UNIVERSIDADE DE ITAÚNA

DAVI VENTURA CARDOSO PERDIGÃO
EDMILSON LINO CORDEIRO
ERIC HENRIQUE DE CASTRO CHAVES

APLICAÇÃO DA ÁRVORE B (B-TREE)

Questão 2 - Prova Final (Projeto e Análise de Algoritmo)

ITAÚNA
2022

SUMÁRIO

1. INTRODUÇÃO

"Método genérico para o armazenamento e a recuperação de dados, voltado para arquivos volumosos, pois proporciona rápido acesso aos dados e possui custo mínimo de overhead". Foi essa a explicação sobre **Árvore B** proposta pela **Boing Corporation**,em **1972**.

Em ciência da computação, uma árvore B é uma estrutura de dados auto-balanceada, que armazena dados classificados e permite pesquisas, acesso sequencial, inserções e remoções. Essa estrutura foi projetada para funcionar bem em discos magnéticos ou outros dispositivos de armazenamento secundário. A árvore B é uma generalização de uma árvore de pesquisa binária em que um nó pode ter mais que dois filhos.

Diferente das árvores de pesquisa binária auto-balanceadas, a árvore B é bem adaptada para sistemas de armazenamento que leem e escrevem blocos de dados relativamente grandes, como discos. Muitos SGBD (Sistema de Gerenciamento de Banco de Dados) usam árvores B ou variações derivadas para armazenar informações.

2. CARACTERÍSTICAS DA ÁRVORE B

- Árvores B são a estrutura subjacente a muitos sistemas de arquivos e bancos de dados. Por exemplo,
    - sistema de arquivos NTFS do Windows;
    - sistema de arquivos HFS do Mac;
    - sistemas de arquivos ReiserFS, XFS, Ext3FS, JFS do Linux;
    - bancos de dados ORACLE, DB2, INGRES, SQL e PostgreSQL.
- Uso do Bottom-up para a criação (em disco). *Nós Folhas → Nó Raíz*.
- Normalmente um nó da árvore B é tão grande quanto uma página de disco inteira.
- Índice extremamente volumoso.
- Buffer-pool pequeno. Apenas uma parcela do índice pode ser carregada em memória principal e suas operações são baseadas em disco.
- Possui uma sequência ordenada de chaves (conjunto de ponteiros). Número de ponteiros = número de chaves + 1 (exemplo: árvore B de ordem 8, máximo de 7 chaves e 8 ponteiros). A raiz contém no mínimo 2 chaves e cada uma das demais nós contém no mínimo M/2 chaves.
- Cada página, exceto a raiz e as folhas, têm no mínimo m/2 descendentes.
- Desempenho proporcional a log k^I (I = tamanho do índice; K = tamanho da página de disco) ou melhor. O número de acessos ao disco exigidos para a maioria das operações em uma árvore B é proporcional a sua altura.

- Todas as folhas estão no mesmo nível, isso faz com que ela seja uma árvore completamente balanceada.
- Formalização da terminologia específica, precisamente as propriedades que devem estar presentes para uma estrutura de dados ser qualificada como árvore-B. Direciona a implementação do algoritmo de remoção da árvore-B.
- Problema: a literatura não é uniforme no uso e definição dos termos.

3. O PROBLEMA

Sabemos que, hoje em dia, o home office é uma grande realidade no mundo todo. Utilizando como exemplo uma grande empresa, segundo a **RAIS** (Relação Anual de Informações Sociais) do Ministério do Trabalho e Previdência, dos **70.000** colaboradores da **Atento**, cerca de **35% (24.500)** estão em home office e o restante se divide entre os modelos híbrido e presencial. Uma grande **vantagem** do modelo home office tanto para colaboradores quanto para empresas é a possibilidade de fazer escalas mais flexíveis de trabalho, sendo possível organizar melhor o tempo do funcionário e as suas atividades. Porém, nesse mesmo benefício se encontra um grande **desafio** para as empresas empregadoras: controlar as horas trabalhadas por cada funcionário.

