

Atividade Prática 04

Árvores B

Universidade Tecnológica Federal do Paraná (UTFPR), campus Apucarana
Curso de Engenharia de Computação
Disciplina de Estrutura de Dados 2 - EDCO4B
Prof. Dr. Luiz Fernando Carvalho
Prof. Dr. Rafael Gomes Mantovani

Instruções:

- Leia todas as instruções corretamente para poder desenvolver sua atividade/programa;
- Evite plágio (será verificado por meio de ferramentas automatizadas). Faça seu programa com os seus nomes de variáveis e lógica de solução. Plágios identificados anularão as atividades entregues de todos os envolvidos.
- Adicione comentários nos códigos explicando seu raciocínio e sua tomada de decisão. Porém, não exagere nos comentários, pois a própria estrutura do programa deve ser auto-explicativa.
- Salve sua atividade em um arquivo único, com todas as funções e procedimentos desenvolvidos. É esse **arquivo único** que deverá ser enviado ao professor.

1 Árvores B

Em Computação, uma Árvore B é uma estrutura de dados em árvore, auto-balanceada, que armazena dados classificados e permite pesquisas, acesso sequencial, inserções e remoções em tempo logarítmico. A árvore B é uma generalização de uma árvore de pesquisa binária em que um nó pode ter mais que dois filhos [2]. 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.

É normalmente usada em bancos de dados e sistemas de arquivos e foi projetada para funcionar especialmente em memória secundária como um disco magnético ou outros dispositivos de armazenamento secundário. As árvores B são semelhantes as árvores AVL, mas são melhores para minimizar operações de Entrada/Saída (E/S) de disco. Muitos sistemas de bancos de dados usam árvores B ou variações da mesma para armazenar informações. Dentre suas propriedades ela permite a inserção, remoção e busca de chaves numa complexidade temporal logarítmica e, por esse motivo, é muito empregada em aplicações que necessitam

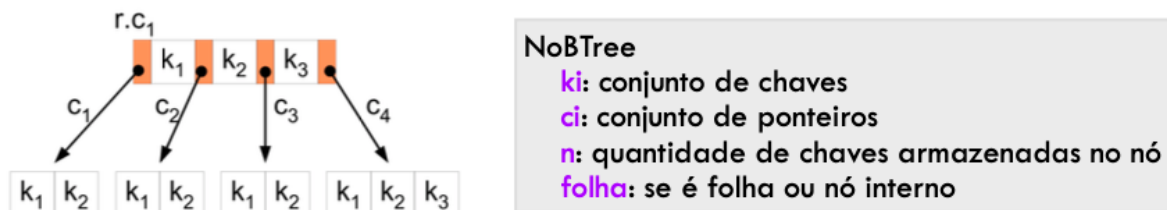


Figura 1: Árvore B (BTree) e seus elementos para implementação.

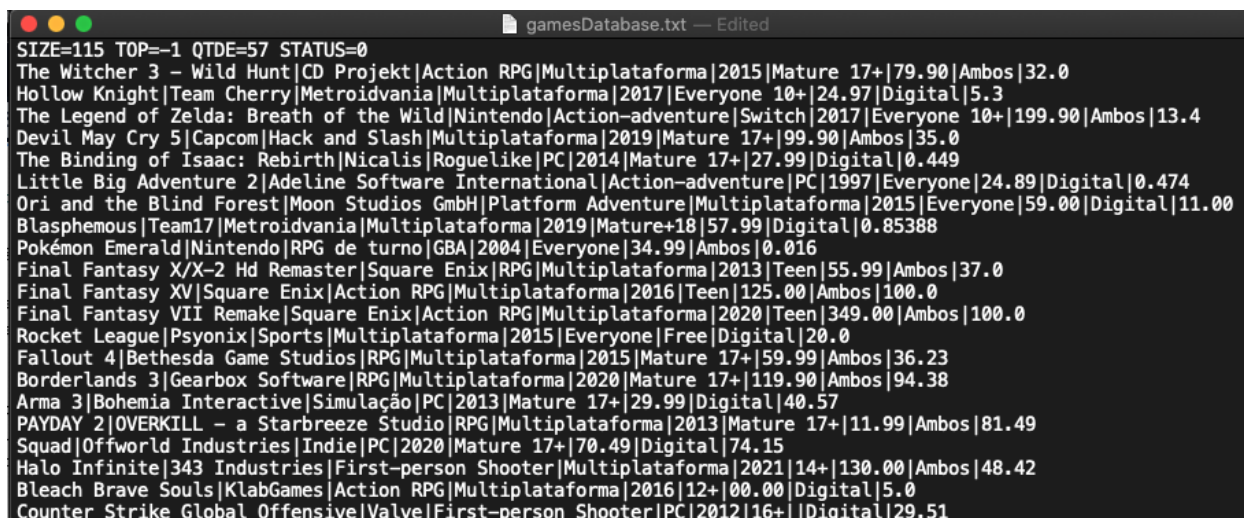
manipular grandes quantidades de informação tais como um banco de dados ou um sistemas de arquivos. Um exemplo de árvore B e as estruturas envolvidas em sua implementação pode ser vista na Figura 1.

2 Descrição da atividade

Ao longo de todo esse semestre, professor M também cansou da vida acadêmica, da rotina da universidade, e quer voltar às origens e aproveitar para jogar todos os games esquecidos no limbo. Ele tirou os consoles da caixa (sim, ele só joga consoles), e instalou tudo visando o entretenimento. Porém, para sua surpresa, há muitos jogos esquecidos e abandonados no limbo das suas caixas, tantos que ele nem consegue mais lembrar qual game ele jogou, ou qualquer tipo de informação está contida em cada game.

Mas, você (sim, você!), sabe que organizar as informações em uma estrutura de dados pode facilitar o professor M na busca pelos seus jogos perdidos, e na escolha de qual jogo redescobrir. O professor M lhe passa um arquivo contendo todas as informações dos jogos que possui e que gostaria de jogar. Para cada game, o professor tem anotado:

- **Título:** título do game;
- **Produtora:** quem desenvolveu o game;
- **Gênero:** qual o gênero do game;
- **Plataforma:** para qual tipo de máquina foi desenvolvido (PC, PS4, Xbox, etc...). Se for para vários, colocar “Multiplataforma”;
- **Ano:** data de lançamento do game;
- **Classificação:** classificação etária do game;
- **Preço:** média de preço para o produto;
- **Mídia:** se está disponível em formato físico, digital, ou em ambos;
- **Tamanho:** tamanho da mídia (em GB),



```

SIZE=115 TOP=-1 QTDE=57 STATUS=0
The Witcher 3 - Wild Hunt|CD Projekt|Action RPG|Multiplataforma|2015|Mature 17+|79.90|Ambos|32.0
Hollow Knight|Team Cherry|Metroidvania|Multiplataforma|2017|Everyone 10+|24.97|Digital|5.3
The Legend of Zelda: Breath of the Wild|Nintendo|Action-adventure|Switch|2017|Everyone 10+|199.90|Ambos|13.4
Devil May Cry 5|Capcom|Hack and Slash|Multiplataforma|2019|Mature 17+|99.90|Ambos|35.0
The Binding of Isaac: Rebirth|Nicalis|Roguelike|PC|2014|Mature 17+|27.99|Digital|0.449
Little Big Adventure 2|Adeline Software International|Action-adventure|PC|1997|Everyone|24.89|Digital|0.474
Ori and the Blind Forest|Moon Studios GmbH|Platform Adventure|Multiplataforma|2015|Everyone|59.00|Digital|11.00
Blasphemous|Team17|Metroidvania|Multiplataforma|2019|Mature+18|57.99|Digital|0.85388
Pokémon Emerald|Nintendo|RPG de turno|GBA|2004|Everyone|34.99|Ambos|0.016
Final Fantasy X/X-2 Hd Remaster|Square Enix|RPG|Multiplataforma|2013|Teen|55.99|Ambos|37.0
Final Fantasy XV|Square Enix|Action RPG|Multiplataforma|2016|Teen|125.00|Ambos|100.0
Final Fantasy VII Remake|Square Enix|Action RPG|Multiplataforma|2020|Teen|349.00|Ambos|100.0
Rocket League|Psyonix|Sports|Multiplataforma|2015|Everyone|Free|Digital|20.0
Fallout 4|Bethesda Game Studios|RPG|Multiplataforma|2015|Mature 17+|59.99|Ambos|36.23
Borderlands 3|Gearbox Software|RPG|Multiplataforma|2020|Mature 17+|119.90|Ambos|94.38
Arma 3|Bohemia Interactive|Simulação|PC|2013|Mature 17+|29.99|Digital|40.57
PAYDAY 2|OVERKILL - a Starbreeze Studio|RPG|Multiplataforma|2013|Mature 17+|11.99|Ambos|81.49
Squad|Offworld Industries|Indie|PC|2020|Mature 17+|70.49|Digital|74.15
Halo Infinite|343 Industries|First-person Shooter|Multiplataforma|2021|14+|130.00|Ambos|48.42
Bleach Brave Souls|KlabGames|Action RPG|Multiplataforma|2016|12+|00.00|Digital|5.0
Counter Strike Global Offensive|Valve|First-person Shooter|PC|2012|16+||Digital|29.51

```

Figura 2: Arquivo de Games do professor M.

Um vislumbre do arquivo de *games* do professor M pode ser visto na Figura 2. Pensando em melhorar o seu processo de consulta aos games, o professor M quer um programa que seja capaz de realizar buscas de maneira eficiente no arquivo apresentado acima. Por exemplo: retornar todos os jogos que contêm uma *substring* específica no título, ou todos os jogos que estão disponíveis em mesma plataforma, ou até mesmo retornar todos os jogos de um mesmo gênero. Essa é sua nova **missão**: desenvolver um programa com Árvores B (*BTree*) que satisfaça as consultas de *games* do arquivo do professor M.

3 Entradas do programa

O programa receberá **três** arquivos texto como parâmetros: um arquivo de dados com os games a serem manipulados, um arquivo de consulta, e um arquivo de saída. Abaixo, iremos detalhar cada um deles.

3.1 Arquivo de dados

O primeiro parâmetro é o arquivo de dados, o mesmo que é apresentado na Figura 2. Ele lista todos os *games* que o professor M gostaria de jogar. O armazenamento dos *games* é feito em um arquivo de registros de tamanho fixo, mas de campos com tamanho variáveis. Cada campo é separado por um pipe (|), e os registros finalizados por uma quebra de linha. Há um registro de cabeçalho (*header*) que contém algumas informações importantes para a execução do programa. Essas informações estão sumarizadas na Tabela 1.

No parâmetro **SIZE**, os registros terão valores fixos de 115 caracteres, contando os pipes e quebra de linha. O parâmetro **TOP** controla o reuso dos registros, depois de operações de leitura e escrita. Nessa atividade o valor será constante em -1, pois não iremos modificar os arquivos, apenas consultá-los. O parâmetro **QTDE** contém a quantidade de registros armazenados no arquivo. O último parâmetro (**STATUS**), indica se o índice, após ser manipulado na

Tabela 1: Parâmetros contidos no cabeçalho do arquivo de entrada

Parâmetro	Descrição	Opções Válidas
SIZE	tamanho dos registros (bytes/caracteres)	115
TOP	índice da posição do topo da pilha lógica de reuso	-1
QTDE	quantidade de registros contidos no arquivo de entrada	[0, ...]
STATUS	indica se os índices encontram-se salvos nos arquivos de dump	0

memória, está salvo nos arquivos de *dump* (backup). Na nossa aplicação, esse valor não será modificado.

3.2 Arquivo de Consulta

Um arquivo texto contendo o tipo de informação a ser usada como referência na busca, e a *string* de busca, uma por linha. Assim, na primeira linha existirá o nome do campo ao qual queremos buscar a informação. Este campo pode ser qualquer um dos campos válidos do registro no arquivo de dados. Caso o arquivo de consulta possua uma *string* na primeira linha diferente dos valores possíveis, o programa deve indicar o erro e não executar.

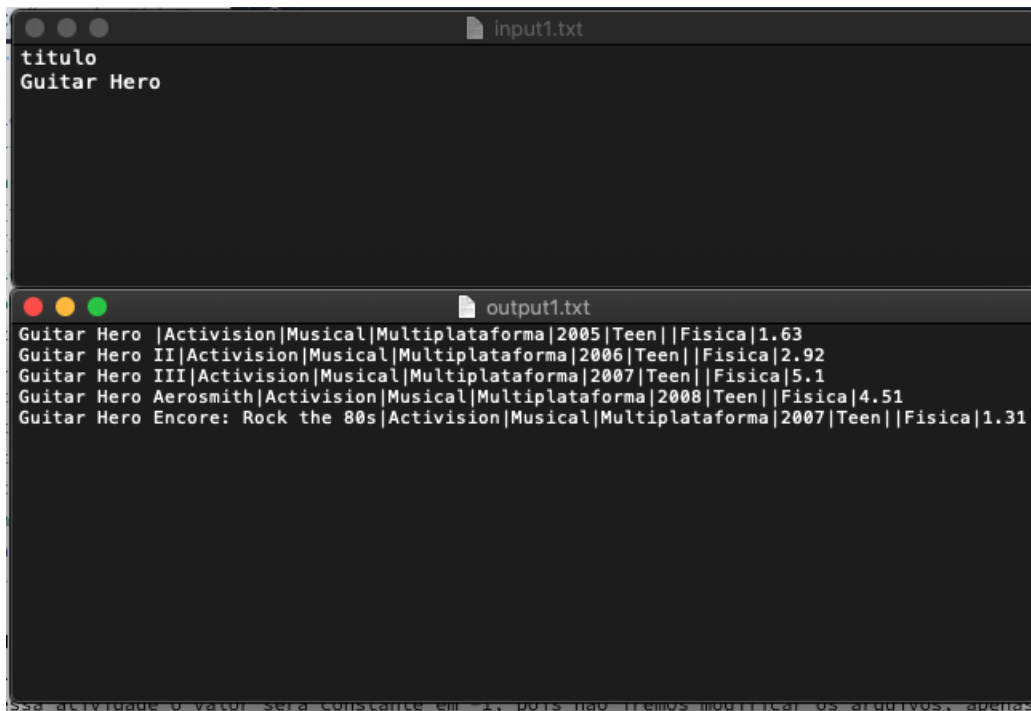


Figura 3: Valores de entrada e correspondente arquivo de saída gerado pelo programa.

Na segunda linha do arquivo teremos a *string* de busca (o valor de consulta). Um exemplo é apresentado na Figura 3. No exemplo em questão, queremos buscar por título, todos os games que possuem a *string* “Guitar Hero” em seu título.

3.3 Arquivo de saída

Como consequência, o arquivo de saída deve conter apenas os registros que satisfazem os critérios de busca especificados pelo arquivo de consulta. Ou seja, no exemplo da Figura 3 são retornados todos os games com **Guitar Hero** no título. Lembre-se que no arquivo de saída devem existir mensagens de erro, ou indicativos, de que a consulta não pôde ser executada quando nenhum valor encontrado ou a consulta foi solicitada do jeito errado.

4 Rodando o programa

Para rodar o programa por linha de comando, manipular os argumentos **argc** e **argv** da função **main**. Para executar o programa por linha de comando, deve-se obedecer o seguinte padrão:

```
[nome do programa] [arquivo de dados] [arquivo de entrada] [arquivo de saída]
```

Exemplo de execução do programa:

```
python3 BTreeGames.py games.txt entrada01.txt saida01.txt
```

5 Orientações gerais

Além da funcionalidade desejada, implementar também o controle de erros, para lidar com exceções que possam ocorrer, como por exemplo:

- problemas nas aberturas dos arquivos de entrada e saída;
- arquivos de entrada vazios (sem informação);
- arquivos de entrada fora do padrão esperado (opções inválidas para consulta);
- etc.

Opcionalmente, para acompanhamento do desenvolvimento, pode-se criar um repositório individual no **github**.

6 Critérios de correção

A nota na atividade será contabilizada levando-se em consideração alguns critérios:

1. pontualidade na entrega;
2. não existir plágio;
3. completude da implementação (tudo foi feito);
4. o código executa;
5. uso de `argc` e `argv` para controle dos arquivos de teste;
6. implementar a leitura dos dados de entrada via arquivo texto;
7. implementação correta das estruturas necessárias (campos, registros e sua manipulação, estruturas, etc);
8. legibilidade do código (identação, comentários nos blocos mais críticos);
9. implementação dos controles de erros (arquivos de entrada inválidos, e erros no programa principal);
10. controle de memória: chamar o destrutor e desalocar a memória de tudo se usar estruturas dinâmicas, fechar os arquivos, etc;
11. executar corretamente os casos de teste.

Em cada um desses critérios, haverá uma nota intermediária valorada por meio de conceitos:

- **Sim** - se a implementação entregue cumprir o que se esperava daquele critério;
- **Parcial** - se satisfizer parcialmente o tópico;
- e **Não** se o critério não foi atendido.

7 Padrão de nomenclatura

Ao elaborar seu programa, crie um único arquivo fonte (.py) seguindo o padrão de nome especificado:

ED2-AT04-BTreeGames-<NOME>.py

Exemplo:

ED2-AT04-BTreeGames-TamaraSausageWater.py

A entrega da atividade será via Moodle: o link será disponibilizado na página da disciplina.

8 Links úteis

- Arquivos em Python:
 - <https://www.geeksforgeeks.org/reading-writing-text-files-python/>
 - https://www.w3schools.com/python/python_file_open.asp
 - <https://www.pythontutorial.net/python-basics/python-read-text-file/>
- Argumentos de Linha de comando no Python:
 - https://www.tutorialspoint.com/python3/python_command_line_arguments.htm
 - <https://realpython.com/python-command-line-arguments/>
 - <http://devfuria.com.br/python/sys-argv/>

Referências

- [1] Michael J. Folk; Bill Zoellick; Greg Riccardi. File Structures, 3rd edition, Addison-Wesley, 1997.
- [2] Thomas H. Cormen,; Ronald Rivest; Charles E. Leiserson; Clifford Stein. Algoritmos - Teoria e Prática - 3^a Ed. Elsevier - Campus, 2012.
- [3] Nivio Ziviani. Projeto de algoritmos com implementações: em Pascal e C. Pioneira, 1999.
- [4] Adam Drozdek. Estrutura De Dados e Algoritmos em C++. Cengage, 2010.