Reduções

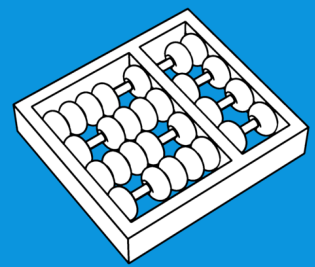
MC558 - Projeto e Análise de Algoritmos II

10/24

17

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

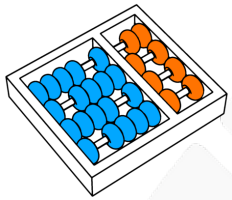
UNICAMP



“Nenhum problema pode ser resolvido até que seja reduzido a uma forma simples.”

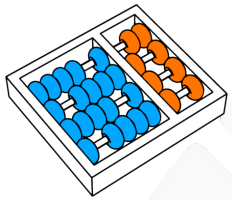
John Pierpont Morgan

Cotas para um problema



Formalizando problema

Como formalizar um problema **GENERICAMENTE**?

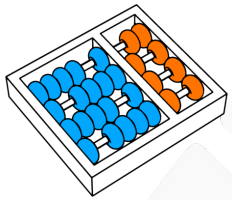


Formalizando problema

Como formalizar um problema **GENERICAMENTE**?

Definição (Problema computacional)

Um problema computacional é uma ***RELAÇÃO*** $P \subseteq \mathcal{I} \times \mathcal{S}$, *onde:*



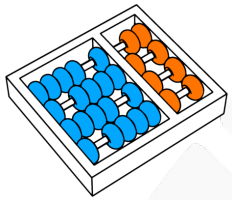
Formalizando problema

Como formalizar um problema **GENERICAMENTE**?

Definição (Problema computacional)

*Um problema computacional é uma **RELAÇÃO** $P \subseteq \mathcal{I} \times \mathcal{S}$, onde:*

- $\mathcal{I}$ *é o conjunto de entradas e*



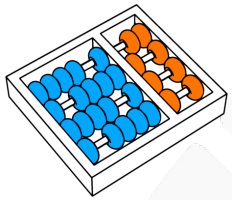
Formalizando problema

Como formalizar um problema **GENERICAMENTE**?

Definição (Problema computacional)

*Um problema computacional é uma **RELAÇÃO** $P \subseteq \mathcal{I} \times \mathcal{S}$, onde:*

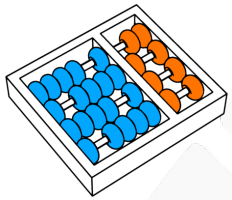
- $\mathcal{I}$ *é o conjunto de entradas e*
- $\mathcal{S}$ *é o conjunto de saídas.*



Algoritmo para um problema

Definição

Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

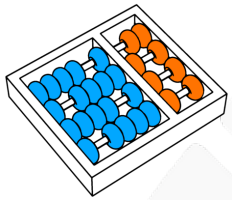


Algoritmo para um problema

Definição

Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

- Escrevemos $I \in P$ para representar uma entrada.

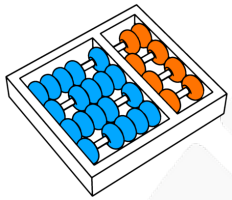


Algoritmo para um problema

Definição

Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

- Escrevemos $I \in P$ para representar uma entrada.
- Escrevemos $A(I)$ para representar a saída do algoritmo.

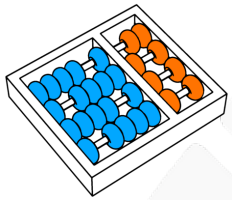


Algoritmo para um problema

Definição

Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

- Escrevemos $I \in P$ para representar uma entrada.
- Escrevemos $A(I)$ para representar a saída do algoritmo.
- Denotamos por n o "tamanho" de I.

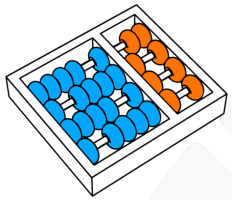


Algoritmo para um problema

Definição

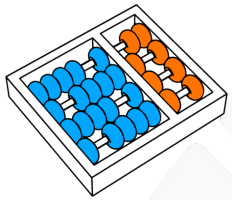
Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

- Escrevemos $I \in P$ para representar uma entrada.
- Escrevemos $A(I)$ para representar a saída do algoritmo.
- Denotamos por n o "tamanho" de I.
- Normalmente n é o números de bits de I.



Revisitando a complexidade de um algoritmo

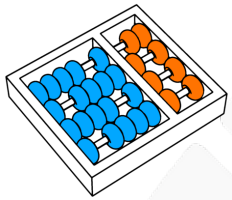
Seja $\textsc{Alg}$ um algoritmo para um problema P e n um parâmetro.



Revisitando a complexidade de um algoritmo

Seja ALG um algoritmo para um problema P e n um parâmetro.

Notação O:

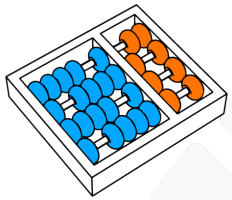


Revisitando a complexidade de um algoritmo

Seja ALG um algoritmo para um problema P e n um parâmetro.

Notação O:

- Se o algoritmo leva tempo **NO MÁXIMO** $f(n)$ para toda entrada de tamanho n, então dizemos que ALG executa em tempo $O(f(n))$.



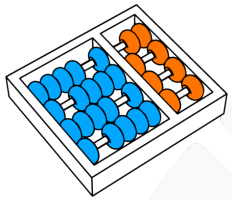
Revisitando a complexidade de um algoritmo

Seja $\textsc{Alg}$ um algoritmo para um problema P e n um parâmetro.

Notação O:

- Se o algoritmo leva tempo **NO MÁXIMO** $f(n)$ para toda entrada de tamanho n, então dizemos que $\textsc{Alg}$ executa em tempo $O(f(n))$.

Notação Ω:



Revisitando a complexidade de um algoritmo

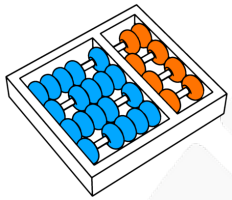
Seja $\textsc{Alg}$ um algoritmo para um problema P e n um parâmetro.

Notação O:

- Se o algoritmo leva tempo **NO MÁXIMO** $f(n)$ para toda entrada de tamanho n, então dizemos que $\textsc{Alg}$ executa em tempo $O(f(n))$.

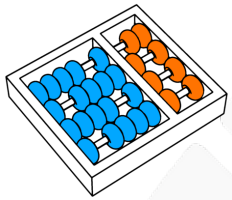
Notação Ω:

- Se o algoritmo leva tempo **PELO MENOS** $g(n)$ para alguma entrada de tamanho n, então dizemos que $\textsc{Alg}$ executa em tempo $\Omega(g(n))$.



Cotas superior e inferior de um problema

Seja P um problema e seja n um parâmetro:

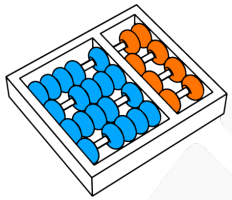


Cotas superior e inferior de um problema

Seja P um problema e seja n um parâmetro:

Definição (Cota superior)

Uma função $f(n)$ é chamada de cota superior para P se ***existe algum algoritmo*** *que resolve P em tempo $O(f(n))$.*



Cotas superior e inferior de um problema

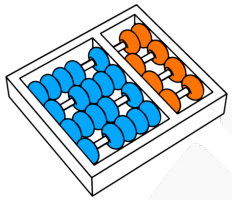
Seja P um problema e seja n um parâmetro:

Definição (Cota superior)

Uma função $f(n)$ é chamada de cota superior para P se ***existe algum algoritmo*** *que resolve P em tempo $O(f(n))$.*

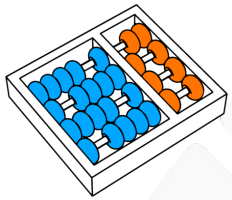
Definição (Cota inferior)

Uma função $g(n)$ é chamada de cota inferior para P se ***todo algoritmo*** *que resolve P leva tempo $\Omega(f(n))$.*



Algoritmo ótimo

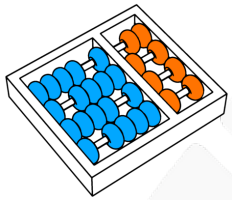
Um algoritmo ALG é **ÓTIMO** para um problema P se:



Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

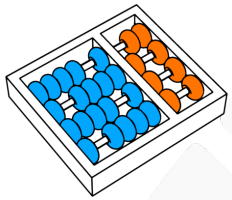
1. ALG resolve P em tempo $O(f(n))$ e



Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

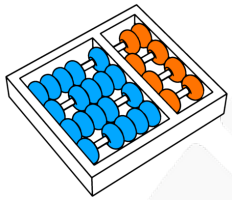


Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:

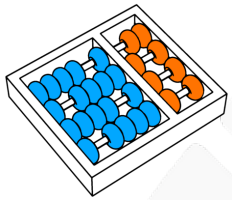


Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.

