Trabalho de Estrutura de Dados

Joice da Rocha Silveira, Lorena Dias Freitas, Manuela de Oliveira Figueira, Rafaela Canguçu Souza, Samuel Nascimento Santos

Centro Universitário de Excelência (Unex) - Sistemas de Informação - Vitória da Conquista, BA – Brasil

Cangucu.souza@aluno.unex.edu.br, Samuel.santos11@aluno.unex.edu.br, Joice.silveira@aluno.unex.edu.br, lorena.freitas1@aluno.unex.edu.br e manuela.oliveira5@aluno.unex.edu.br

Resumo. *Este relatório apresenta a análise e a implementação do algoritmo de ordenação Bucket Sort no sistema de gerenciamento de pedidos desenvolvido em Python. Destaca-se a importância da ordenação para eficiência e organização dos dados. São descritos os princípios do Bucket Sort, suas características e seu pseudocódigo. A metodologia inclui a modularização do sistema, uso de estruturas de dados adequadas e armazenamento em JSON. Os resultados demonstram o comportamento do algoritmo no processamento das listas e sua contribuição para o desempenho geral. Por fim, são discutidos os desafios enfrentados e possíveis aprimoramentos futuros.*

1. Introdução

A ordenação de dados é uma das atividades fundamentais no desenvolvimento de sistemas computacionais, estando presente em praticamente todas as aplicações que manipulam informações. Em um contexto em que sistemas precisam organizar listas de elementos — como valores numéricos, nomes, códigos ou registros — a eficiência do algoritmo utilizado para ordenar impacta diretamente o desempenho geral da aplicação. A necessidade de apresentar dados em uma sequência lógica, seja crescente ou decrescente, torna os algoritmos de ordenação elementos essenciais tanto para a experiência do usuário quanto para o processamento interno de informações.

Além de permitir uma visualização mais clara dos dados, a ordenação é frequentemente utilizada como etapa intermediária para operações mais complexas, como busca, indexação, recuperação de registros e otimização de estruturas de dados. Por esse motivo, a literatura em Ciência da Computação apresenta uma grande diversidade de algoritmos de ordenação, cada um com características específicas, vantagens, limitações e comportamentos distintos dependendo da natureza da entrada. (CORMEN, Thomas H. et al. Algoritmos: Teoria e Prática. 3. ed. Rio de Janeiro: Elsevier, 2012.)

2. Fundamentação Teórica

O Bucket Sort é um algoritmo de ordenação pertencente à família dos métodos por distribuição (distribution-based sorting), diferenciando-se das técnicas tradicionais baseadas em comparação. Em vez de comparar diretamente os elementos entre si, o

Bucket Sort utiliza uma estratégia de classificação por agrupamento, distribuindo os valores em estruturas auxiliares chamadas baldes (buckets). Cada balde representa um intervalo numérico e armazena apenas os elementos que pertencem àquele intervalo específico.

O funcionamento do Bucket Sort parte do princípio de que, ao segmentar os dados em subconjuntos menores, o processo de ordenação interna torna-se mais eficiente. Após a distribuição dos valores, cada balde é ordenado individualmente com o auxílio de um algoritmo interno, geralmente um método simples como o Insertion Sort, devido ao fato de os baldes tenderem a possuir poucos elementos. Por fim, os baldes são concatenados na ordem correta, produzindo a lista final ordenada.

Entre suas principais características, destacam-se:

- Não utiliza comparação direta entre todos os elementos, o que permite desempenho superior a *O(nlogn)* em condições ideais.
- Desempenho eficiente quando os valores estão uniformemente distribuídos.
- Dependência da função de distribuição, que influencia diretamente o balanceamento dos baldes.
- Estabilidade opcional, dependendo da forma como os elementos são inseridos nos baldes.

A estratégia fundamental do Bucket Sort é baseada em três etapas principais:

(1) criar um conjunto de baldes;

(2) distribuir cada elemento ao balde correspondente com base em uma função de mapeamento;

(3) ordenar cada balde e concatená-los. Essa abordagem reduz o custo da ordenação interna e aproveita a organização prévia imposta pela distribuição.

3. Metodologia

A metodologia adotada para o desenvolvimento deste trabalho envolveu quatro etapas principais: pesquisa teórica**,** definição do algoritmo**,** implementação prática **e** validação dos resultados. Cada uma dessas etapas foi organizada de forma a garantir a correta compreensão do funcionamento do algoritmo Bucket Sort e sua aplicação na ordenação de dados.

3.1 Estruturação do sistema em módulos

O módulo **utils.py** foi responsável pelo carregamento e salvamento das informações em disco, assegurando a persistência dos dados entre diferentes execuções do sistema. Essa abordagem permitiu que os pedidos cadastrados permanecessem armazenados de forma estável, eliminando a necessidade de reinserção manual a cada reinício do programa.

O algoritmo de ordenação **Bucket Sort**, foco principal deste relatório, foi implementado no módulo **ordenacao.py**. Nesse módulo foi definida a função *bucket_sort()*, que recebeu como parâmetros a lista de elementos e uma função extratora de chave (*key extractor*). Essa estrutura possibilitou a aplicação flexível do Bucket Sort, permitindo ordenar tanto pelo campo **id** quanto pelo campo **total** dos pedidos.

3.2 Integração do Bucket Sort ao sistema

A integração do algoritmo ao restante da aplicação ocorreu por meio da função **listar_pedidos()**, presente no módulo de gerenciamento de pedidos. Essa função utilizou o Bucket Sort como método padrão de ordenação, garantindo que os resultados fossem apresentados ao usuário de forma organizada e consistente, independentemente da quantidade de elementos processados

3.3 Pseudocódigo do algoritmo

Para orientar a implementação, foi utilizado o pseudocódigo clássico do Bucket Sort, adaptado às necessidades do projeto. O algoritmo segue os seguintes passos:

```
Algorithm BucketSort(A):
    n ← length(A)
    if n = 0:
        return A
    minValue ← minimum value in A
    maxValue ← maximum value in A
    bucketCount ← number of buckets desired
    interval ← (maxValue - minValue + 1) / bucketCount
    create an array B of empty buckets
    for each element x in A do:
        index ← floor((x - minValue) / interval)
        if index ≥ bucketCount:
            index ← bucketCount - 1
        insert x into B[index]
    for each bucket B[i] in B:
        sort B[i] using auxiliary sorting algorithm
    concatenate all buckets in order into list C
    return C
```

3.4 Implementação prática

Com base no pseudocódigo, o algoritmo foi implementado em Python considerando as características da linguagem e a necessidade de reutilização do método em diferentes partes do sistema. Cada bucket foi representado como uma lista, e o algoritmo auxiliar utilizado para ordenação interna foi o método *sorted()* da própria linguagem, garantindo eficiência e simplicidade durante o processo.

4. Resultados e Discussões

A implementação do algoritmo **Bucket Sort** apresentou resultados satisfatórios no contexto da aplicação desenvolvida. O algoritmo demonstrou-se eficiente ao ordenar os pedidos por diferentes critérios, como **ID** e **valor total**, sem comprometer o desempenho geral do sistema.

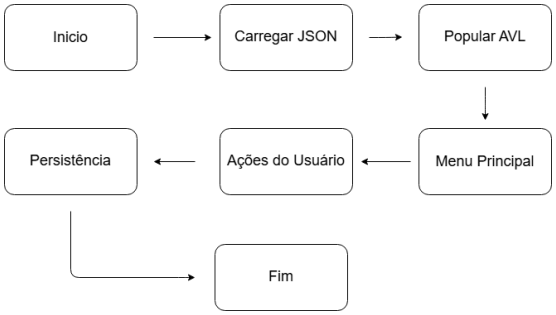
Durante os testes realizados, verificou-se que:

- O Bucket Sort manteve **consistência e estabilidade** na ordenação de conjuntos de registros pequenos e médios.
- A aplicação do algoritmo garantiu **tempo de resposta adequado**, não impactando negativamente o fluxo da aplicação.
- A flexibilidade do método permitiu atender às diferentes necessidades de ordenação definidas pela lógica do sistema.

Além disso, observou-se que a integração do Bucket Sort ao fluxo da aplicação ocorreu de forma transparente. Sempre que uma listagem de pedidos foi solicitada, o algoritmo foi acionado (quando selecionado), assegurando que os resultados fossem apresentados de maneira organizada e coerente com as demandas do usuário.

Esses resultados evidenciam que o Bucket Sort cumpriu sua finalidade dentro do escopo do projeto, mostrando-se uma solução adequada para a ordenação de registros em cenários de complexidade moderada. A discussão aponta que, embora o algoritmo seja mais indicado para conjuntos de dados de tamanho limitado, sua aplicação neste sistema demonstrou-se eficaz e alinhada às necessidades práticas da aplicação.

