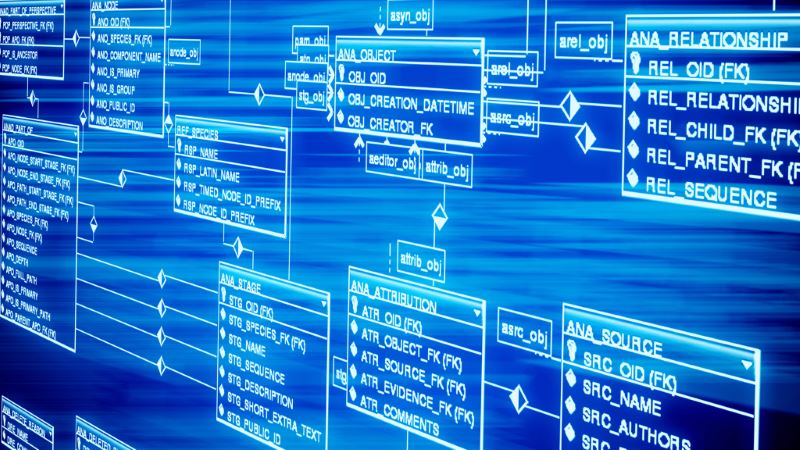
**Relatório AED**



**ISMAT - Instituto Superior Manuel Teixeira Gomes**

Engenharia Informática



Ano Letivo: 2024/2025

Curso: Engenharia Informática

Disciplina: Algoritmia e Estrutura de Dados

Professor: Tiago Candeias

Feito por: João Filipe Pinheiro Carriço da Silva | a22304361

Índice

[Introdução 3](#_Toc201509814)

[Abordagem 4](#_Toc201509815)

[Diagrama de classes 6](#_Toc201509816)

[Pseudocódigo do algoritmo desenvolvido. 11](#_Toc201509817)

[Conclusão 17](#_Toc201509818)

[Bibliografia 18](#_Toc201509819)

# Introdução

Este projeto, desenvolvido em Java, tem como objetivo a criação de um sistema de recomendação de receitas, com base na disponibilidade de ingredientes fornecidos pelo utilizador. A lógica do sistema assenta na utilização de estruturas de dados eficientes e de grafos para modelar as ligações entre ingredientes, permitindo identificar receitas completas e sugerir alternativas mesmo quando há ingredientes em falta.

A aplicação não utiliza apenas estruturas tradicionais como listas e conjuntos, mas também implementa uma **tabela de dispersão personalizada (HashTable)** para mapear relações entre ingredientes, e um **grafo orientado (RecipeGraph)** que representa a coocorrência de ingredientes nas receitas. Estes elementos são fundamentais para identificar conexões e possibilidades de substituição entre ingredientes.

Cada receita é representada por um nome e um conjunto de ingredientes. O sistema permite ao utilizador inserir os ingredientes que tem disponíveis, sendo posteriormente realizada uma análise para determinar:

* **Receitas completas**, onde todos os ingredientes estão disponíveis;
* **Receitas incompletas**, que estão quase completas (com no máximo dois ingredientes em falta).

A aplicação segue uma arquitetura modular, com classes responsáveis pela leitura de dados, armazenamento de receitas, construção do grafo e lógica de recomendação.

A entrada e saída da aplicação foram desenhadas para ser simples e clara, permitindo a sua utilização direta por utilizadores ou a integração com sistemas inteligentes e testes automáticos.

# Abordagem

Para resolver o problema proposto — recomendar receitas com base nos ingredientes disponíveis — foi adotada uma abordagem modular, orientada a objetos, utilizando a linguagem Java. A resolução passa por três componentes principais: **estruturação dos dados**, **representação das relações entre ingredientes** e **lógica de recomendação**.

A primeira etapa consiste na leitura e estruturação dos dados do ficheiro receitas.txt. Esta funcionalidade é implementada na classe RecipeDataLoader, que lê cada linha do ficheiro, extrai o nome da receita e os seus ingredientes, criando objetos Recipe e Ingredient. Cada Recipe contém um conjunto de Ingredient, permitindo a comparação direta entre os ingredientes disponíveis e os exigidos por cada receita.