O modelo mais comum utilizado para controle de ponto conta com quatro registros diários: início da jornada, saída para almoço, retorno do almoço e final da jornada. Continuando com o exemplo da Atento, contabilizando apenas os funcionários em **home office**, existiriam cerca de **98.000** registros diários, **490.000** semanais e **1.960.000** mensais. Supondo que o gestor de RH dessa empresa queira consultar as horas trabalhadas de um funcionário em uma data específica, há de se imaginar que não seja simples o processo para obter-se uma **busca**, tanto pela quantidade de registros, tanto pela quantidade de dados que um funcionário contém (matrícula, Horas Normais, Horas Faltosas, Horas Extras, Função, etc.).

Quando todos os objetos são armazenados em um mesmo database (Ex: Firebird), ou vários databases num único device (Ex: SQL Server) o grau de complexidade do formato interno de arquivo **aumenta** muito. A maioria das informações acabam misturadas dentro do banco de dados e só podem ser distinguidas uma das outras através das informações lógicas armazenadas em tabelas de sistemas. Nestes bancos de dados, as tabelas de sistemas que armazenam dados lógicos sobre todos os objetos do banco de dados são uma parte **vital**, pois se elas forem perdidas ou corrompidas, todo o database não passará apenas de **lixo**, pois não haverá como distinguir, por exemplo, dados de funcionário e seus registros de ponto.

Já em bases com índices de **árvores B**, as informações são dispostas de forma diferente, cada tipo de informação é pesquisada aprofundando-se em **nós** que possuem alguma semelhança com o **item** a ser **pesquisado**, como por exemplo, a matrícula de um funcionário. De fato, pode-se **aprofundar** em nós até um ponto máximo e será nesse ponto máximo que estarão os limites para efetuar a **pesquisa**. Sendo assim, não há necessidade de percorrer um índice inteiro, apenas uma parte do índice. Trabalhando com flat-tables com índices comuns somos obrigados a separar informações semelhantes em tabelas diferentes apenas para não acumular dados em excesso numa única tabela e assim prejudicar a performance do sistema. Usando-se bancos de dados SQL com índices de árvores B não existe divisão, haverão **nós de buscas**, então não fará muita diferença separar dados em tabelas diferentes se ambas estiverem no mesmo banco de dados. Isso apenas aumentará o grau de complexidade na hora de aplicar os JOIN's (junções).

4. O ALGORÍTMO E SUA COMPLEXIDADE

Em um banco de dados com gigabytes de informações usando árvore B, uma pesquisa será resolvida da seguinte forma : serão percorridos os nós dentro do índice e cada nó alcançado separará fisicamente dados que não estejam dentro de uma seletividade, até atingir uma profundidade máxima, será nessa profundidade máxima que estabelecerá os limites para a pesquisa. A palavra seletividade é apenas o termo que usamos quando algum nó possui algum nexo, isto é, semelhança ou aproximação com o que estamos procurando. A seguir segue um exemplo de funções básicas do algoritmo de árvore B, ele apresenta complexidade O(t logt n) em todas as suas operações básicas:

```
void inserir(int chave) {
    struct node *novono;
    int novachave;
    enum chaveStatus valor;
    valor = ins(raiz, chave, &novachave, &novoNo);
    if (valor == Duplicado)
        printf("chave ja disponivel\n");
    if (valor == InserirIt) {
        struct node *novaRaiz = raiz;
        raiz = (struct node *) malloc(sizeof(struct node));
        raiz->n = 1;
        raiz->chaves[0] = novaChave;
        raiz->p[0] = novaRaiz;
        raiz->p[1] = novoNo;
    }
}
```

```
void deletarNo(int chave) {
    struct node *novaRaiz;
    enum KeyStatus valor;
    valor = del(raiz, chave);
    switch (valor) {
        case PesquisaDeErro:
            printf("chave %d não está disponível\n", chave);
            break;
        case MenoresChaves:
            novaRaiz = raiz;
            raiz = raiz->p[0];
            free(novaRaiz);
            break;
    }
}


void Destruir() {
    while (raiz != NULL) {
        DelNode(raiz->chaves[0]);
    }
}


void buscar(int chave) {
    int pos, i, n;
    struct no *ptr = raiz;
    printf("Buscar caminho:\n");
    while (ptr) {
        n = ptr->n;
        for (i = 0; i < ptr->n; i++)
            printf(" %d", ptr->chaves[i]);
        printf("\n");
        pos = searchPos(chave, ptr->chave, n);
        if (pos < n && chave == ptr->chaves[pos]) {
            printf("Chave %d encontrada na posição %d do nó\n", chave,
i);
            return;
        }
        ptr = ptr->p[pos];
    }
    printf("Chave %d não foi encontrada\n", chave);
}
```

