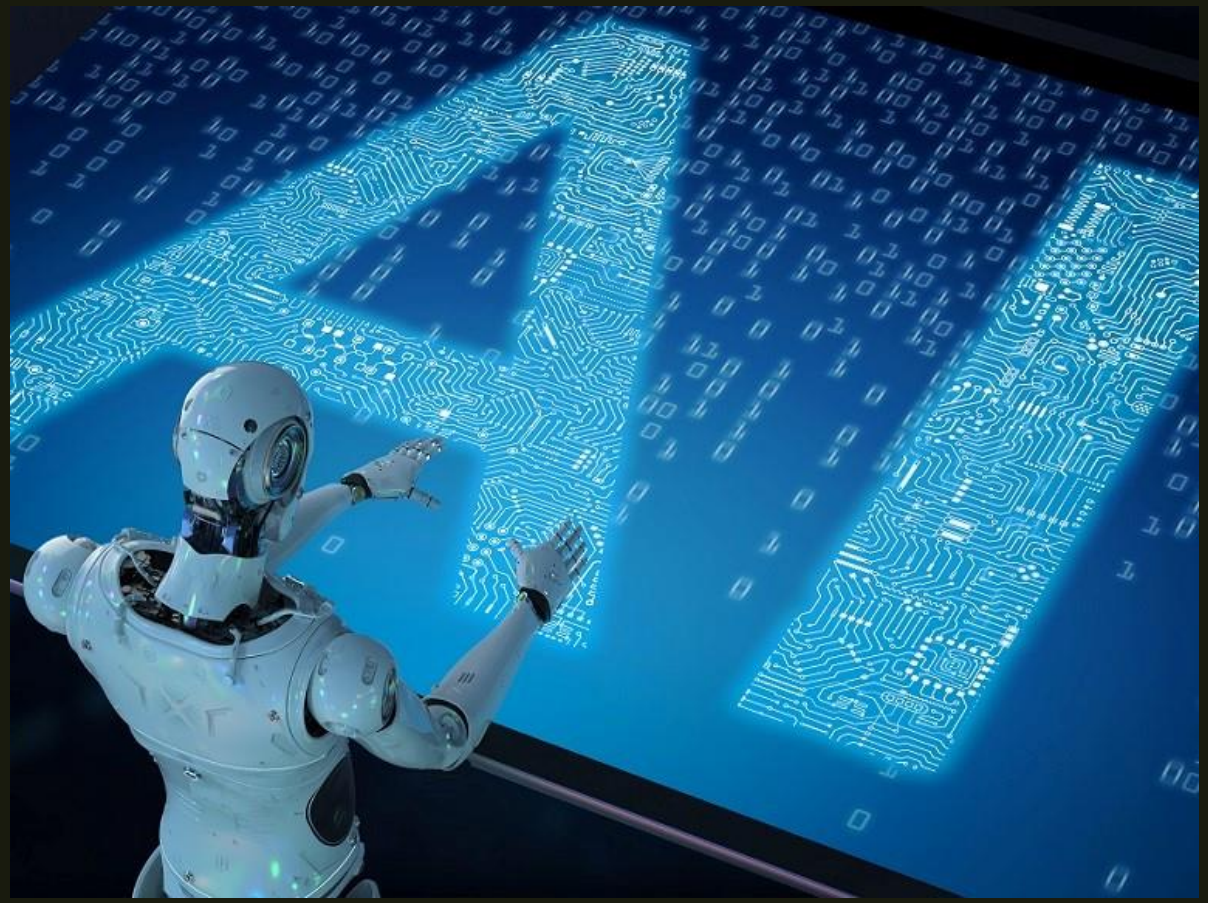
IA NA PRÁTICA:

SOLUCIONANDO O PROBLEMA DO CAIXEIRO VIAJANTE

Prof Me. Luis Gustavo Araujo



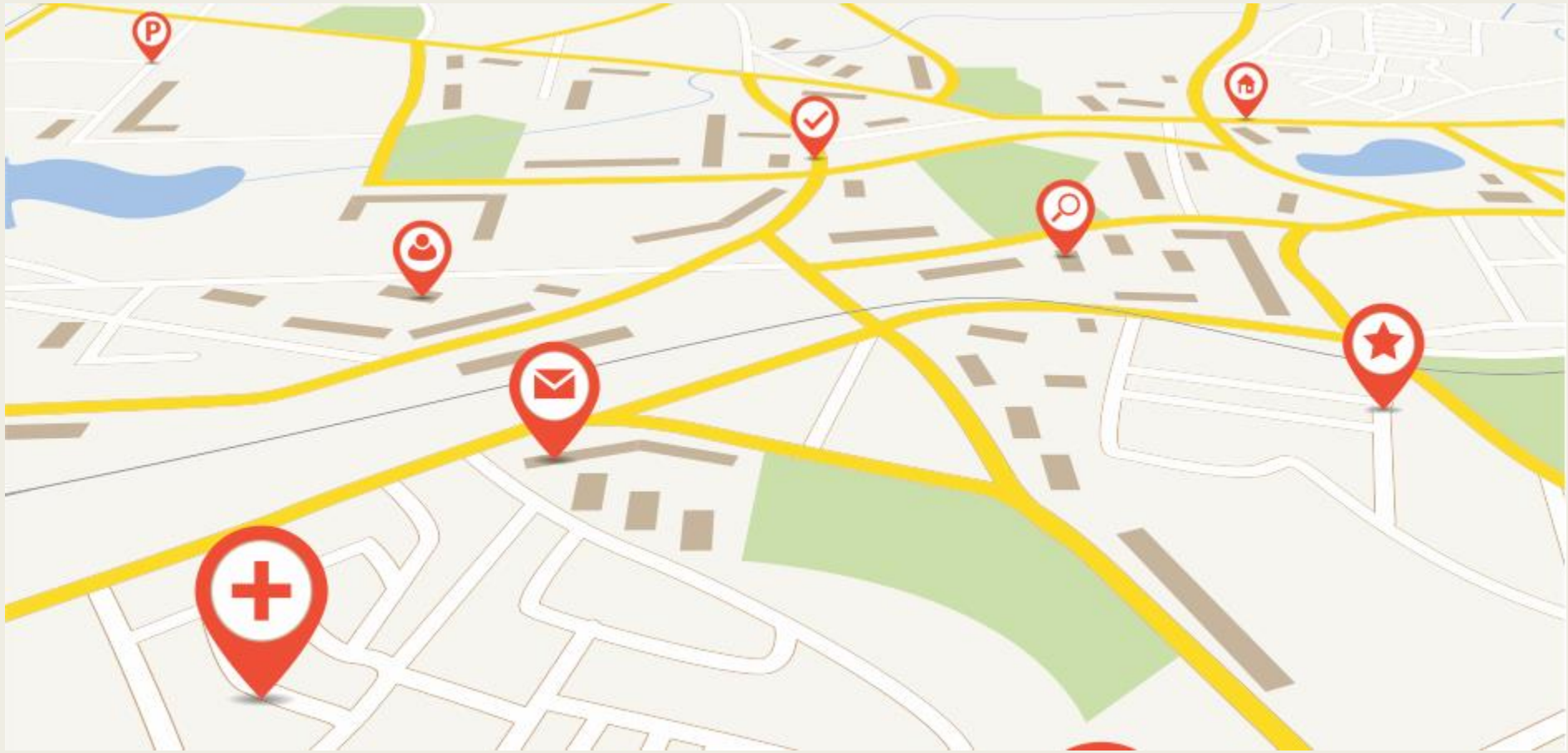
INTELIGÊNCIA ARTIFICIAL

O que é isso?

Inteligência Artificial (Computacional)

“Criada recentemente, a disciplina de Inteligência Artificial (IA) possui diversas frentes de estudo, ou seja, é formada por vários ramos, os quais, segundo Russell e Norvig (2004, p 4), **estão relacionados aos processos de pensamento e raciocínio, à forma de comportamento, à reprodução fiel do desempenho humano e à imitação da racionalidade” (Rosa e Luz, 2009)**

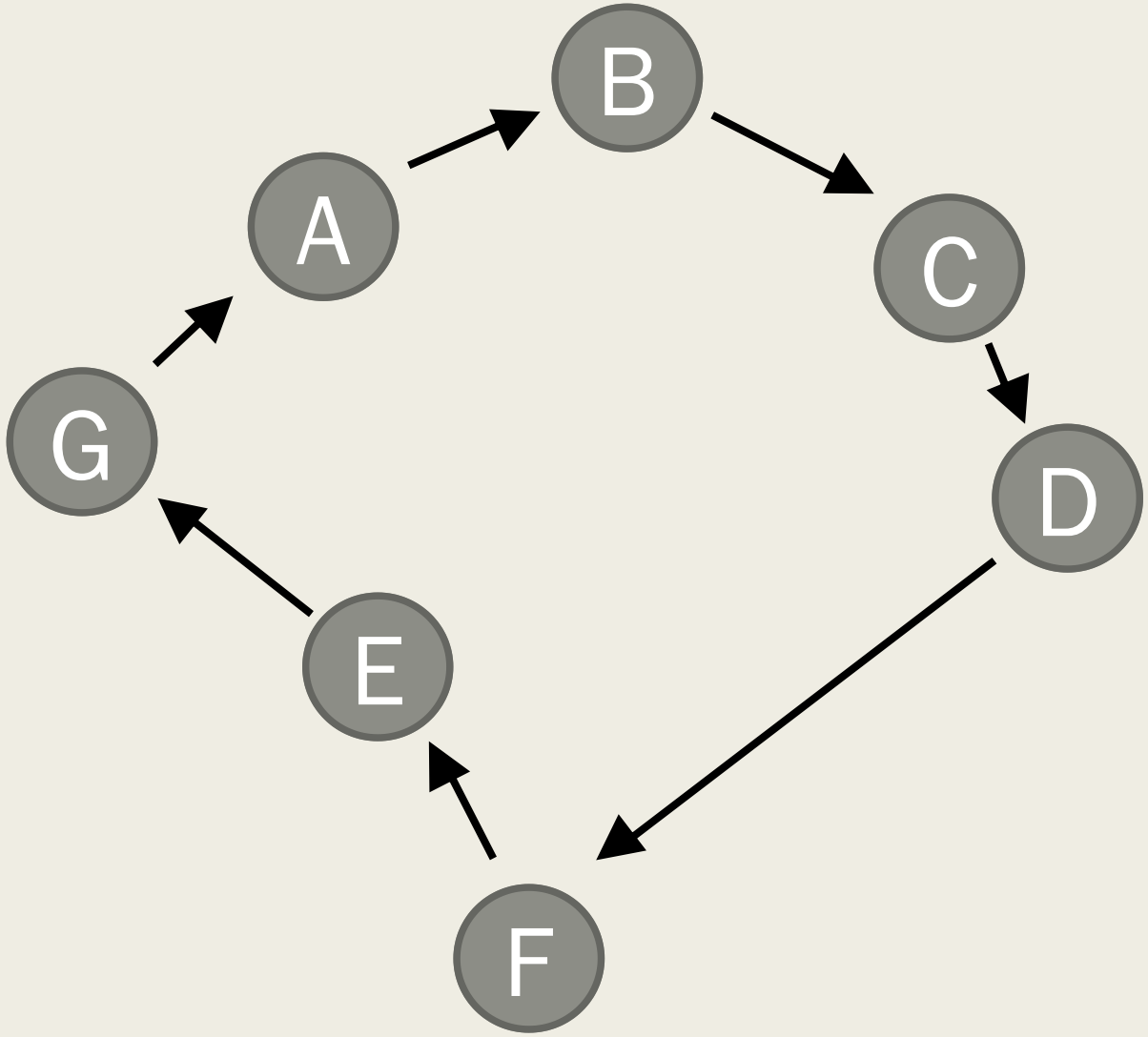
Aplicação de IA em Rotas



Caixeiro Viajante (Problema Clássico)

