Problemas NP-Completos

Fábio Botler

Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro

Aula de hoje

- Algoritmos eficientes
- Problemas tratáveis e intratáveis
- Problemas algorítmicos
- Codificações
- Tipos de problemas

Definição

Um algoritmo A é dito **eficiente** *se sua complexidade for um polinômio no tamanho da sua entrada.*

Definição

Um algoritmo A é dito **eficiente** *se sua complexidade for um polinômio no tamanho da sua entrada.*

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Definição

Um algoritmo A é dito **eficiente** *se sua complexidade for um polinômio no tamanho da sua entrada.*

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$

Definição

Um algoritmo A é dito **eficiente** *se sua complexidade for um polinômio no tamanho da sua entrada.*

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$

Definição

Um algoritmo A é dito **eficiente** *se sua complexidade for um polinômio no tamanho da sua entrada.*

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$
- $O(n^{1000})$

Definição

Um algoritmo A é dito **eficiente** *se sua complexidade for um polinômio no tamanho da sua entrada.*

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$
- $O(n^{1000})$
- $O(n^{1000^{32938}})$

Definição

Um algoritmo A é dito **eficiente** *se sua complexidade for um polinômio no tamanho da sua entrada.*

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$
- $O(n^{1000})$
- $O(n^{1000^{32938}})$
- $O(n^4 \log n)$

Definição

Um algoritmo A é dito **eficiente** *se sua complexidade for um polinômio no tamanho da sua entrada.*

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$
- $O(n^{1000})$
- $O(n^{1000^{32938}})$
- $O(n^4 \log n)$
- $O(n \log \log n)$

Definição

Um problema é dito **tratável** *se existe algoritmo eficiente que o resolva, e* **intratável** *caso contrário.*

Definição

Um problema é dito **tratável** *se existe algoritmo eficiente que o resolva, e* **intratável** *caso contrário.*

Como mostramos que um problema Π é tratável?

Definição

Um problema é dito **tratável** *se existe algoritmo eficiente que o resolva, e* **intratável** *caso contrário.*

Como mostramos que um problema Π é tratável?

$\triangleright$ Basta mostrar um algoritmo eficiente que resolva Π!

Definição

Um problema é dito **tratável** *se existe algoritmo eficiente que o resolva, e* **intratável** *caso contrário.*

Como mostramos que um problema Π é tratável?

$\triangleright$ Basta mostrar um algoritmo eficiente que resolva Π!

Se não conhecemos um algoritmo eficiente para resolver um problema Π, então Π é intratável?

Definição

Um problema é dito **tratável** *se existe algoritmo eficiente que o resolva, e* **intratável** *caso contrário.*

Como mostramos que um problema Π é tratável?

- ▷ Basta mostrar um algoritmo eficiente que resolva Π!

Se não conhecemos um algoritmo eficiente para resolver um problema Π, então Π é intratável?

- ▷ Não.

Definição

Um problema é dito **tratável** *se existe algoritmo eficiente que o resolva, e* **intratável** *caso contrário.*

Como mostramos que um problema Π é tratável?

- ▷ Basta mostrar um algoritmo eficiente que resolva Π!

Se não conhecemos um algoritmo eficiente para resolver um problema Π, então Π é intratável?

- ▷ Não.
- ▷ Precisamos mostrar que TODO algoritmo que resolve Π não é eficiente.

Alguns problemas problemas tratáveis:

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo
- Decidir se um grafo é planar

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo
- Decidir se um grafo é planar
- Decidir se um grafo é bipartido

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo
- Decidir se um grafo é planar
- Decidir se um grafo é bipartido
- Decidir se um número é primo

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo
- Decidir se um grafo é planar
- Decidir se um grafo é bipartido
- Decidir se um número é primo

Alguns problemas que não sabemos se são tratáveis:

Alguns problemas que não sabemos se são tratáveis:

- Encontrar um caminho máximo em um grafo

Alguns problemas que não sabemos se são tratáveis:

- Encontrar um caminho máximo em um grafo
- Encontrar um emparelhamento máximo em um hipergrafo

