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. Para este trabalho, foram usadas as soluções de força bruta e método guloso.

2. Implementação

2.1 Classe ItemMochila

A classe foi criada para auxiliar nas soluções. Tem valor, peso e dois métodos, um que retorna a razão e outro toString para impressão dos dados.

```java
public class ItemMochila {

    private int valor;
    private int peso;

    public int getValor() { return valor; }

    public int getPeso() { return peso; }

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

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

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

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

}
```

2.2 Classe Utils

Nesta classe, há um dos métodos mais importantes, o geraVetor(), 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 a capacidade máxima recebida.

```java
public  class Utils {
    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( bound: 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);
            }
        }
        return dados;
    }
}
```

3. Força Bruta

A força bruta é uma abordagem direta para resolver um problema, geralmente baseada diretamente no 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. 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 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: tam - 1);
        }

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

}
```

4. Método Guloso

A ideia desse método é ordenar os itens pela razão valor/peso, sendo este o critério de prioridade para inserir esses itens na mochila, até sua capacidade ser atingida. A ordenação por razão é feita através do quicksort.

```java
public class Guloso {
    static public int Guloso(ItemMochila[] itens, int capacidade){
        quicksort(itens, inicio: 0, fim: itens.length-1);
        Mochila m = new Mochila(capacidade);
        for (int index = 0; index < itens.length; index++) {
            m.add(itens[index]);
        }
        return m.getValorAtual();
    }
```

```java
static public void quicksort(ItemMochila[] dados, int inicio, int fim) {
    if (inicio >= fim)
        return;
    else {
        int particao = particao(dados, inicio, fim);
        quicksort(dados, inicio, fim: particao - 1);
        quicksort(dados, inicio: particao + 1, fim);
    }
}
```

5. Resultados

5.1 Letra A

O teste foi realizado com uma mochila de capacidade 200, onde o maior vetor resolvido em um tempo menor que 4 segundos foi um vetor com 34 itens.

```java
public class MainPrincipal {
    public static void main(String args[]) {
        long tempoExecucao = 0;
        int capacidade = 240;
        int quantItems = 5;
        ItemMochila[] itens;


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


            itens = Utils.geraVetor(quantItems, ordenado: false, capacidade);

            System.out.println(ForcaBruta.forcaBruta(capacidade, itens, itens.length));



            tempoExecucao = (System.currentTimeMillis() - tempoExecucao);
            System.out.println("Tempo: " + tempoExecucao + "ms");
            quantItems++;
        }
```

5.2 Letra B

Foram comparados os valores resultantes dos Algoritmos Guloso e Força Bruta para um vetor com 34 itens e uma mochila com capacidade 200. No final foram encontrados nas 500 iterações 215 resultados iguais.

```
        System.out.println("LETRA B");

        int iguais = 0;

        for (int i = 0; i < 500; i++) {
            System.out.println("Iteração: " + (i+1));
            itens = Utils.geraVetor((quantItems-1),  ordenado: false, capacidade);

            int fb = ForcaBruta.forcaBruta(capacidade, itens, itens.length);
            System.out.println("Forca Bruta: " + fb);

            int g = Guloso.Guloso(itens, capacidade);
            System.out.println("Guloso: " + g);

            if(fb == g) iguais++;
        }
        System.out.println("\n------------\n IGUAIS: " + iguais);
    }
}
```

Resultados extras com diferentes capacidades:

Capacidade	Itens no Vetor	Resultados Iguais
200	34	215
210	33	203
220	31	219
230	31	236
240	34	201
250	33	214