Fundamentos de Projeto e Análise de Algoritmos

Trabalho prático em grupo – Parte I

Grupo: Davi Santos Silva; Ian Marcel de Campos Ferreira; Kleyann Martins Barros; Rafael Augusto Vieira de Almeida

Instituto de Informática e Ciências Exatas– Pontifícia Universidade de Minas Gerais (PUC MINAS)

Belo Horizonte – MG – Brasil

1. Introdução

O problema da mochila de uma forma simplificada, consiste em inserir uma certa quantidade de itens, cada um deste com um peso e valor, numa mochila com uma capacidade definida. Existem algumas formas de resolver esse problema. Neste trabalho, usou-se as soluções de força bruta e método guloso.

2. Implementação

2.1 Classe Item Mochila

A classe foi criada para auxiliar nas soluções. Tem valor, peso, razão além de um método toString para impressão dos dados.

```java
public class ItemMochila {

    private int valor;
    private int peso;
    private double razao;

    public int getValor() {
        return valor;
    }

    public int getPeso() {
        return peso;
    }

    public void setPeso(int peso) {
        this.peso = peso;
    }
```

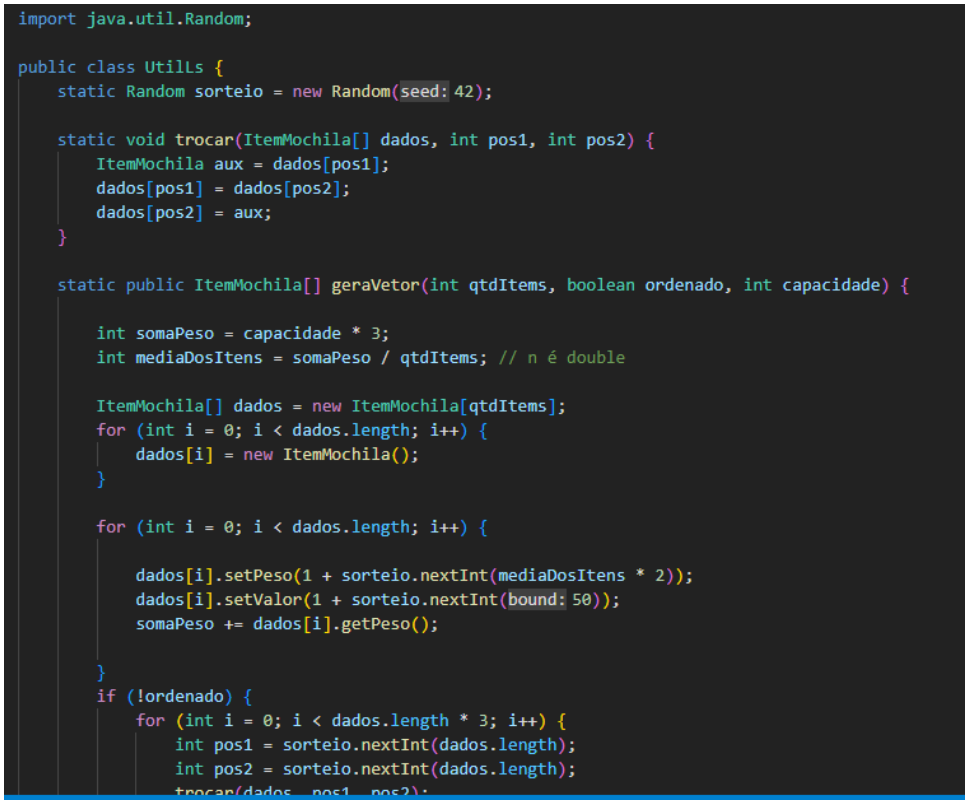
```java
public void setValor(int valor) {
    this.valor = valor;
}

public double getRazao() {
    this.razao = (double) this.valor / this.peso;
    return this.razao;
}

public void setRazao(double razao) {
    this.razao = razao;
}

public String toString() {
    return "peso : " + peso + " valor: " + valor + " razao: " + this.razao;
}
```

2.2 Classe Utils

Está classe contém os métodos de ordenação, sendo eles o quicksort, bubbleSort e o bubbleSortInvertido(ordena o vetor de itens pela razão, do maior para o menor).

Nesta classe, um dos métodos mais importantes é o gerarVetor(), este recebe uma quantidade de itens a criar, gerando um vetor com pesos e valores aleatórios, usando os

métodos da classe Random do Java. Outro parâmetro recebido é a capacidade máxima. A soma dos itens gerados deve ser de aproximadamente três vezes que a capacidade máxima recebida.



```java
import java.util.Random;

public class UtilLs {
    static Random sorteio = new Random(seed: 42);

    static void trocar(ItemMochila[] dados, int pos1, int pos2) {
        ItemMochila aux = dados[pos1];
        dados[pos1] = dados[pos2];
        dados[pos2] = aux;
    }

    static public ItemMochila[] geraVetor(int qtdItems, boolean ordenado, int capacidade) {

        int somaPeso = capacidade * 3;
        int mediaDosItens = somaPeso / qtdItems; // n é double

        ItemMochila[] dados = new ItemMochila[qtdItems];
        for (int i = 0; i < dados.length; i++) {
            dados[i] = new ItemMochila();
        }

        for (int i = 0; i < dados.length; i++) {

            dados[i].setPeso(1 + sorteio.nextInt(mediaDosItens * 2));
            dados[i].setValor(1 + sorteio.nextInt(bound: 50));
            somaPeso += dados[i].getPeso();

        }
        if (!ordenado) {
            for (int i = 0; i < dados.length * 3; i++) {
                int pos1 = sorteio.nextInt(dados.length);
                int pos2 = sorteio.nextInt(dados.length);
                trocar(dados, pos1, pos2);
```

```
33                  trocar(dados, pos1, pos2);
34              }
35          }
36          return dados;
37      }
38
39      static public void bolha(ItemMochila[] dados) {
40          for (int i = dados.length - 1; i > 0; i--) {
41              for (int j = 0; j < i; j++) {
42                  if (dados[j].getRazao() > dados[j + 1].getRazao())
43                      trocar(dados, j, j + 1);
44              }
45
46          }
47      }
48
49      static public void bolhaInvertido(ItemMochila[] dados) {
50          for (int i = dados.length - 1; i > 0; i--) {
51              for (int j = 0; j < i; j++) {
52                  if (dados[j].getRazao() < dados[j + 1].getRazao())
53                      trocar(dados, j, j + 1);
54              }
55
56          }
57      }
58
59      static public int particao(ItemMochila[] dados, int inicio, int fim) {
60          int posicao = inicio - 1;
61          double pivot = dados[fim].getRazao();
62          for (int i = inicio; i < fim; i++) {
63              if (dados[i].getRazao() > pivot) {
64                  posicao++;
65                  trocar(dados, posicao, i);
```

```
59      static public int particao(ItemMochila[] dados, int inicio, int fim) {
60          int posicao = inicio - 1;
61          double pivot = dados[fim].getRazao();
62          for (int i = inicio; i < fim; i++) {
63              if (dados[i].getRazao() > pivot) {
64                  posicao++;
65                  trocar(dados, posicao, i);
66              }
67          }
68          posicao++;
69          trocar(dados, posicao, fim);
70          return posicao;
71      }
72
73      static public void quicksort(ItemMochila[] dados, int inicio, int fim) {
74          if (inicio >= fim)
75              return;
76          else {
77              int particao = particao(dados, inicio, fim);
78              quicksort(dados, inicio, particao - 1);
79              quicksort(dados, particao + 1, fim);
80          }
81      }
82
83  }
```

3. Força Bruta

A força bruta é uma abordagem direta para resolver um problema, geralmente baseada diretamente em o enunciado do problema e as definições dos conceitos envolvidos.

Para o problema da mochila, o algoritmo compara todas as possibilidades de preenchimento da mochila que não ultrapassem o peso máximo estipulado.

3.1 Implementação

Ele começa comparando dois valores para saber qual o maior. Usa recursividade comparando cada posição dos pesos, lembrando que cada item tem um peso, no qual ele atribuiu na main. Todos os pesos dos itens foram gerados através do métodos gerarVetor(). Para cada chamada recursiva da função, segue uma mesma lógica dos valores que foram atribuídos ao item do vetor.

```java
public class ForcaBruta {


    static int valorMaior(int num1, int num2) {
        return (num1 > num2) ? num1 : num2;
    }

    static int valorMaior2(int num1, int num2) {
        if (num1 > num2) {
            return num1;
        } else {
            return num2;
        }
    }

    static int forcaBruta(int capacidade, ItemMochila vetorItem[], int tam) {

        if (tam == 0 || capacidade == 0) {
            return 0;
        }

        if (vetorItem[tam - 1].getPeso() > capacidade) {
            return forcaBruta(capacidade,vetorItem, tam - 1);
        }

        else {
            return valorMaior(vetorItem[tam - 1].getValor()
                    + forcaBruta(capacidade - vetorItem[tam - 1].getPeso(), vetorItem,
                    tam - 1),
                    forcaBruta(capacidade, vetorItem, tam - 1));
        }
```

3.2 Resultados para o Força Bruta em 4s

Como o força bruta é bem mais lento, ele não conseguiu criar mochilas com muitos itens em 4 Segundos. Neste período criou com o máximo de 34 itens. Será mostrado na foto da main.