Exemplo da função ***inserir(int chave)*** que se refere à matrícula do funcionário que desejamos buscar as informações do registro de ponto: 51, 3477, 1320, 1755, 5600, 8375, 13899, 666, 4742, 11170.

➢ **Inserindo o número: 51**

➢ **Inserindo o número: 3477**

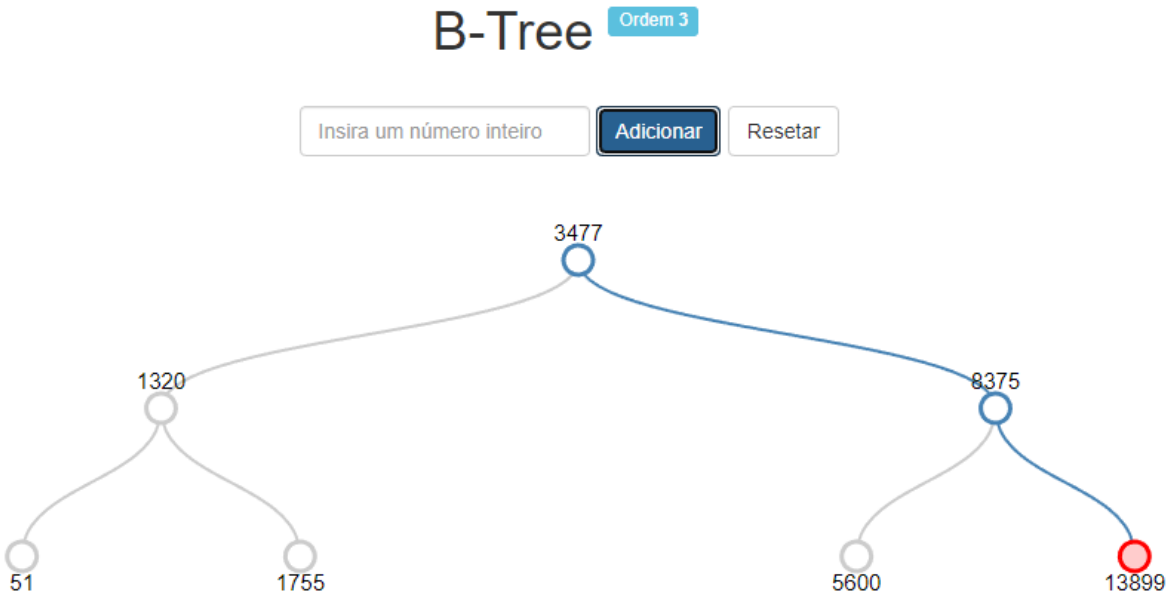
➢ **Inserindo o número: 1320**

➢ **Inserindo o número: 1755**

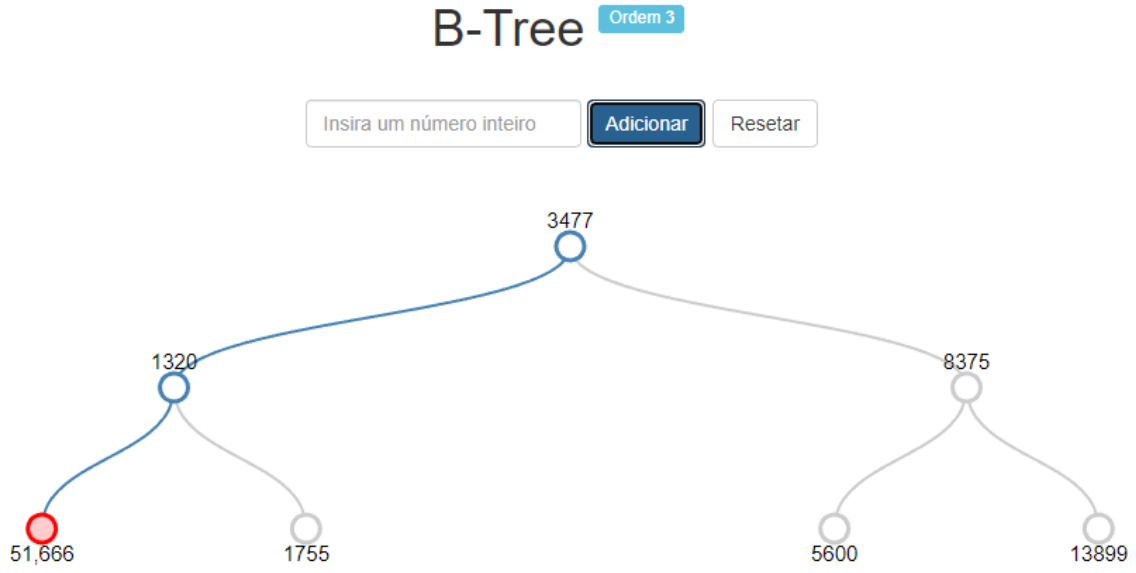
➢ **Inserindo o número: 5600**

➢ **Inserindo o número: 8375**

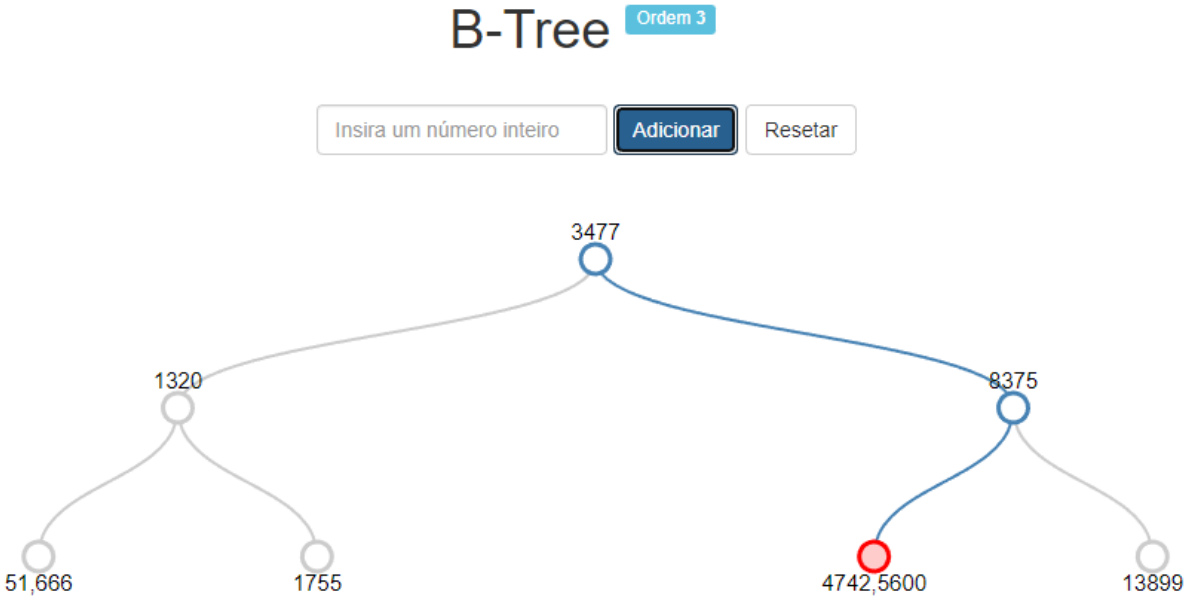
➢ **Inserindo o número: 13899**



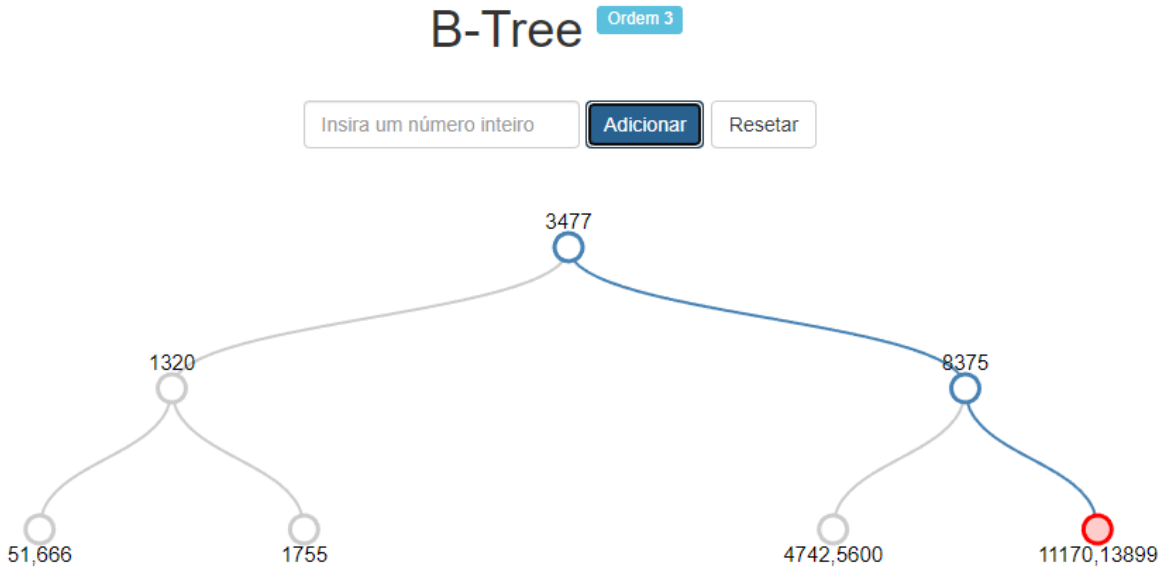
➢ **Inserindo o número: 666**



➢ **Inserindo o número: 4742**



➢ **Inserindo o número: 11170**



5. CONCLUSÃO

