



# LifeInspiration

Projecto de Sistemas de Informação

Life Inspiration V0.2 - KnapSack

# KnapSack

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

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

- Máximo peso possível na mochila: **16**

|   |   |   |   |   |                          |   |         |
|---|---|---|---|---|--------------------------|---|---------|
| 0 | 0 | 1 | 1 | 0 | VALOR :11 ✓<br>POSO 11 ✓ | → | 11      |
| 1 | 1 | 0 | 0 | 1 | VALOR :11 ✓<br>POSO 19 ✗ | → | 11-13=9 |
| 1 | 1 | 1 | 1 | 0 | VALOR :11 ✓<br>POSO 25 ✗ | → | 19-9=10 |
| 1 | 1 | 1 | 1 | 1 | VALOR :11 ✓<br>POSO 30 ✗ | → | 22-14=8 |

Tipos de penalizações

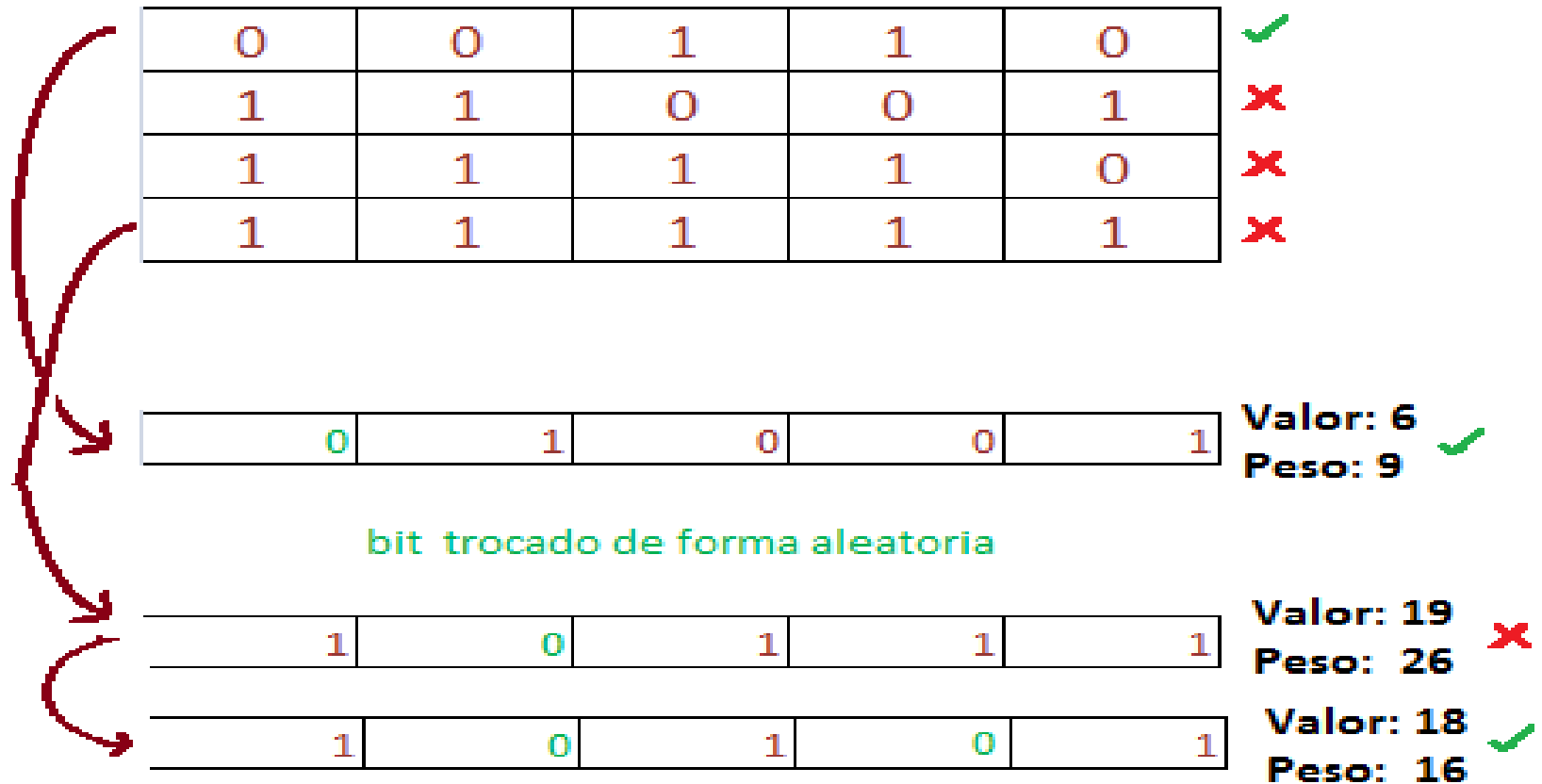
Linear:  $-3^1=3$

(usada no exemplo anterior)

Quadrática:  $-3^2=9$

Penalização por excesso de peso no fitness

# KnapSack – Reparação Aleatória



- Repetir o processo até ter um peso aceitável

# KnapSack – Reparação Aleatória

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



```
reparaAleatorio(Individuo genes[])  
    pesoMax  
    peso[]  
    valor[]  
    dim = tamanho(genes[])  
    para i = 0 até i = dim-1  
        se genes[i] = 1 então  
            val = val+valor[i]  
            pes = pes+peso[i]  
        i = i+1
```

```
se pes <= pesoMax então  
    Individuo genes[].fitness = val  
    retorna genes[]  
senão  
    fazer  
        n = aleatório([0;dim[])  
    enquanto (genes[n] = 0)  
        genes[n] = 0  
        reparaAleatorio(genes[])
```

# KnapSack –Reparação Pseudoaleatória

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



## ► Calculo da relação:

$$\text{Relação} = \frac{\text{Valor}}{\text{Peso}} = \frac{5}{10} = 0,5$$

```
reparaPseudoAleatorio(Individuo genes[])
    pesoMax
    peso[]
    valor[]
    dim = tamanho(genes[])
    para i = 0 até i = dim-1
        se genes[i] = 1 então
            val = val+valor[i]
            pes = pes+peso[i]
            i = i+1
    se pes <= pesoMax então
        Individuo genes[].fitness = val
        retorna genes[]
    ~
```

senão

```
-
    para i = 0 até i < dim
        relacao[i] = valor[i]/peso[i]
    fazer
        k = k+1
        n = menor(ordem=k : relacao[])
    enquanto (genes[n] = 0)
        genes[n] = 0
        reparaPseudoAleatorio(genes[])
```

# KnapSack –Reparação Pseudoaleatória

## ► Exemplo:

|     |      |   |   |     |
|-----|------|---|---|-----|
| 1   | 1    | 0 | 0 | 1   |
| 0,5 | 0,75 | - | - | 0,6 |

|            |                 |                     |
|------------|-----------------|---------------------|
| 0,5        | 0,75            | 0,6                 |
| $0,5/1,85$ | $0,5+0,75/1,85$ | $0,5+0,75+0,6/1,85$ |

**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.

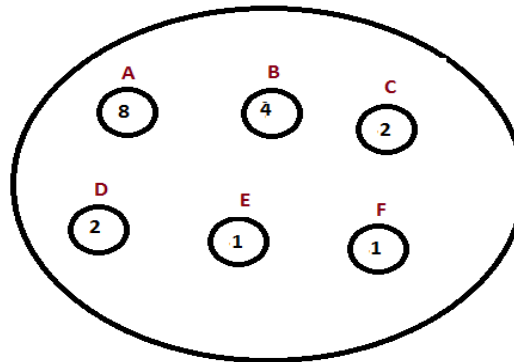
## ► Então:

|              |               |     |   |              |
|--------------|---------------|-----|---|--------------|
| $1,85 - 0,5$ | $1,85 - 0,75$ |     |   | $1,85 - 0,6$ |
| 1,35         | 1,1           | -   | - | 1,25         |
| 0,5          | 0,75          | 0,6 |   |              |
| 0            | 1             | 0   | 0 | 1            |

Valor: 6 ✓  
Peso: 9

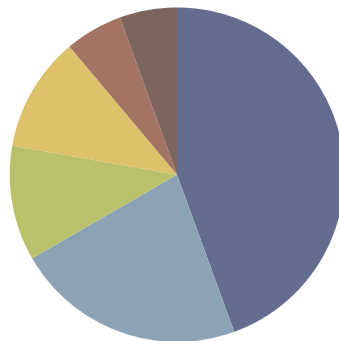
# Operador : Roleta - Seleccção

- ▶ **Exemplo: Selecionar 4 individuos numa população de 6.**



- ▶ 1º) Atribuir uma percentagem a cada individuo de uma população com base no fitness e no total de fitness da população ;

- ▶ **FitnessTotal=18**



■ A  
■ B  
■ C  
■ D  
■ E  
■ F

*para  $i = 0$  até  $i = \text{dim}-1$*

*$a[i].\text{fitness} = \text{fitness}(a[i])$*

*$\text{fitnessTotal} = \text{fitnessTotal} + a[i].\text{fitness}$*

*$i = i + 1$*

# Operador : Roleta - Seleccção

---

- ▶ 2º) Juntar os indivíduos em linha de forma a criar percentagens acumuladas, com base na ordem dos indivíduos;
- ▶ *ordenar\_decrescente(a[].fitness)*
- ▶ *para i = 0 até i = dim-1*
- ▶     *se i = 0 então*
- ▶          *$a[i].probabilidadeAcumulada = a[i].fitness/fitnessTotal$*
- ▶     *senão*
- ▶          *$a[i].probabilidadeAcumulada = (a[i].fitness/fitnessTotal) + a[i-1].probabilidadeAcumulada$*
- ▶     *i = i+1*



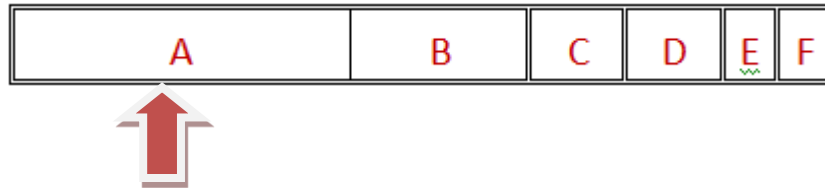


# Operador : Roleta - Selecção

- 3º) Gerar um numero real aleatório entre 0 e 1
  - $n = \text{aleatório}([0;1])$
- 4º) Seleccionam o individuo para onde o número aponta;

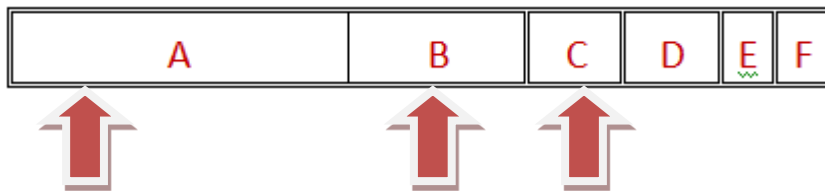
- $\text{Individuo } b[i] = \text{procurar}(n : a[].\text{probabilidadeAcumulada})$

- $N=0.2$



- Repetir os passos 3º e 4º ate ter o número de indivíduos pretendidos.

- $N=0.1, N=0.5, N=0.69$

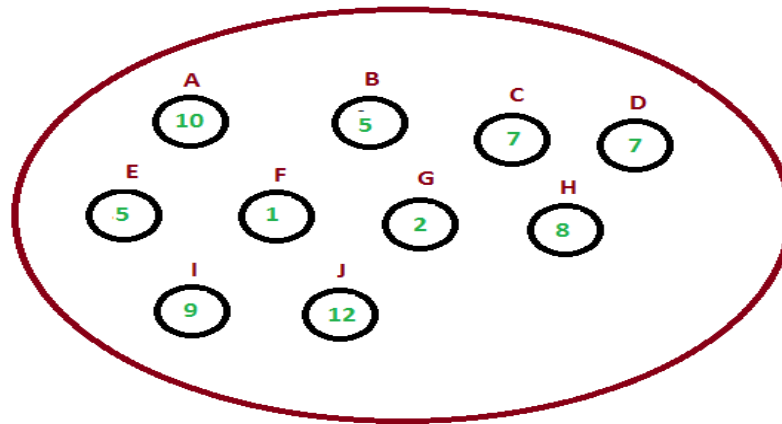


- Individuos seleccionados:

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

# Operador : SUS - Seleção

- ▶ **Exemplo: Escolher 6 indivíduos de uma população de 10**



- ▶ 1º) Juntar todos os indivíduos de uma população pela ordem que se encontram, de forma a fazer uma linha com os indivíduos;



- ▶ 2º) Fazer a soma do fitness do individuo anterior ao seu, com base na ordem em que se encontram na linha;

- ▶ *para  $i = 1$  até  $i = \text{dim}-1$*

- ▶  *$a[i].\text{fitnessAcumulada} = a[i].\text{fitness} + a[i-1].\text{fitnessAcumulada}$*

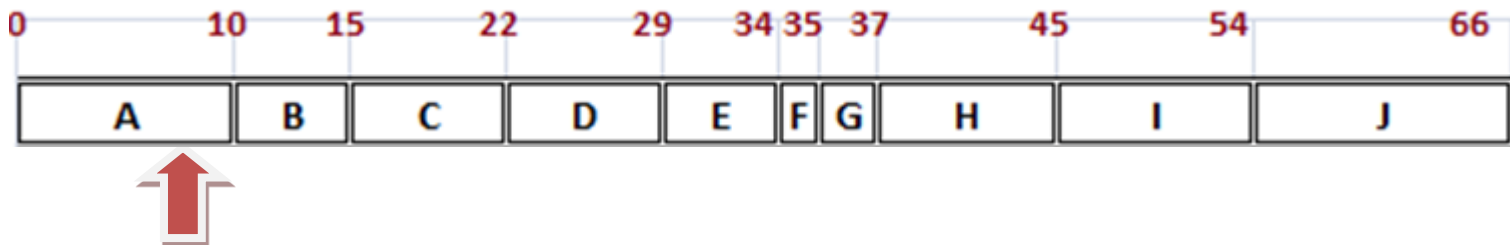
- ▶  *$i = i+1$*

# Operador : SUS - Seleção

- ▶ 3º) Obter o total do somatório de todos os fitness, o fitness do ultimo indivíduo, pois é o acumulado de todos os indivíduos;
  - ▶ Fitness Total: **66** *totalFitness = a[dim-1].fitnessAcumulada*
- ▶ 4º) 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;
  - ▶ Ponto de Partida : **6** *n = aleatório([0 : totalFitness])*
- ▶ 5º) Definir qual vai ser o offset que se vai acrescentar ao ponto de partido.

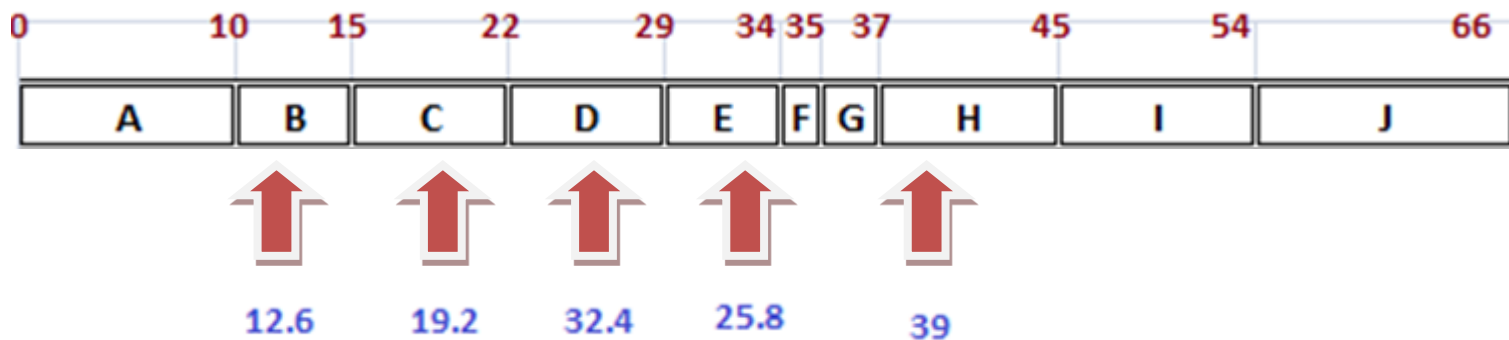
$$\text{offset} = \frac{\text{total fitness}}{\text{total nº individuos}} = \frac{66}{10} = 6.6$$

- ▶ 6º) Selecionar o individuo para onde o ponto aponta;



# Operador : SUS - Selecção

- ▶ 7º) Somar ao ponto o offset;
  - ▶ Ponto:  $6+6.6=12.6$   $n = n+offset$
- ▶ 8º) Repetir os pontos 6º e 7º ate obter o numero de indivíduos seleccionados pretendidos;



*para  $i = 1$  até  $i = individuosPretendidos-1$*

*$n = n+offset$*

*$n = n\%dim$*

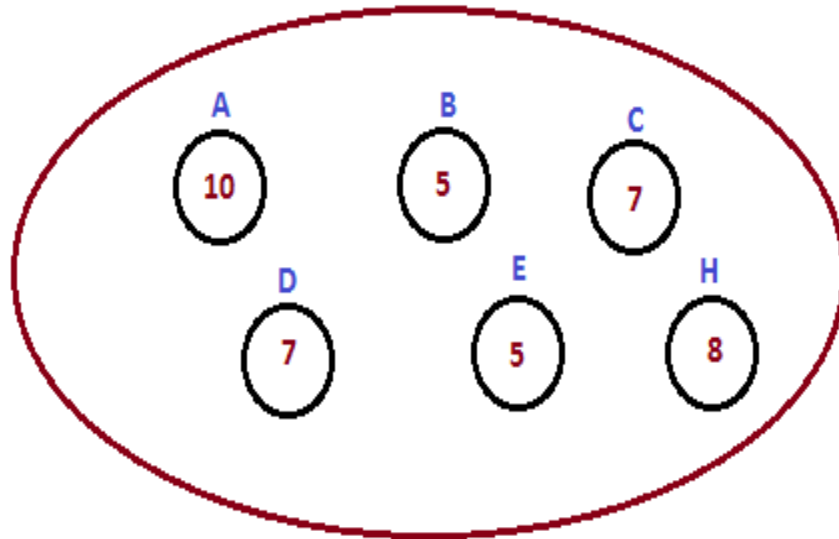
*Individuo  $b[i] = procurar(n : a[].fitnessAcumulada)$*

*$i = i+1$*

# Operador : SUS - Seleccção

---

## ► Indivíduos Seleccionados:



# Operador : Uniform - Crossover

- ▶ **Exemplo: Dois pais com genes do tamanho de 10 bits e gerar dois filhos.**






- ▶ 1º) Selecionar 2 indivíduos, um pai e uma mãe;

|     |   |   |   |   |   |   |   |   |   |   |
|-----|---|---|---|---|---|---|---|---|---|---|
| Pai | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
|     |   |   |   |   |   |   |   |   |   |   |
| Mãe | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

- ▶ 2º) Gerar uma cadeia de bits, máscara, para determinar quais os bits que vão ser trocados;

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|

- ▶ 3º) Para cada bit a “1” na mascara vai haver uma troca

|        |   |   |   |   |   |   |   |   |   |   |
|--------|---|---|---|---|---|---|---|---|---|---|
| Filho1 | 0   | 0 | 0 | 1   | 1 | 1 | 1 | 0   | 0   | 1   |
|        |  |   |   |  |   |   |   |  |  |  |
| Filho2 | 1   | 0 | 0 | 0   | 1 | 1 | 0 | 0   | 1   | 0   |

# Operador : Uniform - Crossover

---

## ▶ Problema:

- ▶ Se a mascara for toda a “0” o indivíduo Filho1 vai ser igual ao Pai e o individuo Filho2 vai ser igual a Mãe.

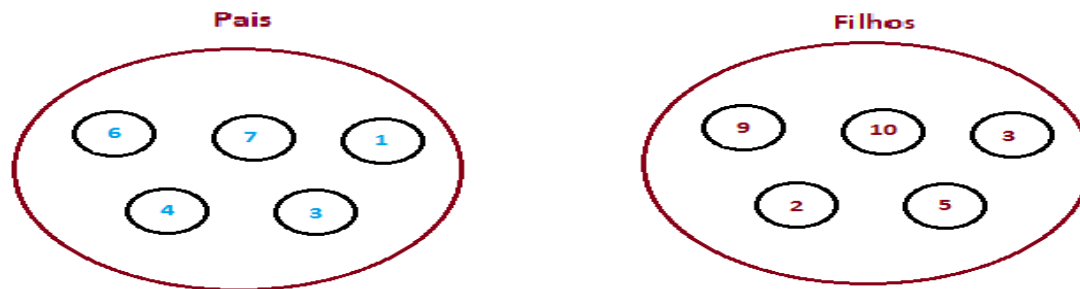
## ▶ Solução:

- ▶ Garantir que 50% dos bits da mascara são a “1” e os restantes a “0”.
- ▶ Os 50% referidos anteriormente podem ser um parâmetro deste operador, sendo 50% o valor recomendado por defeito;



# Operador : Truncation - Substituição

- ▶ **Exemplo: Duas populações com 5 indivíduos cada, onde se quer criar uma nova população com 5 indivíduos ;**
- ▶ 1º) Ter duas populações, ou mais, para aplicar o operador;



- ▶ 2º) Juntar as várias populações numa só ;

```
para i = 0 até i = dim1+dim2-1
    se i < dim1 então
        temp[i] = a[i]
    senão
        temp[i] = b[i-dim1]
    i = i+1
```

```
para i = 0 até i = tamanho(temp[])-1
    fitness(temp[i])
    i = i+1
```



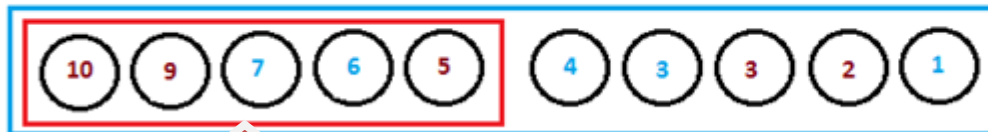
# Operador : Truncation - Substituição

- ▶ 3º) Ordenar, com base no fitness de cada individuo, de forma descendente ;

```
ordenar_decrescente(temp[])
```

- ▶ 4º) Selecionar os indivíduos que surgem primeiro e criar uma nova população ;

Indivíduos ordenados



Os indivíduos das  
duas populações  
juntos e ordenados

Os 5 primeiros indivíduos

```
para i = 0 até i = dim1-1  
    a[i] = temp[i]  
    i = i+1
```

# Operador : Truncation - Substituição

---

- Nova população:

