Árvore Binária de Pesquisa

Ciência da Computação
Laboratório de Ordenação e Pesquisa
Prof. M.Sc. Elias Gonçalves

Árvore Binária de Pesquisa

Objetivos:

→ Encontrar item cuja chave seja igual a do elemento dado na pesquisa.

→ Recuperar item com uma determinada chave.

→ Recuperar dados armazenados em uma base de dados.

Árvore Binária de Pesquisa

Questões envolvidas na escolha do método de pesquisa:

→ Qual a quantidade de dados no conjunto?

→ Qual é a frequência com que operações de inserção e retirada de dados são realizadas?

→ Os dados estão estruturados?

→ Os dados estão ordenados?

→ Há valores repetidos?

Pesquisa em Memória Primária

Métodos de pesquisa:

→ Pesquisa sequencial;

→ Pesquisa binária;

→ Árvore de pesquisa;

→ Pesquisa digital;

Árvore (binária) de Pesquisa/busca

```c
typedef struct no {
    char info;
    int chave;
    struct no *esq;
    struct no *dir;
} NO;
```

→ Mais eficiência nas buscas;

→ As chaves são únicas;

→ A chave define a ordem dos nós;

→ Exemplo de nó:

Quantas vezes a comparação é executada?

→ A complexidade da busca é igual ao número de chamada da função.

→ Em caso da árvore ser completa:

O(log(n)) vezes

Árvore (binária) de Pesquisa/busca

- **Condições:**

Uma árvore binária T é uma árvore binária de busca se:

- Chaves da subárvore esquerda de T forem menores que a chave da raiz de T.
- Chaves da subárvore direita de T forem maiores que a chave da raiz de T.
- Subárvores da esquerda e da direita de T são árvores binárias de busca.

Árvore (binária) de Pesquisa/busca

- **Operações:**
  - Buscar nó com um chave específica;
  - Inserir novo nó;
  - Remover nó.

Deve respeitar a ordem das operações!!!

Árvore (binária) de Pesquisa/busca

- **Buscar um elemento x:**
  - x é igual a chave?
  - x é maior que a chave?
  - x é menor que a chave?

Em qualquer nó

Árvore (binária) de Pesquisa/busca

```
NO *busca ( NO *no, int chave ) {
    if ( no == NULL ) {
        printf( "Nó não encontrado na árvore.\n" );
        return NULL;
    }
    else if ( no->chave == chave ){
        printf( "Nó encontrado na árvore.\n" );
        return no;
    }
    else if ( no->chave > chave )
        return busca( no->esq, chave );
    else if ( no->chave < chave )
        return busca ( no->dir, chave );
}
```

Árvore (binária) de Pesquisa/busca

- **Inserir**
  - Árvore vazia, insere na raiz;
  - Árvore não vazia, compara a chave com a chave da raiz:
    - chave menor que a chave da raiz, insere na subárvore esquerda.
    - Chave maior que a chave da raiz, insere na subárvore da direita.

A inserção ocorre nas folhas!

Árvore (binária) de Pesquisa/busca

```
NO *insere ( NO *raiz, int chave ) {
    if ( raiz == NULL ){
        raiz = ( NO* ) malloc( sizeof(NO) );
        if ( raiz == NULL ) {
            printf( "Erro ao criar nó.\n" );
            exit( 1 );
        }
        raiz->chave = chave;
        raiz->esq = NULL;
        raiz->dir = NULL;
        printf( "Nó criado com sucesso!\n" );
    }
    else if ( chave < ( raiz->chave ) ) {
        printf( "Inserindo na esquerda da raiz!\n" );
        raiz->esq = insere ( raiz->esq, chave );
    }
    else if ( chave > ( raiz->chave ) ) {
        printf( "Inserindo na direita da raiz!\n" );
        raiz->dir = insere ( raiz->dir, chave );
    }
    else {
        printf ( "Esta chave já existe!\n" );
        exit(1);
    }
    return raiz;
}
```

Árvore (binária) de Pesquisa/busca

- **Problema da altura**
- A ordem em que as chaves são inseridas pode fazer com que a árvore fique com a altura muito grande.
- Ex (insira na ordem): 40 60 55 50 45

Sem balanceamento!

Árvore (binária) de Pesquisa/busca

- **Criar árvore balanceada**
  - Seja um vetor v ordenado com as chaves a serem inseridas na árvore.
  - A chave a ser inserida deve ser a do meio do vetor.
  - Chamar a função de inserção recursivamente para a primeira e a segunda parte do vetor (esquerda e direita).

Árvore (binária) de Pesquisa/busca

```
void criaArvoreBalanceada( NO *raiz, int v[], int inicio, int fim  ) {
    if( inicio<= fim ){
        int meio = ( inicio + fim ) / 2;
        raiz = insere ( raiz, v[meio] );
        criaArvoreBalanceada( raiz, v, inicio, meio-1 );
        criaArvoreBalanceada( raiz, v, meio+1, fim );
    }
}
```

Árvore (binária) de Pesquisa/busca

- **Excluir**
  - Há três casos a serem considerados na exclusão:

1. O nó é folha;
2. O nó não é folha e tem uma subárvore;
3. O nó não é folha e tem duas subárvores.

Árvore (binária) de Pesquisa/busca

- **Excluir**

1. O nó é folha → é preciso remover o nó desalocando sua memória reservada.
2. O nó tem uma subárvore → a raiz da subárvore precisa ocupar o lugar do nó que será excluído.
3. O nó possui duas subárvores → a árvore precisa ser reestruturada.

Árvore (binária) de Pesquisa/busca

- **Excluir –** Estratégias para o caso 3:

1. O nó possui duas subárvores → a árvore precisa ser reestruturada.

   → Trocar o valor do nó a ser removido com o valor do nó que tem a maior chave da sua subárvore esquerda.

Árvore (binária) de Pesquisa/busca

- **Excluir**

```
NO *menorNO( NO *raiz ){
    if(raiz == NULL)
        return NULL;

    else if( raiz->esq != NULL )
        return menorNO( raiz->esq );

    return raiz;
}
```

```
NO *exclui( NO *raiz, int chave ){
    if( raiz == NULL )
        return NULL;

    if ( chave > raiz->chave )
        raiz->dir = exclui( raiz->dir, chave );

    else if( chave < raiz->chave )
        raiz->esq = exclui( raiz->esq, chave );
```

Árvore (binária) de Pesquisa/busca

```
    else{
        // O nó é folha: é preciso remover o nó desalocando sua memória reservada.
        if( raiz->esq == NULL && raiz->dir == NULL ){
            free( raiz );
            return NULL;
        }

        // O nó tem uma subárvore: a raiz da subárvore precisa ocupar o lugar do nó que será excluído.
        else if( raiz->esq == NULL || raiz->dir == NULL ){
            NO *aux;
            if( raiz->esq == NULL )
                aux = raiz->dir;
            else
                aux = raiz->esq;

            free(raiz);
            return aux;
        }

        /*
        * O nó possui duas subárvores: a árvore precisa ser reestruturada.
        * Trocar o valor do nó a ser removido com o valor do nó que tem a menor chave da sua subárvore direita.
        * Remover o nó na subárvore onde fez-se a troca.
        */
        else{
            NO *aux = menorNO( raiz->dir );
            raiz->chave = aux->chave;
            raiz->dir = exclui( raiz->dir, aux->chave );
        }
    }
    return raiz;
}
```