RELATÓRIO TRABALHO 1 DE ADA – GAME OF BEANS

Ano Letivo	2020/2021	Semestre	2	Cadeira	ADA
Alunos	Gonçalo Martins Lourenço nº55780				
	Joana Soares Faria nº 55754				

Resolução do Problema

Definimos $S(i,j,p)$ como sendo a função de nos dá a máxima pontuação da Jaba quando as pilhas em jogo começam na i e acabam na j, sendo que é a vez do jogador p jogar.

Na definição desta função sentimos a necessidade de estabelecer duas funções auxiliares:

- $score(k,i,j)$ que nos devolve a pontuação obtida por retirar k pilhas do conjunto de pilhas que começa em i e acaba em j;
- $jogadaPieton(i,j)$ que nos devolve a melhor forma que o Pieton pode jogar, seguindo os seus critérios. Devolve-nos um array a, de duas posições, cuja primeira posição nos diz o número de pilhas que o Pieton vai tirar da esquerda e a segunda nos diz o número de pilhas que o Pieton vai tirar da direita. (Tendo em consideração que se o Pieton retira pilhas da esquerda não irá tirar pilhas da direita e vice-versa).

$$S(i,j,p)\begin{cases} P_i, & se\ i=j\ e\ p=Jaba \\ 0, & se\ i=j\ e\ p=Pieton \\ \max\limits_{1\le k\le \min(D,j-i)} \begin{pmatrix} S(i+k,j,Pieton)+score(k,i,j), \\ S(i,j-k,Pieton)+score(k,i,j) \end{pmatrix}, & se\ i<j\ e\ p=Jaba \\ S(i+a[0],j-a[1],Jaba), \\ a=jogadaPieton(i,j) & se\ i<j\ e\ p=Pieton \end{cases}$$

Nesta função consideramos dois casos base, ambos que representam a situação em que há apenas uma pilha em jogo. Numa das situações é o Pieton a jogar, sendo que ele fica com o total da pontuação desse monte, dando à Jaba a pontuação de zero. Na outra situação é a Jaba a jogar, pelo que acumula a pontuação da pilha à sua pontuação final.

Nos casos gerais tratamos igualmente dos dois casos possíveis: é o Pieton a jogar ou é a Jaba a jogar, e consideramos sempre que $i<j$ pois se $i>j$ então já não haverá mais pilhas e o jogo estará terminado.

No caso de ser a Jaba a jogar a pontuação máxima que poderá obter será a melhor maneira possível de tirar k pilhas ou da direita ou da esquerda, passando a jogada ao Pieton. Vamos então chamar recursivamente a função para saber a melhor forma de jogar nessa nova situação e acrescentamos à pontuação da Jaba o correspondente a tirar as k pilhas da melhor forma.

No caso de ser o Pieton a jogar, então a pontuação que ele retirou não irá para a Jaba, sendo que a melhor pontuação da Jaba corresponderá a chamar recursivamente a função com menos as k pilhas que o Pieton removeu, sendo que é a vez da Jaba jogar.

Complexidade Temporal

Considerando as seguintes quantidades:

- n, número de pilhas;
- p, número de pilhas a considerar;
- k, máximo de pilhas que é possível retirar que pode ser igual à profundidade ou menor se sobrarem menos pilhas ($1 \leq k \leq min(Depth, p)$);
- c, o número de pilhas a considerar retirar na jogada ($1 \leq c \leq k$);

Consideramos agora a complexidade temporal necessária para resolver o problema:

- Primeiramente efetuamos o preenchimento dos casos base na matriz de resultados que se traduz numa complexidade de $\Theta(n)$;
- Seguidamente, o tempo de preenchimento da tabela de resultados dos restantes valores é calculado por:
    - Começamos com um primeiro ciclo que representa o número p de pilhas a considerar ($p = i - j \; e \; 1 \leq p \leq n$). Este ciclo é efetuado n vezes pelo que temos complexidade de $\Theta(n)$;
    - Aninhado neste primeiro ciclo temos outro ciclo que representa o ponto inicial onde começar a considerar a pilha, o i ($1 \leq i \leq n - p$), sendo que executamos este ciclo $n - p$ vezes, obtendo uma complexidade de $\Theta(n - p)$;
    - Finalmente, neste ciclo começamos por calcular a pontuação da Jaba, em que vamos determinar quantas pilhas vamos retirar. Para isso executamos um ciclo k vezes, em que k é o número máximo de pilhas que se pode retirar, e dentro desse ciclo somamos a pontuação de tirar as c pilhas. ($1 \leq k \leq min(Depth, p) \; e \; 1 <= c <= k$). Obtemos assim uma complexidade de $\Theta(k \times c) \times 2$, porque temos de considerar retirar as pilhas da esquerda ou da direita;
    - Acabamos por calcular a pontuação da Jaba se for uma jogada do Pieton e para isso temos de determinar quantas pilhas o Pieton vai retirar e por um raciocínio semelhante ao anterior obtemos novamente $\Theta(k \times c) \times 2$;

