Cod Fishing

Autores:
Ricardo Oliveira - nº42647
Vasco Barnabé - nº42819

22 de Abril de 2021

1 Introdução

O 1º trabalho de Estruturas de Dados e Algoritmos II consiste na implementação de uma solução para o problema Cod Fishing.

Neste problema existem várias variáveis, como barcos, locais de pesca, e as suas localizações.

Assim, a nossa tarefa consiste em escrever um programa que, dadas as localizações e as avaliações de desempenho(rating) dos barcos, as localizações dos locais de pesca e a suas quantidades de peixe, calcular a quantidade de peixe que deve ser capturado, a distância total percorrida pelos barcos para viajar até aos respetivos locais de pesca(admitindo que cada barco só pode ser entregue a um local de pesca, e a cada local de pesca só pode ser atribuído um barco), e a soma dos ratings dos barcos que foram atribuídos aos locais de pesca existentes.

No que diz respeito ao cálculo dos valores pretendidos, existem prioridades, e por isso, a solução implementada tem de resolver o problema de uma forma que:

1. primeiro, maximize a quantidade de peixe capturado;
2. segundo, minimize a distância total percorrida pelos barcos;
3. e terceiro, minimize a soma dos ratings dos barcos.

Como foi referido, estas condições em cima apresentadas são prioridades, e por isso, apenas em caso de ocorrerem duas ou mais situações de igualdade na primeira condição, o programa deve dar prioridade à condição imediatamente abaixo, mantendo esta filosofia até à última condição imposta.
Todos os barcos têm ratings diferentes e todos os locais de pesca têm quantidades de peixe diferentes.
Além disso, nenhum barco pode ser associado a um local de pesca com menos peixe que um outro local de pesca associado a um barco com menor rating.
Assim, para a implementação da solução ao problema apresentado, foi aplicado um algoritmo de programação dinâmica, uma vez que o problema apresenta uma subestrutura ótima, isto é, apresenta soluções que são construídas com recurso a soluções ótimas de subproblemas.

2 Desenvolvimento/Implementação

2.1 Pensamento

Para a resolução deste problema, após a leitura do input, foram criadas duas variáveis(uma para guardar o número de barcos e outra para guardar o número de locais de pesca) e dois arrays(de inteiros). Um deles é responsável por guardar toda a informação sobre os barcos, e o outro por guardar a informação dos locais de pesca. Estes arrays são bidimensionais, tendo um tamanho proporcional ao número de barcos ou de locais de pesca que vão guardar. Consoante o número de barcos ou de locais de pesca, assim os arrays têm esse número de linhas, sendo cada linha responsável por guardar valores de um só barco ou de um só local de pesca. Têm sempre 3 colunas, sendo a primeira delas responsável por guardar a coordenada X do objeto em questão, a segunda por guardar a coordenada Y e a terceira por guardar o rating ou a quantidade de peixe do mesmo objeto(dependendo se é barco ou local de pesca).

Observámos que, neste problema, três situações distintas podem ocorrer, sendo elas:

- Igual número de barcos e de locais de pesca;
- Maior número de locais de pesca, relativamente aos barcos;
- Maior número de barcos, relativamente aos locais de pesca.

Assim, a resolução do problema foi dividida nestes 3 casos, a qual será elucidada de seguida.

2.2 Descrição do Algoritmo

Antes de partir para a resolução das três situações já referidas, uma vez que já temos toda a informação necessária disposta em arrays, pensámos em ordenar os mesmos por ordem de rating e de quantidade de peixe. Assim, o array dos barcos ficou ordenado de forma decrescente(de rating), apresentando o barco com maior rating na primeira posição e o barco com menor rating na última. O mesmo foi feito para o array com os locais de pesca, ficando o local de pesca com maior quantidade de peixe na primeira posição, e o local de pesca com menor quantidade de peixe na última. Toda a leitura e todo o tratamento dos dados de input foi realizado na função *main*.

De seguida, foi criada a função *solve* que é responsável por gerar as soluções para o problema.