O Problema do Caixeiro Viajante (PCV) é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Ele é um problema de otimização NP-difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível. (Wikipédia, 2019)

Rota do Caxeiro (exemplo)



ALGORITMOS GENÉTICOS

Uma Abordagem de Otimização

Algoritmos Genéticos

"Algoritmo Genético é uma técnica de IA que, assim como definido por Goldberg (1989 apud TICONA, 2003, p. 46), foi criada com o intuito de imitar determinados processos observados na evolução natural das espécies"

Algoritmos Genéticos

- **Métodos de Otimização inspirados na Teoria da Evolução:**

  - Indivíduos mais adaptados sobrevivem e transmitem suas características para as gerações seguintes ;

  - Charles Darwin (Origem das Espécies, 1859).

Otimização

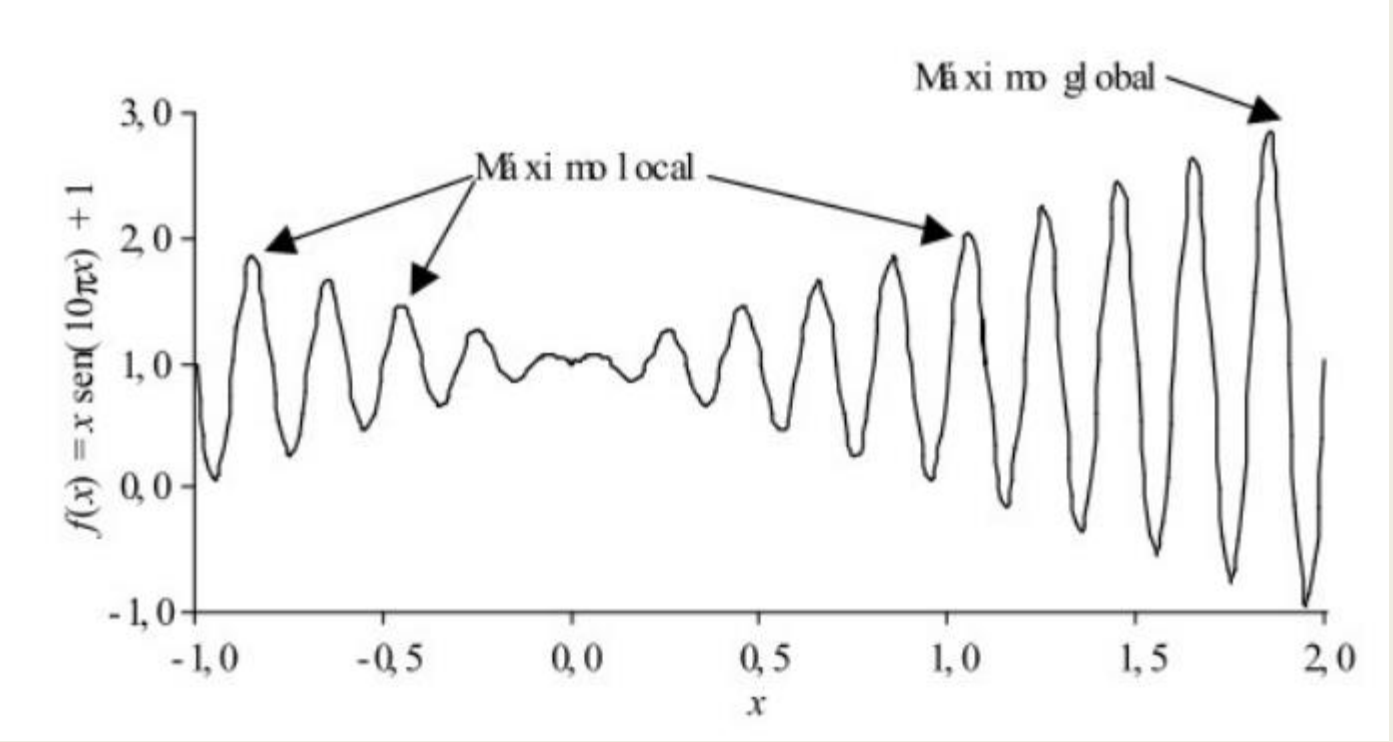
- **Espaço de Busca:**
  - Possíveis soluções de um problema.

- **Função Objetivo:**
  - Avalia cada solução com uma nota.

- **Tarefa:**
  - Encontrar a solução que corresponda ao ponto de máximo (ou mínimo) da função objetivo.

Otimização (Exemplo)

- Achar o ponto máximo da função: f(x) = xsen(10πx) + 1, -1 ≤ x ≤ 2



Otimização (Dificuldades)

- Alguns problemas podem ter espaços de busca muito grandes;
- Muitos algoritmos não são capazes de localizar ótimo global na presença de múltiplos ótimos locais.

Passos do AG

- Geração de um conjunto inicial de soluções que são iterativamente melhoradas;
- População de indivíduos (cromossomos) ;
- Busca de soluções seguem um processo evolutivo;
- Seleção dos mais aptos + Transmissão de características.

Passos do AG

Passo 1: Geração de uma população inicial com indivíduos escolhidos aleatoriamente

Passo 2: Avaliação dos indivíduos

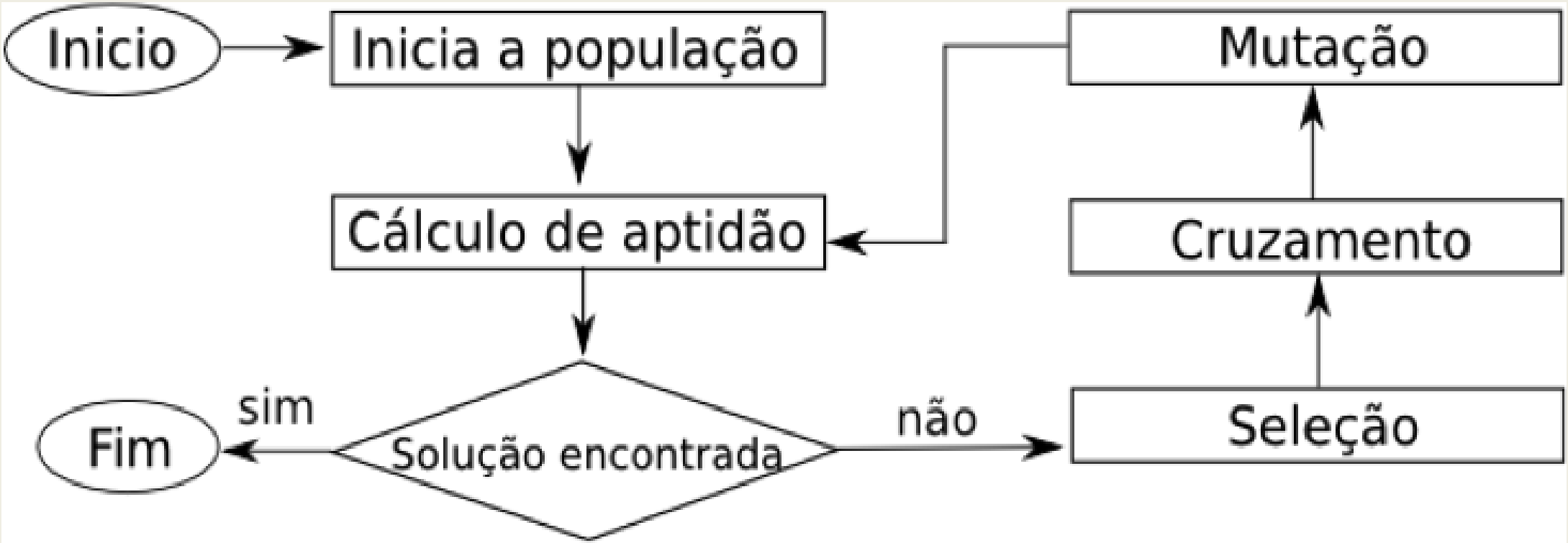
- Cálculo da função de ***fitness*** (usando a função objetivo)

Passo 3: Seleção de indívíduos mais aptos

Passo 4: Geração de uma nova população a partir dos indivíduos selecionados e ir para Passo 2

- Operadores de busca (***crossover*** e ***mutação***)

Passos do AG


Inicio
Inicia a população
Cálculo de aptidão
Solução encontrada
sim
Fim
não
Seleção
Cruzamento
Mutação

Algoritmo do AG

Seja $S(t)$ a população de cromossomos na geração t.

$t \leftarrow 0$
inicializar $S(t)$
avaliar $S(t)$
enquanto o critério de parada não for satisfeito **faça**
    $t \leftarrow t+1$
    selecionar $S(t)$ a partir de $S(t-1)$
    aplicar *crossover* sobre $S(t)$
    aplicar mutação sobre $S(t)$
    avaliar $S(t)$
fim enquanto

Atenção!!!

Pontos importantes a definir:

- Representação dos indivíduos;
- Estratégia de seleção;
- Operadores de busca.

