

Felipe Soares, Kevin Perdomo, Mateus Terra, Pedro Matias

**Projeto e Análise de Algoritmos**

**Algoritmo Genético com Chave Aleatória Viciada**

**para solucionar o problema da mochila binária**

16 de Abril de 2023

Campos dos Goytacazes

[1.0 Manual de uso do programa](#_heading=)

[2.0 O problema](#_heading=)

[2.0 A metaheurística](#_heading=)

[3.0 Resolução do problema](#_heading=h.v6u28toexaml)

[3.1 Adaptando a lógica](#_heading=h.z05too4p6n2k)

[3.2 Parâmetros](#_heading=h.hg5hrbyd8r3a)

[3.3 Leitura dos dados](#_heading=h.s5mv5p7o0m3b)

[3.4 Primeira Geração](#_heading=h.7pvmzm5yufgx)

[3.5 Iniciando uma nova geração](#_heading=h.22myzxld05vd)

[3.6 Reprodução de gerações](#_heading=h.9s0y5ih4jb9p)

[5.0 Evolução dos resultados](#_heading=h.wwdyq4pmmnye)

# Manual de uso do programa

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

Note que o arquivo mochila.txt deve conter na primeira linha a capacidade da mochila, na segunda linha o número n de itens totais da mochila e em cada uma das próximas n linhas, o valor e peso de cada item.

# O problema

Para esse trabalho, estaremos procurando uma solução para o problema da mochila binária, porém para isso primeiro precisamos entender melhor o que é esse problema.

O problema da mochila convencional é um problema de otimização lógica que envolve encontrar a melhor combinação de itens possível para se armazenar em uma mochila, levando em conta que cada item () possui um valor () e um peso ou tamanho () que no total não pode ultrapassar uma capacidade máxima ().

O problema da mochila binária ou problema da mochila booleana, leva em conta o mesmo princípio, a única diferença é que cada item pode ser adicionado a mochila apenas uma vez. Assim, um item pode estar armazenado na mochila ou pode não estar armazenado na mochila .

Assim, o nosso objetivo é que:

Obedecendo ao princípio de que:

# 2.0 A metaheurística

Agora que sabemos um pouco mais sobre como funciona o problema a ser trabalhado, vamos entender melhor a ferramenta que vamos utilizar para resolver esse problema.

Como estamos falando de problemas de otimização, é certo pensar que os meios que usamos para chegar a um resultado demandam muito tempo e processamento, além de nem sempre resultarem na melhor combinação ou se aproximarem o suficiente dele. É com esse foco que surgem as metaheurísticas, estratégias para solucionar esses problemas, gastando o menor tempo possível e chegando na melhor combinação, ou no mínimo próximo o suficiente dela.

A metaheurística que vamos trabalhar é chamada de Algoritmo Genético com Chave Aleatória Viciada (BRKGA). Ele é uma metaheurística evolutiva, ou seja, trabalha de forma a criar variações dos indivíduos existentes de uma geração até que eles evoluam ao ponto mais próximo possível do resultado excelente.

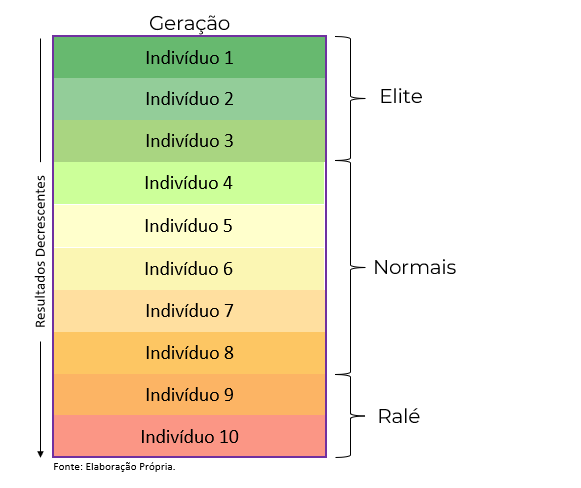
Mais detalhadamente, o BRKGA utiliza gerações de indivíduos onde cada indivíduo é referente a uma combinação possível do problema a ser resolvido. A primeira geração será criada aleatoriamente pelo algoritmo e logo após ordenada de acordo com os valores pedidos pelo problema. A partir dela vamos criar grupos seletos que vão interagir entre si, criando uma nova geração, e assim se manterá por uma quantidade escolhida de gerações.

As gerações são dividas em 3 grupos:

Elite: Grupo formado por uma quantidade pequena dos primeiros indivíduos da geração, representando assim os melhores indivíduos da nossa geração atual. Eles serão repassados para a próxima geração sem modificações.

Ralé: Formado por uma pequena minoria dos piores indivíduos da geração atual. Eles serão descartados e gerados novos indivíduos de forma aleatória. É graças a essa modificação que evitamos que nosso algoritmo fique viciado em um conjunto fixo de dados.

Normais: Aqueles que não fazem parte da elite e nem da ralé. São os indivíduos centrais do nosso algoritmo, eles passarão por uma etapa de reprodução, onde produzirão um novo grupo por meio de reprodução com a elite.



