As Navegações Fenícias

Ana Carolina de Oliveira Xavier
a.xavier004@edu.pucrs.br

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Algoritmos e Estruturas de Dados II
Professor João B. Oliveira
05 de Junho de 2023

Resumo. *Este trabalho tem como objetivo analisar a solução proposta para o Trabalho 2 da disciplina de Algoritmos e Estruturas de Dados II, que visa auxiliar os fenícios a otimizarem o consumo de combustível em suas navegações. A solução foi desenvolvida com base em conceitos de grafo representado por uma matriz de adjacência e busca em largura.*

1. Introdução

O objetivo é desenvolver um algoritmo capaz de informar a quantidade necessária de combustível para a navegação dos fenícios. Este consiste em traçar a rota que parte do porto número 1, percorre os portos de 2 a 9 e, por fim, retorna ao porto 1. Entretanto, há algumas regras que devem ser respeitadas para ser realizada a navegação:

- Os navios se movem apenas para Norte, Sul, Leste e Oeste.
- Os portos são marcados de 1 a 9.
- Cada movimento utiliza uma unidade de combustível.
- Algumas vezes, pode haver momentos que não será possível visitar um porto devido erros de cartografia. No caso, este porto deve ser ”pulado”e a navegação deve seguir em diante.

Para auxiliar na solução do objetivo, foram disponibilizados mapas onde ocorerrão as navegações em formato de arquivo de texto. A Figura 1 é um exemplo de mapa. Esses mapas informam quais são as áreas navegáveis (representadas por pontos), as áreas não navegáveis (representadas por asteriscos) e a localização dos portos (representados de 1 a 9). Além de informar, na primeira linha, quantas linhas e colunas há no mapa.

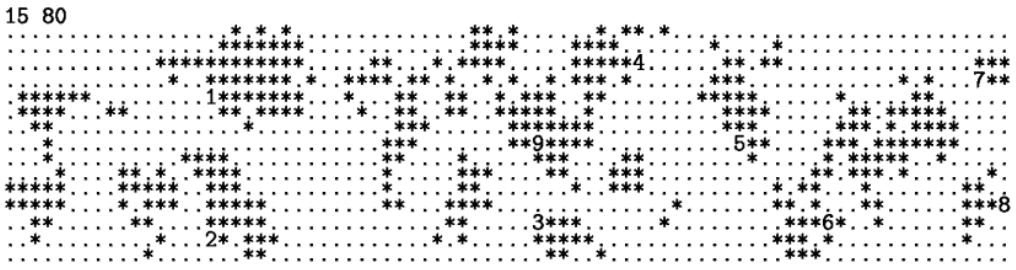

Figura 1. Mapa de navegação

Considerando as regras apresentadas, é possível iniciar o desenvolvimento do algoritmo de solução. Na seção 2, será apresentada a solução encontrada e desenvolvida para resolver o problema. Na seção 3, será apresentado os resultados objetidos a partir dos casos de teste oferecidos. E na seção 4, é apresentado uma breve conclusão sobre o trabalho.

2. Algoritmo de solução

Para representar o mapa em nosso programa, será utilizado um grafo representado por matriz de adjacência. Devido ao formato do mapa, sua leitura é realizada através de repetições para ler cada linha corretamente. O algoritmo inicia lendo as dimensões do mapa a partir da primeira linha, e em seguida, o restante do mapa é armazenado em uma matriz bidimensional de caracteres. Após a leitura do mapa, teremos uma matriz bidimensional que estará armazenando todos as informações necessárias para o desenvolvimento da solução do problema.

Para a solução do problema, inicialmente devemos compreender que para haver o caminhamento entre dois portos, deve-se ser possível chegar de um para o outro através de um caminho navegável, ou seja, sem obstáculos. Caso haver um obstásculo, este porto deve ser ignorado e a navegação deve continuar. Deve ser considerado, então, os caminhos a serem realizados pelas tripulações fenícias, e dessa forma, podemos considerar que a forma mais simples de encontrar todos os caminhos entre os portos é utilizando a matriz de adjacência.

Devido ao tamanho do teste de caso representado na Figura 1, vamos criar um exemplo um pouco menor para facilitar o entendimento do algoritmo:

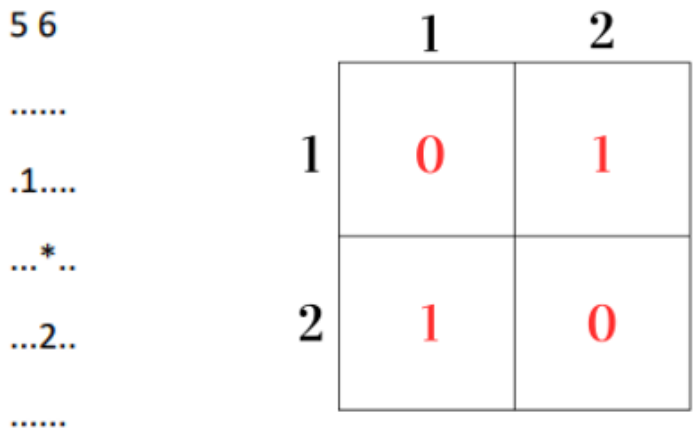

Figura 2. Exemplo simples do algoritmo

Na Figura 2, há um mapa de dimensões bem menores, com apenas 2 portos e 1 obstáculo. As linhas e colunas são representações dos porto. As numerações em vermelho destacam se foi encontrado um caminho entre os dois portos, ou seja, uma intersecção entre eles. Nesse caso, podemos observar que devido não haver obstáculos que impeça a passagem entre os dois portos, é possível sair do porto 1 e chegar ao porto 2.

No algoritmo 1 é descrito o processo para encontrar um porto, passando por parâmetro o mapa e o número do porto que deseja encontrar.

Algorithm 1 Encontrar Porto

```
1: function ENCONTRARPORTO(mapa: matriz de caracteres, numeroPorto: inteiro)
2:     for i de 0 até TAMANHO(mapa) do
3:         for j de 0 até TAMANHO(mapa[i]) do
4:             if EHDIGITO(mapa[i][j]) e PEGAVALORNUMERICO(mapa[i][j]) == numeroPorto then
5:                 return NEW INT[](i, j)        ▷ Retorna a posição do porto encontrado
6:             end if
7:         end for
8:     end for
9:     return null                               ▷ Retorna null se o porto não for encontrado
10: end function
```

Desse modo, o algorito consegue reconhecer se o caractere é um dígito correspondente ao número do porto desejado. Se o porto for encontrado, será retornado a posição do porto como um par de coordenadas.

No Algoritmo 2 temos a implementação do algoritmo de busca em largura para encontrar a distância entre uma posição inicial e uma posição final.

Algorithm 2 Calcular Distância

```
1: for cada movimento em movimentos do
2:     novoX ← x mais a primeira coordenada de movimento
3:     novoY ← y mais a segunda coordenada de movimento
4:     if novoX está dentro dos limites e novoY está dentro dos limites e
5:             (mapa[novoX][novoY] é igual a MAR ou
6:                 (novoX é igual a segunda coordenada de posicaoFinal e novoY é igual a primeira coordenada de posicaoFinal)) e
7:             visitado[novoX][novoY] é falso e
8:             não é um obstáculo em mapa na posição novoX, novoY then
9:         Adicionar [novoX, novoY] à fila
10:        Marcar visitado[novoX][novoY] como verdadeiro
11:    end if
12: end for
13: Incrementar distancia por 1
```

O algoritmo começa inicializando variáveis importantes, como o número de linhas e colunas do mapa, uma matriz booleana para marcar as posições visitadas, uma fila para armazenar as posições a serem visitadas e um conjunto de movimentos possíveis na navegação (Norte, Sul, Leste, Oeste). Assim, a posição inicial é adicionada à fila e marcada como visitada, já a distância é inicializada como zero.

Em seguida, o algoritmo inicia uma repetição principal enquanto a fila não estiver vazia. Dentro dessa repetição, o tamanho atual da fila é obtido e uma nova repetição inicia-se para processar todas as posições presentes na fila neste nível, permitindo percorrer todas as posições do mapa em cada nível de distância.

Em cada posição atual, é verificado os movimentos possíveis a partir dessa posição. Para cada movimento, a nova posição é calculada somando as coordenadas x e y da posição atual com as coordenadas do movimento. Nesse momento, são realizadas uma série de verificações para garantir que a nova posição esteja dentro dos limites do mapa; que seja um espaço navegável e que ainda não tenha sido visitado, além de verificar se a nova posição não é, na verdade, um obstáculo. Caso todas as condições forem atendidas, a nova posição é adicionada à fila e marcada como visitada.

A repetição principal continuará até que a fila seja vazia ou que a posição final seja a encontrada.

O Algoritmo 3 é responsável por calcular a distância total percorrida ao visitar uma série de portos no mapa.

Algorithm 3 Calcular Distancia Total

for $porto$ de 2 até 9 **do**
    $posicaoFinal \leftarrow$ encontrarPorto($mapa$, $porto$)
    $distancia \leftarrow$ calcularDistancia($mapa$, $posicaoAtual$, $posicaoFinal$)
    if $distancia$ é igual a -1 **then**
        Imprimir "Não foi possível encontrar um caminho para o porto $porto$"
        Continuar para o próximo porto
    end if
    Acumular $distancia$ na $distanciaTotal$
    Atualizar $posicaoAtual$ para $posicaoFinal$
end for

A função recebe como entrada um mapa. Cada posição da matriz pode conter informações sobre o terreno ou a presença de um porto.

O algoritmo inicia-se encontrando a posição do primeiro porto (porto 1) através da função explicada anteriormente no Algoritmo 1. E em seguida, ele itera sobre os portos de 2 a 9 em uma repetição.

Para cada porto, ele encontra a posição que se está atualmente e a posição do porto atual, utilizando a função do Algoritmo 1. E em seguida, calcula a distância entre estes dois pontos utilizando a função do Algoritmos 2. Caso não for possível encontrar um caminho entre estes dois pontos, uma mensagem de erro é exibida e a repetição continua para o próximo porto. Em caso contrário, a distância é acumulada na variável "distancia-Total"e a posição atual é atualizada para a próxima iteração.

Após percorrer todos os portos, o algoritmo calcula a distância de volta para o porto 1. Por fim, é retornado o valor final da distância total.

3. Resultados

Após a execução de todos os casos de teste fornecidos pela tripulação fenícia, foram obtidos os seguintes resultados, conforme apresentado na Tabela 1:

Teste	Quantidade de Combustível	Tempo de execução (em segundos)
01	832	0.048s
02	1688	0.106s
04	2954	0.114s
06	3828	0.147s
08	5896	0.227s
10	6344	0.292s
15	8112	0.628s
20	9334	0.771s

Tabela 1. Tabela de resultados

Após uma análise cuidadosa do algoritmo e dos resultados obtidos na tabela, podemos afirmar sua eficácia ao utilizar as informações de forma otimizada, aplicando o conceito de busca em largura em um grafo representado por uma matriz bidimensional. Esse enfoque evita a necessidade de percorrer repetidamente todo o grafo para verificar a existência de caminhos, permitindo identificar o caminho mais eficiente para as tripulações navegarem.

Quanto ao tempo de execução, observamos que o aumento do tamanho dos casos de teste está diretamente relacionado ao aumento do tempo necessário para processá-los. No entanto, é importante ressaltar que o algoritmo se mantém eficiente, uma vez que nenhum dos casos de teste ultrapassou a marca de 1 segundo (ou 1000 milissegundos) de execução.

Esses resultados indicam que o algoritmo é capaz de resolver os desafios de navegação enfrentados pelas tripulações fenícias de forma satisfatória, demonstrando uma abordagem eficiente e adequada para lidar com essas situações.

4. Conclusão

Ao concluir este trabalho, podemos afirmar que obtivemos um resultado satisfatório no desenvolvimento do algoritmo. Utilizando conceitos de grafos, por meio de matrizes de adjacência, e aplicamos o caminhamento em largura para resolver o problema proposto. Conseguindo atender a todos os requisitos estabelecidos e fornecer valores de forma eficiente, garantindo um desempenho adequado na resolução dos desafios de navegação das tripulações fenícias.