vazio ou estão segmentados em dois conjuntos disjuntos, designados por:
  - **subárvore esquerda** e
  - **subárvore direita**.

13

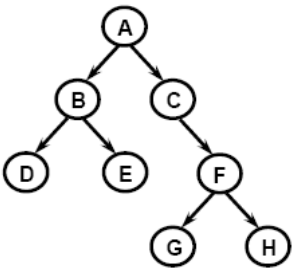
13

Árvores

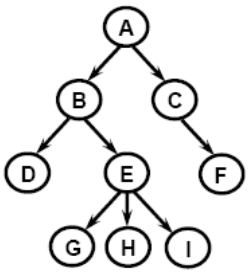
Árvore binária – definição recursiva

Uma árvore binária é um **conjunto finito de elementos** que, **ou está vazio**, ou contém um elemento chamado **raiz da árvore** ligado a dois conjuntos disjuntos em que **cada um é, por sua vez uma árvore binária.**

Árvore binária



Árvore não binária



14

14

Árvores

Árvore binária – definição recursiva

- A noção de subárvore é muito relevante para o tratamento de árvores binárias, nomeadamente para a aplicação do método recursivo de desenvolvimento de algoritmos.

Árvore binária completa

- Uma árvore binária cujos nós terminais se encontram todos no último nível, designa-se por árvore binária completa.
- Árvore binária completa de nível ***m*** é uma árvore em que:
  - cada nó de nível ***m*** é uma folha e em que
  - todos os nós de nível menor do que m têm subárvores direita e esquerda não vazias.

15

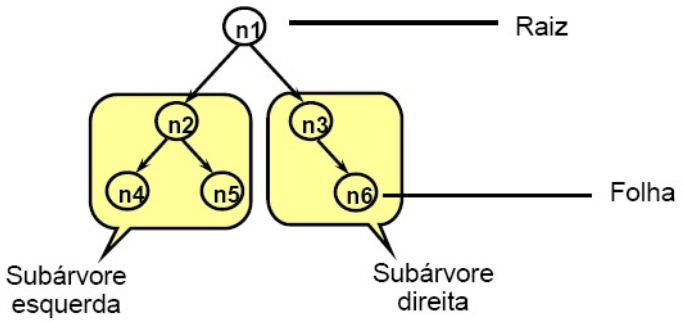
15

Árvores

Dois nós são irmãos se são os filhos direito e esquerdo do mesmo pai;

Um nó que não tem filhos é uma folha;

No exemplo seguinte, n4 e n5 são descendentes de n2 e este é ascendente de ambos.

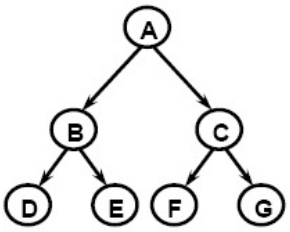


16

16

Árvores

Ao lado pode ver-se um exemplo de uma árvore binária completa de nível 3.



- Uma árvore binária completa com N nós internos tem N+1 nós externos ou folhas.

  No exemplo: 3 nós internos e 4 folhas

- Numa árvore binária completa o quantidade de nós nível i = 2^{i-1}.

  No exemplo:

  Nível 1 – $2^{1-1} = 2^0 = 1$ (A)

  Nível 2 – $2^{2-1} = 2^1 = 2$ (B e C)

  Nível 3 – $2^{3-1} = 2^2 = 4$ (D, E, F e G)

Pergunta: no nível 4 quantos elementos teria? E quantos nós internos?

