

*Resumo: Este artigo descreve uma alternativa para solucionar o problema proposto na disciplina de Algoritmos e Estrutura de Dados II, que consiste em descobrir o número de combinações possíveis de copinhos de sorvete segundo as regras de uma sorveteria. A modelagem do problema e o processo da solução é apresentado, juntamente com o pseudocódigo dos algoritmos mais importantes. Por fim, serão apresentados os resultados obtidos nos seis casos de teste disponibilizados pelo professor.*

## Enunciado do problema

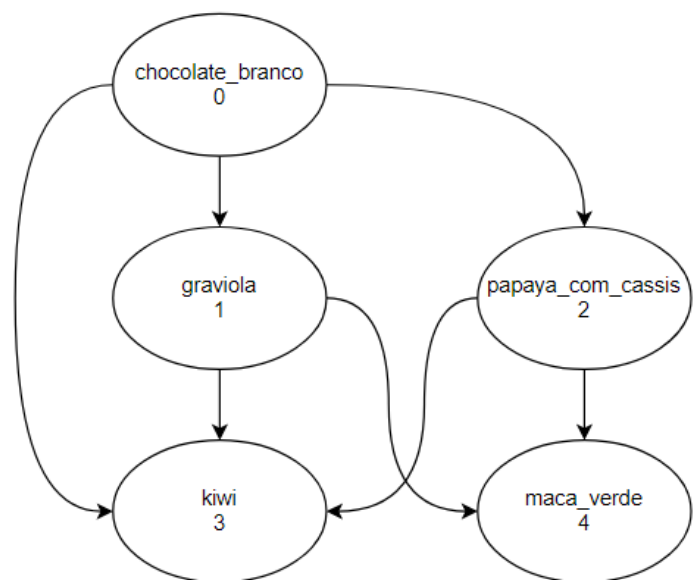
No contexto do problema, meu primo abriu uma sorveteria, mas impôs algumas regras sobre os sorvetes que ele vende, sendo estas:

1. Na sorveteria só são vendidos copinhos com 2 ou 3 bolas de sorvete.
2. É proibido misturar um sabor forte com um sabor fraco (Exemplo: Chocolate com Iogurte. Para saber quais sabores são fortes ou fracos há uma lista).
3. Não é permitido repetir sabores, para que as pessoas provem mais sabores.

A família está preocupada com as regras que ele impôs, e acham que isso pode limitar o número de pedidos da sorveteria, então me foi pedido para criar um programa que cheque o número de combinações possíveis de copinhos de sorvete.

Abaixo um exemplo da lista de proibições da sorveteria junto de um diagrama da lista para facilitar a visualização.

```
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
```



O problema foi modelado para que utilizássemos grafos dirigidos e utilizássemos métodos de busca para obter as combinações possíveis de copinhos, então nós utilizamos das classes **Digraph**, classe construtora do grafo, e da classe **DigrafoBuscaProfundidade**, para realizar a busca dos caminhos por profundidade no grafo.

## Etapas da solução, apresentando exemplos e algoritmos

---

O programa começa lendo um arquivo de texto e quebrando as linhas quando encontra o padrão “->” nelas através da classe **ArquivoUtil**. A classe pega os elementos da linha e checa se eles já foram adicionados na lista, se não, os adiciona

```
1  public class ArquivoUtil {
2
3      public static SaboresLinhas carregar(String path) {
4          In in = new In(path);
5          String[] linhasArquivo = in.readAllLines();
6
7          List<String> saboresDistintos = new ArrayList<>();
8
9          for (String linha : linhasArquivo) {
10             String aLinha[] = linha.split(" -> ");
11             String sabor1 = aLinha[0].trim();
12             String sabor2 = aLinha[1].trim();
13
14             if (!saboresDistintos.contains(sabor1)) {
15                 saboresDistintos.add(sabor1);
16             }
17
18             if (!saboresDistintos.contains(sabor2)) {
19                 saboresDistintos.add(sabor2);
20             }
21         }
22         return new SaboresLinhas(saboresDistintos, linhasArquivo);
23     }
24 }
```

Tendo feita a leitura, o programa cria um dígrafo com o tamanho de sabores distintos de sorvete e conecta os vértices

```
1 // Cria com digraph com o tamanho dos sabores distintas
2 Digraph digraph = new Digraph(saboresLinhas.saboresDistintos.size());
3
4 // Cria as conexoes dos vertices
5 for (String linha : saboresLinhas.linhas) {
6     String linhaSplit[] = linha.split(" -> ");
7     digraph.addEdge(saboresLinhas.saboresDistintos.indexOf(linhaSplit[0]),
8     saboresLinhas.saboresDistintos.indexOf(linhaSplit[1]));
9 }
```

Após o programa ter o Dígrafo construído, é iniciado o caminhamento por profundidade para cada vértice existente

```
1 // Caminha em todos vertices do digraph para adicionar nos caminhos
2 for (int i = 0; i < digraph.V(); i++) {
3     digrafoBuscaProfundidade.caminhar(digraph, i);
4 }
```

É na classe **DigrafoBuscaProfundidade** onde ocorre o principal algoritmo do programa sendo este o **CaminharRecursivo**, conforme a foto abaixo:

```

1  public void CaminharRecursivo(Digraph g, int s, boolean[] marcado, ArrayList<
   Integer> caminho) {
2
   //Recebe o digrafo, um vértice para começar, uma lista boolean para marcar as arestas
   a serem visitadas, e um arraylist para armazenar o caminho
3   for (int v : g.adj(s)) {
   //Para cada aresta adjacente da aresta inicial
4       if (!marcado[v]) {
   //Se ela não for marcada
5           caminho.add(v);
   //Adiciona a aresta em um arraylist de caminhos(Mais ou menos uma pilha)
6           ArrayList<Integer> listaTemporaria = new ArrayList<>(caminho);
   //cria uma lista dos caminhos do vértice atual
7           todosCaminhos.add(listaTemporaria);
   //adiciona o caminho do vértice atual em um arraylist de todos os caminhos
8           CaminharRecursivo(g, v, marcado, caminho);
   //chama recursivamente o método Caminhar recursivo com os novos caminhos adicionados
9           caminho.remove((Integer) (v));
   //remove o caminho atual da lista temporária
10      }
11  }
12  }

```

Após ter sido feito o caminhamento, nós obtemos uma lista de caminhos possíveis de todos os vértices. O próximo passo é contar quais caminhos são possíveis com copinhos com 2 ou 3 bolas de sorvete (o programa utiliza da classe Tupla para auxiliar na contagem de caminhos, transformando os em objetos).

(Classe Tupla)

```

1  package Trabalho2;
2  public class Tupla {
3      int a;
4      int b;
5      int c;
6
7      public Tupla(int a, int b) {
8          this.a = a;
9          this.b = b;
10     }
11
12     public Tupla(int a, int b, int c) {
13         this.a = a;
14         this.b = b;
15         this.c = c;
16     }
17
18 }

```

Contador de 2 sabores:

```
1 // Contabiliza tres sabores
2 for (int i = 0; i < digrafoBuscaProfundidade.todosCaminhos.size(); i++) {
3     ArrayList<Integer> caminho = digrafoBuscaProfundidade.todosCaminhos.get(i);
4
5     for (int j = 0; j < caminho.size(); j++) {
6         for (int k = j; k < caminho.size(); k++) {
7             for (int z = k; z < caminho.size(); z++) {
8                 int a = caminho.get(j);
9                 int b = caminho.get(k);
10                int c = caminho.get(z);
11                if (a != b && a != c && b != c) {
12                    tresSaboresUtil.adicionar(a, b, c);
13                }
14            }
15        }
16    }
17 }
```

Contador de 3 sabores:

```
1 // Contabiliza tres sabores
2 for (int i = 0; i < digrafoBuscaProfundidade.todosCaminhos.size(); i++) {
3     ArrayList<Integer> caminho = digrafoBuscaProfundidade.todosCaminhos.get(i);
4
5     for (int j = 0; j < caminho.size(); j++) {
6         for (int k = j; k < caminho.size(); k++) {
7             for (int z = k; z < caminho.size(); z++) {
8                 int a = caminho.get(j);
9                 int b = caminho.get(k);
10                int c = caminho.get(z);
11                if (a != b && a != c && b != c) {
12                    tresSaboresUtil.adicionar(a, b, c);
13                }
14            }
15        }
16    }
17 }
```

As classes dois/tresSaboresUtil fazem a manipulação dos caminhos e encaminham para a classe tupla para criar os objetos. Seguem abaixo elas:

```

1 package Trabalho2;
2 import java.util.ArrayList;
3
4 public class DoisSaboresUtil {
5     public ArrayList<Tupla> itens = new ArrayList<>();
6
7     // Se nao existe adiciona na lista
8     public void adicionar(int a, int b) {
9         if (!existe(a, b)) {
10             Tupla p = new Tupla(a, b);
11             itens.add(p);
12         }
13     }
14
15     // Verifica se existe
16     private boolean existe(int a, int b) {
17         for (int i = 0; i < itens.size(); i++) {
18             Tupla p = itens.get(i);
19             if (p.a == a && p.b == b) {
20                 return true;
21             }
22         }
23         return false;
24     }
25 }

```

```

1 package Trabalho2;
2 import java.util.ArrayList;
3
4 public class TresSaboresUtil {
5     public ArrayList<Tupla> itens = new ArrayList<>();
6
7     // Se nao existe adiciona na lista
8     public void adicionar(int a, int b, int c) {
9         if (!existe(a, b, c)) {
10             Tupla p = new Tupla(a, b, c);
11             itens.add(p);
12         }
13     }
14
15     // Verifica se existe
16     private boolean existe(int a, int b, int c) {
17         for (int i = 0; i < itens.size(); i++) {
18             Tupla p = itens.get(i);
19             if (p.a == a && p.b == b && p.c == c) {
20                 return true;
21             }
22         }
23         return false;
24     }
25 }

```

Após isso o programa exibe o resultado do caso de teste junto de análises de tempo para as execuções blocos essenciais do programa

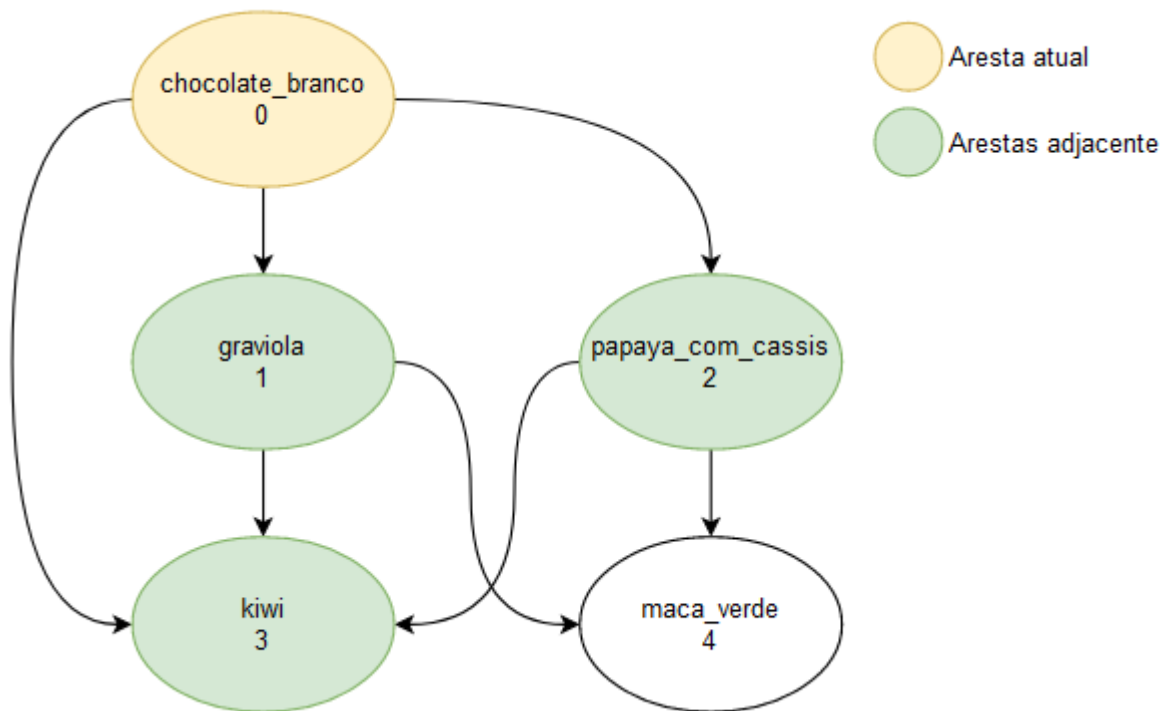
```

1     float segundosCriarGrafoVertice = (fimCriarGrafoVertice - inicioCriarGrafoVertice) / 1000F;
2     float segundosCaminhar = (fimCaminhar - inicioCaminhar) / 1000F;
3     float segundosContar2 = (fimContar2Sabores - inicioContar2Sabores) / 1000F;
4     float segundosContar3 = (fimContar3Sabores - inicioContar3Sabores) / 1000F;
5     float segundosGeral = (tempoGeral - comecoPrograma) / 1000F;
6
7     System.out.println("=====");
8     System.out.println(digraph.toDot() + "\n");
9     System.out.println("=====");
10    System.out.println("Caso de teste com 10 sabores:");
11    System.out.println("    Tempo para criar Grafo e conectar Vértices:      " + segundosCriarGrafoVertice);
12    System.out.println("    Tempo para criar Caminhar pelo grafo:                " + segundosCaminhar);
13    System.out.println("    Tempo para contar 2 sabores:                          " + segundosContar2);
14    System.out.println("    Tempo para contar 3 sabores:                          " + segundosContar3);
15    System.out.println("    Tempo Geral:                                          " + segundosGeral);
16    System.out.println("=====");
17    System.out.println("Sabores: \n      " + saboresLinhas.saboresDistintos);
18    System.out.println("=====");
19    System.out.println("Número de caminhos possíveis: " + digrafoBuscaProfundidade.todosCaminhos.size());
20    System.out.println("    Copinhos com Dois sabores: " + doisSaboresUtil.itens.size());
21    System.out.println("    Copinhos com Tres sabores: " + tresSaboresUtil.itens.size());

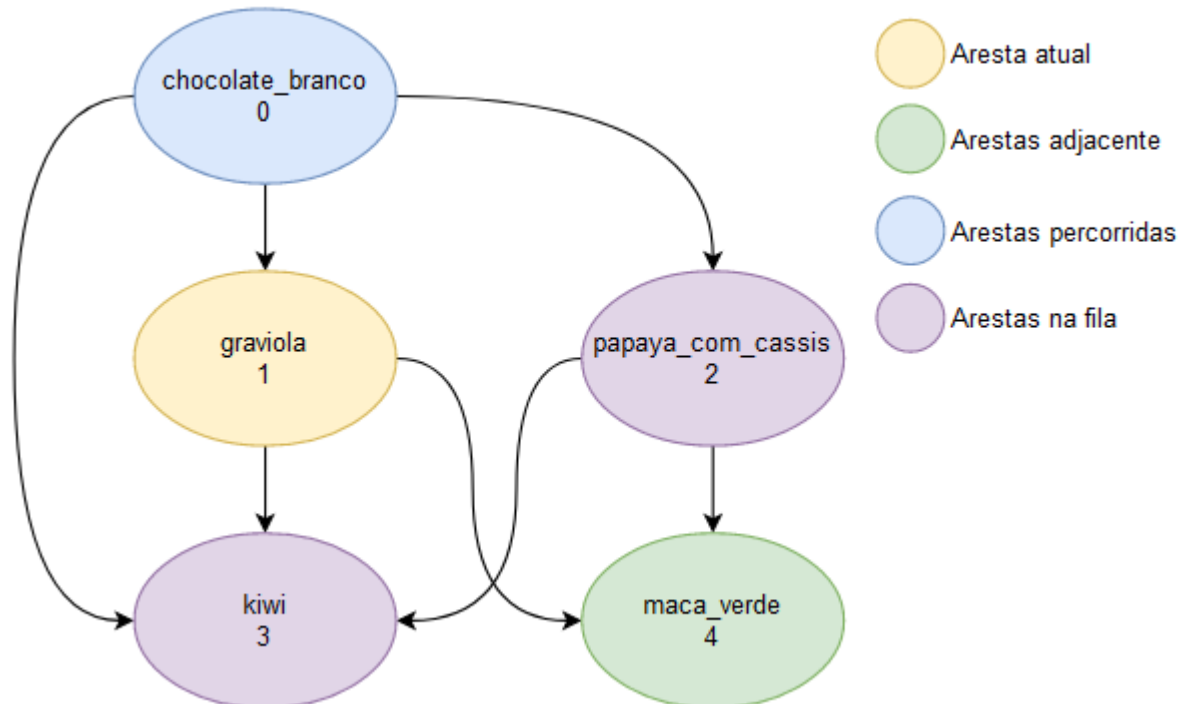
```

## Exemplo de caminhamento:

A partir de uma aresta inicial (Neste caso o chocolate\_branco/0), adicionamos as arestas adjacentes em uma lista de caminhos possíveis, se elas já não estiverem marcadas (como este é o primeiro caminhamento não há arestas marcadas)



Após tendo separado os adjacentes de chocolate branco, passamos para graviola, pois é o próximo da fila, e separamos os seus adjacentes



Como o dígrafo do exemplo é pequeno, não há novas arestas adjacentes, então é percorrida a fila atual de arestas e separados os caminhos para onde eles apontam.

# Resultados dos casos de teste

Seguem abaixo os resultados para os casos de teste disponibilizados.

<pre>===== Caso de teste com 10 sabores:   Tempo para criar Grafo e conectar Vértices:    0.001   Tempo para criar Caminhar pelo grafo:          0.001   Tempo para contar 2 sabores:                   0.0   Tempo para contar 3 sabores:                   0.0   Tempo Geral:                                   0.004 ===== Sabores: [pitanga, creme_russo, tamarindo, abóbora, cookies, abóbora_com_coco, gianduia, pavê, uva, chokito] ===== Número de caminhos possíveis: 15   Copinhos com Dois sabores: 14   Copinhos com Tres sabores: 4</pre>
<pre>===== Caso de teste com 20 sabores:   Tempo para criar Grafo e conectar Vértices:    0.003   Tempo para criar Caminhar pelo grafo:          0.002   Tempo para contar 2 sabores:                   0.001   Tempo para contar 3 sabores:                   0.001   Tempo Geral:                                   0.008 ===== Sabores: [amora, pistache, tâmara, acerola, limão, abacate, prestigio, abacaxi_com_hortelã, danete, brigadeiro, nata, laranja, tangerina, café, melão, carambola, morango, tamarindo, na politano, framboesa] ===== Número de caminhos possíveis: 102   Copinhos com Dois sabores: 78   Copinhos com Tres sabores: 106</pre>
<pre>===== Caso de teste com 30 sabores:   Tempo para criar Grafo e conectar Vértices:    0.002   Tempo para criar Caminhar pelo grafo:          0.002   Tempo para contar 2 sabores:                   0.005   Tempo para contar 3 sabores:                   0.012   Tempo Geral:                                   0.022 ===== Sabores: [pavê, menta, milho_verde, limão, queijo, frutas_vermelhas, nozes, banana, açaí, creme, goiaba, cupuaçu, tangerina, acerola, café, morango, abóbora, chiclete, graviola, nata, amendoim, creme_russo, prestigio, framboesa, frutas_cristalizadas, abóbora_com_coco, sonho_de_valsa, pêssego, chocolate_branco, pitanga] ===== Número de caminhos possíveis: 680   Copinhos com Dois sabores: 236   Copinhos com Tres sabores: 665</pre>
<pre>===== Caso de teste com 40 sabores:   Tempo para criar Grafo e conectar Vértices:    0.005   Tempo para criar Caminhar pelo grafo:          0.01   Tempo para contar 2 sabores:                   0.039   Tempo para contar 3 sabores:                   0.513   Tempo Geral:                                   0.57 ===== Sabores: [prestigio, cenoura, tamarindo, menta, morango_e_nata, graviola, manga, café, pistache, tâmara, chokito, amendoim, cupuaçu, nozes, sonho_de_valsa, leite_condensado, iogurte, a meixa, coco_queimado, carambola, danete, napolitano, chocolate_branco, chiclete, amarula, chocolate, creme, pavê, frutas_vermelhas, tangerina, abóbora_com_coco, cookies, gianduia, açaí, amora, morango, coco, tutti_frutti, abacaxi, frutas_cristalizadas] ===== Número de caminhos possíveis: 7419   Copinhos com Dois sabores: 500   Copinhos com Tres sabores: 2754</pre>
<pre>===== Caso de teste com 50 sabores:   Tempo para criar Grafo e conectar Vértices:    0.004   Tempo para criar Caminhar pelo grafo:          0.016   Tempo para contar 2 sabores:                   0.135   Tempo para contar 3 sabores:                   4.126   Tempo Geral:                                   4.283 ===== Sabores: [cookies, frutas_vermelhas, tamarindo, chocolate_branco, leite_condensado, abacate, ferrero, graviola, chiclete, nata, abacaxi_com_hortelã, manga, amarula, danete, jaca, noze s, creme, goiaba, acerola, chocolate, papaya_com_cassis, tâmara, uva, passas_ao_rum, pitanga, açaí, flocos, amendoim, pavê, tangerina, cupuaçu, coco, laranja, prestigio, iogurte, chokito, abóbora, cenoura, morango_e_nata, menta, brigadeiro, creme_russo, abacaxi, frutas_cristalizadas, sonho_de_valsa, gianduia, framboesa, café, amora, pêssego] ===== Número de caminhos possíveis: 23252   Copinhos com Dois sabores: 806   Copinhos com Tres sabores: 5827</pre>
<pre>===== Caso de teste com 60 sabores:   Tempo para criar Grafo e conectar Vértices:    0.003   Tempo para criar Caminhar pelo grafo:          0.04   Tempo para contar 2 sabores:                   1.496   Tempo para contar 3 sabores:                   128.04   Tempo Geral:                                   129.581 ===== Sabores: [nozes, cenoura, leite_condensado, maracujá, abacaxi, melão, pitanga, abacate, café, maçã_verde, acerola, chocolate, milho_verde, carambola, prestigio, baunilha, gianduia, banana, creme, flocos, pêssego, queijo, tutti_frutti, abacaxi_com_hortelã, chiclete, brigadeiro, cupuaçu, limão, ferrero, jaca, morango, tâmara, coco_queimado, pistache, tangerina, tamarindo , kiwi, goiaba, chocolate_branco, frutas_vermelhas, amarula, nata, manga, cookies, napolitano, menta, groselha, açaí, amendoim, danete, morango_e_nata, pavê, laranja, chokito, iogurte , passas_ao_rum, uva, creme_russo, ameixa, sonho_de_valsa] ===== Número de caminhos possíveis: 162336   Copinhos com Dois sabores: 1255   Copinhos com Tres sabores: 12516</pre>



## Conclusões

---

A ideia do programa é muito boa, foi muito bom para nosso aprendizado aprender a construir grafos direcionados (e não direcionados) e caminhar por eles. Os métodos que construímos são muito úteis e com certeza serão reutilizados em programas futuros.