# Relatorio Trabalho 1 Algoritmos e Estrutura de Dados

Marcos Raach*
Faculdade de Informática — PUCRS

10 de abril de 2024

**Resumo**

*E*ste artigo descreve alternativas de solução para o primeiro trabalho proposto na disciplina de Algoritmos e Estrutura de dados 2 no semestre 01/2024, que trata de uma analise de rotas feitas por bandidos após realizarem um assalto a banco, onde os bandidos deixaram uma trilha de notas ao longo do seu trajeto de fuga. A polícia seguiu essa trilha para capturar os criminosos, recolhendo o dinheiro pelo caminho. Foi desenvolvido um programa em JavaScript para calcular a quantidade total de dinheiro recuperado ao final da perseguição.

## Introdução

Dentro do escopo do primeiro trabalho de Algoritmos e Estrutura de Dados II, surge a necessidade de investigar e resolver um problema, o rastreamento de bandidos em assaltos a banco bem-sucedidos. No presente trabalho, o problema a ser abordado pode ser divido da seguinte forma: após um grande assalto a banco, os criminosos deixaram uma trilha de notas pela cidade durante sua fuga. A polícia, então, necessita seguir essa trilha para efetuar a captura dos bandidos, recolhendo o dinheiro pelo caminho. O desafio é calcular a quantidade total de dinheiro recuperado ao final da perseguição, utilizando informações contidas nos mapas fornecidos pela equipe de perícia.

O objetivo deste relatório é apresentar uma análise detalhada do desenvolvimento de um programa em JavaScript para solucionar esse problema. Serão discutidos os passos adotados para a resolução, as estratégias implementadas e os resultados obtidos. Ao final, serão apresentadas conclusões sobre a eficácia da solução proposta e requer uma abordagem específica devido à complexidade da rota de fuga e à presença do dinheiro deixado pelos bandidos ao longo do percurso. Ao entendermos claramente o contexto e os requisitos do problema, estaremos preparados para seguir adiante e explorar as soluções propostas.

Depois de considerar o problema, tive uma primeira impressão que Python poderia ser a melhor alternativa para resolver o problema, visto que em aula a Professora Eduarda Monteiro falou bastante sobre a tal linguagem e mostrou diversos exemplos com a linguagem. Por não saber muito da linguagem comecei a estudar pelo menos o básico.

## Primeira solução

A primeira solução proposta para o problema de calcular a quantidade total de dinheiro recuperado pela polícia após a perseguição aos bandidos consistiu na implementação de um algoritmo em Python. O algoritmo foi desenvolvido com base em uma lógica simples, porém, como veremos a seguir, apresentou limitações em relação à abordagem adotada para realizar a soma dos valores das notas.

*m.raach@edu.pucrs.br

```
procedimento Algoritmo calcularTotalRecuperado(notas):
    total_recuperado ← 0

    para cada nota na lista de notas:
        se a nota for composta por mais de um dígito:
            Converter a nota  → string
            soma_dígitos ← 0
            para cada dígito na nota:
                Converter o dígito de string  → inteiro
                soma_dígitos ← soma_dígitos + dígito
            total_recuperado ← total_recuperado + soma_dígitos
        Senão:
            Adicionar a nota ao total_recuperado
    fim
    Retornar total_recuperado
```

Apesar da lógica fazer algum sentido, foi completamente esquecido os passos de transições, para virar a direita, esquerda, cima e baixo, então esse código era infinito, não chegou a funcionar, sendo esquecido também o simbolo # que sinalizava o final da rota de fuga dos bandidos.

# Segunda solução

A primeira solução proposta para o problema de calcular a quantidade total de dinheiro recuperado pela polícia após a perseguição aos bandidos consistiu em simular a rota de fuga dos criminosos através de um labirinto. No entanto, esta abordagem apresentou deficiências significativas devido à falta de consideração dos passos de transição para virar à direita, esquerda, cima e baixo, resultando em um código infinito e ineficaz. Além disso, o símbolo # que sinalizava o final da rota de fuga dos bandidos foi negligenciado, comprometendo ainda mais a funcionalidade do algoritmo.

```
procedimento simularLabirinto(labirinto):
    carro_posicao ← 0
    dinheiro_coletado ← []

    para cada linha na labirinto:
        para cada índice , simbolo na enumerate(linha):
            se simbolo for igual a '/':
                carro_posicao ←carro_posicao −1
            senão , se simbolo for igual a '\\':
                carro_posicao ← carro_posicao + 1
            senão , se simbolo for igual a '#':
                Escrever('Bandidos capturados!')
                contarDinheiro(dinheiro_coletado)
                retornar
            senão , se simbolo for um dígito:
                valor_dinheiro ← ''
                enquanto índice for menor que o comprimento da linha e linha[índice] for um dígito:
                    valor_dinheiro ← valor_dinheiro + linha[índice]
                    índice ← índice + 1
                adicionar int(valor_dinheiro) a dinheiro_coletado

    Escrever('Dinheiro coletado em ordem: ' , dinheiro_coletado)
```

```
23
24 procedimento contarDinheiro(dinheiro_coletado):
25     total_dinheiro ← somarElementos(dinheiro_coletado)
26     Escrever('Total de dinheiro coletado: R
```

O problema que encontrei após a professora disponibilizar os resultados esperados, foi que essa solução os números quando estavam na vertical, não eram contabilizados, como a minha experiencia em Python não era tao grande e eu já tinha tentado diversas formas, decidi me aventurar no TypeScript, onde já tenho um conhecimento melhor, por se tratar de uma linguagem que trato diariamente, nele tive alguns problemas de compilação e de tipagens, então para não ocorrer esses erros, fui forçado a usar o JavaScript.

# Terceira solução

Quando me deparei com o desafio de processar um arquivo de labirinto, a primeira ideia que me veio à mente foi usar TypeScript, já que estou familiarizado com essa linguagem e seu sistema de tipagem forte. No entanto, à medida que avançava na implementação, enfrentei alguns obstáculos relacionados à tipagem e à compilação. Isso me fez repensar minha abordagem.

Então, decidi optar por JavaScript, uma linguagem mais flexível e com a estou acostumado também. Apesar de encontrar alguns desafios, como a falta de tipagem forte, percebi que o JavaScript me oferecia uma maneira mais fluida de lidar com o problema.

Abaixo está o pseudocodigo que mostra a minha lógica para a resulação do problema. Isso me permitiu focar na lógica do algoritmo, em vez de perder tempo com as tipagens existentes.

```
1      função lerArquivo(nome , callback):
2      tempoInicial ← obterTempoAtual()
3
4      dados ← lerArquivoSincronamente(nome)
5
6      caracteresLinhas ← dividirLinhasEmCaracteres(dados)
7
8      tempoFinal ← obterTempoAtual()
9      tempoDecorrido ← calcularTempoDecorrido(tempoInicial , tempoFinal)
10
11     chamar callback passando caracteresLinhas e tempoDecorrido
12
13 função executarCaso(caracteresLinhas , tempoDecorrido):
14     linhaAtual ← 0
15     posiçãoAtual ← 0
16     somaParcial ← ""
17     somaTotal ← 0
18     direção ← 1
19
20     enquanto caracteresLinhas[linhaAtual][posiçãoAtual] não for igual a "─":
21         linhaAtual++
22
23     enquanto caracteresLinhas[linhaAtual][posiçãoAtual] não for igual a "#":
24         caso direção seja:
25             caso 1:
26                 se o caractere atual for um número:
27                     enquanto o caractere atual for um número:
```

```
                    somaParcial ← somaParcial + caracteresLinhas[linhaAtual][posiçãoAtual]
                    posiçãoAtual incrementa
                somaTotal ← somaTotal + converterParaInteiro(somaParcial)
                somaParcial ← ""
            se caracteresLinhas[linhaAtual][posiçãoAtual] for igual a "\\":
                direção ← 4
                linhaAtual incrementa
            senão , se caracteresLinhas[linhaAtual][posiçãoAtual] for igual a "/":
                direção ← 3
                linhaAtual decrementa
            senão:
                posiçãoAtual incrementa
        caso 2:
            se o caractere atual for um número:
                enquanto o caractere atual for um número:
                    somaParcial ← somaParcial + caracteresLinhas[linhaAtual][posiçãoAtual]
                    posiçãoAtual decrementa
                somaTotal ← somaTotal + converterParaInteiro(somaParcial)
                somaParcial ← ""
            se caracteresLinhas[linhaAtual][posiçãoAtual] for igual a "\\":
                direção ← 3
                linhaAtual decrementa
            senão , se caracteresLinhas[linhaAtual][posiçãoAtual] for igual a "/":
                direção ← 4
                linhaAtual incrementa
            senão:
                posiçãoAtual decrementa
        caso 3:
            se o caractere atual for um número:
                enquanto o caractere atual for um número:
                    somaParcial ← somaParcial + caracteresLinhas[linhaAtual][posiçãoAtual]
                    linhaAtual decrementa
                somaTotal ← somaTotal + converterParaInteiro(somaParcial)
                somaParcial ← ""
            se caracteresLinhas[linhaAtual][posiçãoAtual] for igual a "\\":
                direção ← 2
                posiçãoAtual decrementa
            senão , se caracteresLinhas[linhaAtual][posiçãoAtual] for igual a "/":
                direção ← 1
                posiçãoAtual incrementa
            senão:
                linhaAtual decrementa
        caso 4:
            se o caractere atual for um número:
                enquanto o caractere atual for um número:
                    somaParcial ← somaParcial + caracteresLinhas[linhaAtual][posiçãoAtual]
                    linhaAtual incrementa
                somaTotal ← somaTotal + converterParaInteiro(somaParcial)
                somaParcial ← ""
            se caracteresLinhas[linhaAtual][posiçãoAtual] for igual a "\\":
```

```
                    direção ← 1
                    posiçãoAtual incrementa
                senão , se caracteresLinhas[linhaAtual][posiçãoAtual] for igual a "/":
                    direção ← 2
                    posiçãoAtual decrementa
                senão:
                    linhaAtual incrementa

    Escrever("Soma total = " , somaTotal)
    Escrever("Tempo de leitura do arquivo: " , tempoDecorrido arredondado para 2 casas decimais , " segu

lerArquivo("labirinto2000.txt" , executarCaso)
```

Neste processo, aprendi que a escolha da linguagem certa pode fazer toda a diferença na resolução de um problema, e JavaScript se mostrou a melhor opção para mim neste caso.

# Resultados

Depois de implementar o algoritmo acima em linguagem JavaScript e executá-lo obtivemos os seguintes resultados:

Tabela 1: Resultados e Tempos de Execução para Diferentes Labirintos

| **Labirinto** | **Resultado** | **Tempo de Execução (segundos)** |
|---|---|---|
| labirinto50.txt | 1919 | 0.007 |
| labirinto100.txt | 11458 | 0.007 |
| labirinto200.txt | 42459 | 0.008 |
| labirinto500.txt | 367423 | 0.021 |
| labirinto750.txt | 1841723 | 0.039 |
| labirinto1000.txt | 2273541 | 0.067 |
| labirinto1500.txt | 5800219 | 0.125 |
| labirinto2000.txt | 10707187 | 0.327 |

# Conclusões

Este trabalho explorou a complexidade de calcular a quantidade total de dinheiro recuperado após a perseguição aos bandidos em uma trilha de notas deixadas durante sua fuga. Ao longo do processo, experimentei diferentes abordagens, desde implementações falhas em Python até a solução prevista mas em outra linguagem, o JavaScript.

Enfrentei desafios significativos ao lidar com a interpretação das notas dispostas verticalmente e a manipulação adequada dos passos de transição. Principalmente em Python, onde não entendi como funcionava perfeitamente as concatenações.

Os resultados obtidos foram bons, demonstrando a eficácia do algoritmo desenvolvido na recuperação do dinheiro em diversos cenários de labirintos. Obviamente, o tempo de execução foi aumentando conforme o tamanho do labirinto, a solução se mostrou capaz de lidar com labirintos de grande escala dentro de limites aceitáveis.

Este trabalho me proporcionou a perceber a importância da escolha da linguagem de programação pois tentei algo novo e não funcionou como o planejado então tive que ir para outra linguagem que eu domino mais, assim como a necessidade de uma abordagem iterativa e adaptativa na resolução

de problemas computacionais. Existem formas mais eficazes para a solução desse problema mas conforme os resultados, o algoritmo se comportou bem. Existe muito código copiado e colado dentro dos casos de direção, tentei usar outras formas após a verificação dos resultados mas não obtive muito sucesso, então o deixei assim.

Em última análise, esta experiência não apenas ampliou meu entendimento sobre algoritmos e estruturas de dados, mas também reforçou a importância do conhecimento em linguagens. Pois apesar de não utilizar Python na minha solução final, foi um aprendizado e tanto.