17

17

Árvores

Travessia de uma árvore binária

- Em muitos problemas é necessário percorrer uma árvore binária, atravessando todos os seus nós e enumerando-os; Numa árvore não há uma ordem "natural" de percurso.
- Designa-se por visitar um nó a ação de atingir um dado nó de uma árvore e efetuar (ou não) uma qualquer operação com a informação nele armazenada;

Existem três métodos para atravessar uma árvore:

- pré-ordem, em-ordem, pós-ordem
- Utiliza-se uma definição recursiva, pelo que percorrer uma árvore implicará percorrer a raiz e percorrer as suas subárvores esquerda e direita.
- Outra definição: Se definirmos V como a operação de ler a raiz de uma árvore binária, L como a operação de percorrer a sua subárvore esquerda e R como a operação de percorrer a sua subárvore direita, os travessias podem também denominar-se: VLR (pré-ordem), LVR(em-ordem), LRV(pós-ordem)

18

18

Árvores

Em pré-ordem, ou pré-fixado ou VLR:

- Visitar a raiz
- Atravessar a subárvore esquerda em pré-ordem
- Atravessar a subárvore direita em pré-ordem.

Em em-ordem, ou central ou LVR:

- Atravessar a subárvore esquerda em ordem
- Visitar a raiz
- Atravessar a subárvore direita em ordem.

Em pós-ordem, ou pós-fixado ou LRV:

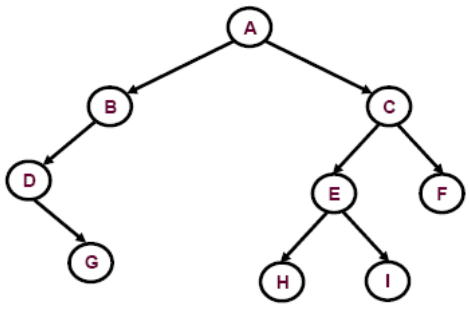
- Atravessar a subárvore esquerda em pós-ordem
- Atravessar a subárvore direita em pós-ordem
- Visitar a raiz.

19

19

Árvores

Exemplo 1: Travessias de uma árvore



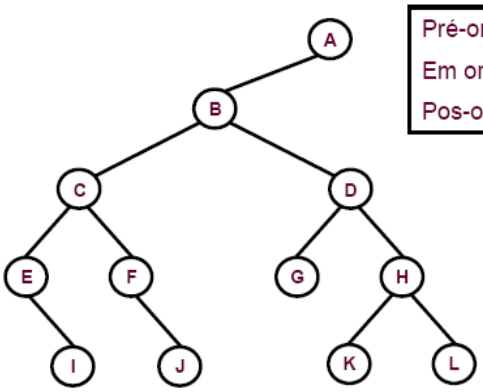
Pré-ordem: A B D G C E H I F
Em ordem: D G B A H E I C F
Pos-ordem: G D B H I E F C A

20

20

Árvores

Exemplo 2: Travessias de uma árvore



Pré-ordem: A B C E I F J D G H K L
Em ordem: E I C F J B G D K H L A
Pos-ordem: I E J F C G K L H D B A

21

21

Árvores

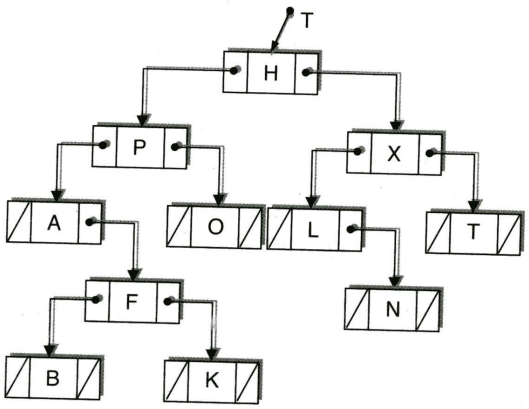
- Uma árvore binária (binary tree) é uma estrutura de dados mais geral que uma lista ligada.
- Cada elemento de uma árvore binária, célula, nodo, ou nó, consiste numa chave e dois apontadores:
  - A chave é a informação que se pretende armazenar na árvore.
  - Os dois apontadores servem apenas para dar estrutura à árvore: um aponta para o "filho esquerdo" da célula, outro para o "filho direito".

22

22

Árvores binárias

Implementação das uma árvore de caracteres



23

23

Árvores

Definição da estrutura de dados que representa a árvore.

O apontador para a raiz da árvore é local à função main().

Implementação da travessia LVR ou inOrder.

```
typedef struct tree{
        char info;
        struct tree *left;
        struct tree *rigth;
} Elemento;
…
void Main () {
Elemento *root=NULL;
…
}

Void inOrder (Elemento *root) {
        If (!root) return;
        inOrder (root -> left);
        If (root->info) printf("%c ", root->info);
        inOrder (root -> rigth);
}
```

24

24

Árvores

Implementação das travessias VLR ou preOrder e LRV ou posOrder

```
void preOrder (Elemento *root) {
        if (!root) return;
        If (root->info) printf("%c ", root->info);
        preOrder (root -> left);
        preOrder (root -> rigth);
}

void posOrder (Elemento *root) {
        if (!root) return;
        posOrder (root->left);
        posOrder (root->rigth);
        If (root->info) printf("%c ", root->info);
}
```

25

25

Árvores

Comparando com as listas ligadas

- As listas ligadas são estruturas lineares com as quais é difícil representar hierarquia entre os seus elementos.
- Para aplicações de recuperação de informação, as implementações recorrendo a listas ligadas têm o problema de não permitirem o acesso aleatório à informação armazenada, pois é sempre necessária uma pesquisa sequencial.
- Por outro lado, a implementação contígua de uma lista é muito ineficiente quando se pretendem fazer alterações frequentes.
- As árvores binárias são uma solução muito boa para aplicações em que é necessário combinar estes dois tipos de operações.

26

26

Árvores

As **árvores binárias de pesquisa** definem-se como árvores binárias nas quais todos os nós satisfazem as seguintes condições:

- A entrada da raiz da subárvore esquerda, se existir, é menor que a entrada da raiz da árvore.
- A entrada da raiz da subárvore direita, se existir, é maior que a entrada da raiz da árvore.
- As subárvores são, por sua vez, árvores binárias de pesquisa.
- Não pode haver entradas duplicadas.

As operações de inserção e remoção têm de ser implementadas de forma a respeitar as restrições impostas pela definição do tipo de dados.

27

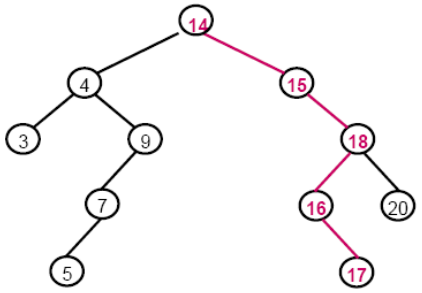
27

Árvores

Árvores binárias de pesquisa

Numa árvore que não é de pesquisa, a pesquisa de um qualquer elemento teria que passar por todos os nós da árvore (tal como no caso das listas ligadas).

Para resolver esta situação, a árvore binária de pesquisa impõem que, para qualquer nó da árvore, todos os elementos da sua subárvore esquerda sejam menores que ele e que todos os elemento da sua subárvore direita sejam maiores que ele.



Por exemplo: para pesquisar o elemento 17 na árvore da figura seria apenas necessário percorrer os elementos assinalados (de cor rosa).

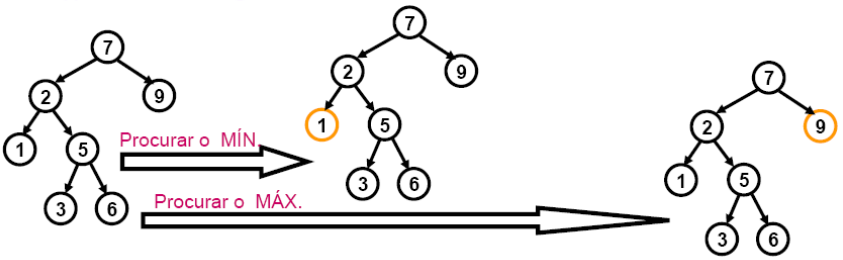
28

28

Árvores

Árvores binárias de pesquisa

- Procurar o mínimo e o máximo é muito eficiente.



