TEORIA DE GRAFOS E COMPUTABILIDADE

COLORAÇÃO

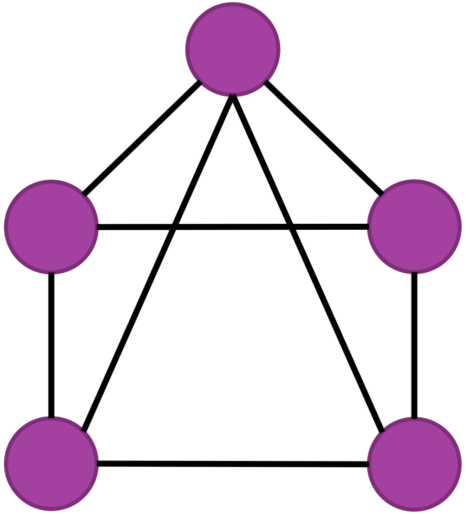
Prof. Alexei Machado

Coloração de grafos

- Francis Guthrie (1852) percebeu que era possível colorir o mapa da Inglaterra usando apenas 4 cores
- Mas... Seriam 4 cores suficientes para colorir qualquer decomposição do plano em regiões?
- Em 1976, usando grafos, Haken e Appel mostram que a resposta era afirmativa

Coloração de grafos

- Dado um grafo G, como colorir seus vértices de maneira que vértices adjacentes não tenham as mesmas cores?
- Qual o menor número possível para esta ***coloração***?

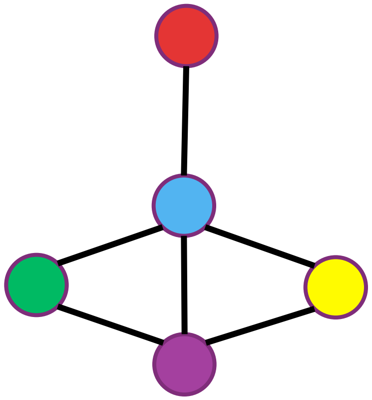


Coloração de grafos

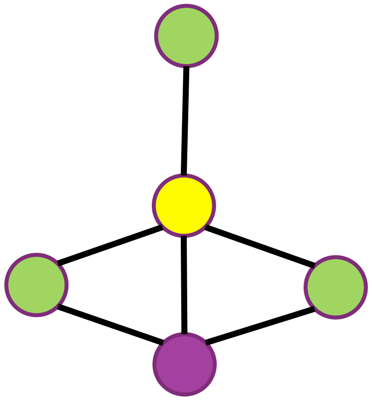
- Dado um grafo G conexo simples, uma ***coloração*** de G é uma atribuição de cores aos vértices de G de maneira que cores diferentes são atribuídas a vértices adjacentes
- Se existe uma coloração para um grafo G que utiliza K cores, então G é um grafo **K-colorido**

Número cromático

- O **número cromático** de um grafo G, X(G), é o **menor** número K para o qual G é K-colorido



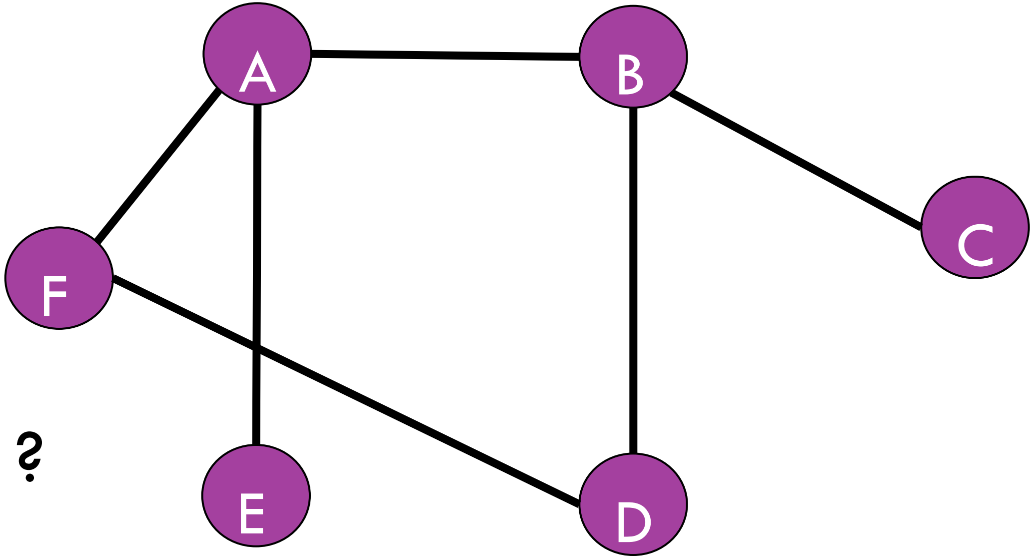
5-colorido



X(G) = 3

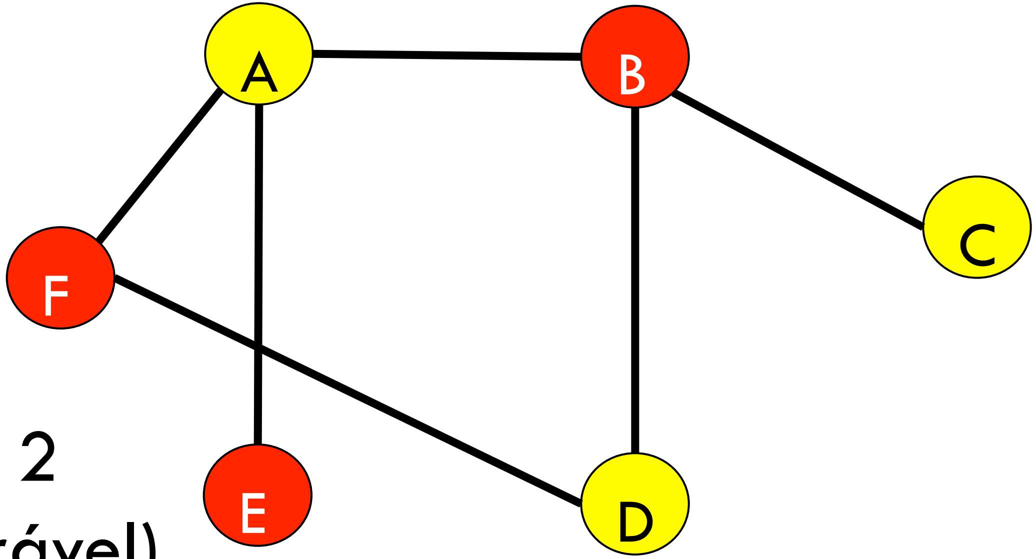
Número de cores

Exercício



$X(G) = ?$

Exercício



X(G) = 2

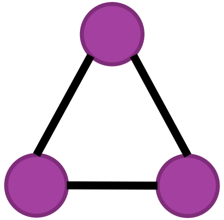
(bicolorável)

Coloração de grafos

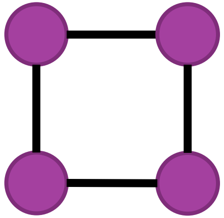
- Coloração se dá em grafos conexos simples:
  - Desconsiderar grafos desconexos. As cores utilizadas em um componente não têm efeito sobre as do outro componente.
  - Arestas paralelas não afetam a coloração.
  - Grafo não pode ter loops.

Coloração de circuitos

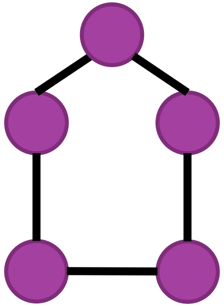
- TEOREMA: Um grafo consistindo simplesmente de um circuito com n≥3 vértices é 2-cromático se n é par e 3-cromático se n é impar



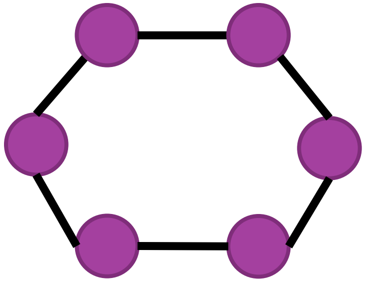
3-cromático



2-cromático



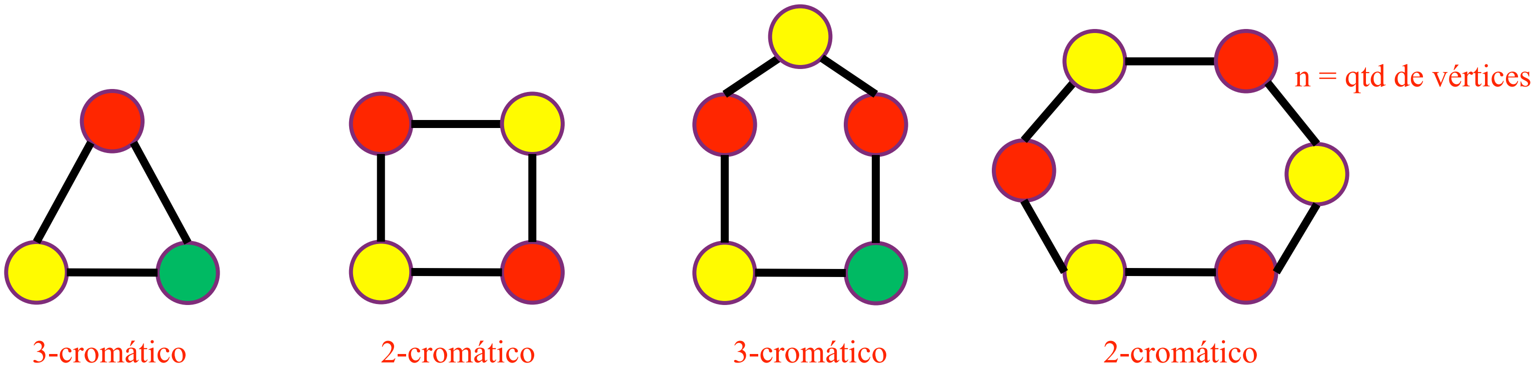
3-cromático



2-cromático

Coloração de circuitos

- TEOREMA: Um grafo consistindo simplesmente de um circuito com n≥3 vértices é 2-cromático se n é par e 3-cromático se n é impar



Coloração de circuitos

- TEOREMA: Um grafo simples G com pelo menos uma aresta é 2-cromático se, e somente se, G não contiver circuitos de tamanho ímpar

Graus e coloração

- Se d é o maior grau dos vértices de um grafo simples G, então $X(G) \leq d+1$
- Se d é o maior grau dos vértices de um grafo simples G, tal que G não contém um grafo circuito com um número ímpar de vértices e nem um grafo completo, de d+1 vértices, então $X(G) \leq d$

