Relatório - Trabalho 1 EDA II

Luis Borges 47297, Acacio Uando 55730
g147

April 5, 2024

Introdução

O bairro animado de Funville, onde o riso enche o ar e as amizades florescem, é o local escolhido pela organizadora de eventos local, a Miss Merrymaker, para realizar uma festa temática para um grupo querido de amigos. Com um compromisso em espalhar alegria e promover conexões, a Miss Merrymaker tem como objetivo criar um encontro inesquecível celebrando sua paixão compartilhada por jogos de tabuleiro. No entanto, ciente do espaço limitado disponível no coração agitado de Funville, ela aloca meticulosamente os convites para a festa, garantindo que a capacidade do local não seja excedida, ao mesmo tempo em que maximiza a presença de entusiastas de jogos de tabuleiro.

Descrição do Problema

O problema consiste em criar um programa para ajudar a Miss Merrymaker a organizar uma festa temática para grupos de jogadores com base em seus interesses em vários jogos de tabuleiro e seu entusiasmo. O programa recebe o espaço disponível (em número de jogadores) para a festa e uma lista de jogos de tabuleiro em ordem crescente de solicitação. Cada solicitação de jogo de tabuleiro é representada pelo nome do jogo, o número de jogadores do jogo e o entusiasmo total que esses jogadores trazem para a festa. O programa deve

levar em conta o espaço disponível para a festa e priorizar o maior entusiasmo possível na festa, garantindo que o número total de jogadores não exceda o espaço disponível do local. O programa deve imprimir o número total de jogadores na festa e o valor total de entusiasmo da festa. Para decidir quais jogos de tabuleiro levar, deve-se priorizar a lista de jogos de tabuleiro com maior entusiasmo. Se duas listas de jogos de tabuleiro tiverem o mesmo entusiasmo, o programa deve selecionar a lista de jogos de tabuleiro que contém a solicitação mais antiga.

Identificação da solução óbvia

A solução óbvia para este problema seria uma abordagem de força bruta, onde todas as combinações possíveis de seleção de jogos seriam consideradas e a melhor delas seria escolhida. No entanto, essa abordagem seria extremamente ineficiente, pois o número total de combinações possíveis aumenta exponencialmente com o número de jogos disponíveis.
Para identificar a solução óbvia, podemos analisar o espaço de busca. Suponha que tenhamos n jogos disponíveis. O número total de combinações possíveis é 2^n, pois para cada jogo, podemos escolher incluí-lo ou excluí-lo da seleção final. Isso resulta em uma complexidade de tempo exponencial, o que não é prático para o número de jogos (até 60) e espaço disponível (até 600000) especificados no enunciado.
Portanto, a solução óbvia, que consideraria todas as combinações possíveis, não é viável devido à sua complexidade computacional. Em vez disso, é necessário utilizar uma abordagem mais eficiente, como a programação dinâmica, como implementado na função findBestSelection do código fornecido. Essa abordagem oferece uma solução ótima em termos de tempo de execução, garantindo que o programa possa lidar com instâncias do problema em um tempo razoável.

Implementação do Programa

O programa é implementado em Java e consiste em várias partes:

1. **Definição da classe BoardGame:** Esta classe representa um jogo de tabuleiro com atributos como nome, número de jogadores e nível de entusiasmo.

2. **Função "findBestSelection":** Esta função recebe uma matriz de objetos BoardGame e o espaço disponível. Utiliza programação dinâmica para calcular a seleção ótima de jogos com base no espaço disponível e no número de jogadores de cada jogo.

    1. **Inicialização de Variáveis:**
        - Obtém o tamanho do array *games* e armazena em n.
        - Inicializa uma matriz *dp* com dimensões $(n+1)\times$(availableSpace+ 1) para armazenar os resultados intermediários do algoritmo de programação dinâmica.
    2. **Cálculo da Matriz DP:**
        - Utiliza programação dinâmica para preencher a matriz *dp*.
        - Itera sobre os jogos disponíveis de 1 até n.
        - Para cada jogo, itera sobre o espaço disponível de 1 até *availableSpace*.
        - Para cada combinação de jogo e espaço, verifica se o número de jogadores do jogo atual é maior do que o espaço disponível.
        - Se o número de jogadores do jogo atual for maior, o valor na matriz *dp* será o mesmo que o valor obtido sem incluir o jogo atual.
        - Se o número de jogadores do jogo atual couber no espaço disponível, o valor na matriz *dp* será o máximo entre o valor obtido sem incluir o jogo atual e o valor obtido incluindo o jogo atual.
    3. **Reconstrução da Seleção Ótima de Jogos:**
        - Inicializa um array *selectedGamesIndices* para armazenar os índices dos jogos selecionados.
        - Inicializa uma variável *index* para rastrear a posição atual no array *selectedGamesIndices*.
        - Inicializa duas variáveis, i e j, para rastrear as posições atuais na matriz *dp* (linha e coluna, respectivamente).
        - Enquanto houver jogos disponíveis ($i > 0$) e espaço disponível ($j > 0$):
            - Verifica se o valor na matriz *dp* na célula atual é diferente do valor na célula anterior na mesma coluna.

- Se for diferente, adiciona o índice do jogo atual ao array *selectedGamesIndices* e atualiza o espaço disponível subtraindo o número de jogadores do jogo atual.
- Decrementa o valor de i para mover para a linha anterior na matriz *dp*.

- Cria um novo array *result* com o tamanho correto (o número de jogos selecionados).
- Copia os índices dos jogos selecionados do array *selectedGamesIndices* para o array *result*.
- Retorna o array *result* contendo os índices dos jogos selecionados.

3. **Funções "calculateTotalPlayers" e "calculateTotalEnthusiasm":** Estas funções calculam respectivamente o número total de jogadores e o entusiasmo total dos jogos selecionados.

4. **Método "main":** No método principal, são lidos os dados de entrada que representam o espaço disponível e informações sobre os jogos de tabuleiro. Em seguida, chama a função findBestSelection para encontrar os jogos ideais e imprime o resultado.

Analize da Complexidade

A complexidade depende da alocação da matriz "dp". Cada célula da matriz "dp" é preenchida com base em duas iterações: uma iteração sobre o número de jogos disponíveis (n) e outra iteração sobre o espaço disponível (availableSpace).

- **Espacial:** O algoritmo utiliza uma matriz *dp* de tamanho $(n + 1) \times (\text{availableSpace} + 1)$, onde n é o número de jogos de tabuleiro e *availableSpace* é o espaço disponível. Portanto, o espaço utilizado é proporcional ao produto desses dois valores, o que resulta em uma complexidade de espaço de $O(n \times \text{availableSpace})$.

- **Temporal:** O algoritmo usa programação dinâmica para calcular a matriz *dp*. Assim, o tempo de execução é proporcional ao número de subproblemas que precisam ser resolvidos, que é dado por $(n + 1) \times (\text{availableSpace} + 1)$, resultando em uma complexidade de tempo de $O(n \times \text{availableSpace})$.

Portanto, tanto a complexidade de temporal quanto a complexidade de espacial são quadráticas, $O(n^2)$.