Estratégias Algorítmicas

-

Relatório Problema de Programação 1

Equipa:

Nº Estudante: 2019218981 Nome: Marco da Cruz Pais

Nº Estudante: 2019215412 Nome: Simão Carvalho Monteiro

1. Descrição do Algoritmo

Primeiramente, inserimos a 1ª peça no tabuleiro com a rotação inicial e chamamos a função recursiva ***solveBoard*** com a primeira posição livre. Nesta função, usa-se o ***map*** para obter as peças que poderão ter uma rotação compatível para encaixar na posição livre do tabuleiro e caso seja encontrada uma que encaixe, marcamos como usada e chamamos novamente a função recursiva com a próxima posição livre do tabuleiro (passo recursivo).

Caso chegue à última posição do tabuleiro, ou seja, colunas e linhas máximas (caso base) e exista uma solução, a função vai-se reduzir a ***true*** e vai ser impressa. Caso não exista uma solução, a função irá se reduzir a ***false*** (condição de rejeição) e o puzzle será impossível.

Aumentar desempenho:

- É verificada a compatibilidade da peça para todas as rotações da peça e só depois é feita a rotação, visto que, ao implementar o inverso obtivemos resultados piores.
- A função de pré processamento, ***preprocessing(…),*** permite aumentar o desempenho do algoritmo pois consegue detetar a impossibilidade de um puzzle fazendo uma contagem dos números que têm ocorrências ímpares, ou seja, caso seja superior a 4, o puzzle vai ser impossível pois os únicos que podem ter ocorrências ímpares são os números de cada canto do tabuleiro.
- Na função acima descrita foram também mapeadas as peças que contêm certos pares de números, assim na chamada recursiva apenas são testadas as peças que realmente poderão ser colocadas num certo local do tabuleiro.

2. Estruturas de Dados

Foram usadas duas classes, ***Piece*** e ***Board***, e um ***Map****.*

- ***Piece*** – Criada para guardar cada peça, com um ***array*** de 4 dimensões e monitorizar o uso de cada uma.
- ***Board*** – Criada para guardar o tabuleiro do puzzle com um ***array*** bidimensional e o seu respetivo tamanho e, também, o conjunto de peças do tipo ***Piece***.
- ***Map*** – Usado para mapear pares de números e as peças que os contêm, fazendo uso de ***arrays*** como chaves e vetores como valores.

3. Assertividade

A nossa solução está correta visto que o nosso algoritmo testa todas as combinações de peças possíveis, incluindo múltiplas rotações de uma mesma peça. Para além do citado anteriormente, foi feito pré processamento que, com o algoritmo eficiente, permite-nos obter uma solução rápida e correta.

4. Análise do Algoritmo

- Complexidade Espacial é O(N) pois é referente ao tamanho do tabuleiro, ou seja, quantas mais peças, maior será o tabuleiro e existe um relacionamento linear entre estes dois.
- Complexidade Temporal é O(N Log N) pois pelo ***Master Theorem*** temos que a = 1, que é o número de subproblemas em cada passo recursivo, b = 1, pois é o tamanho de cada subproblema e c = 1, que se refere ao custo do caso base, o que dá:

$$\log_b a = c,\ T(n) = O(n \log n)$$

5. Referências

As seguintes referências foram usadas com intuito de perceber alguns detalhes e funcionalidades sobre estruturas e funções do C++.

➔ https://www.geeksforgeeks.org/
➔ https://www.cplusplus.com/reference/