Coloração de grafos

- Infelizmente, o problema de determinar a coloração de um grafo é *NP-Completo*

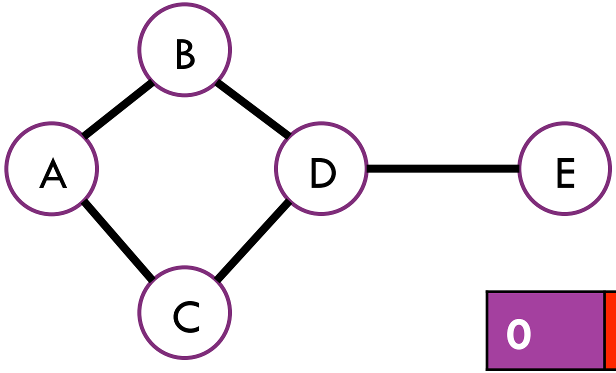
- Na prática, tenta-se resolver o problema com a utilização de *heurísticas*

Algoritmo guloso (coloração sequencial)


```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```

Algoritmo guloso (coloração sequencial)

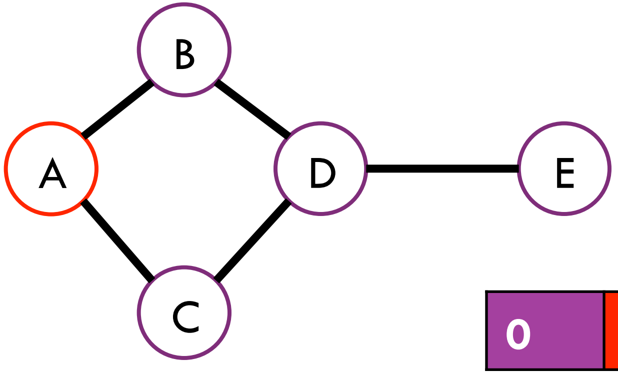
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```

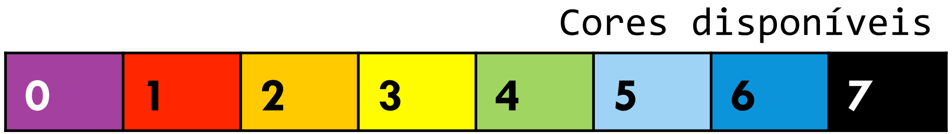


Cores disponíveis

Algoritmo guloso (coloração sequencial)

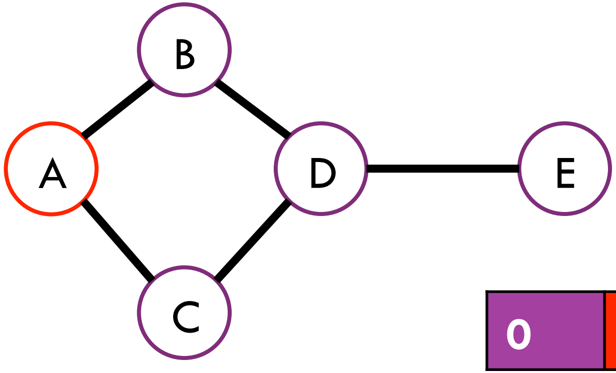
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```





Algoritmo guloso (coloração sequencial)

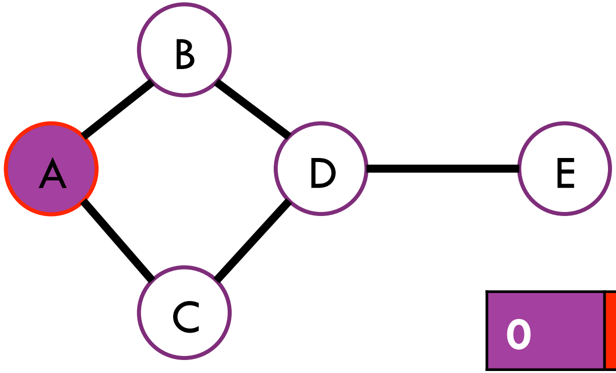
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```



Cores disponíveis

Algoritmo guloso (coloração sequencial)

```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```

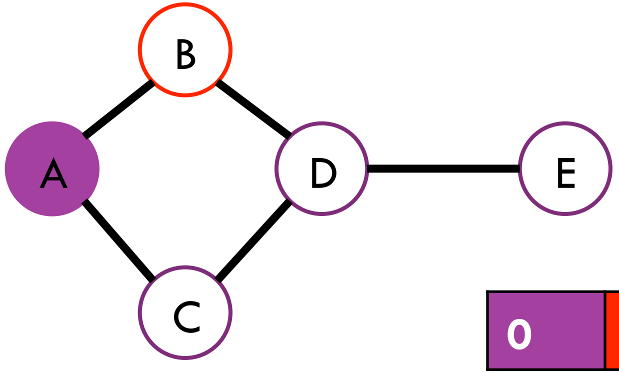


Cores disponíveis

0	1	2	3	4	5	6	7

Algoritmo guloso (coloração sequencial)

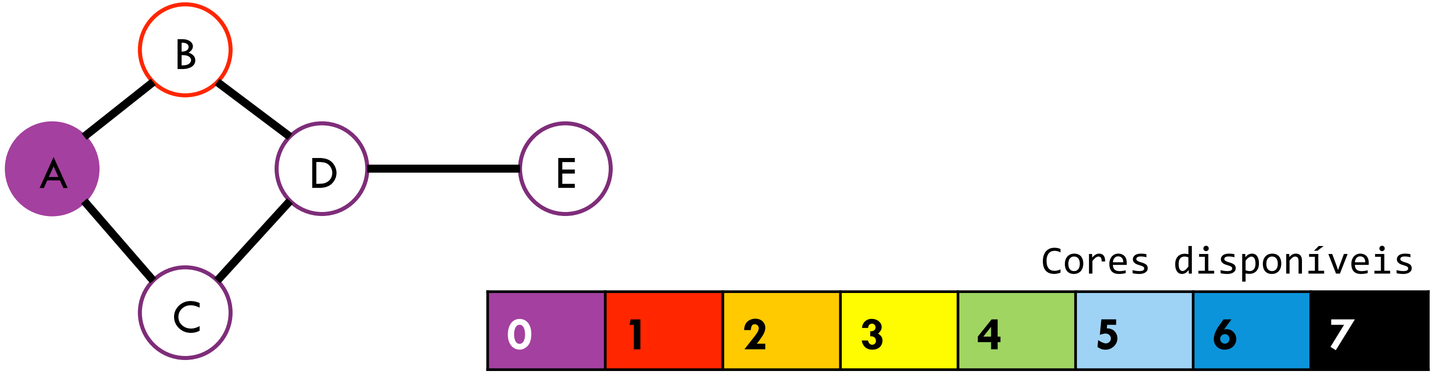
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```



Cores disponíveis

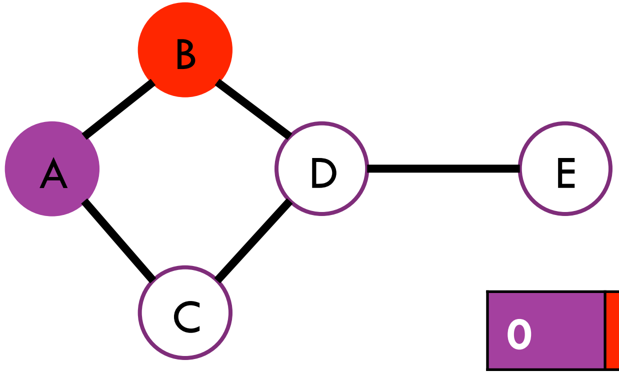
Algoritmo guloso (coloração sequencial)

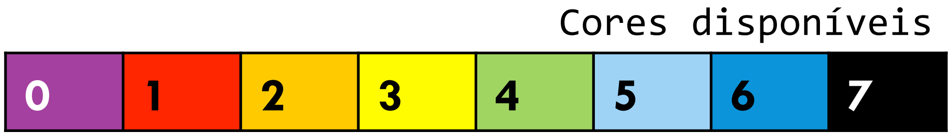
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```



Algoritmo guloso (coloração sequencial)

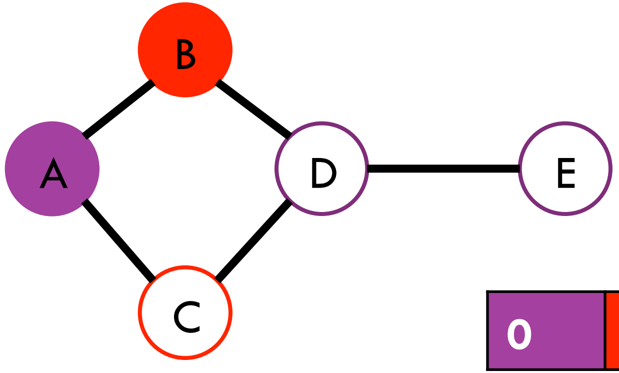
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```





Algoritmo guloso (coloração sequencial)

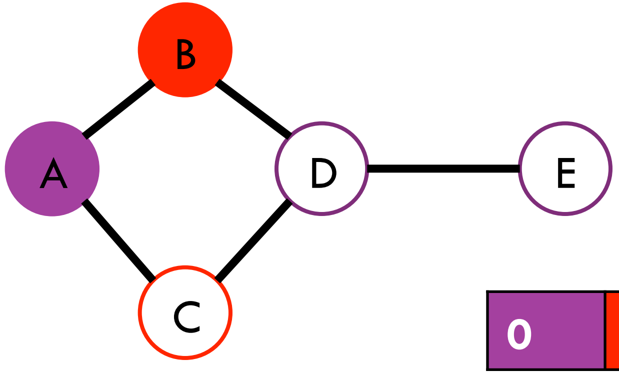
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```

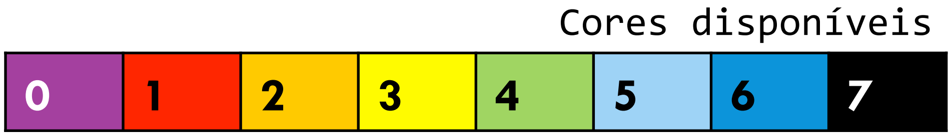


Cores disponíveis

Algoritmo guloso (coloração sequencial)

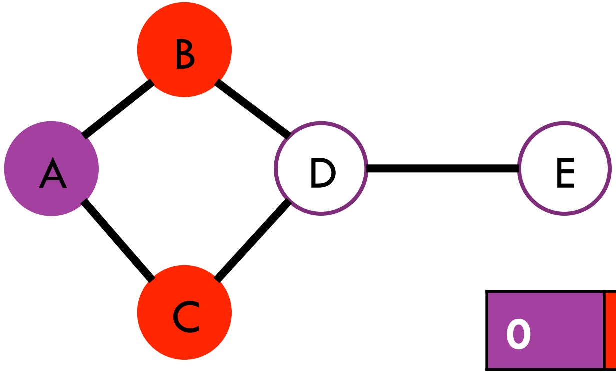
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```

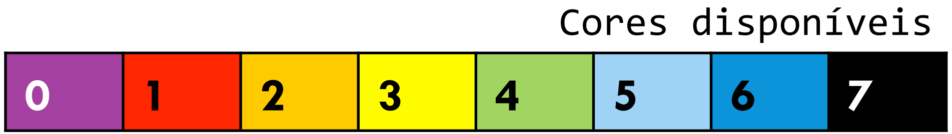




Algoritmo guloso (coloração sequencial)

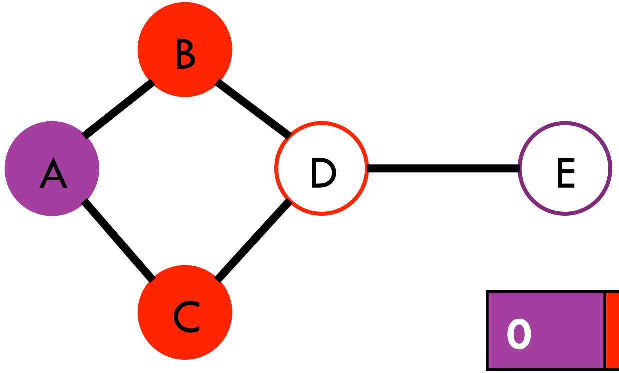
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```





Algoritmo guloso (coloração sequencial)

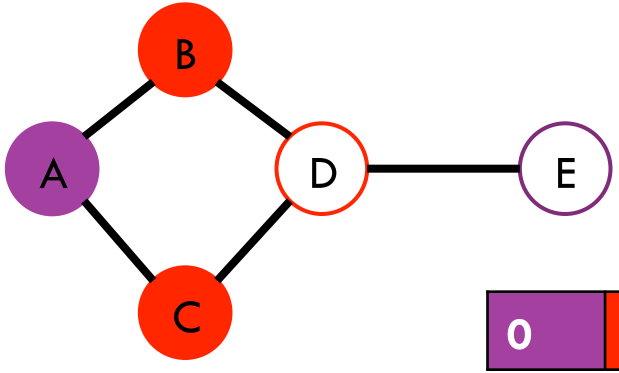
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```

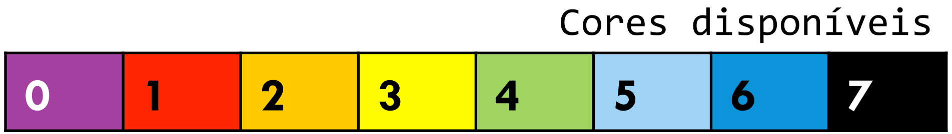


Cores disponíveis

Algoritmo guloso (coloração sequencial)

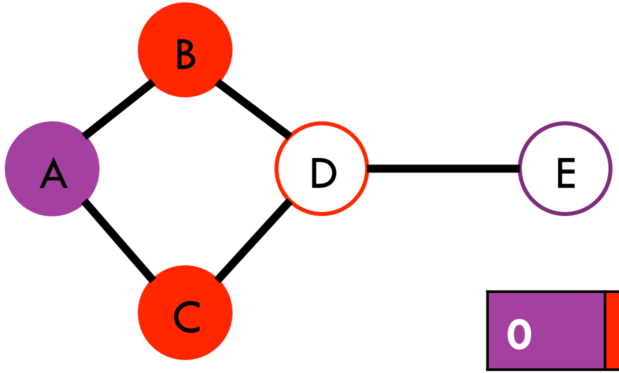
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```

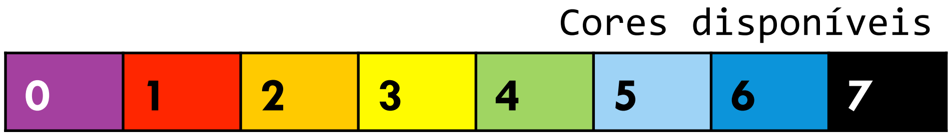




Algoritmo guloso (coloração sequencial)

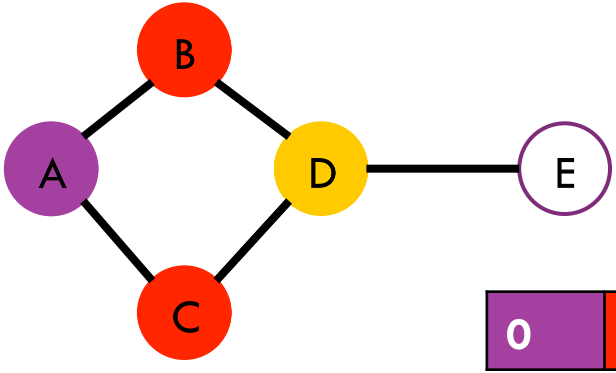
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```





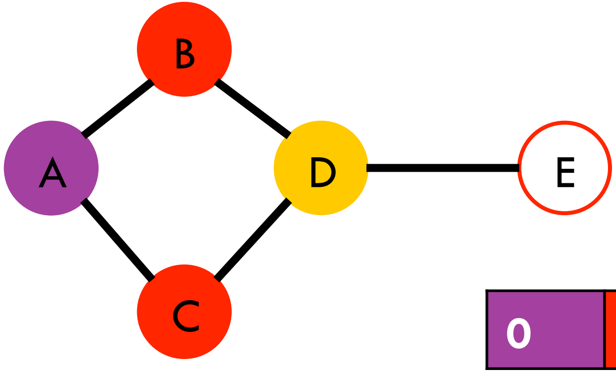
Algoritmo guloso (coloração sequencial)

```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```



Algoritmo guloso (coloração sequencial)

```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```

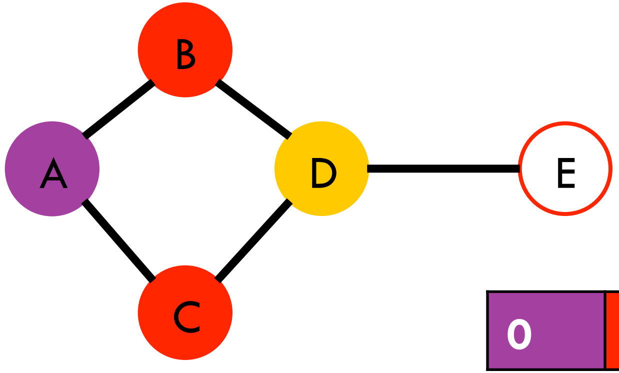


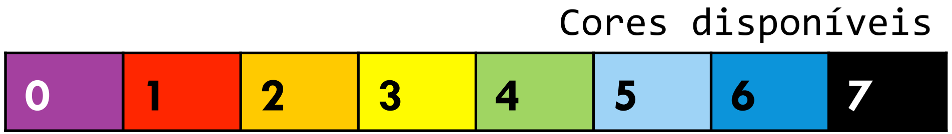
Cores disponíveis

0	1	2	3	4	5	6	7

Algoritmo guloso (coloração sequencial)

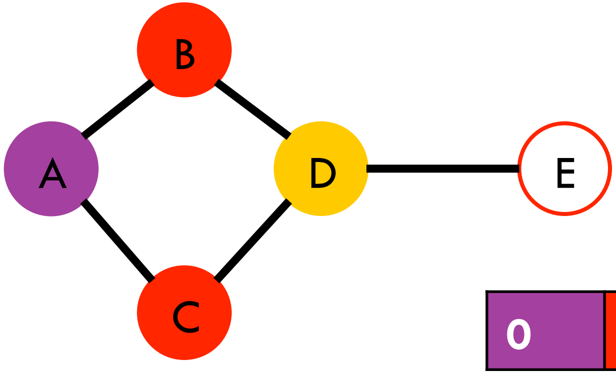
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```





Algoritmo guloso (coloração sequencial)

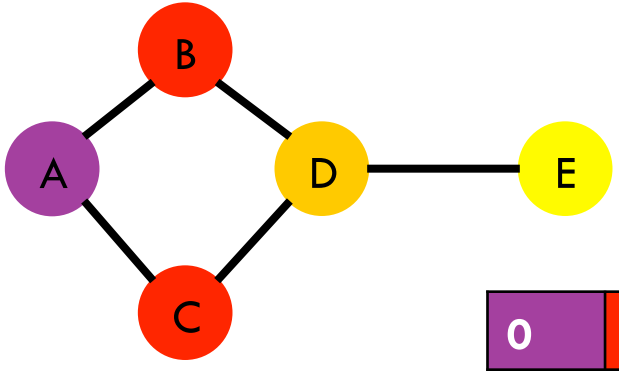
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```



Cores disponíveis

Algoritmo guloso (coloração sequencial)