Tomando todos estes aspetos em consideração obtemos a seguinte complexidade temporal:

$$\Theta(n) \; + \; O(n \times (n - p) \; \times \; ((k \times c) \times 2 + (k \times c) \times 2)) \; = \; \Theta \; (n^2)$$

e como $n > p > k > c$, simplificando obtemos uma complexidade temporal de $\Theta(n^2)$

Complexidade Espacial

Considerando as seguintes quantidades:

- n, número de pilhas;

A complexidade espacial do problema pode dividir-se entre as seguintes:

- Complexidade constante de duas variáveis ($Depth$, ou número de colunas permitido retirar da pilha, e a $String$ correspondente ao nome do primeiro jogador, acrescendo mais as variáveis auxiliares declaradas nos diferentes métodos. Resulta assim em $\Theta(1)$
- Complexidade espacial linear, das pilhas dadas pelo enunciado, $\Theta(n)$;
- Por fim, para guardar as pontuações da Jaba usamos uma matriz de três dimensões, $n + 1$ por $n + 1$ por 2, e assim sendo temos uma complexidade espacial de $\Theta(2 \times n^2)$.

Concluindo, temos uma complexidade espacial de: $\Theta(n) + \Theta(2 \times n^2) = \Theta(n^2)$

Conclusões

Indiquem os pontos fortes e fracos da vossa solução, as alternativas estudadas ou que mereciam ser estudadas, possíveis melhoramentos, etc.

A nossa solução responde ao problema em tempo polinomial, o que é um benefício relativamente ao uso de uma função recursiva que teria chamadas repetidas, o que levaria a uma complexidade temporal.

Uma possível melhoria no algoritmo implementado seria a redução da complexidade espacial, usando apenas uma matriz bidimensional, representando cada célula da matriz como o melhor resultado da pontuação da Jaba numa ronda do jogo (considerando uma ronda uma jogada de cada jogador). Não optamos por esta solução porque o código produzido tinha algumas falhas e tinha uma compreensão um pouco mais difícil.

Anexo – Código Main

```java
/*
 * Ada Trabalho 1 - Game of beans
 *
 * @author Joana Soares Faria n55754
 * @author Goncalo Martins Lourenco n55780
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;


public class Main {

    public static void main(String[] args) throws IOException {

        GameOfBeans game;

        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

        //Tests, Piles, Depth
        int numTests = Integer.parseInt(in.readLine());

        for (int i = 0; i < numTests; i++) {
            //Get P and D
            String[] P_D = in.readLine().split(" ");
            int numPiles = Integer.parseInt(P_D[0]);
            int depth = Integer.parseInt(P_D[1]);

            String[] piles = in.readLine().split(" ");
            int[] aux = new int[numPiles];
            for (int j = 0; j < numPiles; j++) {
                aux[j] = Integer.parseInt(piles[j]);
            }
            String player = in.readLine();
            game = new GameOfBeans(depth, aux, player);
            int solution = game.bestJabaScore();
            System.out.println(solution);

        }

    }

}
```

ANEXO – CÓDIGO CLASS GAMEOFBEANS

```java
/**
 * Ada Trabalho 1 - Game of beans
 *
 * @author Joana Soares Faria n55754
 * @author Goncalo Martins Lourenco n55780
 */

public class GameOfBeans {

    private static final int JABA = 0;
    private static final int PIETON = 1;
    private static final int LEFT = 0;
    private static final int RIGHT = 1;
    private final int[][][] bestScores;
    private final int depth;
    private final int[] piles;
    private final String firstPlayer;


    public GameOfBeans(int depth, int[] piles, String firstPlayer) {
        this.depth = depth;
        this.piles = piles;
        this.firstPlayer = firstPlayer;
        bestScores = new int[piles.length + 1][piles.length + 1][2];
    }

    /**
     * Computes  Pieton's play.
     *
     * @param i left index(-1) of the pile
     * @param j right index(-1) of the pile
     * @return the number of piles removed from which side
     */
    private int[] pietonPlay(int i, int j) {
        int maxScore = Integer.MIN_VALUE;
        int[] answer = {0, 0};

        //left play
        for (int k = 1; k <= depth && (i + k <= j + 1); k++) { // k is the number of piles to remove
            int sum = 0;

            for (int c = 0; c < k; c++) { // c is a counter to sum all the piles to remove
                sum += piles[i + c - 1];
            }

            if (sum > maxScore) {
                maxScore = sum;
                answer = new int[]{k, 0};
            }
        }

```

```java
        //right play
        for (int k = 1; k <= depth && (j - k + 1 >= i); k++) {
            int sum = 0;

            for (int c = 0; c < k; c++) {
                sum += piles[j - c - 1];
            }

            if (sum > maxScore) {
                maxScore = sum;
                answer = new int[]{0, k};
            }
        }

        return answer;
    }


    /**
     * Computes the Jabas's score from removing k piles from the piles i to j
     * k>0 means we should augment i, k<0 means we should move j backwards
     *
     * @param k piles to remove
     * @param i left index(-1) of the pile
     * @param j right index(-1) of the pile
     * @return the score from removing k piles
     */
    private int score(int k, int i, int j) {
        int score = 0;

        if (k < 0) {
            for (int counter = 0; counter < -k; counter++) {
                score += piles[j - 1 - counter];
            }
        } else {
            for (int counter = 0; counter < k; counter++) {
                score += piles[i - 1 + counter];
            }
        }

        return score;
    }

    /**
     * Computes the max score for Jaba, solves the problem
     *
     * @return the maximum Jaba score
     */
    public int bestJabaScore() {
        int player = firstPlayer.equalsIgnoreCase("jaba") ? JABA : PIETON;
```

```
102         //base cases = only one pile left (i=j)
103         for (int i = 1; i <= piles.length; i++) {
104             bestScores[i][i][PIETON] = 0;
105             bestScores[i][i][JABA] = piles[i - 1];
106         }
107 
108         //general case. P is the difference of indices of piles
109         for (int p = 1; p < piles.length; p++) {
110             for (int i = 1; i <= piles.length - p; i++) { //i is the left
     index
111                 int j = i + p; //j is the right index
112                 int maxScoreJaba = Integer.MIN_VALUE;
113                 int maxToRemove = Integer.min(depth, p + 1);
114 
115                 //it is Jaba 's turn to play
116                 for (int k = 1; k <= maxToRemove; k++) { //k is the
    number of piles to remove
117                     //remove from left
118                     int scoreLeft = score(k, i, j);
119                     if (k + i <= piles.length)// does not empty the piles
120                         scoreLeft += bestScores[i + k][j][PIETON];
121                     maxScoreJaba = Integer.max(scoreLeft, maxScoreJaba);
122                     //remove from right
123                     int scoreRight = score(-k, i, j);
124                     if (j - k > 0)// does not empty the piles
125                         scoreRight += bestScores[i][j - k][PIETON];
126                     maxScoreJaba = Integer.max(scoreRight, maxScoreJaba);
127                 }
128 
129                 bestScores[i][j][JABA] = maxScoreJaba;
130 
131                 //it is Pieton 's turn to play
132                 int[] play = pietonPlay(i, j);
133                 int maxScorePieton = 0; //Jaba's best score if is Pieton'
    s play
134                 //does not empty the piles
135                 if (play[LEFT] + i <= piles.length && j - play[RIGHT] > 0
    ) {
136                     maxScorePieton = bestScores[i + play[LEFT]][j - play[
    RIGHT]][JABA];
137                 }
138                 bestScores[i][j][PIETON] = maxScorePieton;
139 
140             }
141 
142         }
143         //solution is the best possible way for Jaba to score with the
    piles from 1 to the last
144         return bestScores[1][piles.length][player];
145     }
146 
147 }
```