FoodDelivery com Selection Sort e Maps

Douglas P. Santos[1], Lara N. Silva[2], Graciano Marôto[3].

***Centro Universitário de Excelência (Unex)* – Sistemas de informação – Vitória da Conquista – Bahia – Brasil**

{ engdouglasps@gmail.com, nuneslarasilva@hotmail.com e gracianoml07@gmail.com.}

Resumo

O trabalho analisa estruturas de dados, com foco na implementação do selection sort e maps. Foi desenvolvido em Python um sistema de gerenciamento de pedidos e produtos, usando funções e menus no terminal. O sistema possibilita cadastrar, atualizar e cancelar pedidos, aplicar descontos e mudar status, armazenando tudo em listas de dicionários em um arquivo json que é inicializado juntamente com o sistema. Os resultados foram positivos, apesar de desafios como a leitura e escrita do arquivo externo e o tratamento de erros. Para o futuro, sugere-se criar interface gráfica e gestão de usuários.

Abstract

The work analyzes data structures, focusing on the implementation of selection sort and maps. An order and product management system was developed in Python, using functions and menus in the terminal. The system makes it possible to register, update and cancel orders, apply discounts and change status, storing everything in dictionary lists in a json file that is initialized together with the system. The results were positive, despite challenges such as reading and writing the external file and error handling. For the future, it is suggested to create a graphical interface and user management.

1. Introdução

Na área de computação, organizar dados é uma tarefa fundamental. A **ordenação** permite que a gente encontre informações mais rápido, faça análises eficientes e melhore o processamento.

O **Selection Sort** é um algorítimo de ordemação simples, que tem como médoto principal dividir a lista em duas partes, uma ordenada e o restante da lista. Esse método pode ser ineficiente para grande base de dados por possuir complexidade de tempo $O(n^2)$, porém é constante e bastante didático, mantendo o uso de memória "in-place" e eficiente em swaps.

1. Fundamentação Teórica

Selection Sort: Trata-se de um método que tem uma lógica muito direta: ele foca em

encontrar o menor item de uma lista e colocá-lo na posição correta, repetindo esse processo até que a lista inteira esteja organizada. Ele procura o menor elemento na lista inteira. Troca esse menor elemento pela primeira posição da lista, a partir desse momento o primeiro elemento é ignorado (já que agora está correto) e procura o menor elemento na parte que sobrou da lista, ao encrontar o segundo menor elemento é feita a troca pela segunda posição da lista, passando a ignora-lo juntamente com o primeiro e olhando para os proximos, repetindo o processo até o fim da lista.

Exemplo Intuitivo:

Pense que você está organizando um monte de cartas de baralho espalhadas na mesa:

- Você olha para **todas** as cartas e pega a menor (o oito, por exemplo).
- Coloca essa carta na **primeira posição** à esquerda.
- Agora, você olha **só** para as cartas que **sobraram**, pega a menor delas e a coloca na **segunda posição**.
- Você repete isso até não sobrar mais nenhuma carta fora do lugar.

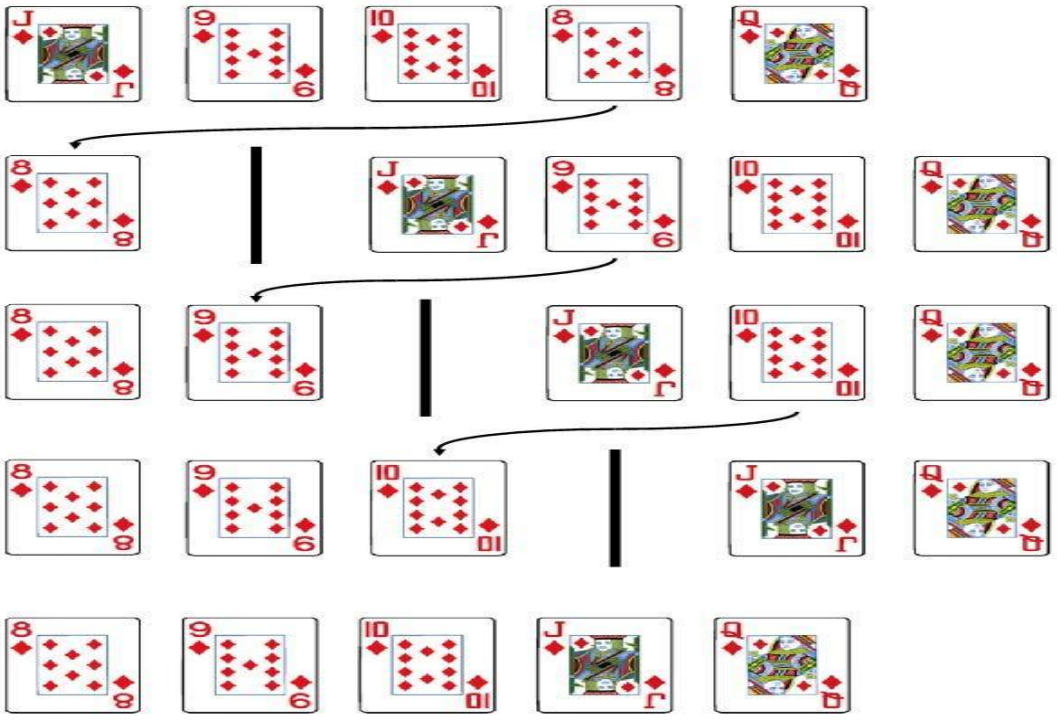

Figure 1. Cards ordering

Complexidade de Tempo:

O Selection Sort é conhecido por ser previsível em sua performance, pois é uma constante independente do tamanho do arquivo $O(n^2)$.
Isso significa que ele **sempre fará a mesma quantidade de comparações**, não importa se a lista já estiver quase ordenada ou totalmente desordenada. Ele tem que "selecionar" o menor item em cada passagem.

