Algoritmos e Estruturas de Dados II

Trabalho 2 (Mapa dos Fenícios)

Lara Alves Kunrath Padilha, Leonardo Garcia Forquim Bertinatto

PUCRS

# 1. Introdução

# Os Fenícios estão voltando a realizar viagens por todos os mares, e querem utilizar das últimas tecnologias disponíveis, tendo como objetivo diminuir o tempo das viagens e reduzir a pegada de carbono, utilizando o mínimo de combustível possível para realizar todos os percursos previstos nos mapas. Eles declararam que, as vezes, é possível que algum mapa esteja errado, colocado um porto em um local que não é possível chegar de barco, então eles gostariam que esse porto fosse removido da rota de viagem, evitando assim dias de viagem e gasto de combustível desnecessários.

# 2. O problema

A partir dos pedidos dos Fenícios nossos problemas eram os seguintes:

- Ler diferentes mapas e processá-los da mesma maneira;

- Passar os mapas para Grafos;

- Percorrer os portos pelo menor caminho possível;

- Calcular o mínimo de combustível para as viagens.

# 3. Processo de solução

O processo de solução inicial possuía diferentes ArrayLists que armazenavam todas as informações fornecidas nos casos de teste, mas ao final disso vimos que o código acabava demorando muito tempo para concluir a execução, podendo levar horas. Com isso, se tornou um quesito para nós melhorar a eficiência do código, visando economizar o tempo do antropólogo ao tentar simular um jogo.

Para isso, optamos por diminuir a quantidade de informação que estávamos salvando, e levamos em consideração apenas o número de rodadas pedido (salvo como contador de partidas), o número de cocos pares e impares de cada macacos e quais eram os destinatários dos respectivos cocos, estes últimos foram salvos em diferentes vetores na classe “Jogo”. Então, tratamos apenas de relacionar os cocos de um macaquinho com o seus respectivos destinatários pares e impares, o tratamento foi realizado na classe “Macaquinho”.

Agora para descobrimos o vencedor, foi feita uma variação da estrutura de busca Bubble Sort, para que encontrássemos o índice do macaco vencedor salvo no ArrayList de macaquinhos, comparando a quantidade de cocos entre eles.

Por fim, para obtermos uma análise mais profunda dos algoritmos utilizados, foi adicionado um contador de tempo, visando saber o tempo de execução de cada caso de teste fornecido de acordo com os diferentes algoritmos que estávamos testando, afim de tentar otimizar o máximo possível o tempo de execução

# 4. Evidências de que o problema foi resolvido

Os resultados esperados eram os seguintes:

5 x 10 | o custo total da viagem é de 304 unidades de combustível

15 x 80 | o custo total da viagem é de 304 unidades de combustível

30 x 80 | o custo total da viagem é de 432 unidades de combustível

50 x 100 | o custo total da viagem é de 484 unidades de combustível

60 x 500 | o custo total da viagem é de 986 unidades de combustível

500 x 1000 | o custo total da viagem é de 3426 unidades de combustível

Os resultados obtidos foram os seguintes:

**Caso de teste 1**

**Caso de teste 2**

**Caso de teste 3**

**Caso de teste 4**

**Caso de teste 5**

**Caso de teste 6**

# 5. Conclusão

Na resolução do problema proposto, nos deparamos primeiramente com a definição de como iríamos armazenar as informações e fazer com que elas estivessem ligadas ao componente principal que é o macaquinho. Para isso, foram utilizadas estruturas de dados ArrayList e vetores. A construção do código é simples, mas descobrimos que diferentes formas de resolver o problema podem afetar a eficiência da execução, podendo levar horas para executar casos de teste muito grandes, a partir disso nos focamos em otimizar a execução para todos os casos de teste, e descobrimos que não era preciso salvar todas as informações como, quantidade de macaquinhos, quantidade de cocos de cada macaquinho, quantidade de pedrinha de cada macaquinho, etc, ao invés disso, salvamos apenas a quantidade cocos impares e pares, e qual o destinatário de cada um, diminuindo assim a quantidade de informações que precisavam ser processadas e calculadas, otimizando o tempo de execução.

Por fim, mesmo que a construção da atividade fosse simples, serviu para pensarmos em diferentes formas de melhorar um código, mesmo que o quesito eficiência não seja avaliado, nos sentimos desafiados a entregar algo melhor que o pedido.