




Relatório – Trabalho prático


Estrutura de Dados e Algoritmos II

JAVA – COD FISHING

120

(10, 20)

30

(30, 20)

20

(10, 10)

60

(30, 10)

Trabalho realizado por:

Diogo Castanho, 42496

Pedro Grilo, 43012

Grupo: g128



1.Introdução

Cod & Co, a empresa de pesca, identificou vários locais de pesca de bacalhau que estão a tentar atribuir aos seus navios de pesca de sua no Norte Atlântico onde o principal objetivo da empresa é maximizar a quantidade total de pesca de bacalhau.

Cada local de pesca contém uma quantidade estimada de peixes e cada barco tem uma classificação de desempenho.

Um local de pesca só pode ser atribuído a um barco e nenhum barco pode ser atribuído a mais de um local de pesca.

Para pescar no local designado, cada barco tem de se mover desde a sua localização atual até onde está localizado o seu local de pesca e devido às Regras de Tráfego, um barco só pode viajar paralelamente aos eixos do sistema de coordenadas (bidimensional).

Visto que viajar distâncias maiores significa custos adicionais de combustível e os salários das tripulações aumentam com sua classificação, a fim de economizar custos, a Cod & Co gostaria de minimizar a distância total percorrida pelos barcos a caminho de seus locais de pesca e as classificações totais de cada barco.

2.Objetivo

Foi-nos proposta a realização de um trabalho que consistia na implementação de um programa que tem como objetivo receber um input com as localizações e classificações dos barcos de pesca e a localização dos pontos de pesca, juntamente com a quantidade de peixes em cada um deles e gerar um output onde fosse encontrada uma solução ótima para o problema (o cálculo da quantidade total de peixes que podem ser capturados, a distância total percorrida e a soma das classificações dos barcos aos quais foi atribuído um lugar).

Esta “solução ótima” deve ser feita da seguinte forma:

- 1º Maximizando a quantidade de peixes capturados;
- 2º Minimizando a distância total percorrida;
- 3ª Minimizando a soma das avaliações das tripulações;

3.Descrição do algoritmo

Considerando o que foi dado nas aulas, o grupo pensou em utilizar programação dinâmica para a resolução deste problema.

A tabulação no algoritmo foi essencial para deixar o mesmo mais simples e de melhor compreensão.

Para começar, guardou-se os dados de cada barco e de cada local de pesca em classes de objetos, funcionando como uma espécie de “structs” mas em classes, uma vez que não existe este tipo de estrutura de dados na linguagem java. Foram criados métodos get() para aceder aos respetivos valores de cada barco e cada local.

3.Descrição do algoritmo - (Continuação)

Posteriormente, seguindo as dicas do professor, foram organizados os arrays onde estavam guardados os ratings de cada barco e as quantidades de peixe por local por ordem crescente (menor para maior), o que facilitou muito na resolução do problema devido às restrições do mesmo.

Foi criada uma função também para o cálculo da distância (percurso) do barco até ao local de pesca para cálculos intermediários.

Com os dados guardados e organizados deu-se início à função principal onde irá correr o algoritmo para encontrar a melhor solução para cada exemplo dado.

Nesta, os argumentos serão os barcos e os locais de pesca e a programação dinâmica ocorrerá por tabulação.

Foram então inicializadas tabelas para a quantidade de peixe, rating e distancia percorrida.

Dentro de dois ciclos 'for' foram dados os:

Casos base para a solução

- I. Quando não existem barcos;
- II. Quando não existem locais de pesca;
- III. Quando não existem ambos;

Restantes casos para solução do problema

- I. Quando o número de barcos é igual ao número de locais de pesca;
- II. Quando o número de barcos é superior ao número de locais de pesca;
- III. Quando o número de locais de pesca é superior ao número de barcos;

3.Descrição do algoritmo - (Continuação)

A função principal (Cod Fishing) irá então retornar um array com os 3 valores pretendidos inicialmente (quantidade de peixe total, distancia total e rating total) que correspondem à solução ótima do problema para o input dado.

