Nome: Érico Panassol
Email: erico.r@edu.pucrs.br

Sorveteria Dos Horrores
Algoritmos e Estrutura de Dados II

22 de novembro de 2022

Resumo

Esse trabalho descreve como foi solucionado o problema da Sorveteria dos Horrores proposto pelo Professor João Batista, ministrando a cadeira de Algoritmos e Estrutura de Dados II. A fim de solucionar o desafio, é preciso encontrar a quantidade de copinhos de sorvete de 2 e 3 sabores que são possíveis nessa sorveteria, seguindo as condições que o dono estabeleceu, que serão apresentadas na seção 1. Serão abordadas duas soluções nesse relatório, sendo a primeira delas com alguns métodos prontos da biblioteca *networkx* (biblioteca que facilita a manipulação de estruturas de dados, tais como os grafos) na linguagem de programação Python e a outra com a implementação de grafos sem o auxílio de tal biblioteca. Também serão feitas análises a cerca dos resultados obtidos, desempenho dos algoritmos e comparação de tempo entre as duas diferentes soluções.

1 Introdução

Com o final do inverno e a chegada dos dias de primavera, seu primo acabou de abrir uma sorveteria que promete ser o sucesso da cidade, só que ele sempre foi o esquisito da família e a sorveteria tem algumas regras que preocupam a parentada:

- A sorveteria tem copinhos para 2 e 3 bolas de sorvete;
- Seu primo nunca coloca um sabor forte (ex.: chocolate mega-ultra-power-100%) em cima de um sabor suave (ex.: iogurte), pois daí ninguém sente o sabor do suave quando chegar nele;
- Seu primo tem uma lista de quem-é-mais-forte-do-que-quem para se orientar na hora de servir os opinhos (de acordo com a opinião dele, mas isso é outra história);
- Ele nunca aceita pedidos om sabor repetido, para que as pessoas possam experimentar mais sabores;

Agora sua família está um pouco preocupada por que essas condições poderiam limitar bastante a quantidade de pedidos que podem ser atendidos. Eles pedem que você (o especialista em informática da família, já que instala Windows pra todo mundo) pegue a lista de quem-é-mais-forte-do-que-quem e diga quantos copinhos diferentes de sorvete são possíveis na sorveteria. Este é um exemplo de lista de sabores, como pode se observar na Figura 1. Ela diz, por exemplo, que graviola deve ser servido antes de kiwi.

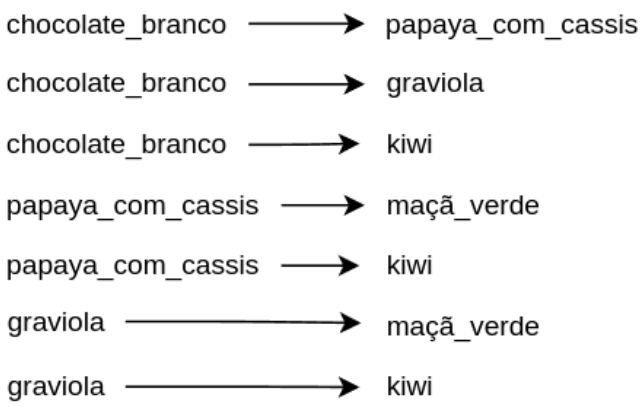

Figura 1: Caso Padrão

O grafo que representa quem-é-mais-forte-do-quem pode ser observado na Figura 2.

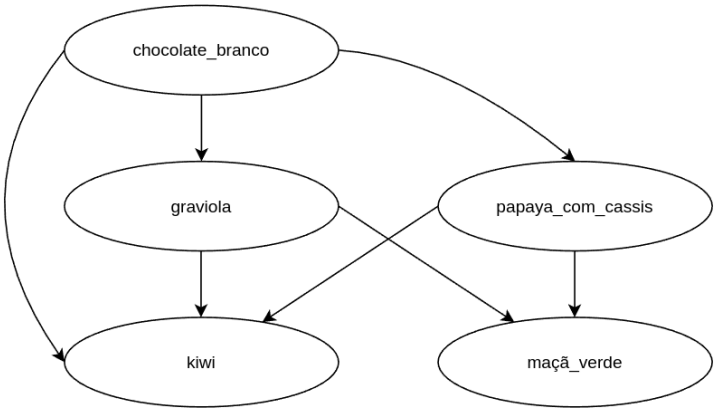

Figura 2: Grafo do Caso Padrão

Depois de analisar esta entrada de dados, você deve responder quantos copinhos de 2 e de 3 sabores podem ser servidos de maneira que os sabores fiquem na ordem desejada. Neste exemplo podem ser servidos 8 copinhos de 2 sabores e 4 copinhos de 3 sabores.

2 Primeira solução

Essa solução foi desenvolvida utilizando a biblioteca *networkx*, pertencente à linguagem de programação Python, que permite o estudo e manipulação de estruturas de dados, como por exemplo, os grafos que foram usados nessa implementação.
Tal implementação inicia-se adicionando cada linha do caso de teste em uma lista chamada *comb_sorvetes*. Logo em seguida essa mesma lista é iterada separando os caracteres de seta e adicionanado apenas os sabores à esquerda e à direita da seta em uma nova lista chamada *sabores*. A lista *sabores* possui apenas os sabores únicos do caso de teste.
Para a criação do grafo, é utilizado um dos métodos da biblioteca que permite a criação de um grafo direcionado instanciando um objeto do tipo: *DiGraph()* chamado *graph*. Para popular o objeto grafo é necessário iterar pela lista *comb_sorvetes* em que cada linha possui dois sabores sendo separados pelo caractere de seta. Logo, para adicionar os sabores no grafo, é preciso buscar apenas os valores que estão a esquerda da seta (sabor na posição 0) e a direita dela (sabor na posição 2), visto que a seta está na posição 1. Para essas inserções o método *add_edge(u, v)* é utilizado, como pode ser observado na Figura 3.

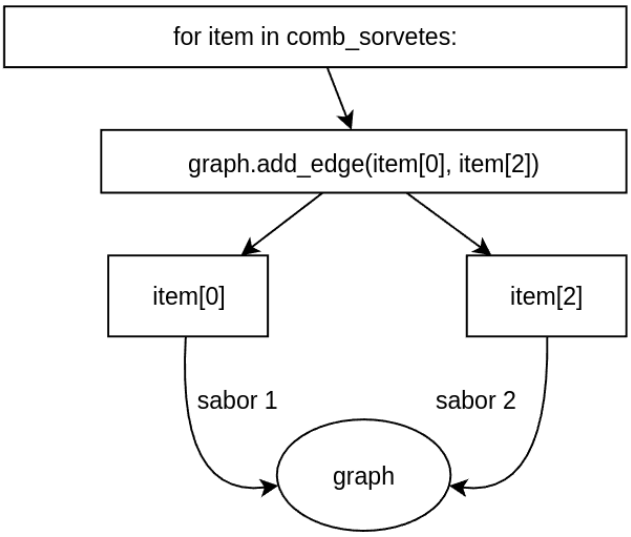

Figura 3: Inserção dos sabores no grafo

Para calcular a quantidade de copinhos de dois sabores, é criada uma lista chamada *copo2*. Para cada sabor da lista *sabores*, primeiramente o sabor é adicionado na lista *copo2* e após, verifica-se se ele possui nodos ancestrais. Para isso, utiliza-se o método *ancestors(graph, sabor)*, como pode ser observado na Figura 4

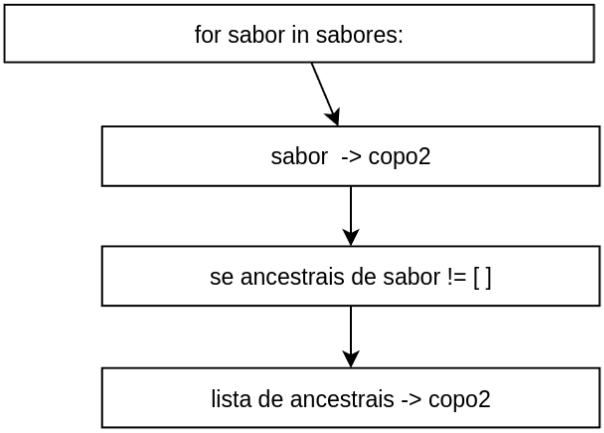

Figura 4: Inserção dos sabores e nodos ancestrais na lista copo2

Com o objetivo de encontrar a quantidade de copinhos de dois sabores, é preciso iterar pelos itens da lista *copo2*. Para cada item nessa lista, existe uma verificação que testa o tipo do item, caso ele seja do tipo *list*, a quantidade de copos de dois sabores é incrementada com o tamanho dessa lista. Isso acontece porque essa lista representa todos os nodos ancestrais do item, com os quais ele pode formar copinhos de dois sabores, como pode ser observado na Figura 5.

