Algoritmo “Trilha do Dinheiro”

Guilherme Bohm, Henrique Knack

Engenharia de Software – PUCRS

10 de abril de 2024

Resumo

Este artigo propõe uma alternativa de solução para o primeiro trabalho proposto na disciplina de Algoritmos e Estrutura de Dados II no semestre 2024/01, que consiste em determinar o somatório de todos os números presentes em um mapa, composto por retas e curvas. A solução se mostra aplicável a outros mapas que sigam as mesmas regras, fornecendo o resultado esperado. A sua eficiência é discutida e os resultados para cada caso de teste é mostrado.

Introdução

Dentro do escopo da área de Algoritmos e Estrutura de Dados, um dos problemas que fomos propostos a solucionar é chamado como "Trilha do Dinheiro". Nele, bandidos assaltaram um banco e fugiram, largando quantias de dinheiro ao longo do caminho. A polícia registrou um mapa de sua fuga, demonstrando suas retas, curvas e locais onde o dinheiro foi deixado, e nosso papel é encontrar uma forma de calcular a quantia total de dinheiro deixada ao longo de toda trilha.

Além disso, algumas considerações também valem ser citadas:

1. O mapa é simbolizado por ‘-’ ou ‘|’ indicando retas, ‘\’ ou ‘/’ indicando curvas, ‘#’ indicando o final do mapa, e por números.
2. A coluna mais à esquerda do mapa fica reservada para o ponto de início, que sempre partirá para a direita.
3. A primeira linha indica o número de linhas e colunas presente na trilha.
4. A solução deve levar em conta a ordem de recuperação do dinheiro

Para solucionarmos este problema, analisaremos a nossa alternativa de solução, discutindo o seu funcionamento e demonstrando sua eficiência. Logo após, apresentamos os resultados para cada caso de teste disponibilizado, juntamente com as conclusões que obtivemos ao longo do desenvolvimento deste trabalho.

Solução

Ao analisar o problema, consideramos como solução ideal transpor todo o mapa para uma matriz de caracteres, já que assim seria possível acessar e navegar entre as diferentes colunas e linhas com facilidade. Com este pensamento criamos a classe “LeitorMapa”, responsável por ler o mapa, criar uma matriz de acordo com o tamanho oferecido, e adicionar elemento por elemento em sua devida posição, e então retornar esta matriz.

Tendo acesso à matriz representativa ao mapa, podemos agora passar a analisar a mesma. Para isto, a classe “AcmeMapa” foi criada, com o principal papel de retornar o somatório das quantias deixadas pelos bandidos. Para conseguirmos este valor, tivemos de seguir o seguinte processo:

1. Encontrar a posição inicial.
2. Implementar uma forma de percorrer a matriz.
3. Para cada elemento, realizar o tratamento no caso de ser um número.
4. Para cada elemento, realizar a verificação se representa uma mudança de direção.

Encontrar a posição inicial necessitou de método simples, que verifica se cada linha da primeira coluna apresenta um valor ou não, já que esta coluna está reservada apenas para a posição inicial. Assim, o método pode ser representado da seguinte forma:

```
Para cada x de 0 até o número total de linhas:
    Se elemento na linha x e coluna inicial = ‘-’ então:
        Retorne x
```

Tendo agora a posição inicial do mapa, podemos agora começar a percorrer a matriz e somar as quantias indicadas. Para isso, levamos em consideração o fato de que o caminho a ser traçado só acaba quando encontramos o sinal de ‘#’, criando um laço que se encerra apenas sob esta condição. Este laço tem o trabalho de analisar o elemento atual, realizar o devido tratamento no caso de número ou mudança de direção, e então se atualizar para o próximo elemento.

Tendo implementado o laço encarregado de percorrer toda a matriz, passamos a implementar as validações necessárias. A primeira delas foi o caso de encontrar um

número, e para isso tivemos que tomar um cuidado especial com os números compostos por mais de um algarismo, já que esses não poderiam ser simplesmente somados ao somatório total, evitando que cada algarismo seja adicionado de cada vez.