29

29

Árvores

Remoção e um registo numa árvore binária de pesquisa.

Esta operação é complicada devido às restrições definidas para as relações entre os elementos armazenados.

É necessário considerar três casos:

- remoção de uma folha,
- remoção de um nó com apenas uma subárvore
- remoção de um nó com duas subárvores.

Para a resolução do último caso há várias hipóteses, a mais simples das quais pendurar uma das subárvores debaixo da outra. Os problemas resultantes para a estrutura da árvore são evidentes.

Uma solução melhor será substituir o nó a remover pelo seu antecessor (ou sucessor) direto.

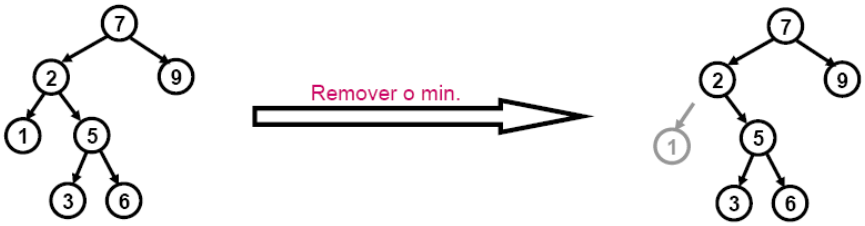
30

30

Árvores

Árvores binárias de pesquisa

- Remover o o valor mais baixo (mínimo).



31

31

Árvores

Árvores binárias de pesquisa

- Remover uma folha.

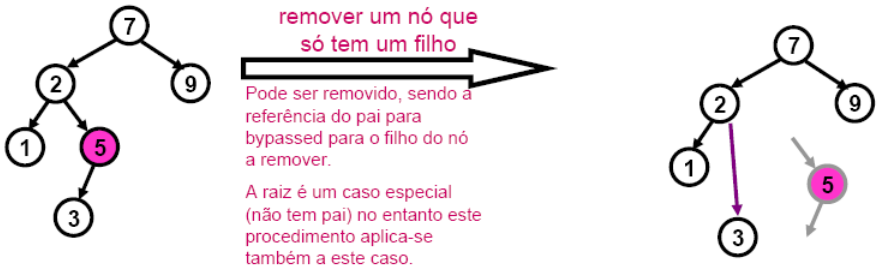
32

32

Árvores

Árvores binárias de pesquisa

- Remover um nó com apenas um filho.

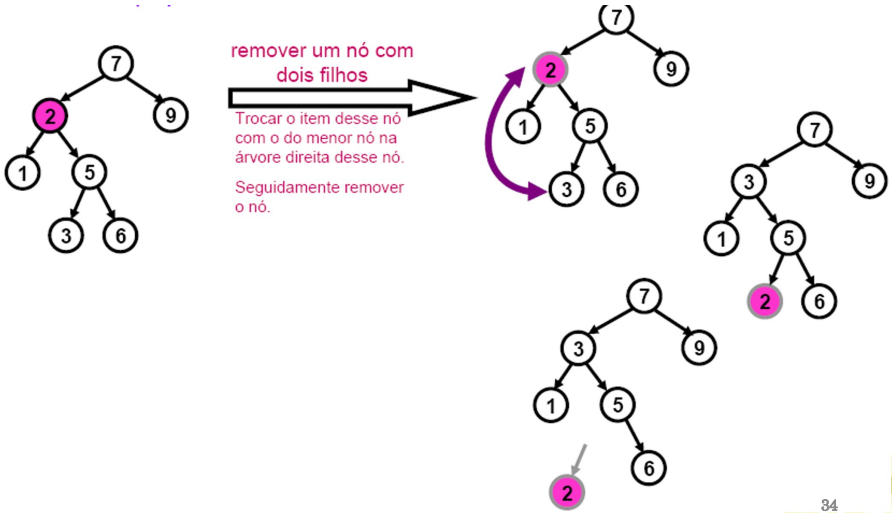


33

33

Árvores

Árvores binárias de pesquisa: remover um nó com dois filhos.



34

34

Árvores

Inserção em árvores binárias de pesquisa:

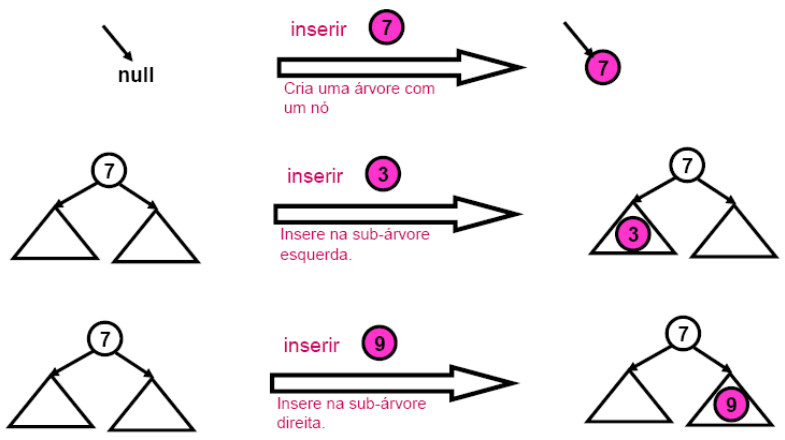
- A ordem pela qual os elementos são inseridos determina a forma da árvore.
- Em relação à operação de pesquisa, o pior caso ocorrerá se os elementos forem inseridos na árvore pela sua ordem natural.
- O algoritmo terá de respeitar as restrições estruturais deste tipo de árvore:
  - Um elemento inferior ao elemento do nó atual terá de ser armazenado na subárvore esquerda.
  - Um elemento superior ao elemento do nó atual terá de ser armazenado na subárvore direita.
- A performance do algoritmo de inserção é equivalente à da pesquisa do elemento a inserir.

35

35

Árvores

Árvores binárias de pesquisa: inserção



36

36

Árvores

Árvore binária de pesquisa:

- Definição da estrutura de dados que representa a árvore binária.
- Definição de um apontador para a raiz da árvore, local à função main().

```
typedef struct {
        int numero;
        char nome[50];
} INFO;
typedef struct tree {
        INFO node;
        struct tree *left;
        struct tree *rigth ;
} Elemento;
…
void main(){
    ...
    Elemento *raiz=NULL;
    …
}
```

37

37

Árvores

Implementação da função que verifica se o elemento com numero (parâmetro num da função) existe na árvore binária de pesquisa.

```
int verificaExiste(Elemento *r, int num) {
      if (!r) {return 0;}
      if (r->node.numero == num){
            printf ("nome = %s", r->node.nome);
            return 1;
      }
      if (r->node.numero < num) {
            verificaExiste(r->rigth, num);
      }
      else {
            verificaExiste(r->left, num);
    }
}
```

38

38

Árvores

Implementação da função que permite a alteração dos dados de um elemento da árvore (dado o numero).

```
int alteraNome(Elemento *r, int num, char novo_nome[]){
      if (!r) return -1;
      if (r->node.numero == num){
            strcpy(r->node.nome, novo_nome);
            return 1;
      }
      if (r->node.numero < num) {
            alteraNome(r->rigth, num, novo_nome);
      }
      else {
            alteraNome(r->left, num, novo_nome);
      }
}
```