Tal como já foi referido, a solução deste problema foi dividida em três casos, tendo sido a primeira situação abordada aquela em que há igualdade no número de barcos e de locais de pesca. Em qualquer caso, apenas um barco pode ser atribuído a um local de pesca, e um barco só pode pescar num local de pesca. Nesta situação de igualdade entre ambos, a todos os locais de pesca vai ser atribuído um barco, e uma vez que nenhum barco pode ser associado a um local de pesca com menos peixe que um outro local de pesca associado a um barco com menor rating, os barcos são associados aos locais de pesca por ordem de ratings e quantidades de peixe, isto é, o barco com maior rating vai para o local de pesca com mais peixe, o barco com segundo maior rating vai para o local com a segunda maior quantidade de peixe, e assim sucessivamente...até que o barco com menor rating seja atribuído ao local de pesca com menor quantidade de peixe. Esta atribuição tornou-se relativamente simples uma vez que os arrays de barcos e de locais de pesca foram inicialmente ordenados segundo ratings e quantidades de peixe. Assim, a cada atribuição, era somado o rating desse barco à soma dos ratings de todos os barcos que já tinham sido atribuídos; era somada a quantidade de peixe da mesma forma, e era somada a distância entre o barco e o local de pesca que lhe foi atribuído, também da mesma forma(para o cálculo desta distância, foi criada a função *returnDistanceBetween_B_and_S* à qual é fornecido o barco e o local de pesca em questão, sendo responsável por calcular e devolver a distância entre ambos).

A segunda situação abordada foi aquela em que o número de barcos é inferior ao número de locais de pesca. Aqui, irão haver locais de pesca aos quais não vai ser atribuído nenhum barco. Uma vez que se pretende maximizar a quantidade de peixe apanhado, terão de ser escolhidos os locais de pesca com maior quantidade de peixe. Como o array que contém os locais de pesca já se encontra ordenado, de forma decrescente, e existindo X barcos, é apenas necessário atribuir locais de pesca aos barcos até ao local de pesca número X(iniciando a contagem a partir do início, do local de pesca com mais quantidade de peixe). Desta forma, os locais de pesca que não serão utilizados, serão os com menor quantidade de peixe, resultando assim, que a quantidade de peixe apanhada é máxima, e os requisitos são cumpridos. Da mesma forma que os barcos e locais de pesca eram atribuídos no caso anterior, assim acontece neste caso, somando ratings, distâncias e quantidades de peixe.

A terceira e última situação abordada, trata-se da situação em que o número de barcos é superior ao número de locais de pesca.

Tendo em conta as três prioridades referidas no tópico 1, neste caso passamos diretamente para a segunda prioridade(minimizar a distância total percorrida pelos barcos), uma vez que ao existirem mais barcos que locais de pesca, a todos os locais de pesca será atribuído um barco, sobrando, assim, barcos. Deste modo, a quantidade de peixe capturado foi previamente calculada somando as quantidades de peixe de cada local de pesca.

Para calcular a distância mínima total percorrida pelos barcos, foi implementado um algoritmo de programação dinâmica, onde foi efetuado o cálculo iterativo da solução ótima, por tabelamento.

Assim, criou-se inicialmente uma matriz, tendo X linhas e Y colunas(sendo X o número de barcos + 1, e Y o número de locais de pesca + 1). A última coluna e a última linha da matriz foram preenchidas com o valor 0, para evitar que os cálculos efetuadas fossem diferentes quando fosse necessário calcular valores para os casos nos limites da matriz. Estando as linhas relacionadas com os barcos, e as colunas relacionadas com os locais de pesca, com o aumento das linhas, assim decresce o rating do barco, ou seja, cada linha corresponde a um barco (exceto a última) e o barco de maior rating encontra-se na linha 0, enquanto que o barco de menor rating se encontra na linha X-1. O mesmo acontece com as colunas, em que na coluna 0 se encontra o local de pesca com maior quantidade de peixe, e na coluna Y-1 o local de pesca com menor quantidade de peixe.

Assim, o sentido do progresso deste algoritmo acontece percorrendo toda a matriz, iniciando no canto inferior direito, e terminando no canto superior esquerdo da mesma, na posição (0,0), onde estará a solução ótima para este problema, a distância mínima total percorrida pelos barcos. Mas, apesar de se tratar da mesma matriz, os cálculos efetuados para definir o valor de cada posição não são os mesmos. Assim, a matriz criada pode dividir-se em duas seções, vermelho e verde:

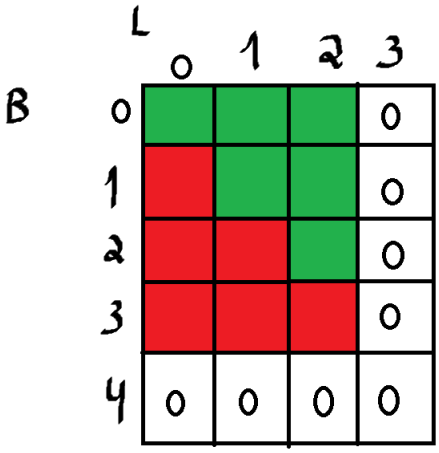

Figura 1: 4 barcos e 3 locais de pesca.

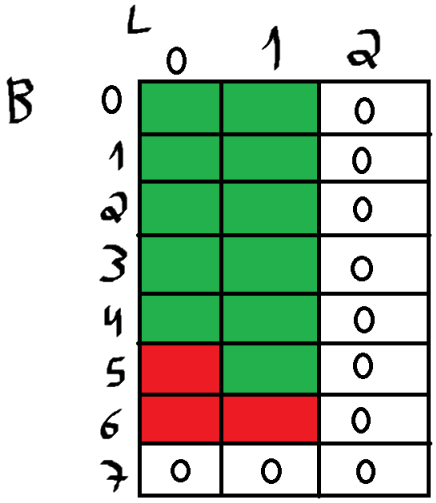

Figura 2: 7 barcos e 2 locais de pesca.

No caso da figura 1, existem 4 barcos e 3 locais de pesca.

Para o cálculo do valor de qualquer posição da matriz, é sempre necessário efetuar o cálculo da distância entre o barco e o local de pesca em questão, e somar o valor obtido ao caso anterior. A diferença na matriz, é se a posição se enquadra na zona vermelha ou na zona verde, sendo a zona vermelha a zona que abrange as situações em que ainda só foram verificados $X \leq K$, para X = número de barcos e K para número de locais de pesca. Nas zonas a verde, já encontramos uma situação em que temos uma superioridade de barcos relativamente aos locais de pesca, e por isso, temos de escolher os melhores barcos, aqueles que, somados, devolvam uma distância mínima total percorrida.

Assim, exemplificando, para a figura 1, para calcular o valor da posição (admitindo a representação da posição na matriz como (linha,coluna)) (2,1), é necessário calcular a distância desse barco a esse local de pesca e somar ao caso anterior, presente na posição (2+1,1+1) = (3,2), sendo assim este o valor da distância mínima percorrida naquela posição. Para calcular, por exemplo, o valor da posição (1,1), é necessário calcular o valor da distância entre esse barco e esse local de pesca, somar ao valor anterior (posição (1+1,1+1)=(2,2), e comparar o resultado

ao valor da casa abaixo (1+1,1)=(2,1); isto faz-se para verificar e escolher qual o melhor barco a ser escolhido, entre aqueles dois(o barco da linha 1 e o barco da linha 2, neste caso). Nesta comparação, o resultado da posição que se pretende calcular, da posição (1,1), será o que engloba o barco daquela linha, apenas se esse valor for ESTRITAMENTE INFERIOR ao valor da linha abaixo, do barco anterior. Isto realiza-se porque, quanto menor for o valor da linha da matriz, maior é o rating dos barcos, e como em caso de igualdade de distância mínima total percorrida pelos barcos, se pretende minimizar os ratings destes(terceira prioridade), é escolhido o barco abaixo, com menor rating, caso a distância seja igual.

Para tal, foi criada uma segunda matriz, igual à matriz das distâncias, para guardar os valores dos ratings dos barcos escolhidos, sendo esta matriz refém da outra, isto é, os valores preenchidos nesta matriz dependem dos valores escolhidos para preencher a matriz das distâncias. É então somado na matriz dos ratings o valor do barco escolhido na matriz das distâncias. O pensamento é o mesmo, tendo no final, na posição(0,0) da matriz, a solução ótima respeitando as prioridades.

Caso a distância calculada na matriz das distâncias seja superior, obviamente, será copiado o valor da casa abaixo, do barco abaixo, que tem menor distância(indo o valor do seu rating para a matriz dos ratings, pelo processo já referido).