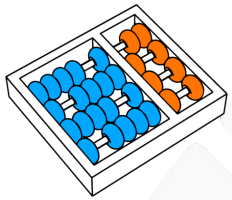


Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.
  - Ordenação tem cota inferior $\Omega(n \log n)$.

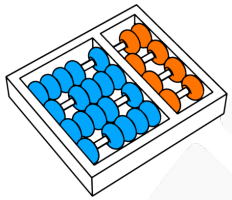


Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.
  - Ordenação tem cota inferior $\Omega(n \log n)$.
- BUSCA-BINARIA é ótimo para busca em vetor ordenado:



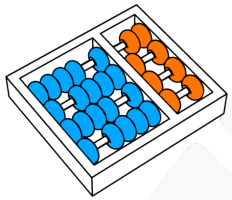
Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.
  - Ordenação tem cota inferior $\Omega(n \log n)$.

- BUSCA-BINARIA é ótimo para busca em vetor ordenado:
  - Tem complexidade $O(\log n)$.



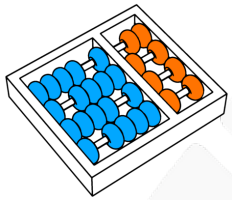
Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

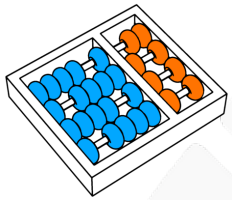
- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.
  - Ordenação tem cota inferior $\Omega(n \log n)$.

- BUSCA-BINARIA é ótimo para busca em vetor ordenado:
  - Tem complexidade $O(\log n)$.
  - Qualquer algoritmo leva tempo $\Omega(\log n)$.



Comparando problemas

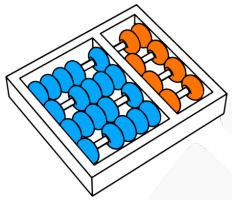
Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?



Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

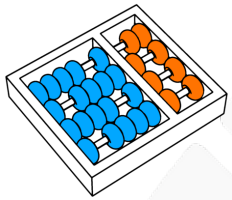
- Comparamos a complexidade de cada algoritmo.



Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

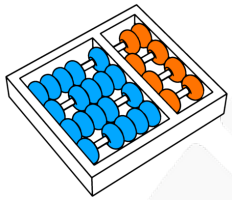


Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?



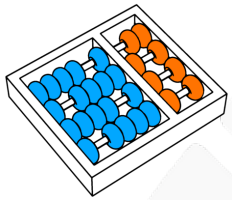
Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.



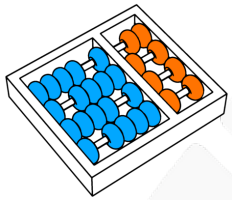
Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.
- Podemos comparar as cotas de cada algoritmo.



Comparando problemas

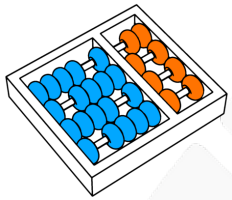
Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.
- Podemos comparar as cotas de cada algoritmo.

Exemplo: Achar o máximo é mais **FÁCIL** que ordenar um vetor!



Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

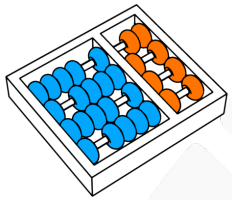
- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.
- Podemos comparar as cotas de cada algoritmo.

Exemplo: Achar o máximo é mais **FÁCIL** que ordenar um vetor!

- Máximo tem cota superior $O(n)$.



Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

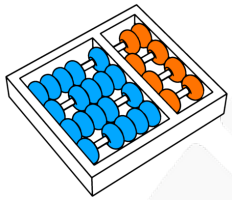
E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.
- Podemos comparar as cotas de cada algoritmo.

Exemplo: Achar o máximo é mais **FÁCIL** que ordenar um vetor!

- Máximo tem cota superior $O(n)$.
- Ordenação tem cota inferior $\Omega(n \log n)$.

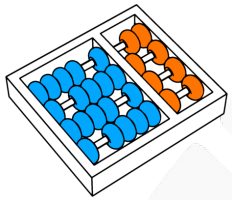
Reduções



Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

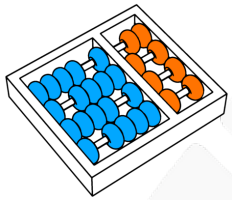


Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.

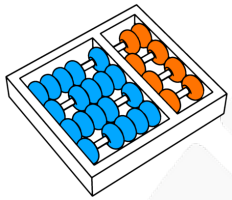


Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.



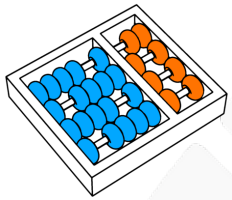
Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.



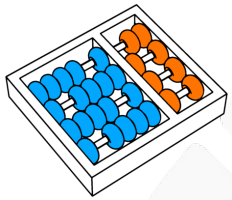
Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.
Como traduzir de **RUSSO PARA PORTUGUÊS**?



Combinando problemas

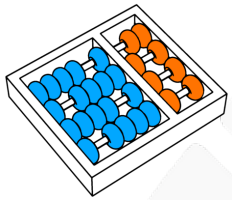
Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.
Como traduzir de **RUSSO PARA PORTUGUÊS**?

- Em geral, lidamos com problemas bem conhecidos.



Combinando problemas

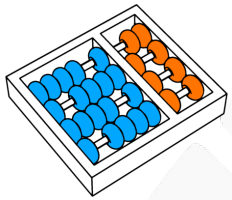
Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.
Como traduzir de **RUSSO PARA PORTUGUÊS**?

- Em geral, lidamos com problemas bem conhecidos.
- Mas eventualmente, topamos com problemas novos.



Combinando problemas

Uma analogia

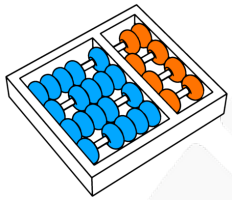
Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.
Como traduzir de **RUSSO PARA PORTUGUÊS**?

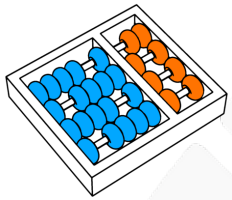
- Em geral, lidamos com problemas bem conhecidos.
- Mas eventualmente, topamos com problemas novos.

  Pergunta: como relacionar esses problemas?



Redução

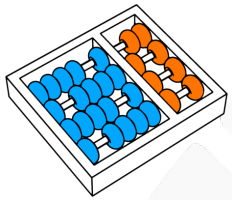
Problema A:



Redução

Problema A:

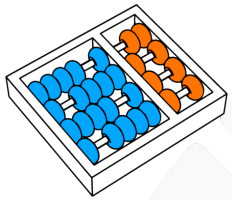
- Instância: I_A



Redução

Problema A:

- Instância: I_A
- Solução: S_A

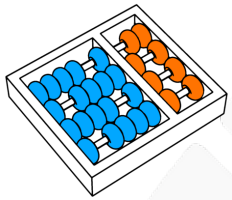


Redução

Problema A:

- Instância: I_A
- Solução: S_A

Problema B:



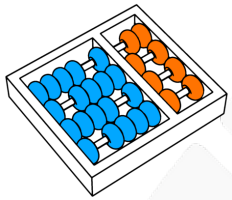
Redução

Problema A:

- Instância: I_A
- Solução: S_A

Problema B:

- Instância: I_B



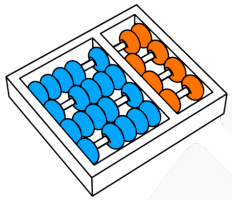
Redução

Problema A:

- Instância: I_A
- Solução: S_A

Problema B:

- Instância: I_B
- Solução: S_B



Redução

Problema A:

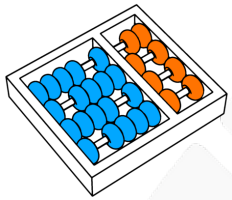
- Instância: I_A
- Solução: S_A

Problema B:

- Instância: I_B
- Solução: S_B

Definição

*Uma **REDUÇÃO** do problema A ao problema B é um par de sub-rotinas τ_I e τ_S tais que:*



Redução

Problema A:

- Instância: I_A
- Solução: S_A

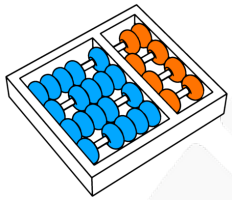
Problema B:

- Instância: I_B
- Solução: S_B

Definição

*Uma **REDUÇÃO** do problema A ao problema B é um par de sub-rotinas τ_I e τ_S tais que:*

- τ_I *transforma uma instância I_A de A em uma instância I_B de B.*



Redução

Problema A:

- Instância: I_A
- Solução: S_A

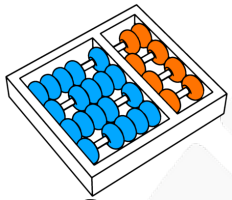
Problema B:

- Instância: I_B
- Solução: S_B

Definição

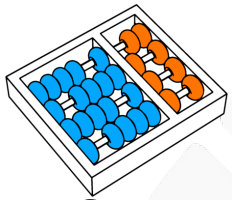
*Uma **REDUÇÃO** do problema A ao problema B é um par de sub-rotinas τ_I e τ_S tais que:*

- τ_I *transforma uma instância I_A de A em uma instância I_B de B.*
- τ_S *transforma uma solução S_B de I_B em uma solução S_A de I_A.*



Redução como um algoritmo

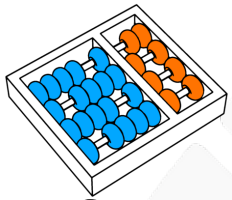
Como podemos resolver o problema A?



Redução como um algoritmo

Como podemos resolver o problema A?

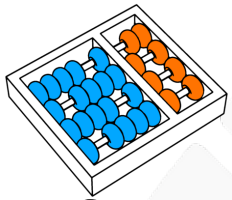
1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.



Redução como um algoritmo

Como podemos resolver o problema A?

1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.



Redução como um algoritmo

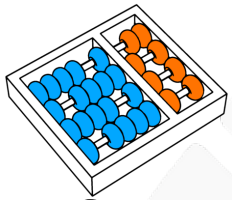
Como podemos resolver o problema A?

1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

Algoritmo: $\textsc{Redução}(I, A)$

---

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A



Redução como um algoritmo

Como podemos resolver o problema A?

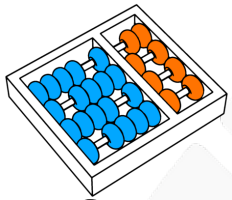
1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

Algoritmo: $\textsc{Redução}(I, A)$

---

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A

Em outras palavras:



Redução como um algoritmo

Como podemos resolver o problema A?

1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

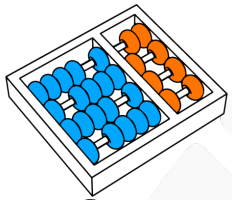
Algoritmo: $\textsc{Redução}(I, A)$

---

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A

Em outras palavras:

- Se sabemos resolver B, então também sei resolver A!



Redução como um algoritmo

Como podemos resolver o problema A?

1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

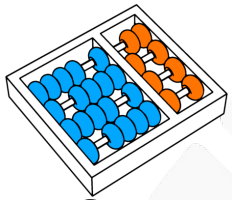
Algoritmo: $\textsc{Redução}(I, A)$

---

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A

Em outras palavras:

- Se sabemos resolver B, então também sei resolver A!
- A é mais “fácil"que B.



Redução como um algoritmo

Como podemos resolver o problema A?

1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

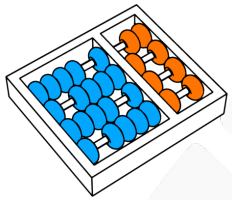
Algoritmo: $\textsc{Redução}(I, A)$

---

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A

Em outras palavras:

- Se sabemos resolver B, então também sei resolver A!
- A é mais “fácil"que B.
- Denotamos $\mathbf{A \preccurlyeq B}$.

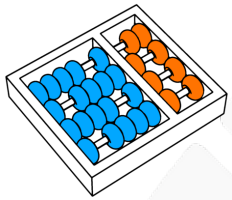


Um problema de origem

Problema (Alocação de Centros (AC))

Entrada: *Um grafo bipartido conexo* $G = ((X \cup Y), E)$ *e uma função de pesos nas arestas* $w : E \to \mathbb{R}_+$.

Saída: *Uma função* $\phi : X \to Y$ *que aloque cada vértice* v *em* X *a um vértice* $\phi[v]$ *em* Y *tal que o peso* $w(v, \phi[v])$ *seja* ***MÍNIMO***.

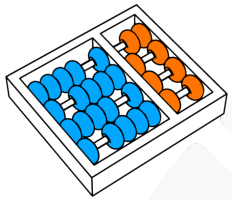


Um problema de destino

Problema (Caminho Mínimo (CM))

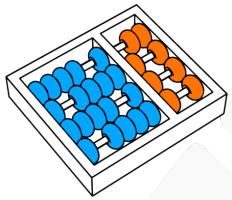
Entrada: *Um grafo direcionado acíclico* $G = (V, E)$*, uma função de peso* $w : E \to \mathbb{R}_+$ *nas arestas e um vértice origem origem* s*.*

Saída: *Um vetor* d *com* $d[v] = \text{dist}(s, v)$ *para* $v \in V$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS****.*



Reduzindo. Transformação da entrada

Recebemos uma **ENTRADA** do problema de origem AC:

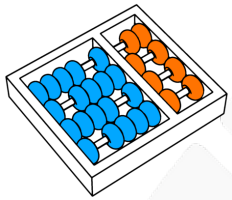


Reduzindo. Transformação da entrada

Recebemos uma **ENTRADA** do problema de origem AC:

Algoritmo: $\tau_I(G, w)$

1 $G' \leftarrow G$
2 $w' \leftarrow w$
3 Adicione um novo vértice s a G'
4 **para cada** $v \in Y$
5 Adicione a aresta (s, v) a G'
6 $w'(s, v) \leftarrow 0$
7 **devolva** (G', w', s)

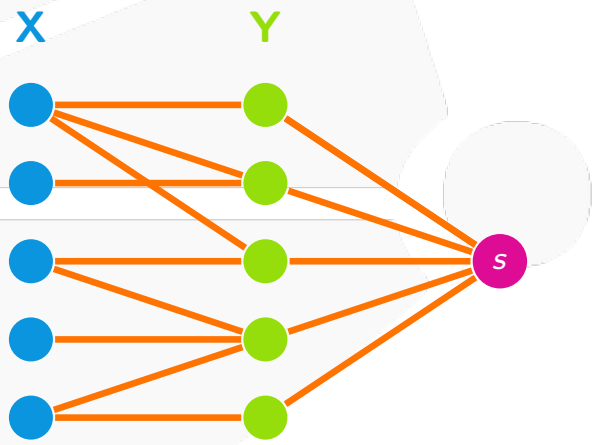


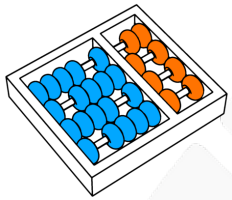
Reduzindo. Transformação da entrada

Recebemos uma **ENTRADA** do problema de origem AC:

Algoritmo: $\tau_I(G, w)$

1 $G' \leftarrow G$
2 $w' \leftarrow w$
3 Adicione um novo vértice s a G'
4 **para cada** $v \in Y$
5 Adicione a aresta (s, v) a G'
6 $w'(s, v) \leftarrow 0$
7 **devolva** (G', w', s)



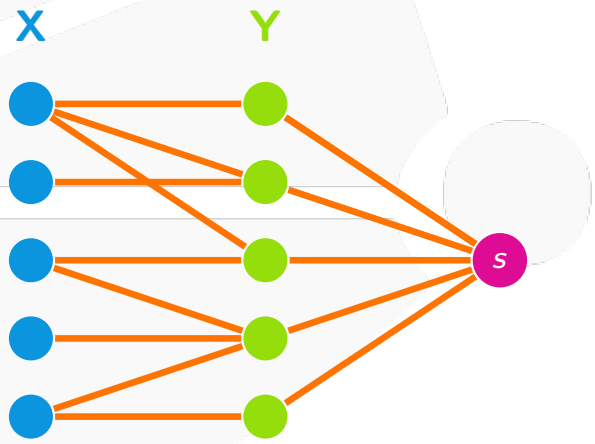


Reduzindo. Transformação da entrada

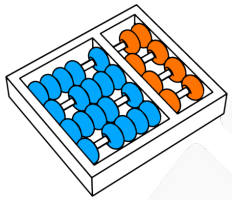
Recebemos uma **ENTRADA** do problema de origem AC:

Algoritmo: $\tau_I(G, w)$

1 $G' \leftarrow G$
2 $w' \leftarrow w$
3 Adicione um novo vértice s a G'
4 **para cada** $v \in Y$
5 Adicione a aresta (s, v) a G'
6 $w'(s, v) \leftarrow 0$
7 **devolva** (G', w', s)

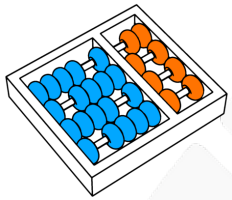


Tempo: $O(Y)$.



Reduzindo. Transformação da saída

Também recebemos uma **SOLUÇÃO** do problema de destino CM:

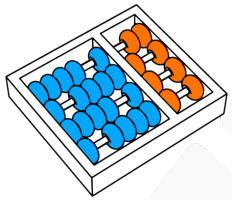


Reduzindo. Transformação da saída

Também recebemos uma **SOLUÇÃO** do problema de destino CM:

Algoritmo: $\tau_S(G, w, \mathbf{d}, \pi)$

1 **para cada** $v \in X$
2 $\quad \phi[v] \leftarrow \pi[v]$
3 **devolva** ϕ

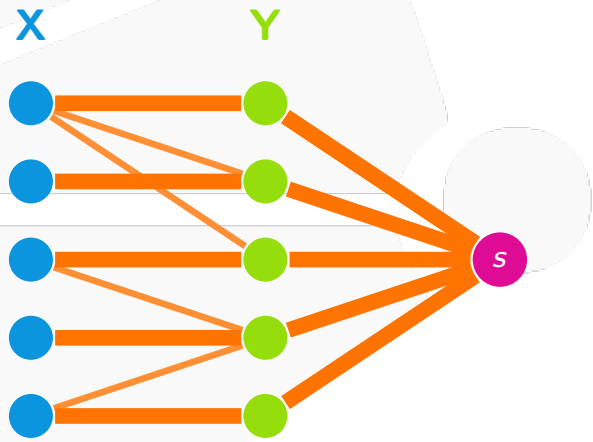


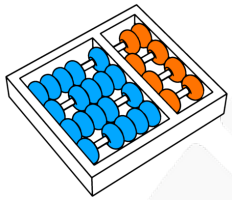
Reduzindo. Transformação da saída

Também recebemos uma **SOLUÇÃO** do problema de destino CM:

Algoritmo: $\tau_S(G, w, \mathbf{d}, \pi)$

1 **para cada** $v \in X$
2 $\quad \phi[v] \leftarrow \pi[v]$
3 **devolva** ϕ



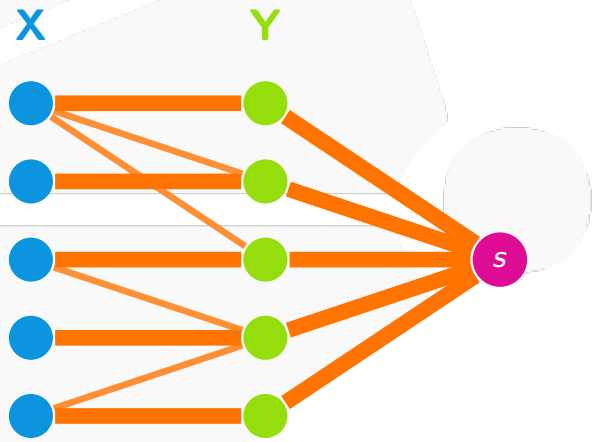


Reduzindo. Transformação da saída

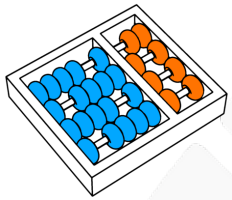
Também recebemos uma **SOLUÇÃO** do problema de destino CM:

Algoritmo: $\tau_S(G, w, \mathbf{d}, \pi)$

1 **para cada** $v \in X$
2 $\quad \phi[v] \leftarrow \pi[v]$
3 **devolva** ϕ

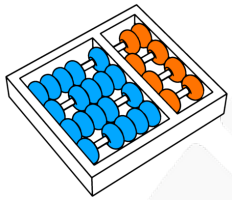


Tempo: $O(X)$.



Reduzindo. AC $\preceq$ CM

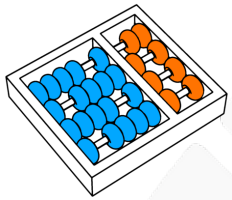
Seja ALG_{CM} um algoritmo para Caminho Mínimo.



Reduzindo. AC $\prec$ CM

Seja $\textsc{Alg}_{\mathsf{CM}}$ um algoritmo para Caminho Mínimo.

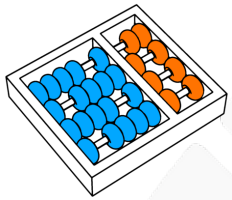
- $\textsc{Alg}_{\mathsf{CM}}$ poderia ser Dijkstra, Bellman-Ford...



Reduzindo. AC $\asymp$ CM

Seja ALG_{CM} um algoritmo para Caminho Mínimo.

- ALG_{CM} poderia ser Dijkstra, Bellman-Ford...
- Pode ser que **NÃO CONHEÇAMOS** um algoritmo para o problema de destino!



Reduzindo. AC $\preceq$ CM

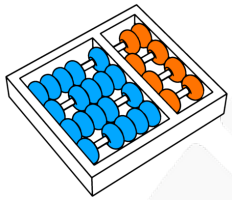
Seja $\textsc{Alg}_{\text{CM}}$ um algoritmo para Caminho Mínimo.

- $\textsc{Alg}_{\text{CM}}$ poderia ser $\textsc{Dijkstra}$, $\textsc{Bellman-Ford}$...
- Pode ser que **NÃO CONHEÇAMOS** um algoritmo para o problema de destino!

Algoritmo: $\textsc{Redução-AC-CM}(G, w)$

---

1 $(G', w', s) \leftarrow \tau_I(G, w)$
2 $(d, \pi) \leftarrow \textsc{Alg}_{\text{CM}}(G', w', s)$
3 $\phi \leftarrow \tau_S(G, w, d, \pi)$
4 **devolva** ϕ



Reduzindo. AC $\preccurlyeq$ CM

Seja $\textsc{Alg}_{\text{CM}}$ um algoritmo para Caminho Mínimo.

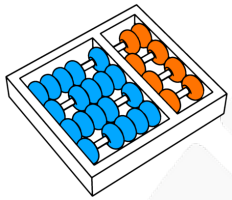
- $\textsc{Alg}_{\text{CM}}$ poderia ser $\textsc{Dijkstra}$, $\textsc{Bellman-Ford}$...
- Pode ser que **NÃO CONHEÇAMOS** um algoritmo para o problema de destino!

Algoritmo: $\textsc{Redução-AC-CM}(G, w)$

---

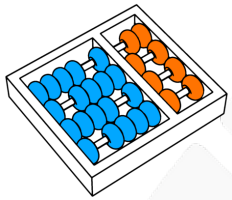
1 $(G', w', s) \leftarrow \tau_I(G, w)$
2 $(d, \pi) \leftarrow \textsc{Alg}_{\text{CM}}(G', w', s)$
3 $\phi \leftarrow \tau_S(G, w, d, \pi)$
4 **devolva** ϕ

Tempo total: [tempo da redução] + [tempo de$\textsc{Alg}_{\text{CM}}$]



Tempo da redução

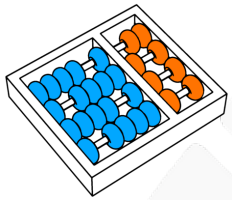
Quanto tempo gastamos só com a redução?



Tempo da redução

Quanto tempo gastamos só com a redução?

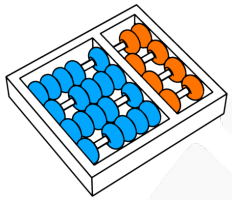
- Não contamos o tempo do algoritmo para o problema B.



Tempo da redução

Quanto tempo gastamos só com a redução?

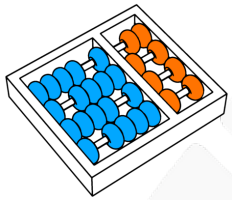
- Não contamos o tempo do algoritmo para o problema B.
- A **COMPLEXIDADE DE UMA REDUÇÃO** $f(n)$ é a soma dos tempos das transformações τ_I e τ_S.



Tempo da redução

Quanto tempo gastamos só com a redução?

- Não contamos o tempo do algoritmo para o problema B.
- A **COMPLEXIDADE DE UMA REDUÇÃO** $f(n)$ é a soma dos tempos das transformações τ_I e τ_S.
- Escrevemos $\mathbf{A \preccurlyeq_{f(n)} B}$.

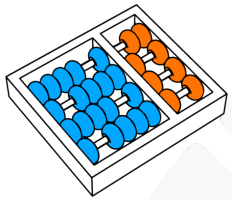


Tempo da redução

Quanto tempo gastamos só com a redução?

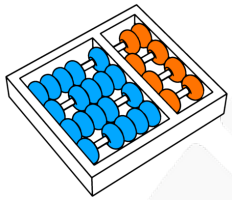
- Não contamos o tempo do algoritmo para o problema B.
- A **COMPLEXIDADE DE UMA REDUÇÃO** $f(n)$ é a soma dos tempos das transformações τ_I e τ_S.
- Escrevemos $\mathbf{A} \preccurlyeq_{\mathbf{f(n)}} \mathbf{B}$.

No caso de Redução-AC-CM: $\mathbf{AC} \preccurlyeq_{|\mathbf{X}|+|\mathbf{Y}|} \mathbf{CM}$.



Reduções polinomiais

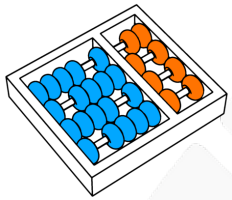
Queremos construir algoritmos rápidos. Mas, o que é “rápido”?



Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é “rápido”?

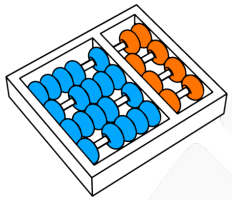
- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.



Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

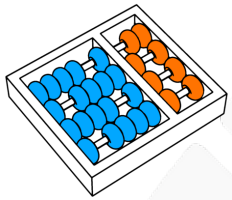
- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.



Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

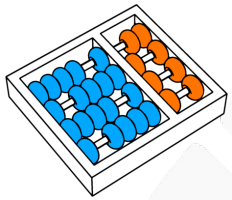


Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?



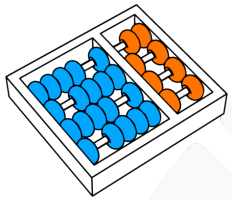
Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?

1. Se B tem um algoritmo de tempo polinomial, então A também.



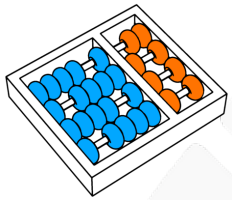
Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?

1. Se B tem um algoritmo de tempo polinomial, então A também.
2. Se A **NÃO** tem algoritmos de tempo polinomial, tampouco B.



Reduções polinomiais

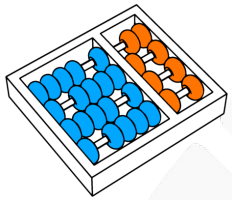
Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?

1. Se B tem um algoritmo de tempo polinomial, então A também.
2. Se A **NÃO** tem algoritmos de tempo polinomial, tampouco B.

- Isso é útil para distinguir problemas fáceis de difíceis!



Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

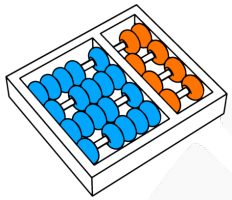
Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?

1. Se B tem um algoritmo de tempo polinomial, então A também.
2. Se A **NÃO** tem algoritmos de tempo polinomial, tampouco B.

- Isso é útil para distinguir problemas fáceis de difíceis!
- Mas, é assunto para depois...

Exemplos de reduções

Sistema Linear (LS) para Sistema Linear Simétrico (SLS)



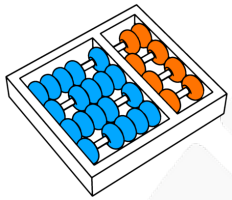
Problema de origem

Problema (Sistema Linear (LS))

Entrada: *Uma matriz M de dimensões $n \times n$ com determinante* ***NÃO*** *nulo e um vetor b de dimensão n.*

Saída: *Um vetor x de dimensão n que satisfaz o seguinte sistema linear:*

$$Mx = b.$$



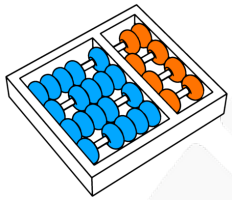
Problema de destino

Problema (Sistema Linear Simétrico (SLS))

Entrada: *Uma matriz M* ***SIMÉTRICA*** *de dimensões $n \times n$ com determinante* ***NÃO*** *nulo e um vetor b de dimensão n.*

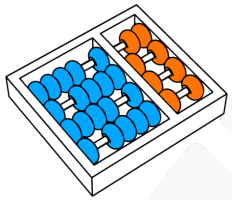
Saída: *Um vetor x de dimensão n que satisfaz o seguinte sistema linear:*

$$Mx = b.$$



Perguntas

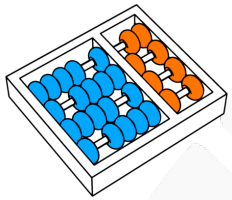
SLS é um caso particular de LS.



Perguntas

SLS é um caso particular de LS.

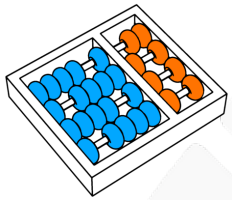
- Logo, trivialmente SLS $\propto$ LS.



Perguntas

SLS é um caso particular de LS.

- Logo, trivialmente SLS $\preceq$ LS.
- Será que LS é estritamente mais difícil?

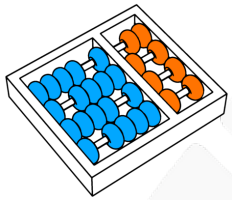


Perguntas

SLS é um caso particular de LS.

- Logo, trivialmente SLS $\preceq$ LS.
- Será que LS é estritamente mais difícil?

A resposta é **NÃO!**



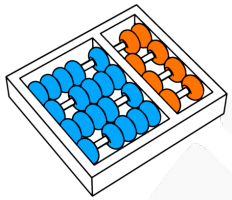
Perguntas

SLS é um caso particular de LS.

- Logo, trivialmente SLS $\preceq$ LS.
- Será que LS é estritamente mais difícil?

A resposta é **NÃO!**

- Iremos reduzir LS para SLS.



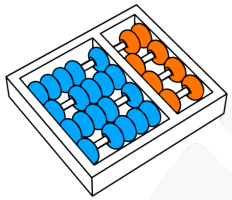
Perguntas

SLS é um caso particular de LS.

- Logo, trivialmente SLS $\preceq$ LS.
- Será que LS é estritamente mais difícil?

A resposta é **NÃO!**

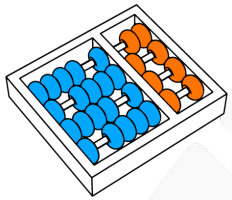
- Iremos reduzir LS para SLS.
- Isso é, LS $\preceq$ SLS.



Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

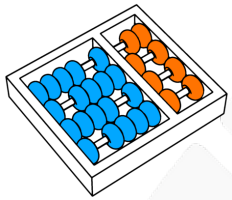


Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

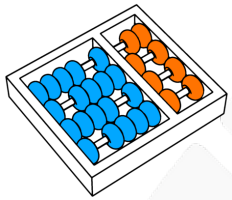


Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:



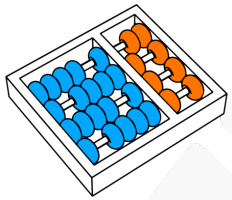
Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.



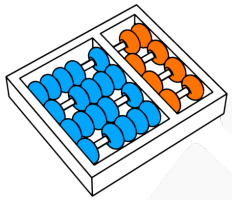
Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.



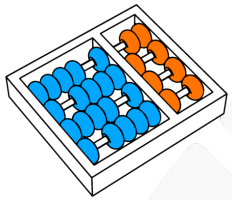
Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.



Um fato simples

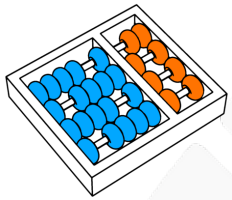
Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.



Um fato simples

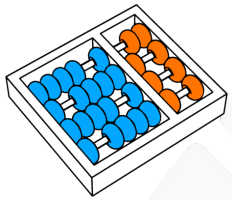
Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.
- Logo, M^T tem inversa Z.



Um fato simples

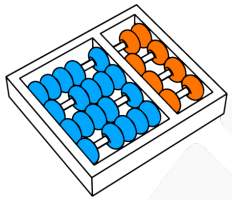
Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.
- Logo, M^T tem inversa Z.
- Multiplicamos $M^T Mx = M^T b$ por Z.



Um fato simples

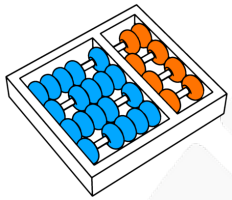
Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.
- Logo, M^T tem inversa Z.
- Multiplicamos $M^T Mx = M^T b$ por Z.
- Obtendo $Mx = b$



Um fato simples

Lema

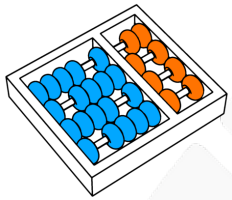
Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.
- Logo, M^T tem inversa Z.
- Multiplicamos $M^T Mx = M^T b$ por Z.
- Obtendo $Mx = b$

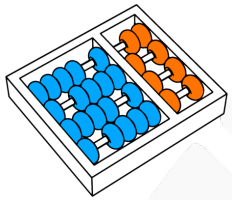
Observe que $M' = M^T M$ é uma matriz simétrica!



LS $\propto$ SLS

Algoritmo: REDUÇÃO-LS-SLS(M, b)

1 $M' \leftarrow M^T M$
2 $b' \leftarrow M^T b$
3 $x \leftarrow \text{ALG}_{\text{SLS}}(M', b')$
4 **devolva** x



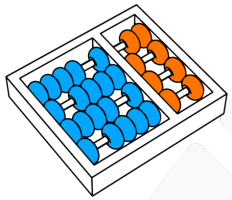
LS $\preccurlyeq$ SLS

Algoritmo: REDUÇÃO-LS-SLS(M, b)

1 $M' \leftarrow M^T M$
2 $b' \leftarrow M^T b$
3 $x \leftarrow \text{ALG}_{\text{SLS}}(M', b')$
4 **devolva** x

Concluímos que de fato LS $\preccurlyeq$ SLS.

Casamento Cíclico de Strings para Casamento de Strings

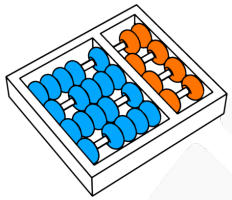


Problema de origem

Problema (Casamento cíclico de strings (CSM))

Entrada: *Um alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{n-1}$ *com* n *símbolos.*

Saída: SIM, *se* B *for um* ***DESLOCAMENTO CÍCLICO*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *então o número* k *de letras deslocadas faz parte da saída.*



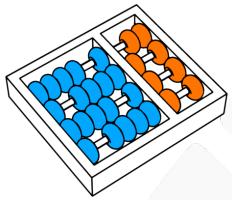
Problema de origem

Problema (Casamento cíclico de strings (CSM))

Entrada: *Um alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{n-1}$ *com* n *símbolos.*

Saída: SIM, *se* B *for um* ***DESLOCAMENTO CÍCLICO*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *então o número* k *de letras deslocadas faz parte da saída.*

Exemplo:



Problema de origem

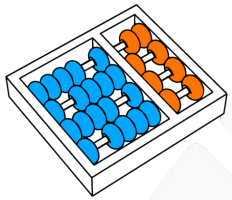
Problema (Casamento cíclico de strings (CSM))

Entrada: *Um alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{n-1}$ *com* n *símbolos.*

Saída: SIM, *se* B *for um* ***DESLOCAMENTO CÍCLICO*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *então o número* k *de letras deslocadas faz parte da saída.*

Exemplo:

- Entrada: $A = acgtact$ e $B = gtactac$



Problema de origem

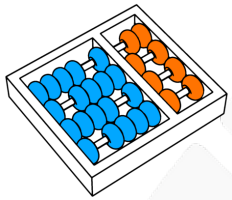
Problema (Casamento cíclico de strings (CSM))

Entrada: *Um alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{n-1}$ *com* n *símbolos.*

Saída: SIM, *se* B *for um* ***DESLOCAMENTO CÍCLICO*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *então o número* k *de letras deslocadas faz parte da saída.*

Exemplo:

- Entrada: $A = acgtact$ e $B = gtactac$
- Saída: SIM, $k = 2$

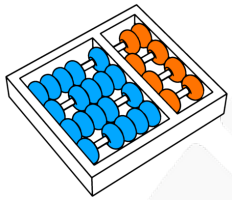


Problema de destino

Problema (Casamento de strings (SM))

Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*



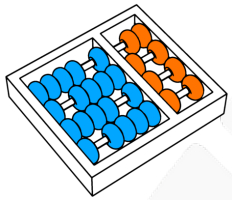
Problema de destino

Problema (Casamento de strings (SM))

Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*

Exemplo:



Problema de destino

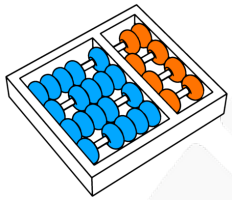
Problema (Casamento de strings (SM))

Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*

Exemplo:

- Entrada: $A = acgttaccgtacccg$ e $B = tac$



Problema de destino

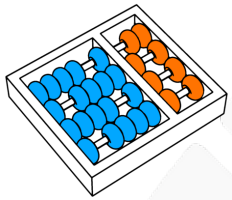
Problema (Casamento de strings (SM))

Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*

Exemplo:

- Entrada: $A = acgttaccgtacccg$ e $B = tac$
- Saída: SIM, $k = 4$



Problema de destino

Problema (Casamento de strings (SM))

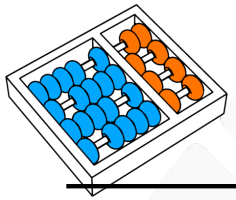
Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*

Exemplo:

- Entrada: $A = acgttaccgtacccg$ e $B = tac$
- Saída: SIM, $k = 4$

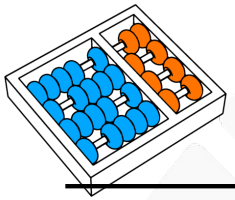
Observação: o problema SM pode ser resolvido em tempo $O(n + m)$ pelo algoritmo KMP de Knuth, Morris and Pratt (1977).



CSM $\asymp$ SM

Algoritmo: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\mathsf{SM}}(A', n', B', m')$

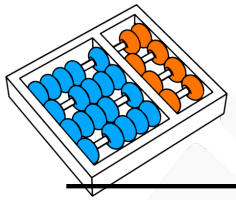


CSM $\prec$ SM

Algoritmo: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\mathsf{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$

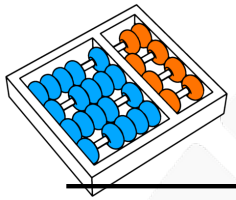


CSM $\preceq$ SM

Algoritmo: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\text{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que k é solução de SM se e somente se k também é solução de CSM.



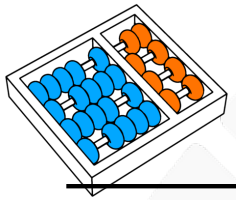
CSM $\preceq$ SM

Algoritmo: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\text{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que k é solução de SM se e somente se k também é solução de CSM.

Exemplo:



CSM $\preceq$ SM

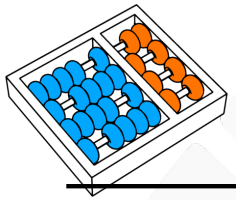
Algoritmo: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\text{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que k é solução de SM se e somente se k também é solução de CSM.

Exemplo:

- $I_{CSM} = (acgtact, gtactac, 7)$.



CSM $\preceq$ SM

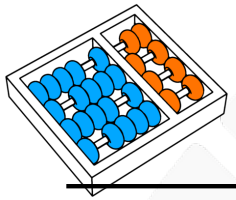
Algoritmo: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\text{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que k é solução de SM se e somente se k também é solução de CSM.

Exemplo:

- $I_{CSM} = (acgtact, gtactac, 7)$.
- $I_{SM} = (acgtactacgtact, 14, gtactac, 7)$.



CSM $\preceq$ SM

Algoritmo: $\textsc{Redução-CSM-SM}(A, B, n)$

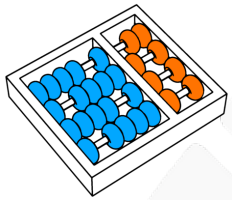
1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\text{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que k é solução de SM se e somente se k também é solução de CSM.

Exemplo:

- $I_{CSM} = (acgtact, gtactac, 7)$.
- $I_{SM} = (acgtactacgtact, 14, gtactac, 7)$.
- $S_{SM} = S_{CSM} = (\text{SIM}, 2)$.

Existência de Triângulo para Multiplicação de Matrizes Quadradas

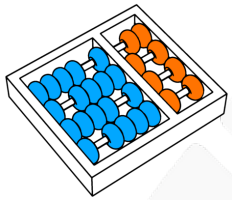


Problema de origem

Problema (Existência de triângulo (PET))

Entrada: *Um grafo conexo* $G = (V, E)$ *sem laços com* $n = |V|$ *e* $m = |E|$.

Saída: *Decidir se* G *contém um triângulo.*



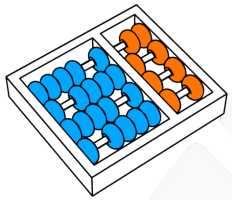
Problema de origem

Problema (Existência de triângulo (PET))

Entrada: *Um grafo conexo* $G = (V, E)$ *sem laços com* $n = |V|$ *e* $m = |E|$.

Saída: *Decidir se* G *contém um triângulo.*

Exemplo:



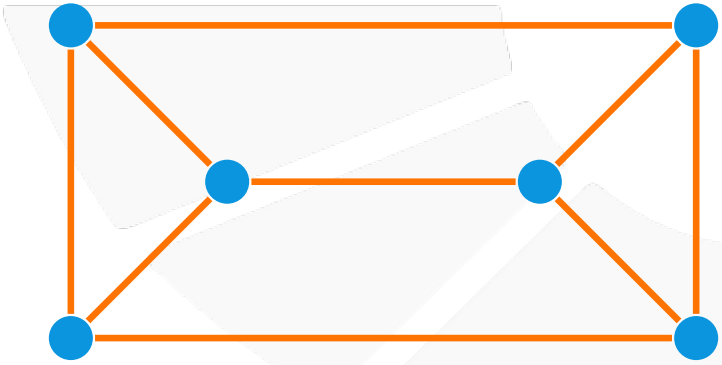
Problema de origem

Problema (Existência de triângulo (PET))

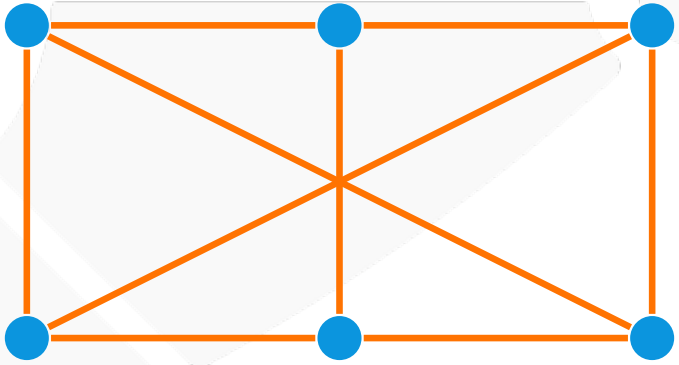
Entrada: *Um grafo conexo* $G = (V, E)$ *sem laços com* $n = |V|$ *e* $m = |E|$.

Saída: *Decidir se* G *contém um triângulo.*

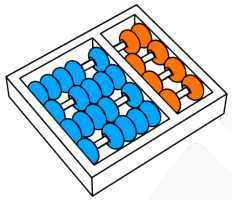
Exemplo:



SIM

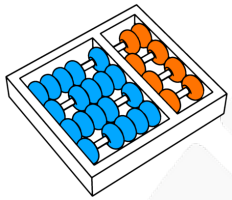


NAO



Observações sobre o PET

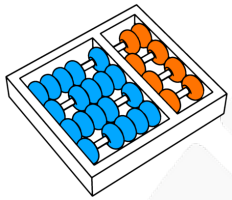
Alguns algoritmos conhecidos:



Observações sobre o PET

Alguns algoritmos conhecidos:

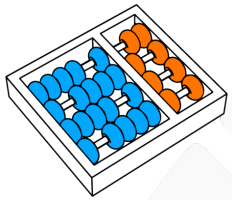
- Um algoritmo trivial de tempo $O(n^3)$:



Observações sobre o PET

Alguns algoritmos conhecidos:

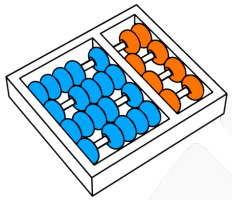
- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.



Observações sobre o PET

Alguns algoritmos conhecidos:

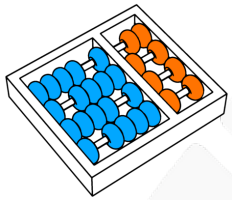
- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.
- Um algoritmo $O(mn)$:



Observações sobre o PET

Alguns algoritmos conhecidos:

- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.
- Um algoritmo $O(mn)$:
  - É muito bom se o grafo é **ESPARSO**.

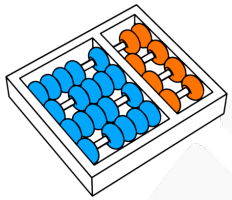


Observações sobre o PET

Alguns algoritmos conhecidos:

- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.
- Um algoritmo $O(mn)$:
  - É muito bom se o grafo é **ESPARSO**.

Vamos supor que o grafo é denso:



Observações sobre o PET

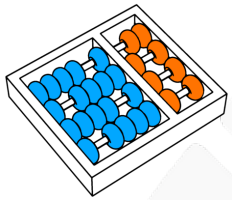
Alguns algoritmos conhecidos:

- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.
- Um algoritmo $O(mn)$:
  - É muito bom se o grafo é **ESPARSO**.

Vamos supor que o grafo é denso:

- G será representado por uma matriz de adjacência A:

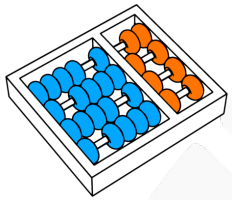
$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$



Um lema útil

Lema

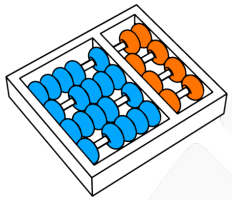
Seja $A^2 = A \times A$, *ou seja,* $a_{ij}^2 = \sum_{k=1}^{n} a_{ik} a_{kj}$.



Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

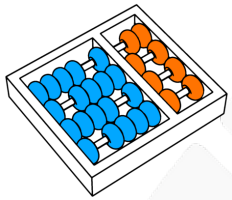


Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

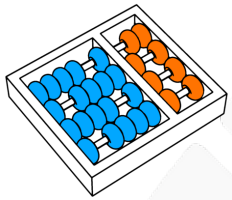


Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:



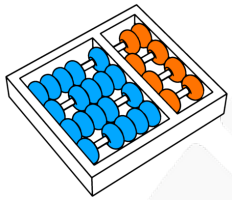
Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.



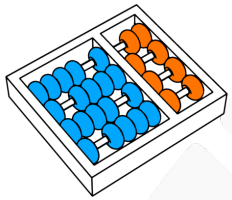
Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.



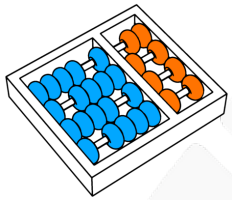
Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).



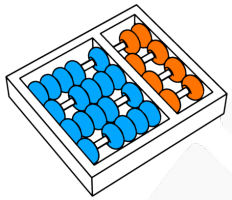
Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).



Um lema útil

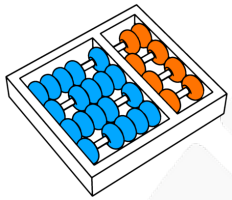
Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).

($\Leftarrow$)
- Seja um caminho (i, k, j) de i até j.



Um lema útil

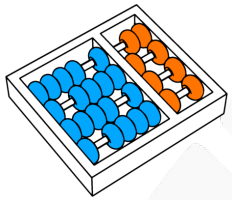
Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).

($\Leftarrow$)
- Seja um caminho (i, k, j) de i até j.
- Então, $a_{ik} = 1$ e $a_{kj} = 1$.



Um lema útil

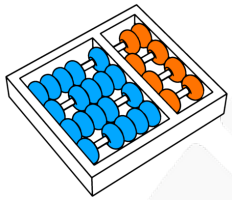
Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).

($\Leftarrow$)
- Seja um caminho (i, k, j) de i até j.
- Então, $a_{ik} = 1$ e $a_{kj} = 1$.
- Daí, $a_{ik} a_{kj} > 0$ e, portanto, $a^2_{ij} > 0$.

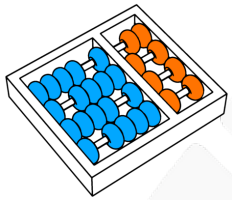


Problema destino

Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*



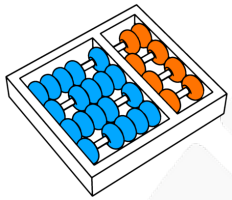
Problema destino

Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:



Problema destino

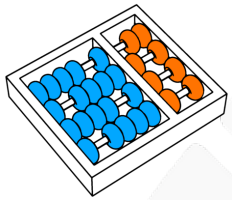
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.



Problema destino

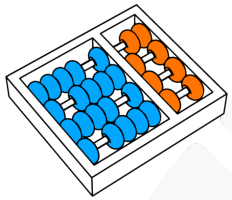
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.
- MMQ pode ser resolvida mais rapidamente:



Problema destino

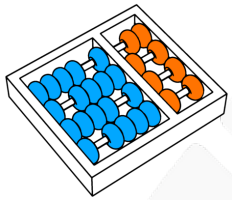
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.
- MMQ pode ser resolvida mais rapidamente:
  - Em tempo $O(n^{2,807})$ pelo algoritmo de Strassen (1969).



Problema destino

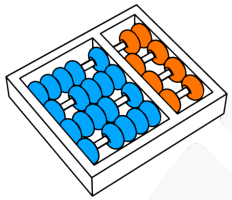
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.
- MMQ pode ser resolvida mais rapidamente:
  - Em tempo $O(n^{2,807})$ pelo algoritmo de Strassen (1969).
  - Em tempo $O(n^{2,376})$ pelo algoritmo de Coppersmith e Winograd (1990).



Problema destino

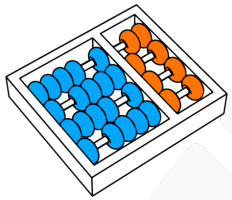
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

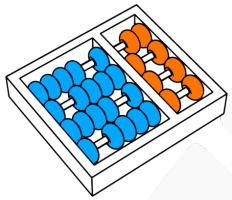
Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.
- MMQ pode ser resolvida mais rapidamente:
  - Em tempo $O(n^{2,807})$ pelo algoritmo de Strassen (1969).
  - Em tempo $O(n^{2,376})$ pelo algoritmo de Coppersmith e Winograd (1990).
  - Em tempo $O(n^{2,3728639})$ pelo de François Le Gall (2014).



Algoritmo de Strassem para MMQ

Dados: $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ e $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$.

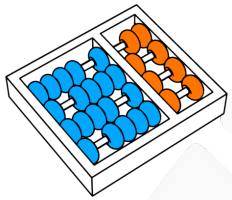


Algoritmo de Strassem para MMQ

Dados: $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ e $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$.

Desejamos calcular:

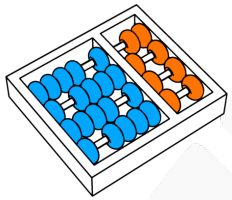
$$A \times B = \begin{bmatrix} A_{11} \times B_{11} + A_{12} \times B_{21} & A_{11} \times B_{12} + A_{12} \times B_{22} \\ A_{21} \times B_{11} + A_{22} \times B_{21} & A_{21} \times B_{12} + A_{22} \times B_{22} \end{bmatrix}.$$



Algoritmo de Strassem para MMQ

Defina:

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \times B_{11} \\
M_3 &= A_{11} \times (B_{12} - B_{22}) \\
M_4 &= A_{22} \times (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \times B_{22} \\
M_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \,.
\end{aligned}
$$



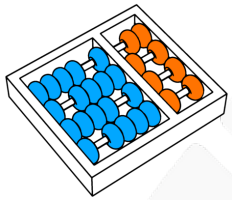
Algoritmo de Strassem para MMQ

Defina:

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \times B_{11} \\
M_3 &= A_{11} \times (B_{12} - B_{22}) \\
M_4 &= A_{22} \times (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \times B_{22} \\
M_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) .
\end{aligned}
$$

Note que:

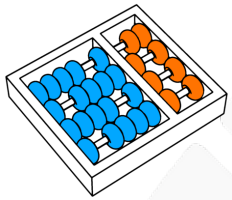
$$
\begin{aligned}
A_{11} \times B_{11} + A_{12} \times B_{21} &= M_1 + M_4 - M_5 + M_7 \\
A_{11} \times B_{12} + A_{12} \times B_{22} &= M_3 + M_5 \\
A_{21} \times B_{11} + A_{22} \times B_{21} &= M_2 + M_4 \\
A_{21} \times B_{12} + A_{22} \times B_{22} &= M_1 - M_2 + M_3 + M_6 .
\end{aligned}
$$



Algoritmo de Strassem para MMQ

Complexidade:

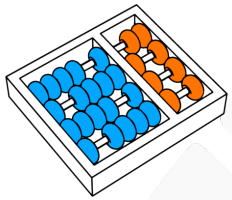
$$T(n) = 7\left(\frac{n}{2}\right) + O(n^2)$$



Algoritmo de Strassem para MMQ

Complexidade:

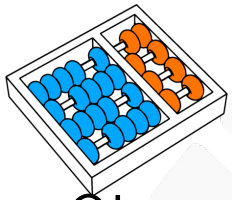
$$T(n) = 7\left(\frac{n}{2}\right) + O(n^2) = O(n^{\log_2 7})$$



Algoritmo de Strassem para MMQ

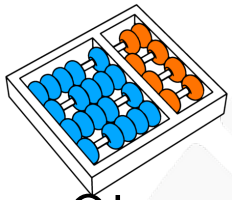
Complexidade:

$$T(n) = 7\left(\frac{n}{2}\right) + O(n^2) = O(n^{\log_2 7}) = O(n^{2,807}).$$



PET $\asymp$ MMQ

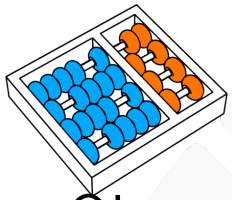
Observe que só existe triângulo com aresta (i, j) se:



PET $\propto$ MMQ

Observe que só existe triângulo com aresta (i, j) se:

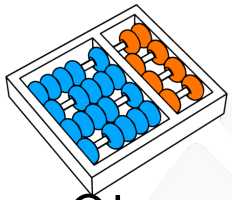
1. Existir um caminho de tamanho 2 de i a j.



PET $\asymp$ MMQ

Observe que só existe triângulo com aresta (i, j) se:

1. Existir um caminho de tamanho 2 de i a j.
2. Existir a aresta (i, j).



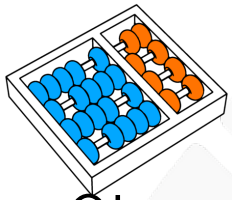
PET $\preceq$ MMQ

Observe que só existe triângulo com aresta (i, j) se:

1. Existir um caminho de tamanho 2 de i a j.
2. Existir a aresta (i, j).

Algoritmo: $\textsc{Redução-PET-MMQ}(A, n)$

```
1 A² ← Alg_MMQ(A, n)
2 para i = 1 até n
3     para j = 1 até n
4         se a²_ij > 0 e a_ij = 1
5             devolva SIM
6 devolva NAO
```



PET $\preceq$ MMQ

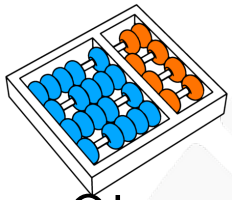
Observe que só existe triângulo com aresta (i, j) se:

1. Existir um caminho de tamanho 2 de i a j.
2. Existir a aresta (i, j).

Algoritmo: REDUÇÃO-PET-MMQ(A, n)

```
1 A² ← ALG_MMQ(A, n)
2 para i = 1 até n
3     para j = 1 até n
4         se a²_ij > 0 e a_ij = 1
5             devolva SIM
6 devolva NAO
```

- **Tempo da redução:** $O(n^2)$.



PET $\preceq$ MMQ

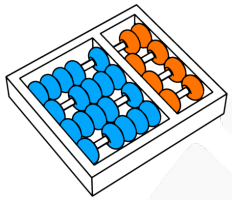
Observe que só existe triângulo com aresta (i, j) se:

1. Existir um caminho de tamanho 2 de i a j.
2. Existir a aresta (i, j).

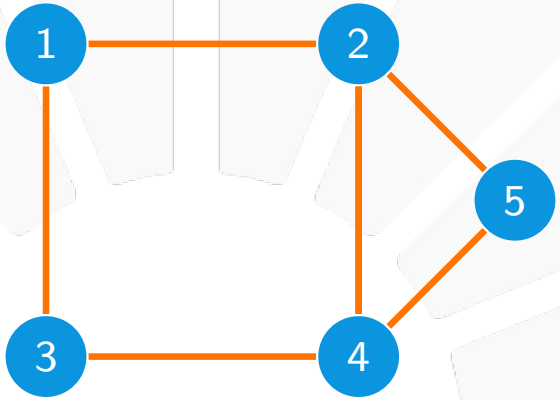
Algoritmo: $\textsc{Redução-PET-MMQ}(A, n)$

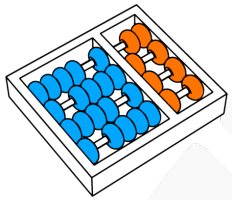
```
1 A² ← Alg_MMQ(A, n)
2 para i = 1 até n
3     para j = 1 até n
4         se a²_ij > 0 e a_ij = 1
5             devolva SIM
6 devolva NAO
```

- **Tempo da redução:** $O(n^2)$.
- **Tempo total:** $O(n^{2{,}3728639})$.

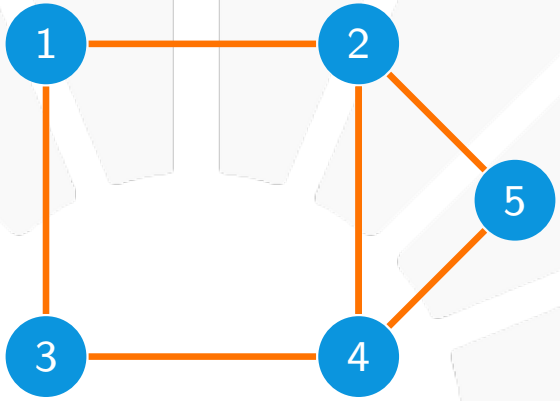


PET ≤ MMQ

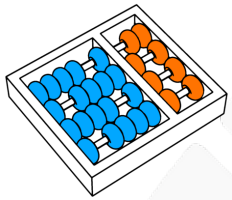




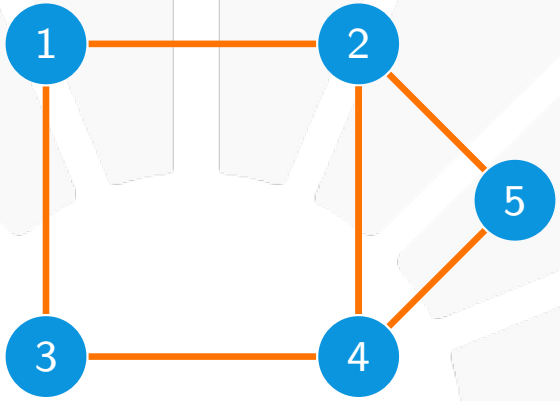
PET ≤ MMQ



A	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0



PET $\propto$ MMQ



A	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

A^2	1	2	3	4	5
1	2	0	0	2	1
2	0	3	2	1	1
3	0	2	2	0	1
4	2	1	0	3	1
5	1	1	1	1	2

Reduções

MC558 - Projeto e Análise de Algoritmos II

10/24

17

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

UNICAMP