Algoritmos gulosos

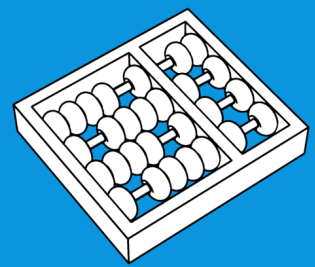
MC458 - Projeto e Análise de Algoritmos I

11/25

24

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

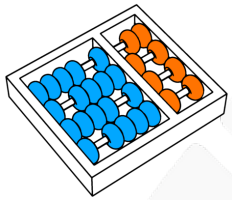
UNICAMP



"Embora a gula seja considerada um dos sete pecados capitais, acontece que algoritmos gulosos frequentemente tem uma performance muito boa."

Stuart Russell.

Algoritmos gulosos



Problemas e subproblemas

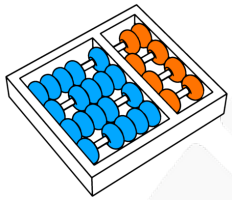
Vamos estudar algoritmos gulosos:

- Decompomos um problema em vários subproblemas.
- De novo, um deles corresponde à **subestrutura ótima**.
- Mas podemos escolher a subestrutura eficientemente.

Comparando as estratégias:

- **Algoritmo de programação dinâmica:**
  1. Primeiro resolvemos todos os subproblemas.
  2. Depois decidimos o subproblema ótimo.

- **Algoritmo guloso:**
  1. Primeiro escolhemos o subproblema ótimo.
  2. Depois resolvemos apenas esse subproblema.



Algoritmos gulosos

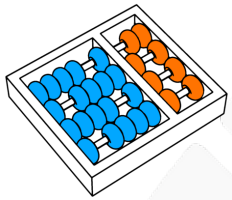
Ideia:

- Realizamos uma sequência de passos.
- A cada passo, fazemos uma escolha.

Premissas dos algoritmos gulosos:

- A escolha é aquela que parece ser a melhor no momento.
- Essa escolha é denominada **escolha gulosa**.
- É feita de acordo com um **critério guloso**.

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim.



Uma receita para algoritmos gulosos

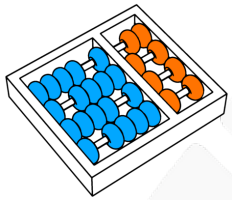
Vários problemas têm a seguinte estrutura:

- Existe um conjunto de elementos E.
- Uma solução é algum **subconjunto** A^* de E.

Um estratégia genérica:

1. Faça $A \leftarrow \emptyset$.
2. Enquanto A não é solução viável:
   (a) Escolha um elemento **e** com algum **critério guloso**.
   (b) Certifique-se de que existe solução A^* contendo $A \cup \{\mathbf{e}\}$.
   (c) Faça $A \leftarrow A \cup \{\mathbf{e}\}$.
3. Devolva A.

Seleção de atividades



Seleção de atividades

Considere n atividades executadas em certo lugar:

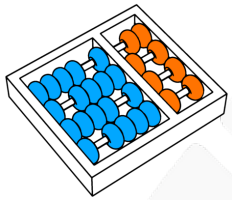
- Podem ser palestras, reuniões em um sala etc.
- Denote essas atividades por $S = \{a_1, \ldots, a_n\}$.

Duração das atividades:

- A atividade a_i começa no tempo s_i e termina no tempo f_i.
- Assim, ela deve ser realizada no intervalo $[s_i, f_i)$.

Definição

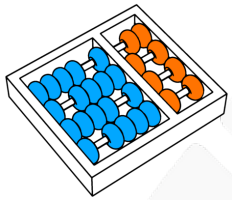
*Duas atividades a_i e a_j são **compatíveis** se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.*



Problema de seleção de atividades

Problema

- ***Entrada:*** *Conjunto de atividades* $S = \{a_1, \ldots, a_n\}$ *e tempos de início* s *e de termino* f*.*
- ***Solução:*** *Subconjunto* A *de atividades compatíveis.*
- ***Objetivo: maximizar*** $|A|$*.*



Uma instância

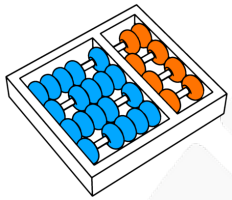
Os tempos de início e de término são:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- As atividades estão em ordem de **tempo de término**.
- Iremos usar essa ordem em seguida.

Exemplos de soluções

- $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis.
- $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas.
- $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e **ótimas**.



Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$:

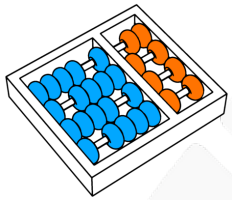
- Ou seja, as atividades estão em ordem de **término**.
- Se não estiverem, podemos ordenar.

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$:

- Atividades que começam depois que a_i termina.
- Além disso, que terminam antes que a_j inicie.

Considere atividades dummies:

- Atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$.
- Assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$.
- O conjunto de todas atividades é $S = S_{0,n+1}$.



