**Instituto Politécnico de Tomar**



Índice

[Roleta – Seleção 3](#_Toc326403705)

[Uniforme –Crossover –Recombinação 6](#_Toc326403706)

[Esquema 1 6](#_Toc326403707)

[Esquema 2 6](#_Toc326403708)

[Observação /Problema 7](#_Toc326403709)

[Operados treincation –substituição 7](#_Toc326403710)

[Mochila :Problema 8](#_Toc326403711)

[Reparação: aleatório 9](#_Toc326403712)

[Reparação:Psenda - Aleatória 10](#_Toc326403713)

[Fluxogramas 11](#_Toc326403714)

[Fitness Penalização 11](#_Toc326403715)

[Reparação Aleatória 11](#_Toc326403716)

[Reparação Pseudo Aleatória 12](#_Toc326403717)

[Operador Roleta 12](#_Toc326403718)

[Operador Crossover 13](#_Toc326403719)

[Operador SUS 14](#_Toc326403720)

[Operador Truncation 14](#_Toc326403721)

[Pseudo-codigo 15](#_Toc326403722)

[Fitness Penalização 15](#_Toc326403723)

[Reparação aleatória 15](#_Toc326403724)

[Reparação aleatória 16](#_Toc326403725)

[Operador crossover 16](#_Toc326403726)

[Operador Roleta 17](#_Toc326403727)

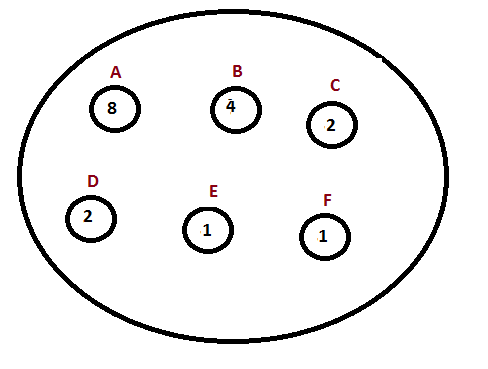
[Operador SUS 17](#_Toc326403728)

[Operador Truncation 18](#_Toc326403729)

## Roleta – Seleção

1. Atribuir uma percentagem a cada individuo de uma população com base no fitness e no total de fitness da população;
2. Juntar os indivíduos em linha de forma a criar percentagens acumuladas, com base na ordem dos indivíduos;
3. Gerar um número real aleatório entre 0 e1
4. Selecionam o individuo para onde o número aponta;
5. Repetir os passos 3º e 4º ate ter o número de indivíduos pretendidos.

Exemplo: Selecionam 4 indivíduos numa população de 6 indivíduos.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |

|  |  |  |  |
| --- | --- | --- | --- |
| 0,2 | 0,1 | 0,69 | 0,5 |
| A | A | C | B |

1. Juntos todos os indivíduos de uma população pela ordem que se encontram, de forma a fazer uma linha com os indivíduos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Indi A** | **Indi B** | **Indi C** | **Indi D** | … **N Indivíduos** |

1. Fazer a soma do fitness do indivíduo anterior ao ser, com base na ordem em que se encontram na linha;
2. Obter o total do somatório de todos os fitness, o fitness do ultimo indivíduo, pois é o acumulado de todos os indivíduos;
3. Gerar um ponto aleatório, inteiro ou real, dentro do intervalo 0 e total do fitness de todos os indivíduos, para ser o nosso ponto de partida;
4. Definir qual vai ser o offset que se vai acrescentar ao ponto de partido. Calcula através