Alguns problemas que não sabemos se são tratáveis:

- Encontrar um caminho máximo em um grafo
- Encontrar um emparelhamento máximo em um hipergrafo
- Encontrar um corte máximo em um grafo

Alguns problemas que não sabemos se são tratáveis:

- Encontrar um caminho máximo em um grafo
- Encontrar um emparelhamento máximo em um hipergrafo
- Encontrar um corte máximo em um grafo
- Decidir se um grafo pode ser colorido com três cores

Alguns problemas que não sabemos se são tratáveis:

- Encontrar um caminho máximo em um grafo
- Encontrar um emparelhamento máximo em um hipergrafo
- Encontrar um corte máximo em um grafo
- Decidir se um grafo pode ser colorido com três cores
- Encontrar uma fatorização de um número composto

Tratáveis	**Desconhecidos**
Caminho mínimo	Caminho máximo
Multiplicação de matrizes	
Emparelhamento em grafos	Emparelhamento em hipergrafos
Corte Mínimo	Corte máximo
Conexidade	
Planaridade	
2-coloração	3-coloração
Primalidade	Fatorização de inteiros

Definição

Um **problema algorítmico** Π *é um par* $(\mathcal{C}_\Pi, Q_\Pi)$*, onde* $\mathcal{C}_\Pi$ *é o* **conjunto de dados**, *também chamados de* **instâncias**, *do problema e uma questão solicitada* Q_Π*, chamada* **objetivo do problema**.

Definição

Um **problema algorítmico** Π *é um par* $(\mathcal{C}_\Pi, Q_\Pi)$*, onde* $\mathcal{C}_\Pi$ *é o* **conjunto de dados**, *também chamados de* **instâncias**, *do problema e uma questão solicitada* Q_Π*, chamada* **objetivo do problema**.

Resolver *um problema* Π *é desenvolver um algoritmo cuja entrada sejam os elementos de* $\mathcal{C}_\Pi$*, e cuja saída, chamada* **solução**, *responda ao objetivo do problema.*

Exemplo

Clique **(G,k)**

DADOS: *Um grafo G e um inteiro $k > 0$*

OBJETIVO: *Encontrar uma clique de tamanho pelo menos k em G.*

Exemplo

Clique **(G,k)**

DADOS:	*Um grafo G e um inteiro $k > 0$*
OBJETIVO:	*Encontrar uma clique de tamanho pelo menos k em G.*

Conjunto de dados:	*o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.*

Exemplo

Clique **(G,k)**

DADOS:	*Um grafo G e um inteiro $k > 0$*
OBJETIVO:	*Encontrar uma clique de tamanho pelo menos k em G.*

Conjunto de dados:	*o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.*
Instância:	*um par (G, k), onde G é um grafo, e k é um inteiro positivo.*

Exemplo

Clique **(G,k)**

DADOS:	*Um grafo G e um inteiro $k > 0$*
OBJETIVO:	*Encontrar uma clique de tamanho pelo menos k em G.*

Conjunto de dados:	*o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.*
Instância:	*um par (G, k), onde G é um grafo, e k é um inteiro positivo.*
Solução:	*um subgrafo completo de G com pelo menos k vértices, se existir.*

Exemplo

Conjunto Independente **(G,k)**

DADOS: *Um grafo G e um inteiro $k > 0$*

OBJETIVO: *Decidir se G possui conjunto independente de tamanho pelo menos k.*

Exemplo

Conjunto Independente **(G,k)**

DADOS: *Um grafo G e um inteiro $k > 0$*

OBJETIVO: *Decidir se G possui conjunto independente de tamanho pelo menos k.*

Conjunto de dados: *o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.*

Exemplo

Conjunto Independente **(G,k)**

DADOS: *Um grafo G e um inteiro $k > 0$*

OBJETIVO: *Decidir se G possui conjunto independente de tamanho pelo menos k.*

Conjunto de dados: *o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.*

Instância: *um par (G, k), onde G é um grafo, e k é um inteiro positivo.*

Exemplo

Conjunto Independente **(G,k)**

DADOS:	*Um grafo G e um inteiro $k > 0$*
OBJETIVO:	*Decidir se G possui conjunto independente de tamanho pelo menos k.*

Conjunto de dados:	*o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.*
Instância:	*um par (G, k), onde G é um grafo, e k é um inteiro positivo.*
Solução:	*SIM, caso G possua um conjunto independente de tamanho pelo menos k, ou NÃO, caso todo conjunto independente de G tenha tamanho menor que k.*

Codificações

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Porque o tamanho da instância é importante?

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Porque o tamanho da instância é importante?

- ▷ Porque é a medida pela qual calculamos a complexidade do algoritmo.

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Porque o tamanho da instância é importante?

- ▷ Porque é a medida pela qual calculamos a complexidade do algoritmo.

Para um grafo usaremos o número de vértices ou de arestas.

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Porque o tamanho da instância é importante?

- ▷ Porque é a medida pela qual calculamos a complexidade do algoritmo.

Para um grafo usaremos o número de vértices ou de arestas.

Em geral uma codificação é desnecessariamente longa quando

- ▷ contém partes irrelevantes para o problema;
- ▷ algum inteiro da instância está codificado no sistema unário.

É natural que a codificação de uma instância aumente quando a instância aumenta.

É natural que a codificação de uma instância aumente quando a instância aumenta.

Porém codificar um número na base unária traz alguns problemas:

É natural que a codificação de uma instância aumente quando a instância aumenta.

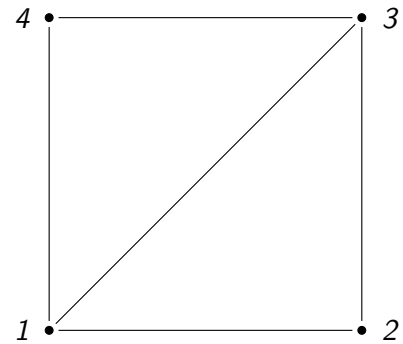
Porém codificar um número na base unária traz alguns problemas:

Exemplo

Base	*Representação*	*Tamanho*
10	*130*	*1*
1	*111111111111111111111111111111* *111111111111111111111111111111* *111111111111111111111111111111* *111111111111111111111111111111* *1111111111*	*130*
2	*10000010*	*8*
3	*11211*	*5*
4	*2002*	*4*

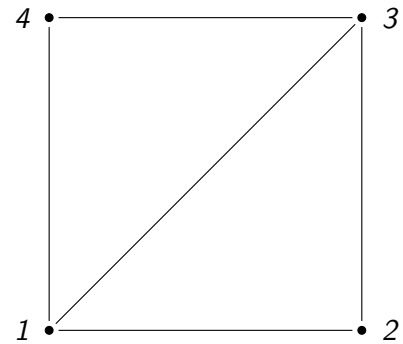
Exemplo

Vamos codificar o grafo.



Exemplo

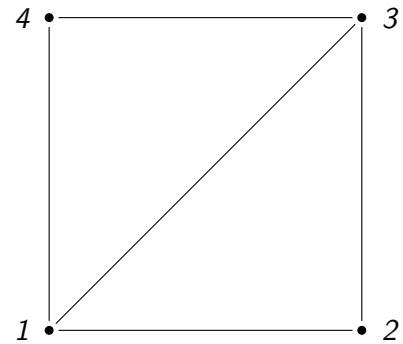
Vamos codificar o grafo.



G pode ser codificado por $\{12, 23, 34, 41, 13\}$

Exemplo

Vamos codificar o grafo.

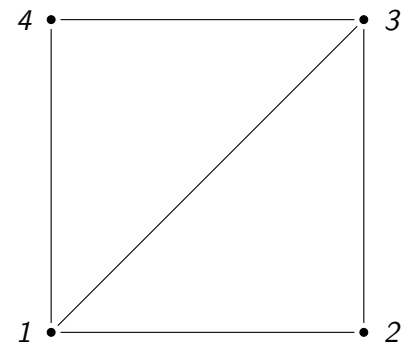


G pode ser codificado por $\{12, 23, 34, 41, 13\}$

Se a aresta xy é codificada por $/x_b, y_b/$*, onde* x_b *é a representação de x na base b.*

Exemplo

Vamos codificar o grafo.



