1. **ÁRVORES - CONCEITOS BÁSICOS**
   1. **Buscando Elementos**

Imagine que queremos buscar um elemento (15) no array abaixo. Como fazer ?

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 8 | 12 | 15 | 20 | 23 | 30 |

E o elemento 17? Será que não poderíamos aproveitar o fato de que o arranjo está ordenado?

Buscando o elemento 17?

Mas, se o número buscado for maior que todos (e.g. 35), neste caso, correremos o arranjo todo. Teria como melhorar?

* 1. **Busca Binária**... É mais eficiente, mas depende de arranjos estáticos.

E se tivéssemos uma lista ligada?

2

8

12

30

23

20

15

Teríamos um problema, pois não saberíamos o elemento do meio. Teríamos que fazer um arranjo de ponteiros para esta lista. Será que não poderíamos ter uma estrutura dinâmica que nos ajude nesta tarefa?

Se aplicássemos uma busca binária na lista ligada?

20

23

30

12

8

2

15

Eis nossa estrutura. Que nome, daremos a ela? Uma árvore.

15

2

8

12

30

23

20

* 1. **Árvore**

Em computação costumamos representar a árvore de forma invertida.

Nível 0 (20=1)

raiz

15

2

8

12

30

23

20

Nível 1 (21=2)

Nível (22= 4)

folhas ...

folhas ...

Subárvore 2

Subárvore 1

* + 1. **Subárvores**

Um conjunto de nós consistindo de um nó chamado raiz, abaixo do qual estão as subárvores que compõem essa árvore.

O número de subárvores de cada nó é chamado de grau desse nó. No exemplo ao lado, todo nó tem grau 02, exceto os de base, que tem grau 0.

* + 1. **Grau; Internos/Externos**

Nós de grau ZERO são chamados de nós externos ou folhas. Os demais são chamados de nós internos.

* + 1. **Descendentes**

Nós, abaixo de um determinado nó, são seus descendentes.

Descendentes de 8: 2 e 12

Descendentes de 15: todos os nós.

Folha não tem descendente.

* + 1. **Nível**

O nível do nó raiz é ZERO. A cada grupo de subárvores, nova geração, teremos novos níveis. Quantidade de nós da árvore será 2n – 1.

* + 1. **Altura**

A altura de uma árvore é a altura da raiz.

A altura de um nó será à distância desse nó, do maior caminho, até uma folha.

Vale notar que a árvore é percorrida sempre da raiz às folhas.

A altura (h) de um nó raiz é o cumprimento do caminho mais longo entre ele e uma folha.

Nem sempre a árvore estará balanceada. Ainda assim, as definições de altura, nível etc valem.

A altura de uma árvore de uma raiz. Da mesma forma, que o endereço de uma árvore na memória será o endereço de seu nó raiz.

* + 1. **Profundidade**

A profundidade de um nó é a distância percorrida da raiz a esse nó.

Profundidade de 15 (raiz): ZERO;

Profundidade de 8: 1;

Profundidade de 12: 2.

* + 1. **Definições**
* Raiz => Não possui ancestral, só pode ter filhos.
* Folha => Não tem filhos (ou melhor, seus filhos são estruturas vazias)
* Nó => Elemento que contém a informação
* Arco ou aresta => Liga dois nós
* Pai => Nó superior de uma aresta
* Raiz => Nó topo – não possui pai
* Folhas => Nós das extremidades inferiores – não tem nós filhos
* Grau => Representa o número de subárvores de um nó.
* Grau de uma árvore (aridade) : É definido como sendo igual ao máximo dos graus de todos os seus nós.
* Nível de um nó => É a distância da raiz da árvore. A raiz tem nível ZERO.
* Altura ou profundidade => É o nível do nó folha que tem o mais longo caminho até a raiz.
* A árvore vazia é uma árvore de altura -1, por definição.
* Uma árvore com um único nó tem altura 1 (o nó é raiz e folha ao mesmo tempo).
* Toda árvore com n > 1 nós possui no mínimo 1 e no máximo n-1 folhas.
* Árvore estritamente binária, cada nó possui 0 ou 2 filhos.
* Árvore binária completa. Se v é um nó tal que alguma subárvore de v é vazia, então v se localiza ou no último (maior) ou no penúltimo nível da árvore.
* Árvore binária cheia. Se v é um nó tal que alguma subárvore de v é vazia, então v se localiza no último (maior) nível da árvore, v é um nó folha.

