A Viagem dos Fenícios

Artur Wiederkehr*
Escola Politécnica – PUCRS

29 de maio de 2023

Resumo

Este artigo discorre sobre o Trabalho 2 da disciplina Algoritmos e Estruturas de Dados 2 que apresenta um problema de navegação pertencente aos fenícios, estes pretendem voltar a velejar e precisam saber qual o menor caminho para cada uma de suas viagens.

Em posse dos mapas e das informações disponibilizadas pelos próprios, investigaremos o cenário e confeccionaremos um algoritmo capaz de solucionar o problema, ressaltando seu comportamento, sua eficiência e possíveis melhorias.

Introdução

Os fenícios pretendem voltar a realizar comércio com seus trirremes, eles concedem alguns de seus mapas em arquivos texto para que possamos conferi-los. As viagens são planejadas iniciando-se no porto 1, passando por todos os outros portos de 2 a 9 necessariamente nesta ordem e retornando ao primeiro porto. Com o intuito de maximizar seu lucro, eles gostariam de fazer a menor viagem possível para economizar combustível, então são apresentadas algumas informações para definir o cenário:

- Os trirremes só podem se mover nas direções norte, sul, leste e oeste.
- Cada locomoção consome uma unidade de combustível.
- Os portos aparecem enumerados nos mapas.
- Nos mapas, os pontos representam água navegável e os asteriscos representam um local não-navegável.
- Eventualmente alguns portos podem estar inacessíveis, quando isso ocorrer, este local deve ser desconsiderado e a viagem deve prosseguir normalmente.

A primeira linha de cada arquivo possui as dimensões do mapa, como demonstrado no exemplo abaixo.

```
8 20
....************...
.....**.....*2.....
.**.*9*.............
.**..*..*1*.........
..8.....*....*.....
....**.6..5...*3....
........**..........
....7...**..4.......
```

Faremos um programa que lerá os mapas e, através de um algoritmo, nos informará qual o mínimo de combustível necessário para realizar a viagem planejada pelos fenícios respeitando as regras informadas.

*:artur.wiederkehr@edu.pucrs.br

Resolução

Ao ler o arquivo, elemento por elemento, há algumas ações a serem realizadas para a implementação da resolução: reserva-se as dimensões do mapa que serão referenciadas como latitude e longitude respectivamente, reserva-se também as coordenadas dos 9 portos em uma lista ordenada de forma crescente (de 1 a 9 nos índices de 0 a 8), cada coordenada é uma lista composta por latitude e longitude, obtidas através de contadores; traduziremos o mapa enquanto o guardamos em uma matriz, quando o elemento lido for um ponto não-navegável passa-se o valor *None* para aquela coordenada, do contrário passa-se *False*.

Antes de iniciarmos as viagens entre portos, instancia-se uma cópia do mapa para que ele retorne a sua forma original após cada viagem, é necessário realizar um *Deep copy*, ou seja, criar outro mapa igual e não referenciar o mesmo.

As viagens entre portos são realizadas por um método chamado por um laço, um *if else* é suficiente para pular um porto caso este esteja inacessível. A fim de selecionar o porto de destino, usa-se o contador do laço + 1 agrupado por resto de 9 ((i+1)%9), usar o operador de resto foi a forma mais fácil de tratar o problema do porto número 9 estar indisponível sem necessitar de verificações adicionais ou colocar o porto 1 no final da lista para a última viagem.

Adota-se o caminhamento em largura como estratégia para atingirmos nosso objetivo através de dois métodos: o "viajar" responsável pela lógica e cálculo da distância e o "pegar vizinhos" que verifica as coordenadas disponíveis para que o caminhamento prossiga.

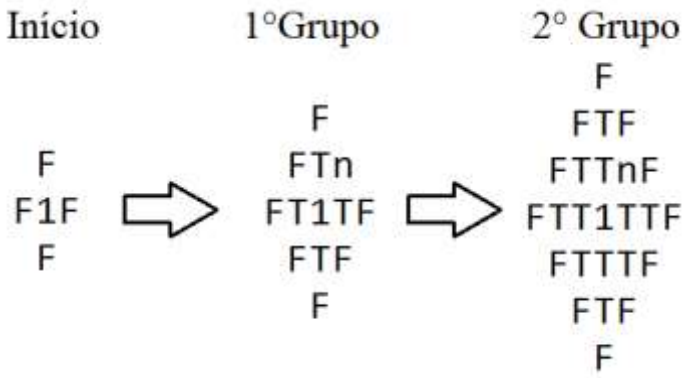
O método viajar recebe 2 coordenadas e o mapa, ele contará grupos de vizinhos ao invés de movimentos, inicialmente a fila 1 recebe o porto de partida, remove-se os elementos da fila 1 unitariamente e então coloca os vizinhos deles na fila 2, esta também remove seus elementos unitariamente e coloca os vizinhos dos vizinhos do porto de partida de volta na primeira fila, as filas recebem grupos de vizinhos alternadamente e sucessivamente até que o porto de destino possua valor *True* ou até que todo o mapa tenha sido explorado. Abaixo o pseudocódigo:

```
def viajar(início, fim, mapa):
        fila 1 adiciona início
        enquanto fila 1 ou fila 2 possuírem coordenadas:
                enquanto fila 1 possuir coordenadas:
                        atual = remove um fila 1
                        pegar vizinhos(atual, fila 2, mapa)
                se fim é verdadeiro:
                        retorna distância + 1
                enquanto fila 2 possuir coordenadas:
                        atual = remove um fila 2
                        pegar vizinhos(atual, fila 1, mapa)
                se fim é verdadeiro:
                        retorna distância + 2
                distância = distância + 2
        retorna 0
```

O método “pegar vizinhos” recebe uma coordenada, uma fila e o mapa, há 4 *ifs* que verificam se os vizinhos da coordenada atual estão dentro do mapa usando a latitude, a longitude e o 0 como limites. Para cada coordenada disponível, se for um lugar navegável, altera-se o valor no mapa para *True* e coloca na fila passada por parâmetro, é graças a está alteração que as filas não recebem elementos repetidos, os *ifs* são separados propositalmente de tal forma que evitemos uma exceção por perguntar o valor de um elemento inexistente. Abaixo apenas uma parte do pseudocódigo, pois o processo de uma direção é similar para as demais:

```
def pegar vizinhos(atual, fila, mapa):
    se vizinho norte existe:
        se vizinho norte é falso:
            vizinho norte no mapa = verdadeiro
            fila adiciona vizinho norte
    se vizinho sul existe:
    [...]
```

Uma ilustração simplificada de como o algoritmo opera no mapa (n representa *None*), nota-se que a área verificada assume o formato de um losango:



Abaixo o resultado de um teste manual realizado com propósito de autenticar o funcionamento do algoritmo, como os mapas seguem o mesmo comportamento apenas com proporções maiores, os resultados devem estar corretos.

```
8 20
....**********...
.....**.....*2.....
.**..9.............
.**.....*1*........
..8......*....*.....
....**.6..5...*3....
........**.........
....7...**..4.......
```

Viagem	Quantidade Mín. de Movimentos
1 para 2	6
2 para 3	6
3 para 4	5
4 para 5	4
5 para 6	3
6 para 7	5
7 para 8	5
8 para 9	5
9 para 1	5
Total	**44**

Por fim, os resultados obtidos pela execução do algoritmo nos mapas fornecidos pelos fenícios:

Mapa	Qtd. Mín. de Combustível (Und.)	Tempo de Execução Médio (s)*
caso01.txt	832	0.1268
caso02.txt	1688	0.4761
caso04.txt	2954	2.0148
caso06.txt	3828	4.5595
caso08.txt	5896	5.9818
caso10.txt	6344	8.7851
caso15.txt	8112	22.0784
caso20.txt	9334	35.8246

*Foram realizadas 10 execuções para calcular a média.

Conclusão

O algoritmo desenvolvido consegue informar, em tempo aceitável, a quantidade mínima de combustível necessário para realizar cada viagem e é relativamente fácil de compreendê-lo.

Existem algumas melhorias que poderiam ser realizadas: o algoritmo seria mais eficaz se ele começasse a expandir sua busca também pelo porto de destino, houve uma tentativa de implementar essa lógica, entretanto não foi bem sucedida; o algoritmo não tem problema em desconsiderar um porto inacessível, entretanto dois portos inacessíveis consecutivamente culminam em um resultado incorreto.