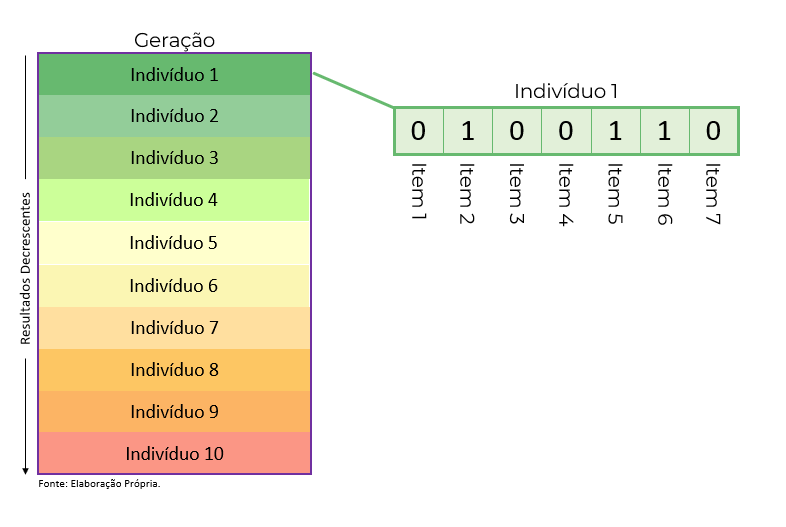
A reprodução de indivíduos normais é feita por meio do cruzamento dos indivíduos normais da geração atual e de indivíduos da elite. É utilizado um parâmetro de reprodução, geralmente de 50%, para que o indivíduo filho seja gerado com as características do pai da elite ou do pai normal.

A principal característica do BRKGA é o método de evolução dos seus indivíduos. Enquanto outras metaheurísticas trabalham para buscar o melhor resultado por meio da aleatoriedade e a tentativa e erro, o BRKGA se utiliza do conceito já existente na biologia de evolução. Ele trabalha mantendo os resultados ótimos de cada geração, modificando os resultados considerados bons para que eles evoluam e trocando os resultados abaixo do esperado. Com essa técnica ele consegue, além de manter os resultados bons, gerar outras combinações aleatórias a cada geração, o que permite que ele tenha sempre novos dados inseridos nas próximas gerações, evitando o vício do algoritmo a um pequeno grupo de informações.

# 3.0 Resolução do problema

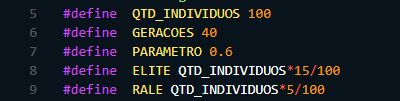
### 3.1 Adaptando a lógica

Para resolvermos o problema da mochila binária utilizando o BRKGA primeiro vamos cruzar os conceitos. O BRKGA trabalha com gerações de indivíduos onde cada indivíduo é referente a uma combinação possível do problema, logo, para o aplicarmos no nosso problema vamos levar em conta que cada indivíduo é referente a uma combinação possível da mochila, que no nosso algoritmo será representado por um vetor elementos binários onde cada elemento é referente a um item da mochila.



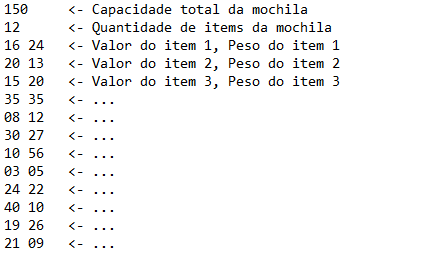
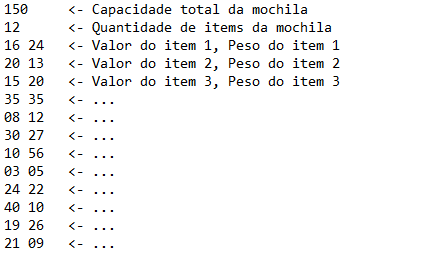
### 3.2 Parâmetros

Inicialmente trabalhamos com os valores de quantidade de indivíduos, quantidade de gerações, porcentagem de indivíduos da que formam a elite e que formam a ralé. Esses valores podem e devem ser modificados de acordo com a quantidade de itens escolhidos para o problema. Os valores atuais foram atribuídos por meio de testes que demonstraram que para a quantidade de itens que estamos utilizando os melhores parâmetros a se seguir seriam os atribuídos



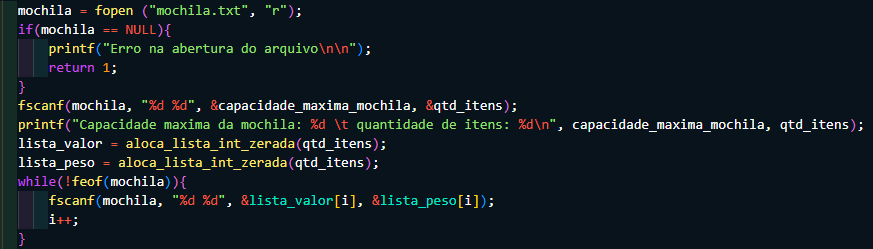
### 3.3 Leitura dos dados

Tendo os parâmetros definidos e a nossa lógica em mente, agora começaremos a leitura dos dados. Os dados se encontram em um arquivo .txt no nosso diretório. Dentro deles estão os principais dados que variam do problema, eles são respectivamente:

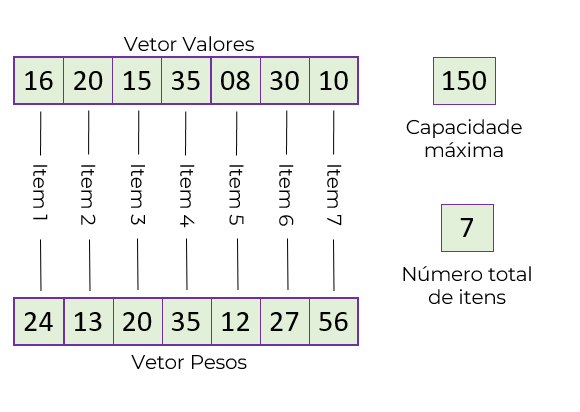
 

*(valores unicamente ilustrativos)*

Para leitura dos dados utilizamos as funções de leitura de arquivos normais com uma verificação de erro caso o programa não consiga ler o arquivo. Após isso, lemos as 2 primeiras linhas, que possuem os dados referentes a capacidade da total da mochila e a quantidade de itens, respectivamente. E terminamos a leitura com um loop que vai armazenar os valores e pesos de cada item em dois respectivos vetores.

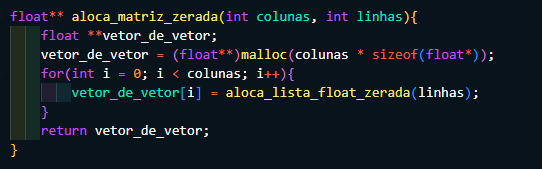


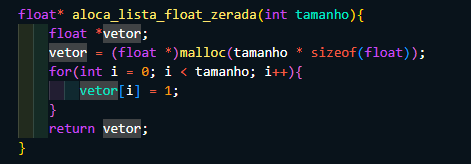
Ao final da leitura recebemos os seguintes dados em vetores e variáveis:



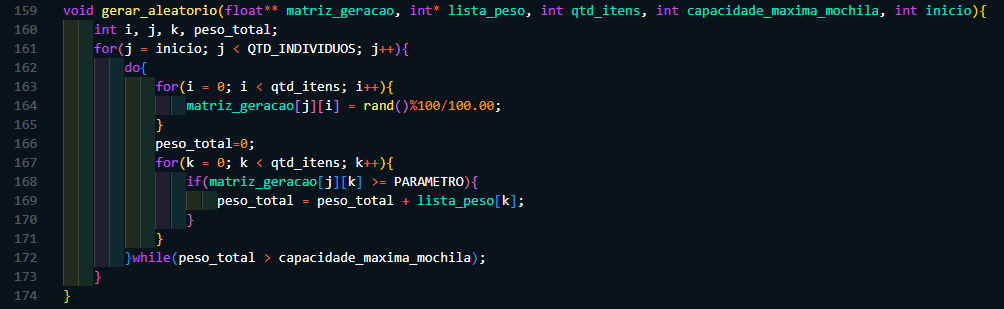
### 3.4 Primeira Geração

Com a leitura finalizada, vamos começar a criar e alocar nossas principais variáveis, a matriz da nossa geração atual, uma matriz duplicada para auxiliar na reprodução dos comuns e dois vetores que também vão participar da reprodução, o vetor “pai” e o vetor “mae”. Para isso vamos utilizar as funções “aloca\_matriz\_zerada” e “aloca\_lista\_float\_zerada”, elas são funções responsáveis por criar as variáveis tridimensionais e bidimensionais, respectivamente, e percorrer todo seu interior alocando valor 0 a todos os seus elementos.





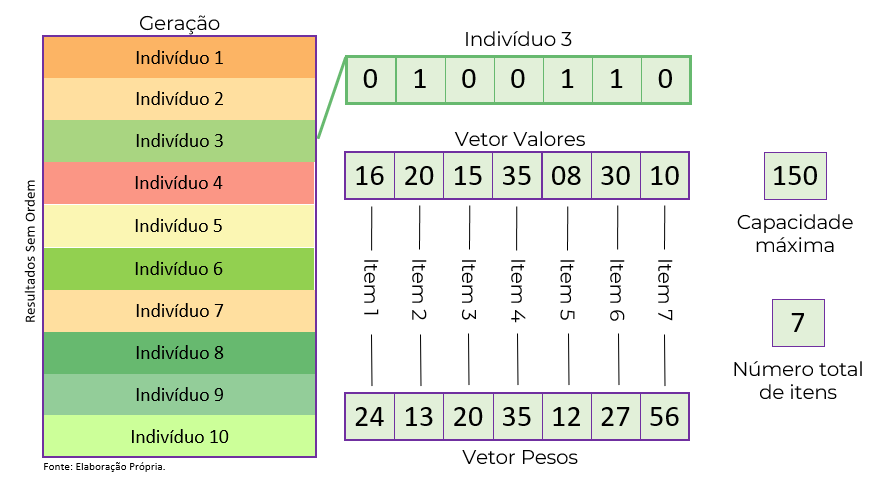
Agora podemos começar a criar a primeira geração de combinações do nosso problema. Lembrando que a primeira geração é uma geração gerada aleatoriamente. Para isso vamos usar nossa função “gerar\_aleatorio”:



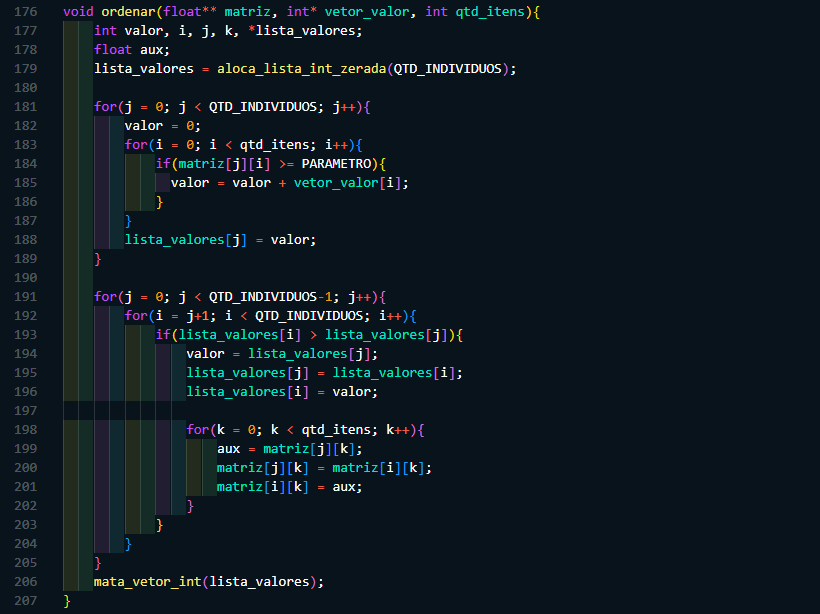
Inicialmente essa função vai precisar de um loop externo (linha 161) para que percorra todos os indivíduos da geração (que até o momento estão zerados), e uma vez dentro de um dos indivíduos precisamos de outro loop para que possamos acessar todos os itens da mochila (linha 163). Usando uma função que gera um número aleatório e dividindo-o por 100 temos um número entre 0 e 1, esse número será “arredondado” a partir do nosso parâmetro pré-definido, assim saberemos se o indivíduo tem (1) ou não tem (0) o item na mochila.

Obviamente como a combinação é aleatória ela pode estar ultrapassando o peso máximo da mochila, para verificarmos isso criamos mais um loop (linha 167) que irá somar os pesos dos itens que o indivíduo possui, do vetor de pesos. Caso a verificação (linha 172) indique que o peso está acima da capacidade máxima o algoritmo é obrigado a recriar toda a mochila de forma aleatória, e se mantém assim até que o peso esteja dentro dos conformes.

Depois de todo o processo temos a seguinte organização:



Com a matriz geração criada e com os indivíduos preenchidos com suas respectivas combinações, temos agora que ordenar toda a matriz em ordem de valor total da mochila, para que possamos seguir para a próxima geração. Para isso vamos utilizar a função “ordenar”:



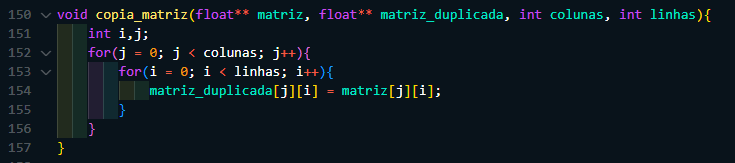
Para essa função, criamos e alocamos um novo vetor chamado “lista\_valores” (linha 179) este vetor vai nos auxiliar a ordenar a matriz geração mais a frente.

Com um loop dentro da matriz geração percorremos todos os nossos indivíduos (linha 181). E dentro de cada indivíduo temos mais um loop para percorrer todos os itens (linha 183), assim vamos verificar se o indivíduo [j] tem o item e se tiver vamos somar o valor do item ao valor total de sua mochila no vetor criado pela função “lista\_valores[j]” (linhas 184 a 188).

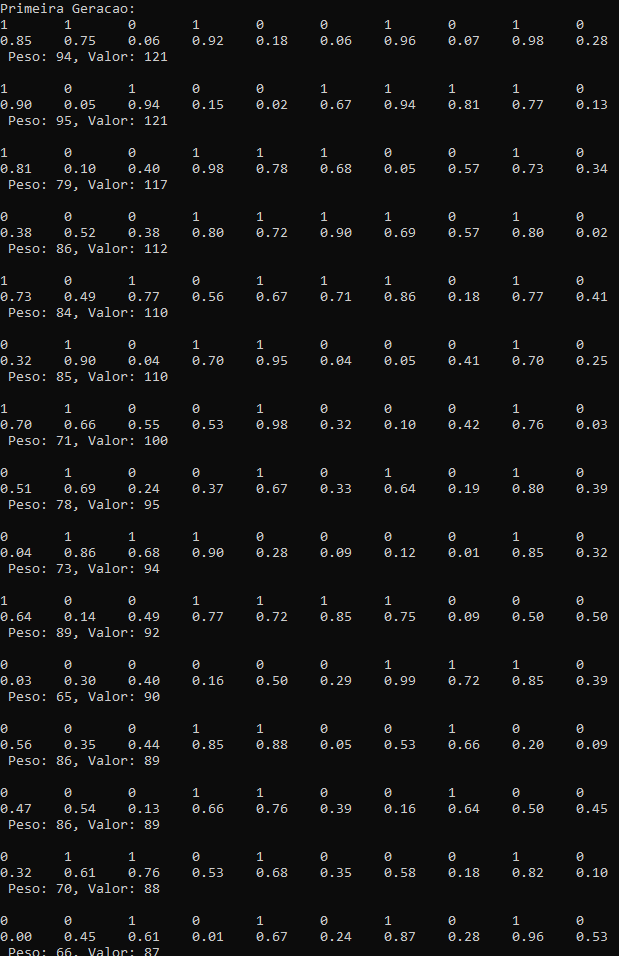
Para a ordenação vamos usar o vetor “lista\_valores” como base e o método de ordenação bubble sort, ordenando ao mesmo tempo o vetor e a matriz geração (linhas 191 a 205).

### 3.5 Iniciando uma nova geração

Agora para finalizar a nossa geração atual e começar o processo de criar uma nova geração, vamos precisar primeiro de criar uma cópia da geração atual, para isso vamos simplesmente usar a função “copia\_matriz” que vai fazer todo o processo.



Agora apenas para a visualização do que acontece no algoritmo temos uma função simples chamada “decodificador” para mostrar os resultados da geração na tela. Resumidamente o que é mostrado é:

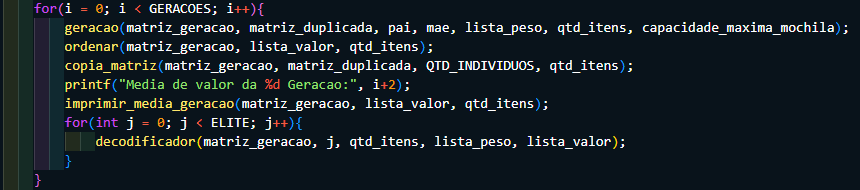


*(os dados continuam para baixo até o fim da geração)*

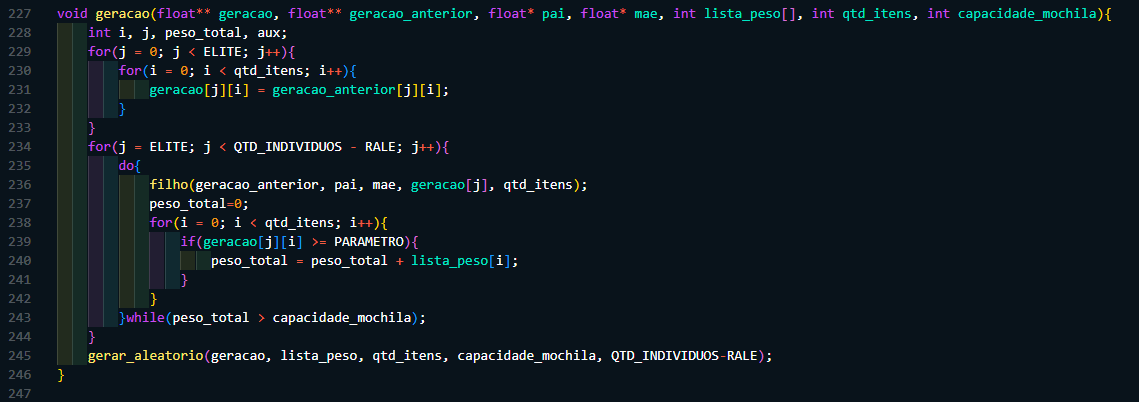
O que está sendo mostrado é a sequência de itens (colunas) para cada indivíduo (linhas), se ele possui (1) ou não possui (0) o item e logo abaixo o valor não “arredondado” da posse do item, baseado no nosso parâmetro (0,6) predefinido. Por fim, a linha abaixo mostra o peso total da mochila de cada indivíduo e o valor total da mochila de cada indivíduo.

### 3.6 Reprodução de gerações

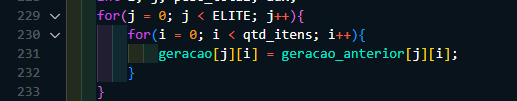
Finalmente podemos dar início a reprodução dos indivíduos e criar novas gerações. Lembrando, precisamos dividir os indivíduos em Elite, Normais e Ralé e cuidar para que cada um passe por seu determinado processo. Daqui em diante chamaremos a nossa “matriz\_duplicada” de geração anterior e a nossa matriz de geração vai ser modificada para se tornar nossa próxima geração.



