Universidade Federal de São Carlos - UFSCar
Departamento de Computação - DC
CEP 13565-905, Rod. Washington Luiz, s/n, São Carlos, SP

Algoritmos Gulosos - Parte 2

Prof. Dr. Alan Demétrius Baria Valejo

CCO-00.2.01 - Projeto e Análise de Algoritmos (*Design And Analysis Of Algorithms*)
1001525 - Projeto e Análise de Algoritmos - Turma A

Roteiro

- Problema do troco
- Mochila fracionária
- Seleção de atividades

Roteiro

A grande dificuldade é qual decisão local utilizar

- Problema do troco
- Mochila fracionária
- Seleção de atividades

Problema do troco
Definição

- Problema do troco ou *coin change problem*
  - **Dado o sistema monetário brasileiro *S* com um conjunto de moedas pré-definidas**
  - **Objetivo: qual é a menor quantidade de moedas para retornar um troco *c*?**
    - Ou seja, consiste em **encontrar a combinação com menor número de moedas** cuja soma seja igual a uma quantia determinada, a partir de uma lista de moedas válidas que possuem disponibilidade infinita.

S = {100, 50, 25, 10, 5, 1} e c = 147

- Caso especial do Problema da Mochila
  - No caso da mochila, busca-se a alocação de objetos que maximiza o valor de uma mochila com uma restrição de peso, dados os pesos e valores de cada objeto
  - O problema do troco seria o da mochila "ao contrário": dado um valor fixo para a mochila, encontrar a combinação de objetos com menor peso que fornece esse valor.

- Qual a solução?

S = {100, 50, 25, 10, 5, 1} e c = 147

Problema do troco
Algoritmo Guloso

1. Ordenar moedas em valor decrescente
2. Selecionar a moeda de valor mais alto e que não excede o valor do troco
3. Se a quantia selecionada não é igual ao valor do troco, volte ao passo 2
4. Caso contrário, o problema está resolvido

Problema do troco
Algoritmo Guloso

1. Ordenar moedas em valor decrescente
2. Selecionar a moeda de valor mais alto e que não excede o valor do troco
3. Se a quantia selecionada não é igual ao valor do troco, volte ao passo 2
4. Caso contrário, o problema está resolvido

1. Ordenar moedas em valor decrescente
2. Selecionar a moeda de valor mais alto e que não excede o valor do troco
3. Se a quantia selecionada não é igual ao valor do troco, volte ao passo 2
4. Caso contrário, o problema está resolvido

Problema do troco
Algoritmo Guloso

1. Ordenar moedas em valor decrescente
2. Selecionar a moeda de valor mais alto e que não excede o valor do troco
3. Se a quantia selecionada não é igual ao valor do troco, volte ao passo 2
4. Caso contrário, o problema está resolvido

1. Ordenar moedas em valor decrescente
2. Selecionar a moeda de valor mais alto e que não excede o valor do troco
3. Se a quantia selecionada não é igual ao valor do troco, volte ao passo 2
4. Caso contrário, o problema está resolvido

Problema do troco
Ilustrando o algoritmo

- Ilustrando com um caso simples
- S = {50, 100, 25, 5, 10, 1} e c = 147

```
moedas = [50, 100, 5, 10, 25, 1]
moedas.sort()
# [100, 50, 20, 10, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

```
moedas = [50, 100, 5, 10, 25, 1]
moedas.sort()
# [100, 50, 20, 10, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

Seleciona a maior moeda de valor “moedas[i]”
Seleciona o maior número possível de moedas de valor “moedas[i]”

```
moedas = [50, 100, 5, 10, 25, 1]
moedas.sort()
# [100, 50, 20, 10, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

Subtrai do troco o valor selecionado

```
moedas = [50, 100, 5, 10, 25, 1]
moedas.sort()
# [100, 50, 20, 10, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

Incrementa o total de moedas

E a complexidade?

```
moedas = [50, 10,  10, 25, 1]
moedas.sort()
# [100, 50, 20, 10, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

```
moedas = [50, 100, 5, 10, 25, 1]
moedas.sort()
# [100, 50, 20, 10, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

$O(n \log_2 n)$

$O(n)$

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

S = {4, 3, 1}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

S = {4, 3, 1}
C = 6 – 4 => 2

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

S = {4, 3, 1}
C = 6 – 4 => 2
C = 2 – 1 => 1

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

S = {4, 3, 1}
C = 6 – 4 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

S = {4, 3, 1}
C = 6 – 4 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {4, 1, 1}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão gananciosa funciona
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de moedas: 1 centavo, 7 centavos e 10 centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavos e um troco de c=6

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

S = {4, 3, 1}
C = 6 – 4 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {4, 1, 1}
Porém, a resposta ótimo é T = {3, 3}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão ga
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de
    centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavo

Como resolver?
Eu posso fazer melhor?

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

S = {4, 3, 1}
C = 6 – 4 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {4, 1, 1}
Porém, a resposta ótimo é T = {3, 3}

- Mas será que a versão gananciosa sempre é a melhor?
  - No caso da lista de moedas válidas no Brasil, ilustrada anteriormente, a versão ga
  - Mas em outros casos, por exemplo, um sistema fictício com apenas três tipos de
    centavos e um troco de c=14
  - Outro caso, tendo apenas três tipos de moedas: 1 centavo, 3 centavos e 4 centavo

Como resolver?
Eu posso fazer melhor?
R: Programação dinâmica

S = {10, 7, 1}
C = 14 – 10 => 4
C = 4 – 1 => 3
C = 3 – 1 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {10, 1, 1, 1, 1}
Porém, a resposta ótimo é T = {7, 7}

S = {4, 3, 1}
C = 6 – 4 => 2
C = 2 – 1 => 1
C = 1 – 1 => 0
T = {4, 1, 1}
Porém, a resposta ótimo é T = {3, 3}

Mochila fracionária
Definição

- Imagine que você é um vendedor de produtos alimentícios diversos
  - Grãos, temperos, farinha, ...
- Para levar os produtos à feira, você precisa transportá-los até um certo limite de peso que a sua mochila aguenta
- Cada grão tem um preço por unidade de massa.
  - A massa de alguns produtos é mais caros do que outros
- Quais e quanto de cada produto você deve levar à feira de modo a maximizar o seu faturamento, caso venda tudo?

Mochila fracionária
Definição

- Imagine que você é um vendedor de produtos alimentícios diversos
  - Grãos, temperos, farinha, ...
- Para levar os produtos à feira, você precisa transportá-los até um certo limite de peso que a sua mochila aguenta
- Cada grão tem um preço por unidade de massa.
  - A massa de alguns produtos é mais caros do que outros
- Quais e quanto de cada produto você deve levar à feira de modo a maximizar o seu faturamento, caso venda tudo?

Mochila fracionária
Definição

- Imagine que você é um vendedor de produtos alimentícios diversos
  - Grãos, temperos, farinha, ...
- Para levar os produtos à feira, você precisa transportá-los até um certo limite de peso que a sua mochila aguenta
- Cada grão tem um preço por unidade de massa.
  - A massa de alguns produtos é mais caros do que outros
- Quais e quanto de cada produto você deve levar à feira de modo a maximizar o seu faturamento, caso venda tudo?

Mochila fracionária
Definição

- Imagine que você é um vendedor de produtos alimentícios diversos
  - Grãos, temperos, farinha, ...
- Para levar os produtos à feira, você precisa transportá-los até um certo limite de peso que a sua mochila aguenta
- Cada grão tem um preço por unidade de massa.
  - A massa de alguns produtos é mais caros do que outros
- Quais e quanto de cada produto você deve levar à feira de modo a maximizar o seu faturamento, caso venda tudo?

Mochila fracionária
Definição

- Imagine que você é um vendedor de produtos alimentícios diversos
  - Grãos, temperos, farinha, ...
- Para levar os produtos à feira, você precisa transportá-los até um certo limite de peso que a sua mochila aguenta
- Cada grão tem um preço por unidade de massa.
  - A massa de alguns produtos é mais caros do que outros
- Quais e quanto de cada produto você deve levar à feira de modo a **maximizar o seu faturamento**, caso venda tudo?

Suponha que todos os produtos serão vendidos

Item	***kg***	**$/*kg***
Orégano	10	2
Pimenta do reino	5	4
Milho de pipoca	7	3
Linhaça	20	12
Pó de ouro	8	35
Arroz	12	40

Item	***kg***	**$/*kg***
Orégano	10	2
Pimenta do reino	5	4
Milho de pipoca	7	3
Linhaça	20	12
Pó de ouro	8	35
Arroz	12	40

Se eu vender 10*kg* de orégano eu terei $20 reais de lucro

Item	***kg***	***$/kg***
Orégano	10	2
Pimenta do reino	5	4
Milho de pipoca	7	3
Linhaça	20	12
Pó de ouro	8	35
Arroz	12	40

Cada kg de arroz me fornece $40 de lucro

Item	***kg***	**$/*kg***
Orégano	10	2
Pimenta do reino	5	4
Milho de pipoca	7	3
Linhaça	20	12
Pó de ouro	8	35
Arroz	~~12~~ 12 – 4 = 8	40

Eu posso fracionar o meu produto. Ou seja, eu não sou obrigado a levar um produto em sua totalidade. Por exemplo, eu posso levar na mochila apenas 2*kg* de arroz.