Esta solução ótima é feita por tabulação, onde haverá chamadas recursivas para cada tabela de forma a chegar ao valor “ótimo”, isto é, o valor do canto inferior direito da tabela devido à ordenação crescente dos arrays feita inicialmente.

4.Função Recursiva

```
for(int i = 0; i <= boats.length; i++) {  
    for(int j = 0; j <= spots.length; j++) {  
        if(i == 0 || j == 0) { //Atribuido valor 0 para os casos base : Onde nao ha barcos, onde nao ha sitios ou onde nao  
            totalDistance[i][j] = 0;  
            totalAmount[i][j] = 0;  
            totalRating[i][j] = 0;  
        }  
        else{  
            if(i == j) { //quando numero de barcos = numero de spots disponiveis  
                totalDistance[i][j] = totalDistance[i-1][j-1] + distancia(boats[i-1],spots[j-1]); //distância do ponto é  
                totalAmount[i][j] = totalAmount[i-1][j-1] + spots[spots.length-j].getAmount(); //o amount total é o amoun  
                totalRating[i][j] = totalRating[i-1][j-1] + boats[i-1].getRating(); //o rating total é o rating total da  
            }  
            else if(i > j) { //quando o numero de barcos > numero de spots disponiveis  
                totalDistance[i][j] = Math.min(totalDistance[i-1][j-1] + distancia(boats[i-1],spots[j-1]), totalDistance[i-1][j] + distancia(boats[i-1],spots[j]));  
                totalAmount[i][j] = totalAmount[i-1][j-1] + spots[spots.length-j].getAmount(); //valor do amount é o amoun  
                if(totalDistance[i-1][j] > totalDistance[i][j]) { //caso a distancia total da linha anterior na mesma colun  
                    totalRating[i][j] = totalRating[i-1][j-1] + boats[i-1].getRating();  
                }  
            }  
            else if(totalDistance[i-1][j] == totalDistance[i][j]){ //caso a distancia total da linha anterior na mesma  
                totalRating[i][j] = totalRating[i-1][j]; //porque o de cima tem menor rating, porque os barcos estão  
            }  
        }  
        else if(i < j) { //quando numero de barcos < numero de spots disponiveis  
            totalDistance[i][j] = totalDistance[i-1][j-1] + distancia(boats[i-1],spots[j-1]); //a distancia é a distâ  
            totalAmount[i][j] = Math.max(totalAmount[i-1][j-1], spots[spots.length-j].getAmount()); //o amount  
            totalRating[i][j] = totalRating[i-1][j-1] + boats[i-1].getRating(); //o rating é o rating d
```

4. Função Recursiva - (Continuação)

Resumindo a parte recursiva do algoritmo e as decisões feitas conforme as limitações impostas pelo enunciado:

Primeiramente, é atribuído valor 0 para os casos base, onde não há barcos, não há locais ou não existem ambos deixando as tabelas com a primeira fila e primeira coluna preenchidas com 0's.

Posteriormente aos casos base são tratados os casos onde:

O número de barcos é o mesmo que número de locais de pesca

Neste caso, a distância total para uma posição é calculada pela distância da sua diagonal $[i-1][j-1]$ (distância calculada referentes aos barcos e locais anteriores) em soma com a distância entre o novo barco e local da nova posição da tabela. Havendo o mesmo número de barcos e locais e estando estes organizados por ordem crescente nos respetivos arrays, esta atribuição na tabela funciona quase de forma “direta” para os valores finais.

O rating total para uma posição pretendida será o rating total da sua diagonal (referentes aos barcos e locais anteriores) em soma com o rating do barco de índice $[i-1]$ no array dos barcos (lembrando sempre a ordem crescente dos mesmos).

A quantidade total de peixes será calculada pela quantidade de peixes da sua diagonal $[i-1][j-1]$ em soma com a quantidade de peixes do local spots $[\text{spots.length} - j]$ no array de locais.

