# Projeto e Análise de Algoritmos
# Aula 09 – Algoritmos Gulosos

Prof. Napoleão Nepomuceno

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

# Objetivos

- Apresentar o princípio por trás de Algoritmos Gulosos;

- Apresentar exemplos de Algoritmos Gulosos.

# Algoritmos Gulosos

- Em geral, algoritmos para Problemas de Otimização:
  - Funcionam através de uma sequência de passos;
  - Em cada passo, uma escolha é feita.

- Para muitos problemas de otimização:
  - Usar programação dinâmica para descobrir a melhor escolha pode ser um exagero;
  - Algoritmos mais simples e mais eficientes podem dar conta do recado

# Algoritmos Gulosos

- Um algoritmo guloso:
  - Faz a escolha que parece melhor no momento.

- Ou seja:
  - Faz uma escolha ótima para as condições locais;
  - Espera que essa escolha leve a uma solução ótima para a situação global.

# Problema do Troco

- Suponha que tenhamos os seguintes valores de moedas:
  - 25
  - 10
  - 5
  - 1
- Problema do troco:
  - Dado um determinado valor *X* a ser dado como troco;
  - Determinar a menor quantidade de moedas que juntas somam *X*.

# Problema do Troco

- Para dar um troco de 92, uma solução viável seria:
  - 2 moedas de 25
  - 2 moedas de 10
  - 3 moedas de 5
  - 7 moedas de 1

- Valor dessa solução seria de 14 moedas.

- Essa solução é ótima?

# Problema do Troco

- Para dar um troco de 92, a solução ótima seria:
  - 3 moedas de 25
  - 1 moedas de 10
  - 1 moedas de 5
  - 2 moedas de 1

- Valor dessa solução seria de 7 moedas.

- Como gerar uma solução ótima?

# Problema do Troco

- Geração de uma solução ótima:

  - A cada passo, incluir a moeda de maior valor possível, de forma a não ultrapassar a quantia do troco.

# Problema de Seleção de Atividades

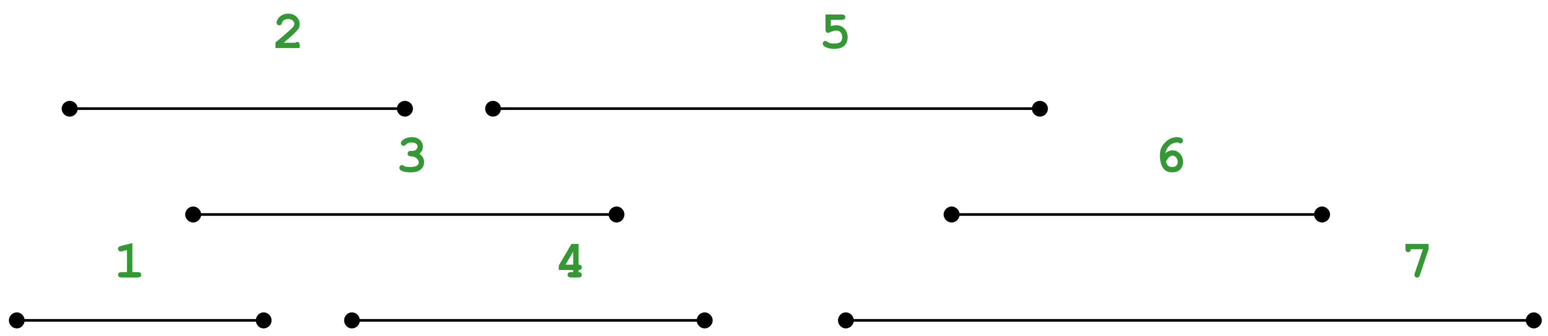
- Problema de programar um recurso entre diversas atividades concorrentes:

  - Suponha que você precise agendar atividades numa dada sala de conferências;
  - A sala só pode receber uma atividade de cada vez;
  - Existem diversas atividades a serem realizadas, cujos tempos de início e término são dados;
  - Selecionar um subconjunto de tamanho máximo de atividades mutuamente compatíveis.

# Problema de Seleção de Atividades

- Entrada: Conjunto de $n$ atividades $S=\{a_1, a_2, ..., a_n\}$:
  - $s_i$ = tempo de início da atividade $a_i$
  - $f_i$ = tempo de término da atividade $a_i$
- Saída: Conjunto $A$ com o número máximo de atividades compatíveis:
  - Duas atividades $a_i$ e $a_j$ são compatíveis se seus intervalos não se sobrepõem.

