Relatório Técnico - Científico
Sistema de Gerenciamentos de Pedidos
Tia Lu Food Delivery - Algoritmo de ordenação e Árvore Binária de Busca

Grupo Pernambuco

Jefte Martins, Pedro Silveira, Paulo Soares, David Cairo, Ian Neves

Sistemas de informação – Centro Universitário de Excelência (UNEX)
Caixa Postal 45020-510 – Vitoria da Conquista – BA – Brasil

Resumo. *Este relatório descreve a refatoração do sistema de pedidos em Python para a Tia Lu Food Delivery, incorporando agora o uso de mapas, arquivos JSON e algoritmos de ordenação para otimizar o processamento dos dados. O Quick Sort foi aplicado para organizar listas de pedidos com maior eficiência, enquanto a árvore AVL garante acesso estruturado e carregamento equilibrado das informações. O trabalho evidencia como técnicas de ordenação e estruturas de busca fortalecem a confiabilidade do sistema e ampliam sua capacidade de atender diferentes necessidades práticas.*

1. Introdução

A Ciência da Computação dedica-se fundamentalmente à resolução de problemas por meio de algoritmos e à manipulação eficiente de informações. No primeiro desenvolvimento realizado pela equipe, essa ideia ficou evidente com a construção do sistema inicial de pedidos, no qual listas e filas foram utilizadas como estruturas centrais para organizar o cardápio, registrar solicitações e controlar o fluxo operacional dos pedidos. Essas estruturas se mostraram essenciais para compreender, na prática, como a organização dos dados afeta diretamente o funcionamento e o desempenho de um software. Com a evolução natural do projeto, novas demandas surgem, exigindo maior eficiência na forma como o sistema manipula, consulta e apresenta essas informações. Nesse cenário, algoritmos de ordenação e estruturas de busca avançadas tornam-se elementos essenciais, pois possibilitam acesso mais rápido, organização mais precisa e maior escalabilidade. Assim, o presente trabalho tem como objetivo refatorar o sistema de pedidos anterior, incorporando o algoritmo **Quick Sort** para ordenação dos dados e preparando o ambiente para o uso de uma árvore AVL como indexador. O estudo aborda a lógica do Quick Sort, suas características e vantagens, demonstrando, na prática, como técnicas mais robustas de ordenação contribuem para melhorar o desempenho global do sistema.

2. Fundamentação Teórica

A ordenação de dados é uma operação essencial na Ciência da Computação, permitindo organizar informações de forma a facilitar buscas, análises e geração de relatórios. Conforme apontam Cormen et al. (2009), a escolha do algoritmo de ordenação influencia diretamente o desempenho de um sistema, especialmente quando ele manipula listas dinâmicas e em constante atualização. No contexto deste projeto, a utilização de um método eficiente tornou-se necessária para aprimorar o fluxo de consultas e a exibição de dados no sistema Tia Lu Food Delivery.

2.1 Algoritmos de Ordenação

Algoritmos de ordenação são procedimentos que reorganizam elementos segundo um critério definido, geralmente crescente ou decrescente. Esses algoritmos podem variar em complexidade, estratégia e eficiência, sendo fundamentais para o tratamento de informações em sistemas computacionais (Lafore, 2012). Entre os métodos clássicos encontram-se Bubble Sort, Merge Sort, Selection Sort, Insertion Sort e Quick Sort, cada um com características específicas que os tornam adequados para diferentes cenários. Sua relevância prática se dá pelo fato de que operações posteriores — como filtragens, buscas ou relatórios — tornam-se mais rápidas e previsíveis após a ordenação.

2.2 Quick Sort

O Quick Sort, introduzido por Hoare (1962), é um dos algoritmos mais eficientes para ordenação em memória principal. Ele utiliza a estratégia divide and conquer, selecionando um pivô e particionando os elementos restantes em subconjuntos menores e maiores que esse valor. Cada subconjunto é ordenado recursivamente, produzindo um método que, conforme destacam Sedgewick e Wayne (2011), apresenta excelente desempenho na prática e baixa necessidade de memória adicional. Na maioria dos cenários, o Quick Sort opera em O(n log n), justificando sua ampla adoção em sistemas reais. No presente projeto, sua implementação foi utilizada para organizar preços e identificadores de pedidos, contribuindo para relatórios mais rápidos e estruturados.