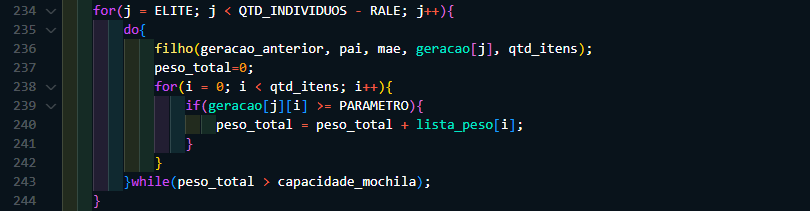
Os procedimentos que vamos fazer para evoluir nossa geração para a próxima, serão padrão para toda geração, então podemos ter um loop envolvendo todo o procedimento e repetir a quantidade de gerações pré-definidas no início do programa.



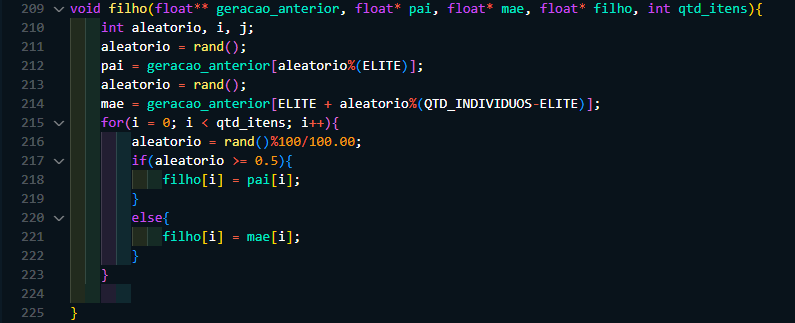
De todos os procedimentos que temos que fazer, nós já realizamos a maioria deles, então, vamos começar com a geração. Ela é a função mais complexa que teremos que fazer, então será dividida em partes para explicação. começando com a elite:



O procedimento da elite é bem simples. Como já falamos, a elite é o grupo de indivíduos que apresentou melhores resultados na nossa geração anterior, assim tudo que precisamos fazer é passar os indivíduos para a próxima geração sem nenhuma modificação, para isso basta usar dois loops simples com o tamanho da elite, padrão pré-definido, e copiar todos os indivíduos.

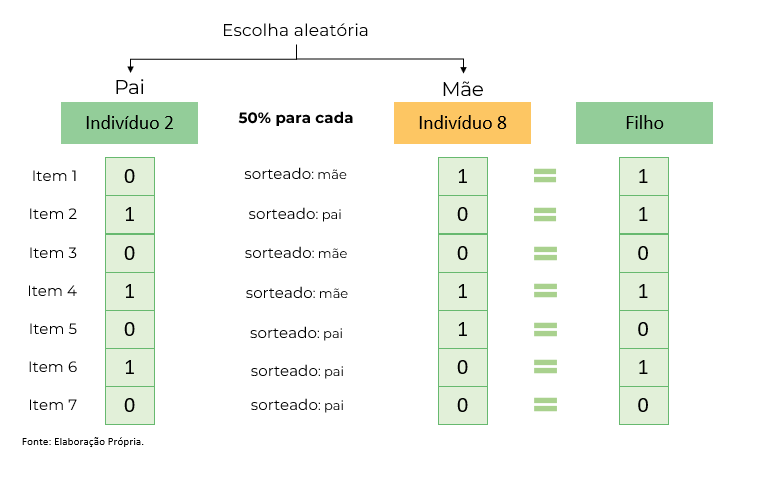


Os normais tem o tratamento mais complexo, passa-se um loop do tamanho da quantidade de indivíduos normais [elite à (quantidade de indivíduos - ralé)] para podermos acessar cada indivíduo na matriz. Já dentro de um indivíduo chamamos a função filho:



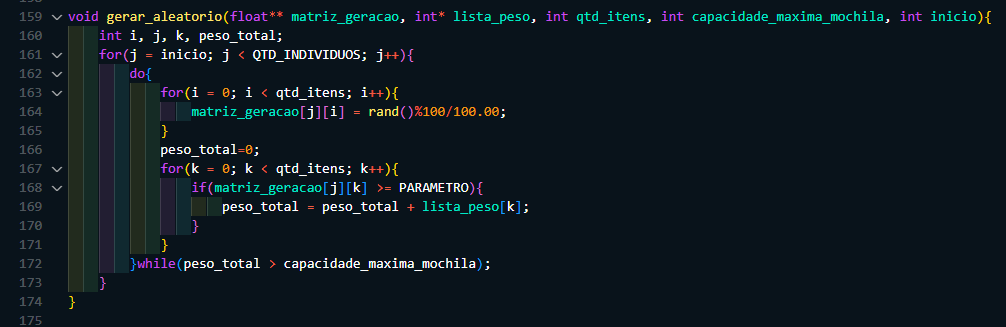
Já dentro da função “filho”, consideramos o indivíduo da nova geração como “filho” e vamos preencher dois vetores que criamos no início do programa, “pai” que será preenchido com a combinação de um indivíduo da elite (escolhido aleatoriamente) e “mae” que será um indivíduo dos comuns da geração anterior (escolhido aleatoriamente).

Já com os dois vetores preenchidos com elementos booleanos (0 ou 1) referentes a cada item, vamos usar um loop para acessar cada item do “filho”, agora devemos decidir qual dos dois pais irá ceder o “cromossomo” para o filho, ou seja, dado o item [i] qual configuração o filho receberá, a do pai ou da mãe? Para isso vamos usar novamente a aleatoriedade, assim o filho terá 50% de chance de receber a configuração do pai e 50% de chance de receber a configuração da mãe.

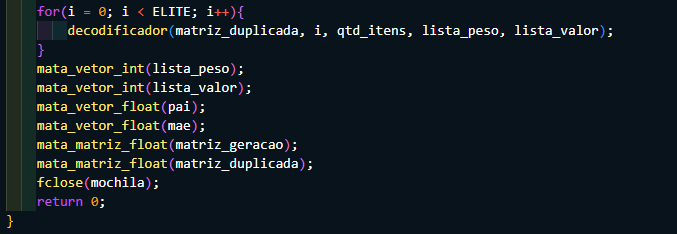


Logicamente a geração de um novo filho por meio de reprodução cromática é algo aleatório, o que permite que o peso máximo da mochila esteja acima do permitido. Para resolver esse problema envolvemos todo o processo dos normais com um loop que permitirá que o programa continue somente se o peso da combinação do indivíduo filho estiver dentro do padrão.

Para finalizar a função “geracao” vamos lidar com a ralé, ele são as piores combinações da nossa geração anterior e devem ser removidos e gerados novamente por meio de aleatoriedade. Para isso podemos chamar novamente a função “gerar\_aleatorio” que utilizamos na geração da primeira geração, mas desta vez passando o início da matriz como a quantidade de indivíduos menos a ralé, sendo a ralé uma quantidade de indivíduos pré-definida no início do programa. Ela irá apenas refazer toda a combinação dos indivíduos da ralé adicionando uma nova.



Continuando no nosso loop de geração, o que faremos agora é passar a nossa nova geração pelos mesmos passos que passamos a primeira. Vamos ordenar ela de acordo com o valor total de cada indivíduo, copiar o seu conteúdo para a “matriz\_duplicada” e mostrar os resultados na tela (agora apenas os resultados de elite, diferente do que foi mostrado da primeira geração). Esse procedimento vai ser repetido até que atinja o número de gerações pré definidas.

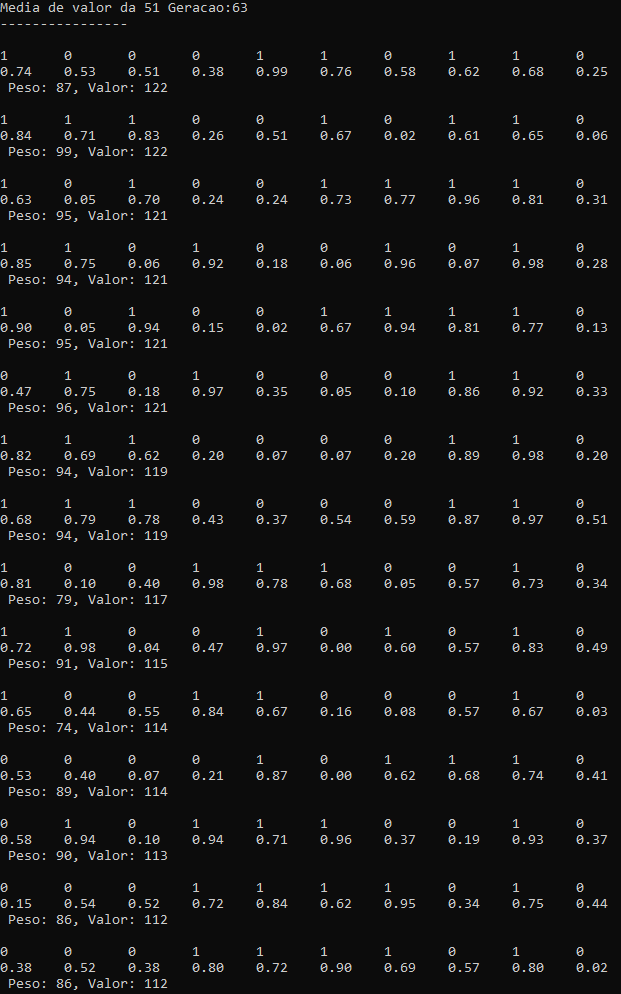


Para finalizar nosso programa, já estando na última geração e com uma combinação final o mais perto do excelente, vamos apenas mostrar na tela os indivíduos da elite da nossa geração final, eles representam as melhores combinações que nosso algoritmo pode encontrar.

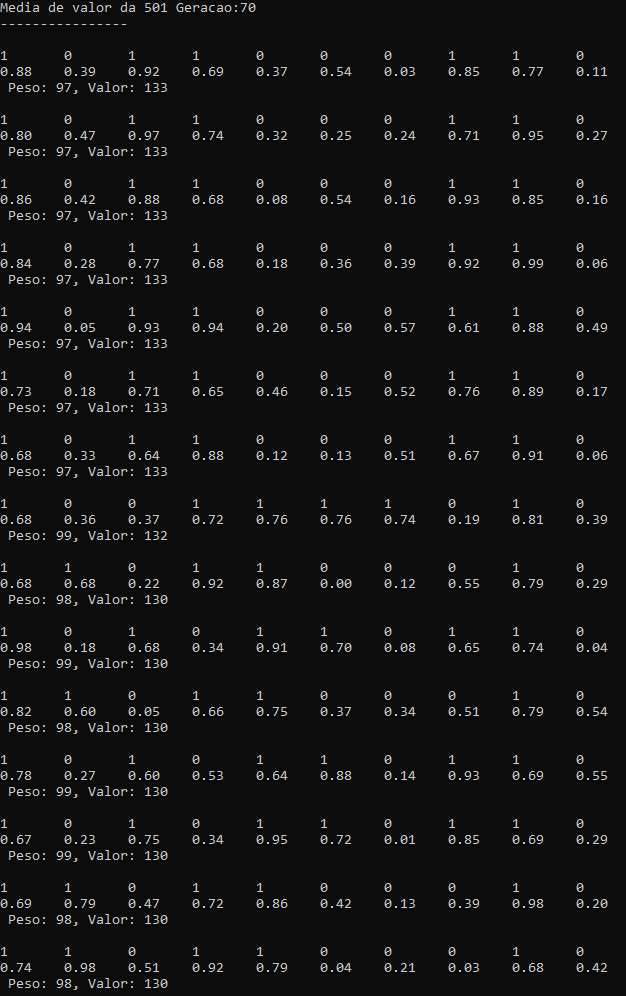
Limpamos as variáveis e fechamos a leitura do arquivo mochila.txt, deixando nosso programa pronto para ser utilizado novamente. Assim finalizamos por completo nosso programa.

# 5.0 Evolução dos resultados

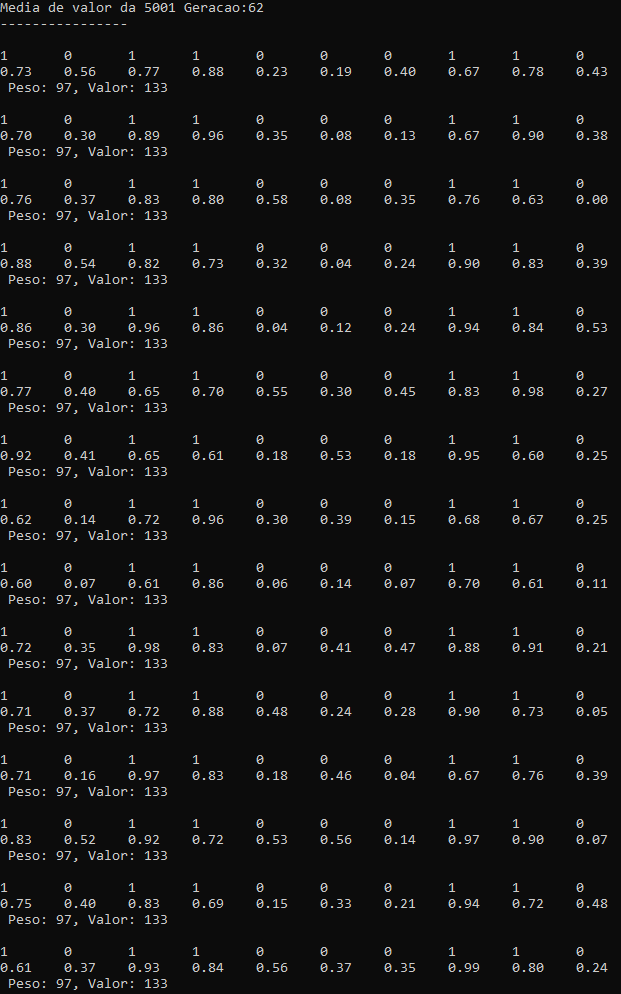
Pelo fato de BRKGA ser um algoritmo evolutivo, quanto mais gerações e indivíduos forem gerados, mais aprimorados vão ser os seus resultados. No caso do problema da mochila binária, melhores serão as combinações que encontraremos. Veja abaixo os resultados de alguns testes feitos, modificando o número de gerações:



50 gerações: Valores e pesos totais das mochilas variando muito. Combinações imprecisas variando muito.



500 gerações: Valores e pesos totais variando pouco. Já é possível ver um padrão de combinação entre as variações.



5000 gerações: Valores e pesos totais das mochilas iguais para toda a elite. Todas as combinações iguais, demonstrando um resultado preciso.