2. Implementação Prática

Para demonstrar a aplicação do Selection Sort, usamos uma função em Python que representa exatamente essa lógica de busca e troca.

A função que utilizamos percorre uma lista de objetos e organiza os elementos com base em um campo específico, que chamamos de "code".

```
1   get_things_sorted(things):
2   n = len(things)
3   for i in range(n - 1):
4       # Assume que o elemento atual é o menor
5       menor = i
6       # Percorre o restante da lista para encontrar o verdadeiro menor
7       for j in range(i + 1, n):
8           # Compara o campo 'code' para ver qual é o menor
9           if things[j]["code"] < things[menor]["code"]:
10              menor = j
11
12      # Se encontrou um item menor, faz a troca
13      if menor != i:
14          things[i], things[menor] = things[menor], things[i]
15
16  return things
17
18
```

Figure 2: Algoritm’s sort function

3. Análise e Resultados

Essa função foi criada e pôde ser utilizada em vários locais do código, além de ter muitas outras possíveis utilizações que ainda não aplicamos. Na prática, a inserimos na ordenação de Clientes, Vendas e Itens por código, já que utilizamos códigos aleatórios de 3 dígitos, podemos ordená-los de forma crescente ou decrescente utilizando o **selection sort**. Isso pode facilitar na geração de relatórios nas três entidades.

3.1 Exemplo de uma seleção na prática

Vamos supor que temos uma lista de números (códigos) com os seguintes números: 5, 3, 1, 4, 2]. E queremos ordená-la de forma crescente, que seria: [1,2,3,4,5], utilizaremos a lógica do **selection sort** para fazer essa ordenação.

3.2 Execução Passo a Passo (Visual)

Table 1: Step-by-step

Passagem	Explicação	Resultado da Lista
1ª	Procura o menor elemento **(1)** na lista toda. Troca o **1** (posição 3) com o **5** (posição 1).	[**1**, 3, 5, 4, 2]
2ª	Ignora o **1**. Procura o menor elemento **(2)** no restante da lista. Troca o **2** (posição 5) com o **3** (posição 2).	[1, **2**, 5, 4, 3]
3ª	Ignora **1** e **2**. Procura o menor elemento **(3)** no restante. Troca o **3** (posição 5) com o **5** (posição 3).	[1, 2, **3**, 4, 5]
4ª	Ignora **1, 2, 3**. Procura o menor elemento **(4)** no restante. Ele já está no lugar certo, então não há troca. O 5 também já está na posição certa.	[1, 2, 3, **4**, **5**]

3.3 Lista Final (Depois da Ordenação)

O resultado é que a lista ficará ordenada da seguinte forma: [1, 2, 3, 4, 5]. Essa é a função do selection sort e pode ser utilizada de diversas formas e economizar muito código e muito tempo de desenvolvimento.

Interpretação:

O algoritmo **funcionou perfeitamente**, seguindo a lógica da **seleção** e **troca**. Ele foi direto ao ponto: comparou item por item e fez as trocas somente quando encontrou o menor valor para a posição correta, resultando na lista totalmente ordenada, do menor para o maior. E quando o número atual já estava na posição certa, ele não fez troca.

4. Conclusões

Durante o desenvolvimento do projeto, encontramos desafios interessantes que tornaram a experiência enriquecedora, implementar um algoritmo de ordenação em uma aplicação já existente exigiu por o em prova a rigidez do código feito anteriormente, mudar a estrutura principal de classes para funções e aplicar listas de dicionários em um arquivo json a parte foi enssencial para o crescimento da ideia de um código sustentável.

O **Selection Sort** é uma ferramenta de aprendizado excelente. Por ser um algoritmo de ordenação simples, é ideal para quem está começando a entender os conceitos de laços de repetição, lógica de controle de fluxo e manipulação de arrays.

Apesar de ser ótimo para fins didáticos, é importante lembrar que ele não é a escolha mais eficiente para trabalhar com listas muito grandes em **projetos de produção**.
Nesses casos, a recomendação é usar algoritmos mais rápidos, como o **Merge Sort**, **Quick Sort** ou o **TimSort** (que é o algoritmo padrão e altamente otimizado do Python).

5. Referências

- CORMEN, T. H. Algoritmos: Teoria e Prática.
- GOODRICH, M.; TAMASSIA, R. Estruturas de Dados e Algoritmos em Python.
- Documentação oficial do Python: https://docs.python
- Photon-Lines Substack: https://photonlines.substack.com/p/visual-focused-algorithms-cheat-sheet