Relatório de Desenvolvimento

Gabriela Dellamora Paim*

16 de Abril de 2024

Resumo

Este artigo descreve o processo para solucionar o primeiro problema proposto na disciplina de Algoritmo e Estrutura de Dados II no 3° (terceiro) semestre: Siga o Dinheiro. O problema propõe que os alunos construam um algoritmo que percorra um mapa, coletando valores (dinheiro), somando o valor total e listando em ordem de coleta os valores coletados.

Introdução

O desafio "Siga o Dinheiro", apresentado no primeiro trabalho da disciplina de Algoritmo e Estrutura de Dados II, narra um assalto a um banco, seguido por uma perseguição policial. Durante a fuga, os criminosos deixaram trilhas de dinheiro que foram coletadas pela polícia. Em um determinado momento, os policiais capturaram os criminosos, encerrando a trilha. Agora, a perícia solicita a colaboração dos alunos para desenvolver um script que calcule o montante total de dinheiro deixado pelos criminosos nas ruas da cidade. Este documento foi escrito durante a construção da solução descrita no artigo, portanto, a ordem dos tópicos segue a ordem de desenvolvimento.

Primeiras Impressões e Análises

Durante a análise, foram observados obstáculos intrínsecos do desafio "Siga o Dinheiro":

1. Como se locomover pelo mapa?
2. Como identificar notas que já foram coletadas em cruzamentos?
3. Como identificar curvas e qual direção deve ser seguida?
4. Como identificar valores? Como decidir a direção que os valores serão lidos?

*g.paim02@edu.pucrs.br

Locomoção Pelo Mapa

Após deliberar sobre os obstáculos citados acima, foi decidido observar o problema de forma matricial, já que o problema poderia ser reescrito como uma iteração e coleta de dados de uma matriz. Levando em consideração que a iteração não poderia ser feita apenas iterando linhas ou apenas colunas, foi decidido utilizar constantes legíveis que definissem a direção a partir da soma dos índices, assim, a direção poderia ser alterada quando necessário durante a iteração. Outro lado positivo dessa abordagem é que, além de intuitiva, também é otimizada, tendo **Complexidade de Tempo linear e Complexidade Espacial constante**. Foi decidido, então, fazer uma verificação a cada iteração, caso o elemento na posição atual fosse uma curva, os elementos ao redor da posição atual seriam verificados para decidir o próximo valor da variável lista "direcao", que poderia ter valor 1 ou -1 para percorrer horizontalmente ou verticalmente (índice x e y).

Listing 1: Constantes para Direções

```
CURVE             =   ['\\', '/']
POSITIONAL_ERROR  =   [0,  0]
UPWARDS           =   [-1, 0]
DOWNWARDS         =   [1,  0]
LEFT              =   [0, -1]
RIGHT             =   [0,  1]
```

Assim, já foi possível construir o *script* para percorrer todo o mapa, já desobstruindo alguns dos obstáculos definidos: Locomoção pelo mapa, identificação de curvas e definição de direção.

Listing 2: Script que Percorre o Caminho do Mapa

```
def travel_through_matrix(matrix: list) -> list:
    '''
    Travel through the matrix and return the total value of the money
        and a list with the money in the order they were found
    Time Complexity: O(n)
    Space Complexity: O(1)
    '''
    # Verify first direction
    direction = INITIAL_POSITION
    if matrix[0][0] == HORIZONTAL_STREET or matrix[0][0].isdigit():
        direction = RIGHT
    elif matrix[0][0] == VERTICAL_STREET or matrix[0][0].isdigit():
        direction = DOWNWARDS

    # Travel through the matrix
    list_money = []
    current_position = INITIAL_POSITION
    try:
```

```
        while matrix[current_position[0]][current_position[1]] !=
            DEAD_END:
            # If the current position is a corner, then change the
                direction
            if matrix[current_position[0]][current_position[1]] in ['\\',
                '/']:
                direction = change_direction(matrix, current_position,
                    direction)

            # Update the current position to the next position
            current_position[0] += direction[0]
            current_position[1] += direction[1]
    except:
        print("Erro! A execucao nao foi finalizada corretamente.")
    return list_money
```

Verificação dos Valores

O primeiro passo para verificação dos valores é verificar a existência de dígitos na matriz. Isso é fácil pois utilizamos a notação decimal neste problema, portanto poderíamos verificar se o elemento atual está dentro de uma lista de dígitos, contendo os dígitos de 0 até 9 ou apenas verificar se o elemento atual é um dígito utilizando a função ".isdigit()" da linguagem Python3. Se sim, podemos iterar os próximos elementos com um módulo que faça essa verificação.

É interessante observar que, mesmo que haja valores em ruas que se intersecionam, não existe valores em curvas, portanto podemos finalizar a verificação caso uma rua seja verificada ou caso uma curva seja detectada.

Listing 3: Iteração pelo mapa

```
def travel_direction(matrix:list, coordinates:list, direction:list) ->
    tuple:
    '''
    Travel through the matrix and return the total value of the money
        and a list with the money in the order they were found
    Time Complexity: O(n)
    Space Complexity: O(1)
    '''
    # Travel through the matrix
    money_list = []
    x, y = coordinates
    while (matrix[x][y] not in CURVE):
        # print(f"| Current Element: {matrix[x][y]} | Current Position:
            {[x,y]} | Current Direction: {direction}|")
        if matrix[x][y].isdigit():
            print("[VERBOSE] DIGIT FOUND!")
            money, [x,y] = verify_value(matrix, [x, y], direction)
```

```
            money_list.append(money)
            continue

        # Update the current position to the next position
        x += direction[0]
        y += direction[1]

        # If error or deadend
        if(matrix[x][y] in [DEAD_END, SPACE]):
            print(f"[VERBOSE] Dead End or SPACE Found! {matrix[x][y]}")
            break
    return (money_list, [x,y], direction)
```

Aprimoramento do Script

Após as implementações acima, o script agora é capaz de percorrer o mapa corretamente, além de coletar os valores na ordem correta e sem repetições. Neste momento, foi decidido aprimorar o script, utilizando conceitos de *Object Oriented Programming*.

Para aprimorar legibilidade, foram criados três objetos: Atlas, Hermes e Pandora. O objeto Atlas contém o mapa, matriz e direções. O objeto Hermes será nosso iterador. Ele possui métodos para percorrer o objeto Atlas, coletando os valores e informações relevantes. Além disso, o objeto Hermes possui Pandora, que contém métodos para armazenar as cédulas coletadas e a lista de cédulas.

Após essa alteração, o desenvolvimento ficou bem intuitivo: no loop inicial, enquanto o policial Hermes não chegar ao final do mapa (DEAD-END) ou enquanto não bater em uma parede (WALL), o policial andará em linha reta até encontrar uma curva (segundo loop), coletando todas cédulas do caminho. Quando o policial encontrar uma curva, ele alterará sua direção e repetirá o loop inicial.

É importante informar que, utilizar do paradigma "OOP" para gerar uma resolução **afeta** o tempo de execução (*Object Oriented Programming* não é referência em tempo de execução, uso de memória otimizado e códigos performáticos em geral), porém decidiu-se utilizar esse paradigma como um método de abstração, facilitando análise do script, com intuito de democratizar a leitura do código, assim, não dependendo do conhecimento em Python do leitor.

BugFix

Durante os testes de desenvolvimento, percebeu-se um erro na definição do problema no script durante a identificação da primeira posição no mapa. Foi construído um método do objeto Atlas para calcular a primeira posição, retornando o índice do único elemento HORIZONTAL-STREET, dado que os únicos

elementos possíveis na primeira coluna da matriz são: início do mapa, curva e rua vertical. Tendo essa limitação, é possível capturar o único HORIZONTAL-STREET do primeiro vetor para definir o início do mapa.

Além de minor fixes, houve problema na decisão de direção. Para evitar não capturar casos na decisão de direção, foi decidido verificar todas as opções possíveis. Assim, por mais que seja um trecho de código extremamente redundante, não abre espaço para casos problemáticos na decisão de direção.

Contagem de Operações

Antes da contagem de operações, é necessário entender qual o fluxo do algoritmo, portanto, o fluxo será descrito. Além disso, para contagem de operações, serão ignorados métodos de bugfix.

Explicação de Fluxo

Inicialmente criamos o objeto Hermes como police e parametrizamos suas flags de debug e repetição nos cruzamentos. Essa parametrização é feita alterando os valores das constantes "DEBUG" e "REPETIR-NOS-CRUZAMENTOS". A parametrização padrão entrega os valores encontrados pelo professor nos casos de teste.

Após isso, é iniciado o primeiro *While* que mantém repetição enquanto o objeto police não encontrar o final do percurso ou enquanto encontrar uma parede. Logo, as condições de parada são finalizar o percurso com sucesso ou falhar em algum momento. Dentro deste *while*, é executada a primeira sub-rotina, que percorre uma linha reta. A sub-rotina executa um laço de repetição enquanto não encontrar uma curva, verificando se o valor de cada elemento iterado é um inteiro. Se sim, uma sub-rotina para coleta desses valores é iniciada, fazendo outro laço de repetição para coletar esses valores com os métodos de Pandora. É importante notar que ambas sub-rotinas atualizam a a posição atual do policial, então elas não se somam, apenas são separadas, portanto podem ser consideradas o mesmo laço para a contagem de operações. Após sair da primeira sub-rotina, é feita uma verificação de direção e o *while* principal é executado novamente. Após a finalização do *while* principal, o programa armazena os valores coletados pelo algoritmo na mesma pasta que está armazenado o input.

Contagem de Operações Manual

A contagem de operações manual é executada levando em consideração o caso que se quer verificar. No caso dessa análise, será levado em conta o pior caso o possível, que seria iterar todos os elementos da matriz.

Assim, o caso considerado na contagem de operações manual será um mapa n x m (com n sendo a quantidade de colunas da matriz e m a quantidade de linhas). Assim, teremos que:

1. No pior caso, iteraremos n x m elementos na matriz;

2. Dentro do laço de repetição para percorrer a rua:

   (a) Temos uma sub-rotina para identificar se o elemento iterado é um número. No pior caso, temos um operador de verificação (if) com duas operações (uma verificação de igualdade e uma verificação de se o elemento é um dígito), portanto, nesta análise, temos 2 operações.

   (b) Se o elemento for um dígito, então temos uma sub-rotina para coletar esses valores. No pior caso, temos outro laço de repetição que percorrerá todos os elementos até encontrar uma curva ou o final da linha. Isso conta para a nossa contagem de operações, já que ele está dentro do laço principal. Dentro desse laço, temos operações de verificação de direção e verificação de finalização de iteração. Considerando que o número de iterações seja o máximo, temos que a complexidade do laço é linear e então o número de operações também é linear.

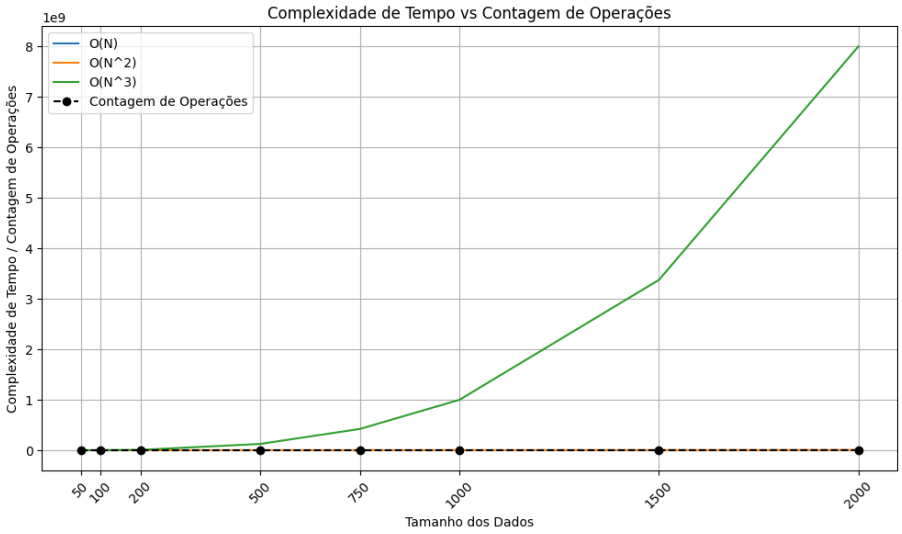
   (c) Ao finalizar o laço, fazemos uma verificação de direção, que é uma operação simples de verificação, portanto, no pior caso, temos uma operação de complexidade constante.

3. Considerando todos esses elementos, temos uma complexidade total de O(n x m) para a contagem de operações no pior caso.

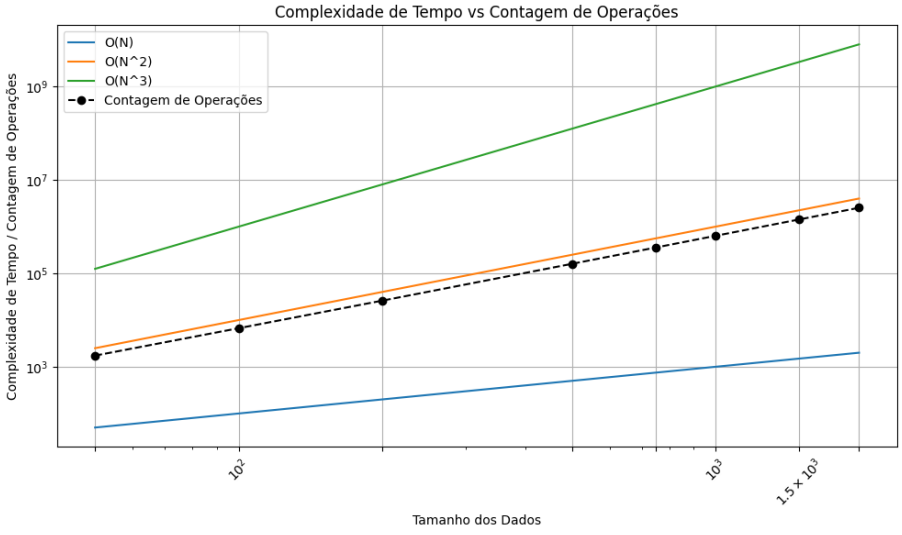
Contagem de Operações Gráfica

Para gerar a contagem de operações automática, foram acumuladas quantas operações são realizadas em *loops* no código. A partir dos valores observados, foi construída uma curva que representa a taxa de crescimento da complexidade temporal do algoritmo.

Após adicionar um contador de operações no código, foi gerado o gráfico de visualização comparativa entre as complexidades de tempo. O gráfico pode ser visto na Figura abaixo.



Observando a contagem de operações do algoritmo, é possível verificar que a tendência de crescimento segue a complexidade temporal de $O(NM)$ ou $O(N)$. Para visualizar melhor, foi aplicada uma visão logarítmica dessas operações.



Assim, verificamos que o algoritmo segue uma complexidade espacial de $O(NM)$, sendo N e M as dimensões da matriz lida. Como as matrizes dos casos de teste são quadradas, representamos a complexidade como $O(N^2)$.

Considerações Adicionais

Durante o desenvolvimento do algoritmo, algumas decisões foram tomadas visando otimização e legibilidade do código:

- **Utilização de Constantes**: Definição de constantes para caracteres especiais do mapa, facilitando a compreensão e manutenção do código;
- **Abordagem Matricial**: A adoção de uma abordagem matricial simplificou a lógica de percorrer o mapa, resultando em uma implementação mais intuitiva e otimizada;
- **Aprimoramento com OOP**: A introdução de conceitos de Orientação a Objetos melhorou a legibilidade e modularidade do código;
- **Refatoração Constante**: Identificação e correção de bugs, além de oportunidades de otimização, contribuíram para a eficiência do algoritmo;
- **Output em arquivo**: receber os valores importantes em arquivo facilitam a verificação dos valores de retorno;
- **Análise de Complexidade**: A contagem de operações forneceu *insights* sobre o desempenho do algoritmo em diferentes cenários.

Essas considerações destacam as escolhas cuidadosas feitas durante o desenvolvimento, garantindo a eficiência e qualidade do algoritmo final.

Conclusão

O artigo descreveu o processo de desenvolvimento de um algoritmo para resolver o problema proposto na disciplina de Algoritmos e Estruturas de Dados II. O algoritmo foi implementado em Python3, utilizando principalmente o paradigma procedural, com algumas adaptações em Orientação a Objetos para melhorar a legibilidade do código. O script foi projetado para percorrer um mapa em busca de valores monetários, somando-os e registrando sua ordem de coleta. A análise incluiu a identificação de obstáculos e a formulação de soluções para cada um deles, bem como uma contagem de operações para avaliar a eficiência do algoritmo. No geral, o desenvolvimento do algoritmo foi bem-sucedido e atendeu aos requisitos do problema proposto.
Para visualizar as versões de desenvolvimento, acesse o releases no GitHub.