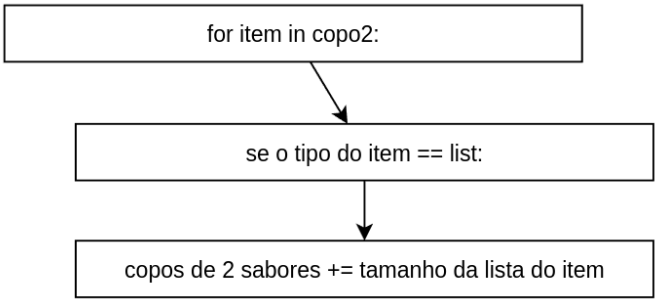

Figura 5: Contagem de copos de 2 sabores

Para calcular a quantidade de copinhos de três sabores, são feitas três iterações pela lista *sabores*. Para cada um dos itens que estão sendo iterados, há uma verificação para ver se eles não são iguais e também há uma verificação para saber se existe um caminho entre os três itens, utilizando-se um método chamado *has_path(graph, u, v)*. Se as duas condições forem satisfeitas, a quantidade de copinhos de três sabores é incrementada por 1, como pode ser observado na Figura 6.

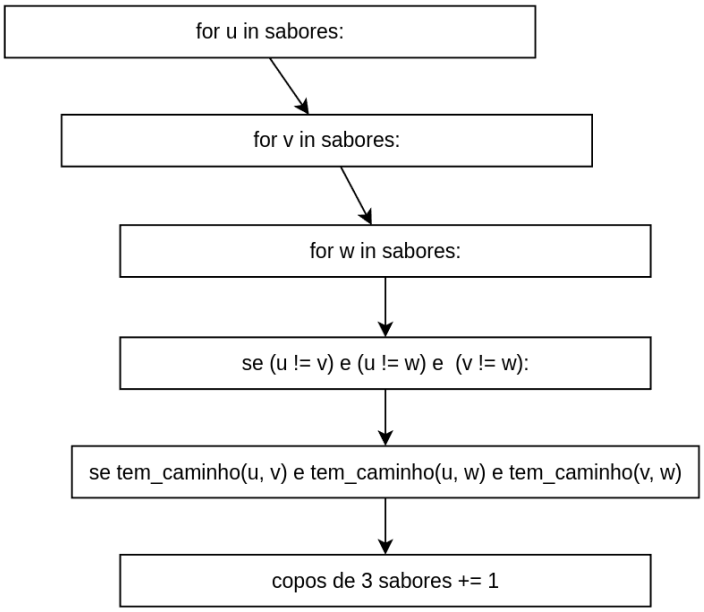

Figura 6: Contagem de copos de 3 sabores

3 Resultados Obtidos: Solução 1

Após executar o algortimo da primeira solução, podemos observar na Tabela 1 os resultados obtidos e os tempos de execução de cada caso de teste fornecido.

Casos	Copo 2	Copo 3	Tempo de Execução em segundos
caso-padrão	8	4	0.00
caso10	7	1	0.02
caso20	92	154	0.07
caso30	287	1249	0.14
caso40	522	3113	0.39
caso50	816	5804	0.83
caso60	1330	14624	1.74

Tabela 1: Resultados do Algortimo 1

É perceptível o aumento do tempo conforme o problema escala, pois os métodos prontos da biblioteca *networkx* não se preocupam com a otimização, mas sim com o funcionamento. Estes métodos podem estar executando outras subrotinas que, para o contexto desse problema, não se fazem necessários.

4 Segunda solução

Essa solução também foi desenvolvida com a linguagem de programação Python porém, a diferença em relação ao primeiro algoritmo, é que não é utilizada nenhuma biblioteca da linguagem, todos os métodos foram desenvolvidos e pensados para esse desafio.
Essa resolução começa de forma similar à outra. Para cada caso de teste o caractere de seta é excluído do arquivo para se ter apenas os sabores mais fortes e mais fracos. Uma lista chamada *sabores* é criada a fim de armazenar todos os sabores que não se repetem no caso de teste e também uma lista chamada *edges* é criada, tal lista é responsável por receber o sabor mais forte e mais fraco, respectivamente, do caso teste. Esses dois sabores são adicionandos em forma de sublista na lista de *edges*, como pode ser observado na Figura 7.