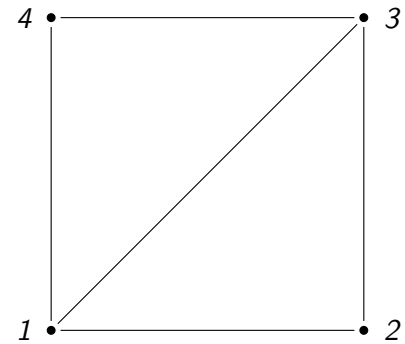
G pode ser codificado por $\{12, 23, 34, 41, 13\}$

Se a aresta xy é codificada por $/x_b, y_b/$*, onde* x_b *é a representação de x na base b.*

- ▷ *Binária:* $/1, 10/10, 11/11, 100/100, 1/1, 11/$

Exemplo

Vamos codificar o grafo.



G pode ser codificado por $\{12, 23, 34, 41, 13\}$

Se a aresta xy é codificada por $/x_b, y_b/$*, onde* x_b *é a representação de x na base b.*

- ▷ *Binária:* $/1, 10/10, 11/11, 100/100, 1/1, 11/$
- ▷ *Unária:* $/1, 11/11, 111/111, 1111/1111, 1/1, 111/$

Tipos de problemas

Há três tipos de problemas algorítmicos.

Tipos de problemas

Há três tipos de problemas algorítmicos.

1) problemas de **decisão**

2) problemas de **localização**

3) problemas de **otimização**

Exemplo

Problema do caixeiro-viajante

Exemplo

Problema do caixeiro-viajante

Versão de decisão

DADOS: *Um grafo G com pesos nas arestas e um inteiro $k > 0$*

OBJETIVO: **Decidir** *se G possui um ciclo Hamiltoniano com peso menor ou igual a k.*

Exemplo

Problema do caixeiro-viajante

Versão de decisão

DADOS: *Um grafo G com pesos nas arestas e um inteiro $k > 0$*

OBJETIVO: **Decidir** *se G possui um ciclo Hamiltoniano com peso menor ou igual a k.*

Versão de localização

DADOS: *Um grafo G com pesos nas arestas e um inteiro $k > 0$*

OBJETIVO: **Encontrar** *em G um ciclo Hamiltoniano com peso menor ou igual a k.*

Exemplo

Problema do caixeiro-viajante

Versão de decisão

DADOS: *Um grafo G com pesos nas arestas e um inteiro $k > 0$*

OBJETIVO: **Decidir** *se G possui um ciclo Hamiltoniano com peso menor ou igual a k.*

Versão de localização

DADOS: *Um grafo G com pesos nas arestas e um inteiro $k > 0$*

OBJETIVO: **Encontrar** *em G um ciclo Hamiltoniano com peso menor ou igual a k.*

Versão de otimização

DADOS: *Um grafo G com pesos nas arestas*

OBJETIVO: *Encontrar em G um ciclo Hamiltoniano de custo* **ótimo** *(mínimo/máximo).*

Qual as relações entre as versões de um mesmo problema?

Qual as relações entre as versões de um mesmo problema?

Podemos usar um para resolver outro?

- ▷ É possível usar a versão de otimização
  para resolver a versão de localização,

Qual as relações entre as versões de um mesmo problema?

Podemos usar um para resolver outro?

- ▷ É possível usar a versão de otimização
  para resolver a versão de localização,
- ▷ e usar a versão de localização para resolver a versão de decisão

Qual as relações entre as versões de um mesmo problema?

Podemos usar um para resolver outro?

- ▷ É possível usar a versão de otimização
  para resolver a versão de localização,
- ▷ e usar a versão de localização para resolver a versão de decisão

Frequentemente, é possível usar a versão de decisão para resolver a versão de otimização!

Problemas NP-Completos

Fábio Botler

Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro