UNISUL

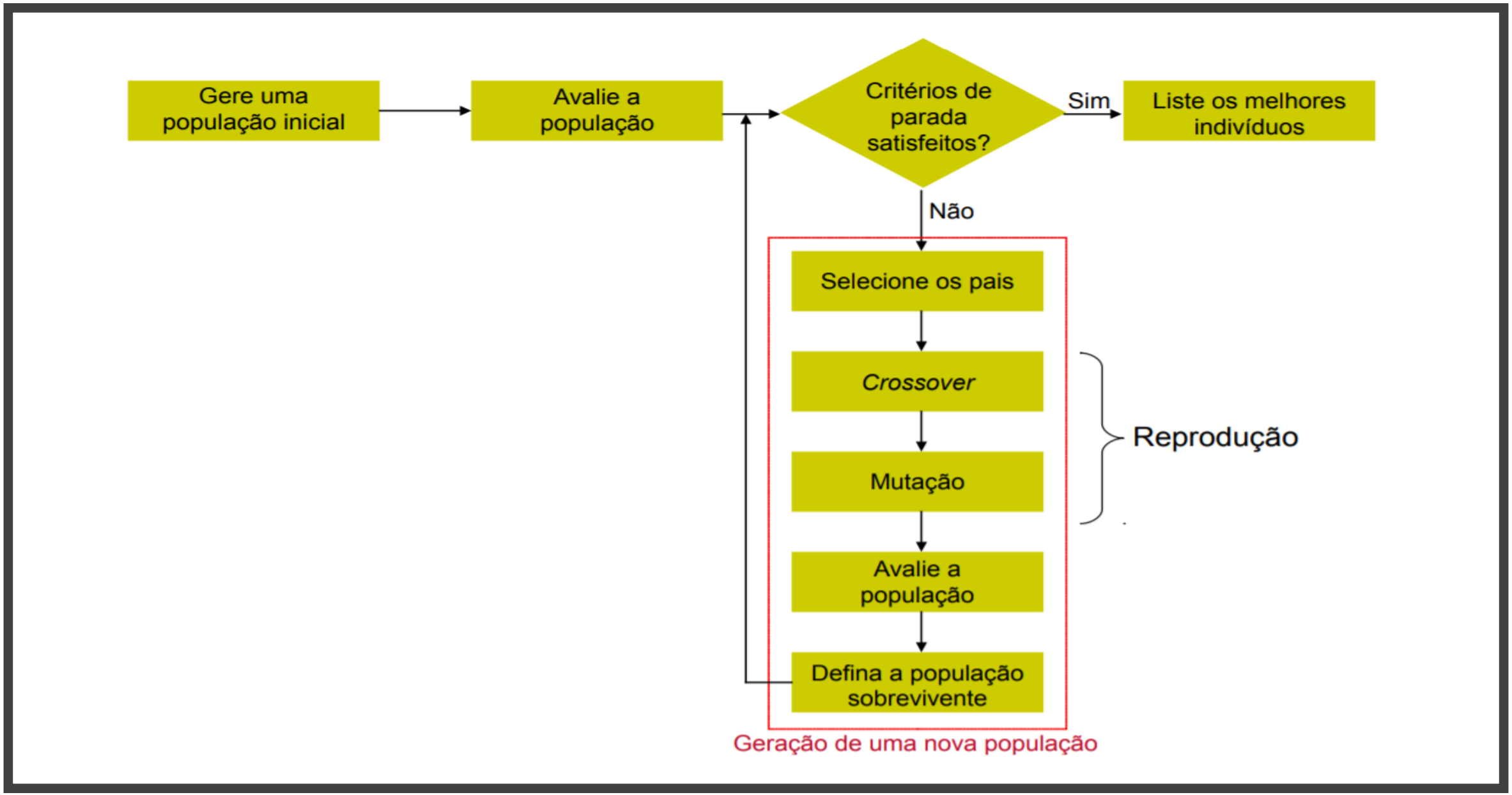

# Algoritmos Genéticos

CIÊNCIA DA COMPUTAÇÃO – PROFESSOR CLÁVISON M. ZAPELINI

Para resolver problemas utilizando Algoritmos Genéticos, precisamos dos seguintes componentes:

- **Representação**: como transformar uma solução em uma sequencia de genes (um cromossomo)?
- **Crossover** (Cruzamento): como podemos mesclar 2 soluções?
- **Mutação**: como podemos causar pequenas variações aleatórias em nossas soluções?
- **Fitness**: como calcular a aptidão de um individuo? Ou em outras palavras, o custo/erro/utilidade de uma solução?
- **Seleção**: como faremos a seleção dos mais aptos?

