# Trabalho Algoritmos e Estruturas de Dados II
# Índices em Arquivo Sequencial-Indexado

**Universidade de Caxias do Sul**

**Área de Ciências Exatas e Tecnologia**

**Aluno: Vinicius Gonçalves**

## Descrição do Funcionamento do Código Fonte para criação de Índices em Arquivos:

- **create-partial-idx-appId.c**

Este programa em C é responsável por gerar um índice parcial com base no campo "appId" de um arquivo de dados binário sequencial (ordenado pelo campo “appId”). O índice parcial construído é uma estrutura que mapeia os valores do campo aos locais físicos dos registros correspondentes no arquivo de dados binário através de blocos. O programa funciona da seguinte maneira:

1. Abre o arquivo de dados binário (sequencial) e cria o arquivo de índice para escrita.
2. Define o tamanho do bloco de registros do arquivo de dados a ser indexado por um único registro no arquivo de índice.
3. Inicializa a variável recordCount para rastrear o número de registros percorridos no arquivo de dados.
4. Inicia um loop para realizar a inserção no arquivo de índice parcial. Para cada iteração, lê um registro do arquivo de dados, preenche uma estrutura struct IndexRecord com o valor do campo "appId" do registro atual e a posição física desse registro no arquivo de dados, escreve essa estrutura no arquivo de índice, atualiza recordCount para avançar para o próximo bloco de registros. O loop continua até que todos os registros tenham sido processados. Resultando em um arquivo com “n” registros, onde n = (total de registros no arquivo de dados / tamanho do bloco).

- **Create-idx-appName.c**

Este código C tem como objetivo criar um índice para registros de um arquivo de dados binários sequenciais, usando o campo "appName" (não ordenado no arquivo de dados). O índice gerado mapeia cada valor único do campo "appName" para a posição física do registro correspondente no arquivo de dados, sendo assim, o arquivo de índice é estruturado como uma tabela direta, tendo a mesma quantidade de registros do arquivo de dados. O programa funciona da seguinte maneira:

1. Abre o arquivo de dados binário para leitura e cria um arquivo de índice para escrita. Lê registros do arquivo de dados, um por um, e constrói um índice associando valores "appName" a posições físicas no arquivo de dados através da struct IndexRecord.
2. Após a criação do índice, o programa fecha os arquivos de dados e de índice adequadamente.
3. Em seguida, reabre o arquivo de índice e realiza a ordenação de seus registros através da função de comparação lexicográfica de strings “compareAppName” aplicada a função “qsort” da biblioteca padrão para ordenar o array de registros de índice em memória (o documento do trabalho não proibia realizar ordenação de registros utilizando a memória principal, apenas havia restrição quanto aos métodos de consulta).

### Descrição do Funcionamento do Código Fonte para criação de Índices em Memória:

Este conjunto de códigos faz parte do programa principal da aplicação e é responsável por criar índices em memória com base em estruturas de Árvores Binárias Auto-Balanceadas (AVL). Esses índices são criados para melhorar a eficiência das consultas aos registros de um arquivo de dados. Existem duas opções disponíveis no menu do programa principal, cada uma criando um índice para um campo diferente:

- **main.c AVL [rating] (Avaliação)**

Nesta opção, o código constrói um índice em memória usando uma estrutura de Árvore Binária Auto-Balanceada (AVL). A estrutura da AVL é a seguinte:

struct AVLNode { float price; struct LinkedListNode* positions; struct AVLNode* left; struct AVLNode* right; int height; };

O arquivo de dados é percorrido linha por linha, e os registros são inseridos na AVL com base no campo "rating" (avaliação). O campo "rating" é um valor em ponto flutuante que varia de 0.0 a 5.0. Para cada novo valor de "rating" encontrado, um novo nó na AVL é criado. Cada nó contém uma lista encadeada de posições físicas, permitindo agrupar registros com os mesmos valores de "rating". Portanto, esta estrutura representa uma AVL de Listas Encadeadas.

No entanto, é importante notar que esse método de indexação em memória pode não ser eficiente quando os valores de "rating" tendem a se repetir com alta frequência (situação presente no arquivo de dados), resultando em listas encadeadas longas. Idealmente, seria mais eficiente aplicar esse método a um campo com maior variabilidade de valores e dispersão entre os registros, o que resultaria em uma AVL com mais níveis e listas encadeadas menores.

- **main.c AVL [price] (Preço)**

Essa opção do menu também cria um índice em memória usando uma AVL. O processo é semelhante ao índice anterior, mas o índice é construído com base no campo "price" (preço). No entanto, esta AVL indexa apenas os aplicativos classificados como pagos (campo "FREE" é falso). Para simplificação da lógica do código, AVL [price] utiliza das mesmas estruturas e funções da AVL [rating]. (Uma das estruturas dos índices de memória ficavam a critério do aluno segundo o documento do trabalho).

## Descrição das Perguntas/Consultas que podem ser realizadas através dos Índices:

A aplicação permite a realização de diversas consultas com o uso de índices. Cada consulta é direcionada a campos específicos dos registros no arquivo de dados.

Abaixo, descrevemos as consultas disponíveis e como são implementadas:

- **Pesquisa Binária Direta no Arquivo de Dados Sequencial-Indexado [appId]**: Essa consulta permite encontrar um registro específico com base no campo "appId" e levando em consideração o arquivo de dados como sendo sequencial e ordenado pelo campo em questão. O usuário insere a chave "appId" desejada, e o sistema utiliza um método de pesquisa binária diretamente no arquivo de dados com uso de seek para localizar o registro correspondente. Tal opção foi desenvolvida para realizar comparações quanto a eficiência de uma busca direta em relação a uma busca com a utilização de um índice. Informa a quantidade de comparações realizada.
- **Consulta de Registros pelo Campo [price] para Apps Pagos/Não Gratuitos (AVL em Memória)**: Nessa consulta, o usuário pode pesquisar registros com base no campo "price" (preço) de aplicativos que não são gratuitos (campo "free" igual a false). A aplicação utiliza uma Árvore Binária Auto-Balanceada (AVL) em memória. O usuário escolhe se deseja listar todos os aplicativos de um determinado preço ou se utiliza o índice para localizar um único registro pelo seu “appId” e “price”.
- **Consulta de Registros pelo Campo [rating] (AVL em Memória)**: Essa consulta permite o usuário pesquisar registros com base no campo "rating" (avaliação). Novamente, uma AVL em memória é usada para acelerar a busca. O usuário pode escolher listar todos os registros com um determinado rating ou buscar por um único registro usando o índice como filtro de busca através dos campos “appId” e “rating”.
- **Consulta de Registros pelo Campo [appId] (Índice Parcial em Arquivo)**: Nesta consulta, o usuário pode inserir um "appId" específico, e o sistema

usa um índice parcial em arquivo para localizar o registro correspondente executando a pesquisa binária sobre o índice parcial. Caso o "appId" seja encontrado no índice, o sistema recupera o registro diretamente. Caso contrário, uma pesquisa binária no arquivo de dados é realizada com base na posição anterior indexada mais próxima da chave passada como parâmetro. Informa a quantidade de comparações, levando em consideração as comparações feitas no arquivo de índice mais as comparações no arquivo de dados.

- **Consulta de Registros pelo Campo [appName] (Índice em Arquivo)**: O usuário pode pesquisar registros com base no nome deles. O usuário insere o nome do aplicativo "appName", e o sistema utiliza o índice em arquivo para localizar os registros correspondentes através de pesquisa binária. Se houver vários registros com o mesmo nome de aplicativo, a aplicação recupera todos eles utilizando-se do fato do arquivo de índice estar ordenado pelo campo “appName”.