

Mateus Ramos, Vitor Baza

**Projeto e Análise de Algoritmos**

**Procedimento de Algoritmo Genético**

*para solucionar o problema do caixeiro viajante simétrico*

18 de outubro de 2024

Campos dos Goytacazes

Sumário

[1.0 Manual de uso do programa 2](#_heading=h.gjdgxs)

[2.0 O problema 2](#_heading=h.30j0zll)

[3.0 A metaheurística 3](#_heading=h.1fob9te)

[3.1 Características 4](#_heading=h.3znysh7)

[4.0 Resolução do problema 4](#_heading=h.2et92p0)

[**4.1 Adaptando a lógica 4**](#_heading=h.tyjcwt)

[**4.2 Parâmetros 5**](#_heading=h.3dy6vkm)

[**4.3 Leitura dos dados 5**](#_heading=h.1t3h5sf)

[**4.4 Busca de solução inicial com algoritmo genetico 6**](#_heading=h.4d34og8)

[**4.4.1 Função geraPopulacaoInicialAleatoria 7**](#_heading=h.hbd9ycvmci2v)

[**4.4.2 Função mostrarPopulacao 7**](#_heading=h.17dp8vu)

[**4.4.3 Função avaliarPopulacao 8**](#_heading=h.l2uuvvplz4k8)

[**4.4.4 Função zeraVeto 8**](#_heading=h.5dot8hb0o2e)

[**4.4.5 Função torneio 9**](#_heading=h.5v2g91pbczrf)

[**4.4.6 Função cruzamento 9**](#_heading=h.hmfrh6oths65)

[**5.0 Fim do código 10**](#_heading=h.3j2qqm3)

[**6.0 Resultados encontrados 10**](#_heading=h.1y810tw)

[**7.0 Referências 11**](#_heading=)

# 1.0 Manual de uso do programa

Para utilizar o programa, basta utilizar um arquivo denominado “grafo.txt” dentro do mesmo diretório do programa e executá-lo.

Note que o arquivo grafo.txt deve conter na primeira linha o número de vértices e o número de arestas do grafo. As demais linhas armazenam a matriz distância D, onde o elemento dij armazena a distância da cidade i para a cidade j (1 ≤ i, j ≤ n).

# 2.0 O problema

O Desafio do Caixeiro Viajante Simétrico (PCVS) é um desafio na teoria dos grafos e na otimização combinatória. O propósito é descobrir a rota mais curta que percorre todos os vértices de um grafo uma única vez e retorna ao ponto inicial. Nesta questão, a distância entre os vértices é a mesma em ambas as direções, isto é, a distância do ponto A ao ponto B é a mesma que do ponto B ao ponto A.

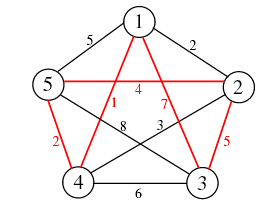


Figura 1 - *Problema do Caixeiro Viajante.* Retirado de *Técnicas de pesquisa operacional aplicadas na otimização de rotas de uma rede de lojas de materiais de construção (Matheus Fernando Moro).*

A solução eficaz deste problema é crucial em diversos usos práticos, tais como o direcionamento de veículos, o planejamento logístico e a otimização de rotas. O PCVS é um caso típico de um problema NP-difícil, o que implica que não existe um algoritmo polinomial capaz de solucioná-lo em um prazo razoável, a menos que P seja igual a NP. Assim, soluções eficazes para casos particulares ou abordagens são frequentemente aplicadas na prática para encontrar soluções adequadas em um período de tempo razoável.

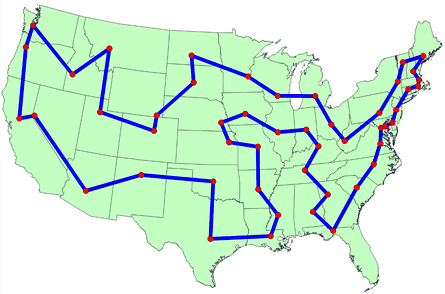


Figura 2: *Qual é a melhor sequência de rota para visitar todas as capitais dos Estados Unidos?* Retirado do post sobre o tema da Wikipédia

# 

# 3.0 A metaheurística

Agora que estamos familiarizados com o Problema do Caixeiro Viajante Simétrico, vamos investigar o instrumento que empregaremos para solucionar essa questão.

Os algoritmos genéticos são métodos de aprimoramento inspirados na evolução natural, empregando princípios como seleção, mutação e cruzamento para solucionar questões complexas de otimização combinatória. Os algoritmos genéticos, concebidos para resolver problemas onde encontrar a solução ideal é um desafio, se sobressaem ao equilibrar a exploração e a exploração do espaço de busca, possibilitando que se aproximem de soluções satisfatórias em um período de tempo razoável. A sua adaptabilidade e flexibilidade fazem deles ferramentas úteis em várias áreas, como o roteamento de veículos, a otimização logística e o projeto de redes.

## 3.1 Características

Os algoritmos genéticos são métodos heurísticos de otimização que se baseiam nos princípios da evolução natural, tais como seleção, cruzamento e mutação, com o objetivo de encontrar soluções aproximadas para questões complexas. Iniciam criando uma população inicial de possíveis soluções e, posteriormente, utilizam operações genéticas para aprimorar essas soluções ao longo de várias gerações.

Os atributos fundamentais de um algoritmo genético incluem:

* **Seleção:** Seleciona as soluções mais adequadas da população atual com base na sua aptidão (fitness), assegurando que as soluções mais eficientes tenham uma probabilidade maior de serem escolhidas para reprodução.
* **Cruzamento (Crossover):** Reúne duas soluções (pais) para gerar novas soluções (filhos), explorando novos territórios do espaço de pesquisa e possivelmente gerando combinações mais eficazes.
* **Mutação:** Realiza pequenas mudanças aleatórias em uma solução para preservar a diversidade populacional e prevenir que o algoritmo se acomode em locais excelentes.
* **Critério de Aceitação:** Estabelece as soluções que farão parte da próxima geração, preservando as de melhor performance e descartando as menos promissoras.

Esta estratégia adaptativa e fundamentada na evolução é extremamente eficiente para questões onde a busca por soluções ótimas é um desafio, como no caso do Problema do Caixeiro Viajante Simétrico. Ela assegura uma exploração equilibrada do espaço de busca e a convergência para soluções adequadas.

# 4.0 Resolução do problema

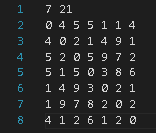
## 4.1 Adaptando a lógica

Para solucionar o Problema do Caixeiro Viajante Simétrico (PCVS) por meio de um algoritmo genético, é necessário integrar algumas ideias básicas. Os algoritmos genéticos funcionam produzindo uma primeira população de soluções (possíveis caminhos) e, posteriormente, alterando essas soluções por meio de operações de seleção, cruzamento e mutação, sempre com o objetivo de alcançar o melhor resultado possível.

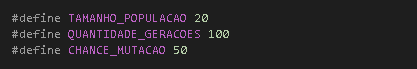
Dentro deste cenário, o algoritmo genético será responsável por criar e aprimorar diversas rotas para o caixeiro viajante, com o objetivo de reduzir a distância total percorrida. Em cada ciclo, as soluções serão aprimoradas, explorando novas combinações de trajetos e ajustando as já existentes até chegarem a uma rota que se aproxima do trajeto mais curto possível, que atravessa todos os pontos do grafo.

## 4.2 Parâmetros

Como parâmetros iniciais foram usados os dados de entrada base do algoritmo: quantidade de vértices, quantidade de arestas e uma matriz adjacência. Todos eles adicionados por um arquivo “grafo.txt”.

*.*

Além dos dados adicionados pelo arquivo de texto, temos também alguns valores criados como variáveis globais, que auxiliam no funcionamento do algoritmo:



A variável QUANTIDADE\_GERACOES armazena a quantidade máxima de gerações que o algoritmo deve realizar, simplificando a realização do ciclo evolutivo. A variável TAMANHO\_POPULACAO especifica o número de pessoas (rotas) que farão parte da população em cada geração. Em última análise, a variável CHANCE\_MUTACAO indica a chance de uma pessoa passar por uma mutação durante o processo de cruzamento.

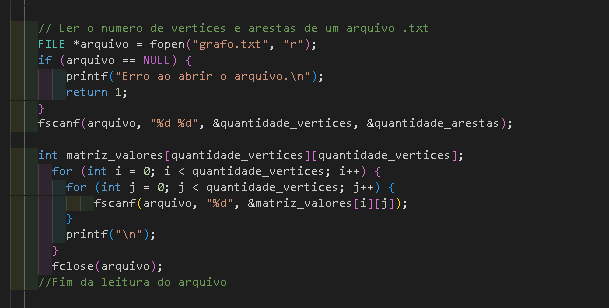
Tanto TAMANHO\_POPULACAO quanto CHANCE\_MUTACAO poderiam ser trocados por um critério mais flexível, fundamentado no desempenho dos indivíduos. No entanto, dado que os experimentos com este algoritmo serão conduzidos em grafos de pequeno porte, essas definições fixas são mais apropriadas para assegurar uma performance eficaz.

## 4.3 Leitura dos dados

Após estabelecer os parâmetros e entender a lógica do algoritmo, começaremos a leitura dos dados. Os dados se encontram em um arquivo.txt localizado no diretório do software. Neste arquivo, encontramos dados cruciais que mudam conforme a questão:



Utilizamos funções de leitura de arquivos para a leitura dos dados, com uma verificação de erro caso o programa não consiga acessar o arquivo. Depois dessa análise, examinamos as duas linhas iniciais, que indicam a quantidade de vértices e a quantidade de arestas, respectivamente. Depois, um loop guarda os valores adjacentes na matriz de distâncias, possibilitando que o algoritmo consulte com facilidade as distâncias entre os vértices.



## 4.4 Busca de solução inicial com algoritmo genetico

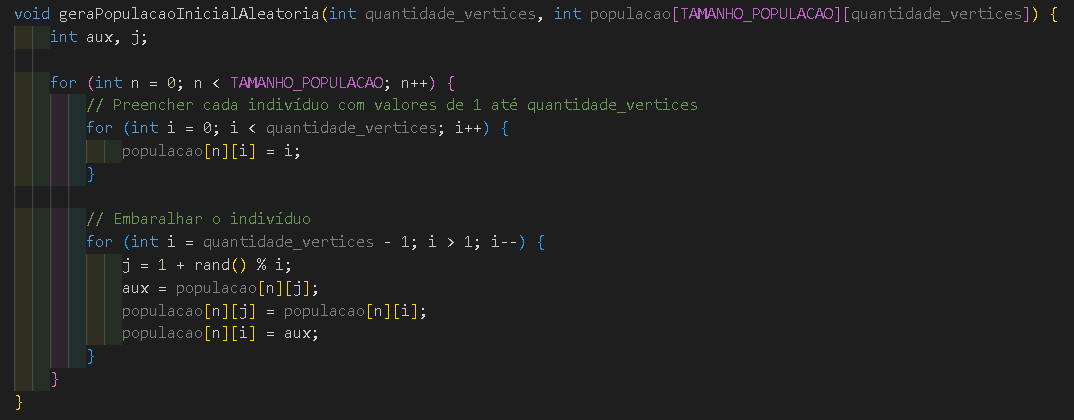
Antes de iniciar a procura por uma solução inicial válida, é necessário estabelecer as variáveis que serão empregadas. Elas englobam populacao, que simboliza a matriz da população corrente de rotas, e valores\_populacao, que guarda o valor (distância total) de cada trajeto identificado.

Após estabelecer essas variáveis, começamos a função geraPopulacaoInicialAleatoria, que será encarregada de gerar a população inicial de soluções.

## 4.4.1 Função geraPopulacaoInicialAleatoria

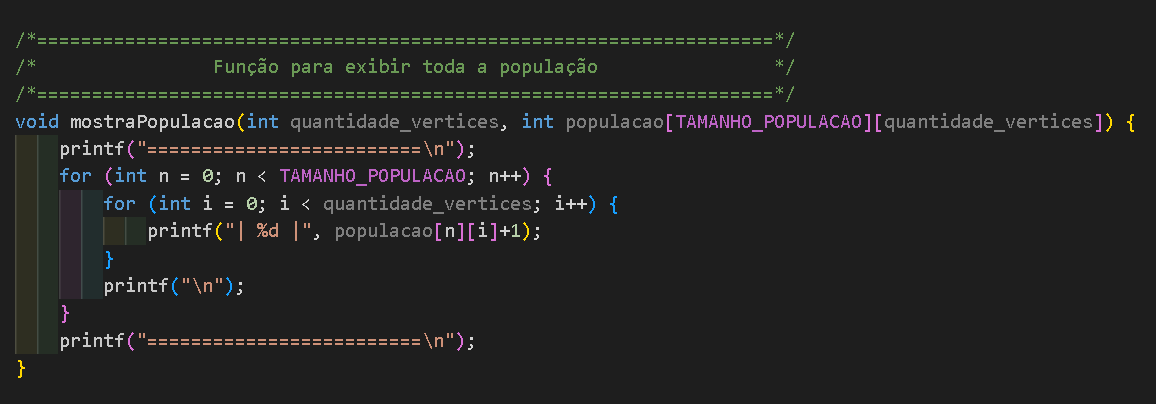
Na função, empregamos um loop para preencher cada pessoa com uma série de vértices, simbolizando uma possível trajetória. Depois, invertemos cada caminho para assegurar que as soluções resultantes sejam diversificadas e exploratórias.

