

Projeto e Análise de Algoritmos II (MC558)

Classes de Problemas

Prof. Dr. Ruben Interian

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Problemas de decisão
- 3 Problema SAT
- 4 Algoritmos não determinísticos
- 5 Síntese

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Problemas de decisão
- 3 Problema SAT
- 4 Algoritmos não determinísticos
- 5 Síntese

Revisão do conteúdo

- Sabemos que existem problemas **mais fáceis** de resolver.
 - Possuem complexidade **polinomial**;
 - A maioria das vezes o **grau** do polinômio é **pequeno**: 1, 2, 3.
 - **Exemplos**: ordenar um vetor, obter a mediana de um vetor, árvore geradora mínima de um grafo, caminho mais curto em grafos, multiplicação de matrizes.

Revisão do conteúdo

- Sabemos que existem problemas **mais fáceis** de resolver.
 - Possuem complexidade **polinomial**;
 - A maioria das vezes o **grau** do polinômio é **pequeno**: 1, 2, 3.
 - **Exemplos**: ordenar um vetor, obter a mediana de um vetor, árvore geradora mínima de um grafo, caminho mais curto em grafos, multiplicação de matrizes.
- Outros problemas **parecem ser mais desafiadores**: eles **não possuem** algoritmos eficientes **conhecidos**.

Objetivo – Intro

Suponha que você está diante de um **problema que você não conhece**. Você tentou diversas formas de resolver ele de forma eficiente, mas não teve sucesso. Como passou um tempo e o problema não está resolvido, seu chefe (no trabalho, na empresa, na academia) está lhe exigindo algum tipo de resultado:

“**Você avançou? Resolveu o problema?**”

O que você irá responder?

Objetivo – Intro

Suponha que você está diante de um **problema que você não conhece**. Você tentou diversas formas de resolver ele de forma eficiente, mas não teve sucesso. Como passou um tempo e o problema não está resolvido, seu chefe (no trabalho, na empresa, na academia) está lhe exigindo algum tipo de resultado:

“**Você avançou? Resolveu o problema?**”

O que você irá responder?

- “Eu não consegui fazer” parece uma resposta frágil.

Outra pessoa conseguirá resolver o problema?..

Objetivo – Intro

Suponha que você está diante de um **problema que você não conhece**. Você tentou diversas formas de resolver ele de forma eficiente, mas não teve sucesso. Como passou um tempo e o problema não está resolvido, seu chefe (no trabalho, na empresa, na academia) está lhe exigindo algum tipo de resultado:

“**Você avançou? Resolveu o problema?**”

O que você irá responder?

- “Eu não consegui fazer” parece uma resposta frágil.

Outra pessoa conseguirá resolver o problema?..

- Agora, se conseguimos mostrar que **ninguém consegue resolver este problema**, teremos realmente uma boa resposta.

Objetivo

Responder à pergunta:

- Como identificar um problema complexo?
- Como mostrar que um problema é complexo?

“Eu não consigo resolver” não é suficiente!

Objetivo – Intro

Idéia: catalogar os problemas em **pelo menos** duas classes:

- A classe de problemas para os quais **conhecemos** um algoritmo eficiente.
- A classe de problemas para os quais **não conhecemos** algoritmo eficiente.

Objetivo – Intro

Idéia: catalogar os problemas em **pelo menos** duas classes:

- A classe de problemas para os quais **conhecemos** um algoritmo eficiente.
- A classe de problemas para os quais **não conhecemos** algoritmo eficiente.

Tendo essas classes, podemos catalogar um novo problema como **complexo** se não conhecemos nenhum algoritmo eficiente para o problema, e ele está **de alguma forma** relacionado aos outros problemas da sua classe.

Objetivo – Intro

Idéia: catalogar os problemas em **pelo menos** duas classes:

- A classe de problemas para os quais **conhecemos** um algoritmo eficiente.
- A classe de problemas para os quais **não conhecemos** algoritmo eficiente.

Tendo essas classes, podemos catalogar um novo problema como **complexo** se não conhecemos nenhum algoritmo eficiente para o problema, e ele está **de alguma forma** relacionado aos outros problemas da sua classe.

Mas, como podemos analisar problemas para os quais **não** há algoritmo eficiente?
Há algo que podemos dizer sobre esses problemas? Algo que eles têm em comum?

Objetivo – Intro

Se não há algoritmo eficiente para um conjunto de problemas, tem **alguma característica comum** que possamos identificar nesses problemas?

Objetivo – Intro

Se não há algoritmo eficiente para um conjunto de problemas, tem **alguma característica comum** que possamos identificar nesses problemas?

SIM: para esses problemas, há algoritmos eficientes de **verificação**.

Objetivo – Intro

Se não há algoritmo eficiente para um conjunto de problemas, tem **alguma característica comum** que possamos identificar nesses problemas?

SIM: para esses problemas, há algoritmos eficientes de **verificação**.

Vamos evidenciar que, para **diversos problemas** que chamamos de complexos:

*É difícil encontrar um algoritmo polinomial que resolve o problema, mas **existe um algoritmo polinomial que verifica** se uma **proposta de solução** resolve de fato o problema.*

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Problemas de decisão**
- 3 Problema SAT
- 4 Algoritmos não determinísticos
- 5 Síntese

Simplificando o conceito de problema: Problemas de decisão

O estudo de classes de complexidade é feito para **problemas de decisão**, ou seja, aqueles em que a resposta é booleana: SIM ou NÃO.

- É mais fácil trabalhar dessa forma: não precisamos lidar com as diferentes estruturas de dados ou formatos que os algoritmos retornam.

Simplificando o conceito de problema: Problemas de decisão

O estudo de classes de complexidade é feito para **problemas de decisão**, ou seja, aqueles em que a resposta é booleana: SIM ou NÃO.

- É mais fácil trabalhar dessa forma: não precisamos lidar com as diferentes estruturas de dados ou formatos que os algoritmos retornam.
- A partir de qualquer problema **P**, podemos criar um **problema de decisão associado a ele**.

Simplificando o conceito de problema: Problemas de decisão

O estudo de classes de complexidade é feito para **problemas de decisão**, ou seja, aqueles em que a resposta é booleana: SIM ou NÃO.

- É mais fácil trabalhar dessa forma: não precisamos lidar com as diferentes estruturas de dados ou formatos que os algoritmos retornam.
- A partir de qualquer problema **P**, podemos criar um **problema de decisão associado a ele**.
- É possível mostrar que, se podemos resolver eficientemente um problema de decisão **DEC** associado a **P**, podemos resolver eficientemente o problema **P**.

Simplificando o conceito de problema: Problemas de decisão

O estudo de classes de complexidade é feito para **problemas de decisão**, ou seja, aqueles em que a resposta é booleana: SIM ou NÃO.

- É mais fácil trabalhar dessa forma: não precisamos lidar com as diferentes estruturas de dados ou formatos que os algoritmos retornam.
- A partir de qualquer problema **P**, podemos criar um **problema de decisão associado a ele**.
- É possível mostrar que, se podemos resolver eficientemente um problema de decisão **DEC** associado a **P**, podemos resolver eficientemente o problema **P**.
- Ou seja, o problema original que queremos resolver, e a sua versão de decisão são **polinomialmente equivalentes**: se há algoritmo eficiente para **P**, há algoritmo eficiente para **DEC**, e vice-versa.

Problemas de decisão

Exemplo de um problema de decisão:

Dado um grafo $G = (V, E)$, decidir se G é bipartido.

Problemas de decisão

Exemplo de um problema de decisão:

Dado um grafo $G = (V, E)$, decidir se G é bipartido.

A **instância** (entrada) é um grafo $G = (V, E)$.

A **saída** é booleana: **True/False**.

Problemas de decisão

Exemplo de um **problema de decisão**:

Dado um grafo $G = (V, E)$, decidir se G é bipartido.

A **instância** (entrada) é um grafo $G = (V, E)$.

A **saída** é booleana: **True/False**.

Outro exemplo de um **problema de decisão**:

*Dado um grafo conexo $G = (V, E)$, pesos inteiros w_e para cada aresta $e \in E$ e um valor inteiro W , decidir se G possui uma árvore geradora de peso **menor ou igual** que W .*

A **versão de otimização** pode ser resolvida eficientemente (**Kruskal, Prim**).

Problemas de decisão

Problema do clique (CLI)

Dado um grafo não-direcionado $G = (V, E)$, encontrar a maior clique em G .

Lembrando: Uma clique é um conjunto de vértices mutuamente adjacentes.

Problemas de decisão

Problema do clique (CLI)

Dado um grafo não-direcionado $G = (V, E)$, encontrar a maior clique em G .

Lembrando: Uma clique é um conjunto de vértices mutuamente adjacentes.

Problema do clique (CLI) – Versão de decisão

Dado um grafo não-direcionado $G = (V, E)$, e um valor k , decidir se G contém uma clique de tamanho pelo menos k .

Problemas de decisão

Problema do clique (CLI)

Dado um grafo não-direcionado $G = (V, E)$, encontrar a maior clique em G .

Lembrando: Uma clique é um conjunto de vértices mutuamente adjacentes.

Problema do clique (CLI) – Versão de decisão

Dado um grafo não-direcionado $G = (V, E)$, e um valor k , decidir se G contém uma clique de tamanho pelo menos k .

Observação: neste caso, se G contém uma clique de tamanho pelo menos k , ele contém uma clique de tamanho k .

Problemas de decisão

O problema **P** que queremos resolver, e a sua versão de decisão são, *em certa medida*, equivalentes:

- Por exemplo, a redução de **CLI** para **CLI_DEC** é **trivial**.

Problemas de decisão

O problema **P** que queremos resolver, e a sua versão de decisão são, *em certa medida*, equivalentes:

- Por exemplo, a redução de **CLI** para **CLI_DEC** é **trivial**.
- Suponha que você possui um algoritmo eficiente (polinomial) para a versão de decisão **CLI_DEC** do problema **CLI**.

Problemas de decisão

O problema **P** que queremos resolver, e a sua versão de decisão são, *em certa medida*, equivalentes:

- Por exemplo, a redução de **CLI** para **CLI_DEC** é **trivial**.
- Suponha que você possui um algoritmo eficiente (polinomial) para a versão de decisão **CLI_DEC** do problema **CLI**.

Se há algoritmo polinomial para a versão de decisão do problema **P**,
há algoritmo polinomial para o problema **P**.

Problemas de decisão

O problema **P** que queremos resolver, e a sua versão de decisão são, *em certa medida*, equivalentes:

- Por exemplo, a redução de **CLI** para **CLI_DEC** é **trivial**.
- Suponha que você possui um algoritmo eficiente (polinomial) para a versão de decisão **CLI_DEC** do problema **CLI**.

Se há algoritmo polinomial para a versão de decisão do problema **P**,
há algoritmo polinomial para o problema **P**.

- O algoritmo a seguir resolve **CLI** em tempo polinomial.

Problemas de decisão

Alg_CLI (G)

```
1:  $S_{min} \leftarrow 1$ 
2:  $S_{max} \leftarrow |V|$ 
3:  $opt \leftarrow 1$ 
4: enquanto  $S_{min} \leq S_{max}$  faça
5:    $S_{med} = (S_{min} + S_{max})/2$ 
6:   se Alg_CLI_DEC(G,  $S_{med}$ ) então
7:      $opt \leftarrow S_{med}$ 
8:      $S_{min} = S_{med} + 1$ 
9:   senão  $S_{max} = S_{med} - 1$ 
10: devolva  $opt$ 
```

Problemas de decisão

Alg_CLI (G)

```
1:  $S_{min} \leftarrow 1$ 
2:  $S_{max} \leftarrow |V|$ 
3:  $opt \leftarrow 1$ 
4: enquanto  $S_{min} \leq S_{max}$  faça
5:    $S_{med} = (S_{min} + S_{max})/2$ 
6:   se Alg_CLI_DEC(G,  $S_{med}$ ) então
7:      $opt \leftarrow S_{med}$ 
8:      $S_{min} = S_{med} + 1$ 
9:   senão  $S_{max} = S_{med} - 1$ 
10: devolva  $opt$ 
```

Este tipo de redução é chamado de “**redução de Turing**”.

Problemas de decisão



Sempre é possível encontrar uma **redução polinomial (de Turing)** do problema de otimização para o problema de decisão: $\text{OPT}_P \propto_{\text{poli}} \text{DEC}_P$, para todo problema P .

Problemas de decisão



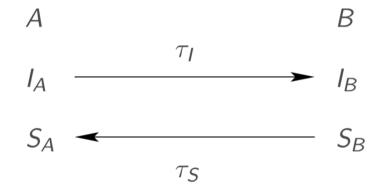
Sempre é possível encontrar uma **redução polinomial (de Turing)** do problema de otimização para o problema de decisão: $\text{OPT}_P \propto_{\text{poli}} \text{DEC}_P$, para todo problema **P**.

Veja que, neste caso, precisamos usar outro tipo de redução:

Redução polinomial (de Turing).

Porém, continua valendo que a versão de otimização **OPT** e a versão de decisão **DEC** do mesmo problema, são **polinomialmente equivalentes**: se há algoritmo polinomial para um problema, há para o outro.

Tipos de reduções



- **Redução:** Alg_B é usado uma única vez. Caso particular da **redução de Turing**.
- **Redução de Karp:** Alg_B é usado uma única vez. **A** e **B** precisam ser problemas de decisão. Alg_B deve responder SIM para I_B **se e somente se** I_A for uma instância SIM para **A**. Caso particular da **redução de Turing**.
- **Redução de Turing:** Alg_B pode ser usado múltiplas vezes. Se a redução é polinomial e o número de chamadas de Alg_B é polinomial (no tamanho da entrada de **A**), então se Alg_B é polinomial, há algoritmo polinomial para **A**.

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Problemas de decisão
- 3 Problema SAT**
- 4 Algoritmos não determinísticos
- 5 Síntese

Problema SAT

Um exemplo **paradigmático e representativo** de problemas difíceis é o problema de satisfazer uma fórmula lógica \mathcal{F} na sua forma normal conjuntiva (**SAT**, *Satisfiability*).

Problema SAT

Um exemplo **paradigmático e representativo** de problemas difíceis é o problema de satisfazer uma fórmula lógica \mathcal{F} na sua forma normal conjuntiva (**SAT**, *Satisfiability*).

Lembrando:

- As variáveis x_1, \dots, x_n são binárias, e as suas negações são denotadas \bar{x}_i ;
- Os operadores lógicos são “+” ($\underline{\text{OU}}$ lógico) e “.” ($\underline{\text{E}}$ lógico);
- As cláusulas são C_1, C_2, \dots, C_m , cada uma da forma $C_i = x_{i1} + x_{i2} + \dots$;
- Fórmula na forma normal conjuntiva (FNC): $\mathcal{F} = C_1 \cdot C_2 \cdot \dots \cdot C_m$.

Problema SAT

Um exemplo **paradigmático e representativo** de problemas difíceis é o problema de satisfazer uma fórmula lógica \mathcal{F} na sua forma normal conjuntiva (**SAT**, *Satisfiability*).

Lembrando:

- As variáveis x_1, \dots, x_n são binárias, e as suas negações são denotadas \bar{x}_i ;
- Os operadores lógicos são “+” (OU lógico) e “.” (E lógico);
- As cláusulas são C_1, C_2, \dots, C_m , cada uma da forma $C_i = x_{i1} + x_{i2} + \dots$;
- Fórmula na forma normal conjuntiva (FNC): $\mathcal{F} = C_1 \cdot C_2 \cdot \dots \cdot C_m$.

Exemplo de fórmula na FNC: $\mathcal{F} = (x_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + x_3) \cdot (x_1 + \bar{x}_3)$.

Problema SAT

Um exemplo **paradigmático e representativo** de problemas difíceis é o problema de satisfazer uma fórmula lógica \mathcal{F} na sua forma normal conjuntiva (**SAT**, *Satisfiability*).

Lembrando:

- As variáveis x_1, \dots, x_n são binárias, e as suas negações são denotadas \bar{x}_i ;
- Os operadores lógicos são “+” (OU lógico) e “.” (E lógico);
- As cláusulas são C_1, C_2, \dots, C_m , cada uma da forma $C_i = x_{i1} + x_{i2} + \dots$;
- Fórmula na forma normal conjuntiva (FNC): $\mathcal{F} = C_1 \cdot C_2 \cdot \dots \cdot C_m$.

Exemplo de fórmula na FNC: $\mathcal{F} = (x_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + x_3) \cdot (x_1 + \bar{x}_3)$.

Em lógica, qualquer fórmula proposicional pode ser transformada em uma fórmula equivalente que está na forma normal conjuntiva. Resolver **SAT** significa saber verificar se **qualquer fórmula** da lógica proposicional pode ser verdadeira.

Problema SAT

Problema da satisfatibilidade (SAT)

Entrada: Fórmula lógica \mathcal{F} com n variáveis na forma normal conjuntiva.

Objetivo: Decidir se existe uma atribuição das variáveis x_1, \dots, x_n para a qual $\mathcal{F} = 1$.

Problema SAT

Problema da satisfatibilidade (SAT)

Entrada: Fórmula lógica \mathcal{F} com n variáveis na forma normal conjuntiva.

Objetivo: Decidir se existe uma atribuição das variáveis x_1, \dots, x_n para a qual $\mathcal{F} = 1$.

Observações:

- Ou seja, precisamos atribuir valores às variáveis de forma que, para esses valores, a fórmula seja verdadeira. **SAT** é um **problema de decisão**.

Problema SAT

Problema da satisfatibilidade (SAT)

Entrada: Fórmula lógica \mathcal{F} com n variáveis na forma normal conjuntiva.

Objetivo: Decidir se existe uma atribuição das variáveis x_1, \dots, x_n para a qual $\mathcal{F} = 1$.

Observações:

- Ou seja, precisamos atribuir valores às variáveis de forma que, para esses valores, a fórmula seja verdadeira. **SAT** é um **problema de decisão**.
- Exemplo: $\mathcal{F} = (x_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + x_3) \cdot (x_1 + \bar{x}_3)$.
Se $x_1 = 1$ e $x_2 = x_3 = 0$, temos $\mathcal{F} = 1$. A saída do **SAT** para esta instância é SIM.

Problema SAT

Problema da satisfatibilidade (SAT)

Entrada: Fórmula lógica \mathcal{F} com n variáveis na forma normal conjuntiva.

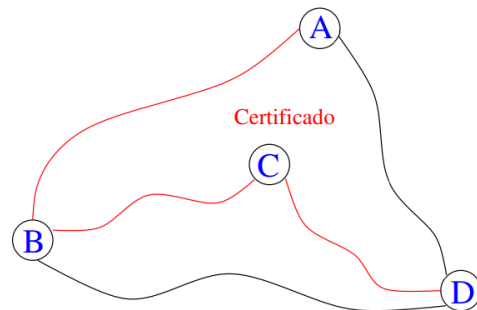
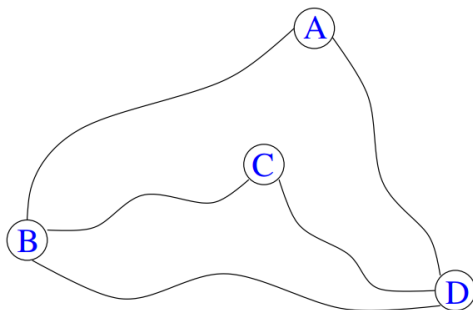
Objetivo: Decidir se existe uma atribuição das variáveis x_1, \dots, x_n para a qual $\mathcal{F} = 1$.

Observações:

- Ou seja, precisamos atribuir valores às variáveis de forma que, para esses valores, a fórmula seja verdadeira. **SAT** é um **problema de decisão**.
- Exemplo: $\mathcal{F} = (x_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + x_3) \cdot (x_1 + \bar{x}_3)$.
Se $x_1 = 1$ e $x_2 = x_3 = 0$, temos $\mathcal{F} = 1$. A saída do **SAT** para esta instância é SIM.
- A **proposta de solução** ($x_1 = 1, x_2 = x_3 = 0$) é o que vamos chamar de **certificado** da instância. É uma especie de “prova” do fato que a instância é SIM.

Certificado: outro exemplo

Problema dos caminhos mínimos de fonte única.



Problema SAT

- **Tarefa:** Encontre um algoritmo para **SAT**.

Problema SAT

- **Tarefa:** Encontre um algoritmo para **SAT**.

O seu algoritmo é **polinomial**?

Problema SAT

- **Tarefa:** Encontre um algoritmo para **SAT**.

O seu algoritmo é **polinomial**?

- **Tarefa:** Dado um certificado (atribuição de valores às variáveis da instância **SAT**), encontre um algoritmo que verifica se \mathcal{F} é verdadeira ou falsa para esta atribuição.

Problema SAT

- **Tarefa:** Encontre um algoritmo para **SAT**.

O seu algoritmo é **polinomial**?

- **Tarefa:** Dado um certificado (atribuição de valores às variáveis da instância **SAT**), encontre um algoritmo que verifica se \mathcal{F} é verdadeira ou falsa para esta atribuição.

O seu algoritmo é **polinomial**?

Problema SAT

- **Tarefa:** Encontre um algoritmo para **SAT**.

O seu algoritmo é **polinomial**?

- **Tarefa:** Dado um certificado (atribuição de valores às variáveis da instância **SAT**), encontre um algoritmo que verifica se \mathcal{F} é verdadeira ou falsa para esta atribuição.

O seu algoritmo é **polinomial**?

Não se conhece algoritmo eficiente para **SAT**. Porém, existe um **algoritmo polinomial** que verifica se uma atribuição de valores para as variáveis resolve de fato o problema.

Problema SAT

- **Conclusão:** não é fácil achar um algoritmo polinomial que resolve **SAT**.
- **Mas**, dada uma proposta de solução (**certificado**) para uma instância do **SAT**, há algoritmo polinomial que verifica se a solução indica que a instância é SIM.

Problema SAT

- **Conclusão:** não é fácil achar um algoritmo polinomial que resolve **SAT**.
- **Mas**, dada uma proposta de solução (**certificado**) para uma instância do **SAT**, há algoritmo polinomial que verifica se a solução indica que a instância é SIM.
- Além do **SAT**, **muitos** outros problemas compartilham essa mesma propriedade.
- Vamos introduzir um *novo modelo de computação* que nos ajudará a identificá-los.

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Problemas de decisão
- 3 Problema SAT
- 4 Algoritmos não determinísticos**
- 5 Síntese

Algoritmos não-determinísticos

- Em um algoritmo **determinístico**, o resultado de cada operação é definido de maneira única. Cada linha do algoritmo computa ou retorna apenas um valor.

Algoritmos não-determinísticos

- Em um algoritmo **determinístico**, o resultado de cada operação é definido de maneira única. Cada linha do algoritmo computa ou retorna apenas um valor.
- Um algoritmo **não-determinístico**, além dos comandos determinísticos usuais, pode usar o comando **Escolha(S)**, onde S é um conjunto de elementos.

Algoritmos não-determinísticos

- Em um algoritmo **determinístico**, o resultado de cada operação é definido de maneira única. Cada linha do algoritmo computa ou retorna apenas um valor.
- Um algoritmo **não-determinístico**, além dos comandos determinísticos usuais, pode usar o comando **Escolha(S)**, onde S é um conjunto de elementos.
- A complexidade de execução do comando **Escolha** é $O(1)$.

Algoritmos não-determinísticos

- Em um algoritmo **determinístico**, o resultado de cada operação é definido de maneira única. Cada linha do algoritmo computa ou retorna apenas um valor.
- Um algoritmo **não-determinístico**, além dos comandos determinísticos usuais, pode usar o comando **Escolha(S)**, onde S é um conjunto de elementos.
- A complexidade de execução do comando **Escolha** é $O(1)$.
- Não existe regra que especifique o funcionamento do comando **Escolha(S)**. Existem $|S|$ resultados possíveis para esta operação.

Algoritmos não-determinísticos

Geralmente, os algoritmos **não-determinísticos** são divididos em **duas fases**:

- Na primeira fase, é possível usar o comando não-determinístico **Escolha**, e uma proposta de solução é construída.
 - A **proposta de solução** gerada ao final da fase de construção do algoritmo não determinístico é chamada de **certificado**.

Algoritmos não-determinísticos

Geralmente, os algoritmos **não-determinísticos** são divididos em **duas fases**:

- Na primeira fase, é possível usar o comando não-determinístico **Escolha**, e uma proposta de solução é construída.
 - A **proposta de solução** gerada ao final da fase de construção do algoritmo não determinístico é chamada de **certificado**.
- A segunda fase apenas usa comandos determinísticos, e simplesmente **verifica** se o certificado resolve de fato o problema, retornando o resultado SIM ou NÃO.
 - Resumo: se o certificado resolve o problema, retornar SIM, senão retornar NÃO.
 - Se na fase de **verificação**, um determinado certificado de uma instância do problema retorna SIM, o chamamos de **certificado válido** dessa instância.

Algoritmos não-determinísticos

Geralmente, os algoritmos **não-determinísticos** são divididos em **duas fases**:

- Na primeira fase, é possível usar o comando não-determinístico **Escolha**, e uma proposta de solução é construída.
 - A **proposta de solução** gerada ao final da fase de construção do algoritmo não determinístico é chamada de **certificado**.
- A segunda fase apenas usa comandos determinísticos, e simplesmente **verifica** se o certificado resolve de fato o problema, retornando o resultado SIM ou NÃO.
 - Resumo: se o certificado resolve o problema, retornar SIM, senão retornar NÃO.
 - Se na fase de **verificação**, um determinado certificado de uma instância do problema retorna SIM, o chamamos de **certificado válido** dessa instância.
- A divisão em fases é apenas uma formalidade para facilitar a nossa análise. O que define que um algoritmo seja determinístico ou não determinístico é a sua capacidade de usar o comando **Escolha**.

Algoritmos não-determinísticos

- Uma **máquina não-determinística** é aquela que é capaz de executar um algoritmo não-determinístico. *É uma abstração!*
- Porém, os recentes progressos na computação quântica mostram que abstrações podem virar realidade, expandindo a capacidade dos algoritmos puramente determinísticos.
- Muitos avanços tecnológicos começaram como abstrações teóricas.

Algoritmos não-determinísticos: Exemplo

Exemplo: Determinar se um valor x pertence a um vetor A de n posições.

Algoritmos não-determinísticos: Exemplo

Exemplo: Determinar se um valor x pertence a um vetor A de n posições.
Um algoritmo não-determinístico seria:

BuscaND (A, x)

- 1: ▷ (* Fase de construção *)
 - 2: $j \leftarrow$ Escolha($\{1, \dots, n\}$)
 - 3: ▷ (* Fase de verificação *)
 - 4: se $A[j] = x$ então devolva SIM
 - 5: senão devolva NÃO
-

Qual a **complexidade** deste algoritmo?

Algoritmos não-determinísticos

Definição: O **tempo de execução** de um algoritmo **não-determinístico** para uma determinada instância é T , se existe uma sequência de comandos **Escolha** tal que o número de operações necessário para que ele retorne SIM é T , e não há outra sequência de comandos **Escolha** com um tempo menor que T .

- Para cada instância de tamanho n , pegamos a melhor sequência de **Escolha**'s. Para todas as instâncias de tamanho n , a complexidade é avaliada pelo tempo de execução de pior caso, como sempre fazemos.

Algoritmos não-determinísticos

Definição: O **tempo de execução** de um algoritmo **não-determinístico** para uma determinada instância é T , se existe uma sequência de comandos **Escolha** tal que o número de operações necessário para que ele retorne SIM é T , e não há outra sequência de comandos **Escolha** com um tempo menor que T .

- Para cada instância de tamanho n , pegamos a melhor sequência de **Escolha**'s. Para todas as instâncias de tamanho n , a complexidade é avaliada pelo tempo de execução de pior caso, como sempre fazemos.
- Um algoritmo **não-determinístico** tem complexidade $O(f(n))$ se existem constantes positivas c e n_0 tais que, para toda instância SIM de tamanho $n \geq n_0$, o tempo de execução é limitado por $cf(n)$.

Algoritmos não-determinísticos

Definição: O **tempo de execução** de um algoritmo **não-determinístico** para uma determinada instância é T , se existe uma sequência de comandos **Escolha** tal que o número de operações necessário para que ele retorne SIM é T , e não há outra sequência de comandos **Escolha** com um tempo menor que T .

- Para cada instância de tamanho n , pegamos a melhor sequência de **Escolha**'s. Para todas as instâncias de tamanho n , a complexidade é avaliada pelo tempo de execução de pior caso, como sempre fazemos.
- Um algoritmo **não-determinístico** tem complexidade $O(f(n))$ se existem constantes positivas c e n_0 tais que, para toda instância SIM de tamanho $n \geq n_0$, o tempo de execução é limitado por $cf(n)$.
- **Se existe** uma sequência de **Escolha**'s que leve o algoritmo a retornar SIM, a sua complexidade está definida. E se **não existe** essa sequência?

Algoritmos não-determinísticos

Definição: O **tempo de execução** de um algoritmo **não-determinístico** para uma determinada instância é T , se existe uma sequência de comandos **Escolha** tal que o número de operações necessário para que ele retorne SIM é T , e não há outra sequência de comandos **Escolha** com um tempo menor que T .

- Para cada instância de tamanho n , pegamos a melhor sequência de **Escolha**'s. Para todas as instâncias de tamanho n , a complexidade é avaliada pelo tempo de execução de pior caso, como sempre fazemos.
- Um algoritmo **não-determinístico** tem complexidade $O(f(n))$ se existem constantes positivas c e n_0 tais que, para toda instância SIM de tamanho $n \geq n_0$, o tempo de execução é limitado por $cf(n)$.
- **Se existe** uma sequência de **Escolha**'s que leve o algoritmo a retornar SIM, a sua complexidade está definida. E se **não existe** essa sequência?
- **BuscaND** tem complexidade $O(1)$. Qualquer algoritmo determinístico para este problema é $\Omega(n)$.

Algoritmos não-determinísticos

Problema do clique (CLI)

Dado um grafo não-direcionado $G = (V, E)$, encontrar a maior clique em G .

Lembrando: Uma clique é um conjunto de vértices mutuamente adjacentes.

Problema do clique (CLI) – Versão de decisão

Dado um grafo não-direcionado $G = (V, E)$, e um valor k , decidir se G contém uma clique de tamanho pelo menos k .

Algoritmos não-determinísticos: Outro exemplo

Algoritmo não-determinístico para o problema do clique (**CLI**):

CliqueND(G, k)

- 1: ▷ (* Fase de construção *)
- 2: $S \leftarrow V$
- 3: $C \leftarrow \{\}$ ▷ vértices que estarão na clique
- 4: **para cada** $i \leftarrow 1, \dots, k$ **faça**
- 5: $u \leftarrow \text{Escolha}(S)$
- 6: $S \leftarrow S - \{u\}$
- 7: $C \leftarrow C \cup \{u\}$
- 8: ▷ (* Fase de verificação *)
- 9: **para cada** par de vértices distintos u, v em C **faça**
- 10: **se** $(u, v) \notin E$ **então devolva** NÃO
- 11: **devolva** SIM

Algoritmos não-determinísticos: Outro exemplo

- Complexidade não-determinística do algoritmo: $O(k + k^2) \subseteq O(n^2)$.

Algoritmos não-determinísticos: Outro exemplo

- Complexidade não-determinística do algoritmo: $O(k + k^2) \subseteq O(n^2)$.
- Não se conhece algoritmo determinístico polinomial para **CLI**.

Máquinas não-determinísticas

- As máquinas não-determinísticas podem ser **imaginadas** como máquinas determinísticas com **infinitos processadores**.
 - Os processadores se comunicam entre *instantaneamente*, ou seja, uma mensagem vai de um processador ao outro em tempo *zero*.

Máquinas não-determinísticas

- As máquinas não-determinísticas podem ser **imaginadas** como máquinas determinísticas com **infinitos processadores**.
 - Os processadores se comunicam entre *instantaneamente*, ou seja, uma mensagem vai de um processador ao outro em tempo *zero*.
- O fluxo de execução de um algoritmo não-determinístico pode ser representado através de uma *árvore*. Cada caminho na árvore iniciando na raiz corresponde a uma sequência de escolhas e a um possível fluxo de execução do programa.

Máquinas não-determinísticas

- As máquinas não-determinísticas podem ser **imaginadas** como máquinas determinísticas com **infinitos processadores**.
 - Os processadores se comunicam entre *instantaneamente*, ou seja, uma mensagem vai de um processador ao outro em tempo *zero*.
- O fluxo de execução de um algoritmo não-determinístico pode ser representado através de uma *árvore*. Cada caminho na árvore iniciando na raiz corresponde a uma sequência de escolhas e a um possível fluxo de execução do programa.
- Em um dado vértice, ao executar o comando **Escolha**(S), são criados $|S|$ filhos, cada um correspondendo a um possível resultado desta operação, alocando-se novos processadores para continuar a execução.

Máquinas não-determinísticas

- Podemos imaginar que a árvore de execução é percorrida em largura. Ao ser atingido o primeiro nível onde uma execução do algoritmo retorna SIM, o processador que chegou a este estado comunica-se instantaneamente com todos os demais, interrompendo o algoritmo.

Máquinas não-determinísticas

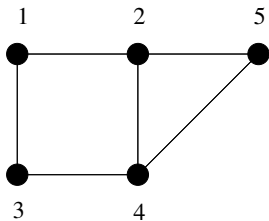
- Podemos imaginar que a árvore de execução é percorrida em largura. Ao ser atingido o primeiro nível onde uma execução do algoritmo retorna SIM, o processador que chegou a este estado comunica-se instantaneamente com todos os demais, interrompendo o algoritmo.
- Outro algoritmo não-determinístico de complexidade $O(n^2)$ para **CLI** é apresentado a seguir. Note que existem sequências de escolhas que podem não deixar que o laço **enquanto** termine. Porém, a complexidade não-determinística só se interessa pela **melhor** sequência de escolhas que leva a retornar SIM.

Algoritmos não-determinísticos

CliqueND2(G, k)

```
1: ▷ (* Fase de construção *)
2:  $j \leftarrow 0$ 
3:  $C \leftarrow \{\}$  ▷ vértices que estarão na clique
4: enquanto  $j < k$  faça
5:    $u \leftarrow \text{Escolha}(V)$ 
6:   se  $u \notin C$  então
7:      $C \leftarrow C \cup \{u\}$ 
8:      $j \leftarrow j + 1$ 
9: ▷ (* Fase de verificação *)
10: para cada par de vértices distintos  $u, v$  em  $C$  faça
11:   se  $(u, v) \notin E$  então devolva NÃO
12: devolva SIM
```

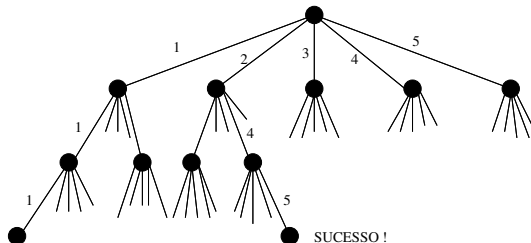
Algoritmos não-determinísticos



Determinar se há uma **CLI** de tamanho 3.

A graph with 5 vertices labeled 1 through 5. Vertices 1, 2, 3, and 4 form a square. Vertex 5 is connected to vertices 2 and 4, forming a triangle with vertex 2.

A seguir, a árvore de simulação determinística de CliqueND2:



Algoritmos não-determinísticos

Exercício: Desenvolva um algoritmo não-determinístico polinomial para **SAT**.

Qual a complexidade do seu algoritmo?

Classes \mathcal{P} e \mathcal{NP}

- **Definição:** \mathcal{P} é o conjunto de problemas que podem ser resolvidos por um **algoritmo determinístico** em tempo **polinomial**.
- **Definição:** \mathcal{NP} é o conjunto de problemas que podem ser resolvidos por um **algoritmo não-determinístico** em tempo **polinomial**.

Classes \mathcal{P} e \mathcal{NP}

- **Definição:** \mathcal{P} é o conjunto de problemas que podem ser resolvidos por um **algoritmo determinístico** em tempo **polinomial**.
- **Definição:** \mathcal{NP} é o conjunto de problemas que podem ser resolvidos por um **algoritmo não-determinístico** em tempo **polinomial**.
- Como todo algoritmo determinístico é um caso particular de um algoritmo não-determinístico, segue que

$$\mathcal{P} \subseteq \mathcal{NP}.$$

Todos os problemas que possuem algoritmos polinomiais estão em \mathcal{P} e \mathcal{NP} .
Vimos que **CLIQUE** e **SAT** estão em \mathcal{NP} .

Classes P e NP

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

Classes P e NP

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

- A maioria dos cientistas acredita que $\mathcal{P} \neq \mathcal{NP}$. Há algumas exceções notáveis!

Classes P e NP

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

- A maioria dos cientistas acredita que $\mathcal{P} \neq \mathcal{NP}$. Há algumas exceções notáveis!
- Como mostrar que $\mathcal{P} \neq \mathcal{NP}$?
*Mostrar que existe um problema $A \in \mathcal{NP}$ tal que **nenhum** algoritmo determinístico polinomial pode resolver A.*

Classes P e NP

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

- A maioria dos cientistas acredita que $\mathcal{P} \neq \mathcal{NP}$. Há algumas exceções notáveis!
- Como mostrar que $\mathcal{P} \neq \mathcal{NP}$?
*Mostrar que existe um problema $A \in \mathcal{NP}$ tal que **nenhum** algoritmo determinístico polinomial pode resolver A.*
- Como mostrar que $\mathcal{P} = \mathcal{NP}$?
*Mostrar que todo problema $B \in \mathcal{NP}$ possui um algoritmo **determinístico polinomial** que o resolve.*

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Problemas de decisão
- 3 Problema SAT
- 4 Algoritmos não determinísticos
- 5 Síntese

Síntese

- O estudo de classes de complexidade é feito para **problemas de decisão**: aqueles em que a saída é SIM ou NÃO.
- Um exemplo **paradigmático e representativo** de problemas difíceis é o problema da satisfatibilidade (**SAT**, *Satisfiability*). Na próxima aula, usaremos **SAT** para provar que diferentes problemas são complexos.
- Um algoritmo **não-determinístico** permite o uso do comando **Escolha(S)**. O tempo de execução do algoritmo não-determinístico está ligado à melhor sequência possível de **Escolha's**.

Material bibliográfico e exercícios

U. Manber. Introduction to Algorithms. – **Cap. 11.**

Exercícios: ver exercícios no final do Capítulo 11.

Dúvidas

Dúvidas?