Melhores momentos

AULA PASSADA

Complexidade de tempo

Seja M uma máquina de Turing **determinística** que pára sobre todas as entradas.

O **tempo de execução** ou **complexidade de tempo** ou **consumo de tempo** de M é a função t de $\mathbb{Z}_{\geq}$ em $\mathbb{Z}_{\geq}$ tal que $t(n)$ é o número máximo de passos de M com uma entrada de comprimento n.

Se $t(n)$ é o tempo de execução de M, dizemos que M **roda em tempo** $t(n)$ e que M é uma máquina de Turing **de tempo** $t(n)$.

Classes de complexidade

Seja $t(n)$ uma função de $\mathbb{Z}_{\geq}$ em $\mathbb{Z}_{\geq}$.

A **classe de complexidade de tempo** TIME$(t(n))$ é formada por todas as linguagens que são decidíveis por uma máquina de Turing uma fita que consome tempo $\mathrm{O}(t(n))$:

$$\text{TIME}(t(n)) = \{L : L \text{ é decidida por uma MT que roda em tempo } \mathrm{O}(t(n))\}.$$

Exemplos:

- $A = \{0^k 1^k : k \geq 0\}$ está em TIME(n^2)
- $\{0^{2^k} : k \geq 0\}$ está em TIME$(n \lg n)$ (AULA 2)

AULA 7

A classe P

MS 7.2

A classe P

P é classe de linguagens decidíveis em tempo polinomial por uma máquina de Turing determinística de uma fita:

$$\mathsf{P} = \bigcup_k \mathsf{TIME}(n^k).$$

A classe P é importante pois

1. P é invariante para todos os modelos de computação polinomialmente equivalentes a uma MT com uma fita.
2. P contém a classe dos problemas que são resolvidos eficientemente.

Exemplo 1: CAM

Queremos um algoritmo que decida se uma cadeia w está na linguagem

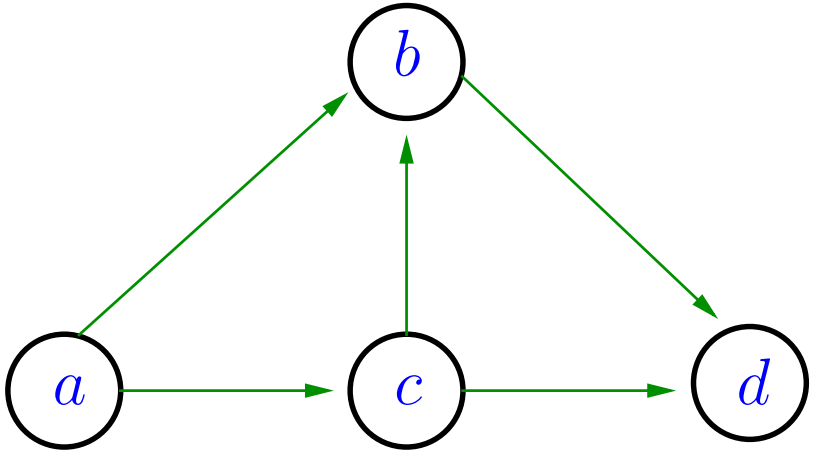
$$\text{CAM} = \{\langle G, s, t\rangle : G \text{ é um grafo orientado que possui um caminho do nó } s \text{ ao nó } t\}$$

$\langle G, s, t\rangle =$ codificação "razoável" de um grafo G e dois de seus vértices s e t.

Codificações "razoáveis" de grafos tem comprimento polinomial no número de nós e arcos

Codificações razoáveis de grafos

Uma **matriz de adjacência** de um grafo orientado (N, A) é uma matriz com valores em $\{0, 1\}$, e indexada por $N \times N$, onde cada entrada (u, v) da matriz tem valor 1 se $uv \in A$, e 0 caso contrário.

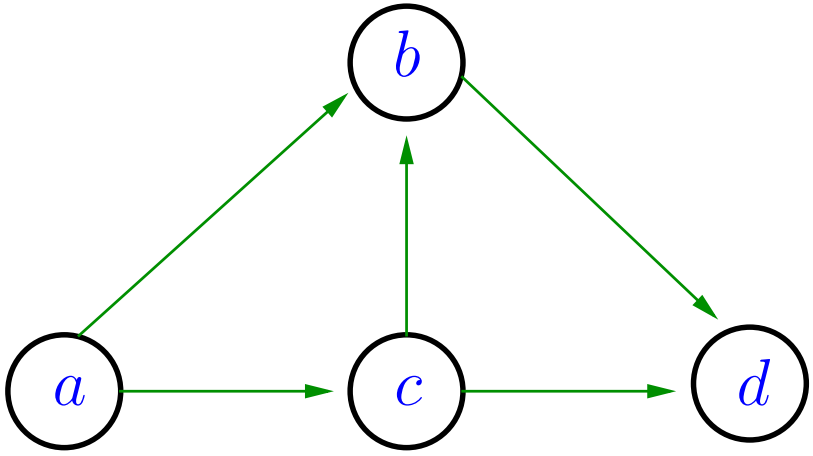


	a	b	c	d
a	0	1	1	0
b	0	0	0	1
c	0	1	0	1
d	0	0	0	0

Consumo de espaço: $\Theta(|N|^2)$ fácil de implementar

Codificações razoáveis de grafos

Uma **matriz de incidências** de um grafo orientado (N, A) é uma matriz indexada por $N \times A$ e com valores em $\{-1, 0, +1\}$, onde cada entrada (k, uv) é -1 se $k = u$, $+1$ se $k = v$, e 0 em caso contrário.



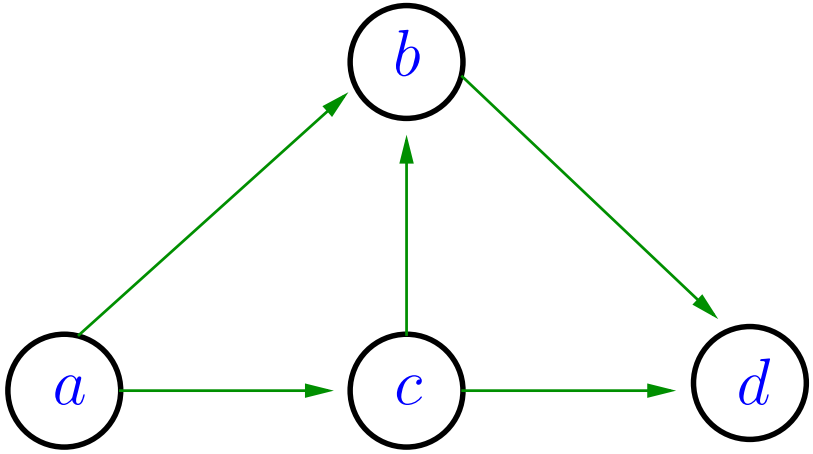
	ab	ac	cb	cd	bd
a	-1	-1	0	0	0
b	$+1$	0	$+1$	0	-1
c	0	$+1$	-1	-1	0
d	0	0	0	$+1$	$+1$

Consumo de espaço: $\Theta(|N|\,|A|)$

Interessante do ponto de vista de programação linear.

Codificações razoáveis de grafos

Na representação de um grafo orientado (N, A) através de **listas de adjacências** tem-se, para cada nó u, uma lista dos arcos deixando u. Desta forma, para cada nó u, o conjunto $A(u)$ é representado por uma lista.



$A(a)$: ab, ac

$A(b)$: bd

$A(c)$: cb, cd

$A(d)$:

Consumo de espaço: $\Theta(n+m)$ (linear)

Manipulação eficiente

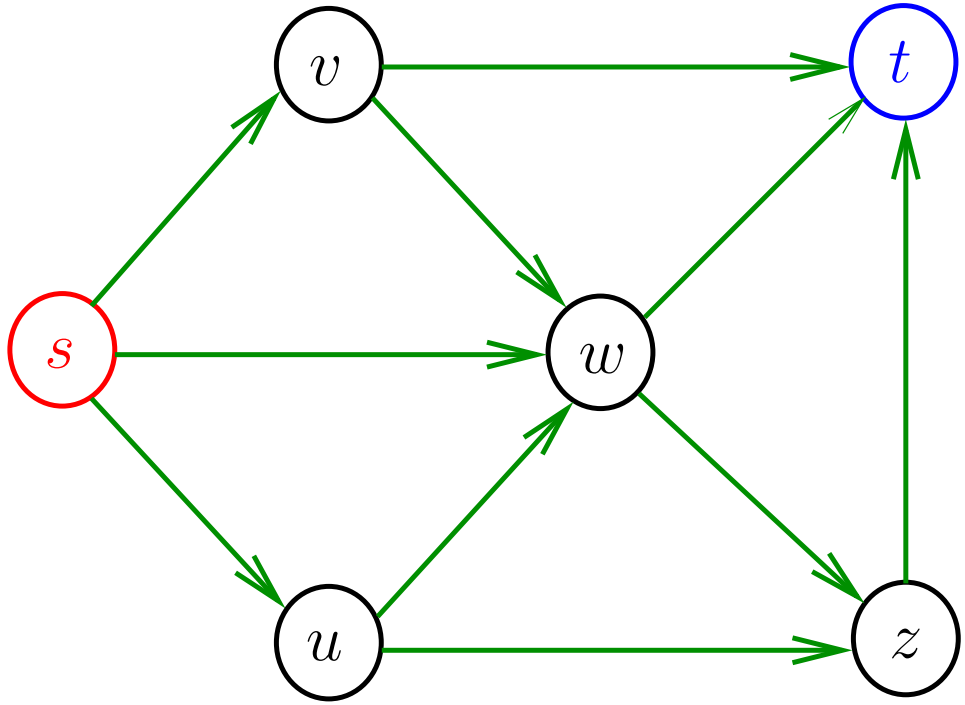
Busca

Problema de busca: Dados vértices s e t de um grafo, encontrar um caminho de s a t

Busca

Problema de busca: Dados vértices s e t de um grafo, encontrar um caminho de s a t

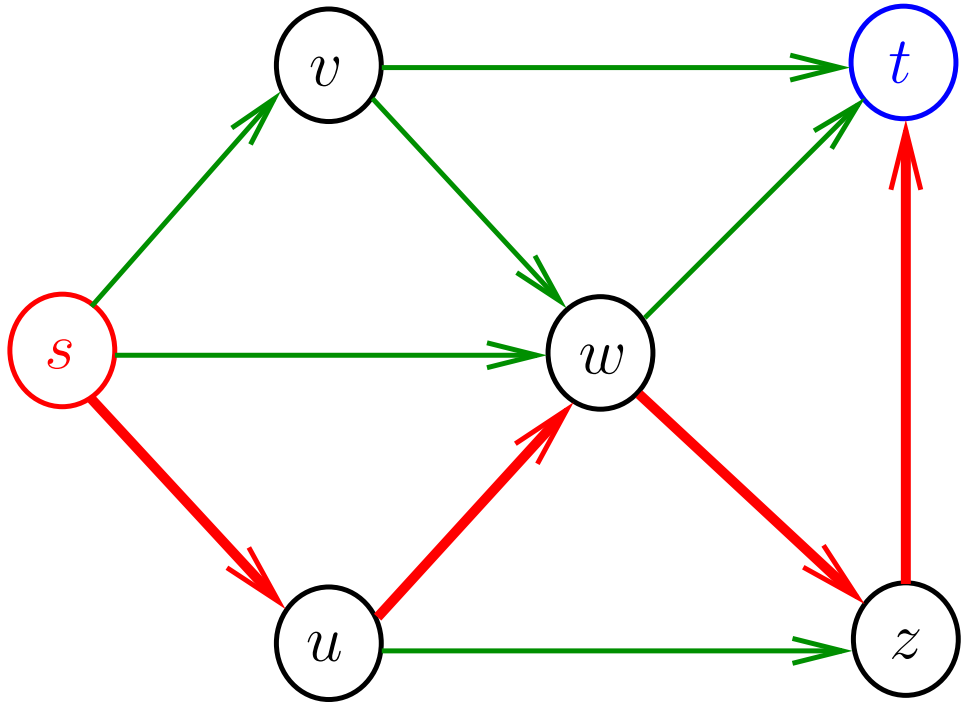
Entra:



Busca

Problema de busca: Dados vértices s e t de um grafo, encontrar um caminho de s a t

Sai:



Algoritmo polinomial para CAM

$M =$ "com entrada $\langle G, s, t \rangle$, a codificação de um grafo G e dois de seus nós s e t:

1. Ponha uma marca sobre o nó s.
2. Repita o seguinte até que nenhum novo nó sejar marcado:
3. Faça uma varredura de todas as arestas de G. Se uma aresta uv for encontrada com u marcado e v não marcado, marque v.
4. Se t estiver marcado *aceite*. Caso contrário *rejeite*."

Consumo de tempo

Suponha $G = (N, A)$.

O passo 1 e 4 são executados apenas 1 vez.

O passo 3 é executado $\leq |A|$.

Os passos 1 e 4 são facilmente implementados de maneira a consumir tempo polinomial.

O passo 3 cosiste em varrer a lista de arcos e verificar se existe algum com uma ponta marcada e outra ponta não marcada.
Isto também pode ser feito em tempo polinomial.

Busca genérico

Recebe dois nós s e t de um grafo $G = (N, A)$ e decide se existe um caminho de s a t.

BUSCA-GENÉRICO (N, E, s, t)

1 $s \leftarrow$ nó qualquer em N

2 $S \leftarrow \{s\}$
3 **enquanto** existe $uv \in A$ com $u \in S$ e $v \notin S$ **faça**
4 $S \leftarrow S \cup \{v\}$

5 **se** $t \in S$
6 **então** existe o caminho $\triangleright$ aceite $\langle G, s, t \rangle$
7 **senão** não existe o caminho $\triangleright$ rejeite $\langle G, s, t \rangle$

Conclusão

CAM está em P.

Exemplo 2: PRI-MES

Queremos um algoritmo que decida se uma cadeia w está na linguagem

$$\text{PRI-MES} = \{\langle a, b\rangle : a \text{ e } b \text{ são primos entre si}\}$$

a e b são primos entre si se 1 é o maior número inteiro que divide ambos.

$\langle a, b\rangle =$ codificação "razoável" de a e b.

Codificações "razoáveis" de a e b tem comprimentos essencialmente $\lg a$ e $\lg b$, respectivamente.

Algoritmo de Euclides para PRI-MES

$E =$ "com entrada $\langle a, b \rangle$, a codificação de a e b:

1. Repita até que $b = 0$:
2. $\quad a \leftarrow a \bmod b$
3. $\quad a \leftrightarrow b$
4. Devolva a."

$R =$ "com entrada $\langle a, b \rangle$, a codificação de a e b:

1. Rode E sobre $\langle a, b \rangle$:
2. Se o resultado é 1, *aceite*, senão *rejeite*.

Consumo de tempo

Depois de 2 execuções seguidas dos passo 2 e 3 de E o valor de b cai no mínimo pela metade.

Assim o número de vezes que os passo 2 e 3 são executados é $\leq 1 + \lg 2b$.

Cada passo de E consome tempo polinomial, logo, a complexidade de E é polinomial.

Conclusão

PRI-MES está em P.

Máximo divisor comum

CLRS 31.1 e 31.2

Divisibilidade

Suponha que a, b e d são números inteiros.

Dizemos que d **divide** a se $a = k\,d$ para algum número inteiro k.
$d \mid a$ é uma abreviação de "d divide a

Se d divide a, então dizemos que a é um **multiplo** de d.

Se d divide a e $d > 0$, então dizemos que d é um **divisor** de a

Se d divide a e d divide b, então d é um **divisor comum** de a e b.

Exemplo:
os divisores de 30 são: $1, 2, 3, 5, 6, 10, 15$ e 30
os divisores de 24 são: $1, 2, 3, 4, 6, 8, 12$ e 24
os divisores comuns de 30 e 24 são: $1, 2, 3$ e 6

Máximo divisor comum

O **máximo divisor comum** de dois números inteiros a e b, onde pelo menos um é não nulo, é o maior divisor comum de a e b.

O máximo divisor comum de a e b é denotado por mdc(a, b).

Exemplo:
máximo divisor comum de 30 e 24 é 6
máximo divisor comum de 514229 e 317811 é 1
máximo divisor comum de 3267 e 2893 é 11

Problema: Dados dois números inteiros não-negativos a e b, determinar mdc(a, b).

Café com leite

Recebe números inteiros não-negativos a e b e devolve mdc(a, b).

Café-Com-Leite (a, b) $\triangleright$ supõe $a \neq 0$ ou $b \neq 0$

```
1   se b = 0 então devolva a
2   se a = 0 então devolva b
3   d ← b
4   enquanto d ∤ a ou d ∤ b faça
5         d ← d − 1
6   devolva d
```

Relações invariantes: na linha 4 vale que

(i0) na linha 4 vale que $1 \leq d \leq b$;
(i1) na linha 4 vale que $k \nmid a$ ou $k \nmid b$ para cada $k > d$; e
(i2) na linha 5 vale que $d \nmid a$ ou $d \nmid b$.

Consumo de tempo

linha	consumo de todas as execuções da linha
1-2	$\Theta(1)$
3	$\Theta(1)$
4	$\mathrm{O}(b)$
5	$\mathrm{O}(b)$
6	$\Theta(1)$
total	$\Theta(3) + \mathrm{O}(b) = \mathrm{O}(b)$

Quando a e b são relativamente primos, ou seja $\mathrm{mdc}(a, b) = 1$, o consumo de tempo do algoritmo é $\Theta(b)$.

Conclusão

O consumo de tempo do algoritmo Café-Com-Leite é $\mathrm{O}(b)$.

No pior caso, o consumo de tempo do algoritmo Café-Com-Leite é $\Theta(b)$.

Se o valor de b dobra, o consumo de tempo pode dobrar.

Seja $\beta := |b|$ o número de bits ou **comprimento** de b.
O consumo de tempo do algoritmo Café-Com-Leite é $\mathrm{O}(2^{\beta})$.

Brincadeira

Usei uma implementação do algoritmo Café-Com-Leite para determinar mdc$(2147483647, 2147483646)$.
Vejam quanto tempo gastou:

```
meu_prompt> time mdc 2147483647 2147483646
mdc: mdc de 2147483647 e 2147483646 e' 1.

real    2m49.306s
user    1m33.412s
sys     0m0.099s
```

Algoritmo de Euclides

Recebe números inteiros não-negativos a e b e devolve mdc(a, b).

```
EUCLIDES (a, b)  ▷ supõe a ≠ 0 ou b ≠ 0
1   se b = 0
2       então devolva a
3       senão devolva EUCLIDES (b, a mod b)
```

"a mod b" é o resto da divisão de a por b.

Exemplo: mdc$(12, 18) = 6$, pois

```
mdc(12,18)
  mdc(18,12)
    mdc(12,6)
      mdc(6,0)
```

Correção

A correção do algoritmo EUCLIDES é baseada no seguinte fato.

Suponha que $a \geq 0$ e $b > 0$. Para cada $d > 0$ vale que

$$d \mid a \text{ e } d \mid b \quad \text{ se e só se } \quad d \mid b \text{ e } d \mid a \bmod b.$$

Em outras palavras, os pares (a, b) e $(b, a \bmod b)$ têm os mesmos divisores.

Outro exemplo

```
mdc(317811,514229)
  mdc(514229,317811)
    mdc(317811,196418)
      mdc(196418,121393)
        mdc(121393,75025)
          mdc(75025,46368)
            mdc(46368,28657)
              mdc(28657,17711)
                mdc(17711,10946)
                  mdc(10946,6765)
                    mdc(6765,4181)
                      mdc(4181,2584)
                        mdc(2584,1597)
                          mdc(1597,987)
                            mdc(987,610)
                              mdc(610,377)
```

Outro exemplo (cont.)

```
mdc(377,233)
  mdc(233,144)
    mdc(144,89)
      mdc(89,55)
        mdc(55,34)
          mdc(34,21)
            mdc(21,13)
              mdc(13,8)
                mdc(8,5)
                  mdc(5,3)
                    mdc(3,2)
                      mdc(2,1)
                        mdc(1,0)
```

$\text{mdc}(317811, 514229) = 1$

Mais brincadeirinha

Usei uma implementação do algoritmo EUCLIDES para determinar mdc$(2147483647, 2147483646)$.
Vejam quanto tempo gastou:

```
meu_prompt> time euclides 2147483647 214748364
mdc(2147483647,2147483646)
  mdc(2147483646,1)
    mdc(1,0)
euclides: mdc de 2147483647 e 2147483646 e' 1.

real    0m0.007s
user    0m0.002s
sys     0m0.004s
```

Consumo de tempo

O consumo de tempo do algoritmo EUCLIDES é proporcional ao número de chamadas recursivas.

Suponha que a função EUCLIDES faz k chamadas recursivas e que na 1a. chamada ao algoritmo tem-se que $a \geq b > 0$.

Sejam

$$(a, b) = (a_0, b_0), (a_1, b_1), \ldots (a_k, b_k) = (\text{mdc}(a, b), 0)$$

os valores dos parâmetros no início de cada chamada.

Portanto, que $a_{i+1} = b_i$ e $b_{i+1} = a_i \bmod b_i$ para $i = 1, 2, \ldots, k$.

Número de chamadas recursivas

Para números inteiros p e q, $p \geq q > 0$ vale que

$$p \bmod q < p/2$$

Desta forma,

$$\begin{array}{lllllllll}
b_2 & = & a_1 \bmod b_1 & = & b_0 \bmod b_1 & < & b_0/2 & = & b/2^1 \\
b_4 & = & a_3 \bmod b_3 & = & b_2 \bmod b_3 & < & b_2/2 & \leq & b/2^2 \\
b_6 & = & a_5 \bmod b_5 & = & b_4 \bmod b_5 & < & b_4/2 & \leq & b/2^3 \\
b_8 & = & a_7 \bmod b_7 & = & b_6 \bmod b_7 & < & b_6/2 & \leq & b/2^4 \\
\vdots & = & \vdots & = & \vdots & < & \vdots & \leq & \vdots
\end{array}$$

O valor do 2o. parâmetro é reduzido a menos da sua metade a cada 2 chamadas recursivas.

Número de chamadas recursivas

Seja t o número inteiro tal que

$$2^t \leq b < 2^{t+1},$$

ou seja, $t = \lfloor \lg b \rfloor$.

Da desigualde estrita concluímos que o número de chamadas recursivas é $\leq 2\lfloor \lg b \rfloor + 1$.

Por exemplo, para $a = 514229$ e $b = 317511$ temos que

$$2\lg(b) + 1 = 2\lg(317511) + 1 < 2 \times 18.3 + 1 = 37.56$$

e o número de chamadas recursivas feitas por EUCLIDES $(514229, 317511)$ é 27.

Conclusões

O consumo de tempo do algoritmo EUCLIDES é $\mathrm{O}(\lg b)$.

Seja $\beta := |b|$ o número de bits ou **tamanho** de b.

O consumo de tempo do algoritmo EUCLIDES é $\mathrm{O}(\beta)$.

Se o valor de β dobra, o consumo de tempo pode dobrar.

Se o tamanho de b dobra, o consumo de tempo pode dobrar.

$b \times \lg b$

b	$\lfloor \lg b \rfloor$
4	2
5	2
6	2
10	3
64	6
100	6
128	7
1000	9
1024	10
1000000	19
1000000000	29
$\vdots$	$\vdots$

A classe NP

MS 7.3

Verificador

Um **verificador** é uma máquina de Turing que sempre pára.

A **linguagem de um verificador** V é

$$L_V = \{w : V \text{ aceita } \langle w, c \rangle \text{ para alguma cadeia } c\}$$

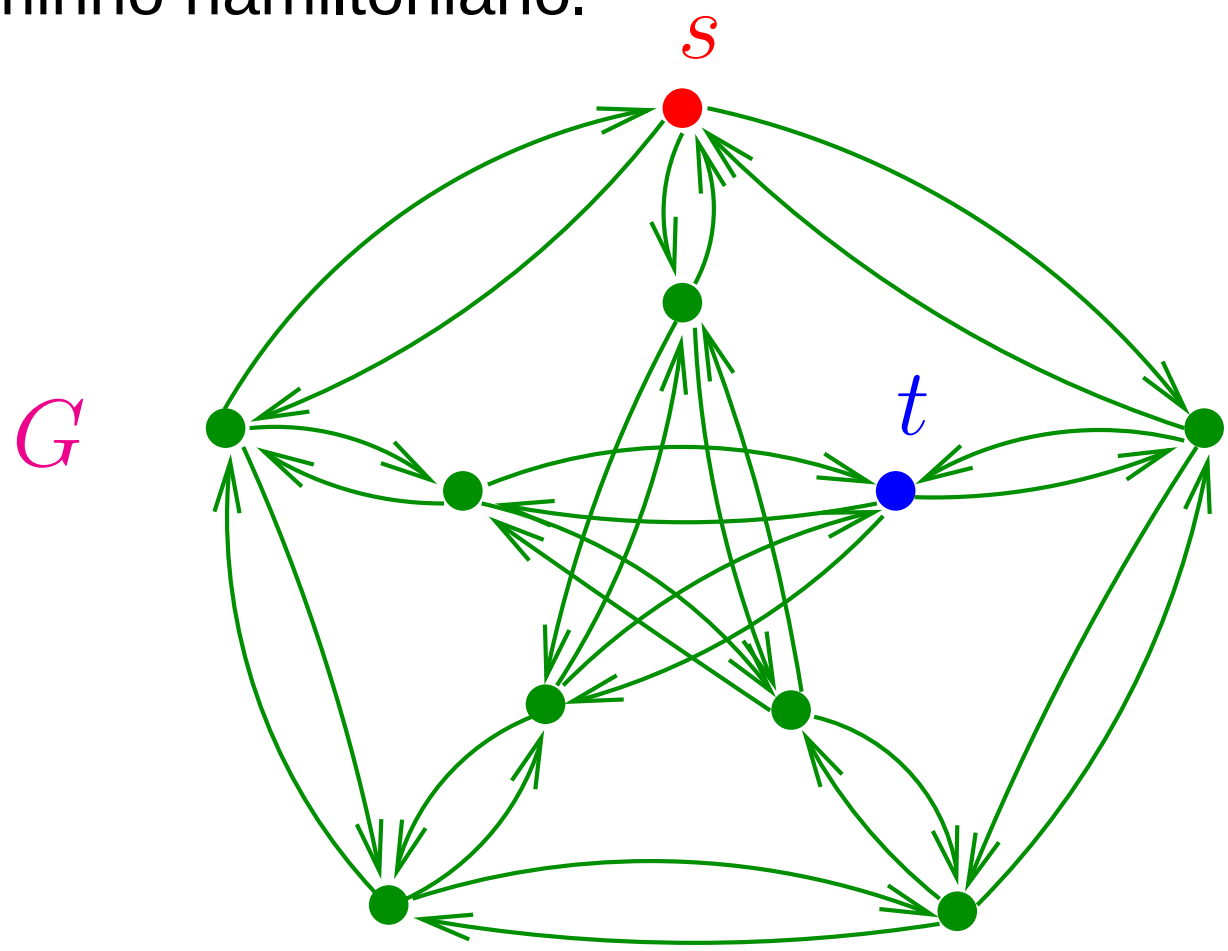
Um **verificador polinomial** consome tempo polinomial no comprimento de w.

A cadeia c é chamada de **prova** ou **certificado** de pertinência de w em L.

V = Artur
c = certificado que Merlin fornece a Artur.

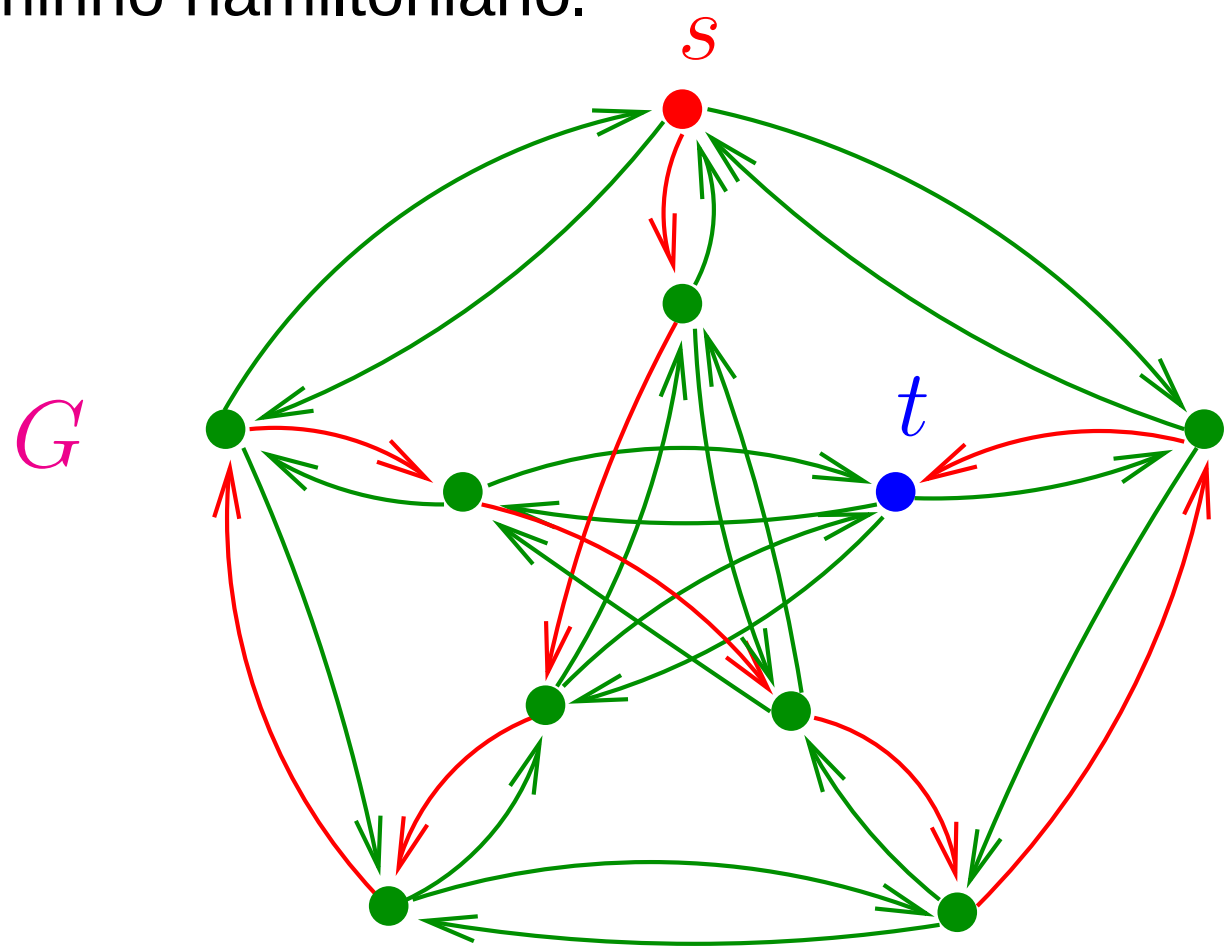
CAMHAM

Problema: Dado um grafo orientado G e nós s e t encontrar um st-caminho hamiltoniano.



CAMHAM

Problema: Dado um grafo orientado G e nós s e t encontrar um st-caminho hamiltoniano.



CAMHAM

Queremos um algoritmo que decida se uma cadeia w está na linguagem

$$\text{CAMHAM} = \{\langle G, s, t\rangle : G \text{ é um grafo orientado que possui um caminho hamiltoniano de } s \text{ a } t\}$$

$\langle G, s, t\rangle =$ codificação "razoável" de um grafo G e dois de seus vértices s e t.

Codificações "razoáveis" de $G = (N, A)$ tem comprimento polinomial em $|N| + |A|$.

CAMHAM

Certificado para pertinência de $\langle G, s, t \rangle$ em CAMHAM é um caminho hamiltoniano P de s a t em G.

Um verificador polinomial V recebe $\langle G, s, t \rangle, \langle P \rangle$ e verifica se P é um caminho hamiltoniano de s a t.

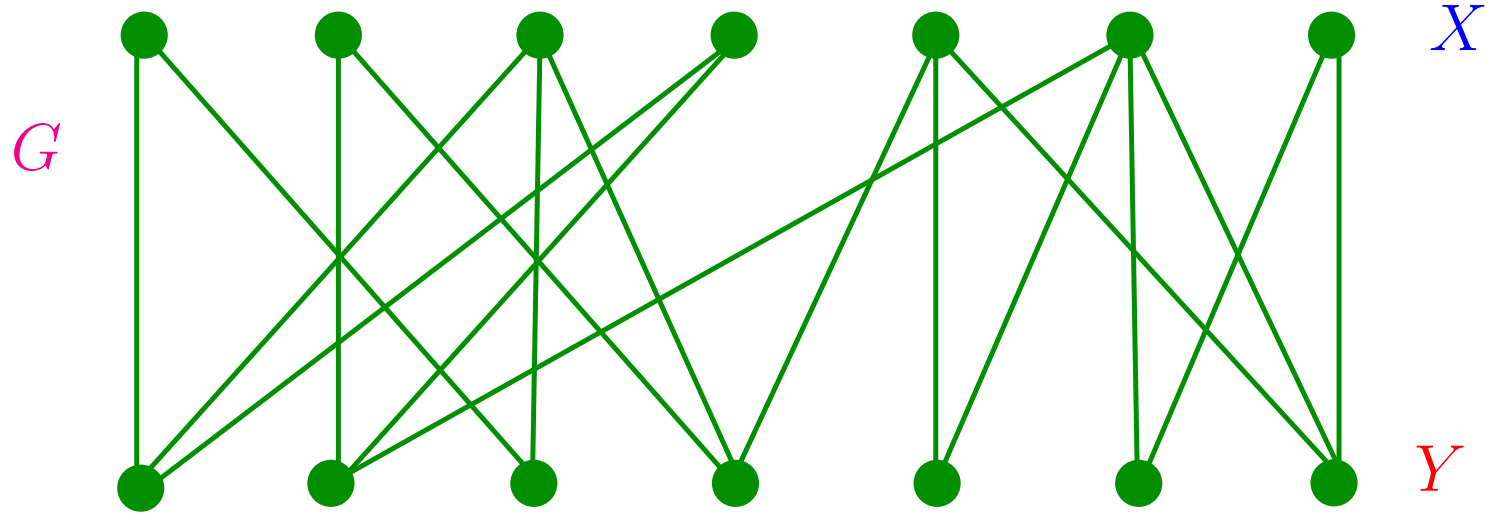
Conclusão: CAMHAM possui um verificador polinomial

Não se sabe CAMHAM está em P.

CASAMENTO

Queremos um algoritmo que decida se uma cadeia w está na linguagem

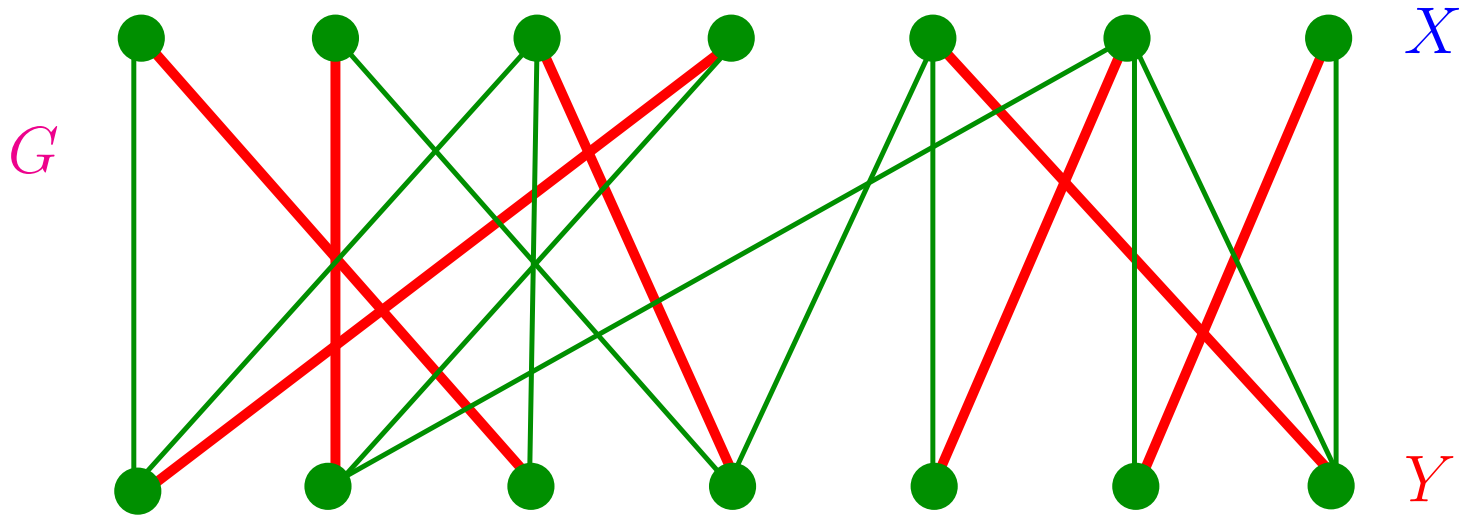
CASAMENTO $= \{\langle G \rangle : G$ é um grafo bipartido que possui um emparelhamento perfeito$\}$



CASAMENTO

Queremos um algoritmo que decida se uma cadeia w está na linguagem

CASAMENTO $= \{\langle G \rangle : G$ é um grafo bipartido que possui um emparelhamento perfeito$\}$



CASAMENTO

Certificado para pertinência de $\langle G \rangle$ em CASAMENTO é emparelhamento perfeito M.

Um verificador polinomial V recebe $\langle G \rangle, \langle M \rangle$ e verifica se M é um emparelhamento perfeito.

Conclusão: CASAMENTO possui um verificador polinomial

Sabe-se que CASAMENTO está em P.

COMPOSTO

Queremos um algoritmo que decida se uma cadeia w está na linguagem

$$\text{COMPOSTO} = \{\langle k \rangle : k = i \times j,\ i, j > 1\}$$

$\langle k \rangle =$ codificação "razoável" de um número inteiro k.

Codificações "razoáveis" de k tem comprimento essencialmente $\lg k$.

Sabe-se que COMPOSTO está em P.

MDC

$$\text{MDC} = \{\langle a, b, d\rangle : a, b, d \text{ são números inteiros tais que } \text{mdc}(a, b) = d\}$$

Certificado para pertinência de $\langle a, b, d\rangle$ em MDC são números inteiros x e y tais que

$$ax + by = d.$$

Um verificador polinomial para MDC recebe $\langle G, s, t\rangle$, $\langle x, y\rangle$ e verifica se $ax + by = d$.

Como vimos MDC está em P.

Certificados

Certificado para MDC: inteiros x, y tais que $ax + by = d$

Certificado para PRI-MES: inteiros x, y tais que $ax + by = 1$

Certificado para CASAMENTO: emparelhamento perfeito

Certificado para COMPOSTO: um divisor de k maior que 1

Certificado para CAMHAM: caminho hamiltoniano de s a t

Certificado para CAM: caminho de s a t

A classe NP

NP é a classe de linguagens que tem um verificador polinomial.

Exemplos:

- CAM está em NP (e também em P)
- PRI-MES está em NP (e também em P)
- CASAMENTO está em NP (e também em P)
- COMPOSTO está em NP (e também em P)
- CAMHAM está em NP (**não se sabe** se está P)