Engenharia Informática

2023/24

Relatório trabalho prático

Board Game Party

Estrutura de Dados e Algoritmos

Trabalho realizado por
Grupo 123

Ruben Farinha, 48329

Diogo Ferreira, 43075

1. Introdução

A Miss Merrymaker está a planear uma festa para celebrar o amor pelos jogos de tabuleiro. No entanto, devido ao espaço limitado, é crucial selecionar o número ideal de jogos de tabuleiro para garantir que o entusiasmo seja maximizado sem exceder o limite de pessoas no espaço. Este relatório apresenta o programa em Java que implementa um algoritmo de programação dinâmica conhecido como "knapsack" para determinar quais jogos selecionar de maneira eficiente, levando em consideração o número máximo de jogos que podem ser acomodados no espaço disponível, priorizando o máximo de entusiasmo possível.

2. Resolução do problema

Considerando os conceitos aprendidos durante as aulas práticas, o grupo reconheceu que a abordagem mais apropriada para resolver o problema seria através da aplicação da programação dinâmica, utilizando um algoritmo clássico conhecido como "knapsack". Após uma análise detalhada do funcionamento desse algoritmo e da compreensão do problema em questão, conseguimos estabelecer uma conexão significativa entre o algoritmo e a natureza do problema em discussão, facilitando assim a resolução deste.
Chegámos então à conclusão de que a função a ser utilizada para resolver o problema teria de ser a seguinte:

$$V(i,\ s) = \vdash^{0 \quad se\ i=0\ ou\ s=0}_{\max(V[i-1,\ s],\ V[i-1,\ s-s[i]]+\ E[i])}$$

Onde i = # jogos e s = espaço

3. Descrição do algoritmo

1. Input:
   Começamos por ler o input com as variáveis int space e int ngames, onde guardamos o número que representa o espaço disponível e o número de jogos, respetivamente. Optámos depois por criar uma class Games, para facilitar a sua manipulação, onde são guardados os seus detalhes: name, players e enthusiasm.
   É inicializada também uma matriz onde vamos armazenar os cálculos com dimensão [nGames +1][space + 1]
2. Algoritmo:

   De seguida percorremos a matriz com um duplo For.
   O primeiro "for", itera sobre cada jogo disponível e o segundo "for" itera sobre a quantidade de espaço disponível.
   O primeiro passo é verificar se é possível o jogo atual ser incluído de acordo com o espaço disponível. Depois de verificado, é calculado o máximo de entusiasmo que pode ser alcançado, comparando a hipótese de incluir o jogo com a hipótese de não incluir o jogo.
   Após a matriz ser preenchida por completo, faz-se o retrocesso na matriz começando no quadrante máximo [nGames][Space], e vai-se verificando se o jogo é incluído pelo valor de entusiasmo, se o entusiasmo mudou é porque o jogo foi incluído, guardamos então o jogo no array selectedGames.

3. Output:
   Por fim com os jogos encontrados, o processo é simples, já guardado anteriormente na variável selectedCount, temos o número de jogos, imprimimos na primeira linha então o número de jogos (selectedCount), o espaço ocupado (space – remainingSpace), e o entusiasmo total presente no ultimo quadrante da matriz ([nGames][space])
   Nas linhas consecutivas damos output do nome dos jogos selecionados que são os jogos guardados em selectedGames, usando um simples for permitiu-nos dar output dos jogos pela ordem de input.

4. Complexidade

O algoritmo knapsack utiliza um loop duplo para percorrer todos os jogos disponíveis e todas as possíveis quantidades de espaço. Portanto, tanto a complexidade temporal como a complexidade espacial do total do algoritmo é aproximadamente O(nGames * space), onde nGames é o número total de jogos e space é o espaço disponível. Isto significa que o desempenho do algoritmo pode diminuir significativamente à medida que o número de jogos e/ou o espaço disponível aumentam.