Resolver esse problema é simplesmente responder a pergunta: **Qual é a decisão gulosa?**

Item	***kg***	**$/*kg***
Orégano	10	2
Pimenta do reino	5	4
Milho de pipoca	7	3
Linhaça	20	12
Pó de ouro	8	35
Arroz	12	40

Resolver esse problema é simplesmente responder a pergunta: **Qual é a decisão gulosa?**

Selecionar o item que possui o maior valor por *kg*?

Item	***kg***	**$/*kg***
Orégano	10	2
Pimenta do reino	5	4
Milho de pipoca	7	3
Linhaça	20	12
Pó de ouro	8	35
Arroz	12	40

Considere c a capacidade máxima da sua mochila em kg

1. Ordene os itens por preço do kg
2. Se a quantidade do item (q_i) atual for $q_i < c$
    1. Adicione todo o item na mochila e subtraia q_i de c
3. Senão # fracionar em c unidades
    1. Adicione c unidades de massa (kg) do item à mochila
    2. Subtraia essa quantidade de c, isto é, $c = 0$
4. Se $c > 0$, vá para o próximo item e volte ao passo 2

Considere c a capacidade máxima da sua mochila em kg

1. Ordene os itens por preço do kg
2. Se a quantidade do item (q_i) atual for $q_i < c$
    1. Adicione todo o item na mochila e subtraia q_i de c
3. Senão # fracionar em c unidades
    1. Adicione c unidades de massa (kg) do item à mochila
    2. Subtraia essa quantidade de c, isto é, $c = 0$
4. Se $c > 0$, vá para o próximo item e volte ao passo 2

Considere c a capacidade máxima da sua mochila em kg

1. Ordene os itens por preço do kg
2. Se a quantidade do item (q_i) atual for $q_i < c$
    1. Adicione todo o item na mochila e subtraia q_i de c
3. Senão # fracionar em c unidades
    1. Adicione c unidades de massa (kg) do item à mochila
    2. Subtraia essa quantidade de c, isto é, $c = 0$
4. Se $c > 0$, vá para o próximo item e volte ao passo 2

Considere c a capacidade máxima da sua mochila em kg

1. Ordene os itens por preço do kg
2. Se a quantidade do item (q_i) atual for $q_i < c$
    1. Adicione todo o item na mochila e subtraia q_i de c
3. Senão # fracionar em c unidades
    1. Adicione c unidades de massa (kg) do item à mochila
    2. Subtraia essa quantidade de c, isto é, $c = 0$
4. Se $c > 0$, vá para o próximo item e volte ao passo 2

Considere c a capacidade máxima da sua mochila em kg

1. Ordene os itens por preço do kg
2. Se a quantidade do item (q_i) atual for $q_i < c$
    1. Adicione todo o item na mochila e subtraia q_i de c
3. Senão # fracionar em c unidades
    1. Adicione c unidades de massa (kg) do item à mochila
    2. Subtraia essa quantidade de c, isto é, $c = 0$
4. Se $c > 0$, vá para o próximo item e volte ao passo 2

```
def mochila_fracionaria(c, kg, valor):
    mochila = [0.0 for i in range(len(valor))] # mochila vazia
    for i in range(len(valor)): # percorrendo de forma inversa
        if (kg[i] <= c): # inserir o item inteiro
            mochila[i] = kg[i]
            c -= kg[i]
        else: # fracionar o item
            mochila[i] = c
            c = 0
            break
    return mochila

c = 1550
valor = [10, 20, 20, 30, 40]
kg = [100, 300, 400, 600, 840]
valor, kg = list(zip(*[(x, y) for x, y in sorted(zip(valor, kg))]))
mochila = mochila_fracionaria(c, kg, valor)]
```

```
def mochila_fracionaria(c, kg, valor):
    mochila = [0.0 for i in range(len(valor))] # mo
    for i in range(len(valor)): # percorrendo de fo
        if (kg[i] <= c): # inserir o item inteiro
            mochila[i] = kg[i]
            c -= kg[i]
        else: # fracionar o item
            mochila[i] = c
            c = 0
            break
    return mochila

c = 1550
valor = [10, 20, 20, 30, 40]
kg = [100, 300, 400, 600, 840]
valor, kg = list(zip(*[(x, y) for x, y in sorted(zip(valor, kg))]))
mochila = mochila_fracionaria(c, kg, valor)]
```

kg
1550 – 840 = 710
710 – 600 = 110
400 – 110 = 290
$
40 * 840 + 30*600 + 20 * 110
33.600 + 18000 + 2200
53.800

- Vamos ilustrar o algoritmo
- Considere $c = 21$

Id	**Item**	***kg***	**$/*kg***
1	Orégano	10	2
2	Pimenta do reino	5	4
3	Milho de pipoca	7	3
4	Linhaça	20	12
5	Pó de ouro	8	35
6	Arroz	12	40

- Vamos ilustrar o algoritmo
- Considere $c = 21$

Id	Item	*kg*	$/*kg*
1	Orégano	10	2
2	Pimenta do reino	5	4
3	Milho de pipoca	7	3
4	Linhaça	20	12
5	Pó de ouro	8	35
6	Arroz	12	40

} Pegar tudo

$c = 12$

$itens = \{6\}$

$kg = \{12\}$

$Ganho = (40 \times 12) = 480$

- Vamos ilustrar o algoritmo
- Considere $c = 21$

Id	Item	*kg*	$/*kg*
1	Orégano	10	2
2	Pimenta do reino	5	4
3	Milho de pipoca	7	3
4	Linhaça	20	12
5	Pó de ouro	8	35
6	Arroz	12	40

} Pegar tudo

$c = 20$

$itens = \{6, 5\}$

$kg = \{12, 8\}$

$Ganho = (40 \times 12) + (35 \times 8) = 760$

- Vamos ilustrar o algoritmo
- Considere $c = 21$

Id	Item	*kg*	$/*kg*
1	Orégano	10	2
2	Pimenta do reino	5	4
3	Milho de pipoca	7	3
4	Linhaça	20	12
5	Pó de ouro	8	35
6	Arroz	12	40

Pegar 1 unidade

$$c = 21$$

$$itens = \{6, 5, 4\}$$

$$kg = \{12, 8, 1\}$$

$$Ganho = (40 \times 12) + (35 \times 8) + (1 \times 12) = 772$$

Mochila fracionária
Exemplo

- Vamos ilustrar o algoritmo
- Considere $c = 21$

Id	**Item**	***kg***	**$/*kg***
1	Orégano	10	2
2	Pimenta do reino	5	4
3	Milho de pipoca	7	3
4	Linhaça	20	12
5	Pó de ouro	8	35
6	Arroz	12	40

Restaram

Item	***kg***	**$/*kg***
Orégano	10	2
Pimenta do reino	5	4
Milho de pipoca	7	3
Linhaça	19	12
Pó de ouro	0	35
Arroz	0	40

- E se não pudéssemos fracionar os itens, ainda funcionaria o algoritmo?
- Considere $c = 15$

Id	***kg***	**$/*kg***
1	3	1
2	12	5
3	8	6
4	5	2
5	10	7
6	7	5

- E se não pudéssemos fracionar os itens, ainda funcionaria o algoritmo?
- Considere $c = 15$

Id	***kg***	**$/*kg***
1	3	1
2	12	5
3	8	6
4	5	2
5	10	7
6	7	5

Pegar tudo

$c = 10$

$itens = \{5\}$

$kg = \{10\}$

$Ganho = (10 \times 7) = 70$

- E se não pudéssemos fracionar os itens, ainda funcionaria o algoritmo?
- Considere $c = 15$

Id	***kg***	**$/*kg***
1	3	1
2	12	5
3	8	6
4	5	2
5	10	7
6	7	5

Ultrapassa

- E se não pudéssemos fracionar os itens, ainda funcionaria o algoritmo?
- Considere $c = 15$

Id	***kg***	**$/*kg***
1	3	1
2	12	5
3	8	6
4	5	2
5	10	7
6	7	5

Ultrapassa

- E se não pudéssemos fracionar os itens, ainda funcionaria o algoritmo?
- Considere $c = 15$

Id	***kg***	**$/*kg***
1	3	1
2	12	5
3	8	6
4	5	2
5	10	7
6	7	5

Ultrapassa

- E se não pudéssemos fracionar os itens, ainda funcionaria o algoritmo?
- Considere $c = 15$

Id	*kg*	$/*kg*
1	3	1
2	12	5
3	8	6
4	5	2
5	10	7
6	7	5

Pegar tudo

$$c = 15$$

$$itens = \{5, 4\}$$

$$kg = \{10, 5\}$$

$$Ganho = (10 \times 7) + (5 \times 2) = 80$$

- E se não pudéssemos fracionar os itens, ainda funcionaria o algoritmo?
- Considere $c = 15$

Id	*kg*	$/*kg*
1	3	1
2	12	5
3	8	6
4	5	2
5	10	7
6	7	5

$$c = 15$$

$$itens = \{3, 6\}$$

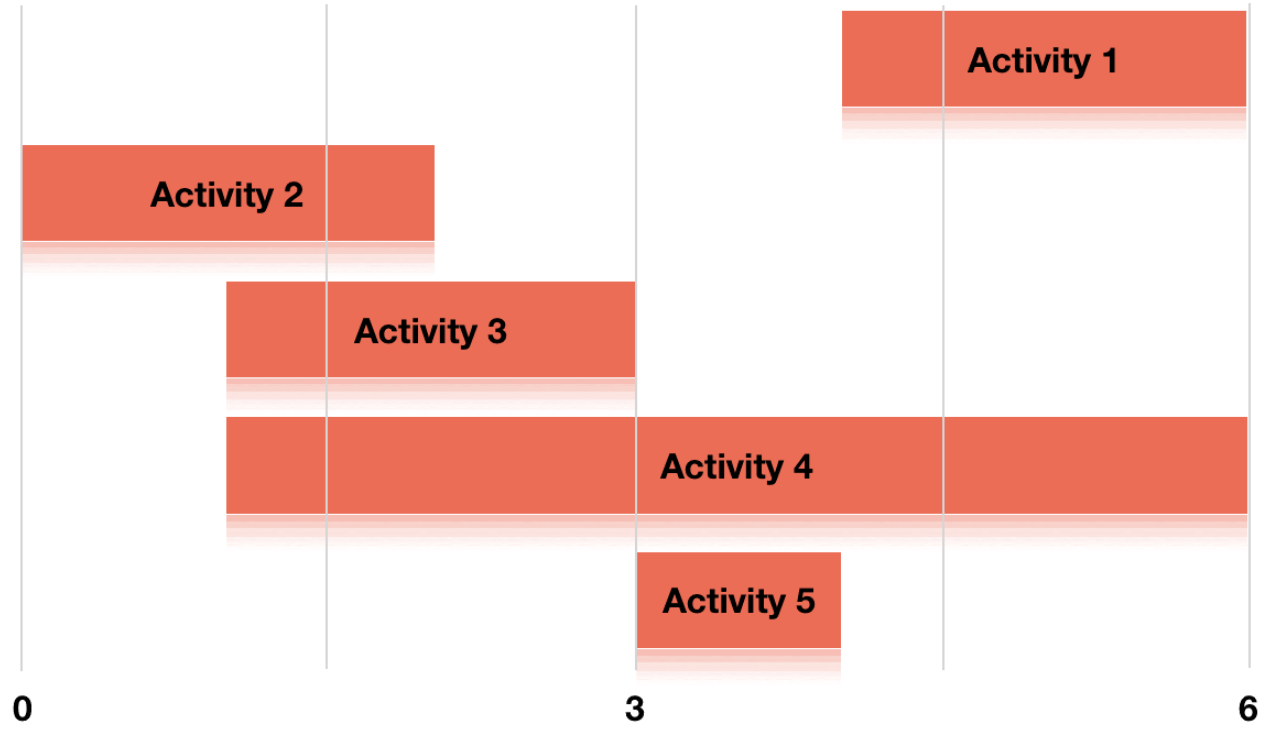
$$kg = \{8, 7\}$$

$$Ganho = (8 \times 6) + (7 \times 5) = 83$$

Melhor solução

Objetivo

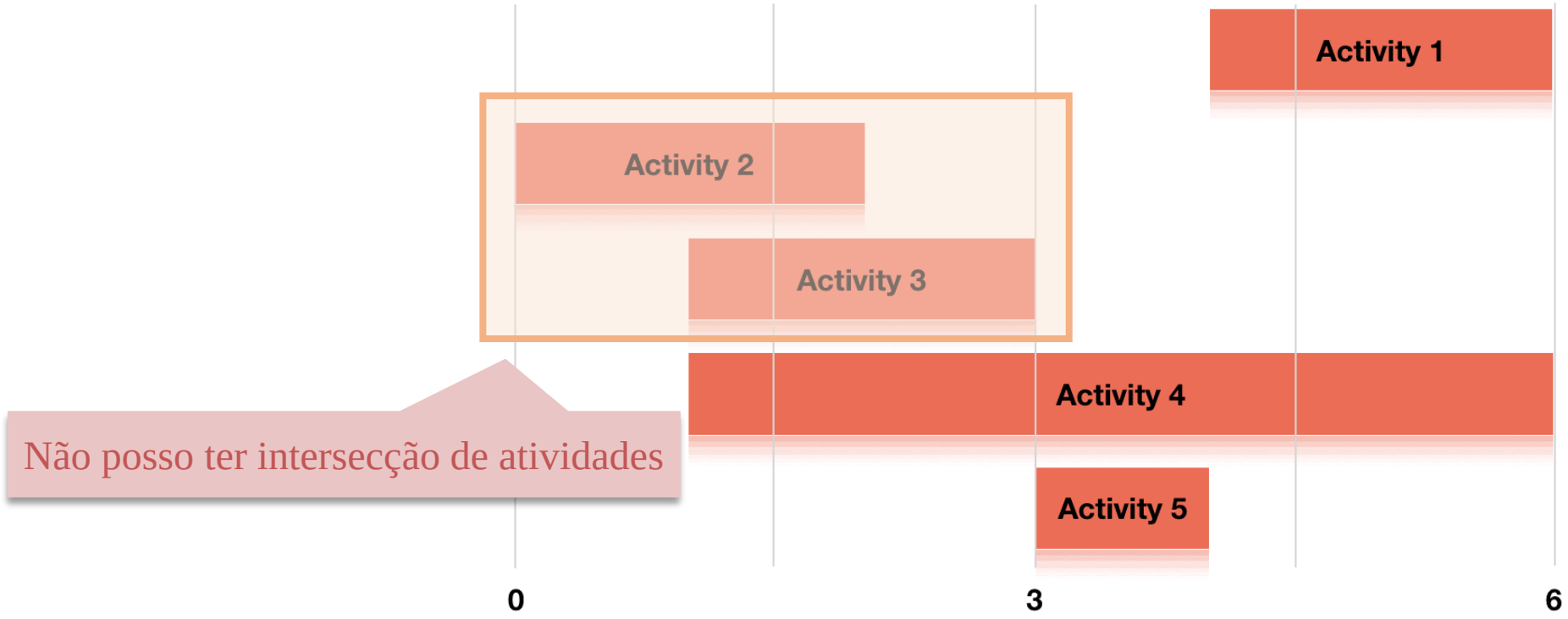
- Selecionar o maior número de atividades que não tenham interseção entre si, dentro de um intervalo de tempo



https://www.codesdope.com/course/algorithms-activity-selection/

Objetivo

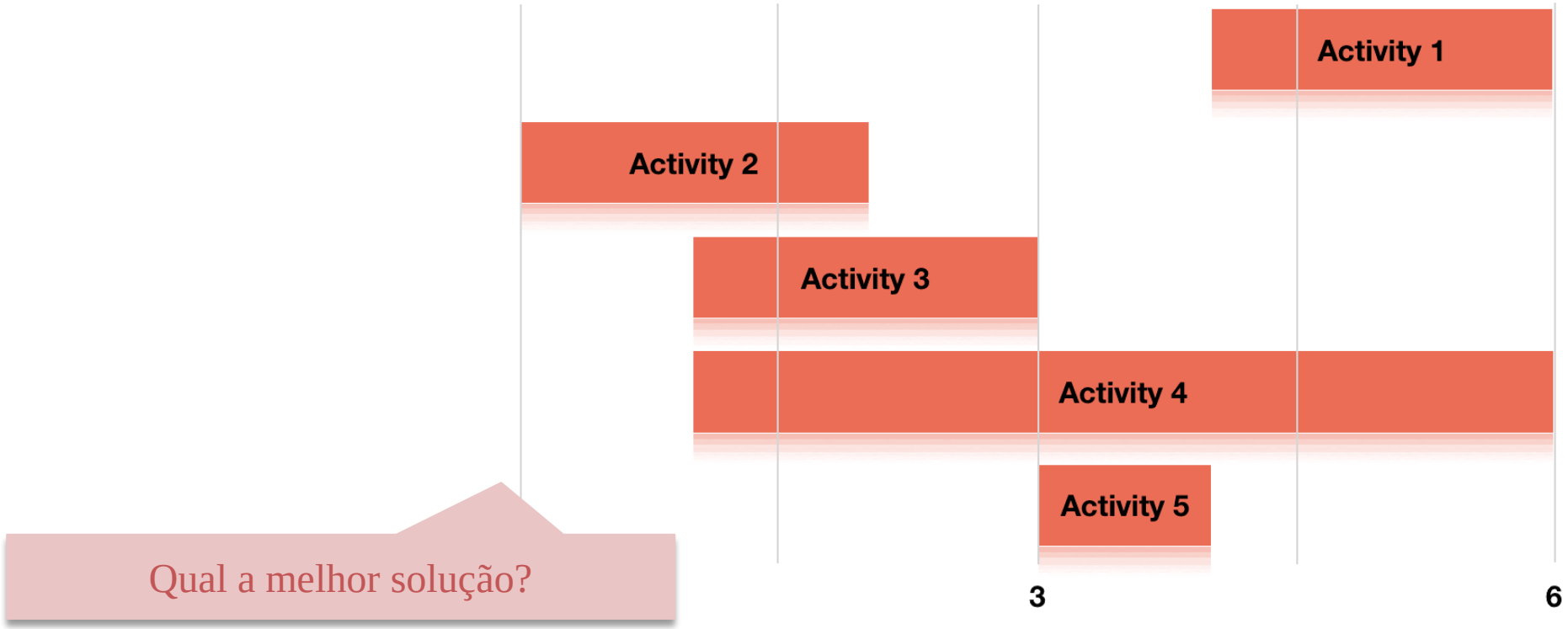
- Selecionar o maior número de atividades que não tenham interseção entre si, dentro de um intervalo de tempo



https://www.codesdope.com/course/algorithms-activity-selection/

Objetivo

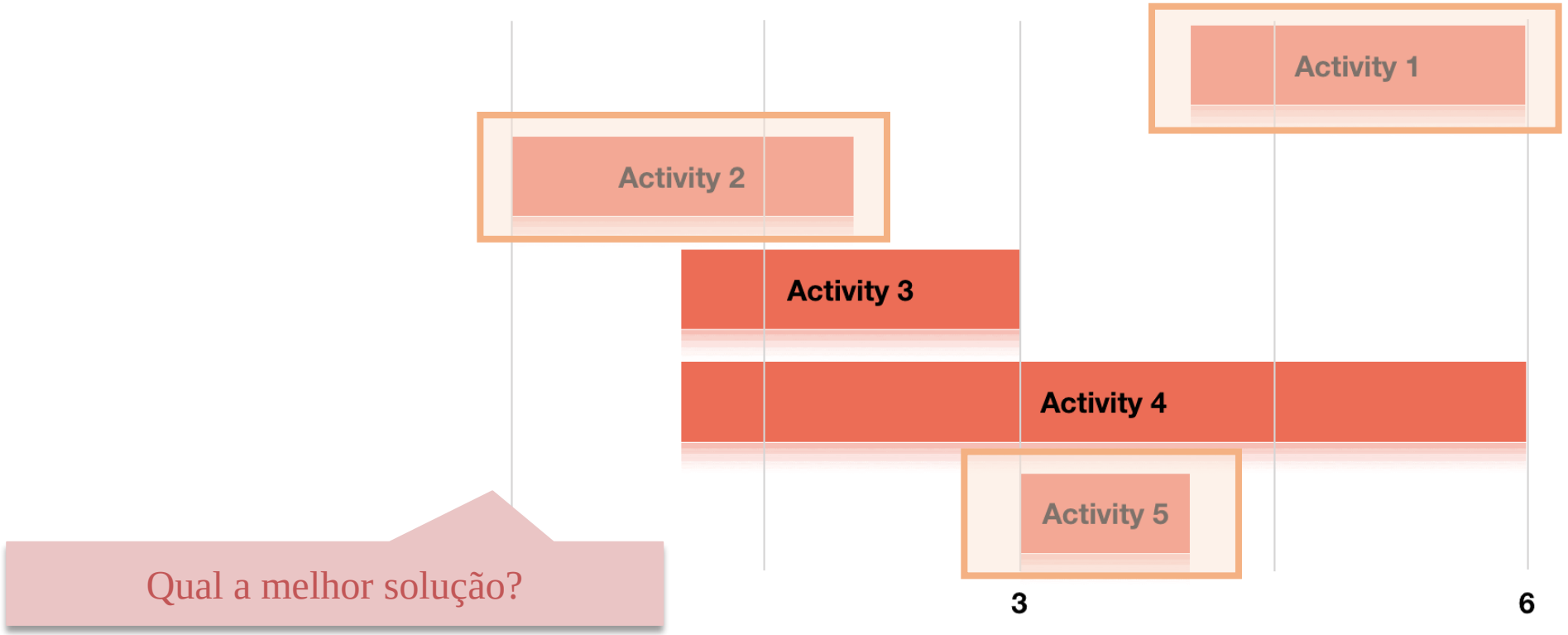
- Selecionar o maior número de atividades que não tenham interseção entre si, dentro de um intervalo de tempo



https://www.codesdope.com/course/algorithms-activity-selection/

Objetivo

- Selecionar o maior número de atividades que não tenham interseção entre si, dentro de um intervalo de tempo



https://www.codesdope.com/course/algorithms-activity-selection/

- Semana da computação

- Semana da computação

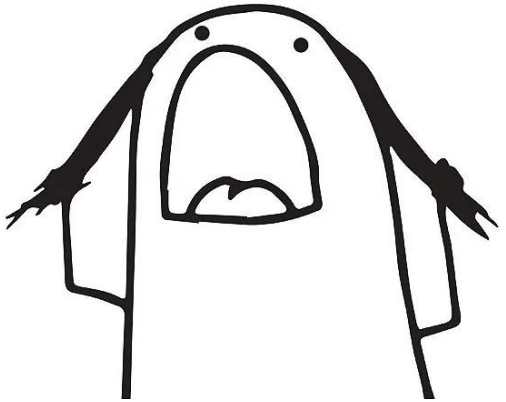
SECOMP
UFSCar

Maximizar o número de certificados ou mini cursos

- Semana da computação

SECOMP
UFSCar

Alguns mini cursos batem o dia/horário



Maximizar o número de certificados ou mini cursos

- Dado um conjunto de atividades, com início e fim definidos

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

Seleção de atividades (a.k.a escalonamento de intervalo)
Ilustrando o problema

- Dado um conjunto de atividades, com início e fim definidos

Não posso ter intersecção

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

- Dado um conjunto de atividades, com início e fim definidos

Qual a decisão gulosa?

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

- Dado um conjunto de atividades, com início e fim definidos

Qual a decisão gulosa?

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

1. Atividades que não tenham nenhum sobreposição

- Dado um conjunto de atividades, com início e fim definidos

Qual a decisão gulosa?

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

1. Atividades que não tenham nenhum sobreposição
2. Atividades que tenham menor início

- Dado um conjunto de atividades, com início e fim definidos

Qual a decisão gulosa?

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

1. Atividades que não tenham nenhum sobreposição
2. Atividades que tenham menor início
3. Atividades menores

- Dado um conjunto de atividades, com início e fim definidos

Qual a decisão gulosa?

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

1. Atividades que não tenham nenhum sobreposição
2. Atividades que tenham menor início
3. Atividades menores

- Eu preciso ordenar?
- Como ordenar? Qual o critério?
- Ordenar ajudar a selecionar as tarefas que não tem intersecção?

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

Atividade	Duração									
	0	1	2	3	4	5	6	7	8	9
1										
2										
3										
4										
5										
6										

Qual atividade foi selecionada primeiro?

Atividade	Início	fim
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

Atividade	Duração									
	0	1	2	3	4	5	6	7	8	9
1										
2										
3										
4										
5										
6										

Algoritmo

1. Inicia o conjunto-resposta *A* vazio
2. Escolha uma atividade *a* de forma gulosa e adicione em *A*
   - Qual é a decisão gulosa?
3. Remova todas as atividades que possuem intersecção com *a*
4. Caso ainda haja atividades, retorne ao passo 2
5. Retorne *A*

Algoritmo

1. Inicia o conjunto-resposta *A* vazio
2. Escolha uma atividade *a* de forma gulosa e adicione em *A*
   - Qual é a decisão gulosa?
3. Remova todas as atividades que possuem intersecção com *a*
4. Caso ainda haja atividades, retorne ao passo 2
5. Retorne *A*

Algoritmo

1. Inicia o conjunto-resposta *A* vazio
2. Escolha uma atividade *a* de forma gulosa e adicione em *A*
   - Qual é a decisão gulosa?
3. Remova todas as atividades que possuem intersecção com *a*
4. Caso ainda haja atividades, retorne ao passo 2
5. Retorne *A*

Devo ordenar?

Algoritmo

1. Inicia o conjunto-resposta *A* vazio
2. Escolha uma atividade *a* de forma gulosa e adicione em *A*
   - Qual é a decisão gulosa?
3. Remova todas as atividades que possuem intersecção com *a*
4. Caso ainda haja atividades, retorne ao passo 2
5. Retorne *A*

Algoritmo

1. Inicia o conjunto-resposta *A* vazio
2. Escolha uma atividade *a* de forma gulosa e adicione em *A*
   - Qual é a decisão gulosa?
3. Remova todas as atividades que possuem intersecção com *a*
4. Caso ainda haja atividades, retorne ao passo 2
5. Retorne *A*

Algoritmo

1. Inicia o conjunto-resposta *A* vazio
2. Escolha uma atividade *a* de forma gulosa e adicione em *A*
   - Qual é a decisão gulosa?
3. Remova todas as atividades que possuem intersecção com *a*
4. Caso ainda haja atividades, retorne ao passo 2
5. Retorne *A*

Seleção de atividades (a.k.a escalonamento de intervalo)
Exemplo

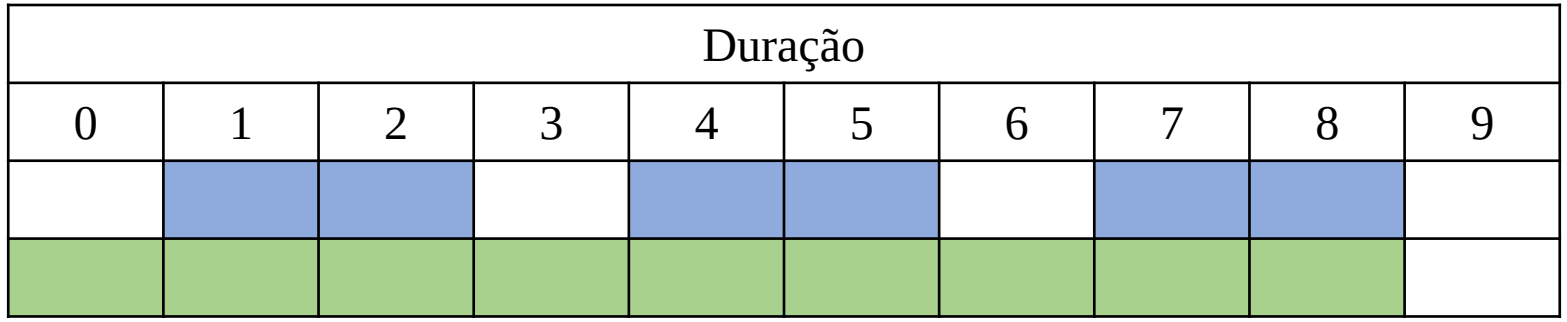
- Decisão gulosa 1
  - Escolher a atividade que começa antes, ou seja, aquela com menor “início”.

- Decisão gulosa 1
  - Escolher a atividade que começa antes, ou seja, aquela com menor “início”.

Duração									
0	1	2	3	4	5	6	7	8	9

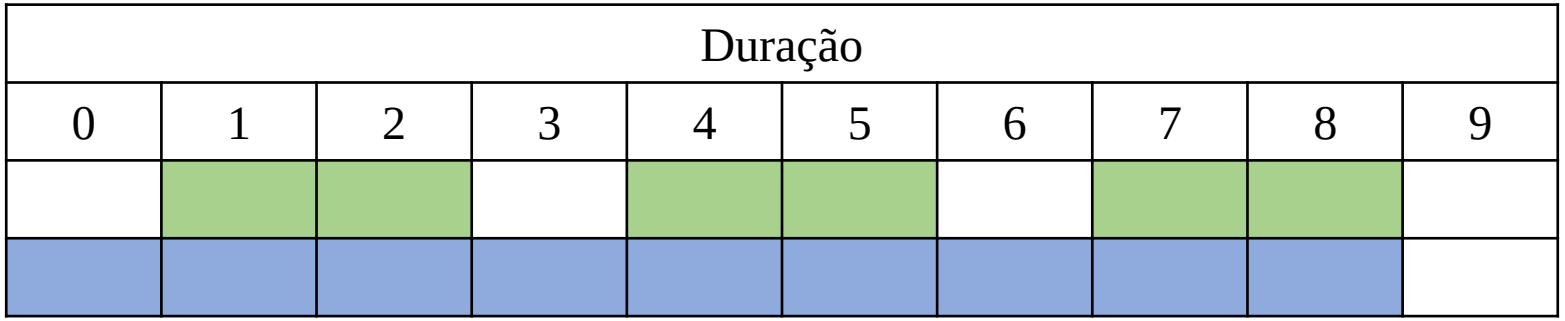
Seleção de atividades (a.k.a escalonamento de intervalo)
Exemplo

- Decisão gulosa 1
  - Escolher a atividade que começa antes, ou seja, aquela com menor “início”.



Resposta do algoritmo

- Decisão gulosa 1
  - Escolher a atividade que começa antes, ou seja, aquela com menor “início”.



Resposta correta

Seleção de atividades (a.k.a escalonamento de intervalo)
Exemplo

- Decisão gulosa 2
  - Escolher a atividade de menor duração, ou seja, aquela com menor “fim-início”.

Seleção de atividades (a.k.a escalonamento de intervalo)
Exemplo

- Decisão gulosa 2
  - Escolher a atividade de menor duração, ou seja, aquela com menor “fim-início”.

Duração									
0	1	2	3	4	5	6	7	8	9

- Decisão gulosa 2
  - Escolher a atividade de menor duração, ou seja, aquela com menor “fim-início”.

Duração									
0	1	2	3	4	5	6	7	8	9

Resposta do algoritmo

- Decisão gulosa 2
  - Escolher a atividade de menor duração, ou seja, aquela com menor “fim-início”.

