A Sorveteria dos Horrores

Gabriel Spiandorello, Lucas Ehara
Engenharia de Software — PUCRS

19 de novembro de 2022

Resumo

Este relatório consiste na resolução do enunciado proposto na disciplina de Algoritmos e Estruturas de Dados II. Buscamos solucionar a problemática utilizando conceitos e fundamentos aprendidos em sala de aula, da forma mais otimizada e sólida possível. Utilizamos a linguagem JAVA para a construção do programa.

Introdução

A problemática proposta consiste em uma sorveteria fictícia que possui diversos sabores e intensidades de sorvete diferentes, bem como dois tamanhos de copinhos para servi-los, onde cabem 2 ou 3 bolas de sorvete em cada copo.

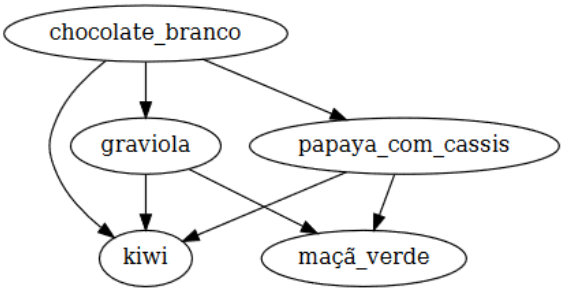
No enunciado do problema, transcorre-se uma história sobre uma pessoa, que nunca coloca sabores repetidos, nem sabores mais intensos sobre mais suaves, exemplo:

- chocolate (sabor forte) em cima de iogurte (sabor fraco);
- chocolate em cima de chocolate (repetição);

Esta mesma pessoa, possui uma lista de todos os sabores em comparação com suas intensidades, chamada “quem-é-mais-forte-do-que-quem”, a fim de se orientar na hora de realizar o pedido.

Exemplo, à esquerda se encontra um pedaço desta lista e à direita uma representação ilustrativa dela.

```
chocolate_branco -> papaya_com_cassis
chocolate_branco -> graviola
chocolate_branco -> kiwi
papaya_com_cassis -> maçã_verde
papaya_com_cassis -> kiwi
graviola -> maçã_verde
graviola -> kiwi
```



A fim de auxiliar na contagem de quantas combinações de sabores seriam possíveis, neste relatório apresentaremos qual foi nossa solução encontrada através do conhecimento adquirido em Algoritmos e Estruturas de Dados II e estudos a parte.

Solução encontrada

Para solucionar a problemática, nós utilizamos uma estrutura de nodos que, quando feita a leitura dos arquivos deixados pelo professor no Moodle, cria todo o grafo, baseado nos sabores mais fracos e mais fortes. Para achar a quantidade de combinações de dois sabores, nós fizemos uma busca por todo o grafo e contamos a quantidade de Nodos “filhos” ou sabores mais fortes que o sabor atual que está sendo percorrido, podendo ser visualizado o algoritmo na Figura 1. Já na quantidade de combinações de três sabores, é percorrido o grafo, contando a quantidade de Nodos “netos” ou sabores mais fortes que o “filho” do sabor atual que está sendo percorrido, podendo ser visualizado o algoritmo na Figura 2.

```
public int possiveisCoposComDoisSabores(){
        int possiveisCoposComDoisSabores = 0; → inicializa uma variável para contar
                                                quantos copos de 2 sabores é possível ser feito
        for(int i = 0; i < sabores.size(); i++){ → percorre a lista com todos os sabores
                possiveisCoposComDoisSabores +=
                sabores.get(i).saboresMaisFortes.size(); → aqui ele contabiliza
                                                quantos sabores mais fortes existem que o sabor atual
        }
        return possiveisCoposComDoisSabores;  → retorna a quantidade de copos
                                                com dois sabores
}
```

Figura 1: Algoritmo para achar a quantidade de copos com dois sabores

```
public int possiveisCoposComTresSabores(){
        int possiveisCoposComTresSabores = 0; → inicializa uma variável para contar
                                              quantos copos de 3 sabores são possíveis serem feitos
        for(int i = 0; i < sabores.size(); i++){ → percorre a lista com todos os sabores
                ArrayList<Nodo> saboresMaisFortes =
                sabores.get(i).saboresMaisFortes; → cria uma lista com todos os
                                                  sabores mais fortes que o sabor atual
                for(int j = 0; j < saboresMaisFortes.size(); j ++){ → percorre a lista com
                                                                    os sabores "netos" do sabor atual
                possiveisCoposComTresSabores +=
                saboresMaisFortes.get(j).saboresMaisFortes.size(); → aqui ele
                        contabiliza quantos sabores "netos" existem a partir do sabor atual
                }
        }
        return possiveisCoposComTresSabores; → retorna a quantidade de copos
                                             com três sabores
}
```

Figura 2: Algoritmo para achar a quantidade de copos com três sabores

Eficiência do Algoritmo

O algoritmo utilizado para a contagem de possíveis combinações de sorvetes de 2 sabores é uma função linear (n). Já o algoritmo utilizado para as possíveis combinações de sorvetes de 3 sabores é uma função polinomial de 2º grau (n^2).

Resultados

Conforme os testes deixados pelo professor na plataforma Moodle, o nosso programa lê todos os arquivos e roda dizendo cada quantidade de combinações com dois e três sabores de sorvete. Ao iniciar o programa, é apresentado um menu, para o usuário escolher qual dos arquivos ele deseja ler. Abaixo é apresentado uma tabela, com o resultado de todos os arquivos disponibilizados pelo professor e o tempo em segundos demorado para fazer cada teste.

Casos de Teste	Resultado	Tempo em segundos
casoscohen10.txt	27 combinações para 2 sabores e 32 combinações para 3 sabores	Menos de 1 segundo
casoscohen20.txt	37 combinações para 2 sabores e 28 combinações para 3 sabores	Menos de 1 segundo
casoscohen30.txt	213 combinações para 2 sabores e 604 combinações para 3 sabores	Menos de 1 segundo
casoscohen40.txt	466 combinações para 2 sabores e 2177 combinações para 3 sabores	Menos de 1 segundo
casoscohen50.txt	808 combinações para 2 sabores e 5782 combinações para 3 sabores	Menos de 1 segundo
casoscohen60.txt	1308 combinações para 2 sabores e 14118 combinações para 3 sabores	Menos de 1 segundo

Conclusões

Retomando a problemática proposta e tendo em vista os estudos e soluções realizadas para resolvê-la, concluímos que o código é muito eficiente para todos os casos de teste presentes no arquivo, entretanto, como a complexidade do algoritmo utilizado para achar a quantidade possível de copos com três sabores foi analisada em n^2, caso fôssemos utilizar nossa solução em testes consideravelmente maiores, acabaria por não ser tão eficiente e teríamos que otimizar o algoritmo devido a alta complexidade dele.