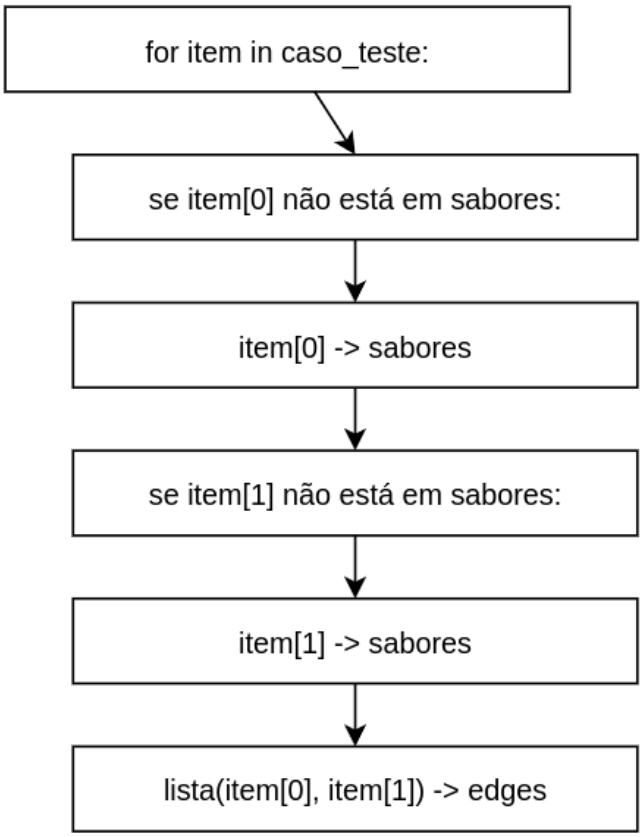

Figura 7: Populando a lista de sabores e edges

Para fazer o caminhamento no grafo recém populado, o método *Depth-first search* é criado. Seu funcionamento é da seguinte forma: o método recebe um nodo por parâmetro que é marcado como 1 num dicionário chamado *marca* e logo em seguida uma iteração é feita pela lista de sabores. Para cada sabor (nodo) na lista, há uma verificação para ver se o nodo passado por parâmetro do método de caminhamento e o sabor que está sendo iterado estão presentes na lista *edges*, e se o sabor que está sendo iterado ainda não foi marcado. Caso essas condições sejam satifeitas, o método de caminhamento (*dfs*) é chamado novamente por recursão sendo o sabor atual o novo parâmetro, como pode ser observado na Figura 8.

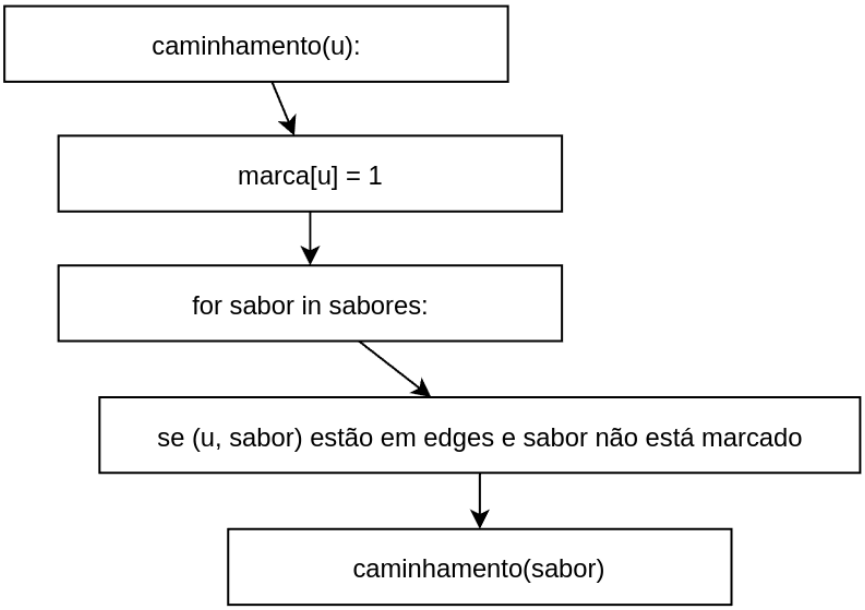

Figura 8: Caminhamento pelo Grafo

Para fazer a contagem da quantidade de copinhos de 2 e 3 sabores, é criado um dicionário chamado *caminhos* e uma lista chamada *aux*. Uma iteração pela lista de sabores é feita e para cada sabor, o dicionário *marca* é esvaziado e a lista *aux* também. O método de *dfs()* é chamado passando por parâmetro o sabor que está sendo iterado e, para cada nodo que está no dicionário *marca*, há uma verificação para saber se esse nodo não é igual ao sabor sendo iterado. Caso essa condição seja satisfeita, a lista *aux* recebe nodo e o dicionário *caminhos* tem sua chave igual ao sabor atual da iteração e o valor a lista *aux*, retornando um dicionário em que a chave é um sabor e o valor é uma lista de todos os sabores que o sabor chave tem ligação, ou seja, possui um caminho, como pode ser observado na Figura 9.

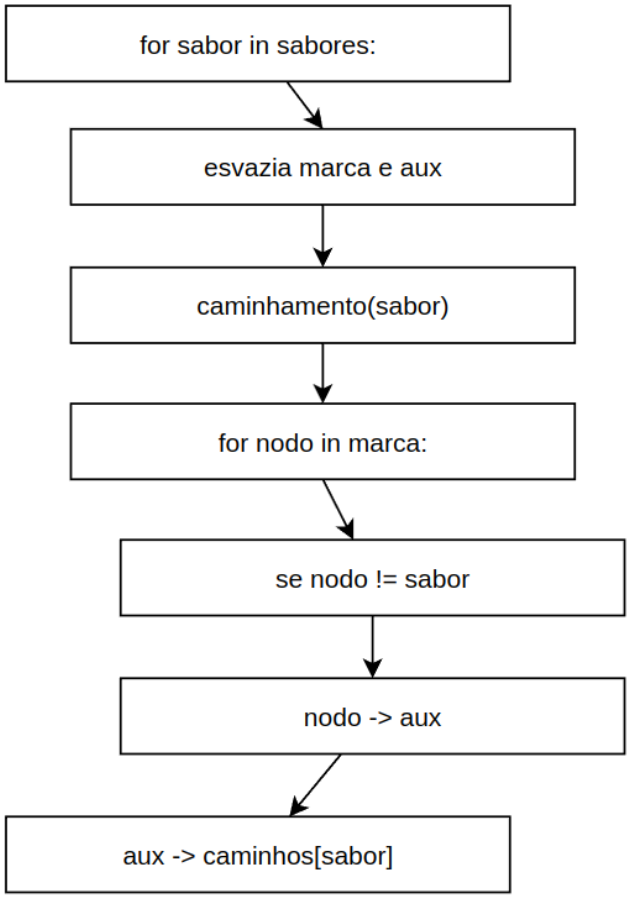

Figura 9: Criação da lista de caminhos

Para encontrar a quantidade de combinações de copinhos de dois sabores, são feitas duas iterações pela lista de sabores, fazendo duas verificações: se o primeiro sabor *u* que está sendo iterado pela lista for diferente do segundo sabor *v* sendo iterado e se o sabor *v* estiver nos valores do dicionário *caminhos* na posição na qual a chave for o sabor *u*. Se as condições forem verdadeiras, a quantidade de copinhos de 2 sabores é incrementada em 1, como pode ser observado na Figura 10.

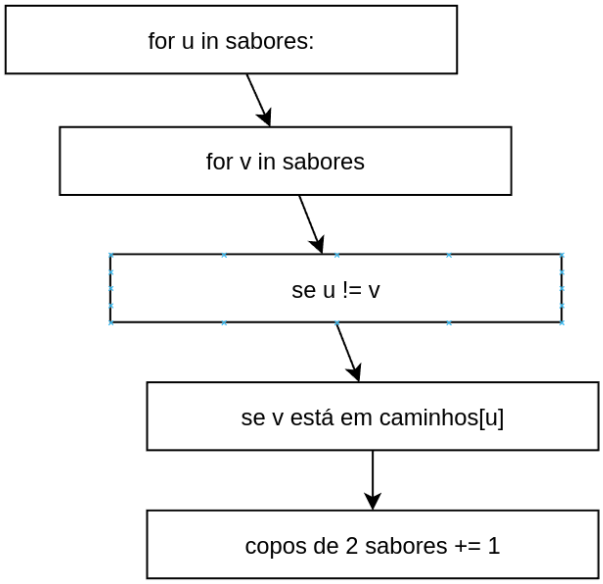

Figura 10: Incremento do copo de 2 sabores

Já para encontrar a quantidade de copinhos de três sabores, são feitas três iterações pela lista *sabores* em que, *u, v* e *w* são os sabores atuais de cada iteração. Também são feitas duas verificações, sendo que a primeira delas serve para checar se cada nodo (*u, v, w*) é diferente entre si e a segunda para ver se *v* está no dicionário *caminhos* na posição *u* e *w* está em *caminhos* na posição *v* e w está em *caminhos* na posição *u*, se ambas as verificações forem verdadeiras, a quantidade de copinhos de 3 sabores é incrementada em 1, como pode ser observado na Figura 11.

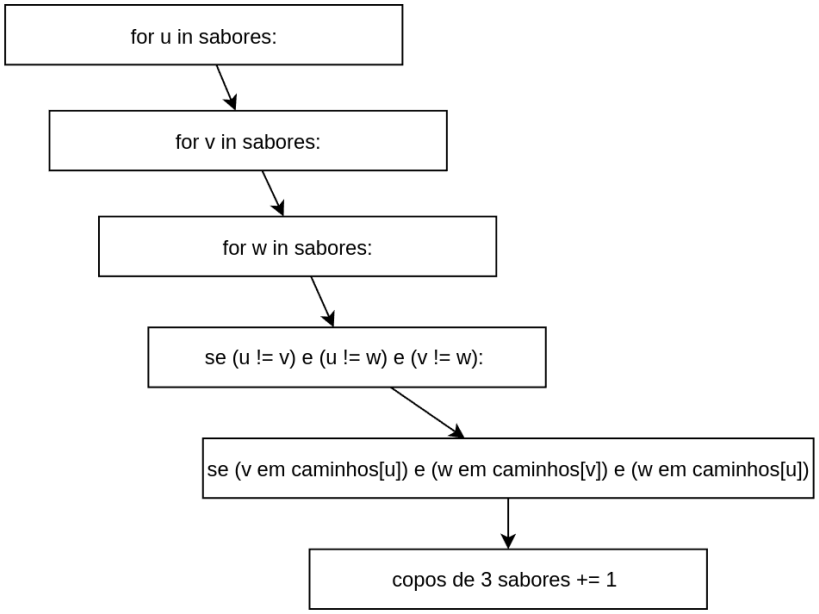

Figura 11: Incremento do copo de 3 sabores

5 Resultados Obtidos: Solução 2

Após executar o algoritmo da segunda solução, podemos observar na Tabela 2 os resultados obtidos e os tempos de execução de cada caso de teste fornecido.

Casos	Copo 2	Copo 3	Tempo de Execução em segundos
caso-padrão	8	4	0.00
caso10	7	1	0.00
caso20	92	154	0.01
caso30	287	1249	0.06
caso40	522	3113	0.08
caso50	816	5804	0.19
caso60	1330	14624	0.44

Tabela 2: Resultados do Algortimo 2

É nítido que essa solução possui um tempo de execução inferior ao primeiro algoritmo apresentado anteriormente na seção 2 deste relatório. Caso a sorveteria venha a possuir mais de ”cem”sabores de sorvetes diferentes, essa solução será a mais adequada para satisfazer o dono da sorveteria, visto que demorará muito menos tempo para ser executada e ele poderá concentrar-se mais em vender os ”cem”sabores de sorvete.
Tal desempenho deve-se justamente porque esse algoritmo é objetivo e conciso. Essa implementação foi elaborada especificamente para esse problema e, caso outros apareçam, é uma tarefa relativamente simples moldar esse algoritmo para suprir as novas demandas e desafios.

6 Comparação dos Resultados

Como pode ser observado na Figura 12, para cada novo caso de teste, o problema escala e exige um maior processamento do algortimo. Os tempos de execução de ambas soluções aumentam conforme o número de sabores de sorvete para serem processados, sendo perceptível que a segunda solução implementada mostrou-se mais eficaz para uso pois atingiu os mesmos resultados em um menor intervalo de tempo.

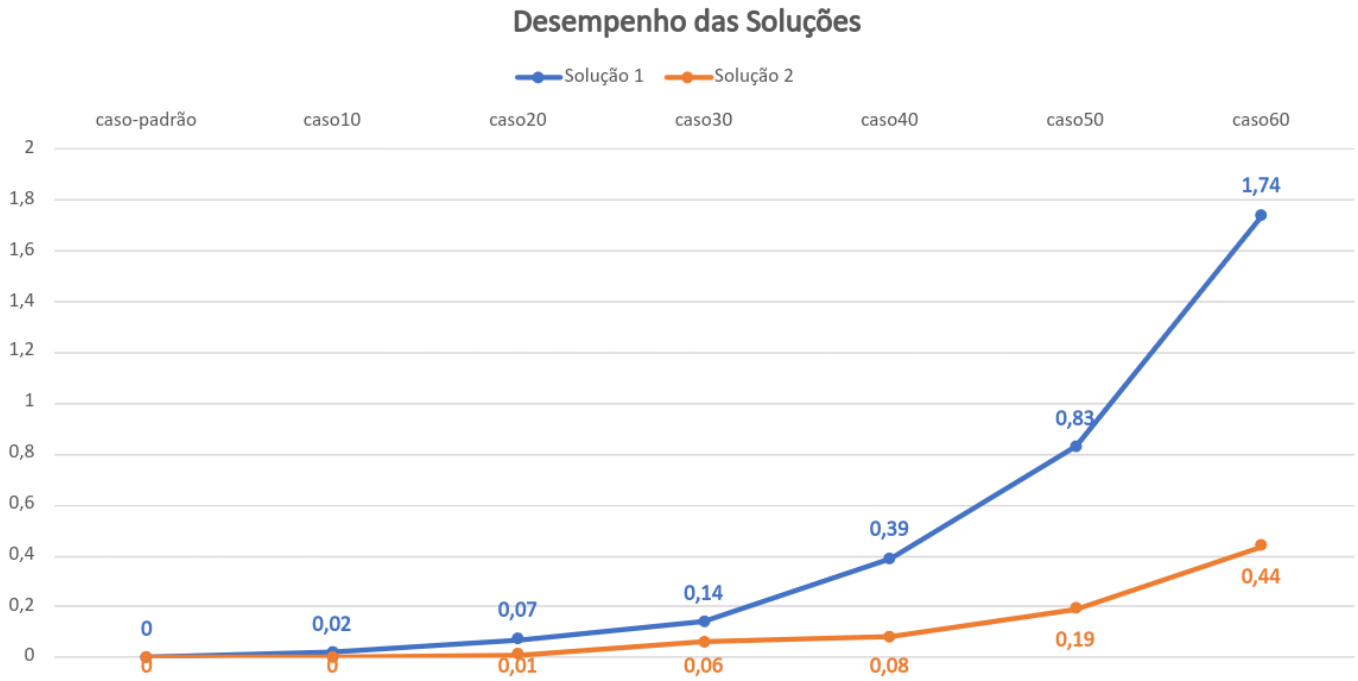

Figura 12: Comparação de Tempo, eixo X são os casos de teste e o eixo Y o tempo em segundos

7 Conclusão

As bibliotecas e métodos prontos que existem para as linguagens de programação, inclusive Python, são de enorme valor e com certeza auxiliam diversos profissionais da Tecnologia da Informação para os desafios e problemas que precisam solucionar no dia a dia. Mas, é de extrema importância que o profissional saiba quando deve usá-las ou quando faz-se necessário implementar uma solução exclusiva para o problema que está sendo tratado, focando em resolver um caso específico ou algum problema único, a fim de se obter uma melhor performance e um menor tempo de execução para a problemática em questão.
Tal afirmação pode ser observada no decorrer da leitura e entendimento desse relatório, pois são abordadas duas implementações diferentes que visam resolver o mesmo problema, sendo que ambas possuem a mesma precisão de resultados, porém o segundo algortimo é mais rápido. Essa diferença de desempenho entre elas ocorre justamente porque o segundo algoritmo foi desenvolvido e pensado especialmente para tratar e solucionar o desafio da Sorveteria Dos Horrores. Os métodos da segunda solução são curtos e objetivos a fim de encontrar os resultados no menor tempo, garantindo um algortimo com boa performance e que entregue os dados esperados.