4. Função Recursiva - (Continuação)

Em seguida, foram tratados os casos onde:

O número de barcos é superior ao número de locais de pesca

Neste caso surgiram problemas e as decisões já não eram assim tão diretas como no caso anterior.

Neste caso, a distância total para uma posição é calculada pela distância da sua diagonal $[i-1] [j-1]$ (distância calculada referentes aos barcos e locais anteriores) em soma com a distância entre o novo barco e local da nova posição da tabela, porém, terá de haver uma comparação com o valor da distância registada para a linha anterior, minimizando assim a distância para o mesmo número de locais de pesca.

A quantidade total de peixes é calculada da mesma forma que no caso anterior uma vez que há sempre uma maximização das quantidades de peixes nos locais escolhidos.

Já para o rating total haverá duas condições que irão decidir o seu valor para a posição pretendida:

- Caso a distância na linha anterior (calculada anteriormente até ao mesmo local) seja superior à distância para a posição pretendida, então o rating total nessa posição será a sua diagonal (rating total na posição $[i-1] [j-1]$ referentes aos barcos seleccionados anteriormente) em soma com o rating do barco de índice $[i-1]$ no array de barcos (considerando a sua ordem crescente);
- Caso a distância na linha anterior (calculada anteriormente até ao mesmo local) seja igual à distância para a posição pretendida, então o rating total nessa posição será igual à distância na linha anterior onde havia um menor rating de barcos, minimizando assim a soma do rating dos barcos, pedido pelo enunciado;

4. Função Recursiva - (Continuação)

Por fim, foram tratados os casos onde:

O número de barcos é inferior ao número de locais de pesca

Neste caso, a distância total para uma posição é calculada pela distância da sua diagonal $[i-1][j-1]$ (distância calculada referentes aos barcos e locais anteriores) em soma com a distância entre o novo barco e local da nova posição da tabela.

O rating total para uma posição pretendida será o rating total da sua diagonal (referentes aos barcos e locais anteriores) em soma com o rating do barco de índice $[i-1]$ no array dos barcos (lembrando sempre a ordem crescente dos mesmos).

A decisão será feita para a quantidade total de peixes. Esta, sendo calculada de forma a maximizar a quantidade de peixes do local, necessita de uma comparação entre o valor da quantidade de peixes na mesma linha mas no local anterior (considerando os locais organizados por ordem crescente de quantidade de peixes) e a quantidade de peixes do local com índice $[\text{spots.length} - j]$ (considerando a ordem crescente dos mesmos no array), sendo selecionado o maior e respeitando as indicações do enunciado.

5. Análise de complexidades

Para as funções:

- ***ordenarArrayRating(Boats[] boats)***
- ***ordenarArrayAmount(Spots[] spots)***

Ambas as funções possuem dois ciclos for percorridos em um índice N (***boats.length* ou *spots.length***), concluindo assim uma complexidade temporal de $O(n^2)$.

A complexidade espacial para as funções é $O(n)$.

Para a função:

- ***codFishing(Boats[] boats, Spots[] spots)***

A complexidade espacial para as funções é $O((m*n)^3)$, sendo m e n (***boats.length* e *spots.length***).

Analisando a função e o algoritmo, no pior caso a função terá complexidade $O(mn)$ sendo m e n (***boats.length* e *spots.length***).

6. Conclusão e Comentários Extra

Concluindo, o grupo achou que a dificuldade do trabalho não fugiu muito às expetativas iniciais uma vez que o algoritmo desenvolvido para resolver o problema foi trabalhado e implementado consoante o que foi dado nas aulas teóricas e o que estava presente em outros exercícios apresentados nas aulas práticas (dentro da programação dinâmica e uso da tabulação para resolução deste tipo de exercícios).

Assim, considerando o resultado obtido para os exemplos dados e o programa ter recebido “classificação” ***accepted*** no mooshak, conclui-se que o objetivo inicial foi atingido e a tarefa correspondente ao 1º trabalho da disciplina foi resolvida.

Nota: Depois da entrega do trabalho reparámos em alguns comentários com erros correspondentes a uma versão anterior do trabalho e não a final.