Figura 1. Fluxograma geral do sistema

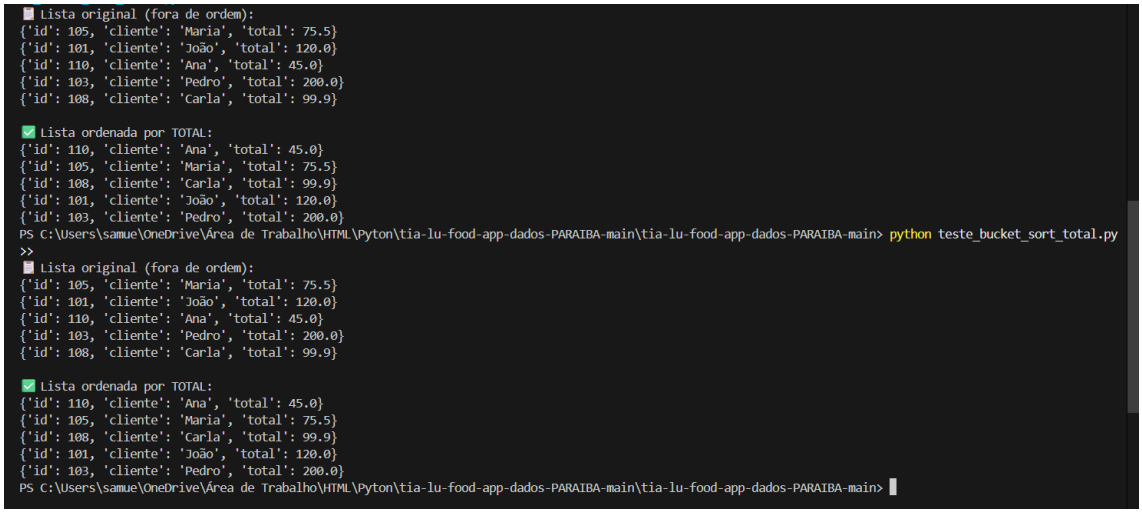


4.3 Comportamento do Bucket Sort no Sistema

O algoritmo foi testado em dois cenários principais: ordenação por **ID** e por **valor total do pedido**. Em ambos os casos, o comportamento mostrou-se estável e consistente, confirmando a eficiência da implementação.

Para fins de ilustração, optou-se por apresentar apenas a **Figura 2**, que demonstra a ordenação dos pedidos pelo valor total. Essa escolha se deve ao fato de que este critério evidencia de forma mais representativa a flexibilidade do Bucket Sort em lidar com diferentes necessidades de ordenação dentro da aplicação.

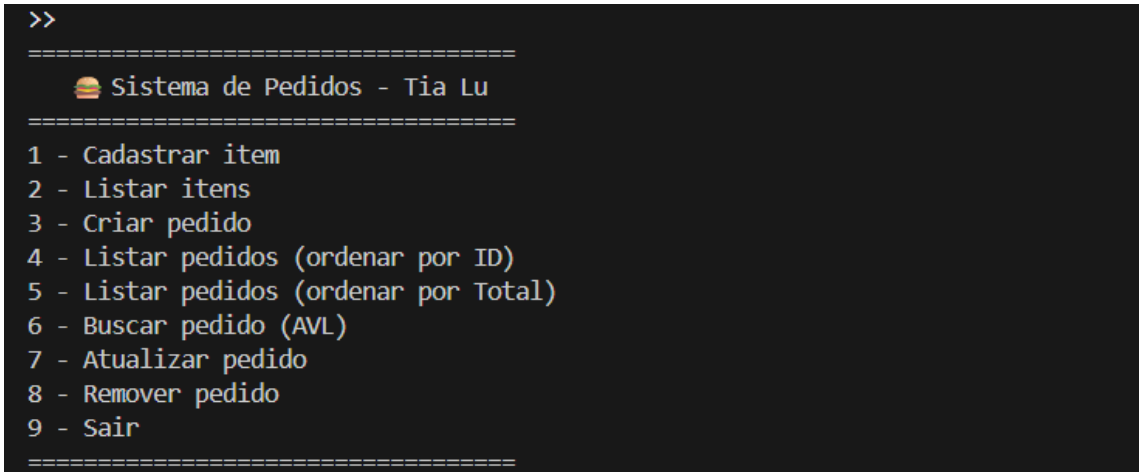
Figura 2. Lista de pedidos ordenados por valor total



Ao final, o sistema completo funcionou como esperado, entregando:

- Cadastro e listagem de itens
- Criação e manipulação de pedidos
- Ordenação eficiente
- Busca rápida pela AVL
- Persistência automática em JSON

Figura 3. Menu principal em execução



5. Considerações Finais

O desenvolvimento do projeto possibilitou uma compreensão aprofundada sobre algoritmos de ordenação, com destaque para o **Bucket Sort**. A experiência prática evidenciou a relevância desse algoritmo no processamento eficiente de informações, especialmente ao lidar com diferentes critérios de ordenação de pedidos.

Entre os principais desafios, destacou-se a **adaptação do Bucket Sort** para funcionar com múltiplos tipos de chave dentro do sistema, garantindo flexibilidade e consistência nos resultados. Apesar dessa dificuldade, a implementação foi concluída com sucesso, apresentando desempenho satisfatório e integração adequada ao fluxo da aplicação.

Como possíveis aprimoramentos futuros, seria interessante explorar **testes automatizados** para validar o comportamento do Bucket Sort em diferentes cenários, além de investigar alternativas de otimização para conjuntos de dados maiores.

Em síntese, o projeto cumpriu seus objetivos e permitiu consolidar, de forma prática, os conhecimentos teóricos sobre algoritmos de ordenação, demonstrando a eficácia do Bucket Sort no contexto proposto.

6. References:

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford.

Algoritmos: Teoria e Prática. 3. ed. Rio de Janeiro: Elsevier, 2012.

Capítulo 2 e 3: introdução à necessidade de algoritmos eficientes.

Capítulo 8: importância da ordenação em sistemas.

SEDGEWICK, Robert; WAYNE, Kevin.

Algorithms. 4. ed. Boston: Addison-Wesley, 2011.

UNIVERSIDADE FEDERAL DE MINAS GERAIS. Algoritmos de Ordenação – Aula 9: Bucket Sort. DCC/UFMG, 2020.

KNUTH, Donald E.

The Art of Computer Programming – Volume 3: Sorting and Searching.

Reading: Addison-Wesley, 1998.