Fitness

Fitness é a adequação da solução proposta. *No caso do Caixeiro é a distância percorrida.*

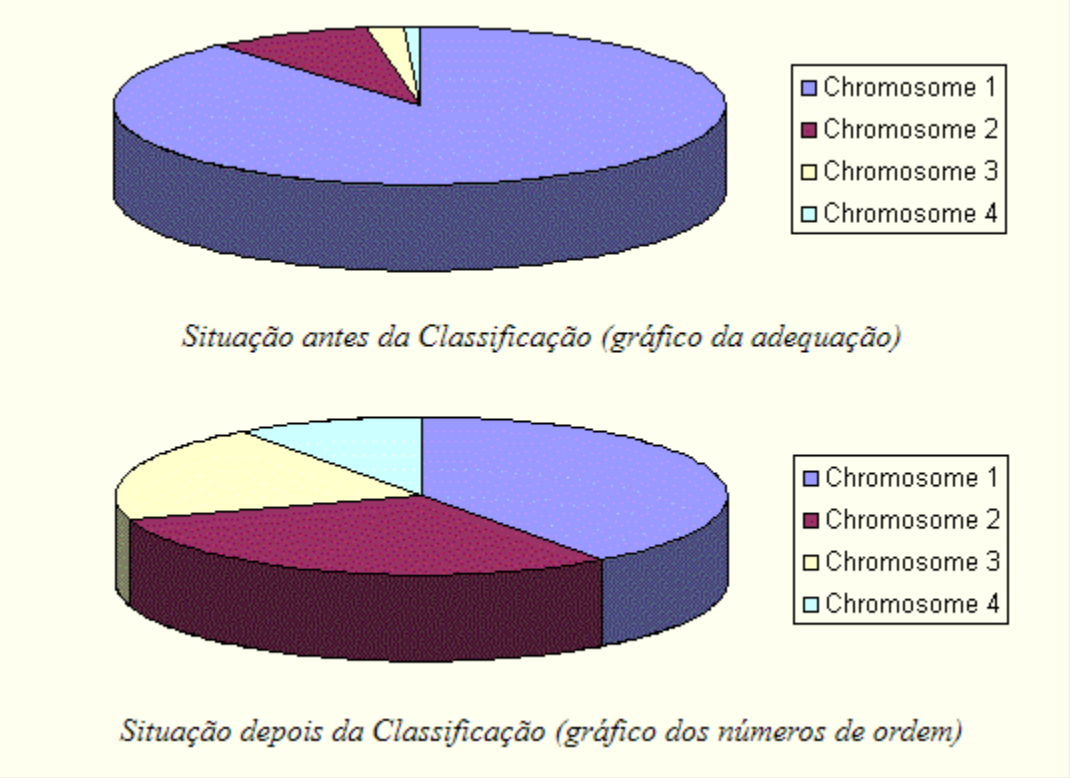
Seleção

Gerada uma população e calculado o fitness de cada indivíduo, devemos selecionar quem irá sobreviver à geração.

Seleção

Métodos:

Roleta;
Classificação;
Estado Estacionários;
Elitismo.



Situação antes da Classificação (gráfico da adequação)

Situação depois da Classificação (gráfico dos números de ordem)

Seleção (Elitismo)

A idéia básica do elitismo já foi introduzida. Quando criamos uma nova população por cruzamento e mutação, nós temos uma grande chance de perder os melhores cromossomos.

Seleção (Elitismo)

Elitismo é o nome do método que primeiro copia os melhores cromossomos (ou os poucos melhores cromossomos) para a nova população.

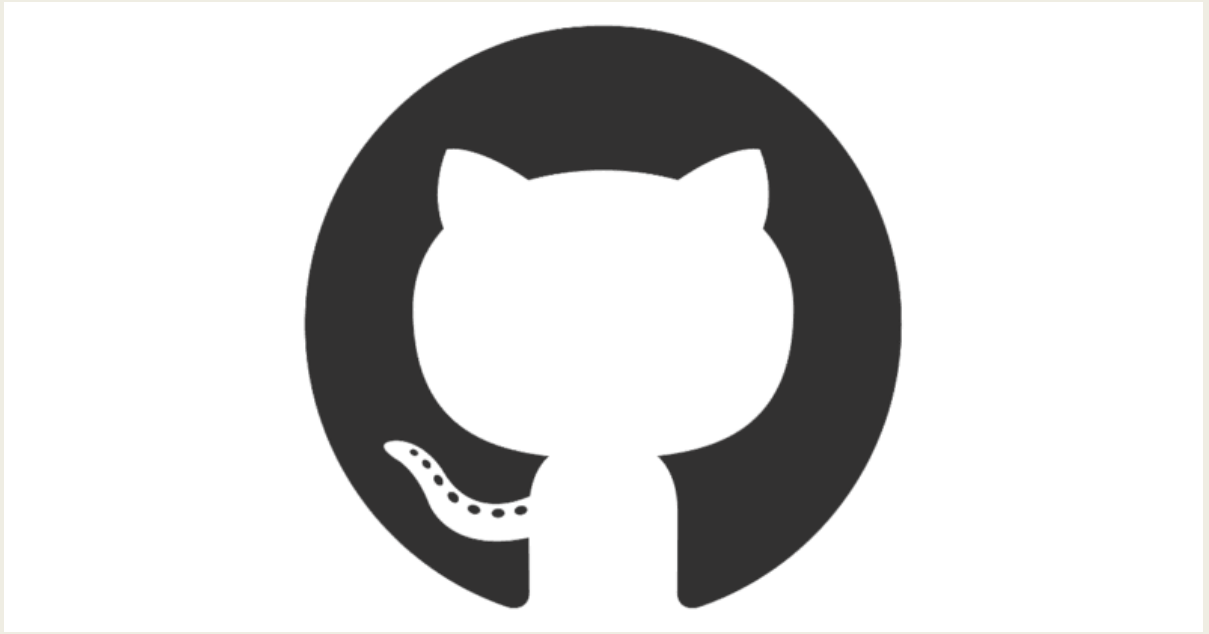
Mutação

Realizamos Mutação nos novos indivíduos, ou seja, os modificamos. O objetivo é ter variabilidade na nova população.

VAMOS PRATICAR

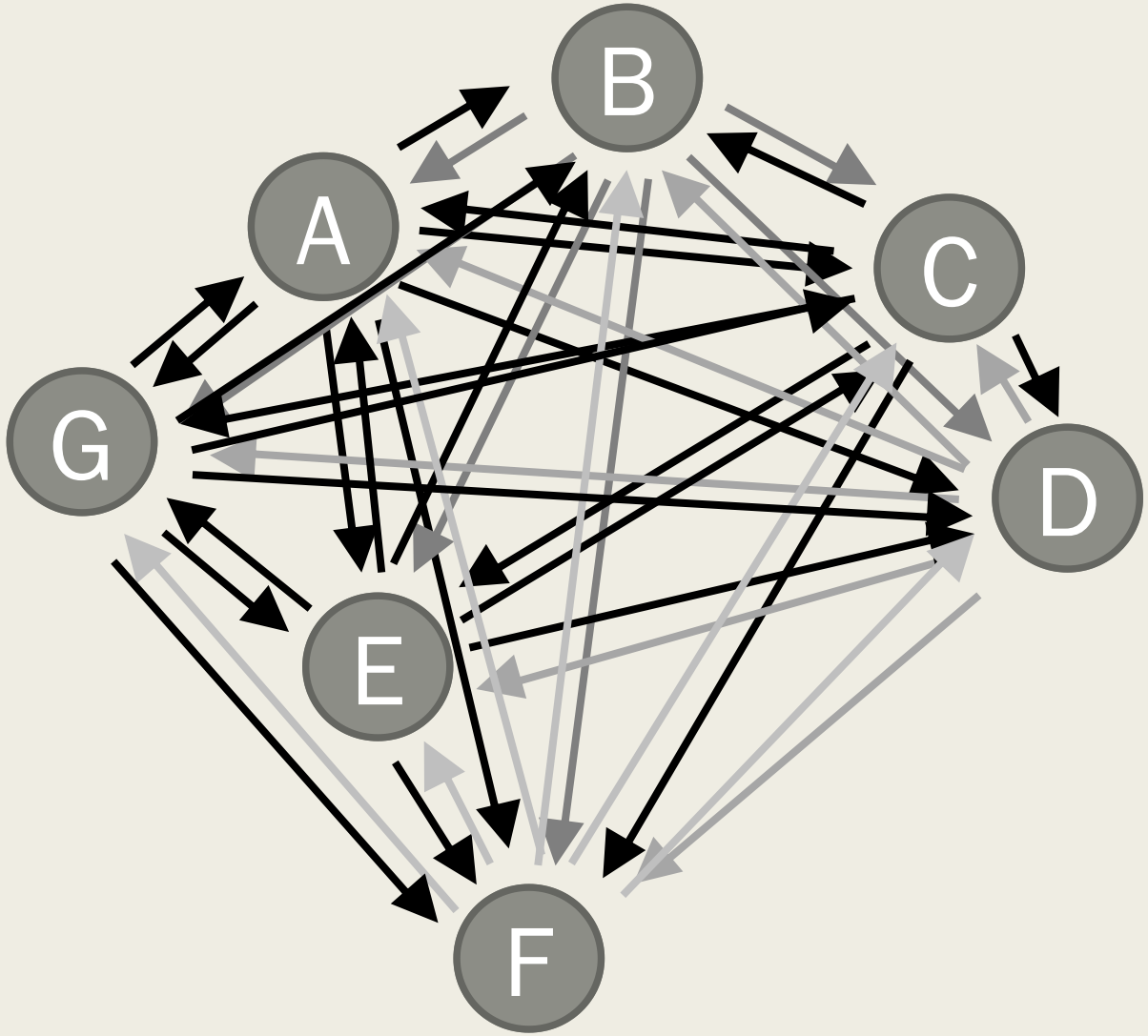
Algoritmos Genético para Solucionar o PCV