1. **ÁRVORES BINÁRIAS**

Uma árvore binária é uma árvore em que, abaixo de cada nó, existem no máximo duas subárvores. Trata-se de uma árvore binária de pesquisa, árvore binária de busca, árvore de busca binária.

Uma árvore binária de pesquisa – ABP é uma árvore binária em que, a cada nó, todos os registros com chaves menores que a deste nó estão na subárvore da esquerda, enquanto os registros com chaves maiores estão na subárvore da direita.

Como representamos computacionalmente uma árvore binária? Unindo nós. E como representamos os nós? Com 02 ponteiros: um ponteiro para a subárvore da esquerda e um ponteiro para a subárvore da direita. Além de um campo para a chave e dados.

|  |  |
| --- | --- |
| **chave** | |
| null | null |

Para nossa ABP, utilizaremos as seguintes funções:

* Inicialização da árvore;
* Inserção de um elemento
* Busca de um elemento
* Contagem do número de elementos
* Impressão dos elementos
* Remoção de Elementos
  1. Definindo a estrutura de dados não linear – árvore binária em linguagem C

#include <stdio.h>

#include <stdlib.h>

#include (malloc.h>

#define false 0

#define true 1

typedef int bool;

typedef int PRIMARY\_KEY;

typedef struct pontNo {

PRIMARY\_KEY chave;

struct pontNo \*sae;

struct pontNo \*sad;

} NO;

typedef NO\* arvore;

arvore Inicializa();

adicionaElemento(arvore raiz, arvore no);

criaNovoNo(PRIMARY\_KEY ch);

* 1. Inicializando a Árvore

Para representarmos nossa árvore, precisaremos tão somente do endereço do nó raiz. E para inicializarmos a árvore basta tonarmos esse endereço NULL.

arvore Inicializa() {

return NULL;

}

Int main(void){

PRIMARY\_KEY ch;

// Inicializa a Raiz

arvore r = Inicializa();

// Criando Novo Nó [Elemento]

printf(“Digite o elemento:”); scanf(“%d”, &ch);

arvore no = criaNovoNo(ch); // exemplo 23, 12, 15, 25, 8

// Inserindo Elemento na Raiz

r = adicionaElemento(r, no);

}

Nota: Se a inserção fosse: 25, 23, 15, 12 e 8 (forma um galho, parece uma lista ligada .) A ordem define a forma da árvore.

* 1. Inserção de Elemento

Nesta árvore não será permitido a duplicação de chaves. Chaves menores à de um determinado nó ficam na subárvore da esquerda deste. E chaves maiores de um determinado nó ficam na subárvore da direita deste.

10 < 15?

Inserção 10.

10 < 8?

15

2

8

12

30

23

20

10 < 12?

10

Se ( a raiz for null ) então

Inserimos na raiz

Senão Se ( a chave do elemento a ser inserido for menor que a da raiz) então

Insere elemento na subárvore da esquerda

Senão

Insere elemento na subárvore da direita

Fim Se

Fim Se

arvore adicionaElemento(arvore raiz, arvore no){

if ( arvore == NULL ) return(no);

if ( no->chave < raiz->chave )

raiz->sae = adiciona(raiz->sae, no);

else raiz->sad = adiciona(raiz->sad, no);

return(raiz);

}

arvore criaNovoNo(PRIMARY\_KEY ch){

arvore\* novo = (arvore\*) malloc sizeof(arvore);

novo->chave=ch;

novo->sae=NULL;

novo->sad= NULL;

return arvore;

}

<https://www.cs.usfca.edu/~galles/visualization/Algorithms.html>

* 1. Busca de Elemento

arvore buscaArvore(PRIMARY\_KEY ch, arvore raiz){

if ( arvore == NULL ) return NULL;

if ( raiz->chave == ch ) return (raiz);

if ( raiz->chave > ch ) return (buscaArvore(ch, raiz->sae);

else return (buscaArvore(ch, raiz->sad);

}

* + 1. Percurso em profundidade em árvores binárias:
* Pré-Ordem ou Pré-Fixado (VLR)

raiz->sae->sad

* Central ou in-Ordem (LVR)

sae->raiz->sad

* Pós-Ordem ou Pós\_Fixado (LRV)

sae->sad->raiz