A segunda etapa foca-se na modelação das ligações entre ingredientes. Esta relação é importante para a recomendação de receitas incompletas, permitindo sugerir ingredientes próximos ou que frequentemente aparecem juntos em receitas. Para isso, foi implementada a classe RecipeGraph, que utiliza uma tabela de dispersão personalizada (HashTable<Ingredient, Ingredient>) para armazenar o grafo de ligações entre ingredientes. Dois ingredientes são ligados se aparecem juntos em pelo menos uma receita.

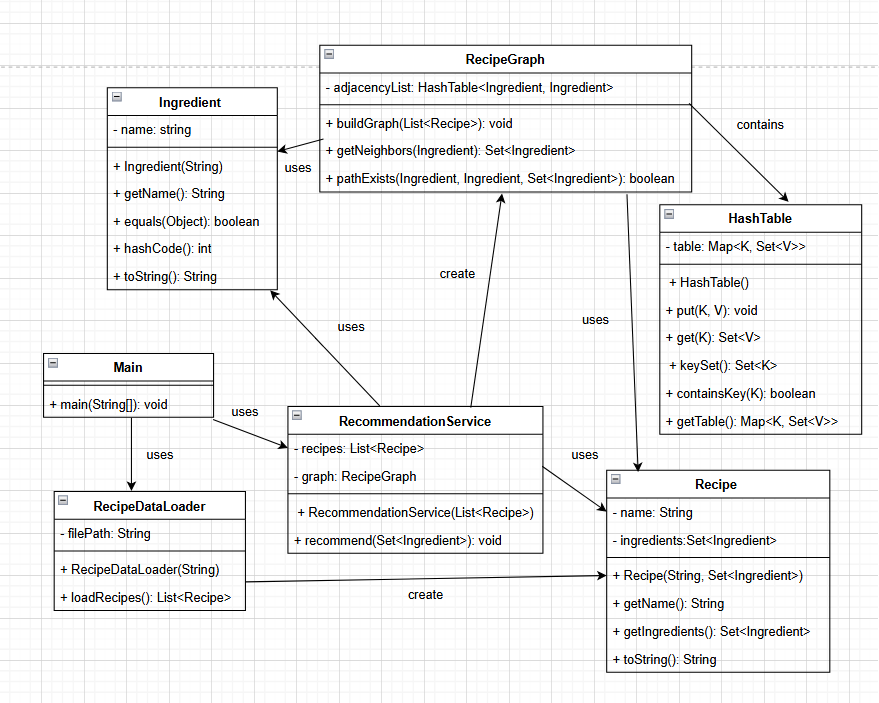
A terceira e última etapa corresponde à lógica de recomendação, centralizada na classe RecommendationService. Esta classe analisa o conjunto de ingredientes disponíveis e determina:

* Quais as receitas completas que o utilizador pode preparar imediatamente;
* Quais as receitas incompletas (com até dois ingredientes em falta), mas com potencial de execução caso sejam adquiridos os ingredientes em falta.

O algoritmo de verificação para receitas incompletas inclui também uma validação por conectividade no grafo: é verificado se existe caminho entre os ingredientes disponíveis e os ingredientes em falta, aumentando a inteligência da recomendação.

A aplicação é lançada a partir da classe Main, que trata a entrada dos ingredientes por parte do utilizador, invoca os serviços e exibe os resultados de forma organizada.

# Diagrama de classes



**Main**

* **Responsabilidade**: Classe de entrada do programa. Contém o método main, que coordena a execução da aplicação.
* **Relações**:
  + **Usa** a classe RecipeDataLoader para carregar as receitas a partir de um ficheiro.
  + **Cria** um objeto da classe RecommendationService e invoca o método de recomendação com base nos ingredientes disponíveis.
* Não realiza nenhuma lógica de negócio. Apenas orquestra o carregamento de dados e chama os métodos apropriados para apresentar os resultados ao utilizador.

**RecipeDataLoader**

* **Atributo**:
  + filePath: Caminho do ficheiro onde estão armazenadas as receitas.
* **Método**:
  + loadRecipes(): Lê as receitas do ficheiro e retorna uma lista de objetos Recipe, cada um com o seu conjunto de ingredientes.
* **Relações**:
  + **Cria** instâncias de Recipe e Ingredient durante a leitura dos dados.
* Responsável unicamente pela leitura e transformação dos dados do ficheiro para objetos de domínio.

**RecommendationService**

* **Atributos**:
  + recipes: Lista de receitas carregadas.
  + graph: Grafo que representa ligações entre ingredientes.
* **Métodos**:
  + recommend(Set<Ingredient>): Imprime receitas completas e incompletas com base nos ingredientes disponíveis.
* **Relações**:
  + **Usa** Recipe, Ingredient e RecipeGraph.
  + **Cria** o grafo de ingredientes através de RecipeGraph.
* Contém a lógica de recomendação principal, distinguindo receitas completas e incompletas.

**Recipe**

* **Atributos**:
  + name: Nome da receita.
  + ingredients: Conjunto de ingredientes necessários.
* **Métodos**:
  + getName(), getIngredients(): Acessores para nome e ingredientes.
  + toString(): Representação textual da receita.
* **Relações**:
  + **Contém** objetos Ingredient.
* Classe de domínio que representa uma receita individual.

**Ingredient**

* **Atributo**:
  + name: Nome do ingrediente (armazenado em minúsculas e sem espaços).
* **Métodos**:
  + getName(), equals(), hashCode(), toString(): Métodos essenciais para comparação, hashing e visualização.
* **Relações**:
  + **Usado em** Recipe, RecommendationService e RecipeGraph como unidade de comparação e conectividade.
* Projetado para funcionar corretamente em estruturas como Set e Map.

**RecipeGraph**

* **Atributo**:
  + adjacencyList: Tabela de adjacências entre ingredientes (grafo não direcionado).
* **Métodos**:
  + buildGraph(List<Recipe>): Constrói as conexões entre ingredientes com base nas receitas.
  + getNeighbors(Ingredient): Retorna os ingredientes ligados a outro.
  + pathExists(start, target, available): Verifica se é possível alcançar um ingrediente a partir de outro, dado um conjunto de ingredientes disponíveis.
* **Relações**:
  + **Usa** HashTable<Ingredient, Ingredient> para estruturar o grafo.
  + **Utiliza** Ingredient como nós.
* Facilita a expansão da lógica para recomendações mais inteligentes baseadas em caminhos entre ingredientes.

**HashTable<K, V>**

* **Atributo**:
  + table: Estrutura Map<K, Set<V>> que associa chaves a múltiplos valores.
* **Métodos**:
  + put(K, V): Adiciona uma associação entre chave e valor.
  + get(K): Retorna os valores associados a uma chave.
  + containsKey(K), keySet(), getTable(): Métodos utilitários para inspeção e navegação.
* **Relações**:
  + **Contido em** RecipeGraph para representar conexões entre ingredientes.
* Estrutura genérica de apoio, mas fundamental para a eficiência do grafo de ingredientes.

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

# Pseudocódigo do algoritmo desenvolvido.

INICIAR programa

criar RecipeDataLoader com ficheiro "receitas.txt"

receitas = RecipeDataLoader.loadRecipes()

criar RecommendationService com receitas

mostrar "Digite seus ingredientes disponíveis (separados por espaço):"

input = ler linha do utilizador

ingredientesDisponiveis = transformar input em conjunto de Ingredient

RecommendationService.recommend(ingredientesDisponiveis)

TERMINAR programa

**Classe RecommendationService**

Atributos:

- recipes: Lista de Recipe

- graph: RecipeGraph

Construtor(recipes):

this.recipes = recipes

this.graph = novo RecipeGraph()

graph.buildGraph(recipes)

Método recommend(available):

mostrar "Receitas completas:"

para cada recipe em recipes:

se available contém todos os ingredientes da receita:

mostrar recipe

mostrar "Receitas incompletas (faltam no máximo 2 ingredientes):"

para cada recipe em recipes:

missing = ingredientes da receita - available

se missing não está vazio e tamanho de missing <= 2:

contemAlgum = false

para cada ing em available:

se ing está nos ingredientes da receita:

contemAlgum = true

sair do ciclo

se contemAlgum:

mostrar "Se comprar [missing] pode cozinhar [nome]: [ingredientes]"

**Classe RecipeGraph**

Atributo:

- adjacencyList: HashTable<Ingredient, Ingredient>

Método buildGraph(recipes):

para cada recipe em recipes:

ingList = lista de ingredientes da receita

para cada par (i, j) em ingList:

adjacencyList.put(ingList[i], ingList[j])

adjacencyList.put(ingList[j], ingList[i])

Método getNeighbors(ingredient):

retorna adjacencyList.get(ingredient)

Método pathExists(start, target, available):

criar queue (fila)

criar visited (conjunto)

adicionar start a queue e visited

enquanto queue não está vazia:

current = queue.poll()

se current == target:

retorna true

para cada neighbor de current:

se neighbor não visitado e (está em available OU neighbor == target):

adicionar neighbor a visited e queue

retorna false

**Classe RecipeDataLoader**

Atributo:

- filePath: String

Construtor(filePath):

this.filePath = filePath

Método loadRecipes():

criar lista recipes vazia

abrir ficheiro filePath

para cada linha no ficheiro:

dividir linha por vírgula

recipeName = primeira parte

ingredients = conjunto vazio

para cada parte a partir da segunda:

adicionar novo Ingredient ao conjunto

adicionar nova Recipe(recipeName, ingredients) à lista

retornar lista récipes

**Classe Recipe**

Atributos:

- name: String

- ingredients: Set<Ingredient>

Construtor(name, ingredients):

this.name = name

this.ingredients = ingredients

Métodos:

getName():

retorna name

getIngredients():

retorna ingredients

toString():

retorna name + " " + ingredientes

**Classe HashTable<K, V>**

Atributo:

- table: Map<K, Set<V>>

Construtor():

inicializa table como novo HashMap

Métodos:

put(key, value):

se key não existe em table:

cria novo HashSet para key

adiciona value ao conjunto de key

get(key):

se key existe:

retorna conjunto de key

senão:

retorna conjunto vazio

keySet():

retorna todas as chaves

containsKey(key):

retorna true se key existe em table

getTable():

retorna o mapa table

**Classe Ingredient**

Atributo:

- name: String

Construtor(name):

this.name = name.toLowerCase().trim()

Métodos:

getName():

retorna name

equals(obj):

se obj for Ingredient e name igual:

retorna true

senão:

retorna false

hashCode():

retorna hash de name

toString():

retorna name

# Conclusão

Este projeto consistiu no desenvolvimento de uma aplicação funcional e prática de recomendação de receitas, tendo como base a disponibilidade de ingredientes fornecida pelo utilizador. Através da utilização de estruturas de dados eficientes, como mapas (Map) e conjuntos (Set), e da construção de um grafo de relações entre ingredientes, foi possível implementar uma lógica robusta que distingue entre receitas completas e receitas incompletas — estas últimas admitindo, no máximo, dois ingredientes em falta.

A clara separação de responsabilidades entre as diversas classes contribuiu significativamente para a modularidade e manutenibilidade do sistema. A classe RecipeDataLoader foi responsável pela leitura dos dados, enquanto a RecommendationService implementou a lógica de recomendação, apoiando-se nas entidades Recipe e Ingredient, que representam os elementos centrais da aplicação. A introdução da classe RecipeGraph permitiu ainda estruturar as relações entre ingredientes, possibilitando futuras extensões, como a recomendação com base na proximidade ou coocorrência entre ingredientes.

Do ponto de vista prático, esta aplicação pode ser facilmente integrada em cenários reais, como frigoríficos inteligentes ou sistemas de apoio à decisão em ambientes culinários digitais. Nestes contextos, a aplicação poderá sugerir receitas com base nos ingredientes identificados como disponíveis, oferecendo assim uma solução eficaz, adaptável e de elevado valor para o utilizador final.

Em suma, o projeto evidencia a aplicação concreta de princípios fundamentais da programação orientada a objetos, de estruturas de dados e de algoritmos sobre grafos, culminando numa solução útil, extensível e com clara aplicabilidade no quotidiano.

# Bibliografia

"Effective Java" – Joshua Bloch

"Object-Oriented Software Engineering: Practical Software Development using UML and Java" – Timothy Lethbridge, Robert Laganière

"Data Structures and Algorithms in Java" – Michael T. Goodrich, Roberto Tamassia

"Algoritmos em Java" – Robert Sedgewick & Kevin Wayne

Introduction to Algorithms" – Cormen, Leiserson, Rivest, Stein (CLRS)

Moodle do Ismat - Algoritmia e Estrutura de Dados