Projeto Base

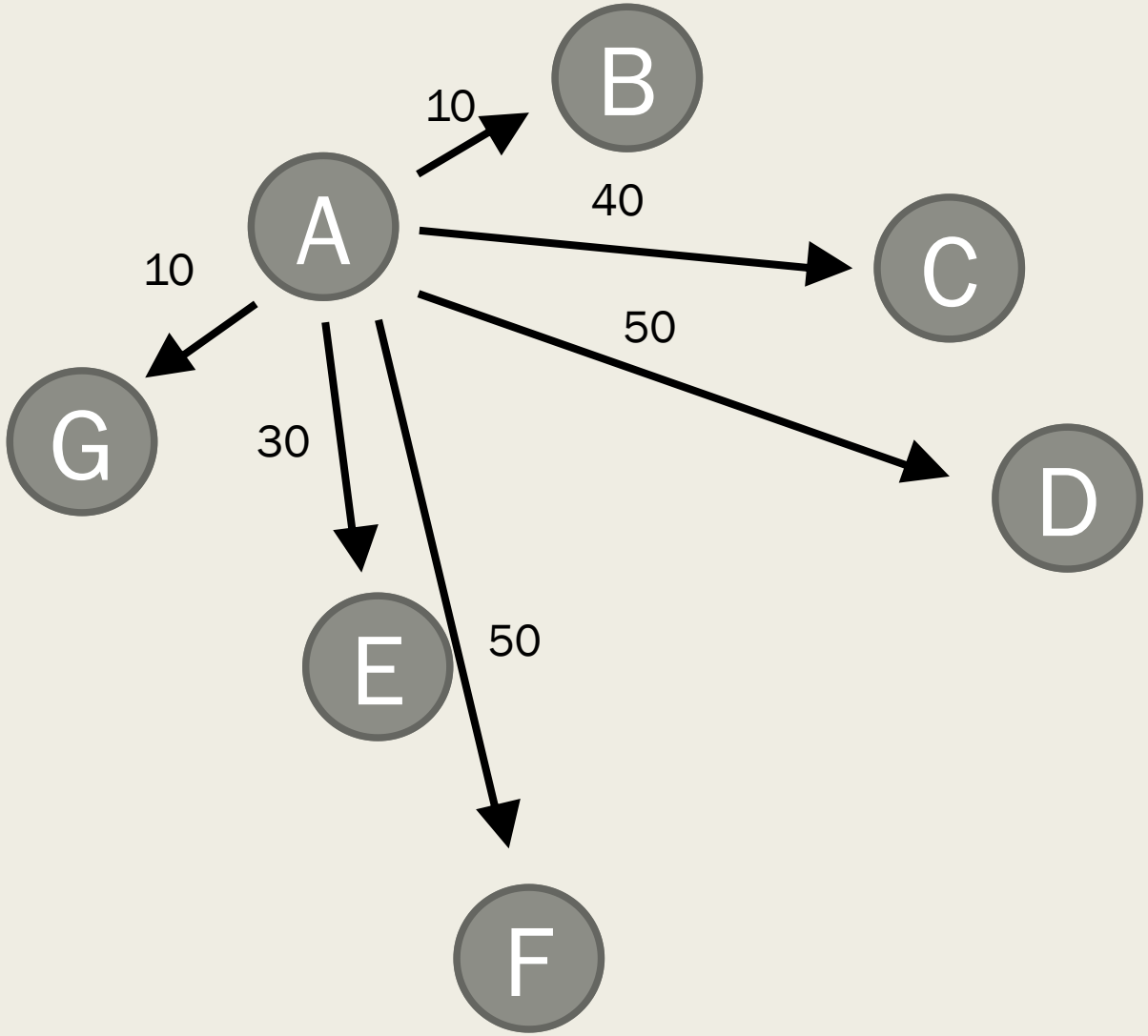


https://github.com/LuisAraujo/OficinaAlgortimosGenetico

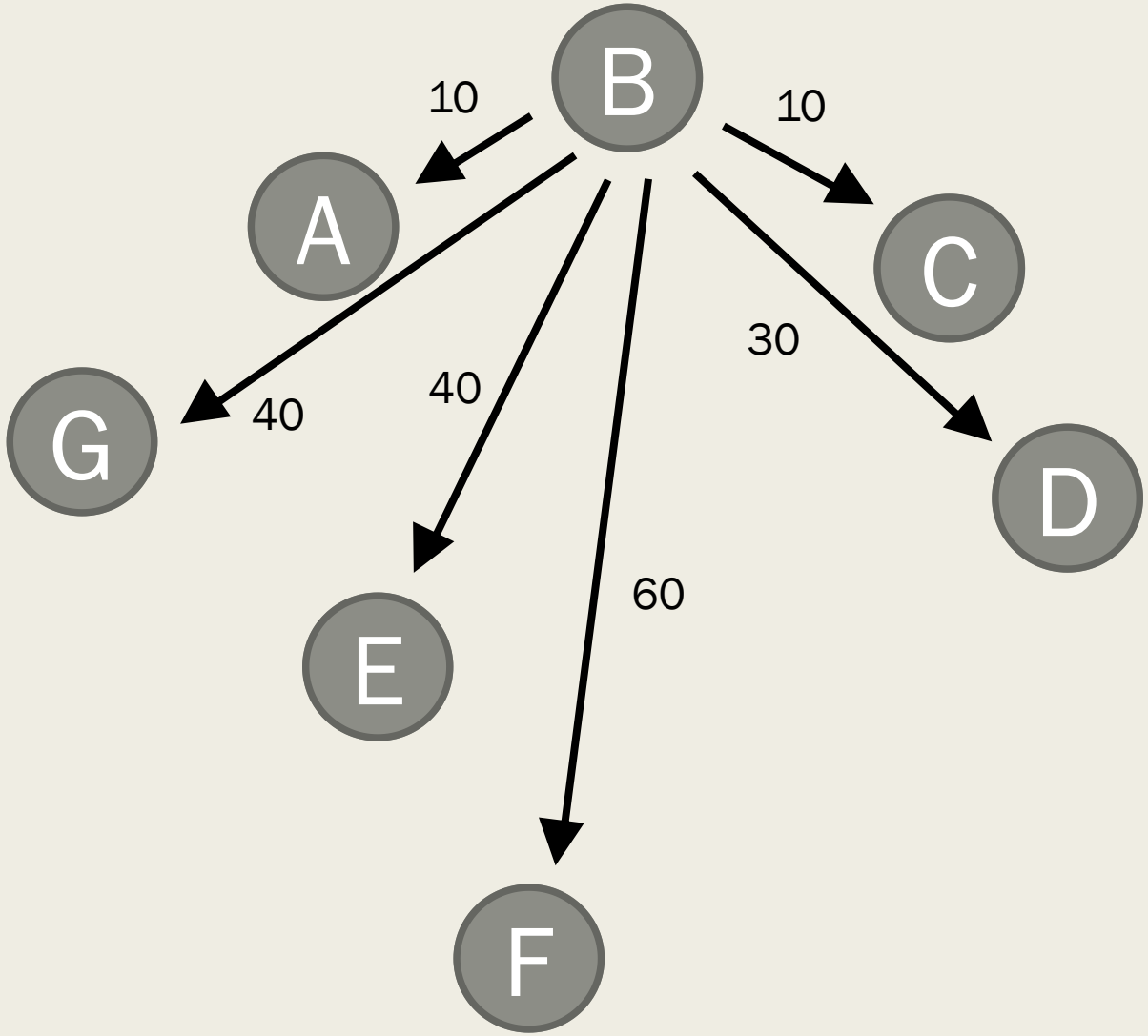
Nosso Grafo



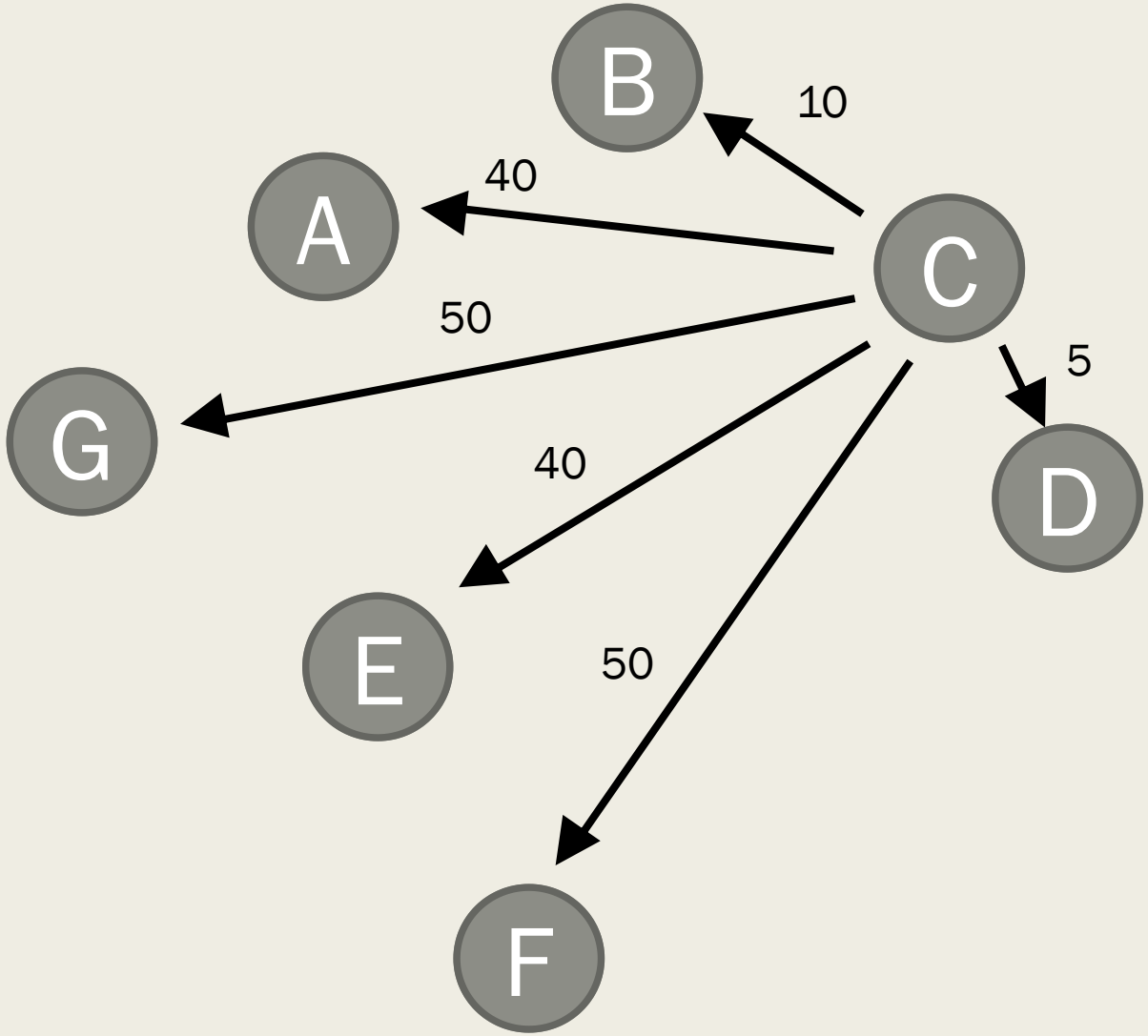
Nosso Grafo (Pesos saindo de A)