As árvores B são um tipo de árvores de busca que foram projetadas para minimizar o número de acessos à memória secundária. Como o número de acessos à memória secundária depende diretamente da altura da árvore, as árvores B foram projetadas para ter uma altura inferior para um dado número de chaves. Para manter o número de registros armazenados e, ao mesmo tempo, diminuir a altura, uma solução é aumentar o grau de ramificação da árvore (o número máximo de filhos que um nó pode ter). Assim, árvores B são árvores de busca que possuem um grau de ramificação geralmente muito maior que 2.

Claro, existem algumas desvantagens em usar índices árvores B, como por exemplo se o banco de dados for pequeno ele obterá menor performance em comparação com índices comuns, então recomenda-se sua aplicação apenas em bancos de dados com grandes quantidades de tabelas, campos e registros.

6. REFERÊNCIAS BIBLIOGRÁFICAS

- ACERVO LIMA. **Introdução de B-Tree**. Disponível em: <https://acervolima.com/introducao-de-b-tree/>. Acesso em: 17 de junho de 2022.
- GEEKSFORGEEKS. **Insertion in a B-Tree**. Disponível em: <https://www.geeksforgeeks.org/insertion-in-a-b-tree/>. Acesso em: 24 de junho de 2022.
- GURU99. **B-Tree in Data Structure.** Disponível em: <https://www.geeksforgeeks.org/insertion-in-a-b-tree/>. Acesso em: 24 de junho de 2022.
- JAVATPOINT. **B-Tree**. Disponível em: <https://www.javatpoint.com/b-tree>. Acesso em: 22 de junho de 2022.
- PAULO, Feofiloff. **Estrutura de Dados: Árvores B**. Disponível em: <https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/B-trees.html>. Acesso em: 17 de junho de 2022.
- PROGRAMIZ. **B-Tree**. Disponível em: <https://www.programiz.com/dsa/b-tree>. Acesso em: 21 de junho de 2022.