UNIVERSIDADE
DE ÉVORA

Trabalho 1 – Board Game Party

Estrutura de Dados e Algoritmos II

Realizado por:

Nome do utilizador no Mooshak: g145
Mauro Serpa 54692
Tiago Pinto 54718
Curso: Engenharia Informática

Introdução

Com este trabalho pretendemos utilizar a linguagem Java para resolver o problema “Board Game Party”, que consiste em encher uma sala, com limite de espaço, com jogos. Os jogos irão ser selecionados a partir do espaço de jogadores que cada um tem e o entusiasmo de cada um, até ocupar o limite de espaço da sala.

Implementação

Para construir o algoritmo do programa, inicialmente começamos pela leitura dos dados, e pela forma como os guardamos. Depois criamos algumas variáveis que iam ser úteis para o algoritmo futuramente, entre as quais foram:

- Arrays para guardar o espaço e o entusiasmo de cada jogo, (P e E, respetivamente);
- Arrays para guardar todos os jogos do input, e apenas os jogos aceites pelo programa, (jogos e jaceites, respetivamente);
- Uma matriz que guarda os valores dos entusiasmos, (tabela);
- Uma variável para guardar o espaço total (p).

Método max

Após a leitura dos dados pelo algoritmo, implementamos um método, chamado max, com o intuito de criar uma tabela com os valores dos entusiasmos, deforma a descobrir o melhor de todos.

Definimos que a matriz tabela, não tem valores na linha e coluna 0, pois nenhum dos jogos tem espaço 0, de seguida assim, preenchemos a tabela a partir do momento em que se acha uma posição que seja igual ao espaço do jogo, e preenchemos com o valor do entusiasmo desse mesmo jogo, enquanto ele vê a posição que se encontra na linha anterior e na coluna j - s e somando ao valor atual. Sempre que a soma desses dois valores for maior ao valor que se encontra na linha anterior ([i -1][j]), esse novo valor é introduzido na tabela, e repetirá até achar um valor maior, assim temos a certeza de que achamos o melhor valor possível. Este valor encontra-se, pelo menos, na última posição da tabela. Este método retorna a tabela com todos os valores do entusiasmo.

Fim da implementação

De seguida criámos um loop deforma a descobrir quais são os jogos que foram aceites. Partindo da última posição da tabela vemos se o elemento da linha anterior é diferente, caso seja verdade, guardamos o nome do jogo em jaceites e faz o mesmo processo no valor que se encontra na linha anterior e na coluna j-s, ao mesmo tempo somamos o espaço de cada jogo e contamos o número

de jogos aceites, caso seja falso, esse jogo não é aceite e verificamos o elemento da linha anterior e na mesma coluna. No final escrevemos o número de jogos aceites, o espaço total e o entusiasmo total e escrevemos o nome dos jogos por ordem que foram aceites primeiro.

Função recursiva utilizada no programa

```
public static int[][] max(int P[], int E[], int S) {
    int[][] tabela = new int[P.length + 1][S + 1];
    for (int i = 1; i <= P.length; i++) {
        int s = P[i - 1]; // espaço de cada jogo que se encontra na posição i-1
        int e = E[i - 1]; // entusiasmos de cada jogo que se encontra na posição i-1
        for (int j = 1; j <= S; j++) {
            tabela[i][j] = tabela[i - 1][j];
            if (j >= s && tabela[i - 1][j - s] + e > tabela[i][j]) {
                tabela[i][j] = tabela[i - 1][j - s] + e;
            }
        }
    }
    return tabela;
}
```

Cálculo das Complexidades

Complexidade Temporal:

Dentro da função max temos um ciclo dentro de um ciclo. No primeiro ciclo, as suas condições, variáveis e operações têm complexidade O(1). A segunda também contém complexidade O(1). Devido ao ciclo ter de percorrer, no máximo, N+1 vezes, a sua complexidade é de O(n). No segundo ciclo o array percorre S vezes, logo tem complexidade O(S). Como se trata de um ciclo dentro de um ciclo, a complexidade é O(nS).

Dentro do main temos 2 loops. No primeiro loop, temos que as condições, as operações e a criação de variáveis têm, em conjunto, complexidade O(1), o loop é percorrido N vezes, logo a sua complexidade é O(N). O segundo loop percorre c vezes, sendo c a variável count, assim tem complexidade O(c). Assim a Complexidade Temporal é O(NS) pois O(1)+O(N)+O(c)+O(NS) = O(NS).

Complexidade Espacial:

Os arrays P, E, jogos e jaceites têm tamanho N, logo a sua complexidade é O(N). A matriz tabela tem tamanho NS, logo a sua complexidade é O(NS). A criação de variáveis, as suas operações e as condições tem complexidade O(1). Ou seja, a complexidade espacial do programa é O(1)+O(N)+O(NS)=O(NS).