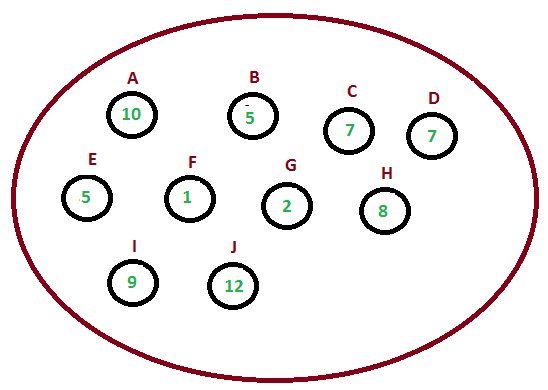
Total fitness

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

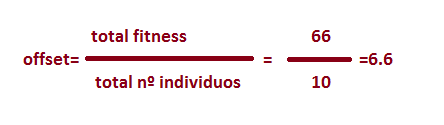
Total nº indivíduos

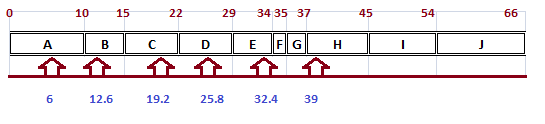
1. Selecionar o individuo para onde o ponto aponta;
2. Somar ao ponto o offset;
3. Repetir os pontos 6º e 7º ate obter o número de indivíduos selecionados pretendidos;
4. Caso se atinja o fim da linha de indivíduos então volta se ao início da linha

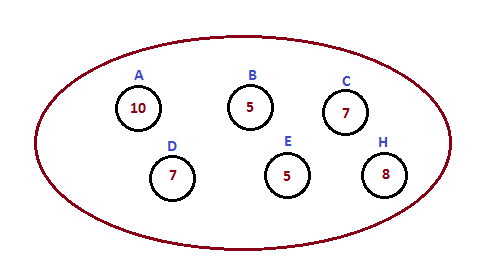
Exemplo: população 10 indivíduos e devemos escolher 6 indivíduos para reprodução



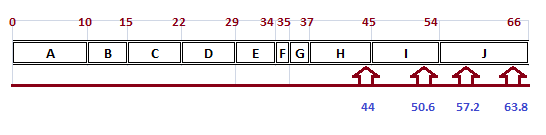
***População***

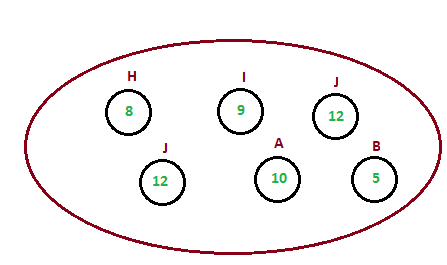






**Indivíduos selecionados**



**Indivíduos Selecionados**

# Uniforme–Crossover –Recombinação

1. Ter dois indivíduos, um “pai” e uma “mãe ”
2. Gerar uma cadeia de bits, ou mascara, para determinar quais os bits que não ser traçados;

Nota: A máscara tem que ter as mesmas dimensões do gene dos indivíduos pais

1. Para cada bit a “1” na mascara vai haver uma troca

Exemplo: Dois pais com gemes de tamanho de 10 bits e gerar dois filhos usando o operador uniform–crossover .

### Esquema 1

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pai | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |  | pais |
| Mãe | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | máscara |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Filho1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |  |
|  |  |  |  |  |  |  |  |  |  |  | filhos |
| Filho2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |  |

### Esquema 2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pai | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |  | pais |
| Mãe | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | máscara |
|  |  |  |  |  |  |  |  |  |  |  |  |
| filho | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |  |
|  |  |  |  |  |  |  |  |  |  |  | filhos |
| filha | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |  |

## Observação /Problema

Se a máscara for toda a “0” o indivíduo filho vai ser igual ao pai e o individuo filha vai ser igual à mãe.

Solução

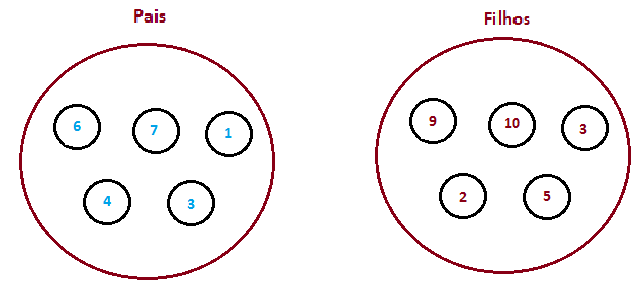
Garantir que 50% dos bits da mascara são a “1” e os restantes a “0”.