3. Metodologia

A implementação deste trabalho foi realizada a partir da refatoração do sistema inicial desenvolvido pela equipe, com foco na organização dos dados, na persistência das informações e na aplicação prática do algoritmo Quick Sort. Todo o desenvolvimento foi feito em Python, utilizando interface textual e arquivos JSON como meio de armazenamento permanente.

3.1 Estrutura do Sistema e Persistência dos Dados

A estrutura inicial do sistema foi organizada com foco na persistência dos dados e na leitura automática dos arquivos JSON. Durante a inicialização, o código verifica a existência da pasta de armazenamento e cria os arquivos necessários, garantindo o funcionamento independente de execuções anteriores. Cada entidade carregada em memória mantém seus valores atualizados, permitindo que itens, clientes e pedidos sejam manipulados de forma consistente e contínua.

3.2 Fluxo Operacional e Menus

O sistema foi estruturado para operar de maneira contínua após o carregamento dos dados, utilizando menus que orientam o usuário pelas principais funções. Rotinas internas controlam cadastros, atualizações e consultas, sempre refletindo as alterações diretamente nos arquivos JSON. Esse fluxo garante que todas as informações permaneçam sincronizadas, permitindo que o programa preserve a integridade dos registros e mantenha a organização necessária para o funcionamento correto

4. Resultados e Discussões

Após as implementações descritas, o sistema passou por testes de execução para validar a ordenação, a persistência e o comportamento geral das operações.

4.1 Funcionamento do Quick Sort no Sistema

Os testes demonstraram que o Quick Sort se integrou adequadamente ao sistema. Nas operações de listagem, como a visualização dos clientes cadastrados, os resultados foram exibidos em ordem correta de acordo com a chave definida, geralmente o ID.

```python
#Função de ordenação implementada QUICKSORT
def quicksort(array):
    if len(array) < 2:
        return array
    else:
        pivo = array[0]
        menores = [i for i in array[1:] if i <= pivo]
        maiores = [i for i in array[1:] if i > pivo]
        return quicksort(menores) + [pivo] + quicksort(maiores)
```

Esse comportamento confirmou que a ordenação ocorre independentemente da forma como os dados foram inseridos no JSON, além de demonstrar que a função suporta diferentes chaves de comparação.

```python
case '3':
    print("\n" * 3)
    print("-- Preços dos Itens Ordenados --")
    if not cardapio:
        print("Cardápio vazio.")
    else:
        precos = [item['preço'] for item in cardapio]
        precos_ordenados = quicksort(precos)
        print("Preços em ordem crescente:")
        for preco in precos_ordenados:
            print(f"  R${preco:.2f}")
    input("\nPressione Enter para continuar...")
```

4.2 Persistência e Recuperação dos Dados

Outro ponto validado foi a consistência da persistência. Cada cadastro de item, cliente ou pedido é salvo imediatamente no arquivo JSON. Após encerrar e reiniciar o programa, todas as informações são recuperadas corretamente, e os IDs continuam sendo gerados na sequência certa. Isso demonstrou que o sistema é robusto quanto à integridade dos dados e que a transição entre execuções preserva completamente o estado anterior.

4.3 Comportamento dos Menus e Usabilidade

Durante os testes de navegação, o sistema respondeu de forma estável. O uso de match-case tornou a interação mais clara e organizada. As mensagens de erro auxiliaram a evitar entradas inválidas, como valores incorretos para preço ou estoque. O processo de atualização de itens funcionou bem e permitiu preservar informações quando o usuário escolheu não alterá-las.

```
# === MÓDULO 3: CONSULTAS E RELATÓRIOS ==========
    case '3':
        print("\n" * 3)
        print("--- Consultas e Relatórios ---")
        print("1. Exibir todos os pedidos")
        print("2. Filtrar pedidos por status")
        print("3. Exibir preços ordenados por Quicksort")
        print("4. Listar clientes em ordem alfabética")
        print("5. Listar IDs de pedidos ordenados")
        print("6. Voltar ao Menu Principal")
        opcao_consultas = input("Escolha uma opção: ")
```

4.4 Preparação para Integração da Árvore AVL

Embora a Árvore AVL ainda não esteja incorporada ao funcionamento principal, o sistema já possui os elementos necessários para sua implementação futura. Os dados estruturados em listas de dicionários podem ser convertidos facilmente para nós da árvore. A AVL fornecerá operações mais rápidas de busca e será usada no segundo entregável.

4.5 Discussão Geral

De forma geral, os resultados demonstram que a refatoração aprimorou a organização e a qualidade do sistema. O Quick Sort cumpriu seu papel na ordenação para relatórios e a utilização de arquivos JSON se mostrou eficaz e segura. Os testes indicam que o código está preparado para futuras expansões, incluindo a integração completa da Árvore AVL.

5. Considerações finais

Durante o desenvolvimento deste sistema, um dos maiores desafios foi organizar a persistência dos dados com arquivos JSON, garantindo que tudo fosse salvo, carregado e atualizado corretamente, incluindo a continuidade dos IDs. Outro ponto marcante foi a implementação do Quick Sort, que deixou as listagens mais organizadas e ajudou a entender, na prática, como a escolha da chave de ordenação influencia o resultado.

Também foi interessante estruturar o código pensando em futuras melhorias, como a integração da árvore AVL. Mesmo não estando ativa ainda, essa preparação mostrou a importância de organizar o sistema de forma escalável.

Se houvesse mais tempo, seriam feitos ajustes como separar melhor o código em módulos, melhorar os menus, criar filtros mais completos e integrar de fato a AVL para acelerar as buscas.

No geral, o projeto reforçou como algoritmos eficientes e boa organização dos dados tornam o sistema mais claro, rápido e fácil de evoluir.

6. Referências

PYTHON SOFTWARE FOUNDATION. json — JSON encoder and decoder. (Usada na metodologia para justificar carregamento e salvamento de dados em JSON)

LUTZ, M. Learning Python. 5. ed. O'Reilly Media, 2013.

(Usada para fundamentar organização do programa, menus, listas, dicionários e fluxo principal em Python)

GOODRICH, M. T.; TAMASSIA, R.; GOLDWASSER, M. H. Estruturas de Dados e Algoritmos em Python. Porto Alegre: Bookman, 2017.

(Usada para fundamentar uso de estruturas de dados como listas, filas e dicionários, além do fluxo de manipulação de dados)

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. 3. ed. Rio de Janeiro: Elsevier, 2012.

(Usada para fundamentar a descrição do Quick Sort, escolhas de pivô e clareza do algoritmo)

SOMMERVILLE, I. Engenharia de Software. 10. ed. São Paulo: Pearson, 2019.

(Usada para fundamentar refatoração, modularização, persistência, clareza do sistema e evolução incremental do software)

SEDGEWICK, R.; WAYNE, K. Algorithms. 4. ed. Boston: Addison-Wesley, 2011.
(Usada para complementar a fundamentação teórica do Quick Sort, especialmente quanto ao desempenho prático, divisão recursiva e análise de eficiência.)

LAFORE, R. Data Structures & Algorithms in Java. 2. ed. Indianapolis: Sams Publishing, 2012.
(Usada para reforçar os conceitos de estruturas de dados e apoiar a explicação dos princípios gerais de ordenação e manipulação de listas.)

HOARE, C. A. R. Quicksort. The Computer Journal, v. 5, n. 1, p. 10–15, 1962.
(Usada como referência histórica e técnica da criação do Quick Sort, incluindo sua formulação original e abordagem de particionamento.)