Subestrutura ótima

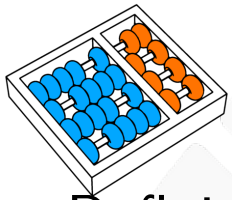
Considere uma solução ótima para as atividades em S_{ij}:

- Suponha que a_k esteja nessa solução ótima.
- As demais atividades da solução devem estar:
  - **antes** do início de a_k.
  - **depois** do término de a_k.

Descobrimos uma **subestrutura ótima**:

- Queremos uma solução ótima para o conjunto S_{ik}.
- Além de uma solução ótima para o conjunto S_{kj}.

Mas **NÃO** sabemos qual é a atividade a_k na solução ótima!



Usando programação dinâmica

Definimos o seguinte **subproblema**:

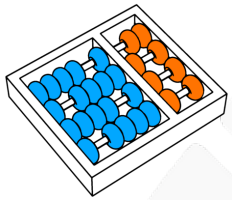
- Considere um par (i, j) para $0 \leq i, j \leq n + 1$.
- Defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}.

Podemos computar $c[i, j]$ com a recorrência:

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original:

- O valor ótimo corresponde à entrada $c[0, n + 1]$.
- É fácil calcular usando **programação dinâmica** (exercício).



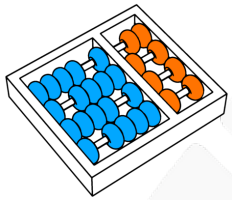
Simplificando

Algumas observações:

- Na programação dinâmica, testamos **todas** as atividades a_k.
- Em um algoritmo guloso, queremos escolher apenas **uma**.

Como escolher de forma gulosa?

- Queremos uma atividade que consome menos "recursos".
- As atividades estão ordenadas por tempo de término.
- a_1 é a escolha que deixa mais tempo para outras atividades.



Escolha gulosa

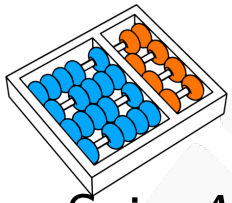
Depois de escolher a_1, qual subproblema resta?

- Queremos atividades que começam depois que a_1 termina.
- Definimos um novo subproblema $S_k = \{a_i \in S : s_i \geq f_k\}$.
- Pela definição da dummy a_0, o problema original é $S_0 = S$.

Teorema (Escolha gulosa)

Considere um subproblema S_k não vazio e seja a_m uma atividade em S_k com o ***menor tempo de término****.*

Então existe uma solução ótima para S_k que contém a_m.



Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k:

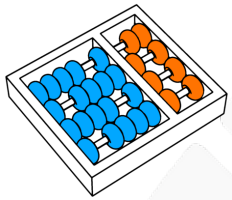
- Se $a_m \in A^*$, então nada há a fazer.
- Então suponha que $a_m \notin A^*$.

Vamos criar **outra solução ótima** A' contendo a_m:

- Seja $a_i \in A^*$ a atividade com menor f_k.
- Defina $A' = A \setminus \{a_i\} \cup \{a_m\}$.

Temos que provar que A' é solução ótima:

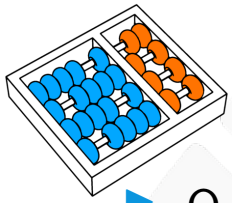
- É claro que $|A^*| = |A'|$.
- Então resta mostrar que A' é viável.
- Nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i.
- Daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m.
- Assim as atividade de A' são mutualmente compatíveis.



Resolvendo recursivamente

A discussão anterior sugere um algoritmo recursivo:

- Suponha que estamos tentando resolver S_k.
- Seja a_m a atividade em S_k com o menor tempo de término.
- Resolva o subproblema S_m e junte com a_m.



Algoritmo recursivo

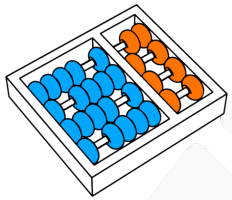
- O vetor f está em ordem de tempo de término.
- Queremos atividades que começam **depois** que k termina.

Algoritmo: SELEÇÃO-ATIVIDADES-REC(s, f, k, n)

```
1 ▷ acha atividade a_m em S_k que termina primeiro
2 m ← k + 1
3 enquanto m ≤ n e s_m < f_k
4     m ← m + 1
5 ▷ escolhe a_m e resolve subproblema S_m
6 se m ≤ n
7     devolva {a_m} ∪ SELEÇÃO-ATIVIDADES-REC(s, f, m, n)
8 senão
9     devolva ∅
```

Análise:

- Vemos cada elemento apenas uma vez, daí o tempo é $\Theta(n)$.
- Pode ser que precisemos ordenar as atividades.



Algoritmo iterativo

Podemos reescrever de maneira iterativa:

Algoritmo: SELEÇÃO-ATIVIDADES-ITER(s, f, n)

```
1 A ← {a_1}
2 k ← 1
3 para m ← 2 até n
4     se s_m ≥ f_k
5         A ← A ∪ {a_m}
6         k ← m
7 devolva A
```

Algoritmos gulosos

MC458 - Projeto e Análise de Algoritmos I

11/25

24

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP