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.*

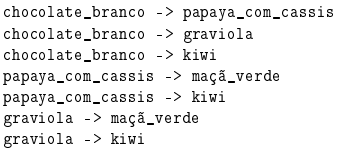
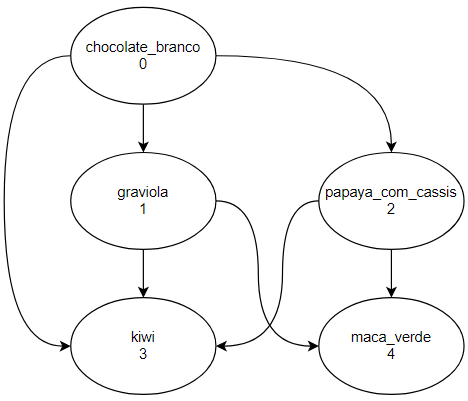
**Enunciado do problema**

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.

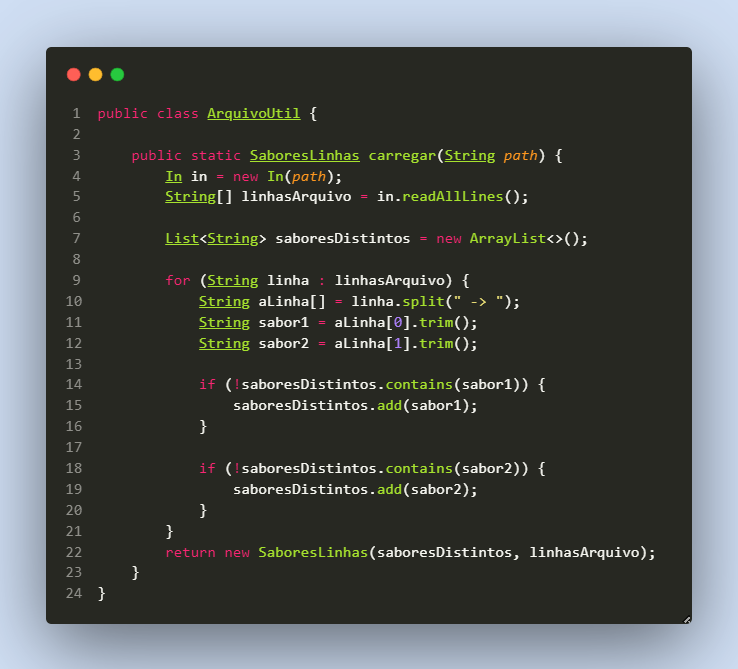
Abaixo um exemplo da lista de proibições da sorveteria junto de um diagrama da lista para facilitar a visualização.

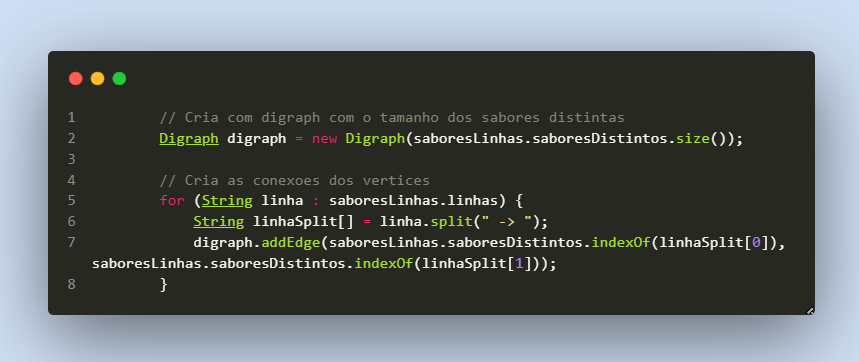
O problema foi modelado para que utilizássemos grafos dirigidos e utilizássemos métodos de busca para obter as combinações possíveis de copinhos, então nós utilizamos das classes **Digraph**, classe construtora do grafo, e da classe **DigrafoBuscaProfundidade**, para realizar a busca dos caminhos por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