Os 50% referidos anteriormente pode ser um parâmetro deste operador, sendo 50% o valor recomendado por defeito

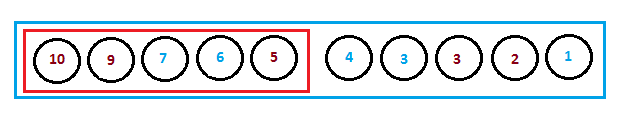
## Operados truncation–substituição

1. Ter duas populações, ou mais para aplicar o operador;
2. Juntar as várias populações numa só
3. Ordenar, com base no fitness de cada individuo, de forma descendente
4. Selecionar os indivíduos que surgem primeiro e criar uma nova população

Exemplo: Duas populações com 5 indivíduos cada, onde se quer criar uma nova população com 5 indivíduos



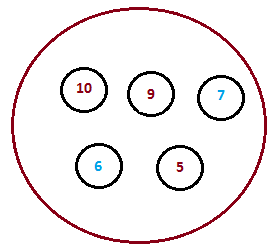
Indivíduos ordenados

 Os indivíduos das

duas populações

juntos e ordenados

Os 5 primeiros indivíduos



Nova os 5 indivíduos

População com melhor fitness

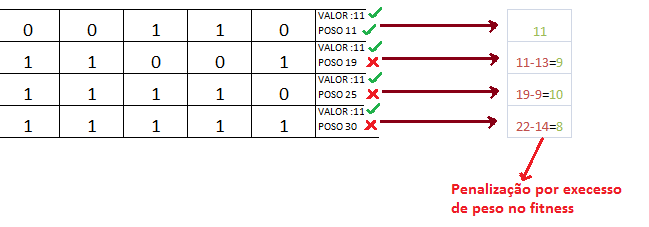
selecionados

# Mochila: Problema

Problema: levar o maior número de peças na mochila, que tem um limite de peso, mas ao mesmo tempo levar o maior valor na mochila.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Peso | 10 | 4 | 1 | 10 | 5 |
| Valor | 5 | 3 | 10 | 1 | 3 |

Máximo peso possível na mochila: 16

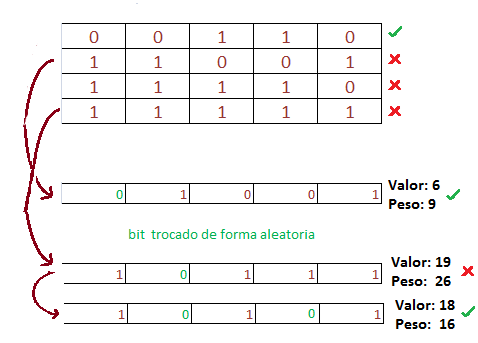


Tipos de penalizações

Linear: -31=3 (usada no exemplo anterior)

Quadrática: -32=9

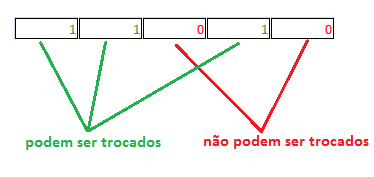
### Reparação: aleatório



Repetir o processo ate ter um peso aceitável

Só troca os bits que estão a “1”, pois são esses que estão a fazer a indivíduos / mochila ter peso a mais.

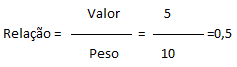
Exemplo: dos bits que podem ser trocados na reparação:



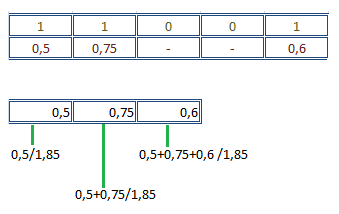
# Reparação: Pseudo-Aleatória

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Peso | 10 | 4 | 1 | 10 | 5 |
| Valor | 5 | 3 | 10 | 1 | 3 |
| Valor/Peso | 0,5 | 0,75 | 10 | 0,1 | 0,6 |

Calculo de Relação



Exemplo:



Atenção: Não podemos

usar dessa forma tão

direta porque as peças

com melhor relação

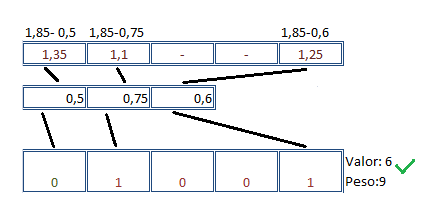
Valor /peso ficam com a

maior probabilidade de

serem escolhidos para

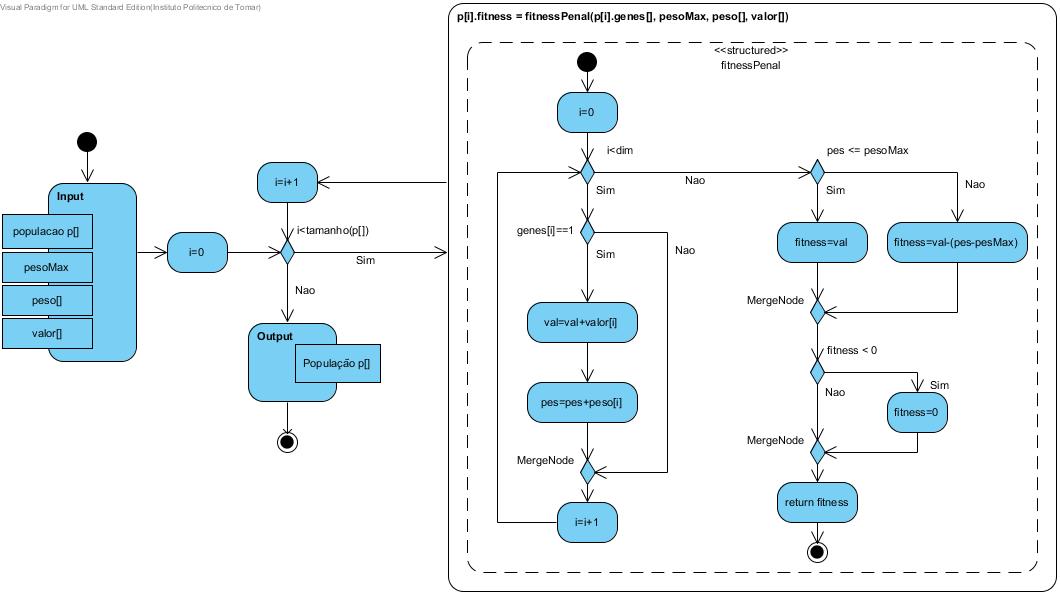
saírem da mochila.

Então:

