Algoritmos e Teoria dos Grafos

Tópico 5b: Recapitulando: Elementos de Complexidade Computacional

Renato Carmo
André Guedes
Murilo Silva

Departamento de Informática da UFPR

2025

Recapitulando: Elementos de Complexidade Computacional

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo A_P para P que "usa um algoritmo A_Q para Q como subrotina".

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo A_P para P que "usa um algoritmo A_Q para Q como subrotina".

Exemplo:
Problema Q: Dado (G, v), computar $\delta_G(v)$
Problema P: Dado G, computar $\Delta(G)$

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo A_P para P que "usa um algoritmo A_Q para Q como subrotina".

Exemplo:
Problema Q: Dado (G, v), computar $\delta_G(v)$
Problema P: Dado G, computar $\Delta(G)$

$\mathrm{A}_P(G)$

Para $v \in V(G)$
    $\delta_G(v) = A_Q(G, v)$
Devolva **maior grau encontrado**

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo A_P para P que "usa um algoritmo A_Q para Q como subrotina".

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo A_P para P que "usa um algoritmo A_Q para Q como subrotina".

Exemplo:
Problema Q: Dado número natural $N \geq 2$, computar lista L de fatores primos de N
Problema P: Dado número natural $N \geq 2$, decidir se N é primo

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo A_P para P que "usa um algoritmo A_Q para Q como subrotina".

Exemplo:
Problema Q: Dado número natural $N \geq 2$, computar lista L de fatores primos de N
Problema P: Dado número natural $N \geq 2$, decidir se N é primo

---

$A_P(N)$

$L = A_Q(N)$
Se $|L| = 1$
    Devolva **sim**
Devolva **não**

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado (G, v), computar $\delta_G(v)$
Problema P: Dado G, computar $\Delta(G)$

$\mathrm{A}_P(G)$

Para $v \in V(G)$
    $\delta_G(v) = A_Q(G, v)$
Devolva **maior grau encontrado**

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado (G, v), computar $\delta_G(v)$
Problema P: Dado G, computar $\Delta(G)$

$A_P(G)$

Para $v \in V(G)$
    $\delta_G(v) = A_Q(G, v)$
Devolva **maior grau encontrado**

A_P é uma redução polinomial?

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado (G, v), computar $\delta_G(v)$
Problema P: Dado G, computar $\Delta(G)$

$\mathrm{A}_P(G)$

Para $v \in V(G)$
    $\delta_G(v) = A_Q(G, v)$
Devolva **maior grau encontrado**

A_P é uma redução polinomial?
Sim, pois A_Q polinomial $\Rightarrow$ A_P polinomial

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".
2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".
2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado número natural $N \geq 2$, computar lista L de fatores primos de N
Problema P: Dado número natural $N \geq 2$, decidir se N é primo

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado número natural $N \geq 2$, computar lista L de fatores primos de N
Problema P: Dado número natural $N \geq 2$, decidir se N é primo

---

$\mathrm{A}_P(N)$

$L = A_Q(N)$
Se $|L| = 1$
    Devolva **sim**
Devolva **não**

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado número natural $N \geq 2$, computar lista L de fatores primos de N
Problema P: Dado número natural $N \geq 2$, decidir se N é primo

---

$\mathrm{A}_P(N)$

$L = A_Q(N)$
Se $|L| = 1$
    Devolva **sim**
Devolva **não**

A_P é uma redução polinomial?

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado número natural $N \geq 2$, computar lista L de fatores primos de N
Problema P: Dado número natural $N \geq 2$, decidir se N é primo

---

$\mathrm{A}_P(N)$

$L = A_Q(N)$
Se $|L| = 1$
    Devolva **sim**
Devolva **não**

A_P é uma redução polinomial?
Sim, pois A_Q polinomial $\Rightarrow$ A_P polinomial

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado número natural $N \geq 2$, computar lista L de fatores primos de N
Problema P: Dado número natural $N \geq 2$, decidir se N é primo

---

$\mathrm{A}_P(N)$

$L = A_Q(N)$
Se $|L| = 1$
    Devolva **sim**
Devolva **não**

A_P é uma redução polinomial?
Sim, pois A_Q polinomial $\Rightarrow$ A_P polinomial

Importante: Não importa a complexidade de A_Q.

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Exemplo:
Problema Q: Dado número natural $N \geq 2$, computar lista L de fatores primos de N
Problema P: Dado número natural $N \geq 2$, decidir se N é primo

---

$\mathrm{A}_P(N)$

$L = A_Q(N)$
Se $|L| = 1$
    Devolva **sim**
Devolva **não**

A_P é uma redução polinomial?
Sim, pois A_Q polinomial $\Rightarrow$ A_P polinomial

Importante: Não importa a complexidade de A_Q. O que importa é a implicação!

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

3. $P \preceq Q$:

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

3. $P \preceq Q$: Notação para existência de redução polinomial de P a Q.

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

3. $P \preceq Q$: Notação para existência de redução polinomial de P a Q.
   (ou, equivalentemente, P é **polinomialmente redutível** a Q)

Recapitulando: Elementos de Complexidade Computacional

1. Uma **redução** do problema computacional P ao problema computacional Q é um algoritmo $\mathcal{A}_P$ para P que "usa um algoritmo $\mathcal{A}_Q$ para Q como subrotina".

2. Dizemos que a redução é **polinomial** se o fato de $\mathcal{A}_Q$ ser um algoritmo polinomial é suficiente para concluir que $\mathcal{A}_P$ também é polinomial.

3. $P \preceq Q$: Notação para existência de redução polinomial de P a Q.
   (ou, equivalentemente, P é **polinomialmente redutível** a Q)

4. $P \preceq Q$ pode ser lido como "Q é pelo menos tão difícil quanto P".

Recapitulando: Elementos de Complexidade Computacional

Recapitulando: Elementos de Complexidade Computacional

1. A classe $\mathcal{NP}$ é a classe dos problemas de decisão para os quais é possivel verificar em tempo polinomial que uma instância é positiva mediante informação adicional ("certificado" ou "testemunha").

2. Um problema computacional $\mathcal{NP}$-difícil é um problema computacional ao qual todo problema da classe $\mathcal{NP}$ é redutível.

Recapitulando: Elementos de Complexidade Computacional

1. A classe $\mathcal{NP}$ é a classe dos problemas de decisão para os quais é possivel verificar em tempo polinomial que uma instância é positiva mediante informação adicional ("certificado" ou "testemunha").

2. Um problema computacional $\mathcal{NP}$-difícil é um problema computacional ao qual todo problema da classe $\mathcal{NP}$ é redutível.

Teorema 6: Se $P \preceq Q$ e Q é polinomial, então P é polinomial.

Teorema 6: Se $P \preceq Q$ e Q é polinomial, então P é polinomial.

Teorema 7: Se $P \preceq Q$ e P é $\mathcal{NP}$-difícil, então Q é $\mathcal{NP}$-difícil.

Teorema 6: Se $P \preceq Q$ e Q é polinomial, então P é polinomial.

Teorema 7: Se $P \preceq Q$ e P é $\mathcal{NP}$-difícil, então Q é $\mathcal{NP}$-difícil.

Corolário 8: Se existir algoritmo polinomial para algum problema $\mathcal{NP}$-difícil, então $\mathcal{P} = \mathcal{NP}$.

Teorema 6: Se $P \preceq Q$ e Q é polinomial, então P é polinomial.

Teorema 7: Se $P \preceq Q$ e P é $\mathcal{NP}$-difícil, então Q é $\mathcal{NP}$-difícil.

Corolário 8: Se existir algoritmo polinomial para algum problema $\mathcal{NP}$-difícil, então $\mathcal{P} = \mathcal{NP}$.

Por isso não se conhece, nem acredita-se que exista algoritmo polinomial para qualquer problema $\mathcal{NP}$-difícil.