Relatório – A sorveteria dos horrores

Arthur Viegas e Henrique Ramires

Escola Politécnica – PUCRS

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

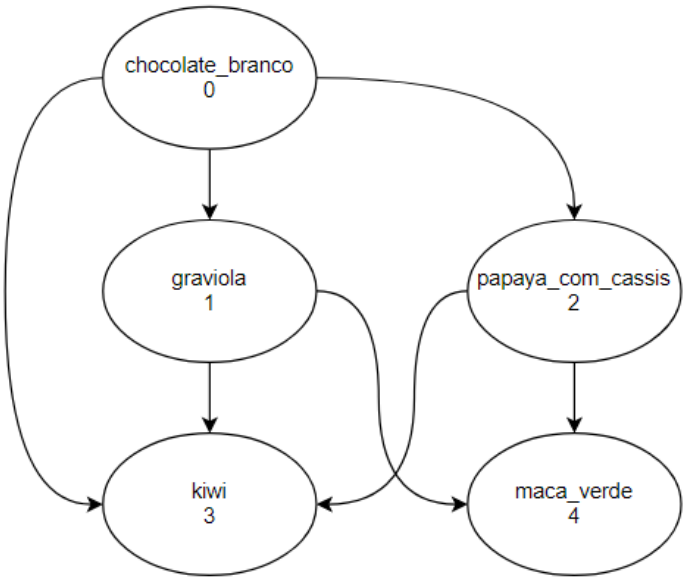
No contexto do problema, meu primo abriu uma sorveteria, mas impôs algumas regras sobre os sorvetes que ele vende, sendo estas:

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

```java
public class ArquivoUtil {

    public static SaboresLinhas carregar(String path) {
        In in = new In(path);
        String[] linhasArquivo = in.readAllLines();

        List<String> saboresDistintos = new ArrayList<>();

        for (String linha : linhasArquivo) {
            String aLinha[] = linha.split(" -> ");
            String sabor1 = aLinha[0].trim();
            String sabor2 = aLinha[1].trim();

            if (!saboresDistintos.contains(sabor1)) {
                saboresDistintos.add(sabor1);
            }

            if (!saboresDistintos.contains(sabor2)) {
                saboresDistintos.add(sabor2);
            }
        }
        return new SaboresLinhas(saboresDistintos, linhasArquivo);
    }
}
```

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]),
  saboresLinhas.saboresDistintos.indexOf(linhaSplit[1]));
8         }
```

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:

```java
    public void CaminharRecursivo(Digraph g, int s, boolean[] marcado, ArrayList<Integer> caminho) {

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

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)

```java
package Trabalho2;
public class Tupla {
    int a;
    int b;
    int c;

    public Tupla(int a, int b) {
        this.a = a;
        this.b = b;
    }

    public Tupla(int a, int b, int c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

}
```

Contador de 2 sabores:

```java
// Contabiliza tres sabores
for (int i = 0; i < digrafoBuscaProfundidade.todosCaminhos.size(); i++) {
    ArrayList<Integer> caminho = digrafoBuscaProfundidade.todosCaminhos.get(i);

    for (int j = 0; j < caminho.size(); j++) {
        for (int k = j; k < caminho.size(); k++) {
            for (int z = k; z < caminho.size(); z++) {
                int a = caminho.get(j);
                int b = caminho.get(k);
                int c = caminho.get(z);
                if (a != b && a != c && b != c) {
                    tresSaboresUtil.adicionar(a, b, c);
                }
            }
        }
    }
}
```

Contador de 3 sabores:

```java
// Contabiliza tres sabores
for (int i = 0; i < digrafoBuscaProfundidade.todosCaminhos.size(); i++) {
    ArrayList<Integer> caminho = digrafoBuscaProfundidade.todosCaminhos.get(i);

    for (int j = 0; j < caminho.size(); j++) {
        for (int k = j; k < caminho.size(); k++) {
            for (int z = k; z < caminho.size(); z++) {
                int a = caminho.get(j);
                int b = caminho.get(k);
                int c = caminho.get(z);
                if (a != b && a != c && b != c) {
                    tresSaboresUtil.adicionar(a, b, c);
                }
            }
        }
    }
}
```

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

```java
package Trabalho2;
import java.util.ArrayList;

public class DoisSaboresUtil {
    public ArrayList<Tupla> itens = new ArrayList<>();

    // Se nao existe adiciona na lista
    public void adicionar(int a, int b) {
        if (!existe(a, b)) {
            Tupla p = new Tupla(a, b);
            itens.add(p);
        }
    }

    // Verifica se existe
    private boolean existe(int a, int b) {
        for (int i = 0; i < itens.size(); i++) {
            Tupla p = itens.get(i);
            if (p.a == a && p.b == b) {
                return true;
            }
        }
        return false;
    }
}
```

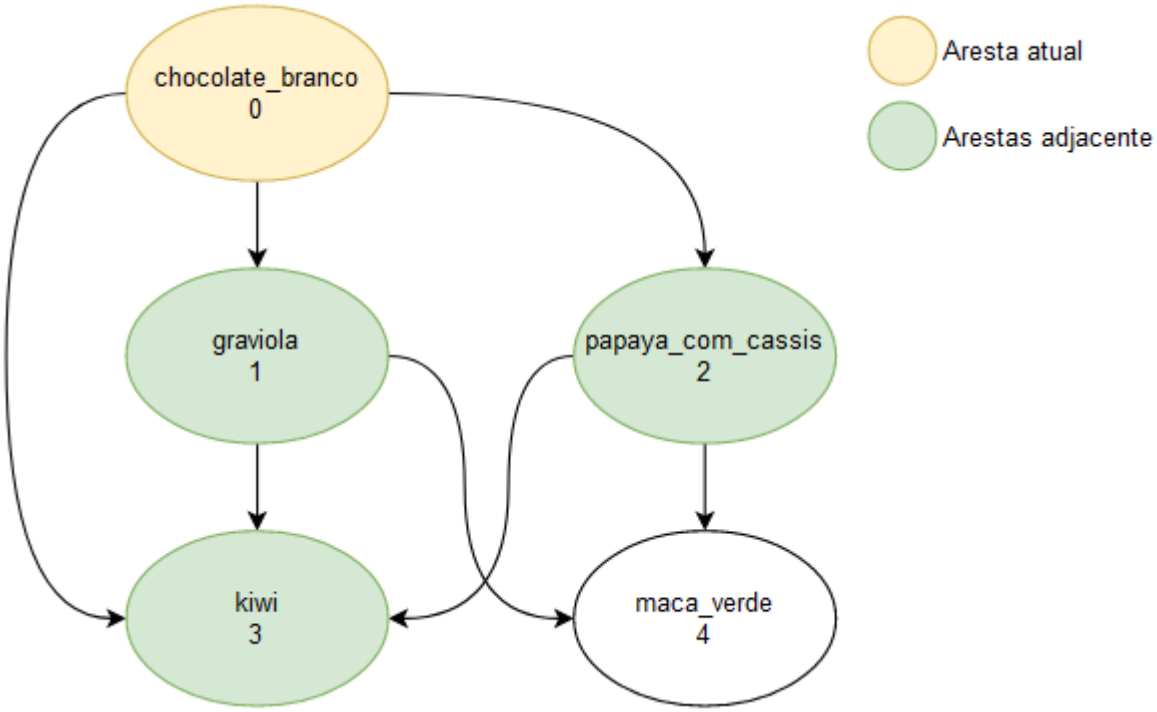
```java
package Trabalho2;
import java.util.ArrayList;

public class TresSaboresUtil {
    public ArrayList<Tupla> itens = new ArrayList<>();

    // Se nao existe adiciona na lista
    public void adicionar(int a, int b, int c) {
        if (!existe(a, b, c)) {
            Tupla p = new Tupla(a, b, c);
            itens.add(p);
        }
    }

    // Verifica se existe
    private boolean existe(int a, int b, int c) {
        for (int i = 0; i < itens.size(); i++) {
            Tupla p = itens.get(i);
            if (p.a == a && p.b == b && p.c == c) {
                return true;
            }
        }
        return false;
    }
}
```

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

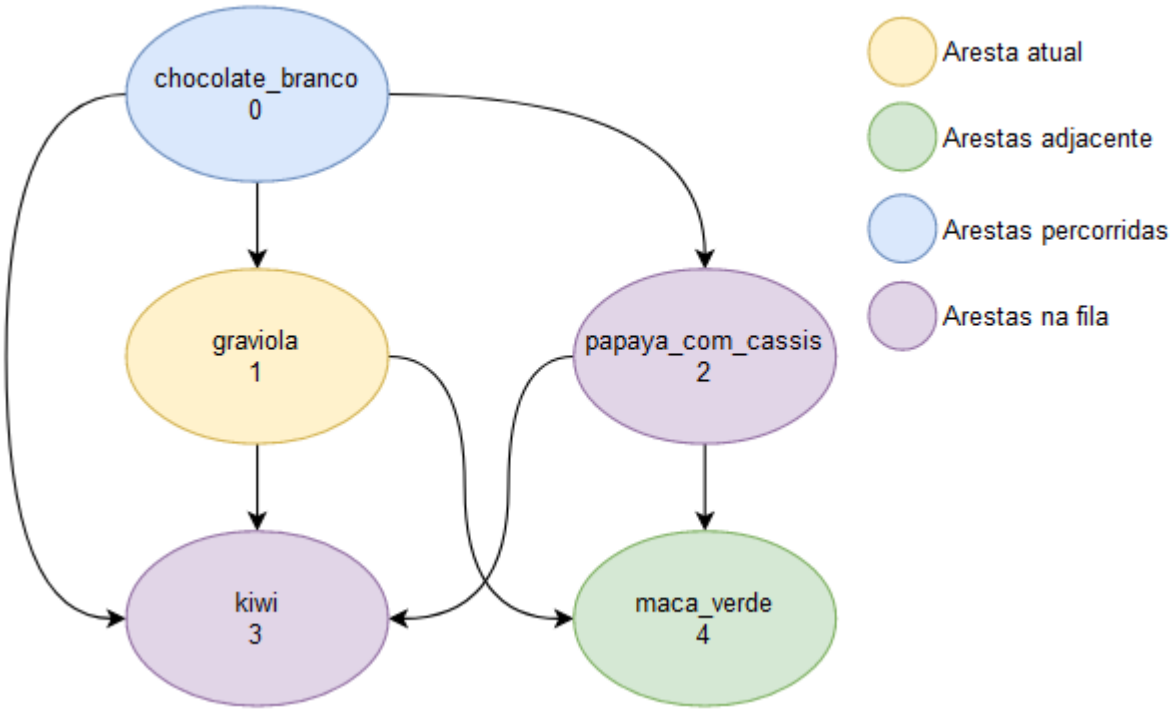
```java
        float segundosCriarGrafoVertice = (fimCriarGrafoVertice - inicioCriarGrafoVertice) / 1000F;
        float segundosCaminhar = (fimCaminhar - inicioCaminhar) / 1000F;
        float segundosContar2 = (fimContar2Sabores - inicioContar2Sabores) / 1000F;
        float segundosContar3 = (fimContar3Sabores - inicioContar3Sabores) / 1000F;
        float segundosGeral = (tempoGeral - comecoPrograma) / 1000F;

        System.out.println("=====================================================");
        System.out.println(digraph.toDot() + "\n");
        System.out.println("=====================================================");
        System.out.println("Caso de teste com 10 sabores:");
        System.out.println("    Tempo para criar Grafo e conectar Vértices:     " + segundosCriarGrafoVertice);