**Etapas da solução, apresentando exemplos e algoritmos**

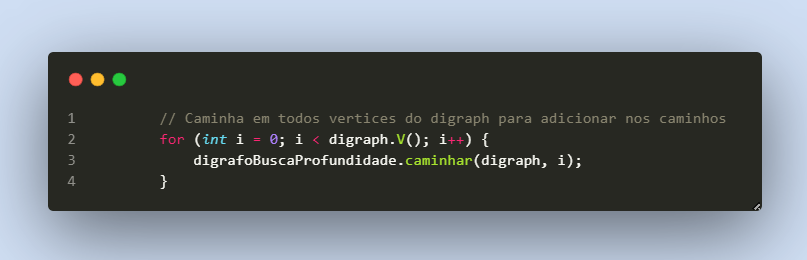
O programa começa lendo um arquivo de texto e quebrando as linhas quando encontra o padrão “ -> ” nelas através da classe **ArquivoUtil**. A classe pega os elementos da linha e checa se eles já foram adicionados na lista, se não, os adiciona



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



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

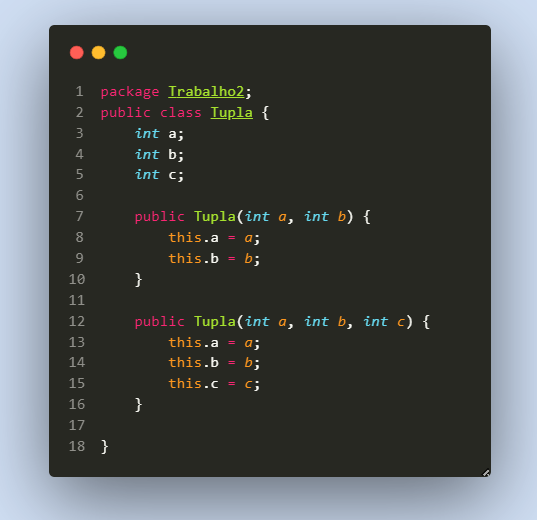


É na classe **DigrafoBuscaProfundidade** onde ocorre o principal algoritmo do programa sendo este o **CaminharRecursivo,** conforme a foto abaixo:



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)



Contador de 2 sabores:



Contador de 3 sabores:



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

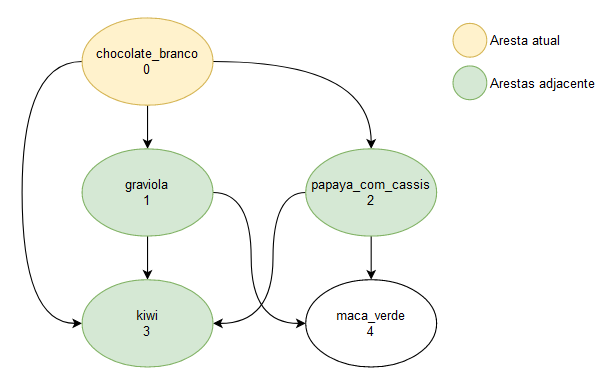


Após isso o programa exibe o resultado do caso de teste junto de análises de tempo para as execuções blocos essenciais do programa

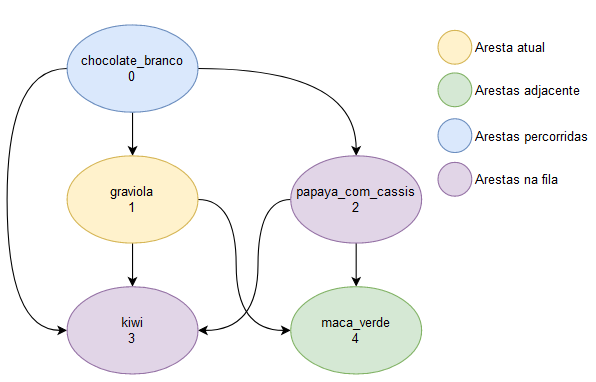


**Exemplo de caminhamento:**

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)



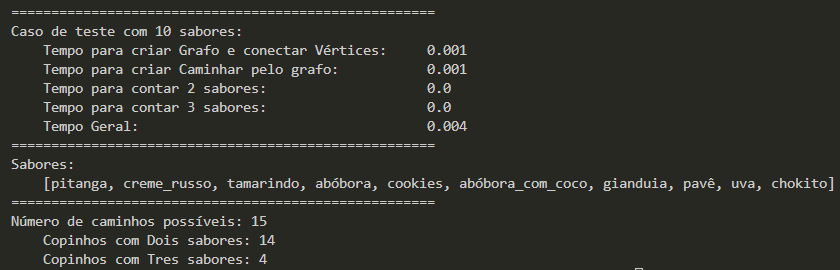
Após tendo separado os adjacentes de chocolate branco, passamos para graviola, pois é o próximo da fila, e separamos os seus adjacentes

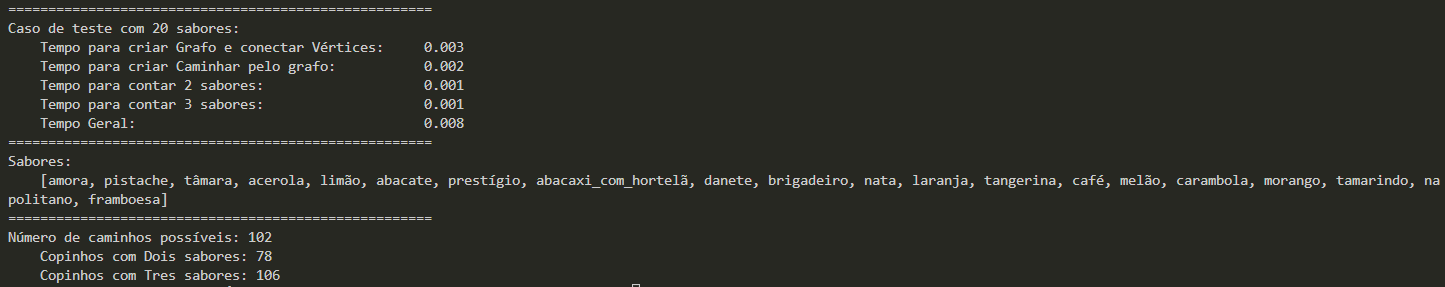


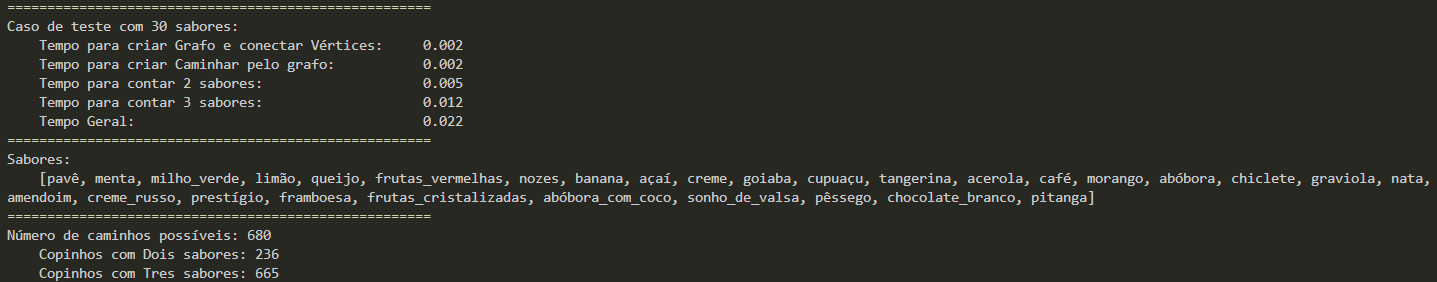
Como o dígrafo do exemplo é pequeno, não há novas arestas adjacentes, então é percorrida a fila atual de arestas e separados os caminhos para onde eles apontam.

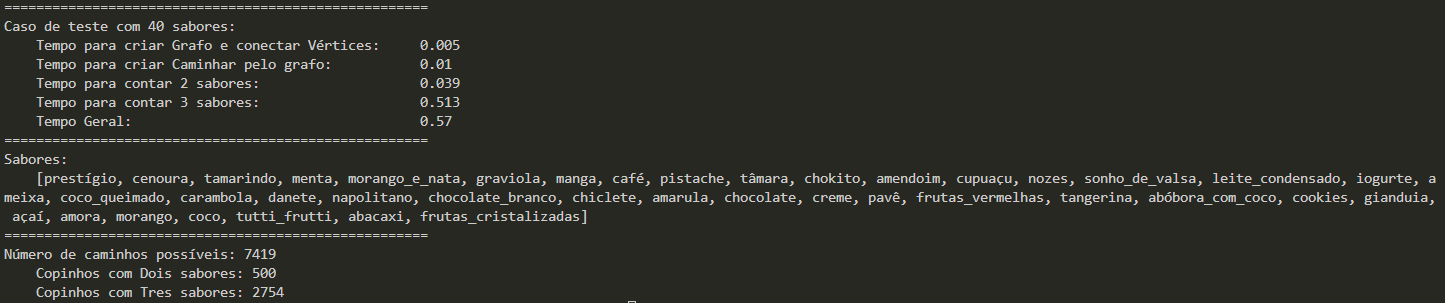
**Resultados dos casos de teste**

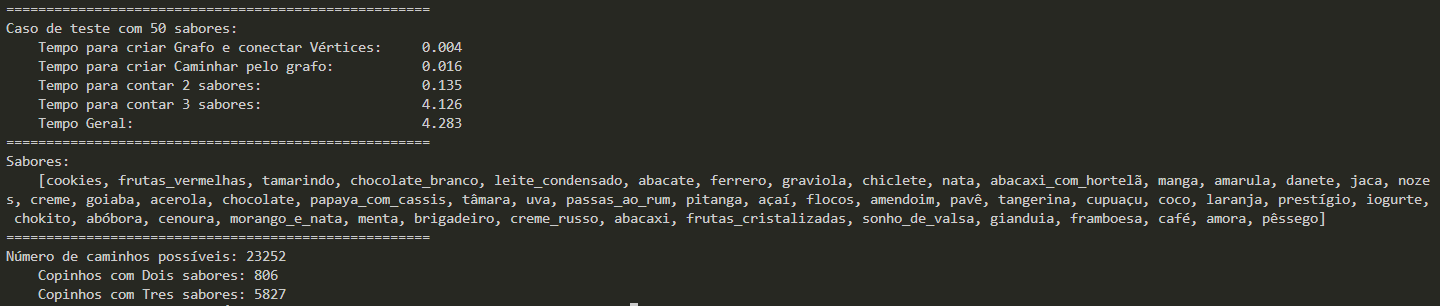
Seguem abaixo os resultados para os casos de teste disponibilizados.

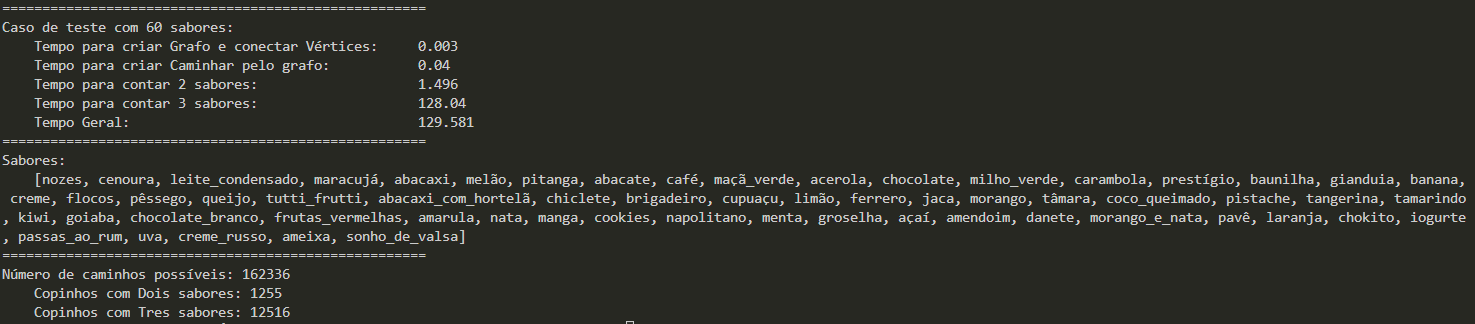












**Conclusões**

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.