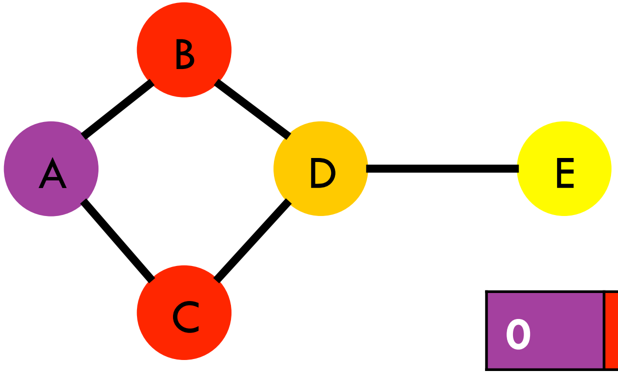
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
     então atribua cor i a v
     senão faça i=i+1 e atribua cor i a v
```



Cores disponíveis

Algoritmo guloso (coloração sequencial)

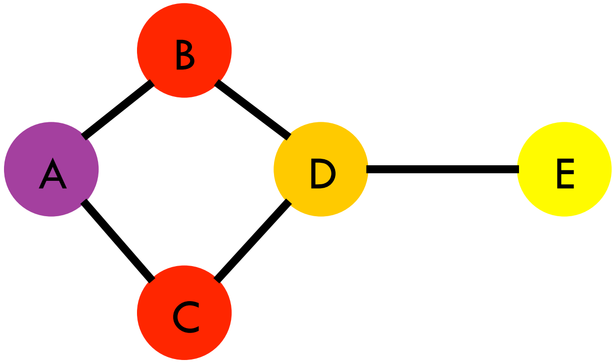
```
i=0
enquanto houver vértice incolor faça
    seja v um vértice incolor
    se uma cor i não é usada por nenhum vizinho de v
      então atribua cor i a v
      senão faça i=i+1 e atribua cor i a v
```



Cores disponíveis

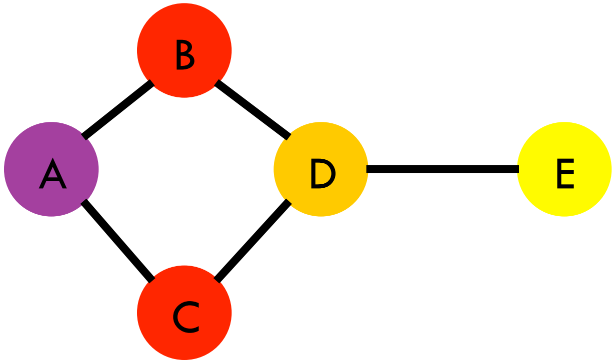
Algoritmo guloso

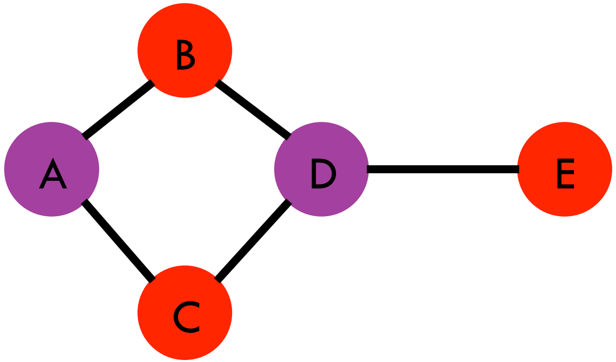
- Obtém a melhor solução?



Algoritmo guloso

- Obtém a melhor solução?



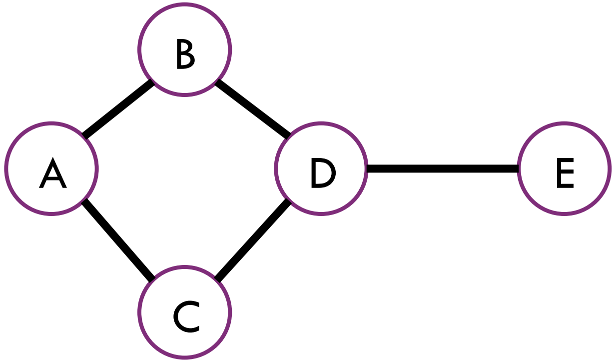


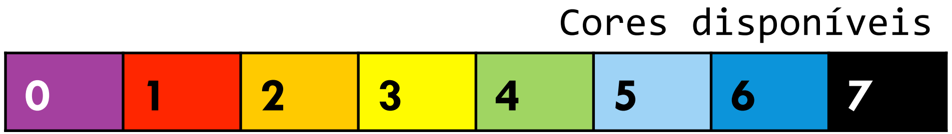
Algoritmo ingênuo (*naive*)


```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```

Algoritmo ingênuo (*naive*)

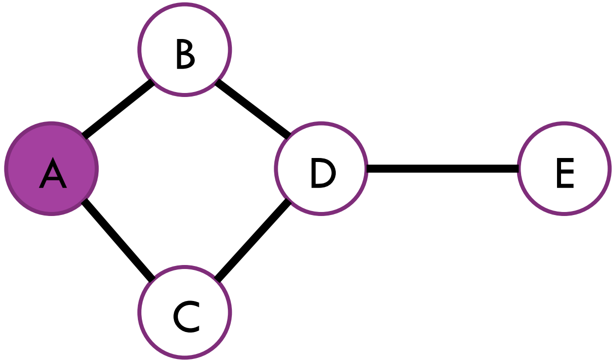
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
               atribuir cor i ao vértice v
```





Algoritmo ingênuo (*naive*)

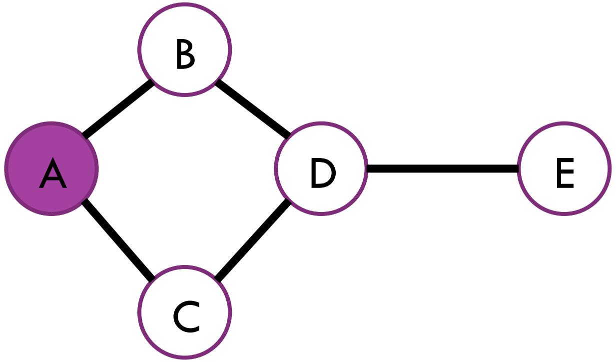
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```



Cores disponíveis

Algoritmo ingênuo (*naive*)

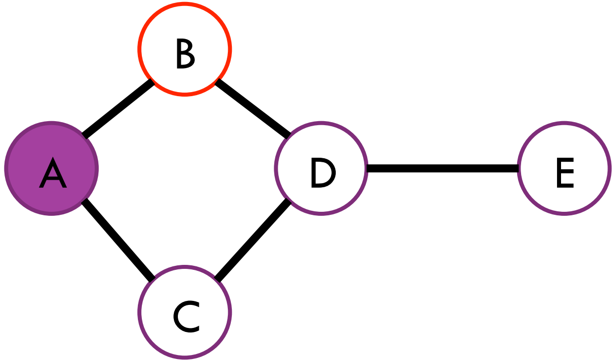
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
               atribuir cor i ao vértice v
```



Cores disponíveis

Algoritmo ingênuo (*naive*)

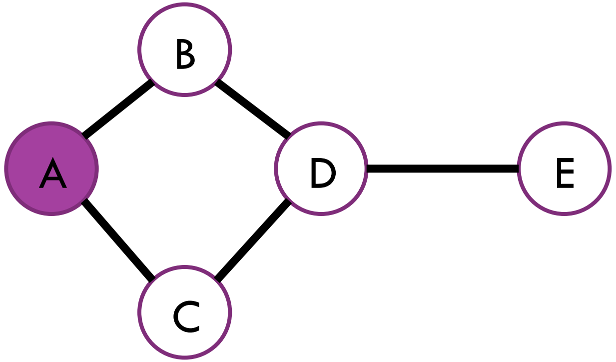
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```



Cores disponíveis

Algoritmo ingênuo (*naive*)

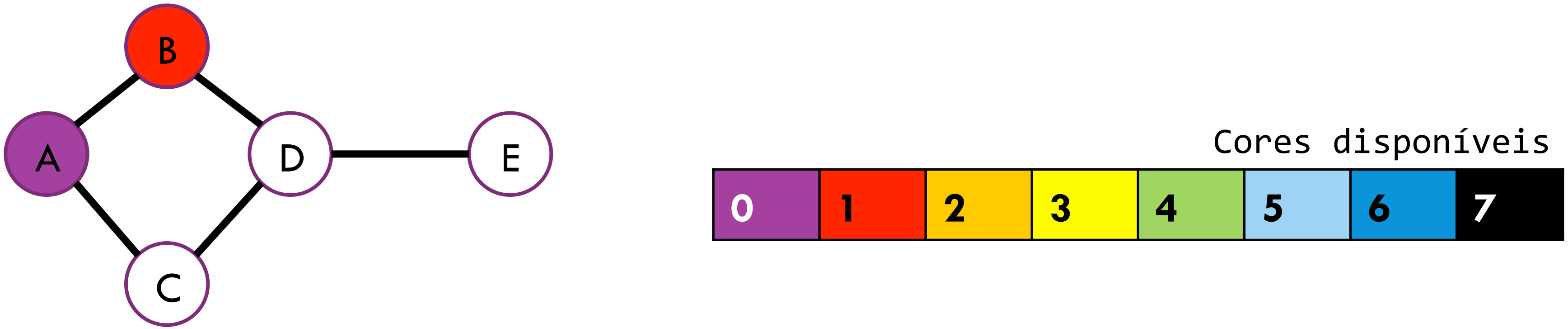
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
               atribuir cor i ao vértice v
```



Cores disponíveis

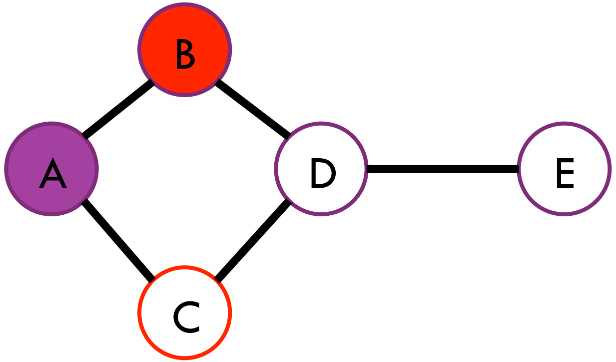
Algoritmo ingênuo (*naive*)

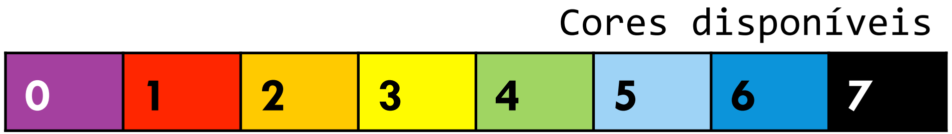
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```



Algoritmo ingênuo (*naive*)

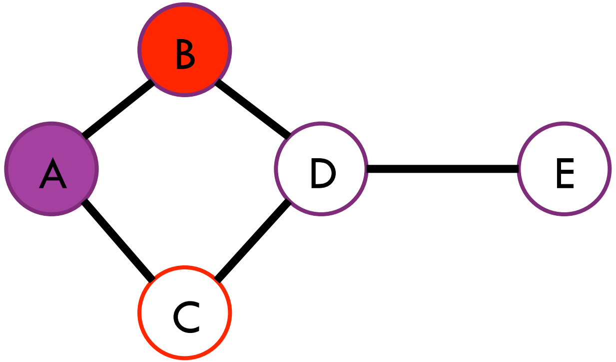
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```





Algoritmo ingênuo (*naive*)

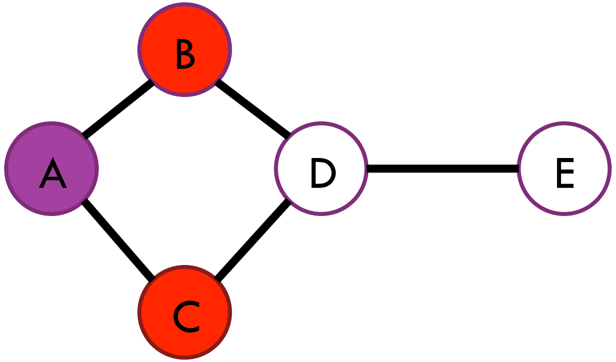
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```



Cores disponíveis

Algoritmo ingênuo (*naive*)

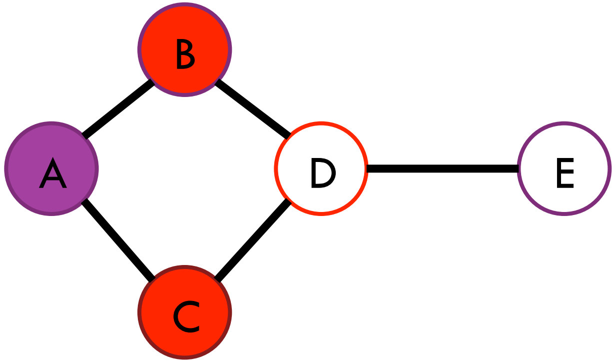
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```

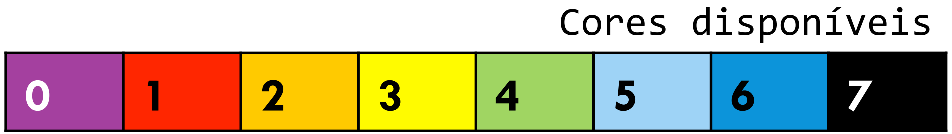


Cores disponíveis

Algoritmo ingênuo (*naive*)

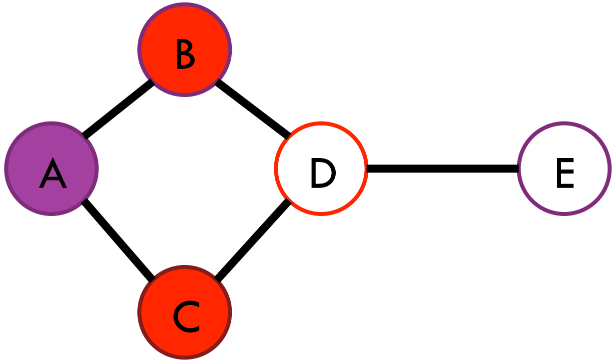
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```

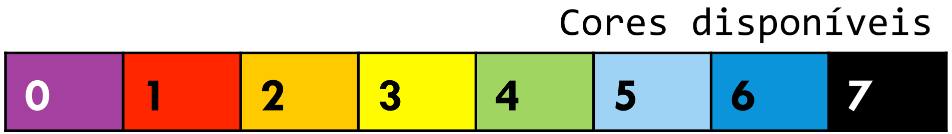




Algoritmo ingênuo (*naive*)

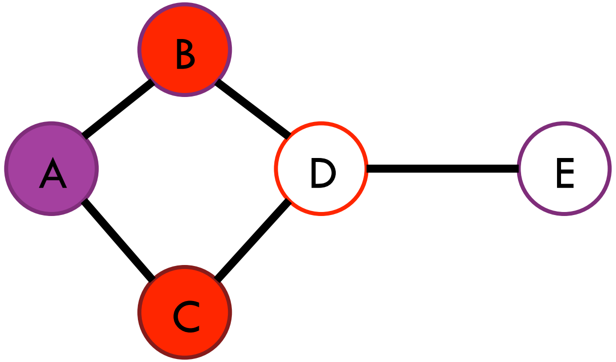
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```

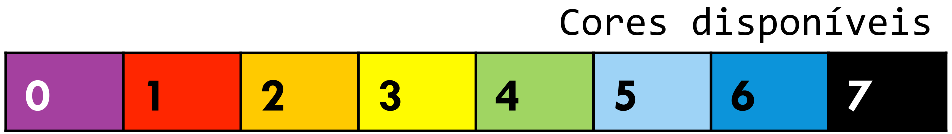




Algoritmo ingênuo (*naive*)

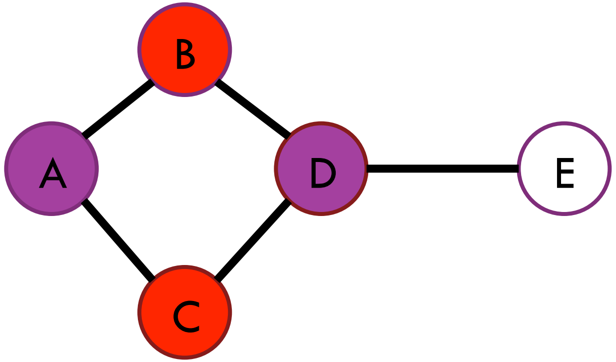
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
               atribuir cor i ao vértice v
```

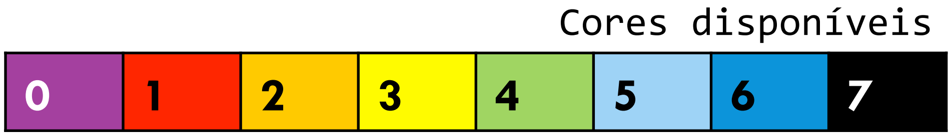




Algoritmo ingênuo (*naive*)

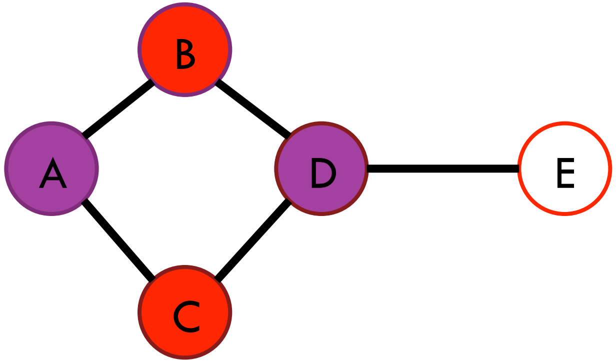
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
               atribuir cor i ao vértice v
```

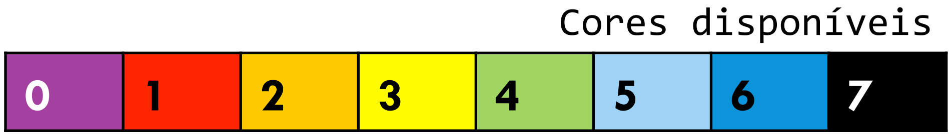




Algoritmo ingênuo (*naive*)

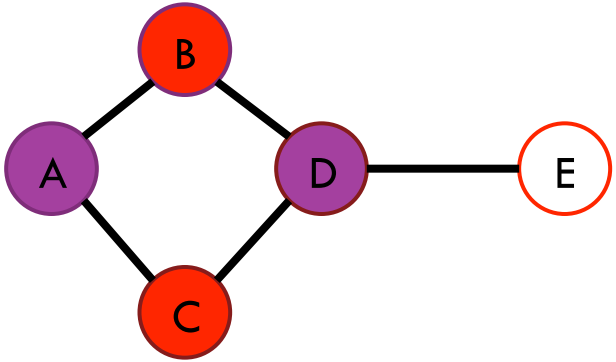
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```

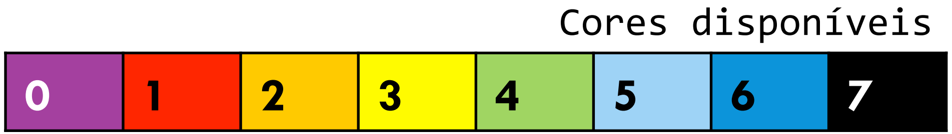




Algoritmo ingênuo (*naive*)

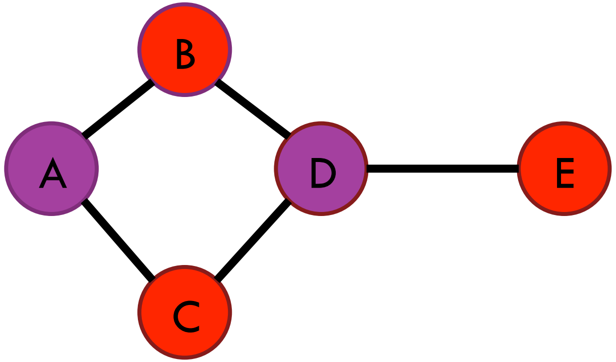
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```





Algoritmo ingênuo (*naive*)

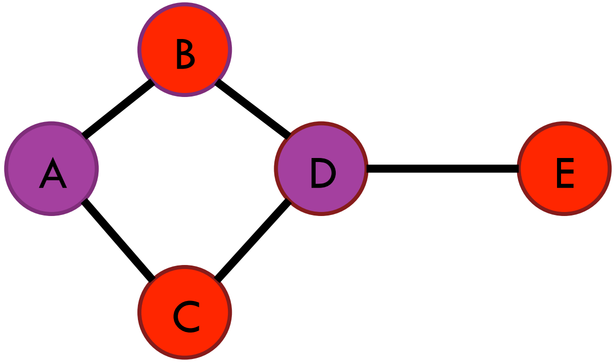
```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```



Cores disponíveis

Algoritmo ingênuo (*naive*)

```
i=0
atribuir cor i ao primeiro vértice.
percorrer sequencialmente os vértices restantes
   para cada vértice v visitado
      k = primeira cor já utilizada e que não pertence a nenhum
      dos vértices adjacentes a v
      atribuir cor k a v
      se os vértices adjacentes já usam todas as cores,
        então i = i+1
              atribuir cor i ao vértice v
```



Cores disponíveis

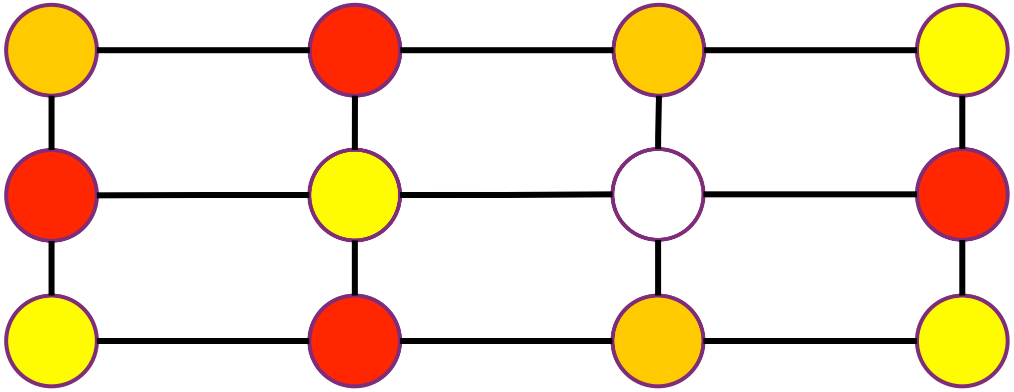
Algoritmo ingênuo (*naive*)

- Quanto maior o grau de um vértice, mais difícil sua coloração

- Algoritmo poderia percorrer vértices em ordem decrescente de grau

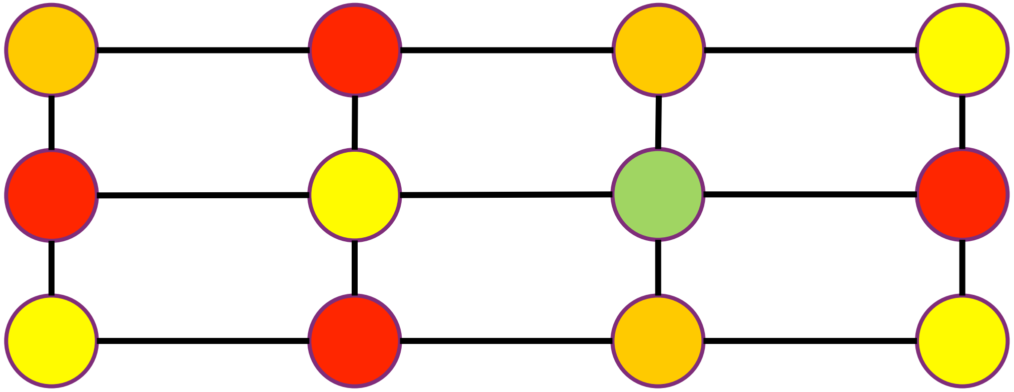
Coloração de grafos

- Muitas outras heurísticas podem ser tentadas, por exemplo, troca de cores em componentes



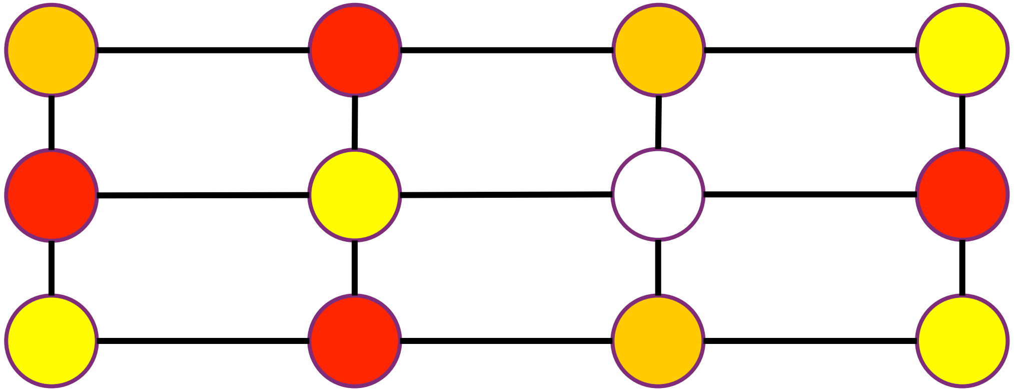
Coloração de grafos

- Muitas outras heurísticas podem ser tentadas, por exemplo, troca de cores em componentes



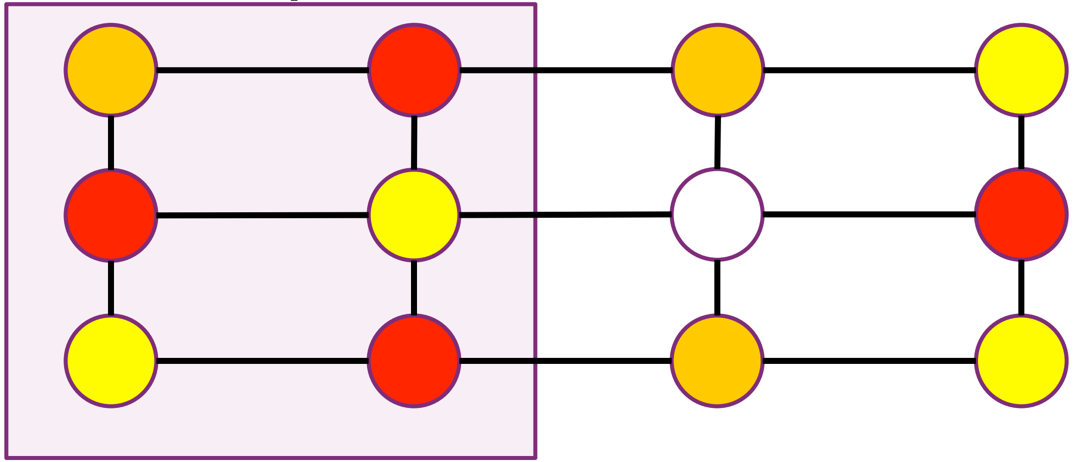
Coloração de grafos

- Muitas outras heurísticas podem ser tentadas, por exemplo, troca de cores em componentes



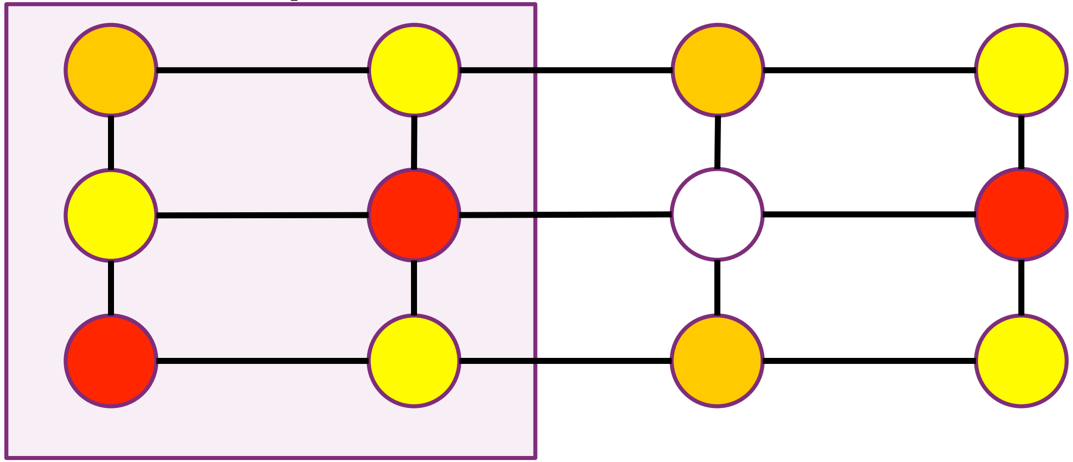
Coloração de grafos

- Muitas outras heurísticas podem ser tentadas, por exemplo, troca de cores em componentes



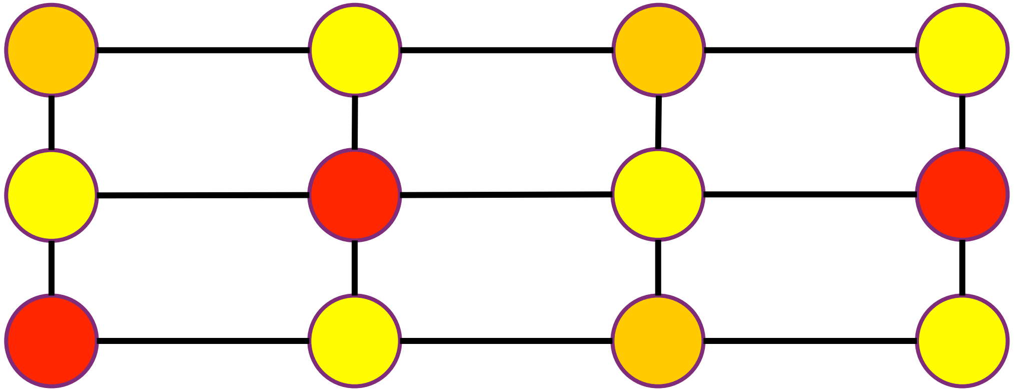
Coloração de grafos

- Muitas outras heurísticas podem ser tentadas, por exemplo, troca de cores em componentes



Coloração de grafos

- Muitas outras heurísticas podem ser tentadas, por exemplo, troca de cores em componentes



Coloração de arestas

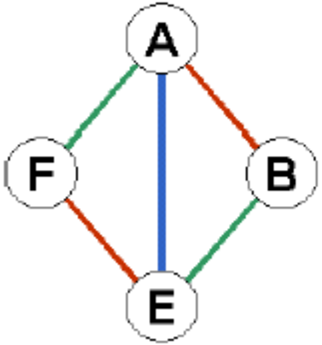
- Uma **coloração de arestas** de um grafo simples G é uma atribuição de cores às arestas de G de maneira que cores diferentes são atribuídas a arestas adjacentes.

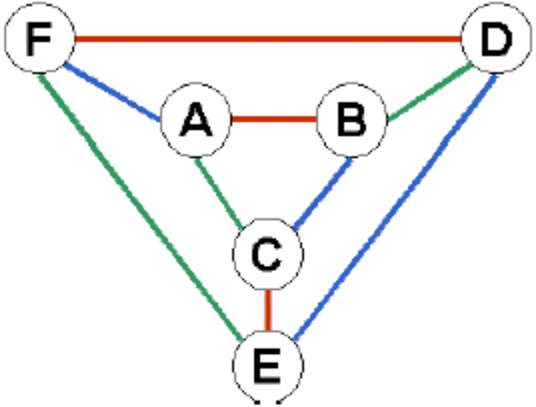
Coloração de arestas

- Se existe uma coloração de arestas para um grafo G que utiliza K cores, então, G é um grafo K-colorido de arestas.

- O **índice cromático** de um grafo G, X'(G), é o menor número K para qual G é K-colorido de arestas.

Coloração de arestas



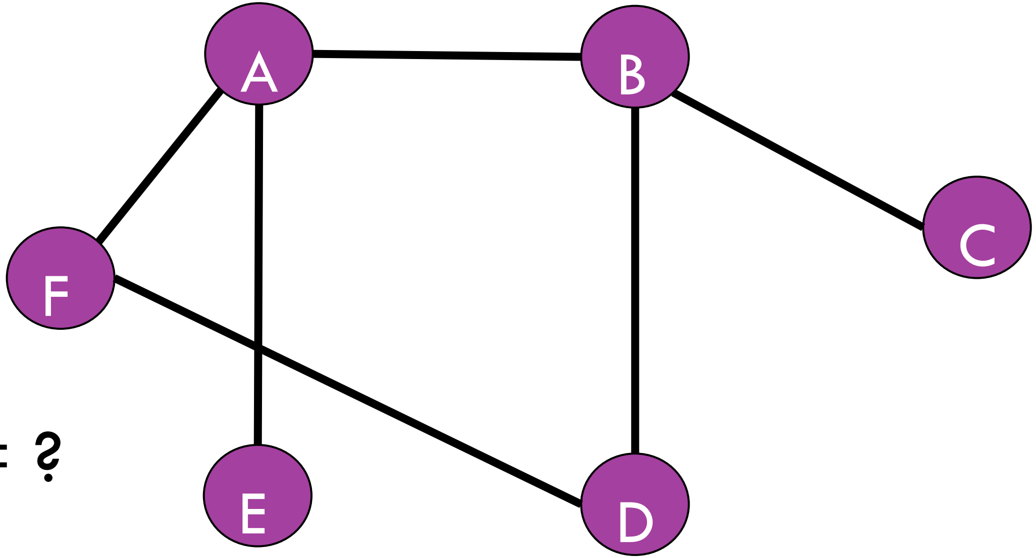


Coloração de arestas

- TEOREMA: Se G é um grafo simples cujo vértice de maior grau tem grau $\delta(G)$, então

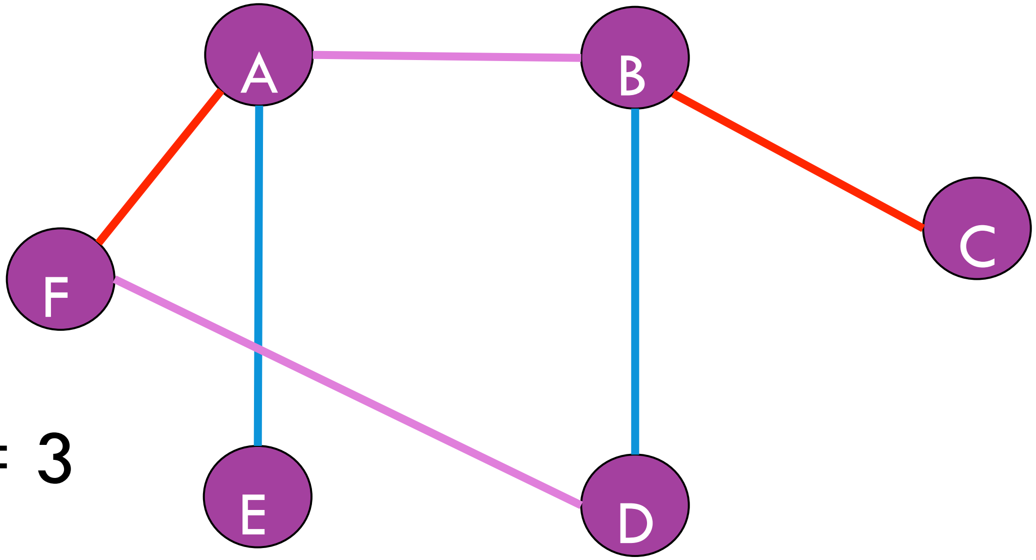
$$\delta(G) \leq X'(G) \leq \delta(G)+1$$

Exercício



$X'(G) = ?$

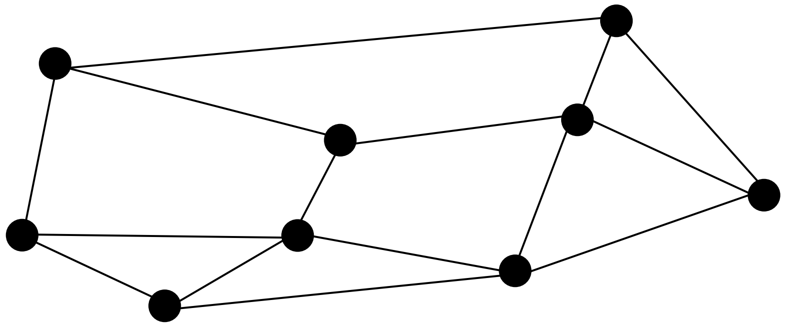
Exercício



$X'(G) = 3$

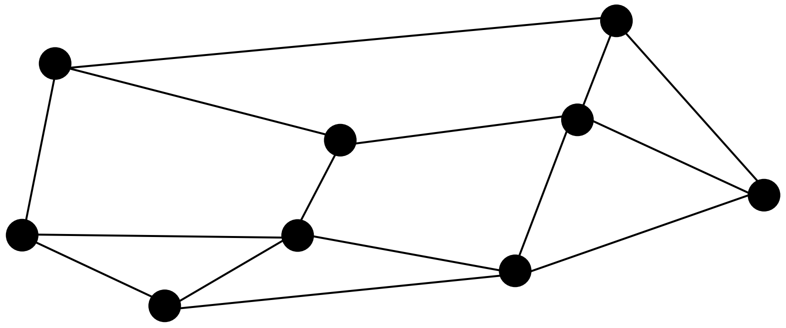
Coloração de grafos planares

- Quantas cores são necessárias para colorir um grafo planar?



Coloração de grafos planares

- Quantas cores são necessárias para colorir um grafo planar?



- Coloração de *faces* do grafo

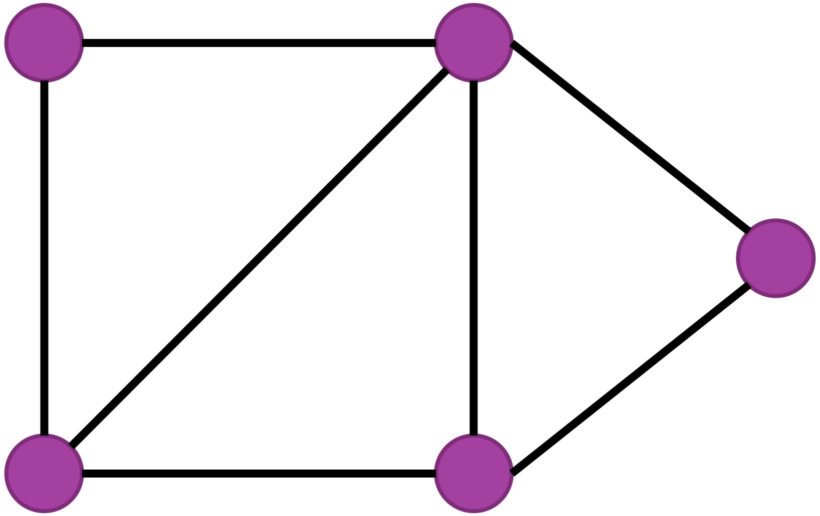
Dualidade

- Dado um grafo G planar, o grafo G*, chamado **dual** de G, é construído da seguinte forma:
  - para cada face f de G, G* tem um vértice

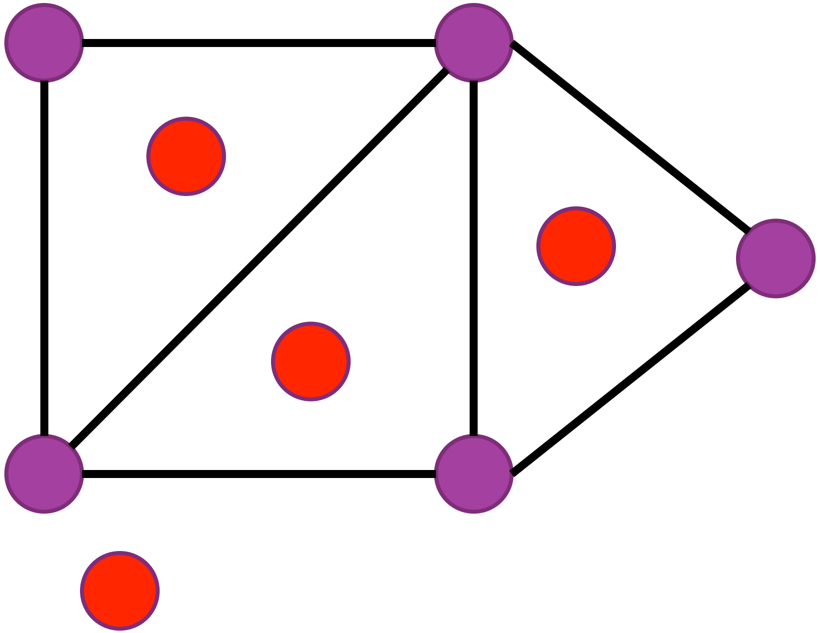
Dualidade

- una os vértices de G* da seguinte forma
  - se 2 regiões f_i e f_k são adjacentes (possuem alguma aresta em comum) coloque uma aresta entre v_k e v_k interceptando a aresta em comum
  - se existirem mais de uma aresta em comum entre f_i e f_k coloque uma aresta entre v_i e v_k para cada aresta em comum
  - se uma aresta está inteiramente em uma região, f_i, coloque um loop no vértice v_i.