Para resolver este problema, utilizamos uma variável do tipo String que vai realizar o controle dos números a serem somados, a fim de garantir a precisão do resultado. Se o caractere atual for um número, este será concatenado a esta String, e quando o caractere atual deixar de ser um número, sinalizando seu final, a variável será convertida para um número, somado ao somatório total e então terá seu valor zerado para um próximo uso. Ao final, a validação ficou semelhante a esta maneira:

```
Se o Caractere Atual é número, então
    Caractere Atual é concatenado em Número Atual
Senão se Número Atual não estiver vazio, então
    Somatório = somatório + número atual
    Número atual = “”
```

Agora como última validação necessária, implementamos o tratamento de casos de mudança de direção. Para isso, criamos uma variável responsável de armazenar a direção atual, sendo inicializada para o leste e variando entre ‘L’, ‘O’, ‘N’, ‘S’. Esta variável também será importante no momento de atualizar a posição atual. Quando o caractere atual for ‘\’ ou ‘/’, representando uma troca de direção, é verificada então a direção atual e está será atualizada de acordo com a mudança indicada pelo mapa. O código ficará semelhante a este para cada direção:

```
Se caractere atual é igual a '/' ou é igual a '\' então:
    Se direção é igual a 'L' então:
        Se caractere é igual a '/' então:
            direção = 'N'
        Senão
            direção = 'S'
```

Para finalizar, o laço irá verificar a direção atual e atualizará o caractere a ser analisado, alterando a linha ou a coluna da matriz. Encerrando esse laço, mais uma verificação a String dos números é feita, a fim de evitar o caso de um número não ser somado quando este é seguido diretamente pelo fim do mapa. Ao final deste processo, o método retorna o somatório calculado.

Dessa forma, encontramos uma maneira eficiente de percorrer por todo mapa e somar assertivamente todas as quantias deixadas pelos ladrões. O algoritmo além de demonstrar uma grande velocidade, também pode ser considerado simples, com uma estrutura clara e definida, sendo quase todo baseado em suas estruturas condicionais e leitura de matriz.

Resultados

Após implementar os algoritmos em Java e executá-los, foi obtido as seguintes somas:

TAMANHO DO MAPA	SOMATÓRIO	TEMPO DE EXECUÇÃO (SEGUNDOS)
50^2	14111	0.016
100^2	25411	0.018
200^2	73424	0.024
500^2	871897	0.056
750^2	20481572	0.069
1000^2	13680144	0.078
1500^2	18727169	0.11
2000^2	20748732	0.142

É notável que, apesar do grande aumento no tamanho da matriz, nossa solução manteve um baixo tempo de execução.

Conclusões

Após a análise do código e dos casos de teste disponíveis, é importante abordarmos a eficiência do algoritmo em termos de complexidade temporal, expressa pela notação O. A complexidade temporal descreve como o tempo de execução de um algoritmo cresce à medida que o tamanho da entrada aumenta.

Para entender a complexidade temporal do algoritmo que implementamos, é necessário examinar as principais operações realizadas em relação ao tamanho da entrada. No caso deste código, o tamanho da entrada está corresponde ao tamanho do mapa.

O algoritmo percorre cada elemento do mapa uma vez, os transportando para uma matriz, e então percorre cada elemento não nulo presente na matriz, realizando operações de comparação e atribuição. Portanto, no pior dos casos, percorremos todo o mapa duas vezes, apresentando uma complexidade de O(2n).

No entanto, é importante observar que o número total de elementos visitados pode ser menor que 2n, já que o caminho pode ser mais curto. Portanto, a complexidade real do algoritmo pode variar dependendo das características específicas da trilha percorrida.

Em síntese, o trabalho realizado apresenta uma solução simples e eficiente para o problema proposto, trazendo os resultados com precisão. Sendo assim, é notável que o tempo de execução do programa foi considerado baixo em relação a quantidade de dados que foram lidos. Portanto apesar do tempo de execução aumentar conforme maior for o tamanho da matriz, o programa conseguiu comportar as exigências propostas com exatidão e demostrou sua eficiência em executar estes códigos.