ADA- Relatório Prático Trabalho 1

Beans Game ii

Rodrigo Ribeiro nº49067

Nádia Mendes nº53175

11-04-2021

Função Recursiva

$$S(i,j,\alpha) = \begin{cases} 0 & \text{if } str > end \\ 0 & \text{if } i = j \cap \alpha = false \\ piles[i] & \text{if } i = j \cap \alpha = true \\ \max_{\{0 \le d \le depth\}} max(\sum_{n=i}^{i+d} piles[n] + S(i+d+1, j, false), \sum_{n=j-d}^{j} piles[n] + S(i, j-d-1, false)) & \text{if } \alpha = true \quad , depth = min(gamedepth - 1, j - i) \\ S(i+pL+1, j, true) & \text{if } \alpha = false \cap \sum_{n=i}^{i+pL} piles[n] \le \sum_{n=j-pR}^{j} piles[n] \\ \quad , pL = bestPietonMoveLeftSide(i,j) & \\ S(str, end - pR - 1, true) & \text{if } \alpha = false \cap \sum_{n=j-pR}^{j} piles[n] > \sum_{n=i}^{i+pL} piles[n] \\ \quad , pR = bestPietonMoveRightSide(i,j) & \end{cases}$$

- Caso o início da subsequência seja maior que o fim, o score da Jaba é nulo.
- Caso apenas exista uma pilha e seja o Pieton a jogar, o score da Jaba é nulo.
- Caso apenas exista uma pilha e seja a Jaba a jogar, o score da Jaba é o número de feijões contidos nessa pilha.
- Caso existam várias pilhas e seja a Jaba a jogar, o score da Jaba é obtido pela a jogada que maximiza o número de feijões recolhidos no turno corrente, mais o score da Jaba apos o Pieton ter jogado na subsequência restante.
- Caso existam várias pilhas e seja o Pieton a jogar, o score da Jaba é igual ao seu score caso fosse ela a primeira a jogar na subsequência que não tem contem as pilhas removidas pelo o Pieton, neste turno.

Complexidade Temporal

Nesta secção cada método é analisado, de forma a descobrir a complexidade temporal total da solução implementada.

Total

Max(BeansGame2.solve) = $\Theta(N^2.D)$, N = npiles-1, D = min(gamedepth-1, npiles-1)

BeansGame2.solve()

```
computeBeanPilesSums();
if(gamedepth > 1) computePietonMoves();
```

Max(computeBeanPileSums, computePietonMoves) = $\Theta(N^2)$, N = npiles-1

```
for (int i=0; i<npiles; i++)
    jabaScore[i][i] =  piles[i];
```

$\Theta(N)$, N = npiles-1

```
for(int i=npiles-2; i>=0; i--)
    for(int j=i+1; j<npiles; j++)
        jabaScore[i][j] = computeJabaScore(i,j);
```

$\sum_{i=N-1}^{0}\sum_{J=i+1}^{N}$ computeJabaScore = $\frac{N*(N-1)}{2}$ * computeJabaScore = $\Theta(N^2 D)$, D = min(gamedepth-1,npiles-1), N = npilies-1

```
if(firstJaba)
    return getJabaScore(0, npiles - 1);
else
    return getJabaScoreAfterPietonTurn(0, npiles - 1);
```

Max(getJabaScore, getJabaScoreAfterPietonTurn) = $\Theta(1)$

BeansGame2. computeJabaScore()

```
for(int i = 0; i<=depth; i++) {
    int leftJabaScore = getBeans(str,str+i) + getJabaScoreAfterPietonTurn(str+i+1, end);
    int rightJabaScore = getBeans(end-i,end) + getJabaScoreAfterPietonTurn(str, end-i-1);

    int maxScore = Math.max(leftJabaScore,rightJabaScore);

    if(jabaScore == null || maxScore > jabaScore) {
        jabaScore = maxScore;
    }
}
```

$\sum_{i=0}^{N}$ max (getBenas, getJabaScoreAfterPietonTurn) = Θ(N), N=min (depthgame-1, j-i)

BeansGame2. getJabaScoreAfterPietonTurn()

```
if(beansTakenLeftSide >= beansTakenRightSide)
    return getJabaScore(str + (pietonMoveLeftSide + 1), end);
else
    return getJabaScore(str, end - (pietonMoveRightSide + 1));
```

getJabaScore = $\Theta(1)$

BeansGame2. getJabaScore ()

$\Theta(1)$

BeansGame2. computeBeanPilesSums()

```
for(int i=0; i<npiles; i++) {
    for(int j=i; j<npiles; j++) {
        if(i==j)
            beans[i][j] = piles[i];
        else
            beans[i][j] = beans[i][j-1] + piles[j];
    }
}
```

$\sum_{i=0}^{N-1}\sum_{J=i}^{N-1} 1 = \sum_{i=0}^{N-1}(N-1) = \frac{(N-1)(N-1+1)}{2} = \Theta(N^2)$, N = npiles-1

BeansGame2. getBeans ()

$\Theta(1)$

BeansGame2. computePietonMoves ()

```
for (int i = npiles-2; i>=0; i--) {
    for (int j=i+1; j<npiles; j++) {
        final int depth = Math.min(gamedepth-1, j-i);

        int beansInAllLeftSidePiles = getBeans(i,i+depth);
        int beansInAllRightSidePiles = getBeans(j-depth, j);

        int leftSideMoveOnLowerDepth = pietonMoves[i][i+(depth-1)];
        int leftSideMaxBeansOnLowerDepth = getBeans(i, i + leftSideMoveOnLowerDepth);

        if(leftSideMaxBeansOnLowerDepth >= beansInAllLeftSidePiles)
            pietonMoves[i][j] = leftSideMoveOnLowerDepth;
        else
            pietonMoves[i][j] = depth;

        int rightSideMoveOnLowerDepth = pietonMoves[j][j-(depth-1)];
        int rightSideMaxBeansOnLowerDepth = getBeans(j - rightSideMoveOnLowerDepth, j);

        if(rightSideMaxBeansOnLowerDepth >= beansInAllRightSidePiles)
            pietonMoves[j][i] = rightSideMoveOnLowerDepth;
        else
            pietonMoves[j][i] = depth;
    }
}
```

$$\sum_{i=N-1}^{0} \sum_{J=i+1}^{N} 1 \;=\; \sum_{i=N-1}^{0} N \;=\; \frac{N^2}{2} \;=\; \Theta(N^2)$$

N = npiles-1

Complexidade Espacial:

```
private final int npiles;
private final int gamedepth;
private final int[] piles;
private final boolean firstJaba;

private final int[][] beans;
private final int[][] pietonMoves;

private final int[][] jabaScore;
```

Complexidade Espacial:

Max(beans, pietonMoves, jabaScore) = $\Theta(N^2)$, N = npiles-1

Conclusão

Além da solução apresentada, exploramos e implementamos uma outra solução, que tem a seguinte função recursiva:

$$S(i,j,\alpha) = \begin{cases} 0 & \text{if } str > end \\ piles[i] & \text{if } i = j \\ \sum_{n=i}^{j} piles[n] - \min_{\{0 \le d \le depth\}} min(S(i+d+1,j,false), S(i,j-d-1,false)) & \text{if } \alpha = true \quad , depth = min(gamedepth-1, j-i) \\ \sum_{n=i}^{j} piles[n] - S(i+pL+1,j,true) & , pL = bestPietonMoveLeftSide(i,j) \quad \text{if } \alpha = false \cap \sum_{n=i}^{i+pL} piles[n] \le \sum_{n=j-pR}^{j} piles[n] \\ \sum_{n=i}^{j} piles[n] - S(str,end-pR-1,true) & , pR = bestPietonMoveRightSide(i,j) \quad \text{if } \alpha = false \cap \sum_{n=j-pR}^{j} piles[n] > \sum_{n=i}^{i+pL} piles[n] \end{cases}$$

Esta função retorna o score de ambos os jogadores. O score do Pieton caso (α=false) e o score do Jaba caso (α=true).

A Jaba em vez de maximizar o seu score após o turno Pieton, minimiza o score do Pieton após o seu turno.

Ambas as soluções possuem uma complexidade temporal de $\Theta(N^2.D)$, devido ao ciclo presente no caso da função recursiva "if α=true".

A complexidade espacial das duas soluções também é idêntica $\Theta(N^2)$

A implementação da solução anterior é mais simples porque apenas necessita de uma matriz que guarda o score da Jaba. A outra solução necessita de mais uma matriz que guarda o score do Pieton.

Optamos por submeter a solução que envolve maximizar o score da Jaba, por ter uma implementação mais clara e compacta.

Duas otimizações foram feitas, com o objetivo de diminuir a complexidade temporal da solução. Tem como custo espacial as matrizes "pietonMoves[npiles][npiles]" e "beans[npiles][npiles]".

A primeira otimização evita o cálculo repetido das escolhas do Pieton a direita e a esquerda para cada subsequência, usando programação dinâmica. A matriz pietonMoves guarda nas posições superiores da sua diagonal as escolhas a esquerda e nas posições inferiores as escolhas a direita.

A segunda otimização evita o cálculo repetido da soma das pilhas pertencentes a cada subsequência, usando programação dinâmica.

Como possível melhoramento, seria interessante identificar os sub-problemas estritamente necessários no cálculo do problema principal. Desta forma, talvez algumas iterações poderiam ser removidas neste ciclo.

```
for(int i=npiles-2; i>=0; i--)
    for(int j=i+1; j<npiles; j++)
        jabaScore[i][j] = computeJabaScore(i,j);
```

Código

Classe Main

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

public class Main {

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

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

        int ngames = Integer.parseInt(in.readLine());

        BeansGame2[] games = new BeansGame2[ngames];

        for(int i=0; i<ngames; i++) {
            String[] line1 = in.readLine().split(" ");
            int npiles = Integer.parseInt(line1[0]);
            int gamedepth = Integer.parseInt(line1[1]);

            String[] line2 = in.readLine().split(" ");
            int[] piles = new int[npiles];
            for(int j=0; j < npiles; j++) {
                piles[j] = Integer.parseInt(line2[j]);
            }

            boolean firstJaba = in.readLine().equals("Jaba");

            games[i] = new BeansGame2(npiles,gamedepth,piles,firstJaba);
        }

        for (BeansGame2 game : games) {
            System.out.println(game.solve());
        }
    }
}
```

Classe BeansGame2

```
import java.util.Optional;

public class BeansGame2 {

    private final int npiles;
    private final int gamedepth;
    private final int[] piles;
    private final boolean firstJaba;

    private final int[][] beans;
    private final int[][] pietonMoves;

    private final int[][] jabaScore;

    public BeansGame2(int npiles, int gamedepth, int[] piles, boolean firstJaba) {
        this.npiles = npiles;
        this.gamedepth = gamedepth;
        this.piles = piles;
        this.firstJaba = firstJaba;

        this.beans = new int[npiles][npiles];
        this.pietonMoves = new int[npiles][npiles];

        this.jabaScore = new int[npiles][npiles];
    }


public int solve() {
    computeBeanPilesSums();
    if(gamedepth > 1) computePietonMoves();

    for (int i=0; i<npiles; i++)
        jabaScore[i][i] =  piles[i];

    for(int i=npiles-2; i>=0; i--)
        for(int j=i+1; j<npiles; j++)
            jabaScore[i][j] = computeJabaScore(i,j);

    if(firstJaba)
        return getJabaScore(0, npiles - 1);
    else
        return getJabaScoreAfterPietonTurn(0, npiles - 1);
}

private int computeJabaScore(final int str, final int end) {
    int depth = Math.min(gamedepth-1, end-str);

    Integer jabaScore = null;
    for(int i = 0; i<=depth; i++) {
        int leftJabaScore = getBeans(str,str+i) + getJabaScoreAfterPietonTurn(str+i+1, end);
        int rightJabaScore = getBeans(end-i,end) + getJabaScoreAfterPietonTurn(str, end-i-1);

        int maxScore = Math.max(leftJabaScore,rightJabaScore);

        if(jabaScore == null || maxScore > jabaScore) {
            jabaScore = maxScore;
        }
    }

    return Optional.ofNullable(jabaScore).orElse(0);
}
```

```
private int getJabaScoreAfterPietonTurn(final int str, final int end) {
    if(str>=end) return 0;

    final int pietonMoveLeftSide = pietonMoves[str][end];
    final int pietonMoveRightSide = pietonMoves[end][str];

    final int beansTakenLeftSide = getBeans(str,str+pietonMoveLeftSide);
    final int beansTakenRightSide = getBeans(end-pietonMoveRightSide, end);

    if(beansTakenLeftSide >= beansTakenRightSide)
        return getJabaScore(str + (pietonMoveLeftSide + 1), end);
    else
        return getJabaScore(str, end - (pietonMoveRightSide + 1));
}

private int getJabaScore(int str, int end) {
    if(str>end || end>(npiles-1) || str<0)
        return 0;
    else
        return jabaScore[str][end];
}

private void computeBeanPilesSums() {
    for(int i=0; i<npiles; i++) {
        for(int j=i; j<npiles; j++) {
            if(i==j)
                beans[i][j] = piles[i];
            else
                beans[i][j] = beans[i][j-1] + piles[j];
        }
    }
}
```

```
private int getBeans(int str, int end) {
    if(str>end || end>(npiles-1) || str<0) return 0;
    return beans[str][end];
}

private void computePietonMoves() {
    for (int i = npiles-2; i>=0; i--) {
        for (int j=i+1; j<npiles; j++) {
            final int depth = Math.min(gamedepth-1, j-i);

            int beansInAllLeftSidePiles = getBeans(i,i+depth);
            int beansInAllRightSidePiles = getBeans(j-depth, j);

            int leftSideMoveOnLowerDepth = pietonMoves[i][i+(depth-1)];
            int leftSideMaxBeansOnLowerDepth = getBeans(i, i + leftSideMoveOnLowerDepth);

            if(leftSideMaxBeansOnLowerDepth >= beansInAllLeftSidePiles)
                pietonMoves[i][j] = leftSideMoveOnLowerDepth;
            else
                pietonMoves[i][j] = depth;

            int rightSideMoveOnLowerDepth = pietonMoves[j][j-(depth-1)];
            int rightSideMaxBeansOnLowerDepth = getBeans(j - rightSideMoveOnLowerDepth, j);

            if(rightSideMaxBeansOnLowerDepth >= beansInAllRightSidePiles)
                pietonMoves[j][i] = rightSideMoveOnLowerDepth;
            else
                pietonMoves[j][i] = depth;
        }
    }
}
```