39

39

Árvores

Implementação da função que insere novo elemento na árvore binária de pesquisa.

```
void insere (Elemento **r, INFO newElem) {
    Elemento *aux=*r;
    if (*r==NULL){
       *r=cria(newElem);
       return; // 1° elemento da àrvore
    }
    if (aux->node.numero < newElem.numero)
       if (aux->rigth == NULL) aux->rigth=cria(newElem);
       else insere(&aux->rigth, newElem);
    else
       if (aux->left == NULL) aux->left=cria(newElem);
       else insere(&aux->left, newElem);
}
```

40

40

Árvores

Implementação da função **cria()**, invocada na função anterior, que reserva espaço de memória para o novo elemento na árvore.

```
Elemento *cria (INFO newElem) {
      Elemento *novo=NULL;
      novo=(Elemento *) calloc(1, sizeof(Elemento));
      novo->node=newElem;
      novo->left=NULL;
      novo->rigth=NULL;
      return novo;
}
```

41

41

Árvores

Implementação das funções menor() e maior() que retornam, respetivamente, o menor e o maior numero da árvore binária de pesquisa.

```
int menor(Elemento *r) {
    if (!r) return -1;
    if (r->left == NULL) return r->node.numero;
    return (menor(r->left));
}

int maior(Elemento *r) {
    if (!r) return -1;
    if (r->rigth == NULL) return r->node.numero;
    return (maior(r->rigth));
}
```

42

42

Árvores

Implementação da função limpa() que liberta o espaço de memória ocupado pelos elementos da árvore.

```
void limpa(Elemento **r) {
    Elemento *aux=*r;
    if (!*r) return;
    if (aux->left!=NULL) limpa(&aux->left);
    if (aux->rigth!=NULL) limpa(&aux->rigth);
    aux->left=NULL;
    aux->rigth=NULL;
    free(aux);
}
```

43

43

Árvores

Implementação da função size() que, recursivamente conta o numero de elementos da árvore.

```
int size(Elemento *r) {
    if (!r) return 0;
    return (1 + size(r->left) + size(r->rigth));
}
```

44

44

Árvores

Árvore binária balanceada (AVL):

- Uma árvore diz-se balanceada (ou AVL) se, e só se, para cada um dos seus nós as alturas das subárvores da esquerda e da direita forem iguais ou diferirem apenas numa unidade.

ou

- Uma árvore balanceada ou AVL é uma árvore binária de pesquisa em que as subárvores esquerda e direita da raiz não diferem de mais do que uma unidade nas suas alturas.
- Por sua vez, cada uma destas subárvores é também uma árvore AVL.

45

45

Árvores

Árvore balanceada:

- As árvores AVL (o nome resulta das iniciais dos seus inventores) são casos particulares de árvores de pesquisa binária em que as operações de inserção e remoção são desenhadas para manter a árvore muito próxima de um estado balanceado em cada instante.
- Numa árvore perfeitamente balanceada, as subárvores de cada nó têm a mesma altura.
- Na prática, isto pode não ser possível mas, construindo a árvore cuidadosamente, é possível garantir que as subárvores de cada nó não diferem de mais de uma unidade nas suas alturas.
- As árvores B e B+ são usadas por alguns SGBD (Sistema Gestor de Base de Dado).

46

46

Bibliografia

- Programação Avançada Usando C, António Manuel Adrego da Rocha, ISBN: 978-978-722-546-0.
- Schildt, Herbert: C the complete Reference, McGraw-Hill, 1998.
- Algoritmia e Estruturas de Dados, José Braga de Vasconcelos, João Vidal de Carvalho, ISBN: 989-615-012-5.
- Elementos de Programação com C - Pedro João Valente D. Guerreiro, 3ª edição, ISBN: 972-722-510-1.
- Introdução à Programação Usando C, António Manuel Adrego da Rocha, ISBN: 972-722-524-1.
- https://www.youtube.com/watch?v=ikPPdBDZnz4 – to see

47