4. Método Guloso

A ideia desse método é ordenar os itens pela razão valor/peso, sendo está divisão será o critério de prioridade para inserir esses itens na mochila, até sua capacidade ser atingida. Quem faz a organização por razão é o bolhaInvertido() ou quickSort() da classe Utils, como mostra a foto abaixo nos resultados da main

5. Resultados

5.1 Letra A

Como descrito no tópico 3.2, realizou-se teste com as letras A e B do trabalho, na qual o maior valor de itens criados e resolvidos foi 34 usando o forcaBruta.

```
public class MainPrincipal {
    Run | Debug
    public static void main(String args[]) {

    long tempoExecucao =0;
    int capacidade=200;
      int quantItems=5;
    ItemMochila[] item = new ItemMochila[quantItems];
    mochila m = new mochila(capacidade);
    int tam;


            //instanciando
            for (int i = 0; i < item.length; i++) {
                item[i] = new ItemMochila();
            }



     //Letra A
    while(tempoExecucao<4000){
     System.out.println(" mochila com capacidade :" + capacidade + " e :" + quantItems + " itens");
     tempoExecucao = System.currentTimeMillis();


     item=UtilLs.geraVetor(quantItems, ordenado: false, capacidade);
     tam = item.length;

     System.out.println(ForcaBruta.forcaBruta(capacidade, item, tam));
```

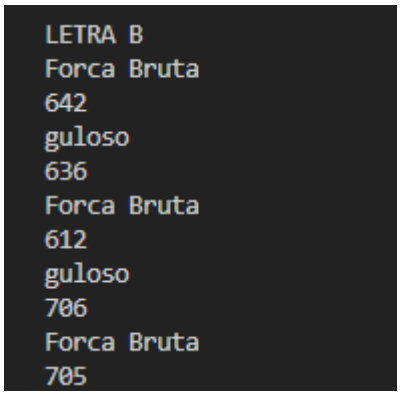
```
    tempoExecucao = (System.currentTimeMillis() - tempoExecucao);
    System.out.println("tempo em milisegundos =" + tempoExecucao);
    quantItems++;
   }

    System.out.println(x: "LETRA B");
    for (int i = 0; i < 5; i++) {
       System.out.println(x: "Forca Bruta");
        item=UtilLs.geraVetor((quantItems-1), ordenado: false, capacidade);//com essa qtd 34
        //fazendo o forca Bruta
        tam=item.length;
        System.out.println(ForcaBruta.forcaBruta(capacidade, item, tam));

        //fazendo o guloso
        System.out.println(x: "guloso");
     //Utills.bolhaInvertido(item);
      UtilLs.quicksort(item, inicio: 0, tam-1);
     for (int index = 0; index < item.length; index++) {
     m.adicionar(item[index]);
     }
     System.out.println(m.getValorAtual());


    }
   }
}
```

```
597
tempo em milisegundos =3381
 mochila com capacidade :200 e :33 itens
521
tempo em milisegundos =1570
 mochila com capacidade :200 e :34 itens
560
tempo em milisegundos =4823
```

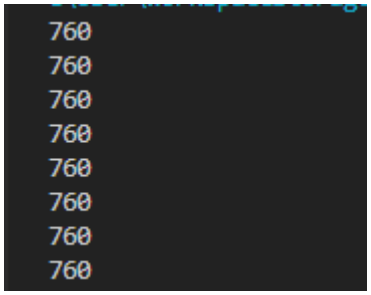
5.2 Letra B

Foi comparado a soma dos valores dos itens que estão na mochila que o Algoritmo Guloso e o Força Bruta geravam para uma mochila com quantidade de itens de 34. Não foi encontrado nas iterações resultados iguais. Mas houve valores muito próximos, como 760 e 765.

Cada linha representa a soma dos valores dos itens que foram colocados na mochila em cada iteração do Algoritmo Guloso (como foram 500 não foi possível colocar o print completo devido a quantidade de caracteres).



Os prints abaixo foram os da classe de teste do Algoritmo Guloso (desconsiderar) e do Força Bruta.



Cada linha representa a soma dos valores dos itens que foram colocados na mochila em cada iteração do Força Bruta.