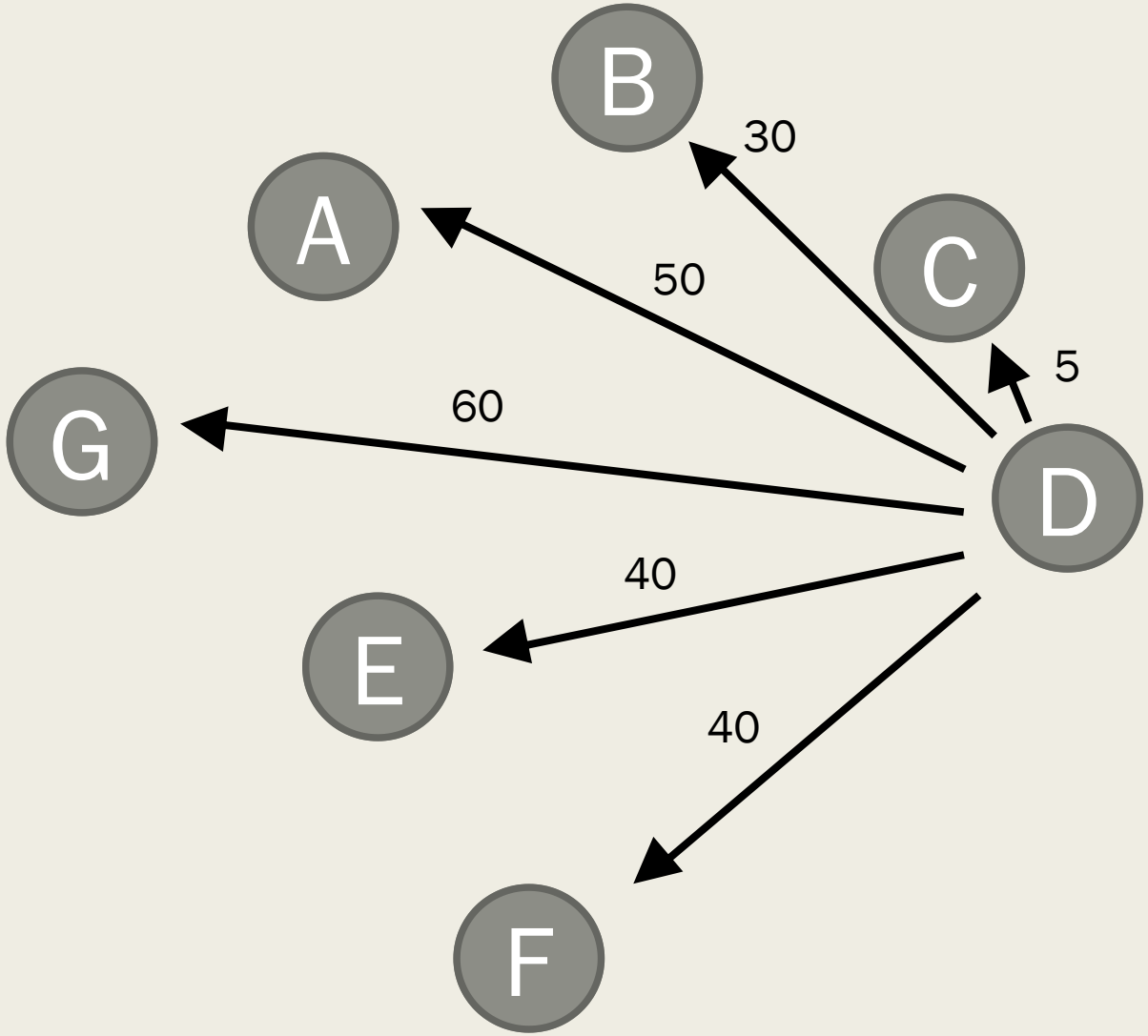
Nosso Grafo (Pesos saindo de B)



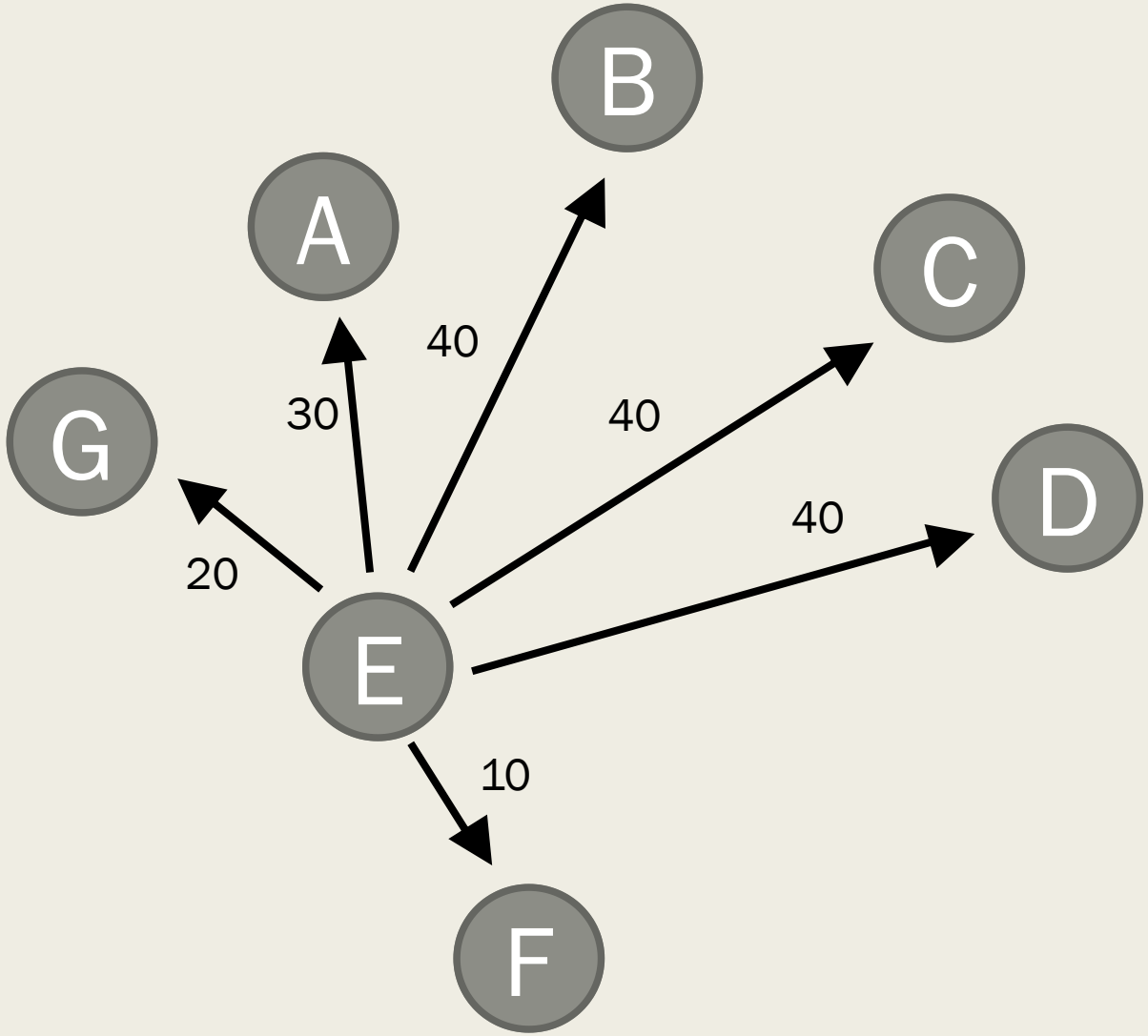
Nosso Grafo (Pesos saindo de C)



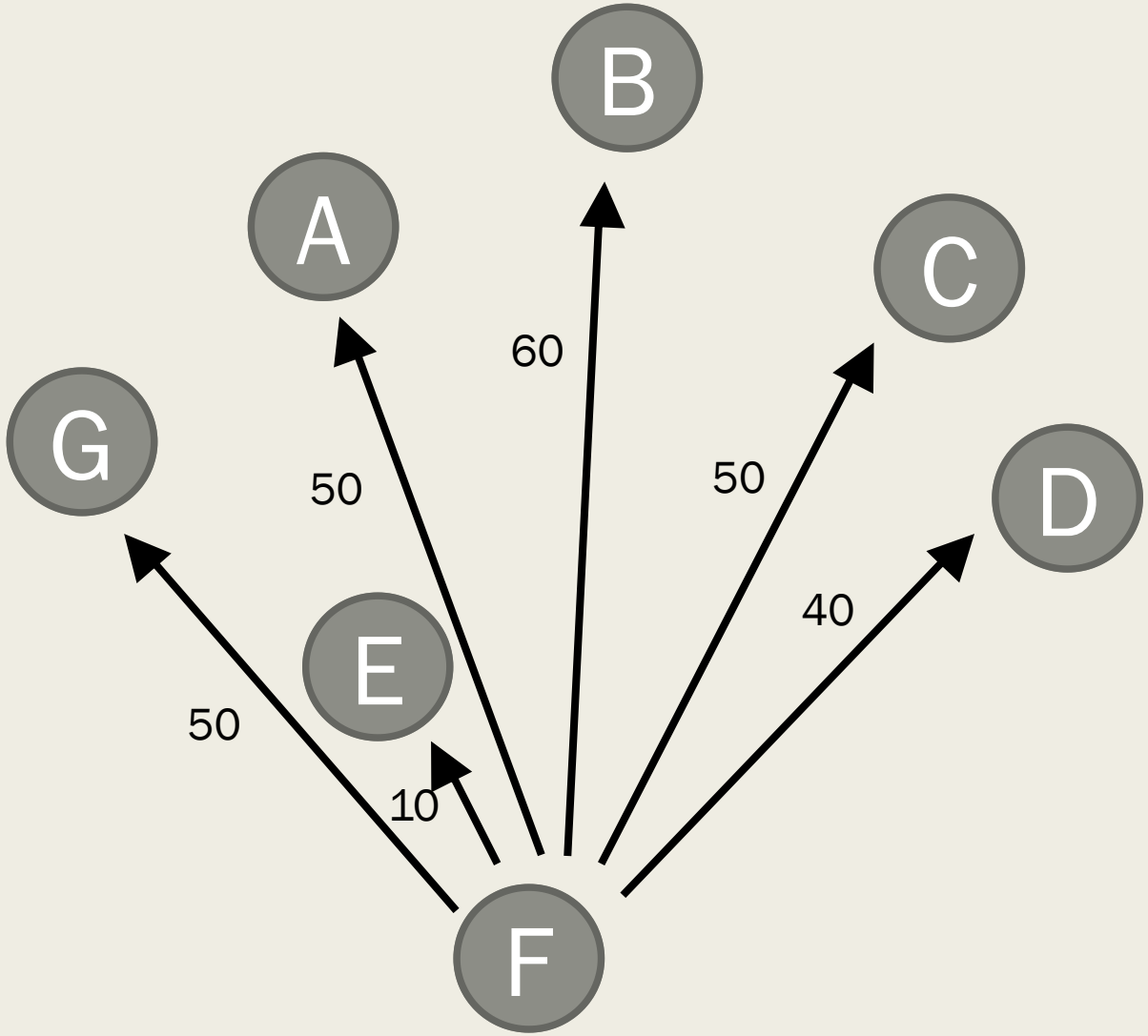
Nosso Grafo (Pesos saindo de D)



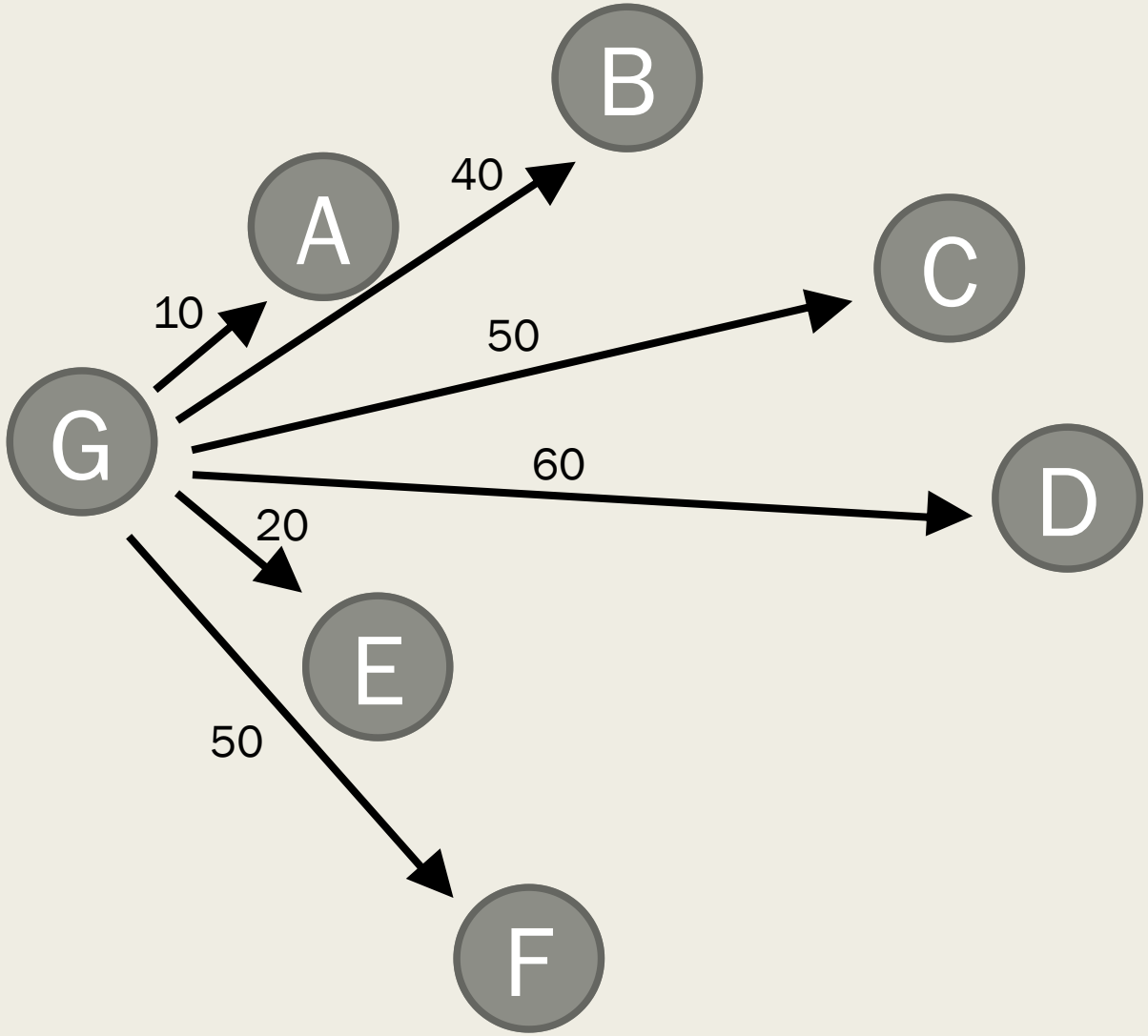
Nosso Grafo (Pesos saindo de E)



Nosso Grafo (Pesos saindo de F)



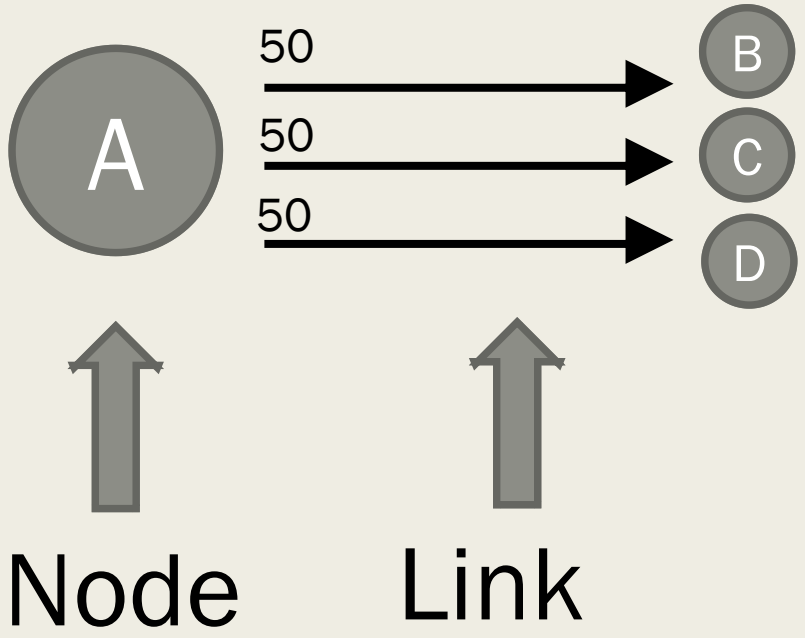
Nosso Grafo (Pesos saindo de G)



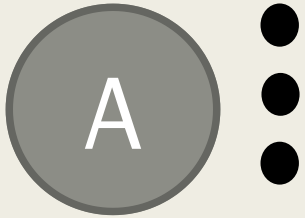
Nosso Grafo (Em Matriz)

	A	B	C	D	E	F	G
A	0	10	40	50	30	50	10
B	10	0	10	30	40	60	40
C	40	10	0	5	40	50	50
D	50	30	5	0	40	40	60
E	30	40	40	40	0	10	20
F	50	60	50	40	10	0	50
G	10	40	50	60	20	50	0

Nosso Grafo (Representação)



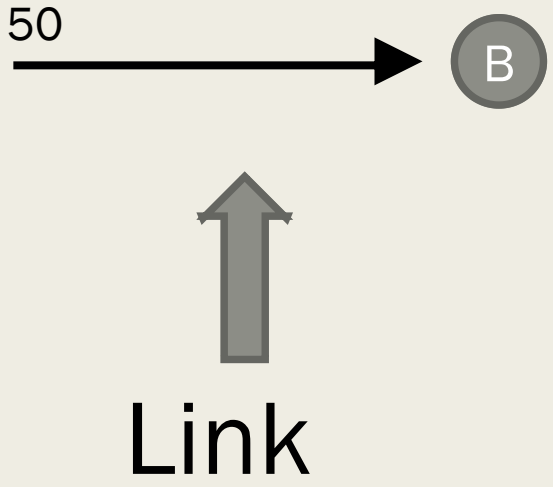
Nosso Grafo (Node)



Node

```
public class Node {
    String label;
    ArrayList<Link> links;
}
```

Nosso Grafo (Representação)




```
public class Link {

    int weight;
    Node node;
}
```

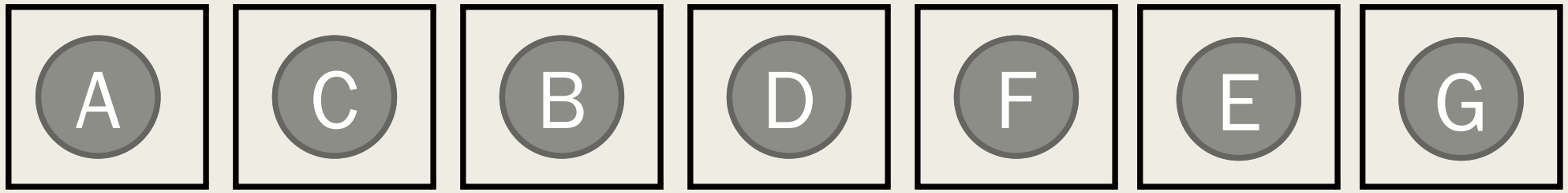
Nosso Grafo (Em Código)

```
public static Graph startGraph() {

    Node n1 = new Node("A");
    Node n2 = new Node("B");
    Node n3 = new Node("C");
    Node n4 = new Node("D");
    Node n5 = new Node("E");
    Node n6 = new Node("F");
    Node n7 = new Node("G");

    //A
    n1.addLink(new Link(n2, 10));
    n1.addLink(new Link(n3, 40));
    n1.addLink(new Link(n4, 50));
    n1.addLink(new Link(n5, 30));
    n1.addLink(new Link(n6, 50));
    n1.addLink(new Link(n7, 10));
```

```
//B
n2.addLink(new Link(n1, 10));
n2.addLink(new Link(n3, 10));
n2.addLink(new Link(n4, 30));
n2.addLink(new Link(n5, 40));
n2.addLink(new Link(n6, 60));
n2.addLink(new Link(n7, 40));

//C
n3.addLink(new Link(n1, 40));
n3.addLink(new Link(n2, 10));
n3.addLink(new Link(n4, 5));
n3.addLink(new Link(n5, 40));
n3.addLink(new Link(n6, 50));
n3.addLink(new Link(n7, 50));
```

Gene e Cromossomo

- Gene serão os Nodes;
- Cromossomo é um conjunto de Nodes em uma dada ordem.

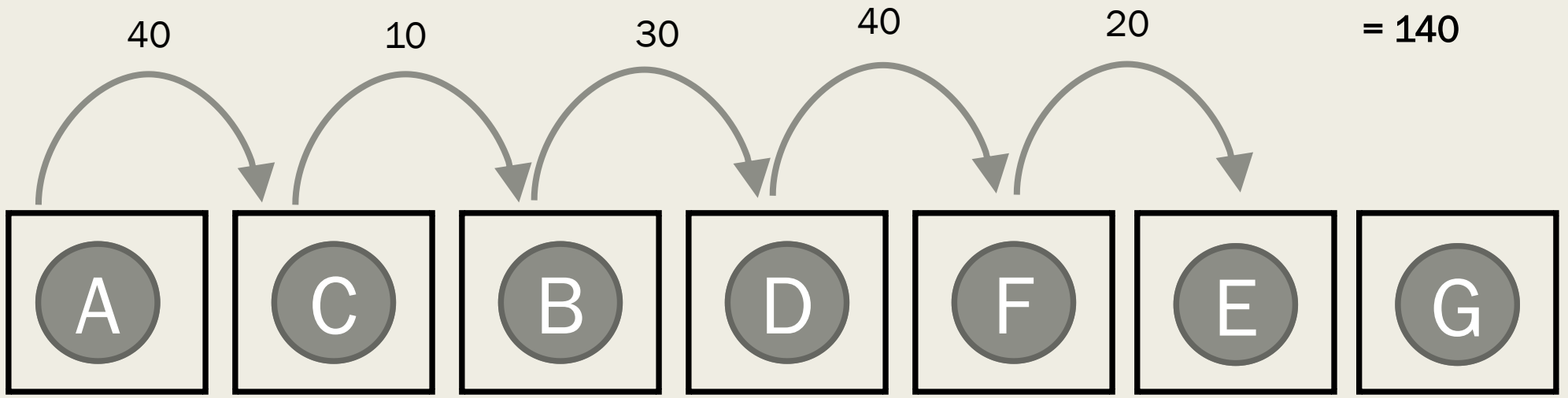


Gene e Cromossomo

```
public class Cromossomo {
    ArrayList<Node> nodes;
    int fitness;
}
```

Cálculo do Fitness

- Basta somarmos as distâncias entre os vértices.



Cálculo do Fitness

- Basta somarmos as distâncias entre os vértices.

```
public void calcFitness() {
    for(int i = 0; i < this.nodes.size() - 1; i++)
        fitness += nodes.get(i).getLinkTo(nodes.get(i+1));
}
```

População

- Conjunto de Cromossomos.

```
public class Population {

    ArrayList<Cromossomo> individuals;

    public Population(ArrayList<Cromossomo> individuals) {
        this.individuals = individuals;
    }
```

População

- Calcula o fitness de todos e ordena-os.

```
@SuppressWarnings("unchecked")
public void calcFitness() {
    for(int i = 0;  i< this.population.size(); i++)
        this.population.get(i).calcFitness();

    Collections.sort(this.population, new Comparator() {
        @Override
        public int compare(Object o1, Object o2) {
            Cromossomo c1 = (Cromossomo) o1;
            Cromossomo c2 = (Cromossomo) o2;
            if(c1.fitness > c2.fitness)
                return 1;
            else if(c1.fitness < c2.fitness)
                return -1;
            else
                return  0;
        }

    });
}
```

Gerando População Inicial

```
public static Population geratePopulation(Graph graph,
        int sizePopulation){
    ArrayList<Cromossomo> cromossomos = new ArrayList<Cromossomo>();

    for(int i = 0; i < sizePopulation; i++) {
        ArrayList<Node> a = new ArrayList<Node>();
        for(int j=0; j < graph.nodes.size(); j++)
            a.add(graph.nodes.get(j));

        Cromossomo c = new Cromossomo();

        for(int j=0; j<graph.nodes.size(); j++) {
            int n = (int) ( Math.random()* a.size());
            c.nodes.add( a.remove(n) );
        }
        cromossomos.add(c);
    }
    Population pop = new Population(cromossomos);
    return pop;
}
```

Gerando População Inicial

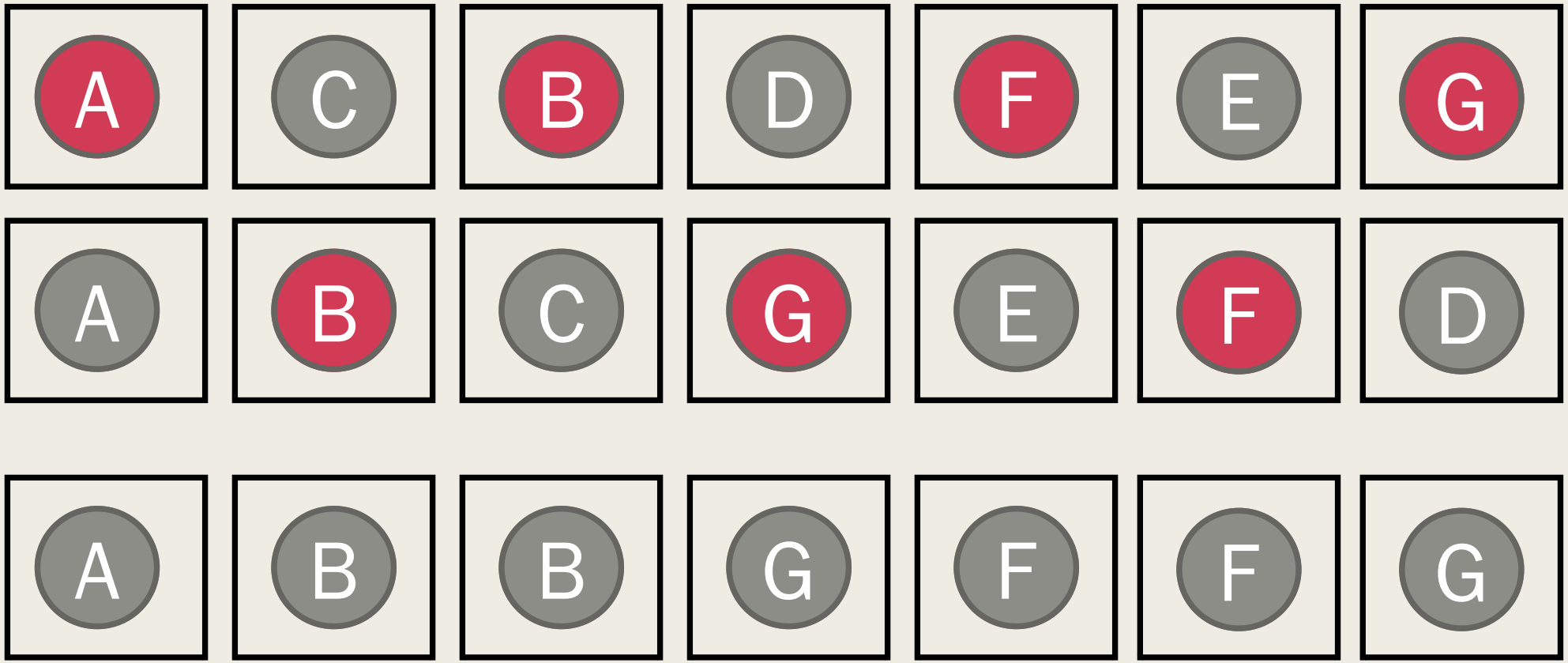
```
public static void main(String[] args) {

    Graph graph = startGraph();
    Population p = geratePopulation(graph, 1000);
    p.calcFitness();
```

Gerando População Inicial

```
Show Population
[E] [G] [B] [D] [C] [A] [F]   - fitness 185
[G] [C] [B] [E] [F] [D] [A]   - fitness 200
[C] [G] [D] [F] [E] [A] [B]   - fitness 200
[D] [B] [A] [F] [G] [E] [C]   - fitness 200
[C] [D] [B] [G] [F] [A] [E]   - fitness 205
[E] [A] [F] [B] [C] [D] [G]   - fitness 215
[D] [C] [G] [E] [A] [F] [B]   - fitness 215
[G] [D] [B] [F] [E] [A] [C]   - fitness 230
[G] [B] [D] [A] [E] [C] [F]   - fitness 240
[A] [F] [G] [D] [C] [E] [B]   - fitness 245
```

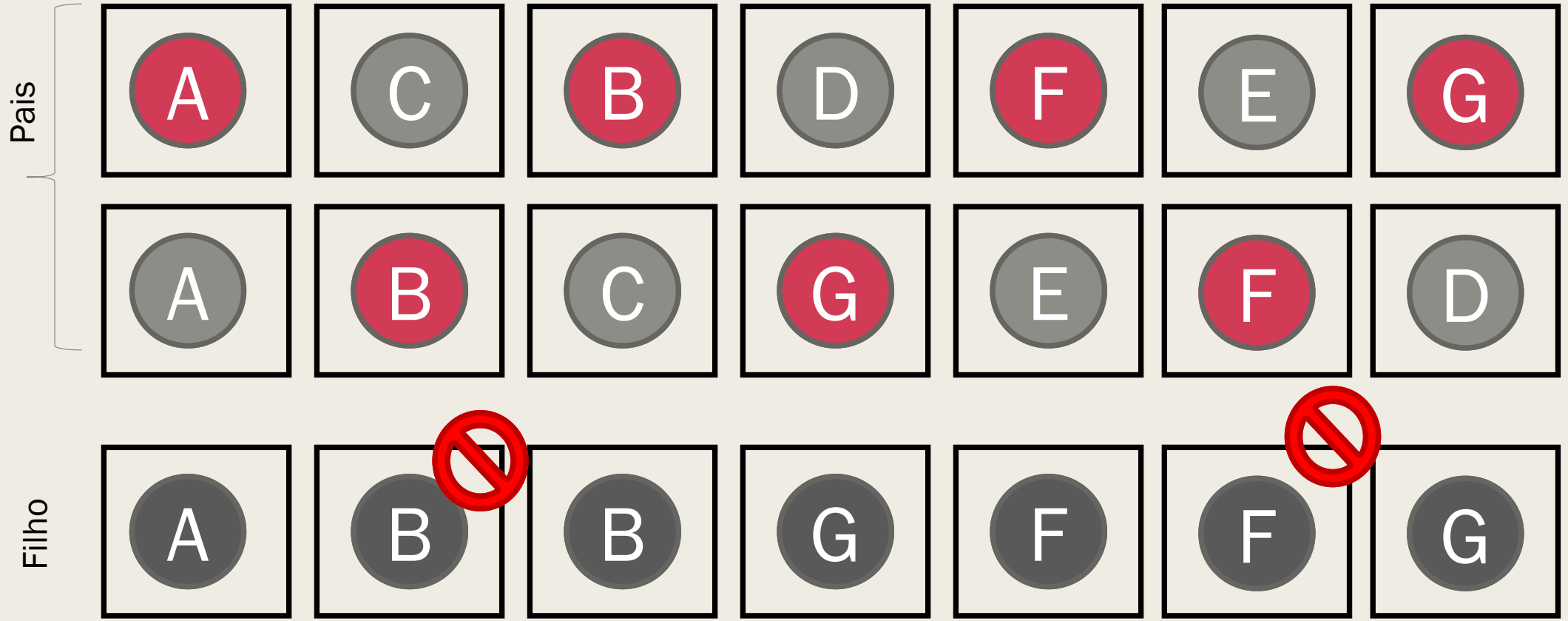
Crossover

- Crossover podem gerar anomalias.