O algoritmo cria uma combinação aleatória de vértices para cada rota, que será avaliada nas próximas iterações. Este procedimento de criação inicial é crucial para assegurar que o algoritmo possua um alicerce robusto de soluções a partir das quais possa progredir.



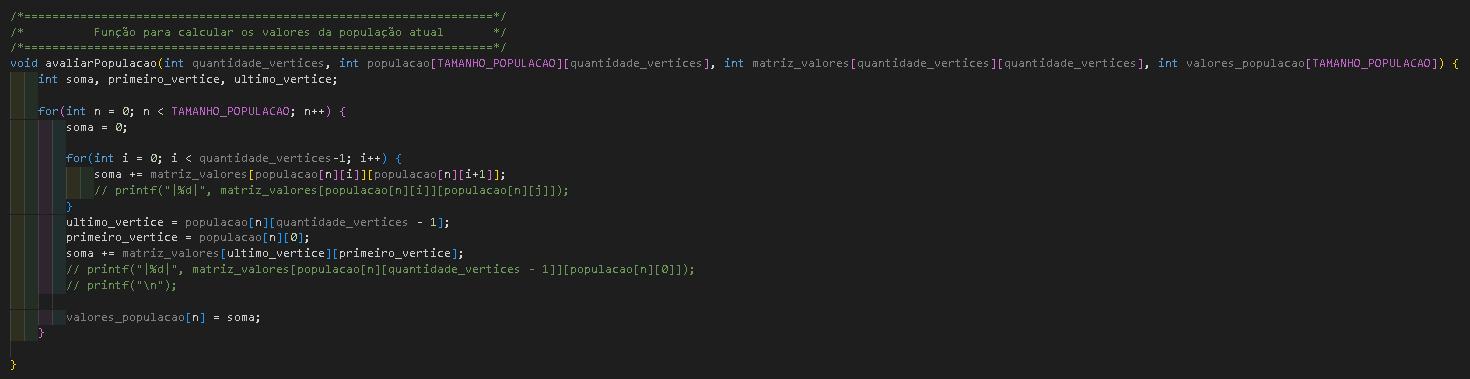
### 4.4.2 Função mostrarPopulacao

A função mostraPopulacao é empregada para mostrar o número atual de pessoas. Ela percorre cada indivíduo da população e exibe os vértices que formam o trajeto representado, aumentando cada vértice em 1 para tornar a leitura dos resultados mais fácil. Esta função é eficaz para observar e confirmar a diversidade da população ao longo do processo de evolução.



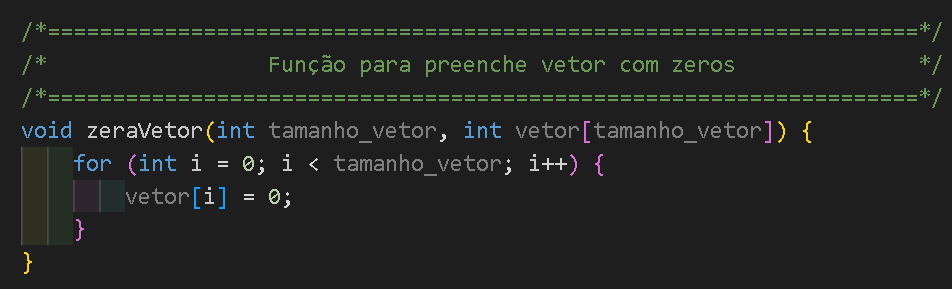
### 4.4.3 Função avaliarPopulacao

A função avaliarPopulacao determina o custo de cada pessoa na população, utilizando a matriz de valores (distâncias) disponibilizada. A função inicia uma variável soma para cada pessoa, acumulando o custo total do trajeto ao somar as distâncias entre cada par de vértices consecutivos. No término do loop, adiciona-se a distância entre o último e o primeiro vértice para encerrar o ciclo. O custo total é guardado no vetor valores\_populacao, que será usado para avaliar e escolher os indivíduos mais eficientes na próxima geração. Esta função desempenha um papel crucial na determinação da qualidade das soluções produzidas.