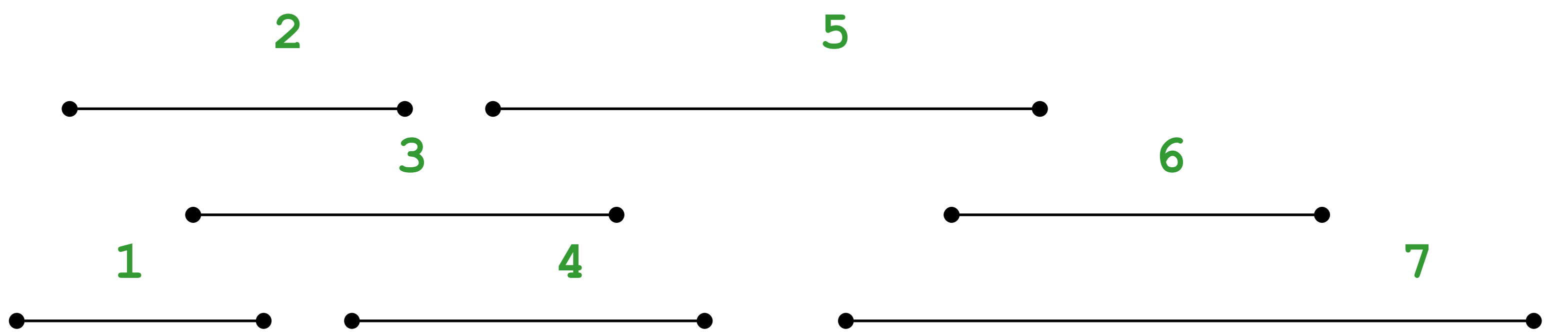
No exemplo, atividades em cada linha são compatíveis.



# Problema de Seleção de Atividades

- Duas atividades $a_i$ e $a_j$ são compatíveis se seus intervalos, $[s_i, f_i)$ e $[s_j, f_j)$, não se sobrepõem
  - $s_i \geq f_j$ = atividade $a_i$ inicia depois que $a_j$ *termina*
  - $s_j \geq f_i$ = atividade $a_j$ inicia depois que $a_i$ *termina*

No exemplo, atividades em cada linha são compatíveis.



# Problema de Seleção de Atividades

- Assuma que as atividades estão ordenadas por tempo de término: $f_1 \leq f_2 \leq \ldots \leq f_n$.

- Suponha que uma solução ótima inclua a tarefa $a_k$:
  - Isso gera dois subproblemas:
    - Selecionar atividades compatíveis do subconjunto $a_1, \ldots, a_{k-1}$ que terminam antes de $a_k$ começar;
    - Selecionar atividades compatíveis do subconjunto $a_{k+1}, \ldots a_n$ que começam depois que $a_k$ termina.
  - A solução de cada um dos dois subproblema precisa ser ótima

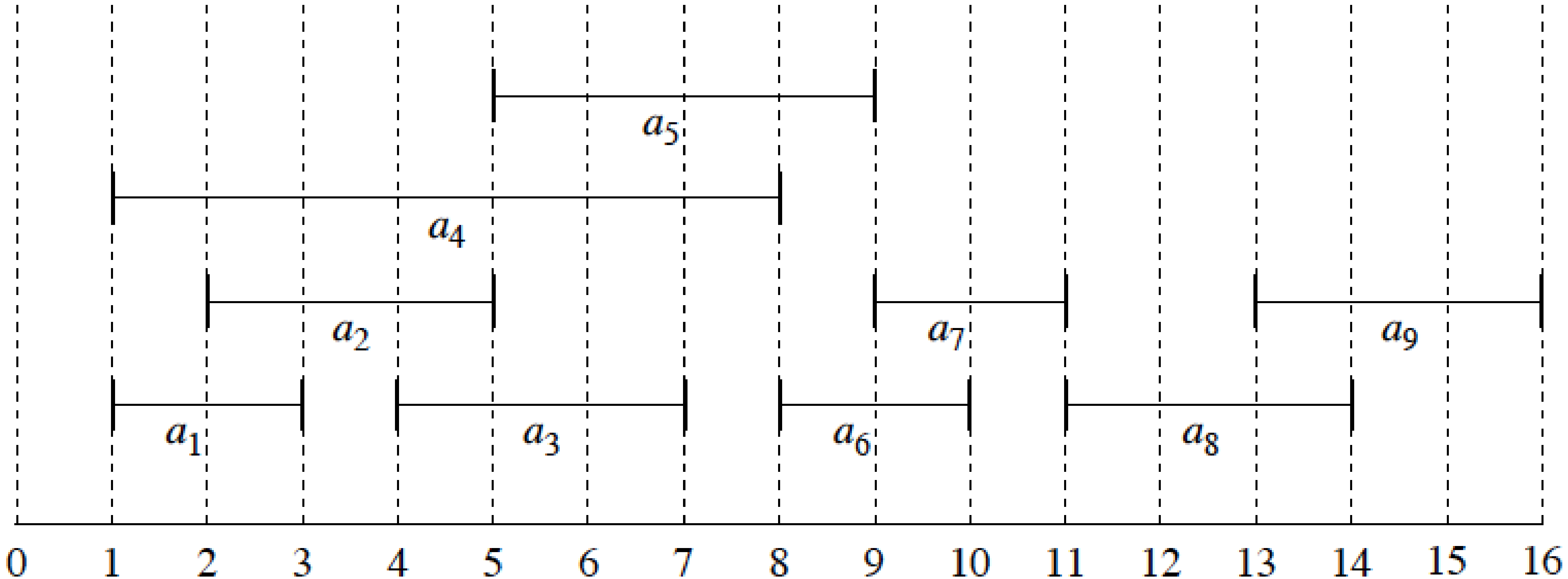
# Problema de Seleção de Atividades

- Primeiro, criamos duas atividades artificiais:
  - $a_0$, com intervalo $[s_0=0, f_0=0)$;
  - $a_{n+1}$, com intervalo $[s_{n+1}=\infty, f_{n+1}=\infty)$.

- Seja $S_{ij}$ o subconjunto de atividades em $S$ que iniciam após $a_i$ terminar e que terminam antes de $a_j$ iniciar;
- Seja $c[i,j]$ = tamanho do maior subconjunto de atividades compatíveis em $S_{ij}$;
- $c[0,n+1]$ = valor da solução ótima.

# Problema de Seleção de Atividades

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 2 | 4 | 1 | 5 | 8 | 9 | 11 | 13 |
| $f_i$ | 3 | 5 | 7 | 8 | 9 | 10 | 11 | 14 | 16 |



# Problema de Seleção de Atividades

- Como computar $c[i,j]$?

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \phi \\ \max\limits_{i<k<j}\{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \phi \end{cases}$$

- Para computar $c[i,j]$ é necessário, Para cada atividade $a_k$, onde $i < k < j$ :
  - Encontrar o valor da melhor solução que inclui $k$;
  - O que gera dois subproblemas: $c[i,k]$ e $c[k,j]$.
  - Em seguida, escolhe-se a solução de maior valor dentre as soluções de cada um dos $a_k$, onde $i < k < j$.

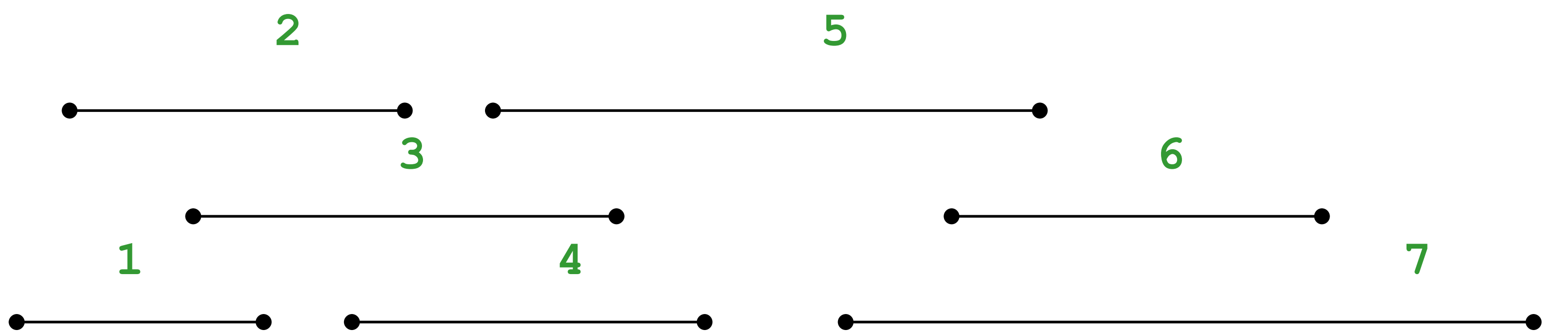
# Problema de Seleção de Atividades

- O problema possui subestrutura ótima e sabemos calcular o valor de uma solução ótima recursivamente ...
  - Poderíamos usar programação dinâmica.

- Mas...
  - O problema também possui propriedade de escolha gulosa;
  - Ou seja, uma solução globalmente ótima é alcançada fazendo-se escolhas localmente ótimas.

# Problema de Seleção de Atividades

- Seja o Problema de Seleção de Atividades do conjunto $S$, ordenado pelo tempo de término.
- Teorema: Existe uma solução ótima $A \subseteq S$ tal que $a_1 \in A$.
- Rascunho da prova:
  - Se existe uma solução ótima $B \subseteq S$ que não inclui $a_1$, podemos sempre substituir a primeira atividade em $B$ por $a_1$.
  - Continuaremos com o mesmo número de atividades e, portanto, a solução continua ótima.



# Problema de Seleção de Atividades

- Se um problema possui a Propriedade de Escolha Gulosa, a resolução é facilitada:
  - Fazer a escolha gulosa antes de solucionar subproblema(s) e avaliar suas soluções;
  - Resolver subproblema que aparece como resultado após a realização da escolha gulosa;
  - Combinar a escolha gulosa com a solução do subproblema.

# Problema de Seleção de Atividades

- Para a Seleção de Atividades:
  1) Ordenar a lista de atividades por tempo de término;
  2) Agendar a primeira atividade;
  3) Agendar a próxima atividade da lista que inicia após o término da última atividade agendada;
  4) Repetir passo 3 até que o final da lista de atividades seja atingido.
- Intuição:
  - Escolher a atividade compatível que mantenha a sala com o maior tempo disponível após sua realização.

# Problema de Seleção de Atividades

```
GREEDY-ACTIVITY-SELECTOR(s, f)
n ← length[s]
A ← {a_1}
i ← 1
for m ← 2 to n
        do if s_m ≥ f_i
                then A ← A ∪ {a_m}
                        i ← m
return A
```