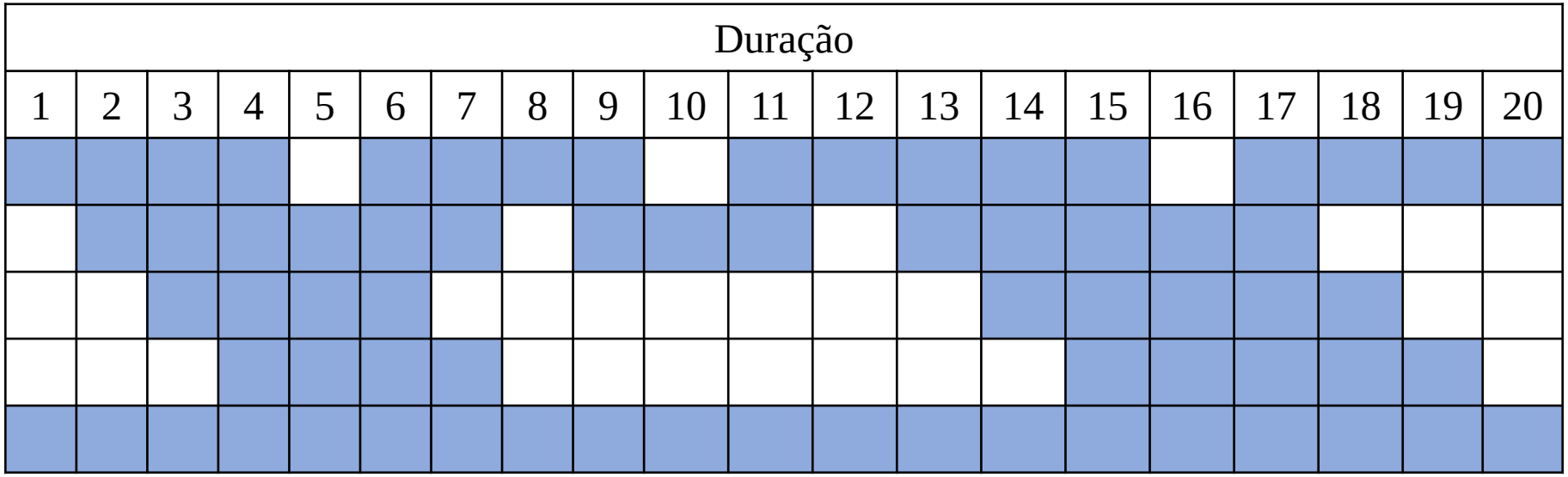
Duração									
0	1	2	3	4	5	6	7	8	9

Resposta correta

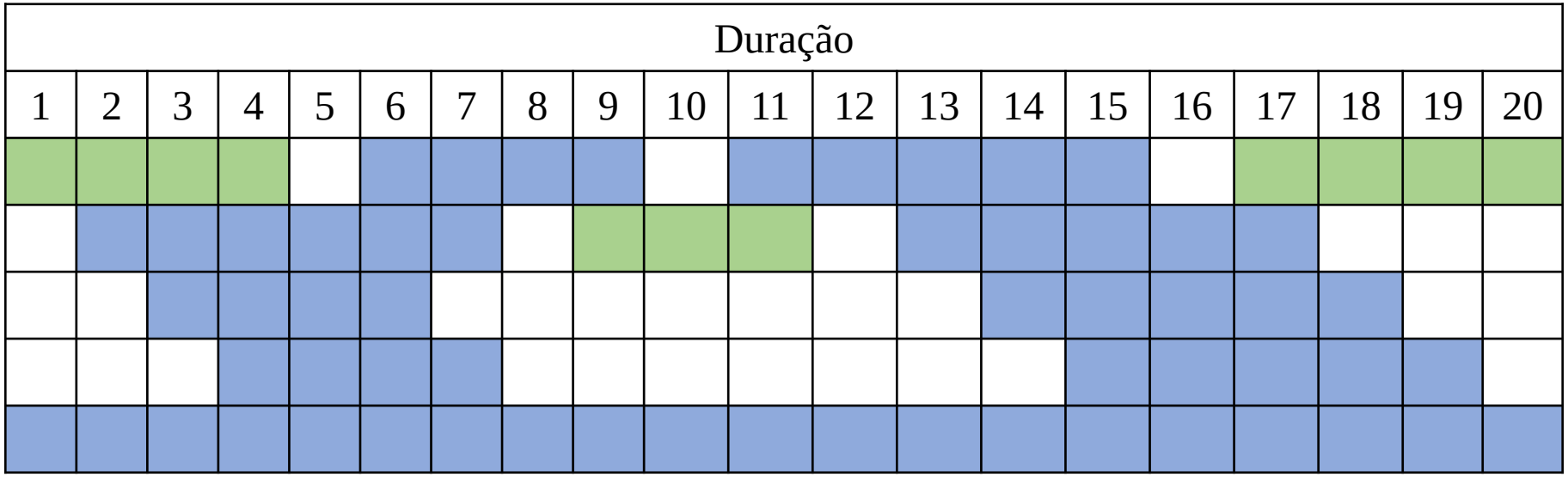
Seleção de atividades (a.k.a escalonamento de intervalo)
Exemplo

- Decisão gulosa 3
  - Escolher a atividade com menor número de intersecções.

- Decisão gulosa 3
  - Escolher a atividade com menor número de intersecções.

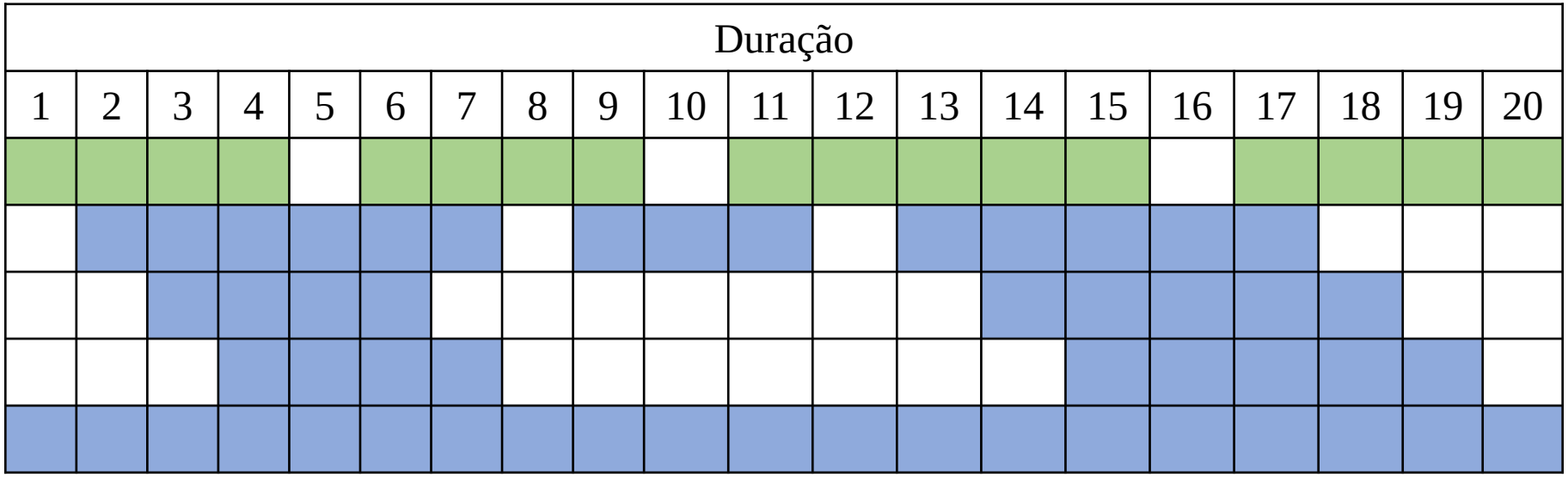


- Decisão gulosa 3
  - Escolher a atividade com menor número de intersecções.



Resposta do algoritmo

- Decisão gulosa 3
  - Escolher a atividade com menor número de intersecções.

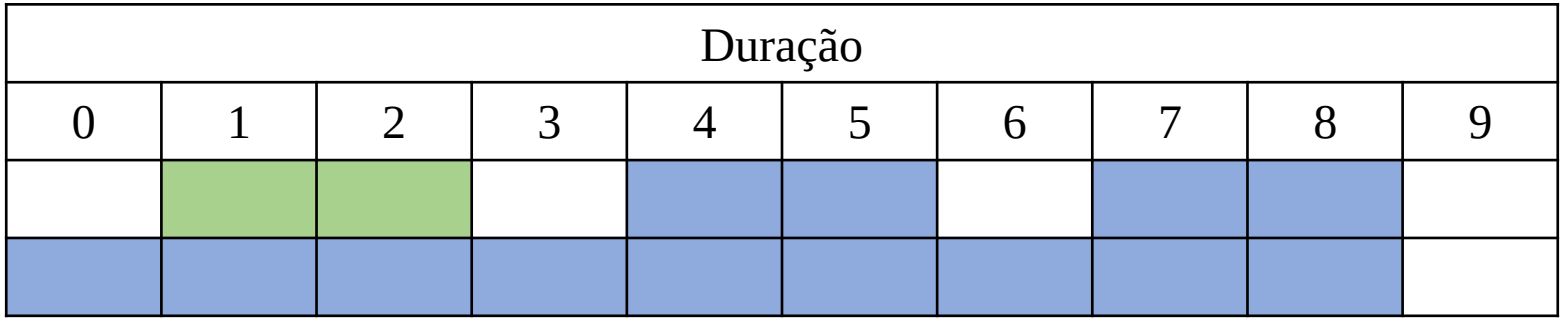


Resposta correta

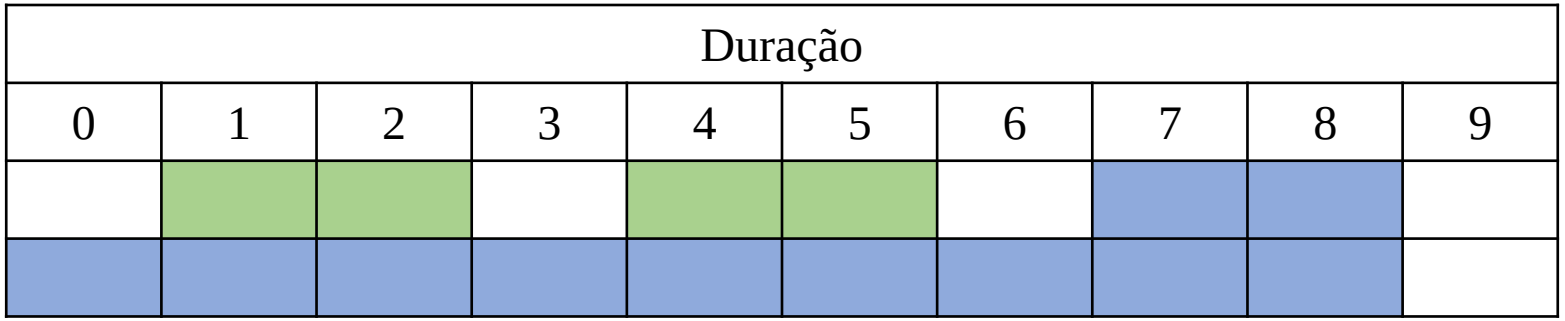
- Decisão gulosa 4
  - Escolher a atividade que termina antes.

Duração									
0	1	2	3	4	5	6	7	8	9

- Decisão gulosa 4
  - Escolher a atividade que termina antes.



- Decisão gulosa 4
  - Escolher a atividade que termina antes.



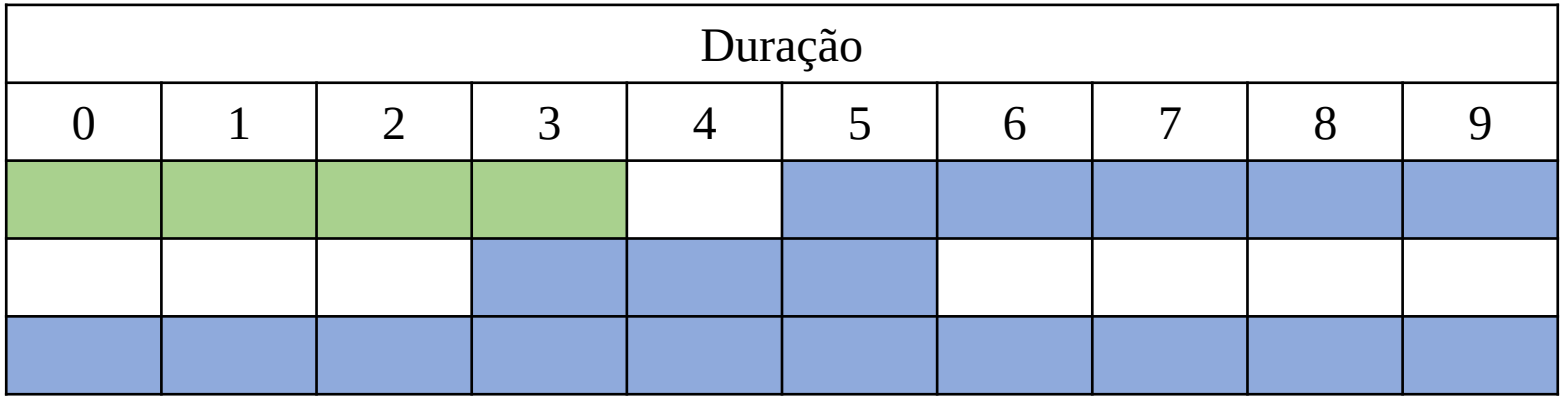
- Decisão gulosa 4
  - Escolher a atividade que termina antes.

Duração									
0	1	2	3	4	5	6	7	8	9

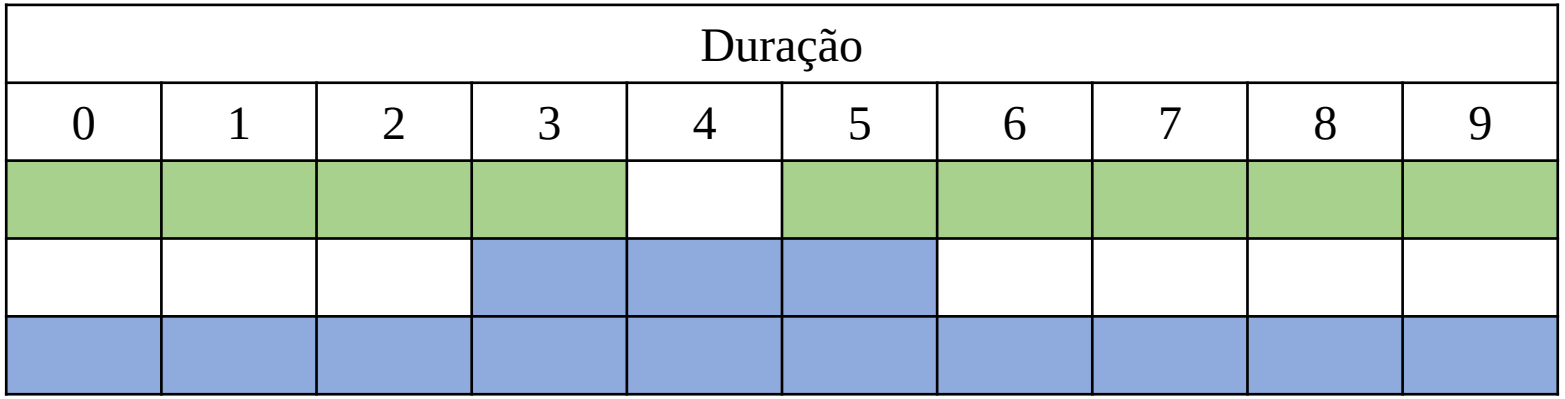
- Decisão gulosa 4
  - Escolher a atividade que termina antes.

Duração									
0	1	2	3	4	5	6	7	8	9

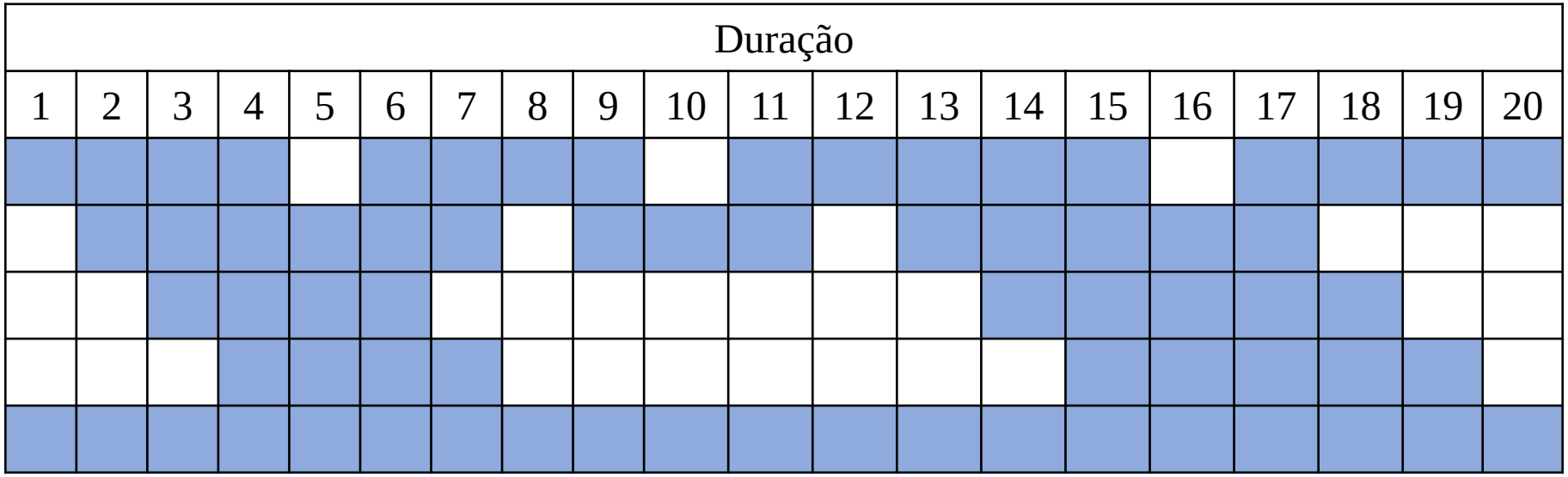
- Decisão gulosa 4
  - Escolher a atividade que termina antes.



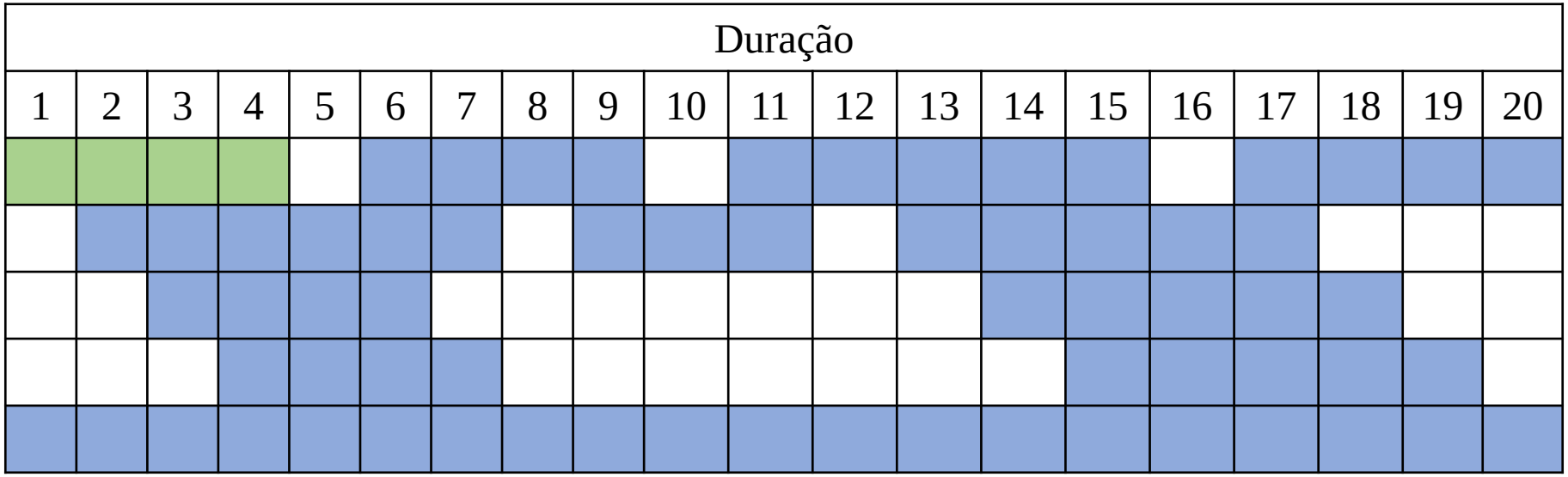
- Decisão gulosa 4
  - Escolher a atividade que termina antes.



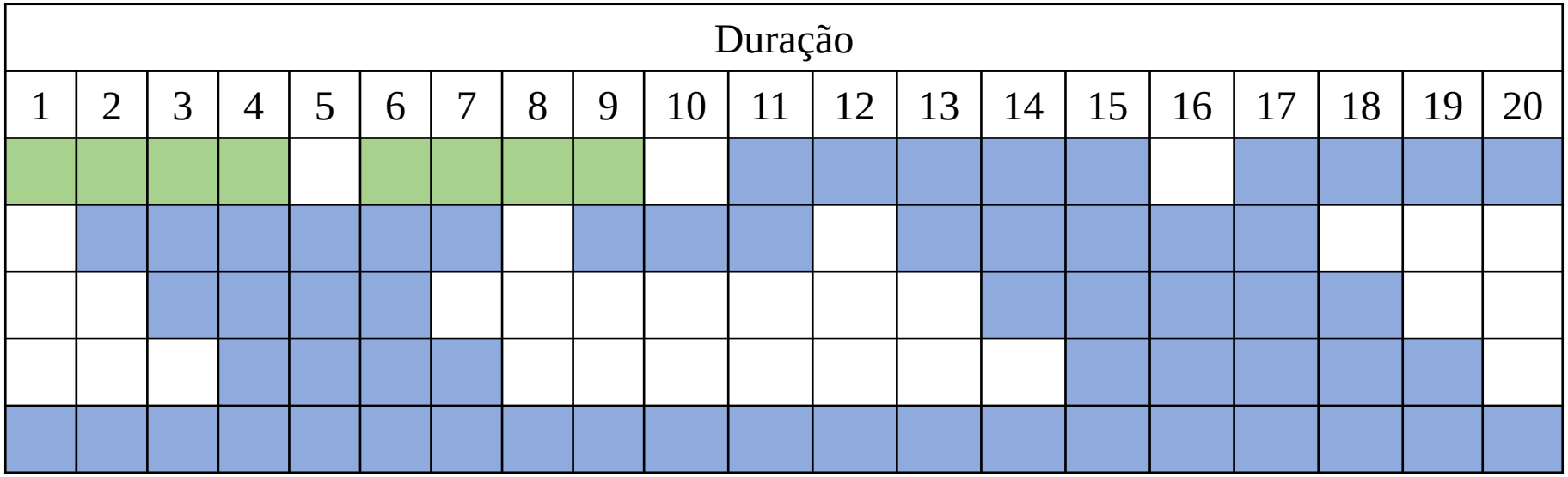
- Decisão gulosa 4
  - Escolher a atividade que termina antes.



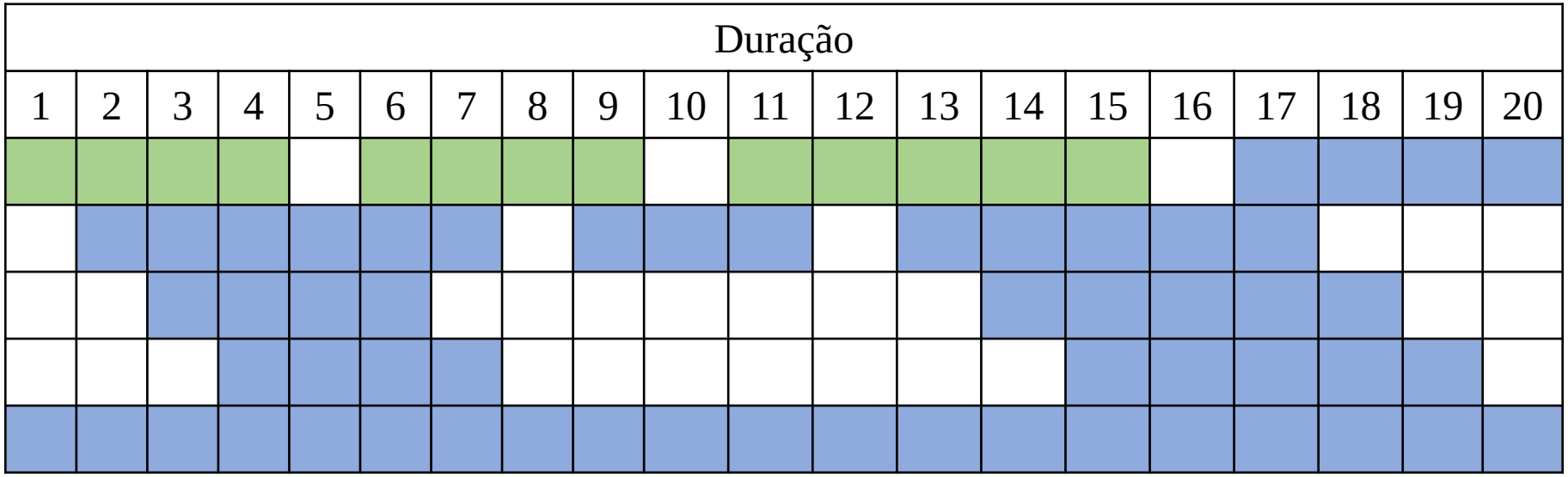
- Decisão gulosa 4
  - Escolher a atividade que termina antes.



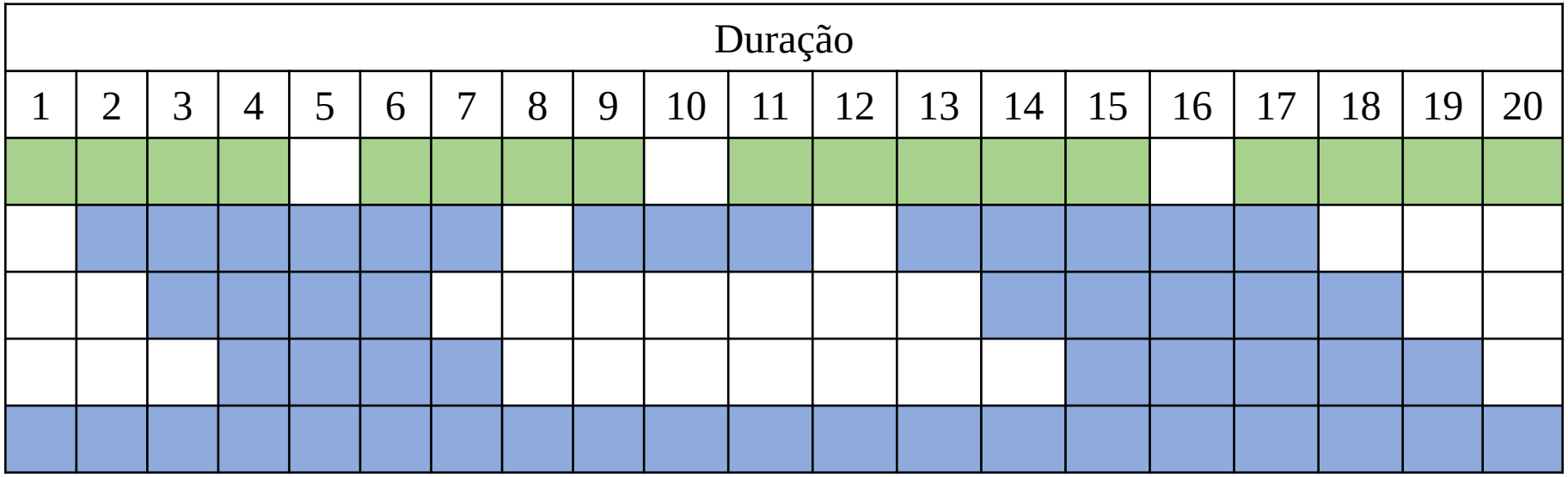
- Decisão gulosa 4
  - Escolher a atividade que termina antes.



- Decisão gulosa 4
  - Escolher a atividade que termina antes.



- Decisão gulosa 4
  - Escolher a atividade que termina antes.



```
def selecao_de_atividades(atividades, inicio, fim):
    n = len(atividades)
    selecao = []
    selecao.append(atividades[0])
    k = 0
    for i in range(1, n):
        # A próxima atividade i tem intersecção
        # com a atividade anterior k?
        if(inicio[i] >= fim[k]):
            selecao.append(atividades[i])
            k = i
    return selecao

atividades = [5, 2, 1, 4, 3]
inicio     = [3, 0, 4, 1, 1]
fim        = [4, 2, 6, 6, 3]
fim, inicio, atividades = zip(*sorted(zip(fim, inicio, atividades)))
selecao = selecao_de_atividades(atividades, inicio, fim)
```

```
def selecao_de_atividades(atividades, inicio, fim):
    n = len(atividades)
    selecao = []
    selecao.append(atividades[0])
    k = 0
    for i in range(1, n):
        # A próxima atividade i tem intersecção
        # com a atividade anterior k?
        if(inicio[i] >= fim[k]):
            selecao.append(atividades[i])
            k = i
    return selecao

atividades = [5, 2, 1, 4, 3]
inicio     = [3, 0, 4, 1, 1]
fim        = [4, 2, 6, 6, 3]
fim, inicio, atividades = zip(*sorted(zip(fim, inicio, atividades)))
selecao = selecao_de_atividades(atividades, inicio, fim)
```

$O(n)$

$O(n \log_2 n)$

Próxima aula

- Algoritmos em Grafos

Obrigado

Dúvidas

Email: alanvalejo@ufscar.br

Acessar o fórum no Moodle