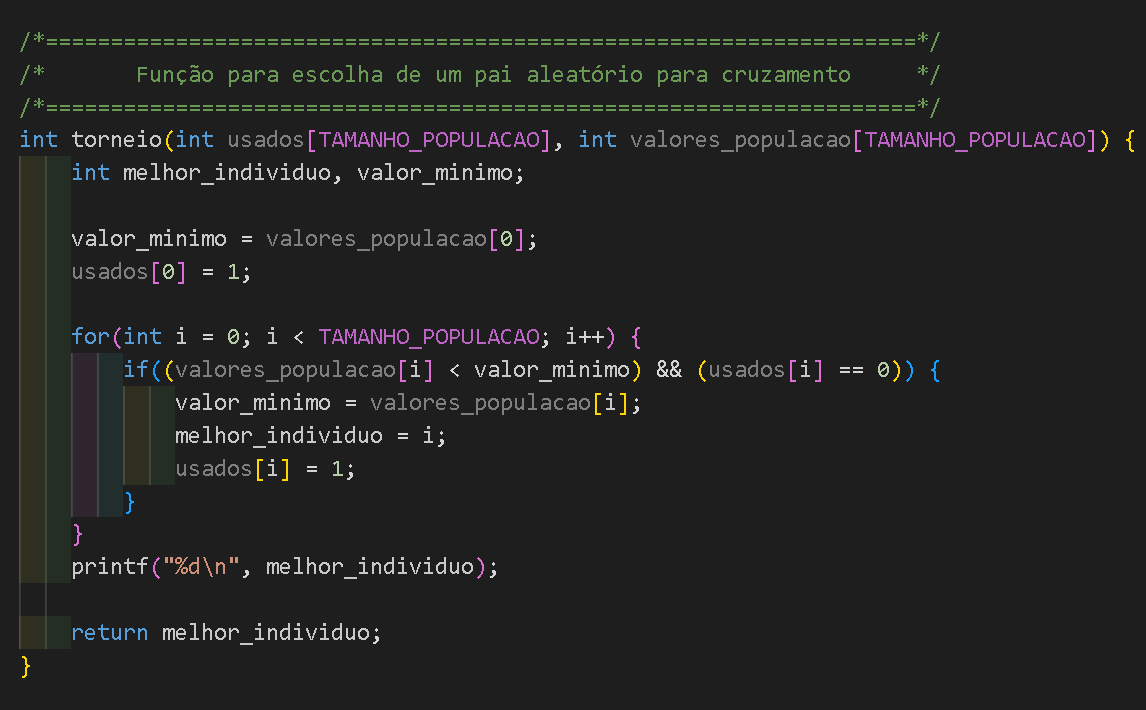
### 4.4.4 Função zeraVeto

A função zeraVetor é responsável por começar um vetor com zeros, realizando iterações sobre cada elemento e definindo seu valor como zero. Esta função é comumente empregada para recarregar o vetor que controla quais indivíduos já foram utilizados em uma competição.