# Algoritmos Genéticos


Gere uma população inicial
Avalie a população
Critérios de parada satisfeitos?
Sim
Liste os melhores indivíduos
Não
Selecione os pais
Crossover
Mutação
Avalie a população
Defina a população sobrevivente
Reprodução
Geração de uma nova população

**Vetores de bits**

01010 = Tablet + Colar / 00111 = Celular + Colar + Anel

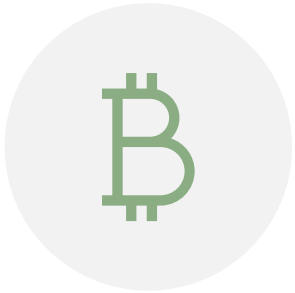
**Permutação**

CBADE (cidade C, depois B, A, D, E e retorna para C) / DABEC

**Contínua**

2x - 3y + 5z - 4w + 2xy – 3xz + 1 → 0.1 1.7 9.1 -3 / -1 0.2 5.2 1.4

# Representação Algoritmos Genéticos

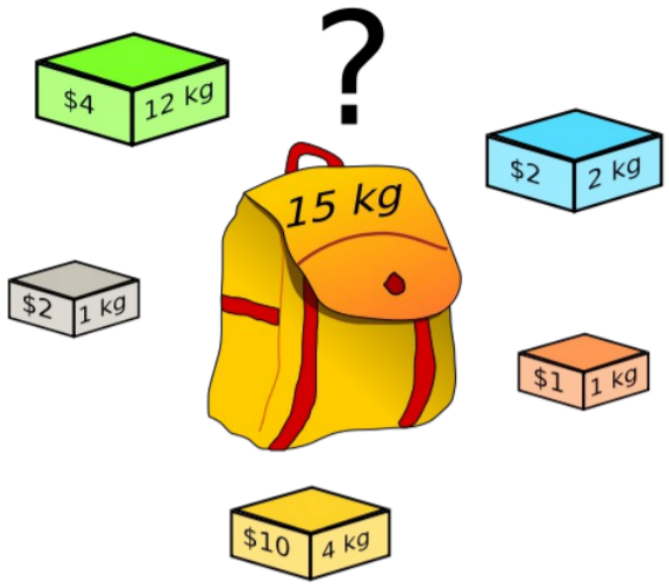


No caso do problema da mochila, queremos o maior valor possível sem ultrapassar o limite da mochila.

Podemos representar a “aptidão” de uma solução como a soma dos valores dos objetos e com uma penalidade para soluções que passam do limite de volume.

# Fitness - Algoritmos Genéticos

# Problema da mochila (knapsack)




- Vários itens que gostaria de levar em uma mochila
- Cada item com um peso e um benefício (valor)
- Há uma capacidade limite de peso
- Deve-se carregar itens com o máximo valor total sem superar o limite de peso

- Problema de otimização combinatória (NP-Completo)
- Estudado por mais de um século (desde ~1897)
- Resolvido por variados algoritmos
- Aplicações
  - Gravação de arquivos desperdiçando o mínimo espaço em cada mídia
  - Corte e empacotamento
  - Carregamento de veículos
  - Alocação de recursos em geral
    - Naves espaciais

# Problema da mochila (knapsack)

**Exemplo:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| • Item: | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| • Benefício: | 5 | 8 | 3 | 2 | 7 | 9 | 4 |
| • Peso: | 7 | 8 | 4 | 10 | 4 | 6 | 4 |

- Mochila suporta, no máximo, 22 quilos
- Adicione itens de modo a ter o máximo benefício

# Problema da mochila (knapsack)

- Codificação: 0 = não existe, 1 = existe na mochila

  Cromossomo: 1010110

| Item | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Cromossomo | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| Existe? | sim | não | sim | não | sim | sim | não |

  → Itens pegos: 1, 3 , 5, 6

- Geração aleatória de população com *n* cromossomos:
  a) 0101010
  b) 1100100
  c) 0100011

# Fitness - Algoritmos Genéticos

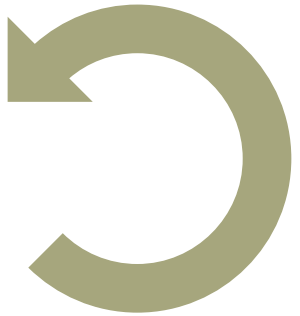
1. 0101010: Benefício = 19, Peso = 24 estourou, penalidade de **100** 19 – 100 = **-81**
2. 1100100: Benefício = **20**, Peso = 19
3. 0100011: Benefício = **21**, Peso = 18
4. 1110111: Benefício = 36, Peso = 33 estourou, penalidade de **100** 36 – 100 = **-64**

No caso do caixeiro viajante, bastaria somar o custo de cada trecho do caminho na solução.

No caso da maximização de função, poderíamos simplesmente computar o resultado da função com os valores da solução.

# Fitness - Algoritmos Genéticos

Um método simples seria selecionar as N melhores soluções, mas isto tem a desvantagem de tornar o algoritmo mais propenso a cair em máximos locais.



Uma solução mais usada é a da roleta.

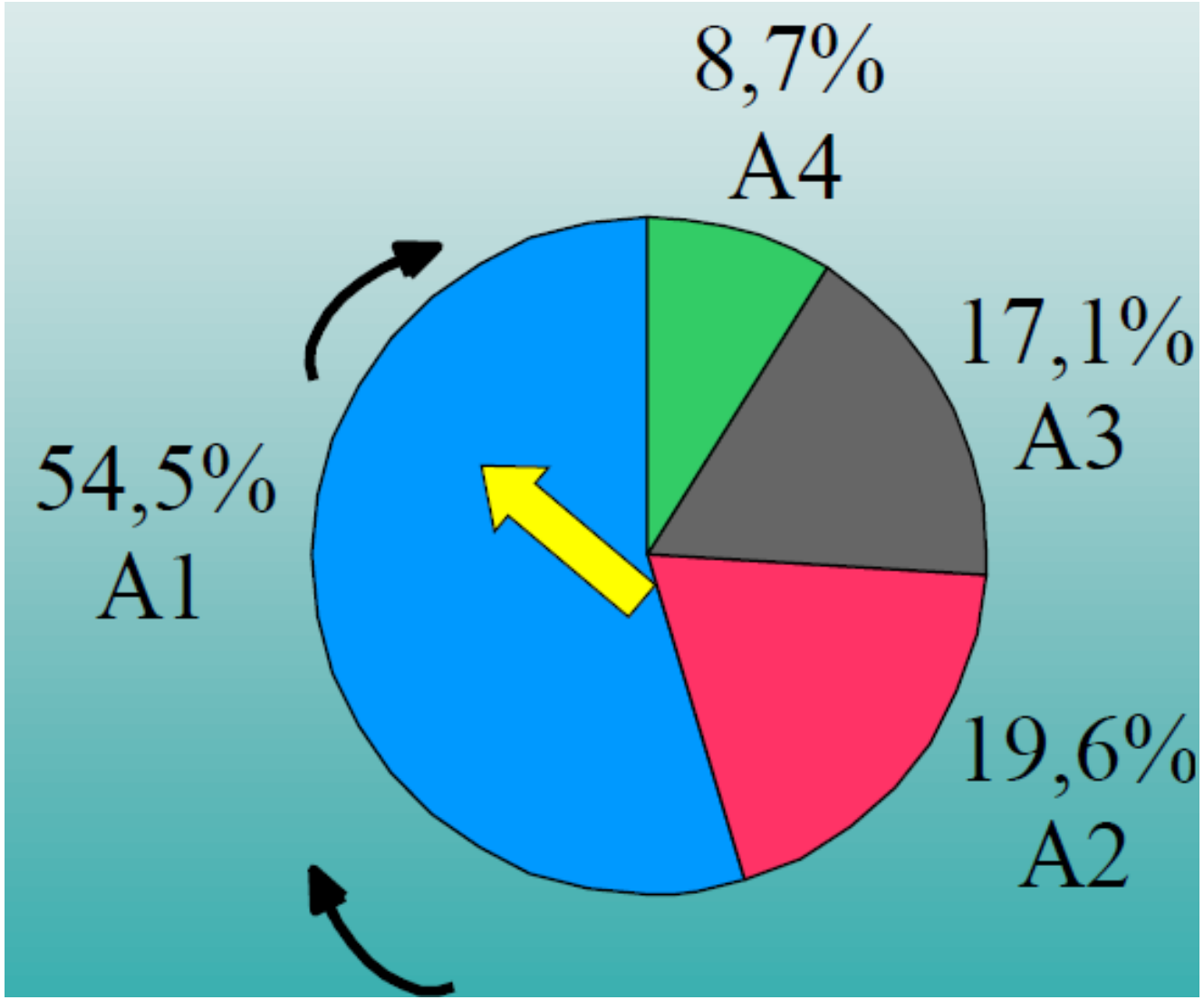
# Seleção - Algoritmos Genéticos

# Seleção - Algoritmos Genéticos

Com a roleta, cada solução tem uma probabilidade de ser selecionada proporcionalmente ao seu fitness

Com isso, mesmo as piores soluções podem se reproduzir, apesar de mais raro

# Seleção - Algoritmos Genéticos



# Fitness - Algoritmos Genéticos

1. 0101010: Benefício = 19, Peso = 24 estourou, penalidade de **100** 19 – 100 = **-81**
2. 1100100: Benefício = **20**, Peso = 19
3. 0100011: Benefício = **21**, Peso = 18
4. 1110111: Benefício = 36, Peso = 33 estourou, penalidade de **100** 36 – 100 = **-64**

# Seleção - Algoritmos Genéticos

-81 20 21 -64

- **Primeiro devemos subtrair o menor valor (-81) de todos os valores para eliminar os negativos**
  - -81 - (-81) = 0
  - 20 - (-81) = 101
  - 21 – (-81) = 102
  - -64 – (-81) = 17

# Seleção - Algoritmos Genéticos

- **Agora calculamos o fitness total:** 0 + 101 +102 + 17 = **220**
- **E as proporções:**

0 / 220 = 0

101 / 220 = 0,459

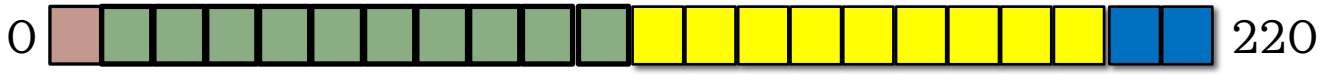
102 / 220 = 0,4636

17/220 = 0,0772

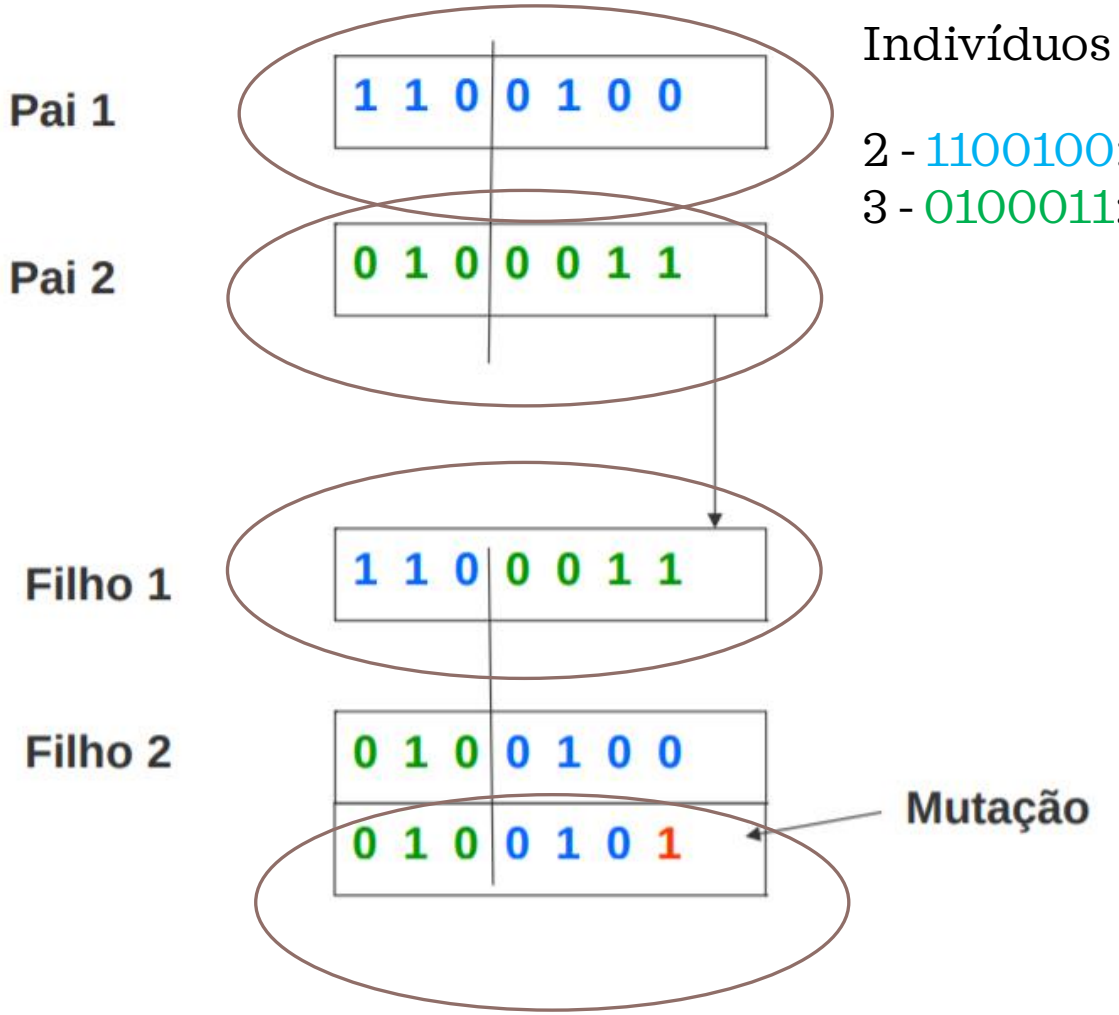
- **Estas são as probabilidades de selecionar cada solução!**

# Seleção - Algoritmos Genéticos

- **Agora calculamos o fitness total:** 0 + 101 +102 + 17 = **220**



- Crossover & mutação




Indivíduos selecionados

2 - 1100100: Benefício = **20**, Peso = 19
3 - 0100011: Benefício = **21**, Peso = 18

- Aceitação, substituição & teste
  - Definição de nova prole em uma nova população
  - Uso da nova população gerada para um próxima rodada do algoritmo
  - Se a condição final é satisfeita, então **finaliza**. Condições de finalização:
    - Número de gerações
    - Melhoramento da melhor solução
  - Caso contrário, retorne ao passo de **fitness**

# Problema da mochila para exercício:

**Entrada de dados**

Perguntar para o usuário quantos item tem disponível
Perguntar para o usuário o valor de cada item
Perguntar para o usuário o peso de cada item
Perguntar para o usuário a capacidade da mochila

**1ª Etapa do AG**

- Definir uma população inicial (quantos indivíduos?) – Usuário define!
- Criar aleatoriamente os indivíduos (possíveis soluções – vetor de bites)
- Calcular o fitness de cada individuo

- Mostar os indivíduos e seus fitness.

- **Modelo em Java do problema da mochila**

**https://www.cs.bgu.ac.il/~orlovm/teaching/assignments/intro-2009a-evo-knapsack.pdf**

- **Bibliotecas e Frameworks para Algoritmos Genéticos de modo generalista**

- Pyevolve (Framework open-source para Algoritmos Genéticos e Programação Genética – Python)

**http://pyevolve.sourceforge.net/**

- GAUL (Biblioteca open-source para Algoritmos Genéticos e metaheurísticas - Linguagem C)

**http://gaul.sourceforge.net/**

- JGAP (Pacote open-source para Algoritmos Genéticos – Java)

**http://jgap.sourceforge.net/**

- JAGA (Pacote open-source para Algoritmos Genéticos e Programação Genética – Java)

**http://www.jaga.org/**

- GAlib (Framework open-source para Algoritmos Genéticos – C++)

**http://lancet.mit.edu/ga/**

- EvolveDotNet (Framework open-source para Algoritmos Genéticos – C#)

**http://code.google.com/p/evolvedotnet/**

- jMetal (Framework open-source para otimização multiobjetivo que contém Algoritmos Genéticos – Java)

**http://jmetal.sourceforge.net/**