Assim, resumindo, cada preenchimento efetuado de forma direta na matriz das distâncias é também efetuado na matriz dos ratings, colocando em cada posição o rating do barco daquela posição somado dos ratings já anteriormente somados; e no caso do preenchimento na matriz das distâncias depender de uma comparação, é escolhido o barco(entre dois barcos, o da linha em questão e o da linha abaixo) que representa a menor distância percorrida. Em caso de igualdade nessa comparação, é escolhido o barco com menor rating. Deste modo, as prioridades são cumpridas e são calculadas as soluções ótimas para o problema.

A figura 2, é um outro exemplo desta situação, onde existem 7 barcos e 2 locais de pesca, onde é aplicado o mesmo algoritmo.

2.3 Complexidade Temporal e Espacial

2.3.1 Complexidade Temporal

O tempo que o programa leva a executar varia com a dimensão do input. Uma vez que o input máximo para o valor de barcos e para locais de pesca é de 4000 elementos(para cada variável), é fácil pensar que o pior caso ocorrerá quando o valores de barcos e de locais de pesca são máximos. No programa criado, caso haja uma igualdade no número de barcos e no número de locais de pesca, este terá uma complexidade de O(n), sendo n o número de barcos ou de locais de pesca. No entanto, este não corresponde ao pior caso, não sendo, assim, O(n) a complexidade temporal do programa. O pior caso ocorre quando o número de barcos é diferente do número de locais de pesca, mais especificamente, quando há mais barcos que locais de pesca. Neste caso, como já foi referido, são criadas duas matrizes(e percorridas ao mesmo tempo) em que cada linha corresponde a um barco e cada coluna corresponde a um local de pesca. Toda a matriz tem de ser percorrida, o que leva a que um ciclo interior seja executado n vezes sempre que o ciclo exterior foi executado. Sendo o ciclo exterior executado n vezes, leva a que a complexidade deste programa seja O(n x n), ou seja, $O(n^2)$.

2.3.2 Complexidade Espacial

O pior caso ocorre quando o número de barcos é inferior ao número de locais de pesca, onde é necessário alocar espaço para duas matrizes. Não por serem duas matrizes, mas sim, por serem matrizes(arrays bidimensionais), a complexidade espacial deste programa é $O(n^2)$, sendo n o número de barcos.

2.4 Comentários

Como foi referido, a solução deste problema foi dividida por três casos: igual número de barcos e de locais de pesca, maior número de barcos relativamente aos locais de pesca, ou menor número de barcos relativamente aos locais de pesca.

No entanto, este problema poderia ter sido resolvido dividindo apenas em dois casos, juntando os casos em que há igualdade nos valores dos barcos e locais de pesca e quando há mais barcos, e, utilizando apenas o algoritmo que foi criado para o caso de haver mais barcos, sem efetuar qualquer alteração. À primeira vista, esta até seria a melhor solução, reduzindo assim o tamanho do programa(menos código).

Ainda assim, foi decidido manter a divisão da solução deste problema nos três casos referidos, uma vez que não é necessária a criação das matrizes para a resolução do caso em que há igualdade de barcos e de locais de pesca. Isolando este caso, o número de ciclos que o programa irá executar será muito menor, apenas n vezes, sendo n o número de barcos ou de locais de pesca, e não n^2(evita assim, também, que sejam criadas duas matrizes desnecessariamente), caso o problema fosse dividido em dois casos, sendo, assim, mais vantajoso existir esta divisão em três.

3 Conclusão

Com este trabalho foi-nos possível entender que, para a resolução de um mesmo problema, há várias soluções possíveis, havendo umas mais eficientes do que outras. O nosso objetivo foi, desde o início, implementar a solução mais eficiente.

Por isso, foi usado o método de programação dinâmica para a resolução deste problema, uma vez que este se tratava de um problema de otimização(problema em que se procura minimizar ou maximizar algum valor associado às suas soluções). Foi então, por isso, criado um algoritmo iterativo para o cálculo do valor de uma solução ótima, por tabelamento.

Para concluir, esta foi uma oportunidade de, após uma aprendizagem teórica, aplicar de forma prática programação dinâmica, método base na solução criada para este problema, onde nos foi possível entender o quão vantajoso pode ser criar a solução mais eficiente; foi também uma oportunidade de treino na linguagem de programação utilizada(Java), e de obtenção de conhecimento e experiência, o que leva a uma melhor tomada de decisões quando se desenvolve um programa.