Computabilidade

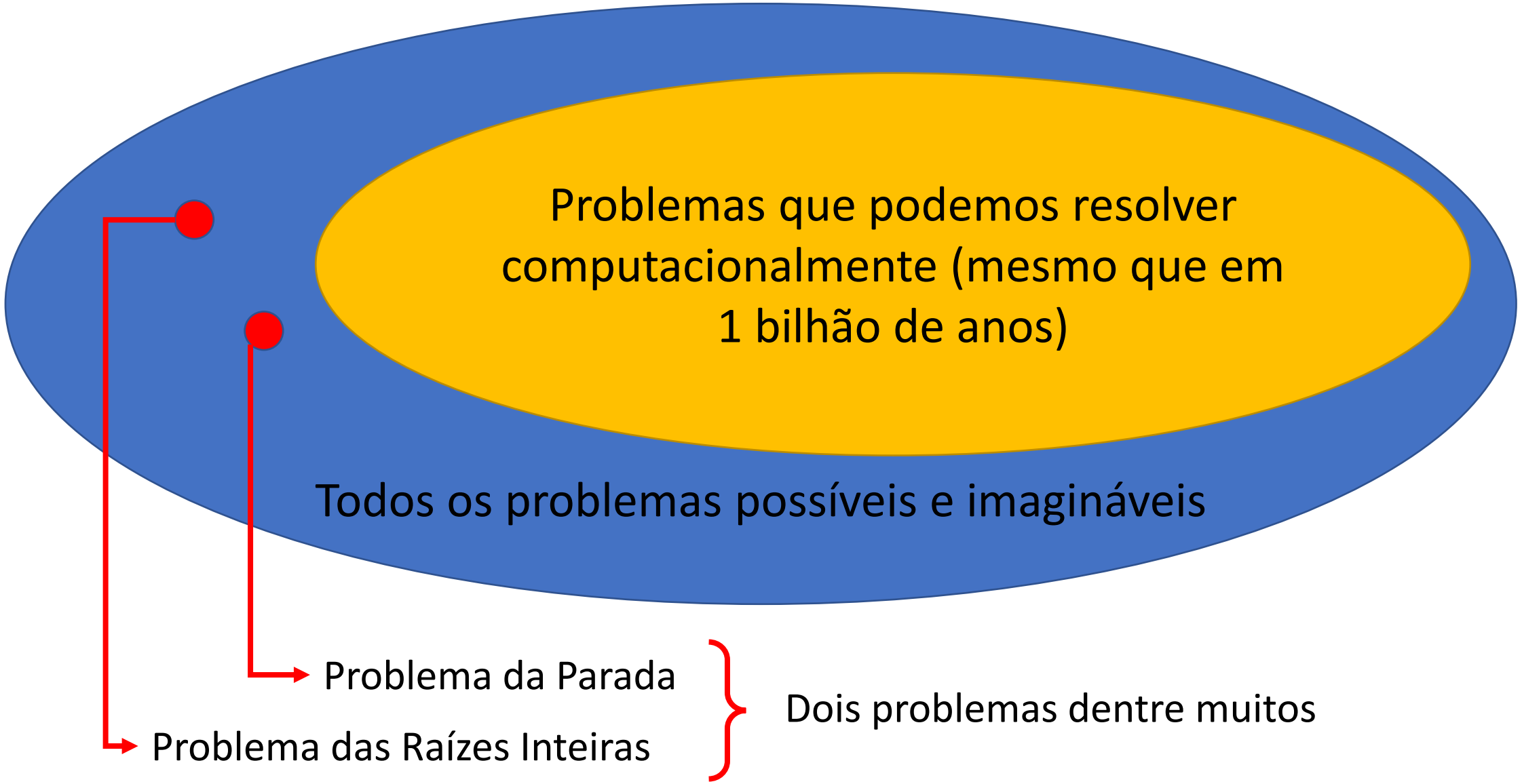
Aula 6 – Problemas NP-Completo

Prof. Rodrigo Mafort

Últimas aulas

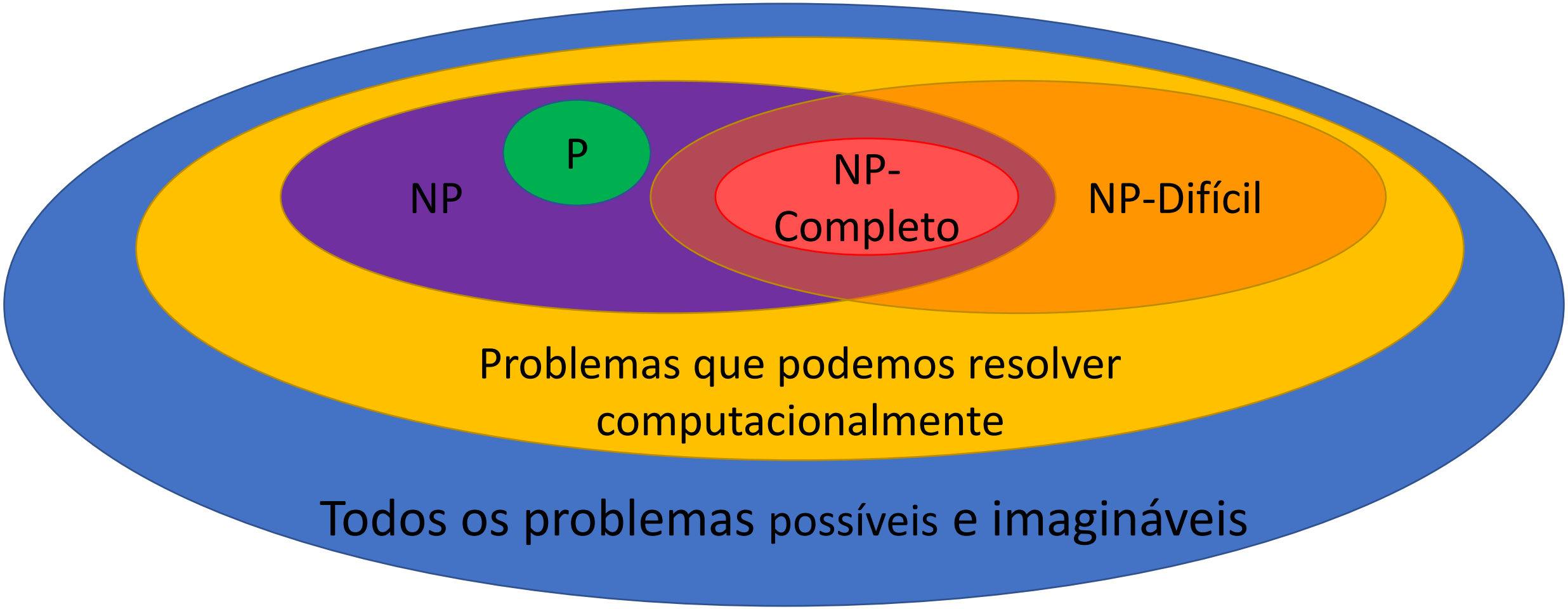
- Revisão de Complexidade Computacional
- Revisão de Autômatos
- Máquinas de Turing
- Problemas Indecidíveis Computacionalmente (Problema da Parada)
- Classes de Problemas: P, NP, NP-Difícil e NP-Completo

Universo de Problemas



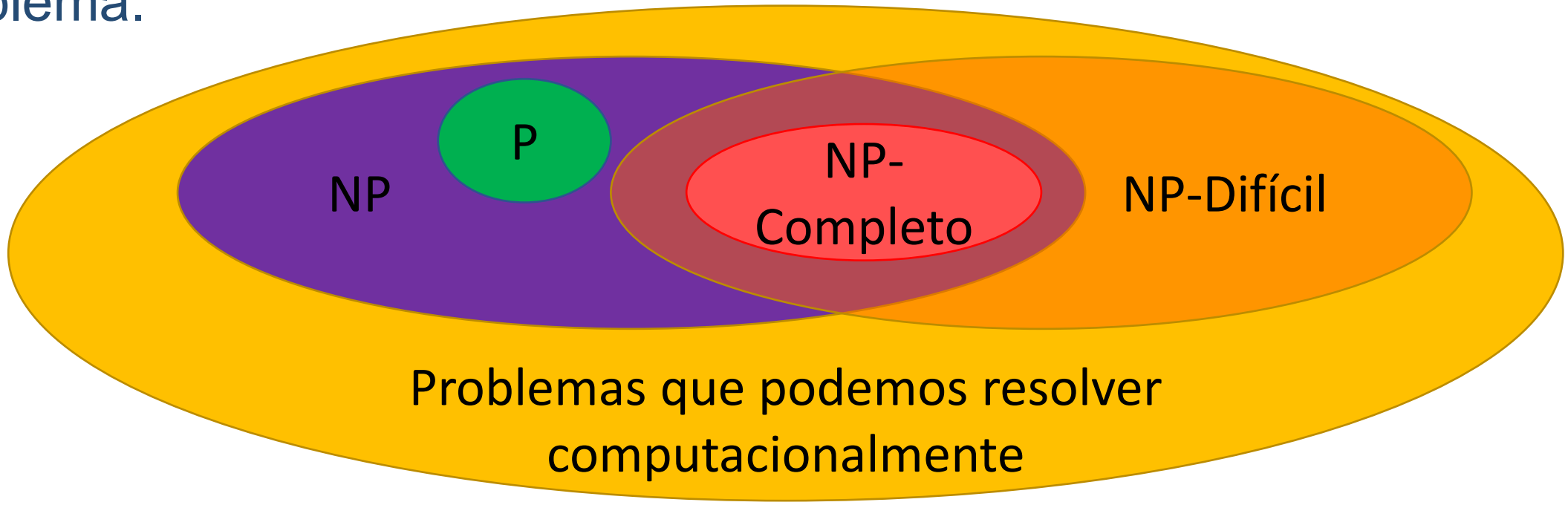
Universo de Problemas

- Podemos classificar os problemas contidos no conjunto de problemas que podemos resolver computacionalmente.



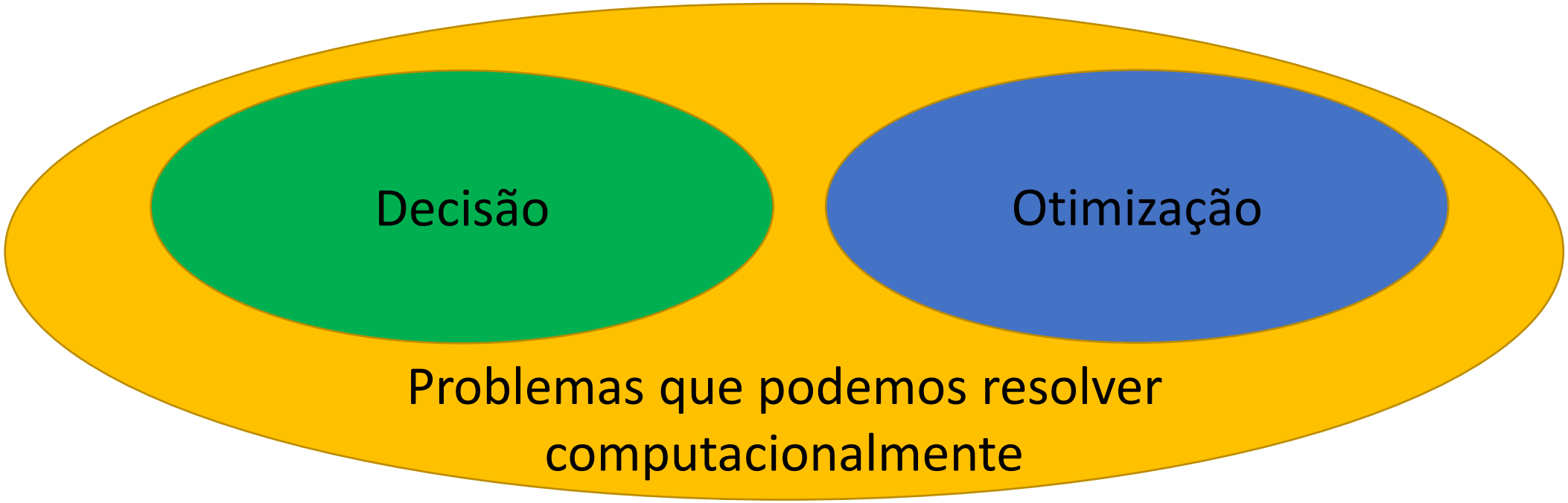
Classes de Problemas

- Essa classificação pode ser realizada em função da complexidade de tempo ou da complexidade de espaço.
- Em geral, os problemas são classificados em função da complexidade de tempo do melhor algoritmo já encontrado para esse problema.



Problemas de Decisão e Otimização

- Dentre o universo de problemas que podemos resolver computacionalmente, temos uma divisão importante:
  - Problemas de Decisão
  - Problemas de Otimização



Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o mapa ao lado usando apenas $k = 5$ cores de forma que dois países vizinhos não tenham cores iguais?

Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o mapa ao lado usando apenas $k = 5$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: SIM

Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o mapa ao lado usando apenas $k = 4$ cores de forma que dois países vizinhos não tenham cores iguais?

Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o mapa ao lado usando apenas $k = 4$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: SIM

Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o mapa ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

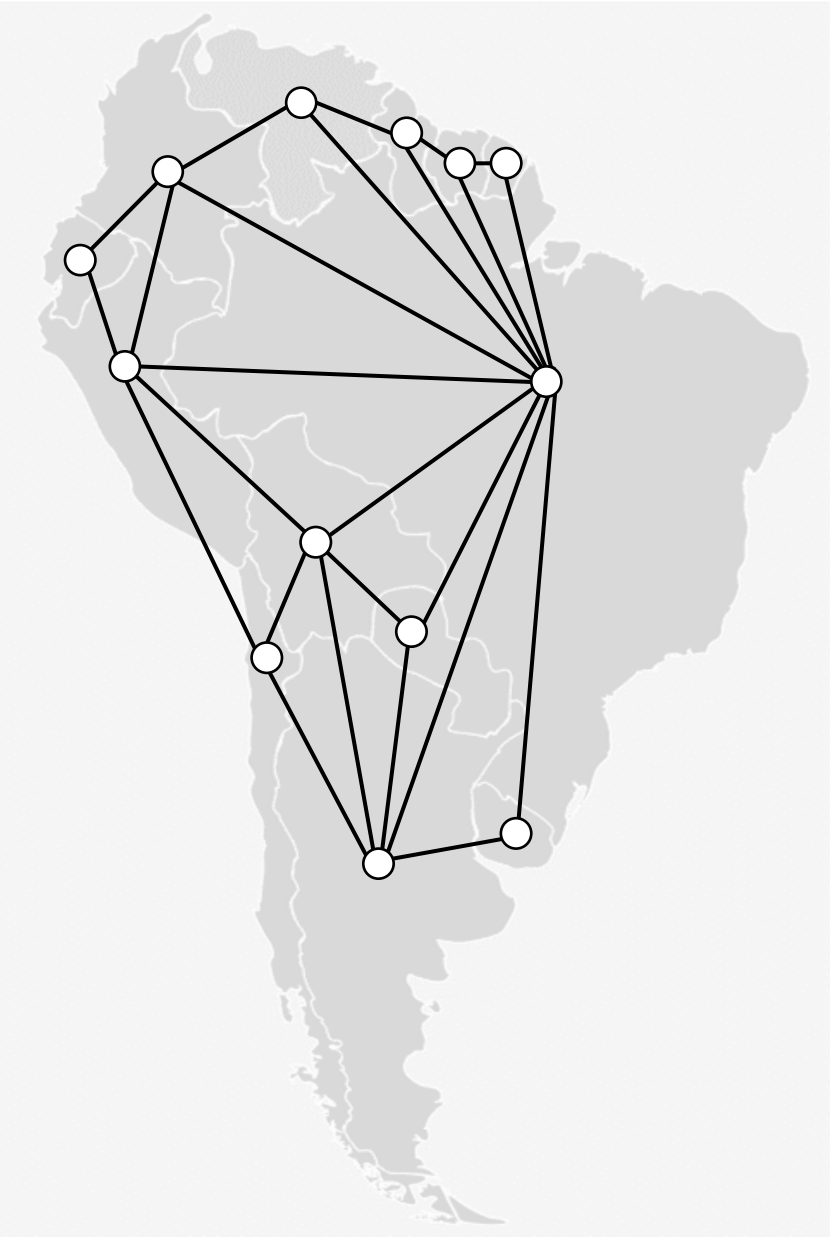
  - Resposta: ???

Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o GRAFO ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: ???

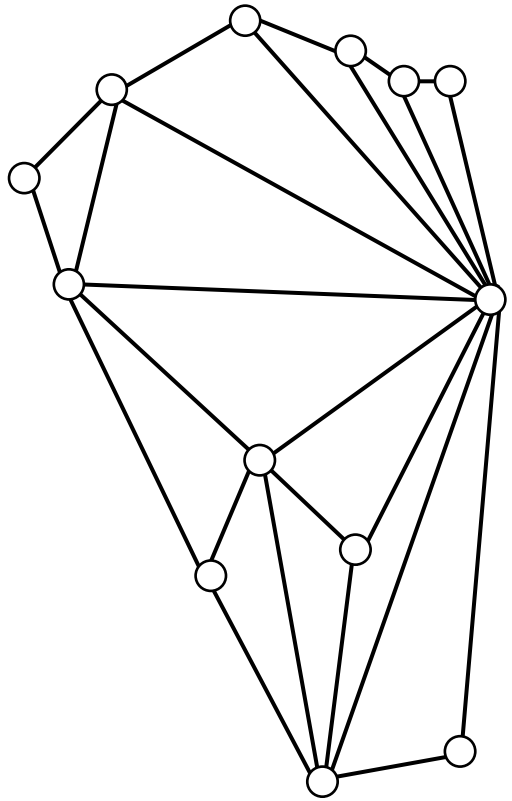


Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o GRAFO ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: ???

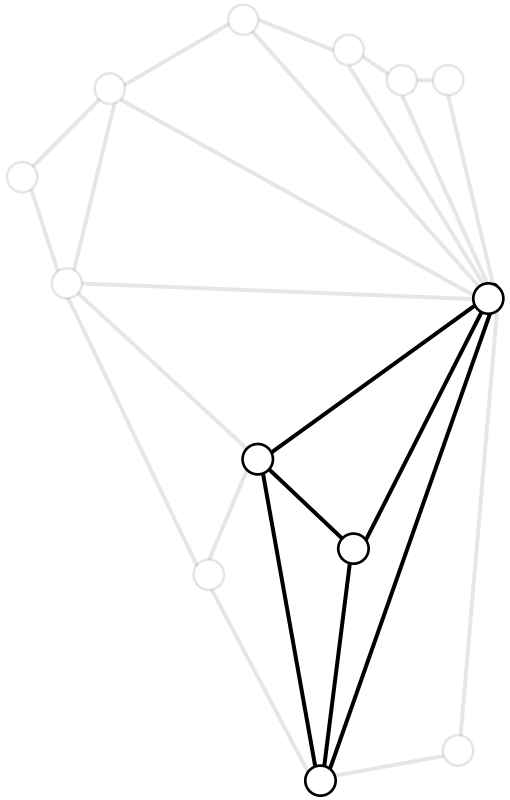


Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o GRAFO ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: ???

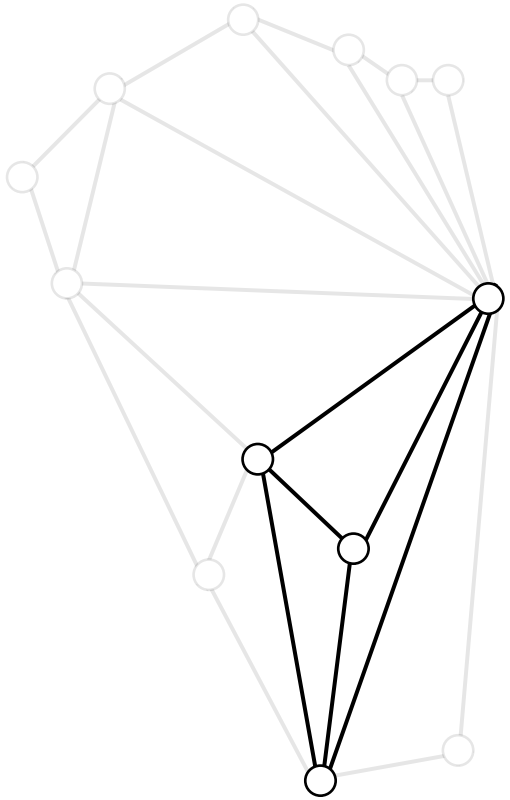


Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o GRAFO ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: Não! Essa parte do grafo requer no mínimo 4 cores, pois cada vértice é vizinho dos outros três.

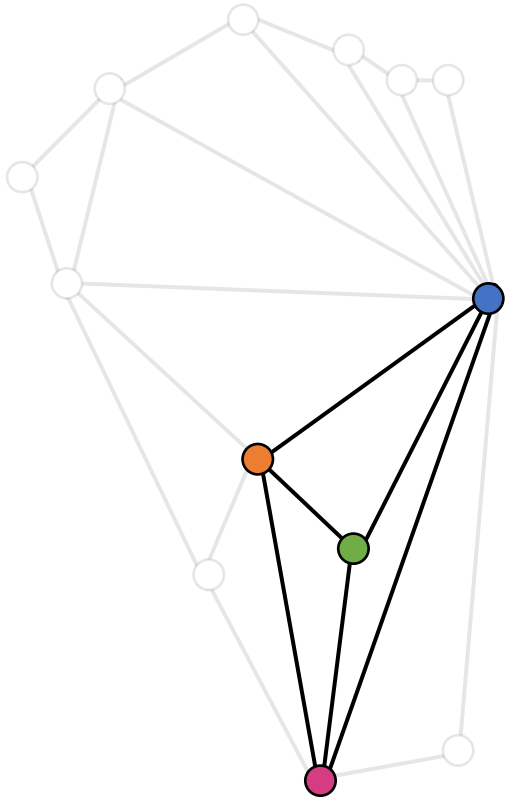


Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o GRAFO ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: Não! Essa parte do grafo requer no mínimo 4 cores, pois cada vértice é vizinho dos outros três.

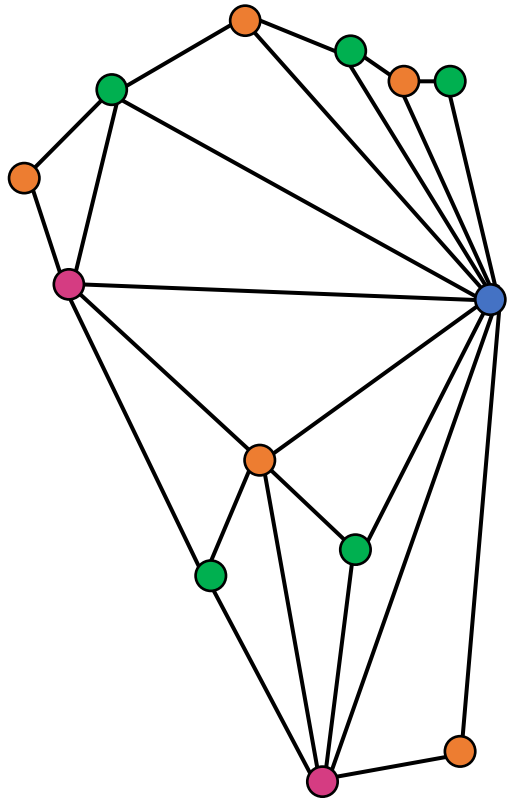


Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o GRAFO ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: Não!

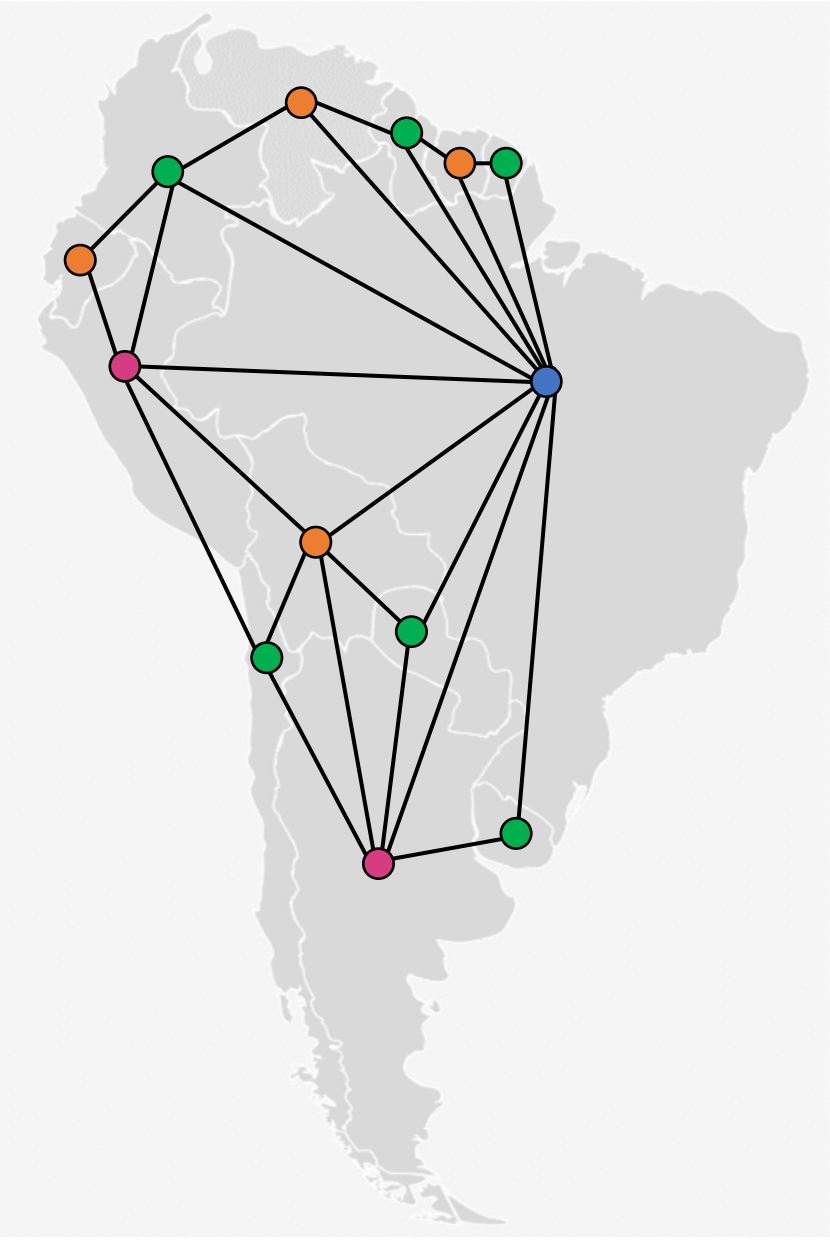


Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o GRAFO ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

  - Resposta: Não!

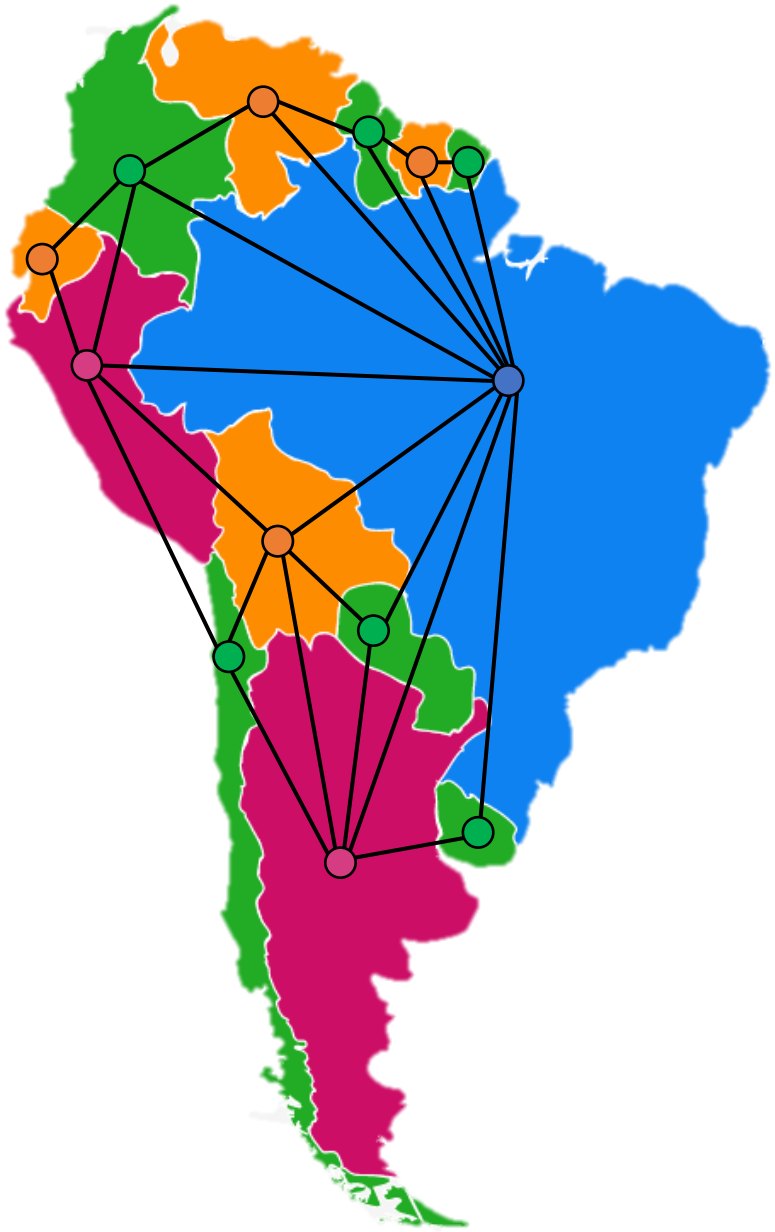


Problemas de Decisão

- São problemas cuja resposta é SIM ou NÃO.

- Por exemplo:
  - É possível colorir o mapa ao lado usando apenas $k = 3$ cores de forma que dois países vizinhos não tenham cores iguais?

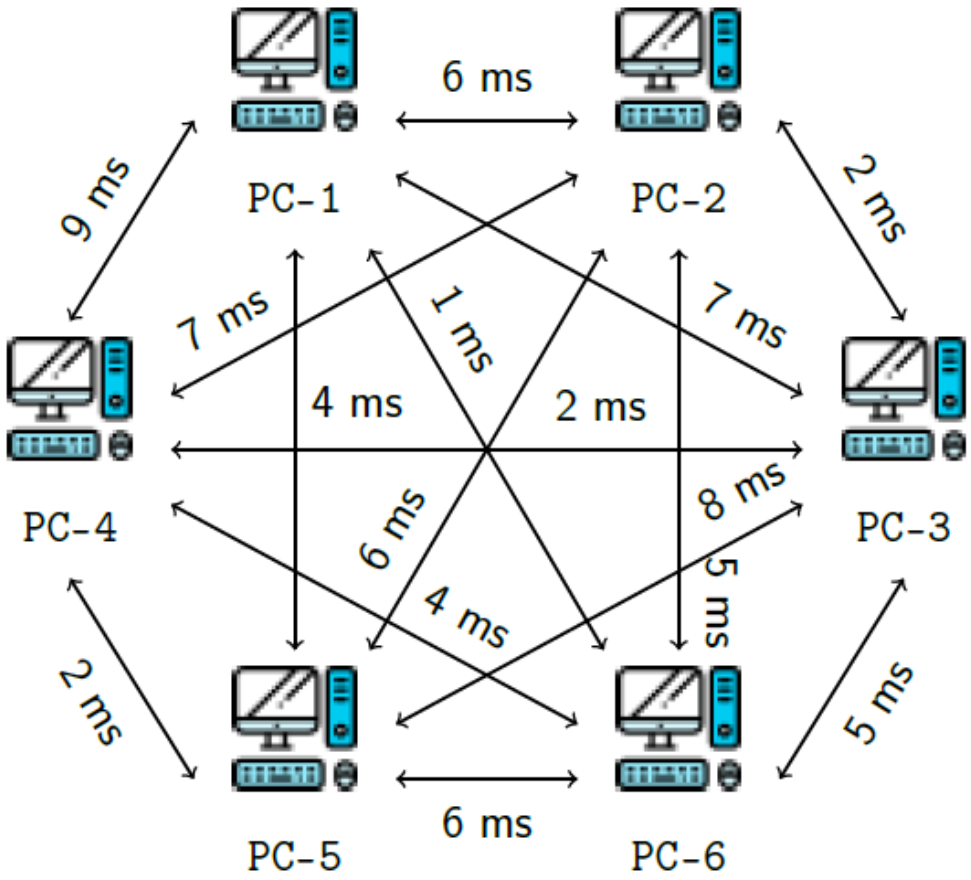
  - Resposta: Não!



Problemas de Decisão

- Existe alguma forma de enviar uma informação na rede abaixo, começando no computador $ini = 1$, passando por todos os demais computadores e retornando ao ponto de partida (ini) em menos de $t = 20ms$?

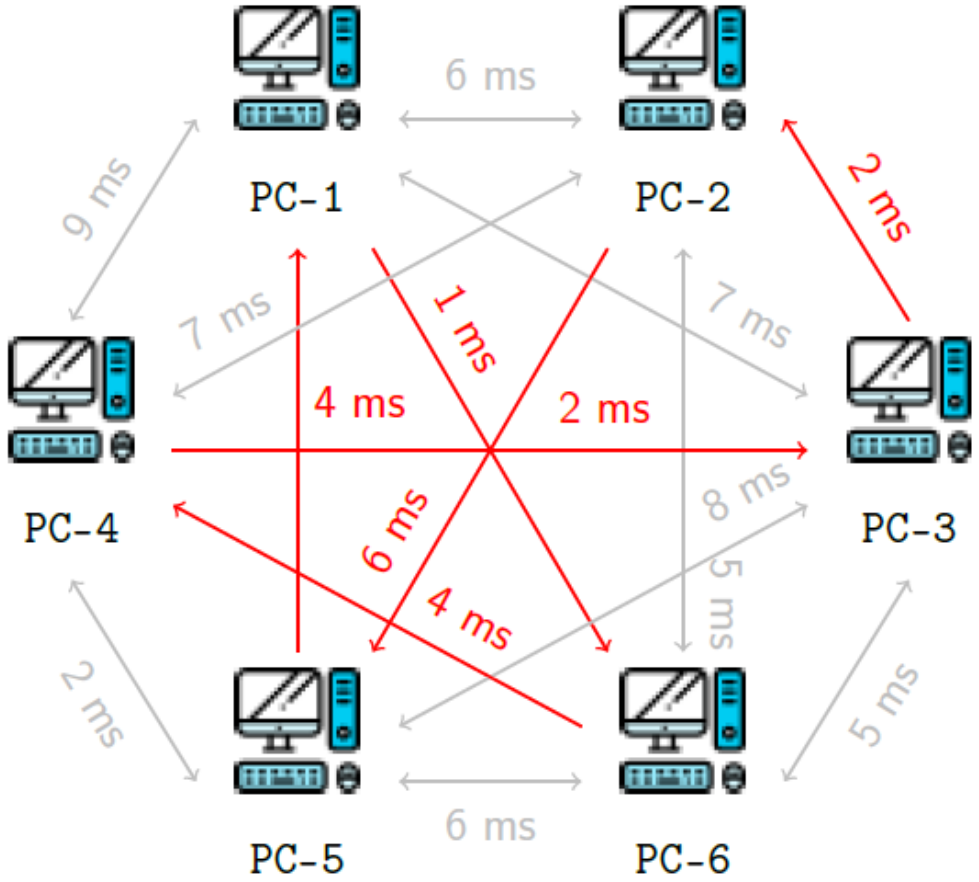
- Resposta:
  - Sim?
  - Não?



Problemas de Decisão

- Existe alguma forma de enviar uma informação na rede abaixo, começando no computador $ini = 1$, passando por todos os demais computadores e retornando ao ponto de partida (ini) em menos de t = $20ms$?

- Resposta:
  - **Sim**



Problemas de Decisão

- Problema da Mochila 0-1:
  - O Problema da Mochila 0-1 considera que existem n itens, com pesos $p_1, p_2, \cdots, p_n$ e valores associados $v_1, v_2, \cdots, v_n$, e uma mochila de capacidade P
  - Pergunta: Existe alguma combinação dos itens $x_1, x_2, \cdots, x_n$, onde para cada item i: $x_i = \begin{cases} 1, & \text{se o item i for escolhido} \\ 0, & \text{em caso contrário} \end{cases}$ que satisfaça:
  - $\sum_{i=1}^{n} x_i * p_i \leq P$ (não exceder o peso máximo P
  - $\sum_{i=1}^{n} x_i * v_i \geq V$ (a mochila deve contar no mínimo o valor V)

Problemas de Decisão

- Objetivo: Verificar se existe uma solução que atenda as condições previamente determinadas.

- Problema Coloração: No máximo k cores
- Problema da Comunicação: No máximo t milissegundos
- Problema da Mochila 0-1: Levar no mínimo o valor V

- A resposta é SIM (existe essa solução) ou NÃO (não existe uma solução que atenda a essas exigências).

Problemas de Otimização

- Diferente dos problemas de decisão, os problemas de otimização busca a melhor solução possível.

- Problema da Coloração: Qual é o menor número possível de cores que permite pintar o mapa sem que dois países vizinhos tenham cores iguais?

- Problema da Comunicação: Qual é o menor tempo possível para que uma mensagem partindo do computador ini passe por todos os computadores uma única vez e retorne ao ponto de partida?

- Problema da Mochila: Qual é o maior valor que podemos carregar na mochila, considerando o limite de peso e os itens (valores e pesos unitários)?

Problemas de Decisão vs Otimização

- Vale observar que todo problema de otimização possui uma versão de decisão equivalente:
- Por exemplo: Problema da Coloração - Otimização: Qual é o menor número de cores necessárias para pintar o mapa:

- Problema da Coloração - Otimização:
  1. É possível pintar o mapa usando $k = 1$ cores?
  2. É possível pintar o mapa usando $k = 2$ cores?
  3. É possível pintar o mapa usando $k = 3$ cores?

     ⋮
  n. É possível pintar o mapa usando $k = n$ cores, onde n equivale ao número de países/regiões no mapa?

  O valor k do primeiro caso que retornar SIM é a resposta da versão de otimização.

Problemas de Decisão vs Otimização

- Vale observar que todo problema de decisão possui uma versão de otimização equivalente:
- Por exemplo: Problema da Mochila 0-1 - Decisão: Existe alguma combinação de itens que possibilite valor V?

- Problema da Mochila 0-1 - Otimização:
  - Qual é o maior valor X que podemos carregar na mochila, considerando o limite de peso e os itens (valores e pesos unitários)?

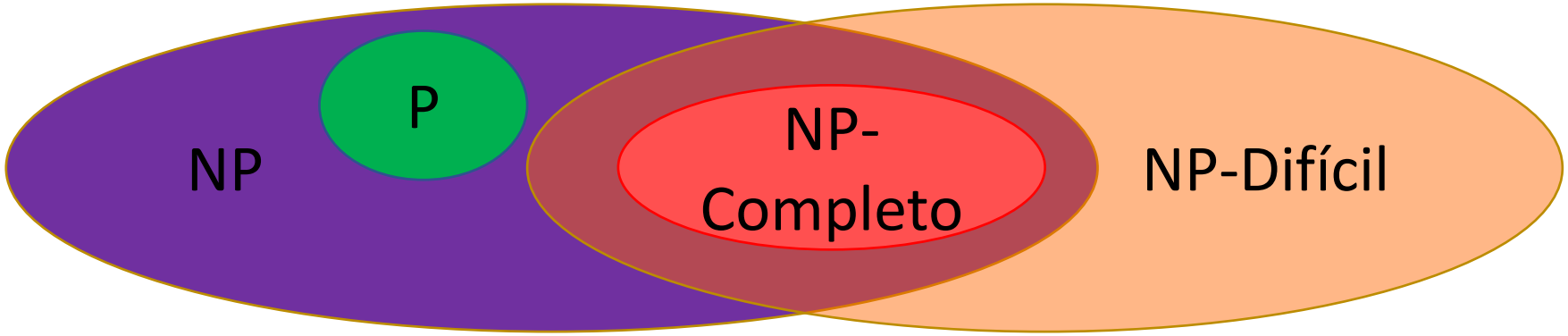
  - Se $X \geq V$, então a resposta da versão de decisão é SIM
  - Se $X < V$, então a resposta da versão de decisão é NÃO

Problemas de Decisão vs Otimização

- Essa relação entre Problemas de Decisão e Otimização implica que saber resolver um implica que sabemos resolver o outro (com algum trabalho extra).
- Nosso estudo das Classes de Problema vai focar nos Problemas de Decisão.
- Entretanto é importante lembrar que o nível de dificuldade de um é equivale ao outro.

Classes de Problemas

- Os Problemas de Decisão podem ser classificados em algumas categorias.
- Dentre outras, as principais são:
  - P
  - NP
  - NP-Difícil
  - NP-Completo



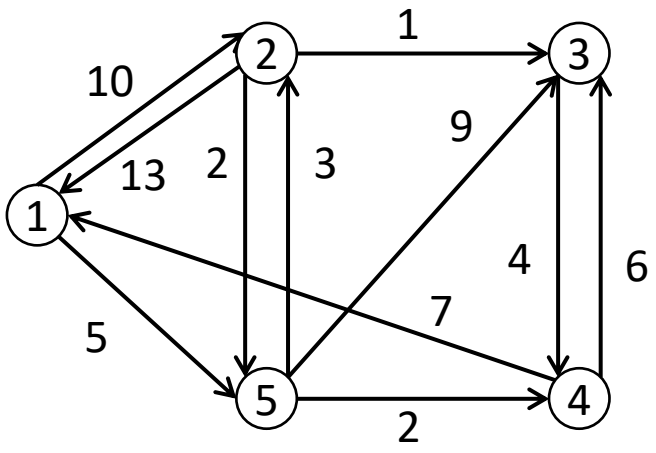
A Classe P

- A Classe P contém os problemas de decisão para os quais se conhece um algoritmo de tempo $O(n^k)$, onde k é uma constante (não cresce junto do n), capaz de resolver o problema.

- Entretanto, como vimos nos slides anteriores:
  - Se sabemos resolver a versão de decisão, sabemos resolver a versão de otimização.
  - Logo: A Classe P contém todos os problemas que podemos resolver computacionalmente em tempo $O(n^k)$

Exemplo – Classe P

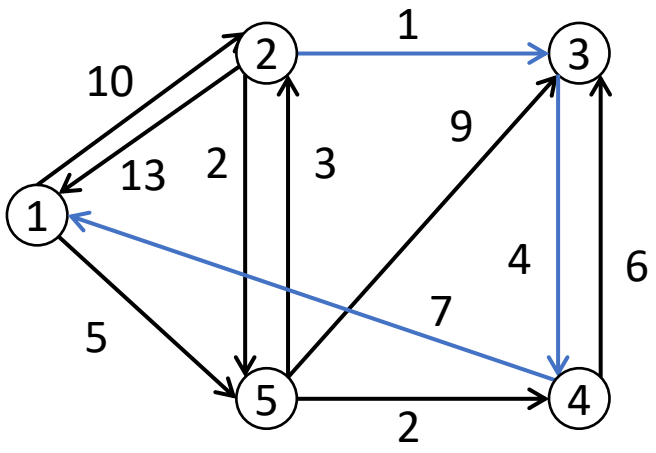
- Entradas: Um grafo com custos associados as arestas e dois vértices v e u
- Questão: Existe um caminho que parte de v e chega até u com custo menor ou igual à k?
- Resposta: Sim ou Não



Ponto de Partida: 2
Ponto de Chegada: 1
Custo Máximo: $k \leq 12$
Resposta: ???

Exemplo – Classe P

- Entradas: Um grafo com custos associados as arestas e dois vértices v e u
- Questão: Existe um caminho que parte de v e chega até u com custo menor ou igual à k?
- Resposta: Sim ou Não



Ponto de Partida: 2
Ponto de Chegada: 1
Custo Máximo: $k \leq 12$
Resposta: SIM

Existe um algoritmo que identifica o menor caminho e seu custo?

Sim. Algoritmo de Dijkstra.
Esse algoritmo é capaz de encontrar o menor caminho entre o vértice inicial e todos os demais vértices do grafo.

Complexidade do Algoritmo: $O(n^3)$

A Classe P

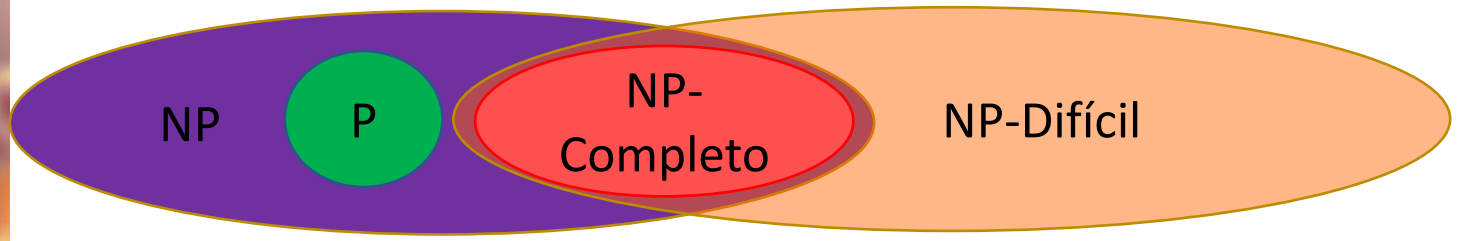
- Dado um problema qualquer.
- Existe um algoritmo de tempo polinomial que resolve esse problema?
  - Sim: O problema está na Classe P
  - Não: Precisamos avaliar mais para classificar esse problema

- Importante: Resolver um problema implica em ser capaz de encontrar a solução ótima para todas as entradas possíveis desse problema.
- Algoritmos aproximativos tem tempo polinomial, mas não encontram a solução ótima. As soluções nesses casos são aproximadas.

A Classe NP

- Se P contém todos os problemas de decisão para os quais se conhece um algoritmo de tempo polinomial.

- Então NP deve conter os problemas de decisão para os quais NÃO se conhece um algoritmo de tempo polinomial.
  - Não!
  - Se fosse assim $P \neq NP$.
  - Mas esse é um problema em aberto!
  - Hoje sabemos apenas que $P \subseteq NP$.



A Classe NP

- O nome NP vem de tempo polinomial não determinístico (*nondeterministic polynomial time*).

- Vale relembrar o conceito de não determinístico:
  - Máquina de Turing Determinística: Todas as possíveis transições devem ser apresentadas (equivale ao conceito de um algoritmo).
  - Máquina de Turing Não Determinística: Explora todos as possíveis transições. A máquina entra em estado de aceitação se alguma combinação de transições levar ao estado de aceitação.

- Como existe uma associação entre Máquinas de Turing e algoritmos: Existe um algoritmo hipotético que é capaz de resolver o problema em tempo polinomial.

A Classe NP

- Como os computadores atuais são determinísticos, eles (ainda?) não conseguem resolver esses problemas em tempo polinomial.

- Mas como verificamos se um problema está em NP se não temos como simular máquinas/algoritmos não determinísticos?

- Vamos considerar que existe uma máquina mágica e hipotética: um oráculo. Esse oráculo é supostamente capaz de resolver o problema e entregar uma resposta.

- Observe que esse oráculo pode ser uma máquina/algoritmo não determinístico. Basta que ele entregue uma resposta para o problema.

A Classe NP

- Sabendo a resposta do oráculo, podemos verificar se ela é uma resposta válida?

- Se for possível CERTIFICAR a resposta SIM em tempo polinomial, o problema está em NP.

- Certificar: Verificar se a resposta é válida para as entradas ou não.

A Classe NP

- Exemplo: Problema do Caixeiro Viajante

- Entradas: Um grafo completo (existe um caminho entre cada par de vértices) com pesos associados as arestas, vértice inicial i, parâmetro k.
- Pergunta: Existe um caminho que comece no vértice i, passe por todos os vértices do grafo e retorne ao ponto de partida com custo total menor ou igual à k?
- Resposta: SIM ou NÃO

A Classe NP

- Suponha que o oráculo respondeu que existe um caminho que atenda essas condições.

- Pergunta: O que você precisa para CERTIFICAR a resposta SIM do oráculo (verificar se ela é válida ou não)?
  - O caminho.

- Como você poderia CERTIFICAR esse caminho?
  - Percorrer o caminho e somar o custo de cada aresta percorrida.

- Qual é a complexidade desse algoritmo certificador?
  - $O(n)$ – o tempo necessário para percorrer o caminho

A Classe NP

- Observe que:
  - Existe um oráculo (máquina de Turing Não Determinística) que pode responder qualquer entrada do problema em tempo polinomial.
  - Podemos CERTIFICAR a resposta SIM em tempo polinomial.

- Logo:
  - Problema do Caixeiro Viajante está em NP.

- Por que esse problema não está em P?
  - Simples: Não sabemos até hoje um algoritmo (determinístico) de tempo polinomial que possa resolver esse problema
  - As soluções conhecidas até hoje são de tempo $O(n!)$ ou aproximativas.

A Classe NP

- Por que CERTIFICAR a resposta SIM e não TODAS as respostas?

- Simples:
  - Resposta SIM: Basta verificar se a resposta satisfaz as condições do problema e da entrada.
  - Resposta NÃO: Como verificar se a resposta NÃO é válida?
    - Seria necessário avaliar todas as possibilidades de respostas para verificar se alguma atende aos critérios (do problema e da entrada).
    - E isso NÃO pode ser feito em tempo polinomial.

A Classe NP

- Por que CERTIFICAR a resposta SIM e não TODAS as respostas?

- SIM: Basta uma resposta para demonstrar que existe uma resposta.
  - Basta um exemplo para provar que é possível

- NÃO: É necessário exaurir todas as respostas possíveis para provar que ela não existe. E isso é muito mais difícil...

A Classe NP

- Exemplo: Problema da Cobertura de Conjuntos

- Considere um conjunto $U = \{1, 2, \cdots, n\}$ e um conjunto S, formados por subconjuntos de U.
- É possível cobrir todos os elementos de U usando no máximo k elementos do conjunto S?

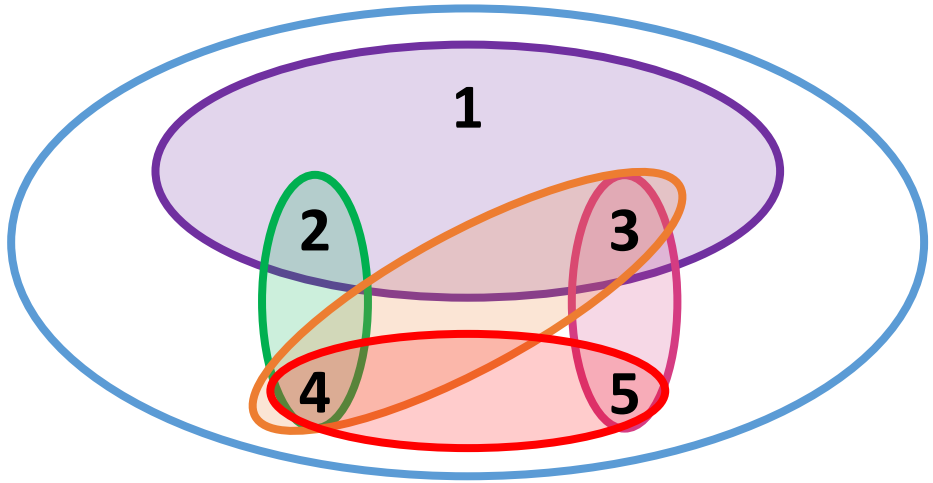
- Resposta: SIM ou NÃO

A Classe NP

- Exemplo: Problema da Cobertura de Conjuntos

- Considere um conjunto $U = \{1, 2, 3, 4, 5\}$ e um conjunto $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4,5\}, \{3, 5\}\}$.
- É possível cobrir todos os elementos de U usando no máximo 3 elementos do conjunto S?

- Resposta: SIM ou NÃO

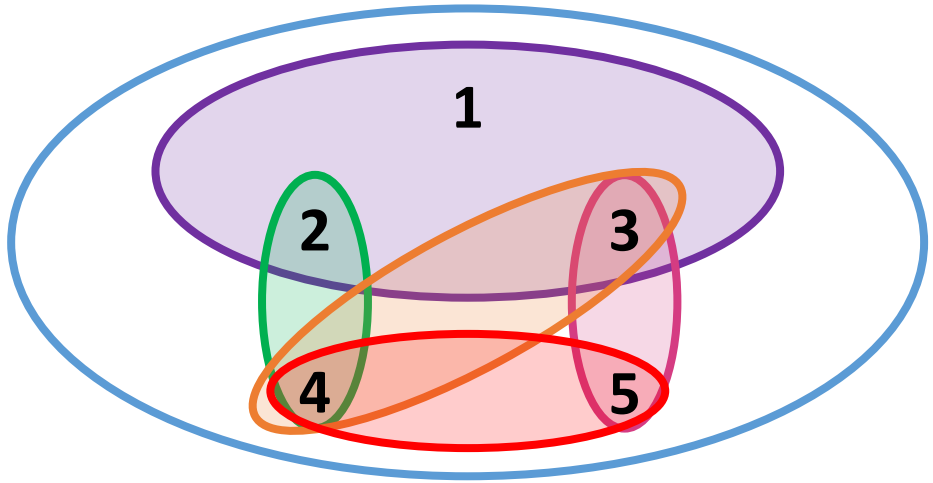


A Classe NP

- Exemplo: Problema da Cobertura de Conjuntos

- Considere um conjunto $U = \{1, 2, 3, 4, 5\}$ e um conjunto $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 5\}\}$.
- É possível cobrir todos os elementos de U usando no máximo 3 elementos do conjunto S?

- Resposta: **SIM** ou NÃO
- Certificado: $\{1, 2, 3\}, \{2, 4\}, \{3, 5\}$

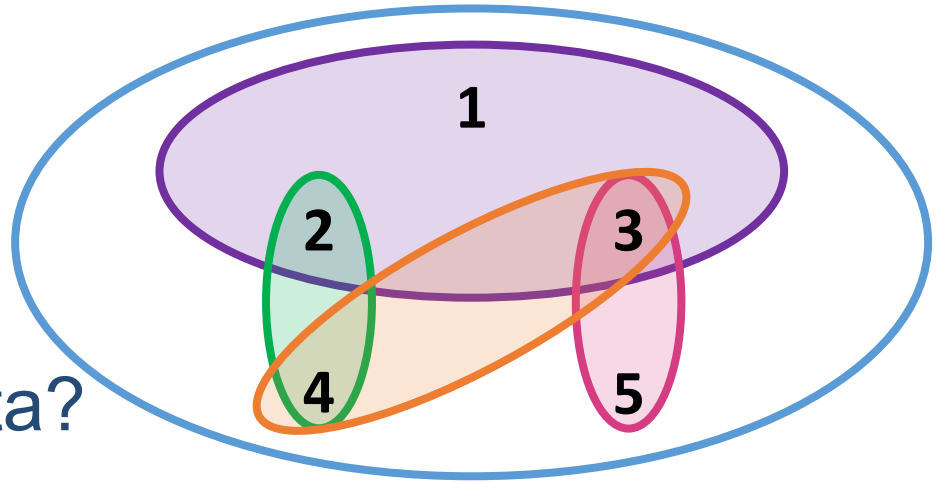


A Classe NP

- Exemplo: Problema da Cobertura de Conjuntos

- Considere um conjunto $U = \{1, 2, 3, 4, 5\}$ e um conjunto $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 5\}\}$.
- É possível cobrir todos os elementos de U usando no máximo **2** elementos do conjunto S?

- Resposta: SIM ou NÃO
- NÃO (aparentemente)
- Mas como certificar essa resposta?



A Classe NP

- Como certificar a resposta SIM:
  - Basta percorrer cada conjunto da resposta, marcando os vértices.
  - Se, ao final, todos os vértices foram marcados e a resposta contém no máximo k conjuntos então a solução é válida.
  - Complexidade: $O(n)$.

A Classe NP

- Como certificar a reposta NÃO:
  - Necessário testar todas as combinações possíveis de conjuntos com no máximo k elementos.
  - Para cada combinação, marcar todos os vértices. Se todos forem marcados, a resposta NÃO está incorreta.
  - Se nenhuma combinação marcar todos os vértices, o NÃO foi certificado.
  - Complexidade:
    - Considerando que existem no máximo $\sum_{x=1}^{n} \frac{n!}{x!(n-x)!} = 2^n - 1$ subconjuntos.
    - Como no pior caso precisamos testar cada um deles
    - A complexidade dessa certificação é $O(2^n)$

Relação entre P e NP

- P: Sabemos resolver em tempo polinomial.
- NP: Dada uma solução SIM, podemos certificá-la em tempo polinomial.

- Sabendo disso: Qual é a relação entre P e NP?
  - $P \subseteq NP$ ($P = NP$ ou $P \subset NP$) . Por que?

- Se sabemos resolver, sabemos certificar (o SIM e o NÃO).
- Basta resolver e verificar se as respostas são equivalentes.

- Logo: Todo problema em P está também em NP.

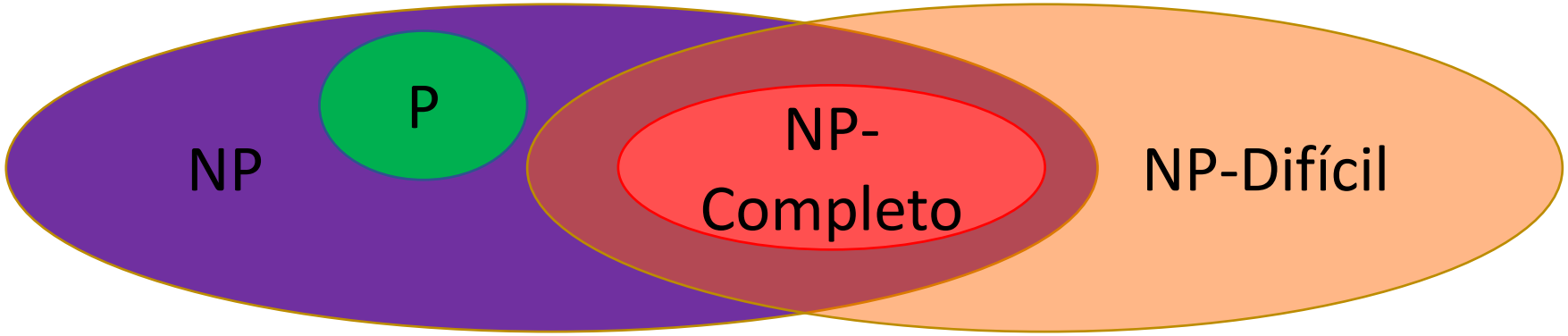
A Classe NP

- A Classe NP é restrita aos problemas de decisão.

- Por que?
  - Como certificar uma resposta para um problema de otimização?
  - Problema de Otimização: Melhor solução possível
  - Dada a solução, como podemos CERTIFICAR que ela é a melhor possível?
  - Seria necessário testar todas as outras soluções em busca de uma ainda melhor.
  - Como vimos, isso é muito custoso computacionalmente

Classes de Problemas

- Os Problemas de Decisão podem ser classificados em algumas categorias.
- Dentre outras, as principais são:
  - **P**
  - **NP**
  - NP-Difícil
  - NP-Completo



NP-Difícil

- A classe NP-Difícil contém os problemas mais difíceis de se resolver
- São problemas tão difíceis que TODO problema em NP pode ser transformado em tempo polinomial em um problema NP-Difícil.
- Isso implica que:
  - Se **UM** problema NP-Difícil for resolvido em tempo polinomial então **TODOS** os problemas em NP também podem ser resolvidos em tempo polinomial.
- Vale observar que os problemas de otimização podem estar contidos na classe NP-Difícil.

NP-Difícil

- Suponha um problema da classe NP-Difícil
- Suponha um outro problema qualquer
- Se:
  - For possível transformar a entrada do problema NP-Difícil em uma entrada do segundo problema **E**
  - For possível usar o segundo problema para resolver o primeiro **E**
  - Uma resposta SIM para o segundo problema significar uma resposta SIM para o segundo problema
- Então:
  - Podemos resolver o problema difícil usando o segundo problema
  - Isso prova que o segundo problema é pelo menos tão difícil quanto o primeiro.
  - Além disso, se eu conseguir resolver o segundo problema em tempo polinomial, eu posso resolver TODOS os problemas de NP em tempo polinomial.

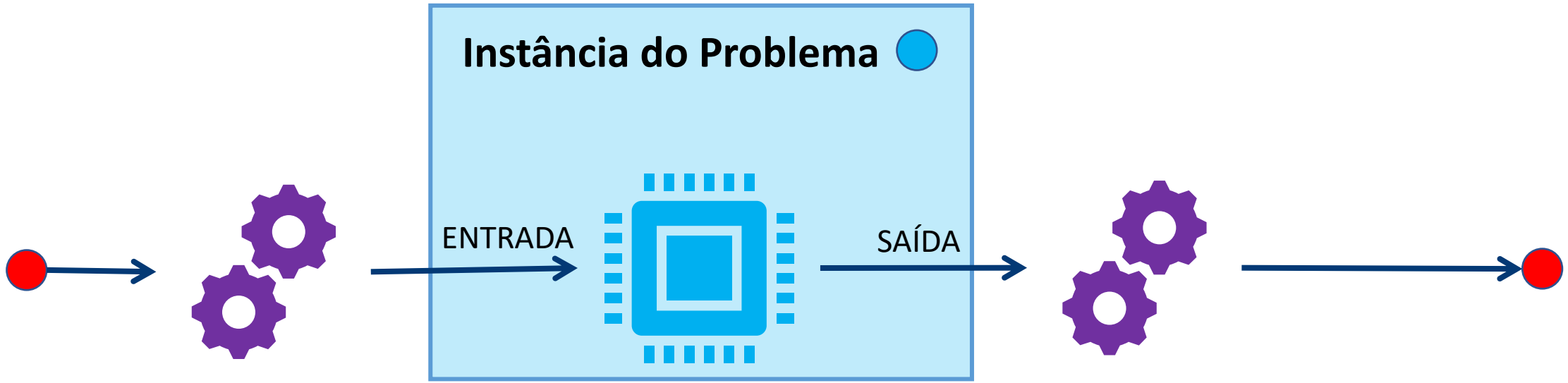
Transformação Polinomial

- Uma transformação polinomial de um problema Π_1 em um problema Π_2 é uma função f, computável em tempo polinomial, que, para qualquer instância I_1 de Π_1, constrói uma instância I_2 de Π_2, tal que I_1 tem resposta SIM para Π_1 se e somente se I_2 tem resposta SIM para Π_2. Indicamos por $\Pi_1 \propto \Pi_2$.

- Em resumo: Podemos resolver Π_1 através de Π_2

Transformação Polinomial: $\bullet \propto \bullet$

- Suponha um problema $\bullet$
- Suponha um outro problema qualquer $\bullet$



Transformar a entrada do problema $\bullet$ em uma entrada para o problema $\bullet$ em tempo polinomial

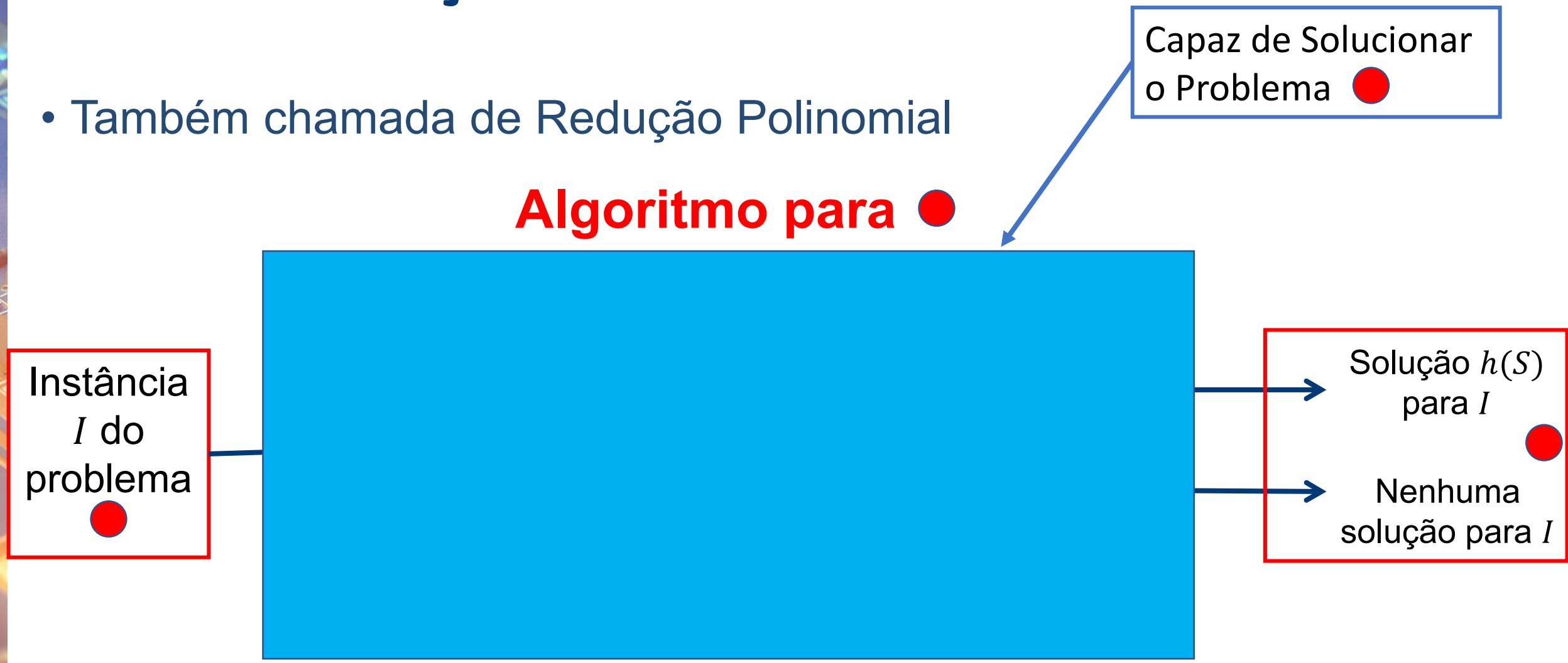
Resolver o problema $\bullet$

Transformar a saída do problema $\bullet$ em uma saída para o problema $\bullet$ em tempo polinomial

Saída para o problema $\bullet$

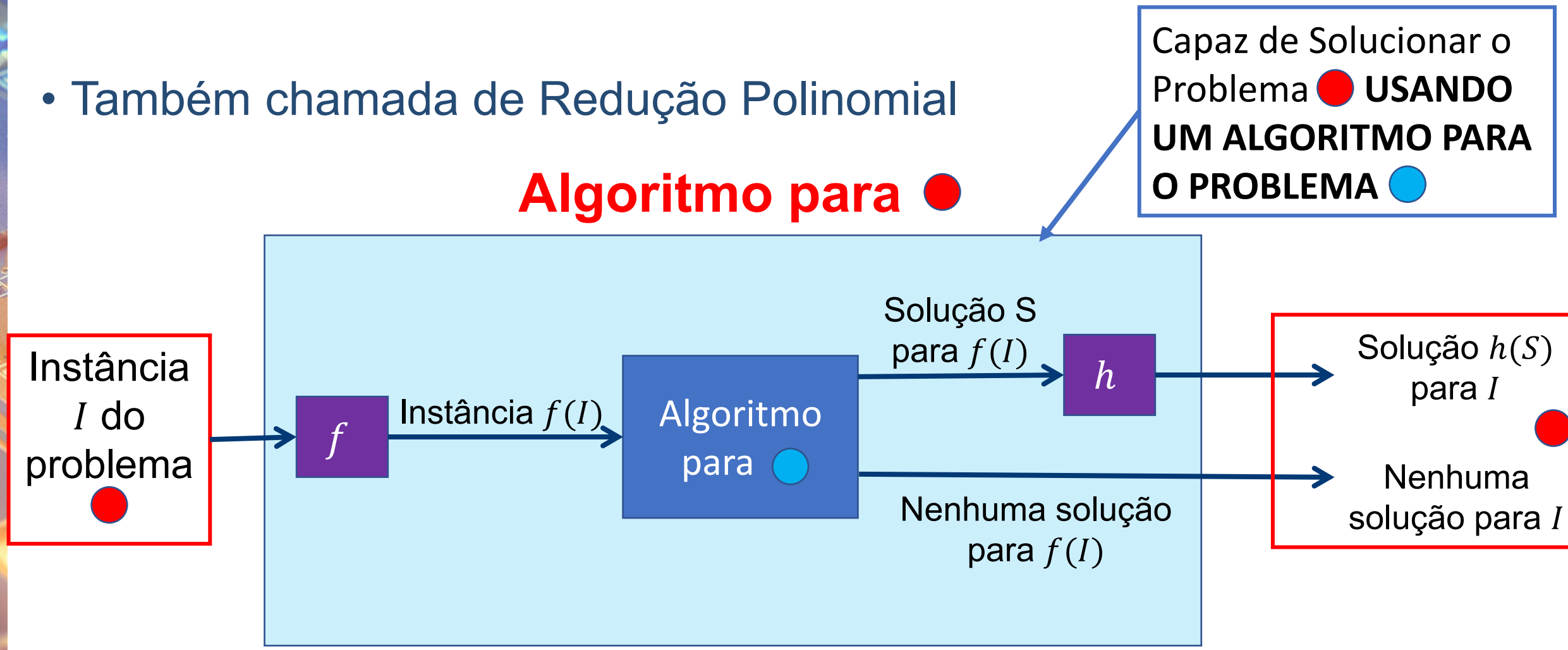
Transformação Polinomial: ● ∝ ●

- Também chamada de Redução Polinomial



Transformação Polinomial: ● ∝ ●

- Também chamada de Redução Polinomial



Onde f e h são transformações que requerem tempo polinomial

Exemplo de Transformação Polinomial

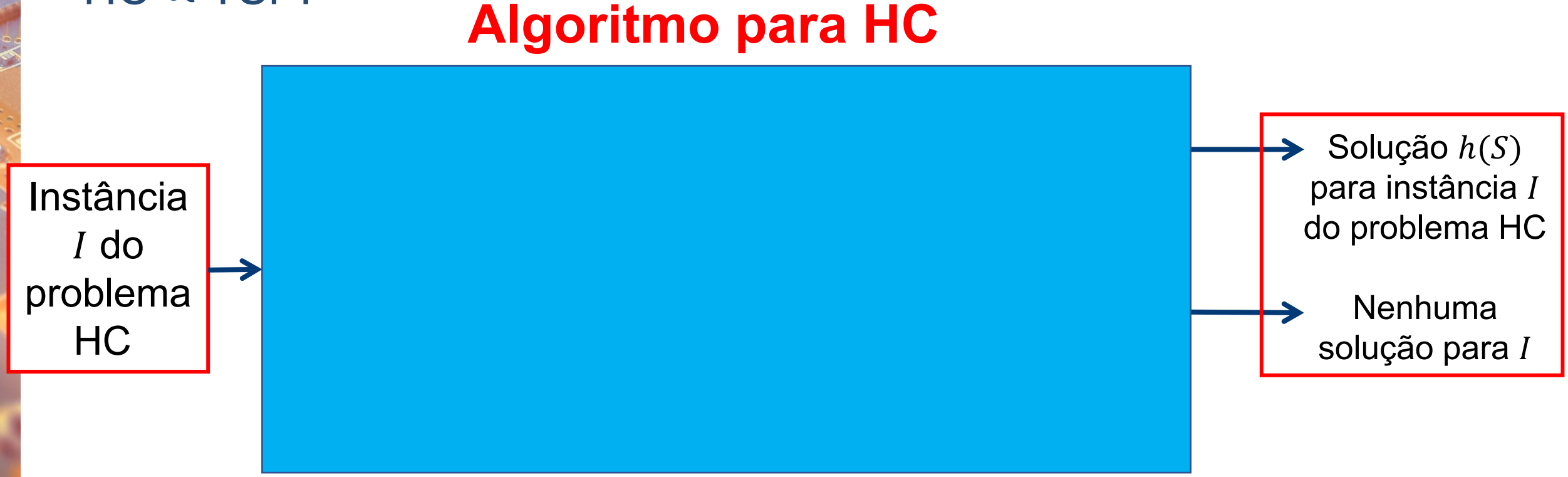
- Problema do Ciclo Hamiltoniano (HC):
  - Entradas: Grafo $G = (V, E)$
  - Questão: G apresenta um caminho que começa em um vértice qualquer, passa por todos os vértices do grafo uma única vez e retorna ao ponto de partida?
  - Resposta: SIM ou NÃO

Exemplo de Transformação Polinomial

- Problema do Caixeiro Viajante (TSP):
  - Entradas: Grafo Completo $G = (V, E)$ com custos associados as arestas e um valor k
  - Questão: G apresenta um caminho que começa em um vértice do grafo, passa por todos os vértices uma única vez e retorna ao ponto de partida com custo (soma dos custos das arestas percorridas) menor ou igual à k?
  - Resposta: SIM ou NÃO

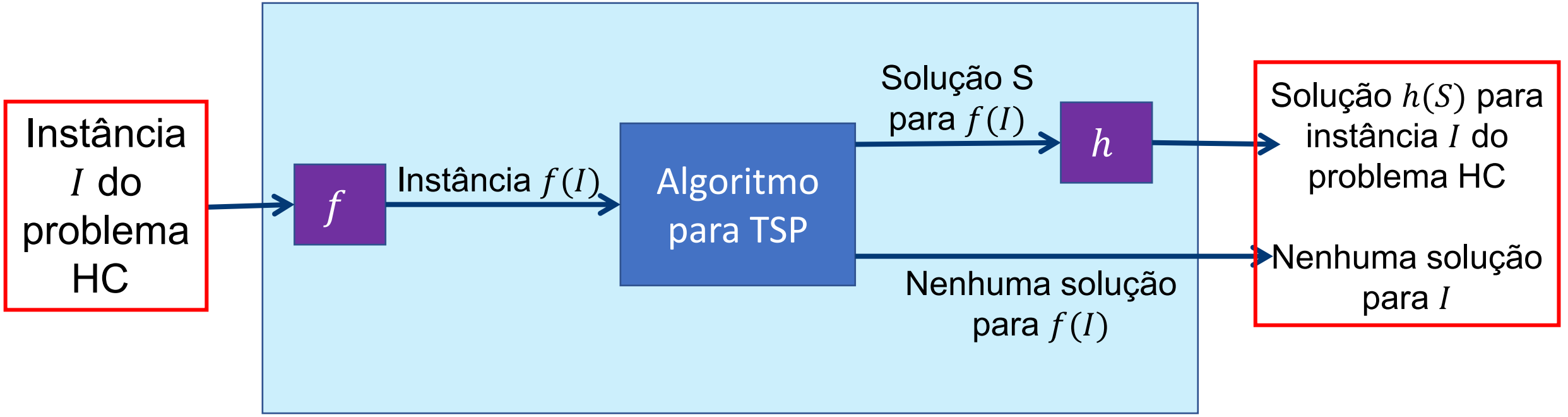
Exemplo de Transformação Polinomial

- É possível estabelecer uma transformação polinomial entre Ciclo Hamiltoniano e Caixeiro Viajante?
- HC $\propto$ TSP:



HC $\propto$ TSP

Algoritmo para HC



Onde f e h são transformações que requerem tempo polinomial

- Como definir f e h?

HC $\propto$ TSP

- Função f:
  - Dada uma instância $G = (V, E)$ de Ciclo Hamiltoniano (HC), construímos uma instância do Caixeiro Viajante: um grafo $G' = K_n = (V, E')$, com peso $w(e)$ associado a cada aresta $e \in E'$ tal que:
  - $w(e) = \begin{cases} 1, & se\ e \in E \\ n, & se\ e \notin E \end{cases}$
  - Onde K_n equivale a um grafo completo (com todas as arestas possíveis) com n vértices e $n = |V|$.
- Vale observar que o grafo G' pode ser construído em tempo polinomial.

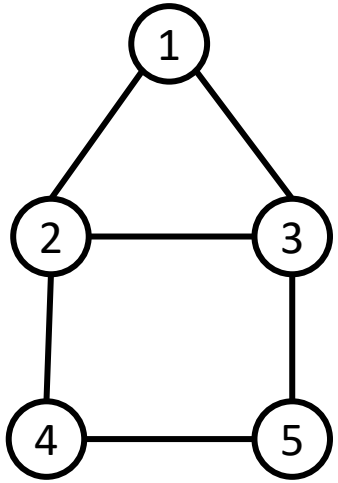
HC $\propto$ TSP

- Função h:
  - G tem resposta SIM para HC se, e somente se, G' tem resposta SIM para TSP, considerando $k = n$.

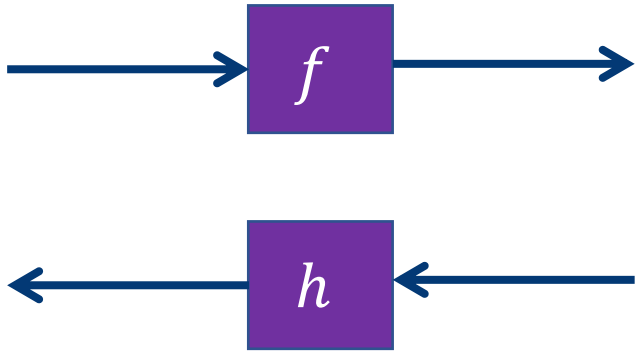
  - Isso quer dizer:
    - ($\Rightarrow$) Se G tem um ciclo hamiltoniano então G' tem um ciclo de custo n que passa por todos os vértices.
    - ($\Leftarrow$) Se G' tem um ciclo de custo n que passa por todos os vértices então G tem um ciclo hamiltoniano

HC ∝ TSP

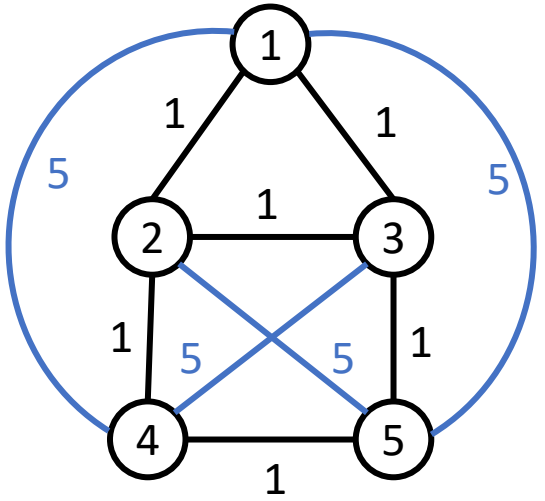
- Exemplo:



Instância HC
Grafo $G = (V, E)$



Transformação
Polinomial



Instância TSP
Grafo $G' = K_n = (V, E')$
$k = n$

Possui ciclo hamiltoniano sss possui ciclo de custo n

HC $\propto$ TSP

- ($\Rightarrow$) Se existe um ciclo hamiltoniano em G, então o mesmo ciclo em G' tem peso n.
- Cada aresta desse ciclo em G tem peso 1 em G', logo a soma dos custos desse ciclo é de exatamente n
- Logo G' tem resposta SIM.

HC ∝ TSP

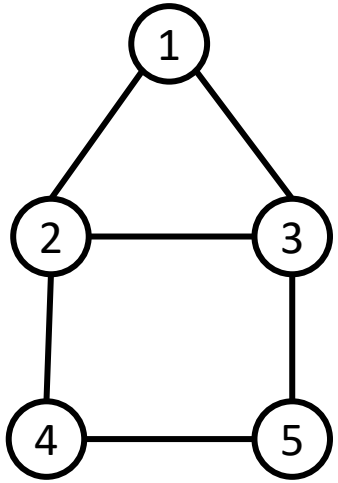
- ($\Leftarrow$) Se G' tem resposta SIM para TSP, então existe um ciclo C que passa por todos os vértices com custo $\mathrm{k} = n$ em G'

- Se o ciclo C passa por todos os vértices e tem custo n então ele inclui apenas arestas de custo 1.
- Se ele inclui apenas aresta de custo 1, essas mesmas arestas existem em G.
- Se todas as arestas de C existem em G, G possui um ciclo hamiltoniano.

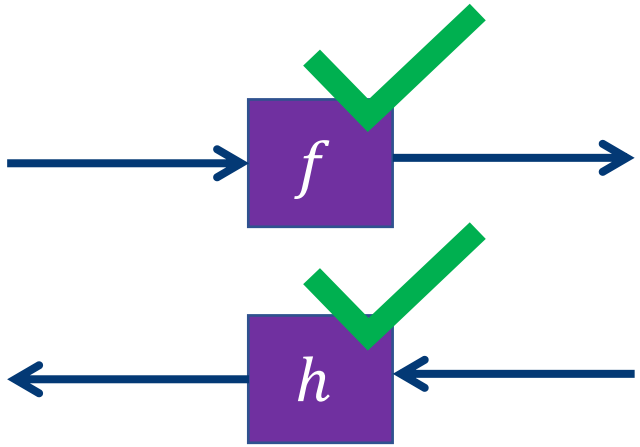
- Logo: Se G' tem resposta SIM para TSP então G tem resposta SIM para HC

HC ∝ TSP

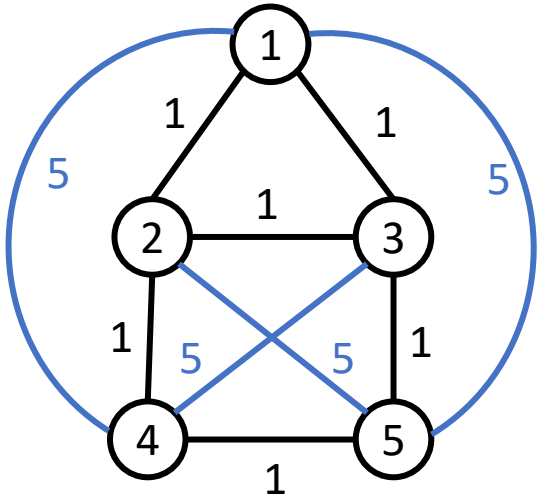
- Exemplo:



Instância HC
Grafo $G = (V, E)$



Transformação
Polinomial



Instância TSP
Grafo $G' = K_n = (V, E')$
$k = n$

Possui ciclo hamiltoniano sss possui ciclo de custo n

HC ∝ TSP

- Considerando que:
  - É possível transformar uma instância de HC em uma instância do TSP em tempo polinomial.

  - Uma resposta SIM para TSP implica uma resposta sim para HC
  - Uma resposta SIM para HC implica uma resposta sim para TSP

- Então:
  - HC pode ser reduzido em tempo polinomial à TSP.
  - Podemos resolver HC através de um algoritmo para o TSP.

Transformação Polinomial

- A transformação polinomial tem duas aplicações nos estudos dos algoritmos:

1. Eu não sei resolver o problema A, mas sei transformar em tempo polinomial o problema A em B e resolver o problema B.
   - O algoritmo para B se torna um algoritmo para A.

2. Eu sei que um problema A é muito difícil. Se eu posso transformar em tempo polinomial todas as instâncias do problema A em instâncias do problema B ($A \propto B$), estou demonstrando que B é pelo menos tão difícil quanto A.
   - Caso contrário, B seria uma solução para A e não seria tão difícil assim...

Transformação Polinomial

- Se $\Pi_1 \propto \Pi_2$ e $\Pi_2 \in P$, então $\Pi_1 \in P$.
  - Se toda instância de Π_1 pode ser transformada em tempo polinomial em uma instância de Π_2
  - E podemos resolver instâncias de Π_2 em tempo polinomial
  - Então: podemos resolver todas as instâncias de Π_1 em tempo polinomial (via Π_2)

- Se $\Pi_1 \propto \Pi_2$ e $\Pi_2 \propto \Pi_3$ então $\Pi_1 \propto \Pi_3$
  - Se podemos transformar em tempo polinomial Π_1 em Π_2 e Π_2 em Π_3
  - Então também podemos transformar Π_1 em Π_3 em tempo polinomial.

Transformação Polinomial

- Um problema Π é *NP-Difícil* se:
  - Todo problema $\Pi_0 \in NP$, $\Pi_0 \propto \Pi$

- Dessa forma, para caracterizar um problema como NP-Difícil seria necessário demonstrar que todo problema em NP pode ser transformado em tempo polinomial em Π

- Isso é praticamente impossível, uma vez que são muitos problemas em NP.

Transformação Polinomial

- Um problema Π é *NP-completo* se:
  - $\Pi \in NP$
  - Todo problema $\Pi_0 \in NP$, $\Pi_0 \propto \Pi$

- Se o problema Π está em NP e vale a regra anterior, então Π está em NP-Completo.

Transformação Polinomial

- Se:
  - Π_1 e $\Pi_2 \in NP$
  - Π_1 é NP-completo
  - $\Pi_1 \propto \Pi_2$
- Então:
  - Π_2 é NP-completo.

- Com base nessas afirmativas, podemos concluir que basta demonstrar uma transformação polinomial de um problema NP-Completo em um outro problema para demonstrar que esse segundo problema também é NP-Completo.

Transformação Polinomial

- Com base nessas afirmativas, podemos concluir que basta demonstrar uma transformação polinomial de um problema NP-Completo em um outro problema para demonstrar que esse segundo problema também é NP-Completo.

- Qual é o problema?
  - Como provar que um problema é NP-Completo sem que já exista um problema NP-Completo?

  - Seria necessário provar que todo problema em NP pode ser transformado nesse problema?

Teorema de Cook-Levin

- Cook e Levin demonstraram o primeiro problema NP-Completo.
- Teorema de Cook-Levin: Satisfatibilidade Booleana é NP-Completo

- Problema da Satisfatibilidade Booleana (SAT):
  - Entradas: Variáveis $x_1, x_2, \ldots, x_n$ e Cláusulas $C_1, C_2, \ldots, C_n$ formadas por disjunções de variáveis.
  - Questão: Existe uma atribuição de valores booleanos (verdadeiros e falsos) às variáveis que implique que a conjunção de todas as cláusulas seja verdadeira.

SAT – Satisfatibilidade Booleana

- Dadas as variáveis: $x_1, x_2, \ldots, x_n$
- Podemos definir uma fórmula como uma conjunção de disjunções das variáveis: $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3} \vee x_4)$
- Onde: $\vee$ e $\wedge$ equivalem aos operadores lógico OU e E respectivamente

- Questão: Existe uma atribuição de valores às variáveis que torne a fórmula verdadeira (satisfaça a fórmula)?
- Respostas: SIM ou NÃO

SAT – Satisfatibilidade Booleana

- $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4)$

- Supondo $x_1 = Verdadeiro$ e $x_3 = Falso$
- $(\overline{x_1} \vee \overline{x_3})$: Falso OU Verdadeiro = Verdadeiro
- $(x_1 \vee x_3)$: Verdadeiro OU Falso = Verdadeiro
- Isso vale para todas as variáveis e cláusulas

Teorema de Cook-Levin

- Teorema de Cook-Levin: Satisfatibilidade Booleana é NP-Completo

- Para demonstrar que SAT é NP-Completo precisamos:
  1. Demonstrar que SAT está em NP.
  2. Demonstrar que todo problema em NP pode ser reduzido em tempo polinomial a SAT.

1) SAT está em NP

- Essa é a parte fácil da prova.
- Supondo que um oráculo forneceu uma resposta: uma atribuição de verdadeiro e falso para as n variáveis
- Podemos verificar se essa atribuição satisfaz todas as cláusulas em tempo $O(m)$, onde m é o número de cláusulas.
- Logo: SAT está em NP.

2) Todo problema em NP pode ser reduzido em tempo polinomial a SAT.

- Como é impossível demonstrar que essa condição é verdadeira para cada problema em NP, a base da demonstração envolve encontrar uma generalização que englobe todos os problemas em NP.

- O Teorema de Cook-Levin estabelece condições ou padrões para que uma linguagem pertencente a NP seja reconhecida.

- Essas condições são então “mapeadas” como uma fórmula de SAT.

- Como todo problema de NP pode ser reduzido em tempo polinomial a SAT, então SAT é um problema NP-Difícil.

Teorema de Cook-Levin

- Teorema de Cook-Levin: Satisfatibilidade Booleana é NP-Completo

- Para demonstrar que SAT é NP-Completo precisamos:
  1. Demonstrar que SAT está em NP.
  2. Demonstrar que todo problema em NP pode ser reduzido em tempo polinomial a SAT. (SAT é NP-Difícil)

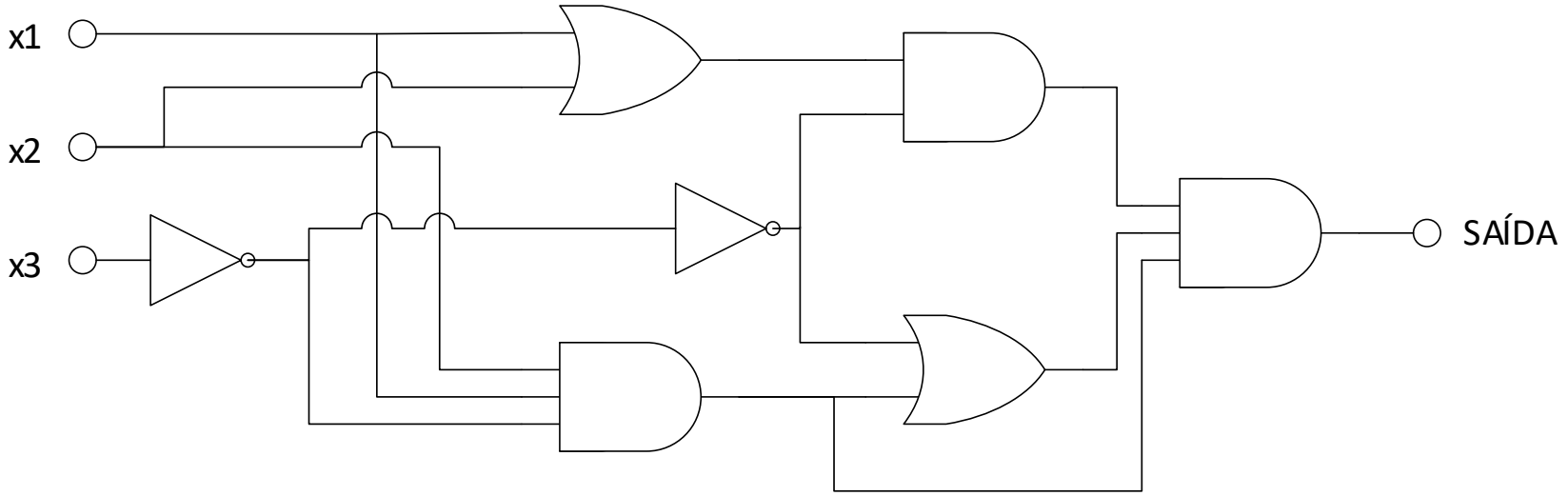
- Uma vez que as duas condições foram demonstradas, temos o primeiro problema NP-Completo: Satisfatibilidade Booleana.

3-SAT e Circuit SAT (CSAT)

- Além da versão padrão de SAT é possível definir variantes do problema:
- 3-SAT: Cada cláusula é formada por exatamente 3 variáveis.
  - $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$

3-SAT e Circuit SAT (CSAT)

- Além da versão padrão de SAT é possível definir variantes do problema:
- Circuit Sat: Dado um circuito lógico formado por portas lógicas E, OU e NÃO com n entradas, onde cada entrada pode assumir o estado LIGADO ou DESLIGADO. Existe uma forma combinação de valores para as entradas tal que o resultado do circuito seja LIGADO?



Como provar que 3-SAT é NP-Completo?

1. Provar que 3-SAT está em NP.
   - Simples: Podemos certificar qualquer resposta SIM em tempo polinomial. Basta verificar se todas as cláusulas são verdadeiras.

2. Provar que SAT $\propto$ 3-SAT
   - Como podemos resolver SAT através de 3-SAT e SAT é NP-Difícil, então 3-SAT também é NP-Difícil

   - Como podemos converter uma instância de SAT em 3-SAT?
     - Converter cláusulas de qualquer tamanho em cláusulas de tamanho exatamente 3

SAT $\propto$ 3-SAT

- Considere uma instância qualquer de SAT com n variáveis $\{u_1, u_2, \ldots, u_n\}$ e m cláusulas $\{C_1, C_2, \ldots, C_n\}$.
- Queremos converter essa instância em uma instância de 3-SAT.
- Cada cláusula C_i pode ter:
  - 1 variável
  - 2 variáveis
  - 3 variáveis (ótimo, esse caso já está OK).
  - 4 ou mais variáveis

SAT $\propto$ 3-SAT

- Se a cláusula C_i tem 1 variável: $C_i = \{ u \}$
  - Vamos criar mais duas variáveis w_1^i e w_2^i
  - E vamos transformar a cláusula C_i em quatro novas cláusulas C_1^i, C_2^i, C_3^i, C_4^i
  - $C_1^i = (u \vee w_1^i \vee w_2^i)$
  - $C_2^i = (u \vee w_1^i \vee \overline{w_2^i})$
  - $C_3^i = (u \vee \overline{w_1^i} \vee w_2^i)$
  - $C_4^i = (u \vee \overline{w_1^i} \vee \overline{w_2^i})$

- Observe que a única forma das quatro cláusulas retornarem verdadeiro é $C_i = \{ u \}$ ser satisfazível. Logo $C_1^i \wedge C_2^i \wedge C_3^i \wedge C_4^i$ é satisfazível se e somente se C_i também é.
- Por exemplo, Se $u =$ Falso w_1^i = Verdadeiro e w_2^i = Verdadeiro, então C_4^i é falso. Testem todas as 8 combinações!

SAT $\propto$ 3-SAT

- Se a cláusula C_i tem 2 variáveis: $C_i = \{ u_1, u_2 \}$
  - Vamos criar mais uma variáveis w_1^i
  - E vamos transformar a cláusula C_i em duas novas cláusulas C_1^i, C_2^i
  - $C_1^i = (u_1 \vee u_2 \vee w_1^i)$
  - $C_2^i = (u_1 \vee u_2 \vee \overline{w_1^i})$

  - Se $C_i = \{ u_1, u_2 \}$ é satisfazível, então C_1^i e C_2^i também são
  - Se $C_i = \{ u_1, u_2 \}$ não é satisfazível, então C_1^i é satisfazível e C_2^i não é ou o exato oposto.

  - Logo: Se $C_1^i \wedge C_2^i$ é satisfazível se, e somente se, C_i também for.

SAT $\propto$ 3-SAT

- Se a cláusula C_i tem 3 variáveis: $C_i = \{u_1, u_2, u_3\}$
  - Vamos apenas replicar a cláusula.
  - Caso fácil...

SAT $\propto$ 3-SAT

- Se a cláusula C_i tem 4 ou mais variáveis: $C_i = \{ u_1, u_2, \ldots, u_k \}$
  - Vamos dividir C_i em duas ou mais cláusulas
  - Para isso vamos criar novas variáveis $\{ w_j^i : 1 \leq j \leq k-3 \}$
  - Vamos criar novas cláusulas correspondentes as anteriores divididas:
  - $\{ C_1^i = (u_1, u_2, w_1^i),\ C_2^i = \left(\overline{w_1^i}, u_3, w_2^i\right),\ C_3^i = \left(\overline{w_2^i}, u_4, w_3^i\right), \ldots, C_k^i = \left(\overline{w_{k-3}^i}, u_{k-1}, u_k\right) \}$

  - Observe que a primeira cláusula contém as duas primeiras variáveis e uma nova variável de “ligação”
  - Já a última cláusula contém uma variável de “ligação”, conectando a penúltima cláusula, e as duas últimas variáveis.
  - Todas as demais cláusulas contém duas variáveis de ligação: uma conectando a cláusulas anterior, outra conectando a próxima e uma variável de C_i

SAT $\propto$ 3-SAT

- Se a cláusula C_i tem 4 ou mais variáveis: $C_i = \{u_1, u_2, \ldots, u_k\}$
  - As novas cláusulas $\{C_1^i, \ldots, C_{k-3}^i\}$ são satisfazíveis se e somente se C_i é satisfazível.

  - (=>) $\{C_1^i, \ldots, C_{k-3}^i\}$ é satisfazível então C_i também é
    - Observe que as TODAS as variáveis de conexão foram marcadas como verdadeiras, cada variável "ajuda" em uma cláusula e atrapalha na próxima. Considerando que a última é uma negação, ela não será satisfeita. Logo, as variáveis de ligação não são suficientes para satisfazer as cláusulas.
    - Desta forma, se $\{C_1^i, \ldots, C_{k-3}^i\}$ é satisfazível então existe uma variável u_i verdadeira, o que também satisfaz C_i.

  - (<=) C_i é satisfazível então as novas cláusulas $\{C_1^i, \ldots, C_{k-3}^i\}$ também são
    - Se C_i é satisfazível, então pelo menos uma variável é verdadeira
    - Seja u_i a variável verdadeira.
    - Podemos atribui falso a variável de conexão da cláusula em que u_i está e em todas as demais cláusulas que a sucedem. Essa atribuição satisfaz as novas cláusulas.

SAT $\propto$ 3-SAT

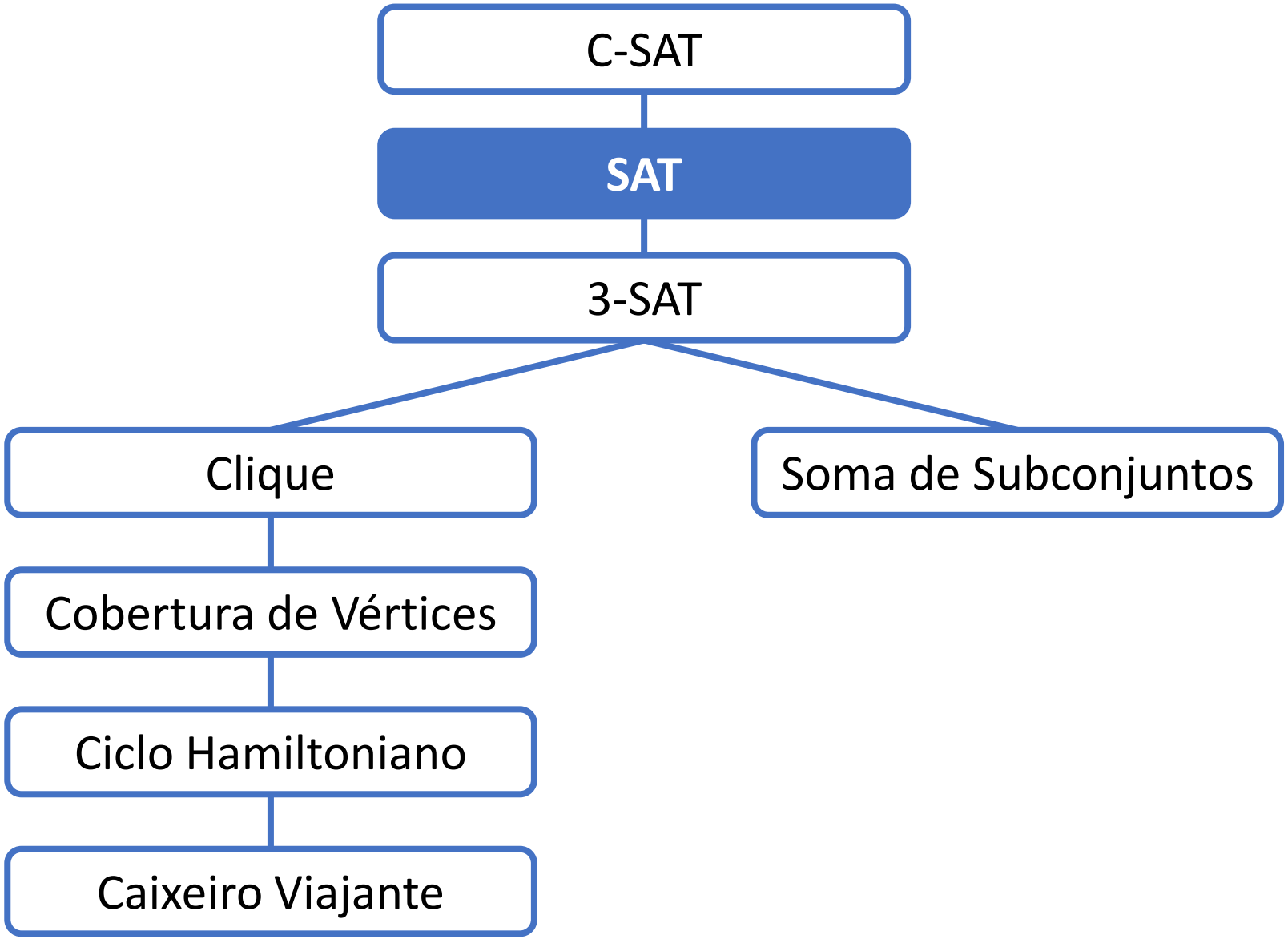
- Considere uma instância qualquer de SAT com n variáveis $\{u_1, u_2, \ldots, u_n\}$ e m cláusulas $\{C_1, C_2, \ldots, C_n\}$.
- Queremos converter essa instância em uma instância de 3-SAT.
- Cada cláusula C_i pode ter:
  - 1 variável ✓
  - 2 variáveis ✓
  - 3 variáveis (ótimo, esse caso já está OK). ✓
  - 4 ou mais variáveis ✓

- Como podemos converter as instâncias em tempo polinomial de forma que Sim para uma implica em Sim para a outra: SAT $\propto$ 3-SAT
- Desta forma: Podemos resolver SAT através de 3-SAT.

A força de SAT

- Como SAT foi categorizado como o primeiro problema NP-Completo então:
  - Para categorizar todos os demais problemas NP-Completo precisamos demonstrar que é possível estabelecer uma transformação polinomial desses problemas em SAT.
  - Isso implica que:
    - Se existir um algoritmo polinomial para qualquer problema em NP-Completo então esse algoritmo pode resolver SAT
    - E se SAT admitir um algoritmo polinomial então qualquer problema em NP pode ser resolvido por este algoritmo.
    - E portanto: P = NP.

A força de SAT



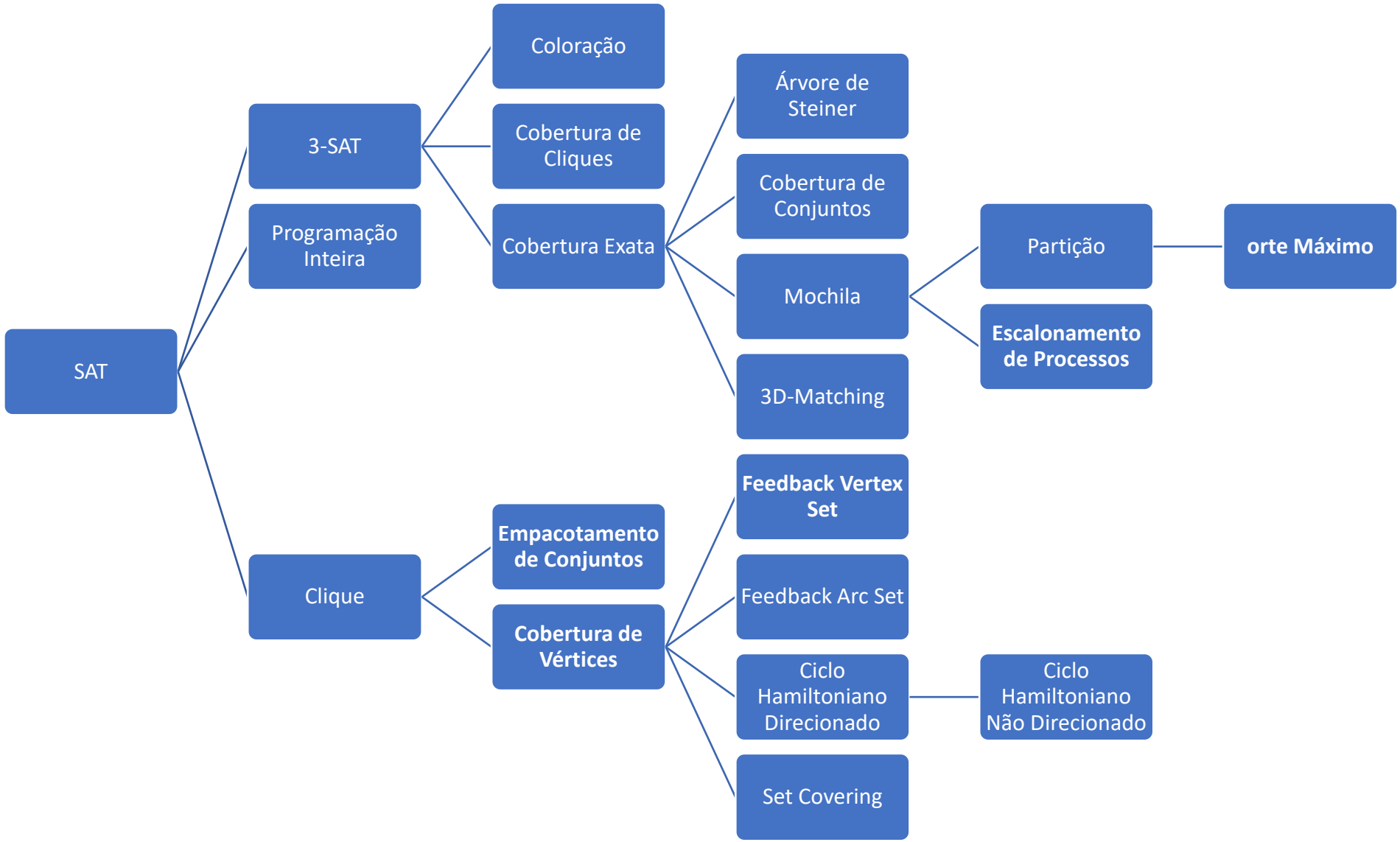
21 problemas NP-completos de Karp

- Richard Karp, baseado no Teorema de Cook-Levin, demonstrou os 21 primeiros problemas NP-Completos.

- Com base nesse problemas, muitos outros puderam ser provados NP-Completos.

- Cook e Karp receberam o Prêmio Turing por suas contribuições à Computação.

21 problemas NP-completos de Karp



Problemas NP-Completo

- O que acontece quando restringimos um problema?
  - Por exemplo: Sabemos que o problema da k-coloração é NP-Completo.
  - Mas o que acontece quando restringimos esse problema: $k = 3$ em grafos planares.
  - O nível de dificuldade permanece?
  - Não! Muitas vezes ao restringir o problema, ele fica tratável computacionalmente (passa a admitir um algoritmo de tempo polinomial).

- Logo, quando identificamos que um problema é uma restrição de outro:
  - Ou ele tem o mesmo nível de dificuldade (a restrição não facilitou)
  - Ou ele é mais fácil (as instâncias restritas podem ser resolvidas mais facilmente).

Problemas NP-Completo

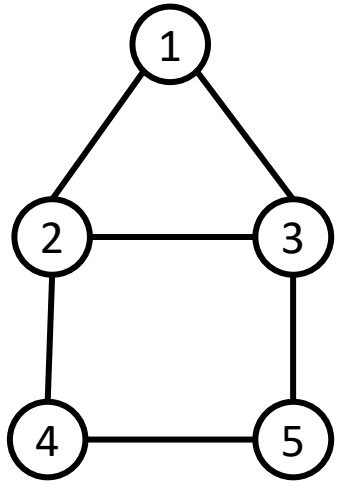
- O que acontece quando generalizamos um problema?
  - O problema já é complicado em uma instância controlada e reduzida.
  - O que acontece quando consideramos qualquer instância?
  - O problema fica mais fácil ou mais difícil?
  - Ele fica mais difícil.

- Logo, quando percebemos que um problema é uma generalização de um outro (uma versão mais abrangente)
  - O primeiro problema é pelo menos tão ou mais difícil que o segundo.

- Isso é trivial: Se as instâncias difíceis estão contidas no universo da generalização, então a generalição não poderia ser mais fácil...

Para pensar

- Como poderíamos demonstrar que Clique é NP-Completo?
- Problema da Clique:
  - Entradas: Grafo $G = (V, E)$ e inteiro k
  - Pergunta: Existe um subconjunto $V' \subseteq V$, com pelo menos k vértices tal que todos os vértices desse conjunto são vizinhos entre si?
  - Saída: Sim ou Não.



k = 4? NÃO.

k = 3? SIM.

Teorema: Clique é NP-completo

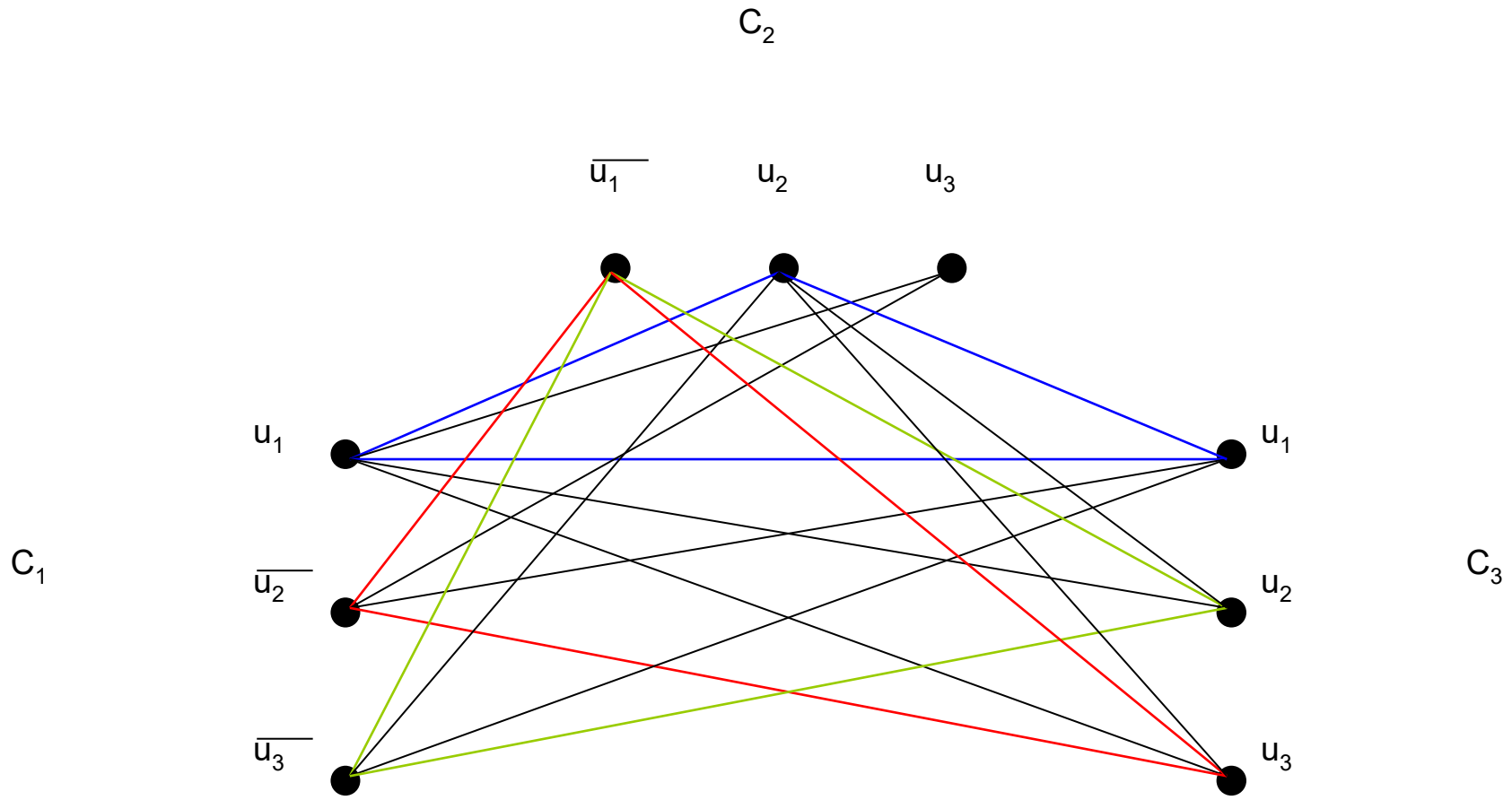
- Clique $\in$ NP: Para um $G = (V, E)$ usamos o conjunto V' V de vértices da clique como um certificado SIM para G. Verificar se V' é uma clique leva um tempo polinomial: para cada par u, v V', ver se $(u, v) \in E$.

- Vamos provar que Clique é NP-Difícil através de 3-SAT. Dada uma fórmula booleana F com m cláusulas $C_1, C_2, \ldots, C_m$, será construído um grafo G tal que F é satisfatível se e somente se G possui uma clique de tamanho m.

- Para cada cláusula $C_r = (u_1^r, u_2^r, u_3^r)$ será adicionada uma tripla v_1^r, v_2^r, v_3^r de vértices em V.
- Vamos adicionar uma aresta entre dois vértices v_i^r, v_j^s se:
  - v_i^r e v_j^s estão em triplas diferentes, ou seja, r s, e
  - seus literais correspondentes são consistentes, ou seja, u_i^r não é a negação de u_j^s.

Teorema: Clique é NP-completo

- Exemplo: $F = (u_1 \vee \overline{u_2} \vee \overline{u_3}) \wedge (\overline{u_1} \vee u_2 \vee u_3) \wedge (u_1 \vee u_2 \vee u_3)$



Teorema: Clique é NP-completo

- ($\rightarrow$) Seja F satisfatível. Então, cada cláusula C_r contém pelo menos um literal Verdadeiro e cada literal corresponde a um vértice v_i^r. Se selecionarmos um literal verdadeiro de cada cláusula, obtemos um conjunto V' com m vértices. Para quaisquer dois vértices v_i^r e $v_j^s \in V'$, onde $r \neq s$, ambos correspondem a literais u_i^r e u_j^s com valor V na atribuição verdade e, assim, u_i^r não pode ser negação de u_j^s. Logo, a aresta $(v_i^r, v_j^s) \in E$, e V' é clique.
-
- ($\leftarrow$) Suponha que G tenha uma clique V', |V|=m. Nenhuma aresta em G conecta vértices na mesma tripla; logo, V' contém exatamente um vértice por tripla. Podemos atribuir V a cada literal u_i^r tal que $v_i^r \in V'$, sem tornar V literais complementares, já que G não contém arestas entre literais inconsistentes. Como cada cláusula é satisfeita, F é satisfeita.

Primeiro Trabalho de Computabilidade

- Tema: Problemas NP-Completos e Reduções Polinomiais
- O trabalho deverá ser feito em duplas

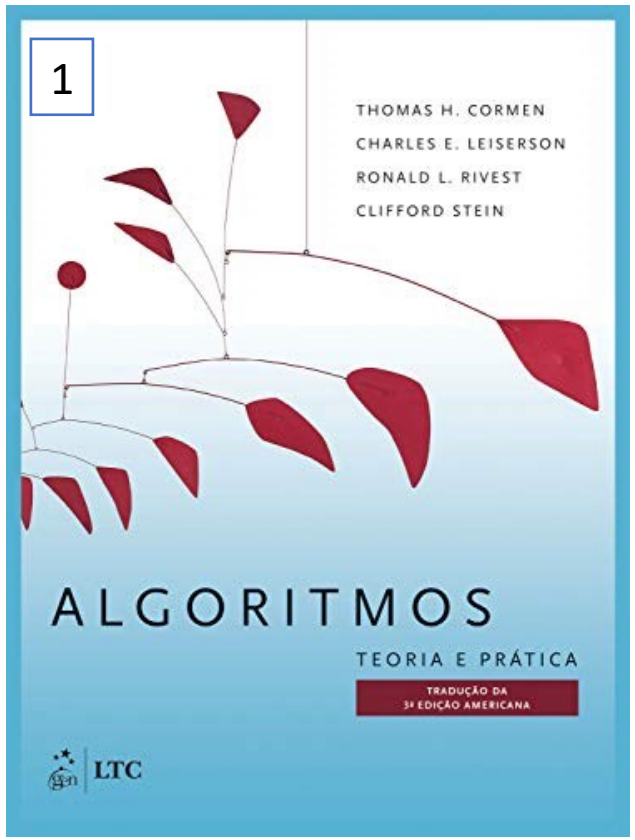
- Fase 1) Identifiquem problemas NP-Completos que:
  - Sua dupla julgue esse problema interessante
  - Vocês consigam acompanhar/entender a transformação polinomial que caracterizou esse problema como NP-Completo.
  - Na próxima aula, apresentar uma lista com três problema e suas respectivas transformações polinomiais (diferentes das vistas em sala).
  - A partir das listas de problemas e duplas, vamos buscar uma atribuição de temas tal que cada dupla tenha um tema diferente.
  - Prazo: Uma semana

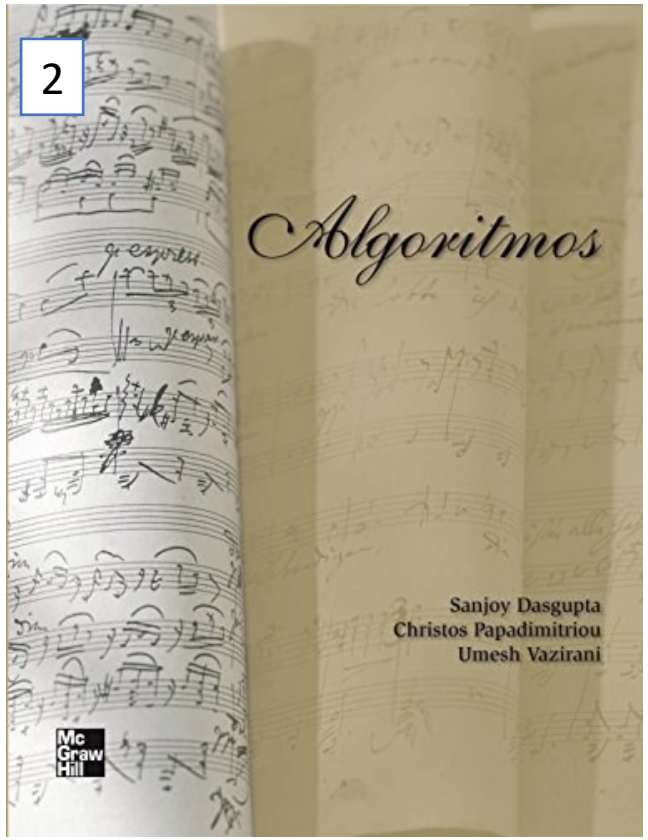
Primeiro Trabalho de Computabilidade

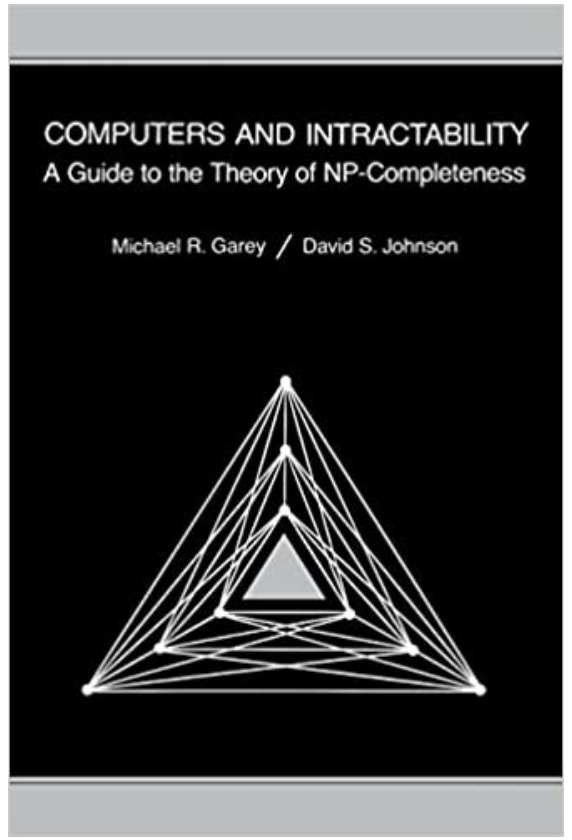
- Tema: Problemas NP-Completos e Reduções Polinomiais
- O trabalho deverá ser feito em duplas

- Fase 2) Na aula seguinte (prazo 15 dias a contar dessa aula), apresentar a prova de que o problema escolhido é de fato NP-Completo.

Bibliografia Recomendada para o Trabalho:







1. http://catalogo-redesirius.uerj.br/sophia_web/index.asp?codigo_sophia=301976
2. http://catalogo-redesirius.uerj.br/sophia_web/index.asp?codigo_sophia=301974
3. http://catalogo-redesirius.uerj.br/sophia_web/index.asp?codigo_sophia=302475