Trabalho 2 – Classificação e Pesquisa de Dados 2023/02

mini-Steam: algoritmo localizador de jogos para um loja digital

Você foi selecionado como o desenvolvedor principal para um projeto ambicioso: a criação de um novo mecanismo de busca para uma emergente loja digital de jogos. O objetivo é garantir que os gamers ao redor do mundo possam encontrar seus jogos preferidos em um piscar de olhos.

Você vai se deparar com milhares de jogos, desde os mais populares RPGs até os *indies* mais desconhecidos. É seu trabalho garantir que o "mini-Steam" – algoritmo proprietário a ser desenvolvido – possa localizá-los de forma rápida e precisa.

Em específico, deve-se permitir que os gamers encontrem jogos baseados em seus preços ou seus gêneros e, para isso, você terá o desafio de integrar estruturas de dados avançadas. Seguem informações da loja de jogos:

- **Catálogo de jogos:** Cada jogo possui um ID único, título, desenvolvedor e preço. Cada jogo pertence a um ou mais gêneros (por exemplo, RPG, Ação, Estratégia).
- **Gerenciamento de jogos por preço:** deve ser realizado por meio de uma **Árvore Binária de Busca** (BST) para adicionar ou encontrar um jogo com base em seu preço. Deve ser possível realizar buscas por preço simples (por exemplo, todos os jogos que custam R$100 reais) ou por faixa de preço, levando em consideração um valor mínimo e um valor máximo (por exemplo, jogos com preço entre R$50 e R$200 reais ou jogos com preço inferior a R$50 reais). Para simplificação, pode-se considerar que preços são valores positivos inteiros.
- **Gerenciamento de jogos por gênero:** deve ser feito por meio de um sistema *hashing (**hash table**)* onde cada jogo, ao ser adicionado, também é indexado por seu(s) gênero(s). O gênero será usado como uma chave que indexa para uma lista de IDs de jogos, o que permitirá a recuperação rápida de todos os jogos em um determinado gênero. Deve existir uma opção de busca de jogos por gênero.

Desafio extra – Busca rápida por título: implemente uma estrutura de dados que permita a inserção e busca eficiente de jogos por título. Considere que os títulos de jogos podem variar em comprimento e que os usuários podem nem sempre lembrar do título completo.

- **Dica:** Uma *Trie* (também conhecida como árvore de prefixos) pode ser uma estrutura de dados adequada para essa tarefa, pois permite a busca eficiente de palavras por seus prefixos.

Critérios de avaliação:

- O foco do trabalho será na integração eficaz entre os componentes BST e *hash table* no mecanismo de busca, em específico:

    1. Implementação correta e eficiente da BST para gerenciamento dos jogos por preço (criação da estrutura de árvore, adição dos nodos e navegação);
    2. Algoritmo de busca por preço simples sobre a BST;
    3. Algoritmo de busca por faixas de preço sobre a BST;
    4. Indexação eficaz baseada em gênero usando *hashing*;
    5. Tratamento adequado de jogos que pertencem a múltiplos gêneros e colisões, considerando jogos de um mesmo gênero;
    6. Consulta adequada de acordo com o valor a ser buscado na tabela *hash*.

- Para além da apresentação em seminário (mais detalhes abaixo), serão avaliadas todas as peculiaridades solicitadas na especificação, em conjunto com a criatividade do aluno na implementação da solução:

    - Decisões de implementação serão avaliadas e distinguem uma implementação das demais;
    - Certifique-se de modularizar as funcionalidades do jogo em funções (evite trechos de código redundantes) com escopo definido e comportamento adequado (estrutura e organização do código fazem parte da avaliação!);
    - O código do jogo deve estar devidamente indentado.

- Haverão *checkpoints* nas aulas anteriores à entrega como forma de acompanhamento. Trabalhos serão apresentados em seminário junto à turma. **Instruções gerais para a apresentação:**

    - Durante a apresentação, o aluno será questionado sobre a implementação do trabalho;
    - Deverá ser explicado como a solução implementada funciona, como as estruturas foram manipuladas e como os algoritmos de pesquisa foram desenvolvidos;
    - Avalie e discuta sobre a complexidade temporal e de espaço;
    - Mostre exemplos práticos com dados fictícios e destaque, se aplicável, como a natureza e a ordem de inserção dos dados influencia a eficiência da busca pelos resultados;
    - Comente sobre os principais desafios enfrentados e aprendizados no desenvolvimento do trabalho.

Orientações:

Linguagem de programação de livre escolha (sugestão: Python), desde que **não** se utilize de métodos de pesquisa "*built-in*" ou de bibliotecas de terceiros para implementação das estruturas. Os algoritmos de busca e o gerenciamento das estrutura deve ser desenvolvido em código próprio;

Além da especificação anterior, que descreve o programa a ser implementado, deve-se considerar que:

- O trabalho deverá ser desenvolvido **individualmente**;
- **Data de entrega:** até dia 27/11/2023 (segunda-feira) às 12h00m, ou seja, ao meio-dia do dia da aula de apresentação;
- **Forma de entrega:** submeter o arquivo do código-fonte e quaisquer outro arquivo adicional (ou link para repositório no GitHub) que se fizer necessário na tarefa do Google Classroom;
- Serão utilizadas ferramentas de análise de similaridade de código-fonte (*e.g.*, MOSS, JPlag). Cópias e plágios receberão nota **zero**. Em caso de suspeita, o professor poderá solicitar ao aluno que explique, em detalhes, a implementação realizada. Lembre-se: se for utilizar de ferramentas de IA, use-as com sabedoria, para entender conceitos e ter inspirações. Não assuma que a resposta gerada estará certa sem avaliar com senso crítico e com base nos conhecimentos absorvidos ao longo da disciplina.
- O conteúdo abordado neste trabalho será alvo de avaliação na avaliação final e exame da disciplina.
- Abaixo é disponibilizado um template básico para iniciar a estruturação do código (**livre escolha**, adapte se e como achar necessário).

```
class Jogo:
    def __init__(self, jogo_id, titulo, desenvolvedor, preco, generos):
        self.jogo_id = jogo_id
        self.titulo = titulo
        self.desenvolvedor = desenvolvedor
        self.preco = preco
        self.generos = generos  # Lista, pois um jogo pode pertencer a múltiplos gêneros

class NoJogo:
    def __init__(self, jogo):
        self.jogo = jogo
        self.esquerda = None
        self.direita = None

class ArvoreJogos:
    def __init__(self):
        self.raiz = None

    def inserir(self, jogo):
        # TODO: Inserir jogo na BST com base no preço

    def buscar_por_preco(self, preco):
        # TODO: Buscar jogos por preço simple

    def busca_por_faixa_preco(self, preco_minimo, preco_maximo):
        # TODO: Recuperar jogos dentro de uma faixa de preço

class HashGeneros:
    def __init__(self):
        self.genero_para_jogos = {}

    def adicionar_jogo(self, jogo):
        # TODO: Adicionar jogos na hash table de acordo com seu(s) gênero(s)

    def obter_jogos(self, genero):
        # TODO: Buscar jogos de um determinado gênero

class MotorBuscaJogos:
    def __init__(self):
        self.catalogo_jogos = ArvoreJogos()
        self.generos = HashGeneros()

    # ... Outros métodos para adicionar jogos, busca, etc.
```