Crossover

- Crossover podem gerar anomalias.



Crossover

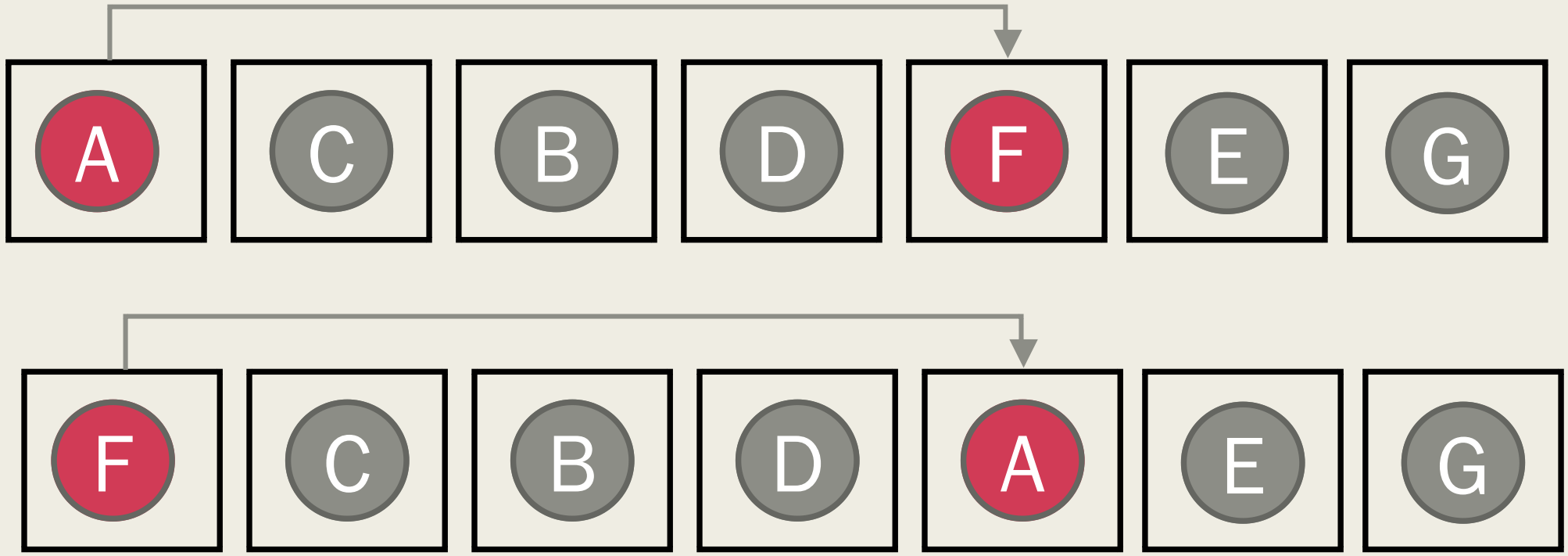
- No Crossover é mais fácil punir do que remediar!!!!
  - Soluções com anomalias recebem fitness baixo ou alto.

Crossover

```
public static Cromossomo crossover(Cromossomo c1, Cromossomo c2) {
    Cromossomo newc = new Cromossomo();

    for(int i = 0; i < c1.nodes.size(); i++)
        if(i%2 == 0)
            newc.nodes.add(i, c1.nodes.get(i));
        else
            newc.nodes.add(i, c2.nodes.get(i));


    return newc;
}
```

Mutação

- Um método de mutação pode ser trocar a ordem dos genes escolhidos aleatoriamente.



Mutação

```
public static Cromossomo mutation(Cromossomo c) {

    int i = (int) Math.random() * c.nodes.size();
    int j = (int) Math.random() * c.nodes.size();
    Node n = c.nodes.get(i);
    c.nodes.set(i, c.nodes.get(j));
    c.nodes.set(j, n);

    return c;
}
```

Gerando a Segunda População

```
public static void main(String[] args) {

    Graph graph = startGraph();
    Population p = geratePopulation(graph, 1000);
    p.calcFitness();
    p = selection(p);
    p = mutation(p);
```

Segunda Geração (antes da avaliação)

```
Show Population Geração 2
[G] [A] [B] [D] [E] [F] [C]   - fitness 150
[C] [B] [E] [G] [A] [D] [F]   - fitness 170
[C] [F] [E] [A] [A] [B] [D]   - fitness 0
[G] [D] [B] [E] [C] [A] [E]   - fitness 0
[E] [C] [G] [G] [C] [E] [F]   - fitness 0
[G] [E] [A] [F] [F] [G] [E]   - fitness 0
[E] [D] [G] [A] [C] [B] [F]   - fitness 220
[F] [C] [D] [G] [B] [E] [A]   - fitness 225
[G] [C] [A] [B] [F] [D] [E]   - fitness 240
[A] [E] [D] [F] [C] [G] [B]   - fitness 250
```

Avaliando a Segunda Geração

```
public static void main(String[] args) {

    Graph graph = startGraph();
    Population p = geratePopulation(graph, 1000);
    p.calcFitness();
    p = selection(p);
    p = mutation(p);
    p.calcFitness();
```

Cálculo do Fitness adicionando a Punição.

```
public void calcFitness() {
    int[] count = new int[7];

    for(int i = 0; i < this.nodes.size() - 1; i++) {
        fitness += nodes.get(i).getLinkTo(nodes.get(i+1));
        if(nodes.get(i).label.equals("A")) {
                count[0]++;
        }else if(nodes.get(i).label.equals("B")) {
            count[1]++;
        }
        //.../
    }

    //punindo
    for(int i = 0; i < count.length; i++) {
        if(count[i]>1)
            fitness = Integer.MAX_VALUE;
    }
}
```

Segunda Geração (após da avaliação)

```
Show Population Geração 2
[C] [D] [B] [A] [E] [G] [F]  - fitness 290
[E] [F] [A] [G] [C] [D] [B]  - fitness 310
[D] [G] [C] [E] [F] [A] [B]  - fitness 440
[E] [A] [D] [F] [G] [B] [C]  - fitness 440
[F] [B] [G] [D] [C] [E] [A]  - fitness 470
[E] [C] [F] [A] [B] [D] [G]  - fitness 480
[D] [G] [C] [A] [G] [C] [E]  - fitness 2147483647
[D] [A] [A] [D] [G] [F] [B]  - fitness 2147483647
[D] [A] [C] [F] [F] [B] [B]  - fitness 2147483647
[F] [C] [G] [A] [C] [D] [A]  - fitness 2147483647
```

Criando n Gerações

```
int maxgeneration = 1000;
int currentgeneration = 0;
Cromossomo bestfitness = null;


Graph graph = startGraph();
Population p = geratePopulation(graph, 1000);
p.calcFitness();

while ( currentgeneration  < maxgeneration ){
    p = selection(p);
    p = mutation(p);
    p.calcFitness();
    currentgeneration++;
    }
```

Criando Gerações (Resultados)

população = 100, gerações = 1.000.000

```
Show Population
[F] [E] [G] [A] [C] [B] [D]   - fitness 120
[C] [B] [D] [A] [G] [E] [F]   - fitness 130
[G] [A] [E] [F] [D] [B] [C]   - fitness 130
```

```
[F] [E] [A] [B] [D] [C] [B]   - fitness 95
[F] [E] [G] [A] [C] [B] [D]   - fitness 120
[C] [B] [D] [A] [G] [E] [F]   - fitness 130
```

Criando Gerações (mais paradas)

```
int maxgeneration = 1000;
int currentgeneration = 0;
Cromossomo bestfitness = null;
int gerationofbestfitness = 0;
int maxgenerationofbestfitness = maxgeneration/2;
Graph graph = startGraph();
Population p = geratePopulation(graph, 1000);
p.calcFitness();

while ( (currentgeneration  < maxgeneration) &&
        (currentgeneration - gerationofbestfitness <
                maxgenerationofbestfitness ) ){
    p = selection(p);
    p = mutation(p);
    p.calcFitness();
    currentgeneration++;

    if(p.individuals.get(0) != bestfitness) {
        bestfitness = p.individuals.get(0);
        gerationofbestfitness = currentgeneration;
    } } }
```

Verificando Resultados

```
System.out.println("Última Geração: Geração"+ npop);
System.out.println("Melhor indivíduo "+ np.population.get(0));
System.out.println("Ultima População");
System.out.println(np);
```

Resultados (sem mutação)

```
Última Geração: Geração1001
Melhor indivíduo [G] [A] [B] [E] [F] [D] [C]  - fitness 115

Ultima População
[G] [A] [B] [E] [F] [D] [C]  - fitness 115
[A] [G] [E] [F] [B] [C] [D]  - fitness 115
[D] [C] [A] [B] [E] [G] [B]  - fitness 155
```

Resultados (com mutação de 50%)

```
Última Geração: Geração200001
Melhor indivíduo [B] [C] [D] [A] [G] [E] [F]  - fitness 105

Ultima População
[B] [C] [D] [A] [G] [E] [F]  - fitness 105
[E] [G] [A] [B] [C] [F] [D]  - fitness 140
```

Sobre os Resultados

- Podemos criar formas de corrigir os cromossomos com anomalias;
- Podemos aumentar o número de indivíduos na população;
- Podemos aumentar o número máximo de geração;
- Podemos ter uma mutação diferente.
- Podemos modificar o método de mutação.

Referências

- ROSA, Thatiane de Oliveira, LUZ, Hellen Souza. Conceitos Básicos de Algoritmos Genéticos: Teoria e Prática. In: XI Encontro de Estudantes de Informática do Tocantins, 2009, Palmas. Anais do XI Encontro de Estudantes de Informática do Tocantins. Palmas: Centro Universitário Luterano de Palmas, 2009. p. 27-37. Disponível em: http://tinyurl.com/ylouf6

- GONÇALVES, Alexandre. Algoritmos Geéticos (Slide). UFSC Disponível em: https://www.inf.ufsc.br/~alexandre.goncalves.silva/courses/14s2/ine5633/slides/aulaAG.pdf

- OBITKO, Marek. Introdução à Algoritmos Genéticos (Site). Traduzido por Hermelindo Pinheiro Manoel Disponível em: https://www.obitko.com/tutorials/genetic-algorithms/portuguese/selection.php

IA NA PRÁTICA:

SOLUCIONANDO O PROBLEMA DO CAIXEIRO VIAJANTE

Prof Me. Luis Gustavo Araujo
luis.araujo@unifacs.br