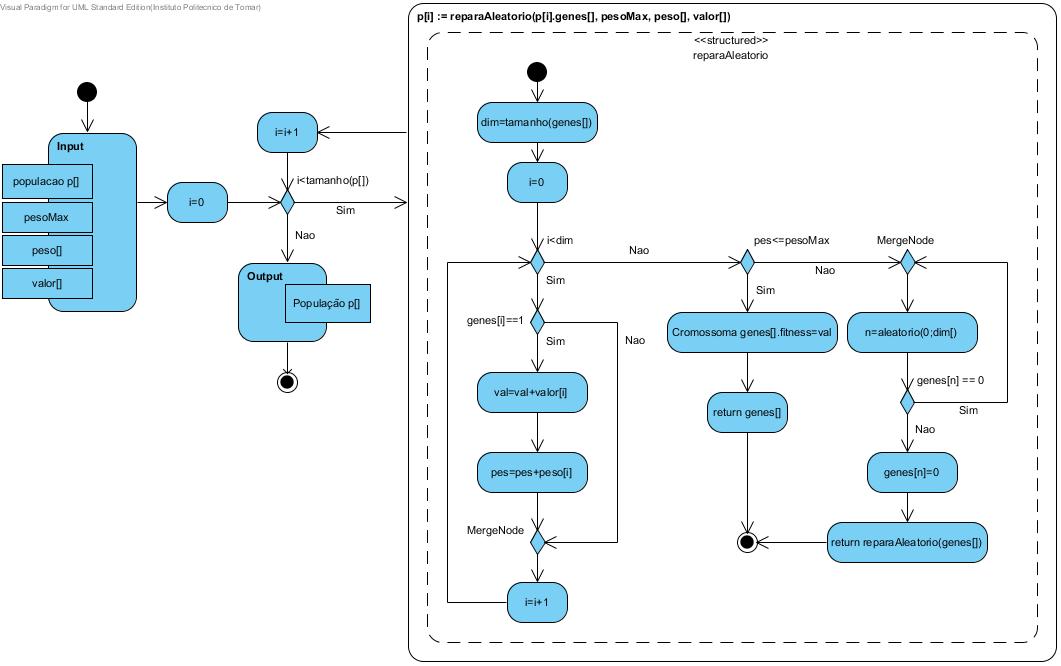


# Fluxogramas

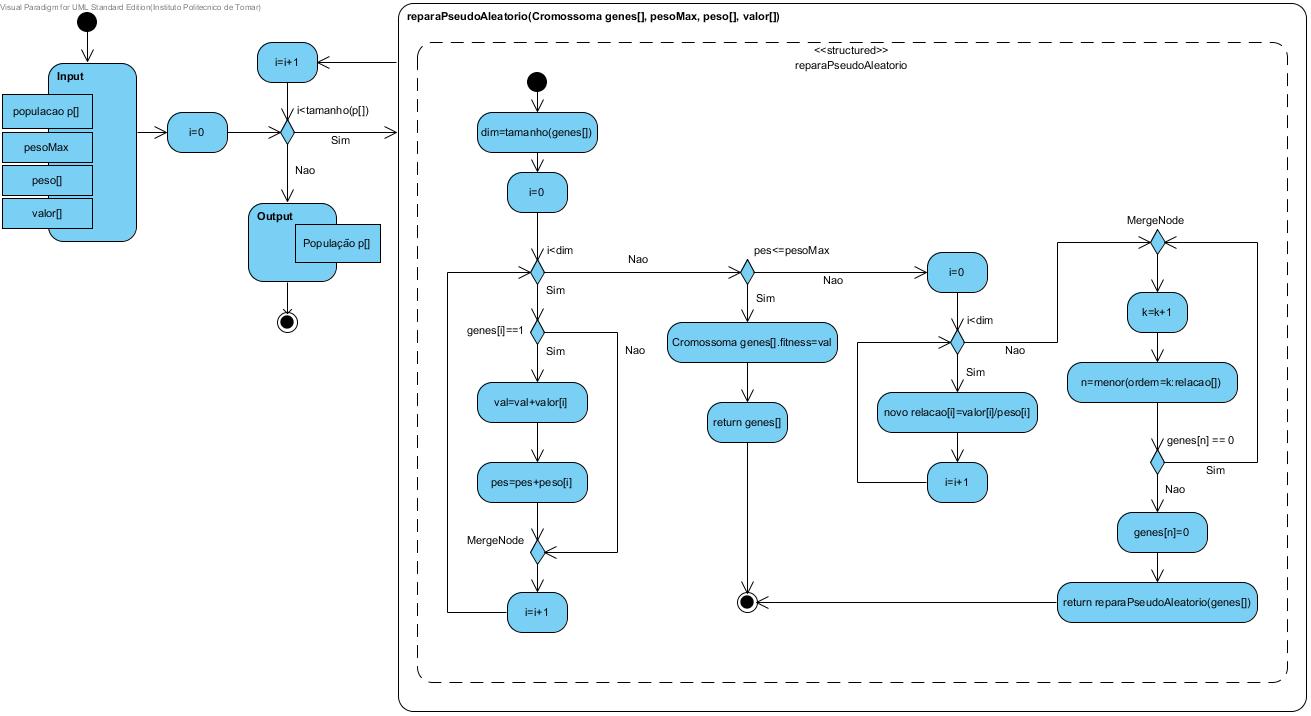
## Fitness Penalização



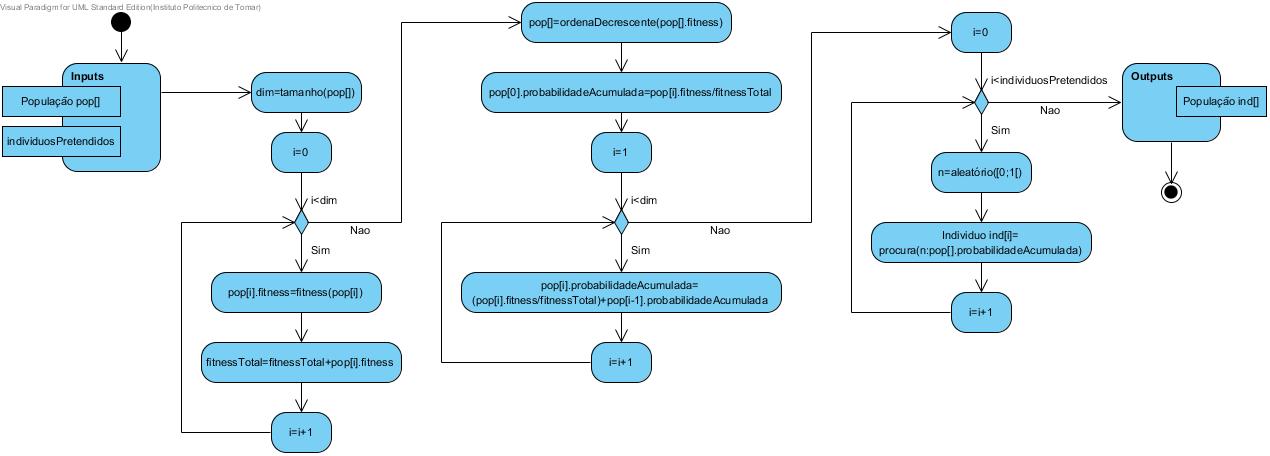
## Reparação Aleatória



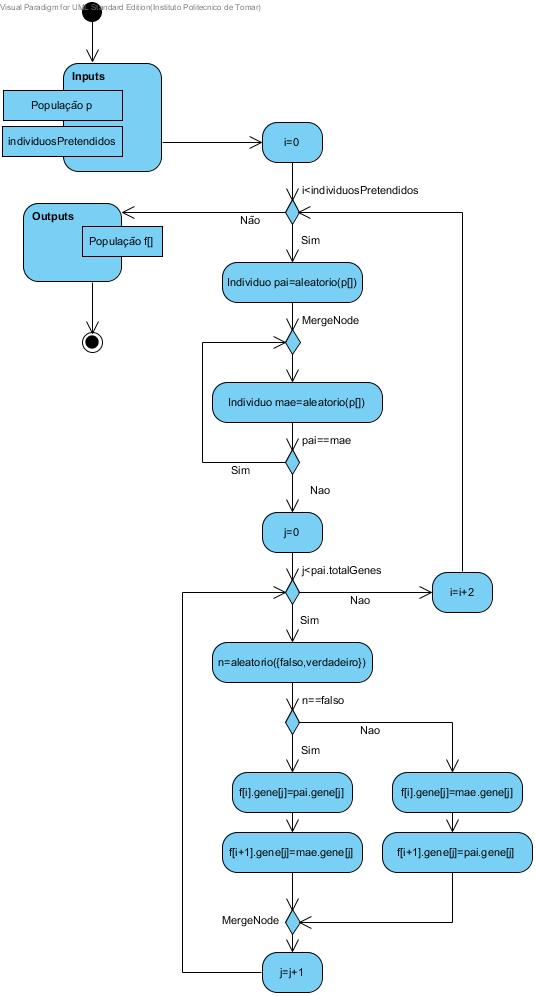
## Reparação Pseudo Aleatória



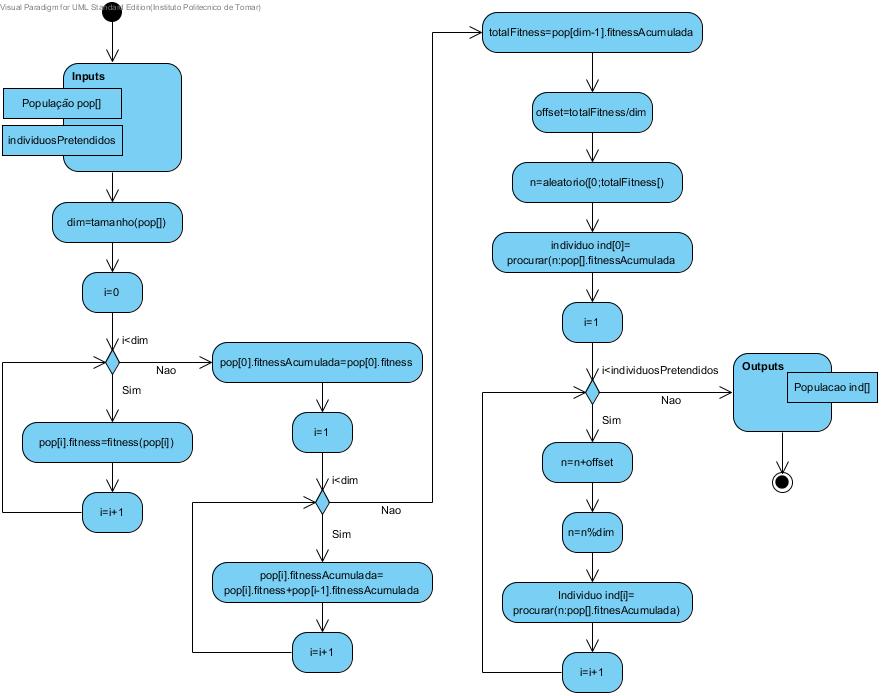
## Operador Roleta



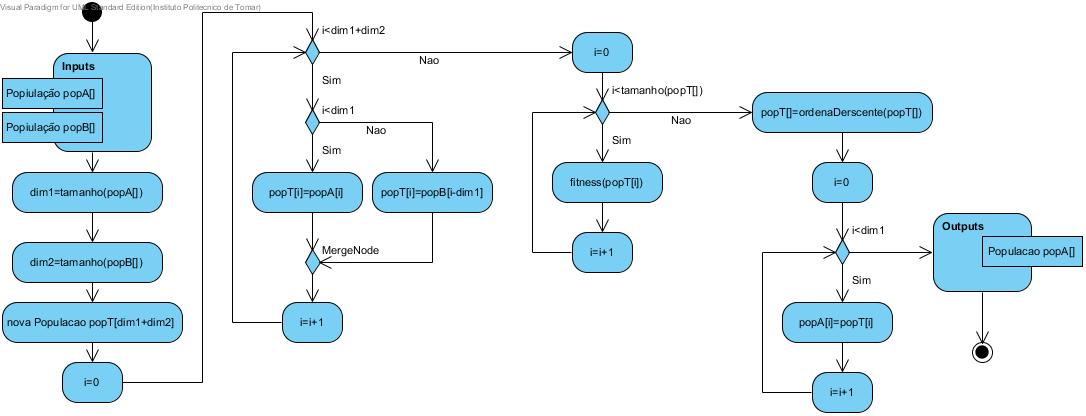
## Operador Crossover



## Operador SUS



## Operador Truncation



# Pseudocódigo

## Fitness Penalização

entra(População p[], pesoMax, peso[], valor[])

para i := 0 ate i < tamanho(p[])

p[i].fitness := fitnessPenal(p[i].genes[], pesoMax, peso[], valor[])

i := i+1

sai(População p[])

fitnessPenal(Individuo genes[], pesoMax, peso[], valor[])

para i := 0 ate i < tamanho(peso[])

se genes[i] = 1 entao

val := val+valor[i]

pes := pes+peso[i]

i := i+1

se pes <= pesoMax entao

fitness := val

senao

fitness := val-(pes-pesoMax)

se fitness < 0 entao

fitness := 0

retorna fitness

## Reparação aleatória

entra(População p[], pesoMax, peso[], valor[])

para i := 0 até i < tamanho(p[])

p[i] := reparaAleatorio(p[i].genes[], pesoMax, peso[], valor[])

i := i+1

sai(População p[])

reparaAleatorio(Individuo genes[], pesoMax, peso[], valor[])

dim := tamanho(genes[])

para i := 0 ate i < dim

se genes[i] = 1 entao

val := val+valor[i]

pes := pes+peso[i]

i := i+1

se pes <= pesoMax entao

Individuo genes[].fitness := val

retorna genes[]

senão

fazer

n := aleatorio([0;dim[)

enquanto (genes[n] = 0)

genes[n] := 0

retorna reparaAleatorio(genes[])

## Reparação aleatória

entra(População p[], pesoMax, peso[], valor[])

para i := 0 até i < tamanho(p[])

p[i] := reparaPseudoAleatorio(p[i].genes[], pesoMax, peso[], valor[])

i := i+1

sai(População p[])

reparaPseudoAleatorio(Cromossoma genes[], pesoMax, peso[], valor[])

dim := tamanho(genes[])

para i := 0 até i < dim

se genes[i] = 1 entao

val := val+valor[i]

pes := pes+peso[i]

i := i+1

se pes <= pesoMax então

Cromossoma genes[].fitness := val

retorna genes[]

senão

para i := 0 até i < dim

novo relacao[i] := valor[i]/peso[i]

fazer

k := k+1

n := menor(ordem:=k : relacao[])

enquanto (genes[n] = 0)

genes[n] := 0

retorna reparaPseudoAleatorio(genes[])

## Operador crossover

entra(População p[], individuosPretendidos)

para i := 0 até i < individuosPretendidos

Individuo pai := aleatório(p[])

fazer

Individuo mae := aleatório(p[])

enquanto (pai = mae)

para j := 0 ate j < pai.totalGenes

n := aleatório({falso,verdadeiro})

se n = falso então

f[i].gene[j] := pai.gene[j]

f[i+1].gene[j] := mae.gene[j]

senao

f[i].gene[j] := mae.gene[j]

f[i+1].gene[j] := pai.gene[j]

j := j+1

i := i+2

sai(População f[])

## Operador Roleta

entra(População pop[], individuosPretendidos)

dim := tamanho(pop[])

para i := 0 ate i < dim

pop[i].fitness := fitness(pop[i])

fitnessTotal := fitnessTotal+pop[i].fitness

i := i+1

pop[] := ordenaDecrescente(pop[].fitness)

pop[0].probabilidadeAcumulada := pop[i].fitness/fitnessTotal

para i := 1 ate i < dim

pop[i].probabilidadeAcumulada := (pop[i].fitness/fitnessTotal)+pop[i-1].probabilidadeAcumulada

i := i+1

para i := 0 ate i < individuosPretendidos

n := aleatório([0;1[)

Individuo ind[i] := procura(n : pop[].probabilidadeAcumulada)

i := i+1

sai(População ind[])

## Operador SUS

entra(Populacao pop[], individuosPretendidos)

dim := tamanho(pop[])

para i := 0 ate i < dim

pop[i].fitness := fitness(pop[i])

i := i+1

pop[0].fitnessAcumulada := pop[0].fitness

para i := 1 ate i < dim

pop[i].fitnessAcumulada := pop[i].fitness + pop[i-1].fitnessAcumulada

i := i+1

totalFitness := pop[dim-1].fitnessAcumulada

offset := totalFitness/dim

n := aleatorio([0;totalFitness[)

individuo ind[0] := procurar(n : pop[].fitnessAcumulada)

para i := 1 ate i < individuosPretendidos

n := n+offset

n := n%dim

Individuo ind[i] := procurar(n : pop[].fitnessAcumulada)

i := i+1

sai(Populacao ind[])

## Operador Truncation

entra(Populacao popA[], População popB[])

dim1 := tamanho(popA[])

dim2 := tamanho(popB[])

nova Populacao popT[dim1+dim2]

para i := 0 ate i < dim1+dim2

se i < dim1 então

popT[i] := popA[i]

senao

popT[i] := popB[i-dim1]

i := i+1

para i := 0 ate i < tamanho(popT[])

fitness(popT[i])

i := i+1

popT[] := ordenaDecrescente(popT[])

para i := 0 ate i < dim1

popA[i] := popT[i]

i := i+1

sai(Populacao popA[])