

Case - Otimização de logística portuária

Prof- Alexandre Montanha



Diego Vinicius - 12523158675

Jhonattan Mariano - 824120739

João Gobbi - 824145710

Matheus Yusuke - 825146805

Nicolas Trindade - 824135758

Estrutura de Dados e Análise de Algoritmos

3-Semestre/2025

Descrição do Problema:

Você é responsável por projetar um algoritmo que otimiza o descarregamento de containers de um navio para armazéns e posterior transporte por caminhões. Cada container possui um volume conhecido, porém os volumes são variados.

Seu algoritmo deve receber como entrada:

- Uma lista com os volumes dos containers a serem descarregados.
- A capacidade fixa de armazenamento dos armazéns (Carga X).
- A capacidade fixa de transporte dos caminhões (Carga Y).

O objetivo é calcular:

- O número mínimo de armazéns necessários para armazenar todos os containers.
- O número mínimo de caminhões necessários para transportar a carga total armazenada.

1- Identificação da complexidade do problema: Classifique o problema em P, NP e NP-Completo, explicando sua decisão com argumentos teóricos.

Classe: NP-Completo

O problema de alocação dos containers nos armazéns corresponde ao conhecido Bin Packing Problem, uma variação do problema da mochila. Esse problema é **NP-Completo**, pois:

Está na classe **NP**: uma solução pode ser verificada

É tão difícil quanto qualquer outro problema em **NP** (se resolvêssemos isso eficientemente, poderíamos resolver outros problemas **NP** rapidamente também).

Embora a segunda parte do problema (dividir as cargas dos armazéns para os caminhões) pareça simples, ela também representa um segundo nível de empacotamento, o que a torna equivalente em complexidade.

Portanto, o problema geral proposto (minimizar armazéns e caminhões com restrições de capacidade) é corretamente classificado como **NP-Completo**.

2. Estratégias de resolução sugeridas: Indique qual técnica algorítmica é mais apropriada para resolver o problema (programação dinâmica, gulosa, força bruta, etc.) e justifique brevemente sua escolha.

Utilizamos uma estratégia gulosa baseada no algoritmo **First Fit Decreasing (FFD)** para a alocação dos containers nos armazéns e também para a distribuição dos armazéns nos caminhões.

Como o problema é NP-Completo, algoritmos exatos são inviáveis para grandes volumes de dados. Por isso, usamos heurísticas que encontram soluções boas rapidamente.

Containers → Armazéns:

Ordenamos os containers do maior para o menor volume e os alocamos no primeiro armazém onde ainda couber (First Fit Decreasing).

Armazéns → Caminhões:

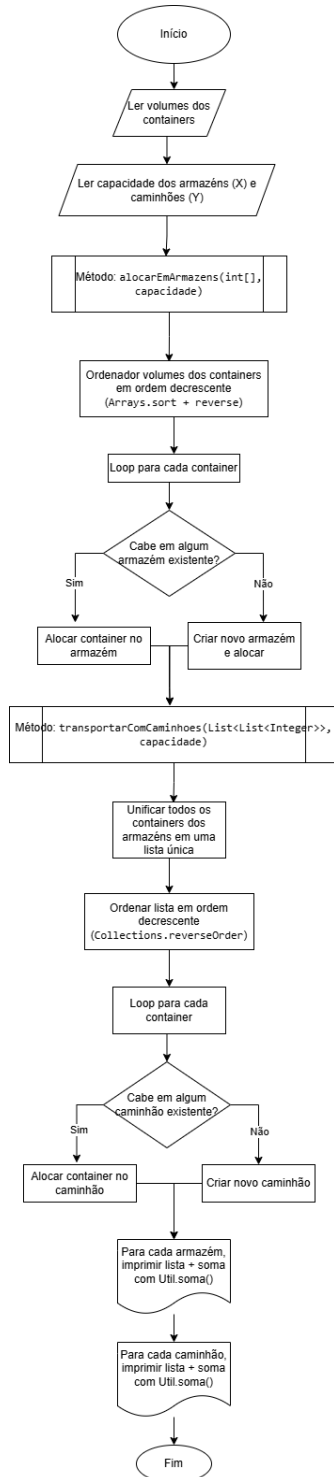
Aplicamos a mesma lógica: Ordenamos os containers do maior para o menor volume, e em ordem alocamos no primeiro caminhão disponível com espaço restante. Se não houver, abrimos um novo caminhão.

Essa abordagem mantém o algoritmo rápido e eficiente, com complexidade $O(n \log n)$ por causa da ordenação, e oferece soluções com boa qualidade para o problema.

3- Desenvolvimento do algoritmo: Linguagem escolhida: Java

Github: <https://github.com/Japu431/Algoritmos-e-Estruturas-de-Dados>

Fluxograma:



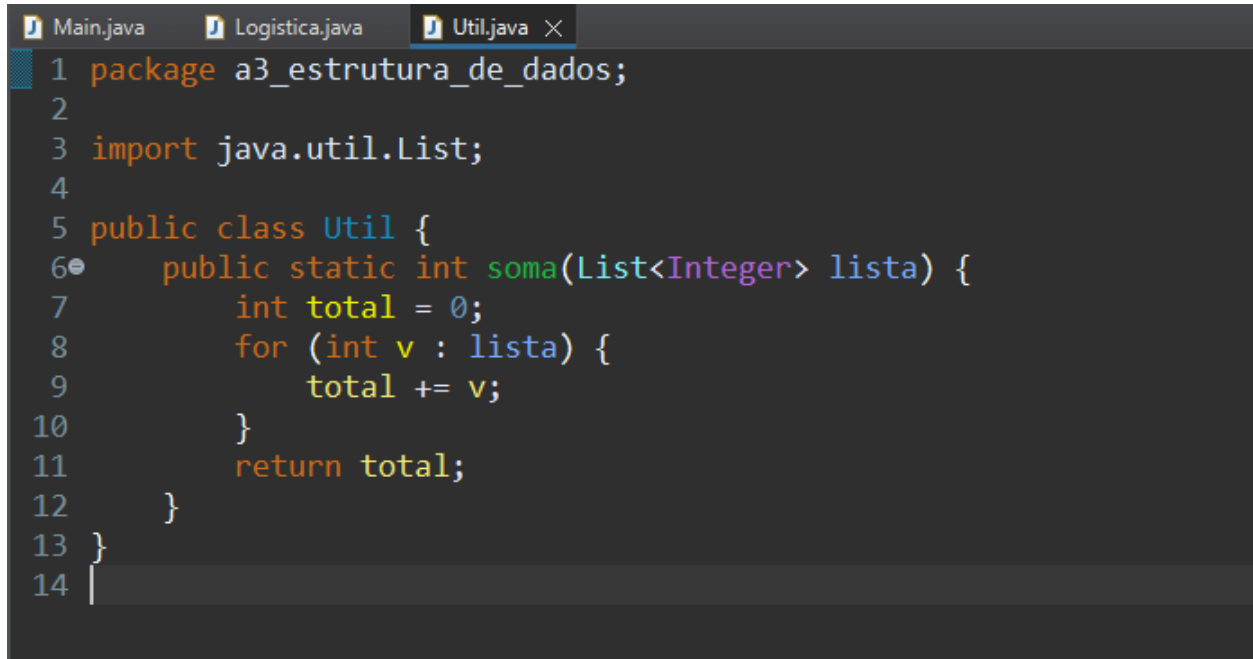
Arquivo Main.java:

```
1 package a3_estrutura_de_dados;
2
3 import java.util.*;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8
9         // Entrada dos volumes
10        System.out.println("Digite os volumes dos containers:");
11        String[] entrada = scanner.nextLine().split(" ");
12        int[] containers = new int[entrada.length];
13        for (int i = 0; i < entrada.length; i++) {
14            containers[i] = Integer.parseInt(entrada[i]);
15        }
16
17        // Capacidade dos armazéns e caminhões
18        System.out.print("Capacidade de cada armazém (X): ");
19        int capacidadeArmazem = scanner.nextInt();
20
21        System.out.print("Capacidade de cada caminhão (Y): ");
22        int capacidadeCaminhao = scanner.nextInt();
23
24        // Processamento
25        List<List<Integer>> armazens = Logistica.alocarEmArmazens(containers, capacidadeArmazem);
26        List<List<Integer>> caminhoes = Logistica.transportarComCaminhoes(armazens, capacidadeCaminhao);
27
28        // Saída dos armazéns
29        int totalArmazens = 0;
30        System.out.println("\nNúmero mínimo de armazéns necessários: " + armazens.size());
31        for (int i = 0; i < armazens.size(); i++) {
32            List<Integer> a = armazens.get(i);
33            int soma = Util.soma(a);
34            totalArmazens += soma;
35            System.out.println("Armazém " + (i + 1) + ": " + a + " => " + soma + "m³");
36        }
37        System.out.println("Total armazenado: " + totalArmazens + "m³");
38
39        // Saída dos caminhões
40        int totalCaminhoes = 0;
41        System.out.println("\nNúmero mínimo de caminhões necessários: " + caminhoes.size());
42        for (int i = 0; i < caminhoes.size(); i++) {
43            List<Integer> c = caminhoes.get(i);
44            int soma = Util.soma(c);
45            totalCaminhoes += soma;
46            System.out.println("Caminhão " + (i + 1) + ": " + c + " => " + soma + "m³");
47        }
48        System.out.println("Total transportado: " + totalCaminhoes + "m³");
49
50        scanner.close();
51    }
52 }
```

Arquivo Logistica.java

```
1 package a3_estrutura_de_dados;
2 import java.util.*;
3
4 public class Logistica {
5
6     // Alocação nos armazéns com First Fit Decreasing
7     public static List<List<Integer>> alocarEmArmazens(int[] containers, int capacidade) {
8         Arrays.sort(containers);
9         List<List<Integer>> armazens = new ArrayList<>();
10        List<Integer> capacidades = new ArrayList<>();
11
12        for (int i = containers.length - 1; i >= 0; i--) {
13            int volume = containers[i];
14            boolean alocado = false;
15
16            for (int j = 0; j < armazens.size(); j++) {
17                if (capacidades.get(j) + volume <= capacidade) {
18                    armazens.get(j).add(volume);
19                    capacidades.set(j, capacidades.get(j) + volume);
20                    alocado = true;
21                    break;
22                }
23            }
24
25            if (!alocado) {
26                List<Integer> novo = new ArrayList<>();
27                novo.add(volume);
28                armazens.add(novo);
29                capacidades.add(volume);
30            }
31        }
32        return armazens;
33    }
34
35    // Transporte respeitando containers indivisíveis
36    public static List<List<Integer>> transportarComCaminhoes(List<List<Integer>> armazens, int capCaminhao) {
37        List<Integer> todosContainers = new ArrayList<>();
38        for (List<Integer> a : armazens) {
39            todosContainers.addAll(a);
40        }
41
42        todosContainers.sort(Collections.reverseOrder());
43
44        List<List<Integer>> caminheiros = new ArrayList<>();
45        List<Integer> capacidades = new ArrayList<>();
46
47        for (int container : todosContainers) {
48            boolean alocado = false;
49            for (int i = 0; i < caminheiros.size(); i++) {
50                if (capacidades.get(i) + container <= capCaminhao) {
51                    caminheiros.get(i).add(container);
52                    capacidades.set(i, capacidades.get(i) + container);
53                    alocado = true;
54                    break;
55                }
56            }
57
58            if (!alocado) {
59                List<Integer> novo = new ArrayList<>();
60                novo.add(container);
61                caminheiros.add(novo);
62                capacidades.add(container);
63            }
64        }
65
66        return caminheiros;
67    }
68 }
```

Arquivo Util.java



```
1 package a3_estrutura_de_dados;
2
3 import java.util.List;
4
5 public class Util {
6     public static int soma(List<Integer> lista) {
7         int total = 0;
8         for (int v : lista) {
9             total += v;
10        }
11        return total;
12    }
13 }
14 |
```

Programa em execução:

Dados de entrada:

- Containers com volumes: [40, 50, 60, 20, 30, 80, 70]
- Capacidade do armazém: 100
- Capacidade dos caminhões: 150

Saida:

Numero minimo de armazens necessarios: 4

Armazém 1: [80, 20] => 100m³

Armazém 2: [70, 30] => 100m³

Armazém 3: [60, 40] => 100m³

Armazém 4: [50] => 50m³

Total armazenado: 350m³

Numero minimo de Caminhões necessarios: 3

Caminhão 1: [80, 70] => 150m³

Caminhão 2: [60, 50, 40] => 150m³

Caminhão 3: [30, 20] => 50m³

Total transportado: 350m³

4- Análise da complexidade: Utilizamos o Algoritmo Guloso de Aproximação — First Fit Decreasing (FFD)

Nosso algoritmo possui **duas etapas** principais:

- Alocação dos containers nos armazéns
- Distribuição dos armazéns para os caminhões

Em ambas as etapas, aplicamos o algoritmo **First Fit Decreasing (FFD)**, uma heurística gulosa, que exige ordenação e uma varredura sequencial com decisões locais.

Vamos analisar a complexidade de cada etapa usando **notação Big-O**:

Alocação dos containers nos armazéns (First Fit Decreasing)

Ordenação dos containers:

Usamos `Arrays.sort()` seguido de inversão manual para obter ordem decrescente.

Isso tem custo de:

$O(n \log n)$ (para ordenar n containers)

Distribuição dos containers:

Para cada container, percorremos no pior caso todos os armazéns já criados para tentar alocar.

No pior cenário (container por armazém), teremos no máximo n armazéns.

Isso resulta em: **$O(n^2)$** (para alocar os n containers em armazéns)

Total para esta etapa:

$O(n \log n + n^2) \rightarrow O(n^2)$

Alocação dos containers nos caminhões (First Fit Decreasing)

- m armazéns foram gerados na etapa anterior.
- **Calculo do volume de cada armazém:**
 $O(m)$
- **Ordenação dos armazéns por volume (decrescente):**
 $O(m \log m)$
- **Aplicação do First Fit Decreasing:**
Cada armazém é alocado no primeiro caminhão com espaço suficiente.
No pior caso, percorre todos os caminhões abertos $\rightarrow O(m^2)$

Total para esta etapa:

$O(m \log m + m^2) \rightarrow O(m^2)$

Conclusão:

Etapa	Complexidade
Ordenar containers	$O(n \log n)$
Alocar containers (First Fit)	$O(n^2)$
Ordenar armazéns	$O(m \log m)$
Alocar em caminhões (First Fit)	$O(m^2)$

Ambas as fases do algoritmo utilizam o First Fit Decreasing (FFD), uma heurística gulosa que realiza uma ordenação inicial seguida de alocação sequencial.

A primeira fase (containers \rightarrow armazéns) e a segunda fase (armazéns \rightarrow caminhões) possuem a mesma estrutura de custo: **ordenar ($O(n \log n)$) + alocar ($O(n^2)$)**, resultando em **complexidade final $O(n^2)$** .

A escolha dessa estratégia permite um bom equilíbrio entre desempenho e simplicidade, sendo eficaz para problemas práticos de logística.