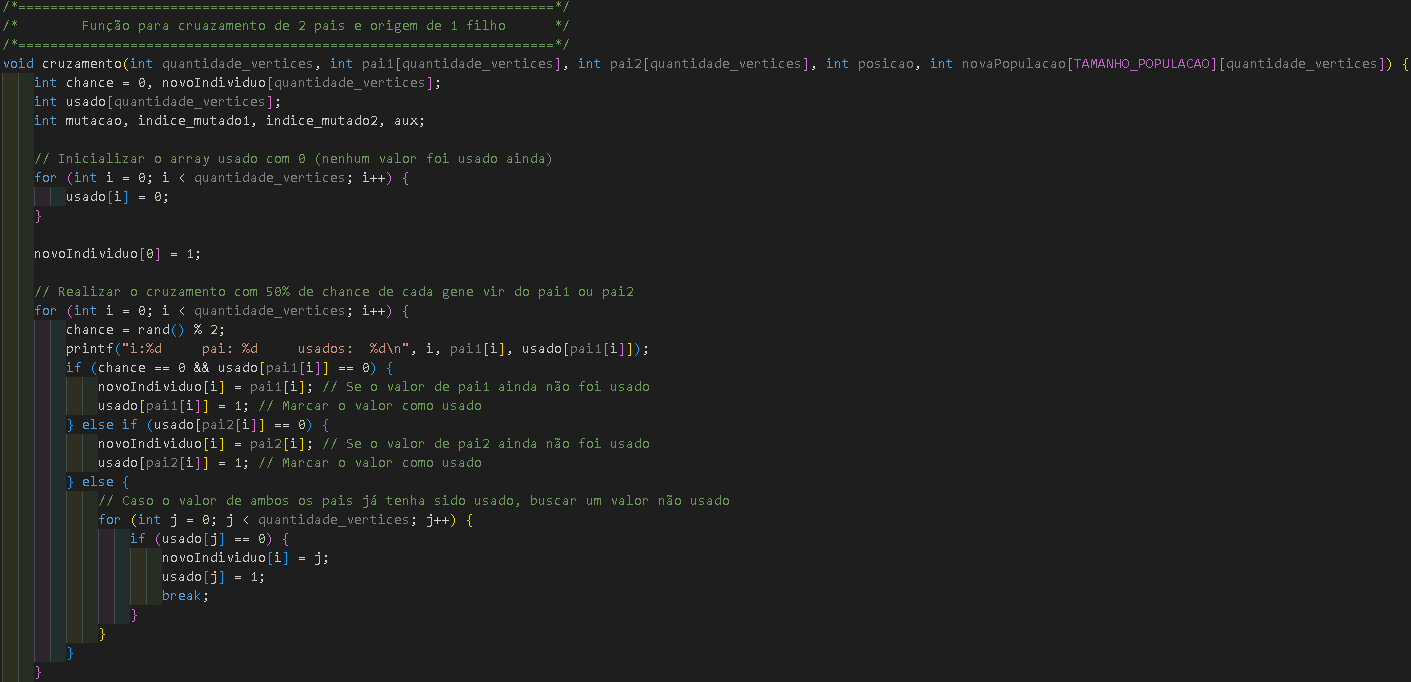
### 4.4.5 Função torneio

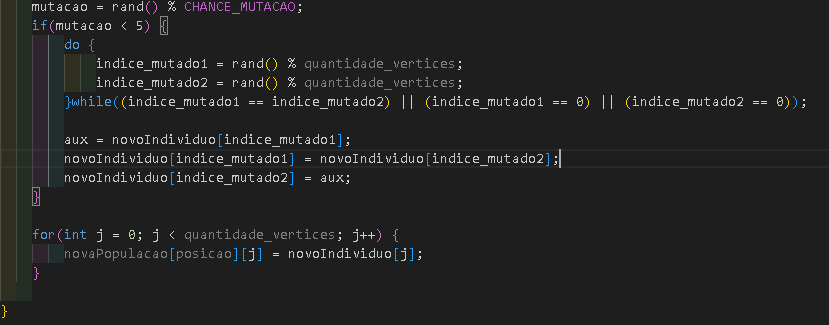
A função de torneio é encarregada de escolher um indivíduo através de um processo de competição. Ela estabelece inicialmente o valor\_minimo como o valor do primeiro membro da população e o classifica como usado. A função percorre a população, analisando se algum indivíduo não utilizado possui um custo inferior ao valor\_minimo e atualiza a escolha do indivíduo mais eficiente encontrado. O índice do indivíduo mais eficiente é devolvido, sendo essencial para a escolha dos pais para o cruzamento.



### 4.4.6 Função cruzamento

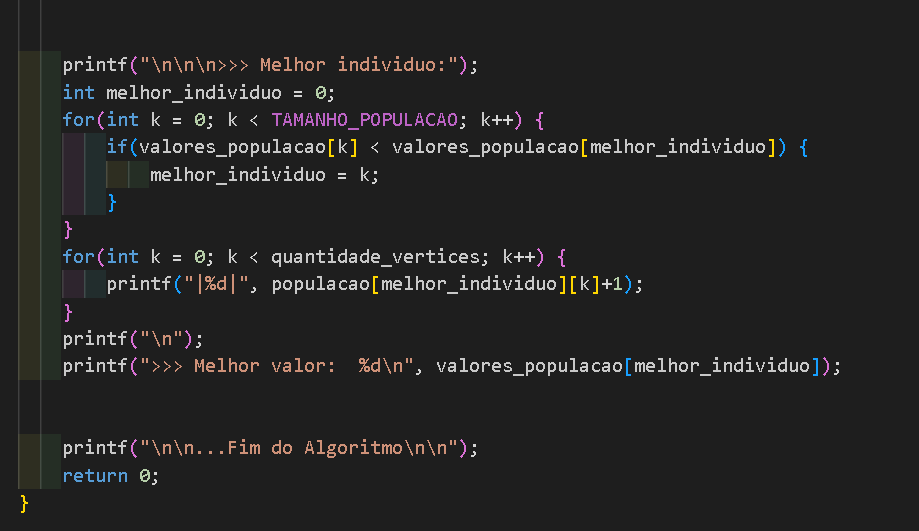
A função de cruzamento é responsável por gerar um novo ser a partir de dois progenitores escolhidos. Criamos um vetor denominado novoIndivíduo que simbolizará o filho resultante, usando uma lógica de probabilidade para determinar se cada gene (vértice) do novo indivíduo será herdado do pai1 ou do pai2. A função assegura a ausência de redundâncias de vértices no novo indivíduo, substituindo-os por vértices ainda não empregados. Depois de formar o novo ser, uma mutação pode acontecer com uma probabilidade específica, alternando aleatoriamente dois vértices. Por fim, o novo indivíduo formado é incorporado à nova população, desempenhando um papel crucial na criação de novas soluções.





# 5.0 Fim do código

O código se encerra com a execução do algoritmo, apresentando a solução mais eficaz juntamente com o custo mínimo. O algoritmo progride a população ao longo das gerações, efetuando avaliações, escolhas e combinações para encontrar a solução mais eficaz.



# 6.0 Resultados encontrados

Resultados aleatorizados por cada perturbação e o melhor resultado gerado pelo algoritmo, testados no grafo de grafo de Petersen.



# 7.0 Referências

**Algoritmos Genéticos.** Disponível em: <https://sites.icmc.usp.br/andre/research/genetic/>. Acesso em: 25 ago. 2024.

**Algoritmos Genéticos Aplicados Ao Problema Da Mochila Binária.** Disponível em: <https://pt.scribd.com/document/715911860/Algoritmos-Geneticos-Aplicados-Ao-Problema-Da-Mochila-Binaria>. Acesso em: 25 ago. 2024.

BATISTA, V. E. **Algoritmo Genético para o Problema da Mochila** - Vitor Emanuel Batista. Disponível em: <https://vitorebatista.medium.com/algoritmo-gen%C3%A9tico-para-o-problema-da-mochila-5910f90f9488>. Acesso em: 25 ago. 2024.

CARVALHO, Rubens. **Problema da Mochila**. Disponível em: <https://www.ime.unicamp.br/~mac/db/2015-1S-122181-1.pdf>. Acesso em: 25 ago. 2024.

PACHECO, Marco Aurélio Cavalcanti**. Algoritmos genéticos: princípios e aplicações**. Disponível em: <https://inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/CE-intro\_apost.pdf>. Acesso em: 25 ago. 2024.

]

DOS REIS, J. VON A. **Meta-heurística Algoritmo Genético aplicada no Problema da Mochila Binária**. Disponível em: <https://www.youtube.com/watch?v=ZCSbi3BnLB8>. Acesso em: 25 ago. 2024.

EDUARDO, P. **Algoritmo Genético E o Problema da Mochila – Trilhas Python**. Disponível em: <http://www.trilhaspython.com.br/2020/12/09/algoritmo-genetico-e-o-problema-da-mochila/>. Acesso em: 25 ago. 2024.

**Genetic algorithms**. Disponível em: <https://www.geeksforgeeks.org/genetic-algorithms/>. Acesso em: 25 ago. 2024.

ÓTIMO, Ponto **Algoritmo Genético aplicado ao Problema da Mochila.** Disponível em: <https://www.youtube.com/watch?v=FYF6lS\_BHKA>. Acesso em: 25 ago. 2024.

SANTOS, Philippe Leal Freire dos. **“Heurísticas para o Problema da Partição Cromática de**

**Custo Mínimo”**. 2018. Disponível em: https://site.ic.uff.br/wp-content/uploads/2021/09/872.pdf.

Acesso em: 16 fev. 2023.