        System.out.println("    Tempo para criar Caminhar pelo grafo:           " + segundosCaminhar);
        System.out.println("    Tempo para contar 2 sabores:                    " + segundosContar2);
        System.out.println("    Tempo para contar 3 sabores:                    " + segundosContar3);
        System.out.println("    Tempo Geral:                                    " + segundosGeral);
        System.out.println("=====================================================");
        System.out.println("Sabores: \n    " + saboresLinhas.saboresDistintos);
        System.out.println("=====================================================");
        System.out.println("Número de caminhos possíveis: " + digrafoBuscaProfundidade.todosCaminhos.size());
        System.out.println("    Copinhos com Dois sabores: " + doisSaboresUtil.itens.size());
        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)



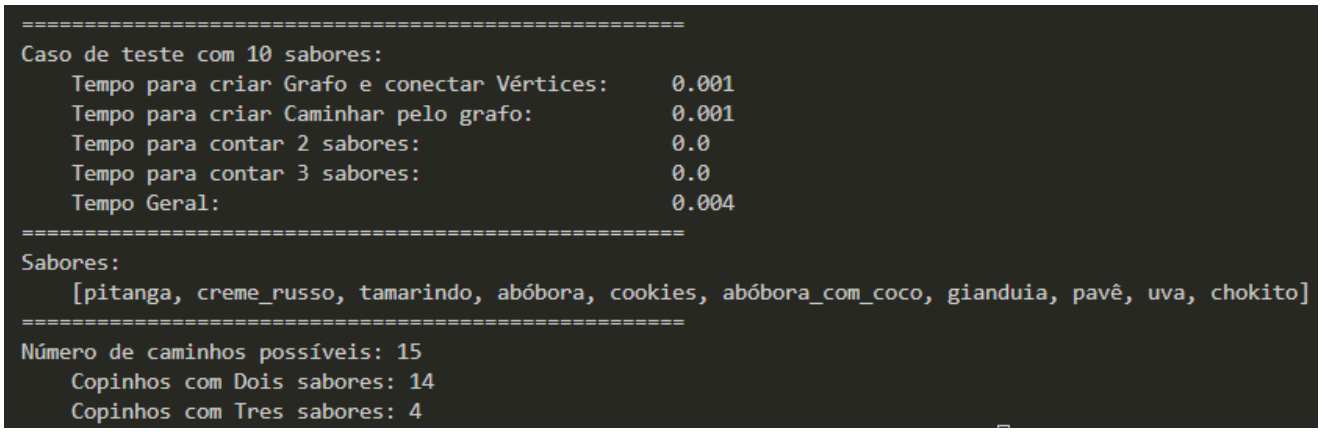
Após tendo separado os adjacentes de chocolate branco, passamos para graviola, pois é o próximo da fila, e separamos os seus adjacentes

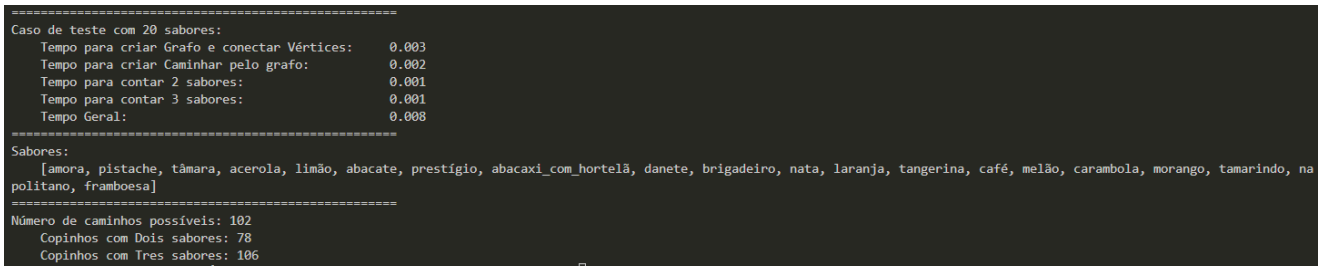


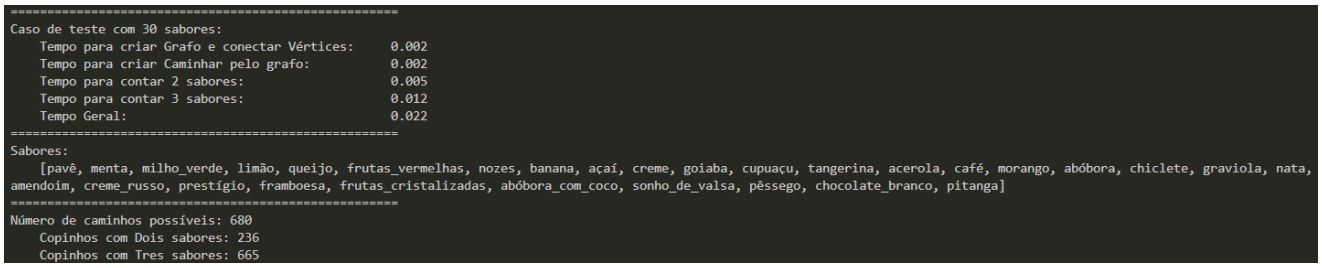
Como o dígrafo do exemplo é pequeno, não há novas arestas adjacentes, então é percorrida a fila atual de arestas e separados os caminhos para onde eles apontam.

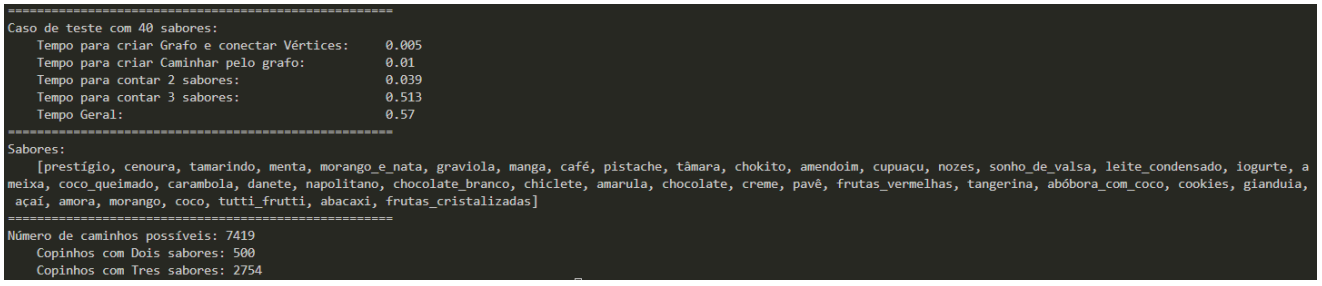
Resultados dos casos de teste

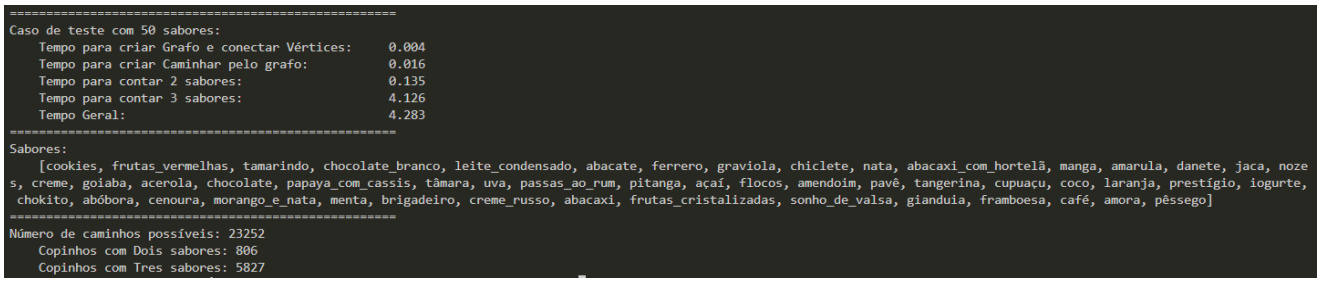
Seguem abaixo os resultados para os casos de teste disponibilizados.

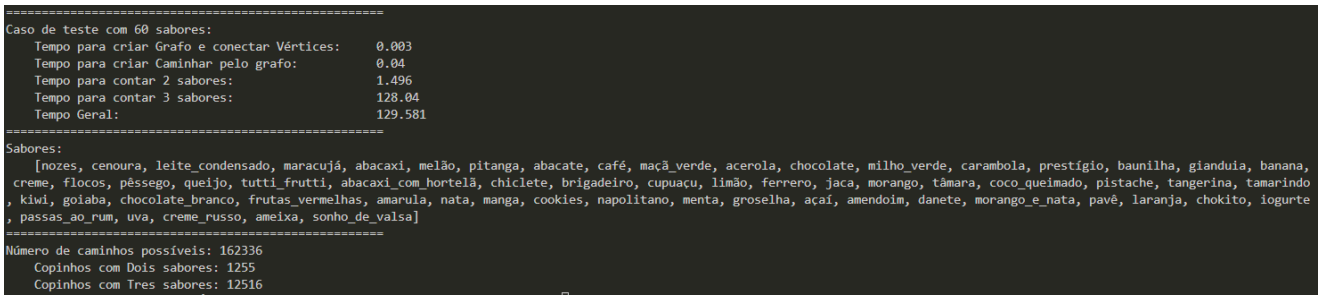












Conclusões

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.