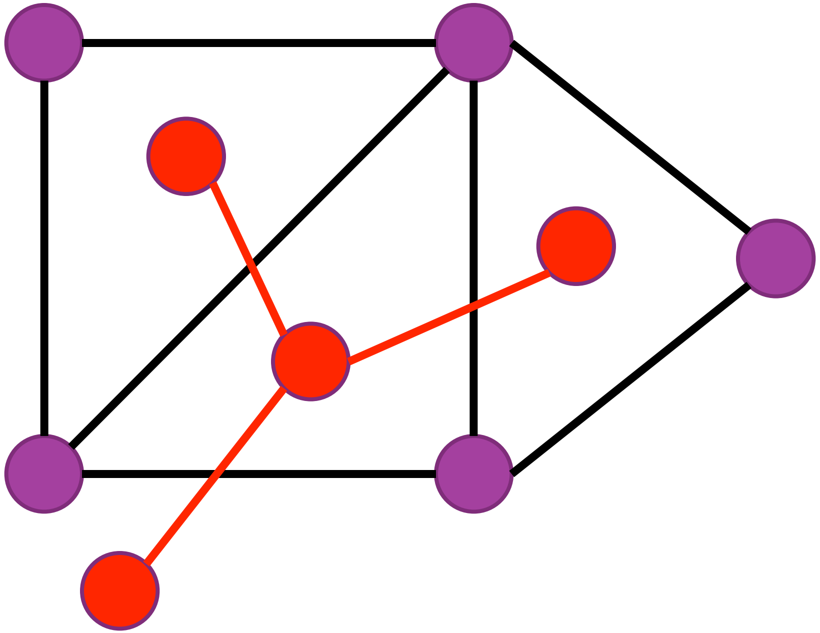
Dualidade



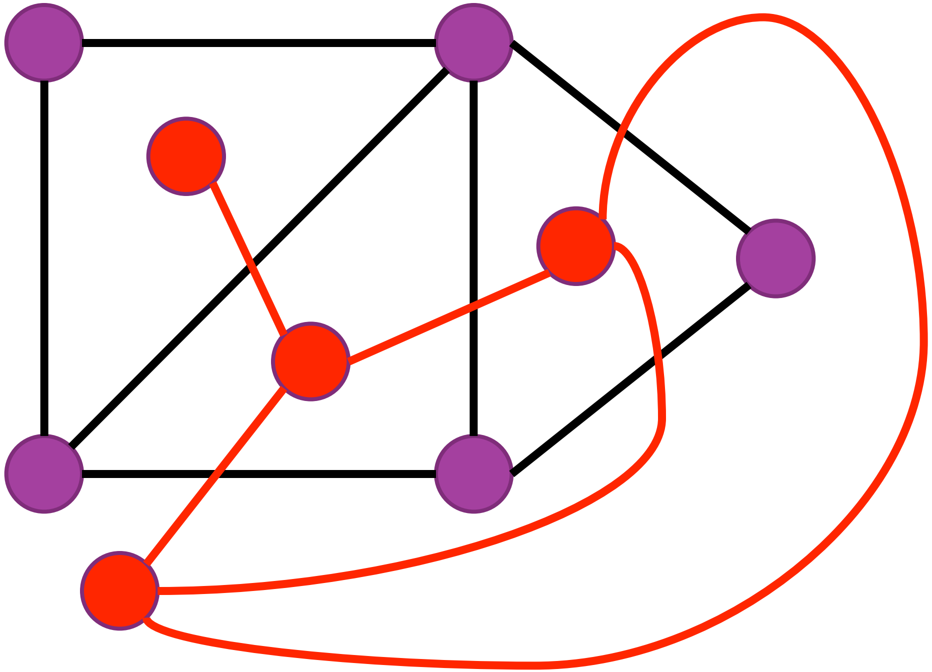
Dualidade



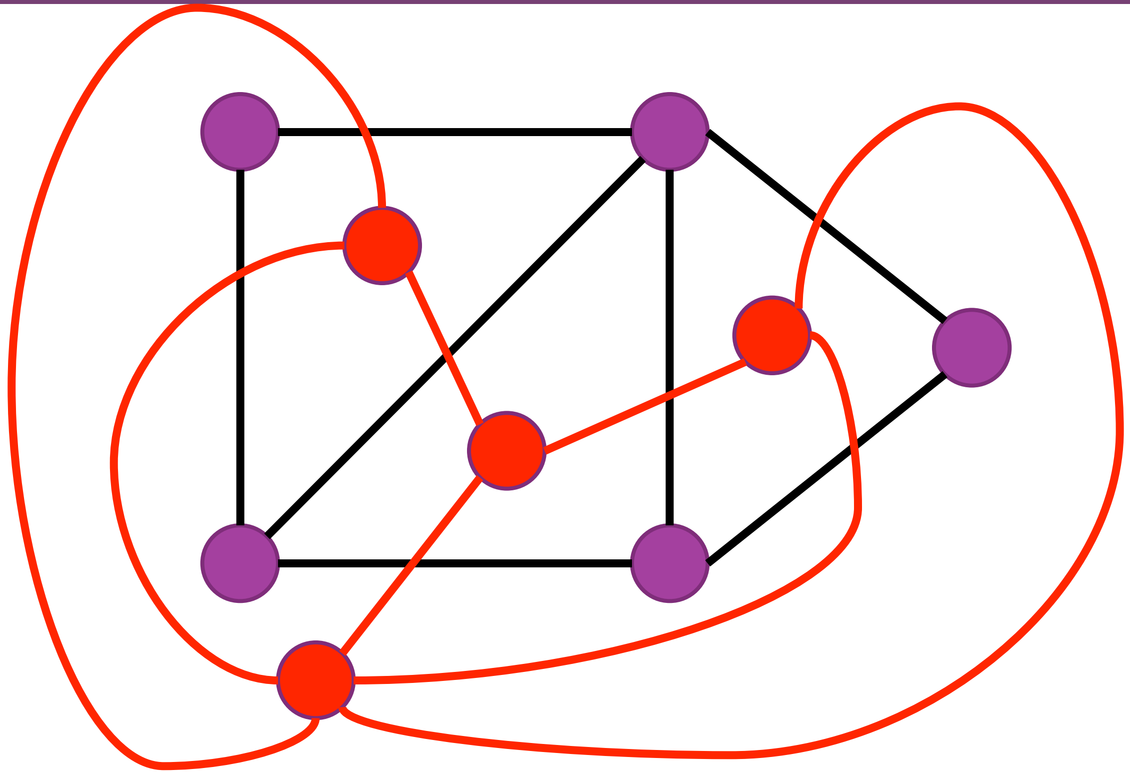
Dualidade



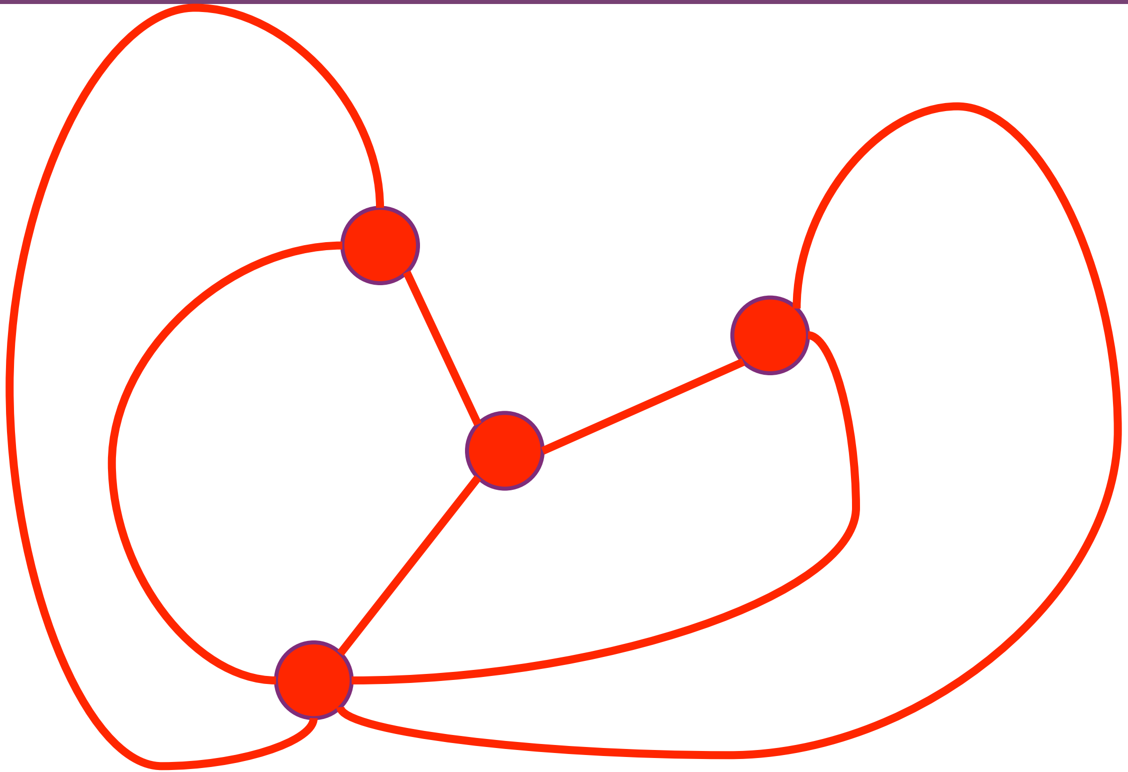
Dualidade



Dualidade

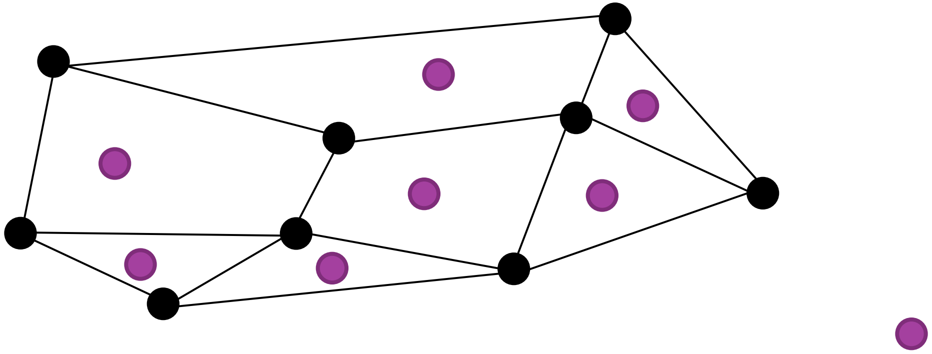


Dualidade



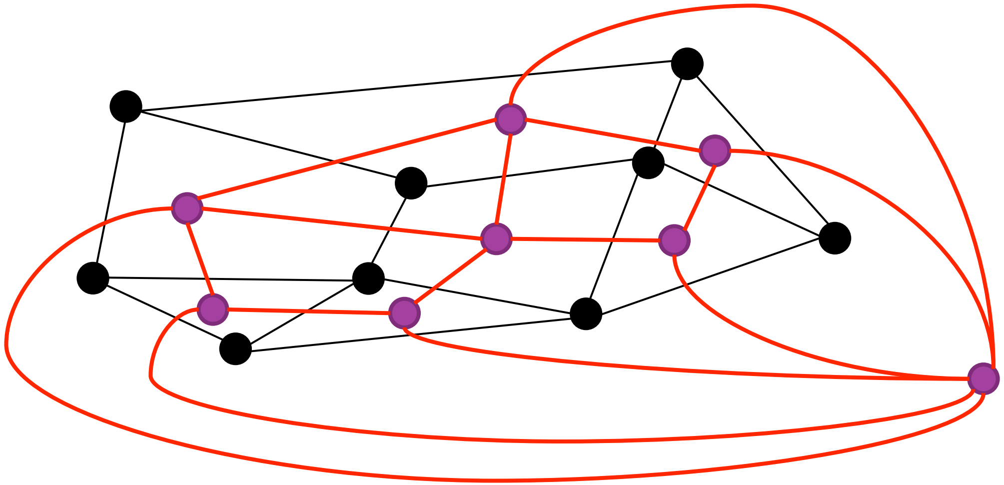
Coloração de grafos planares

- Considerando o dual de G (G*), a coloração de faces de G equivale à coloração de vértices de G*.



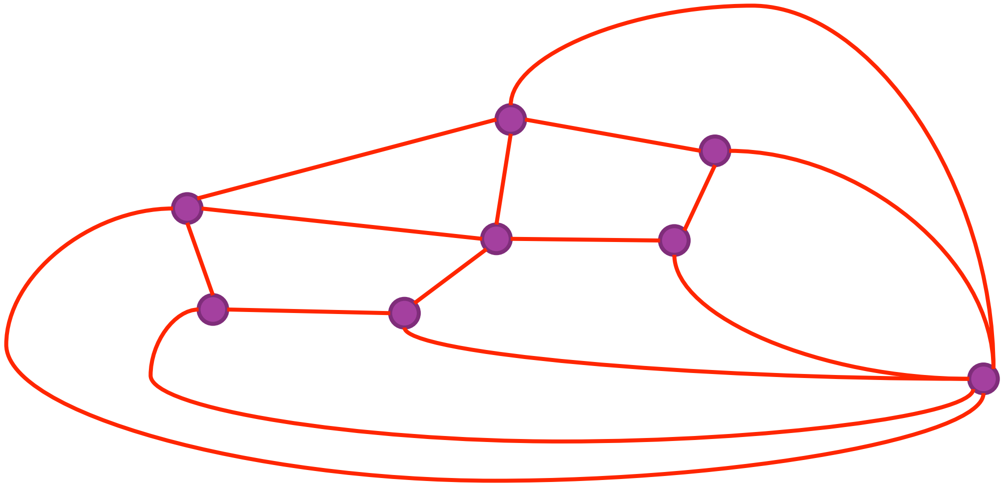
Coloração de grafos planares

- Considerando o dual de G (G*), a coloração de faces de G equivale à coloração de vértices de G*.



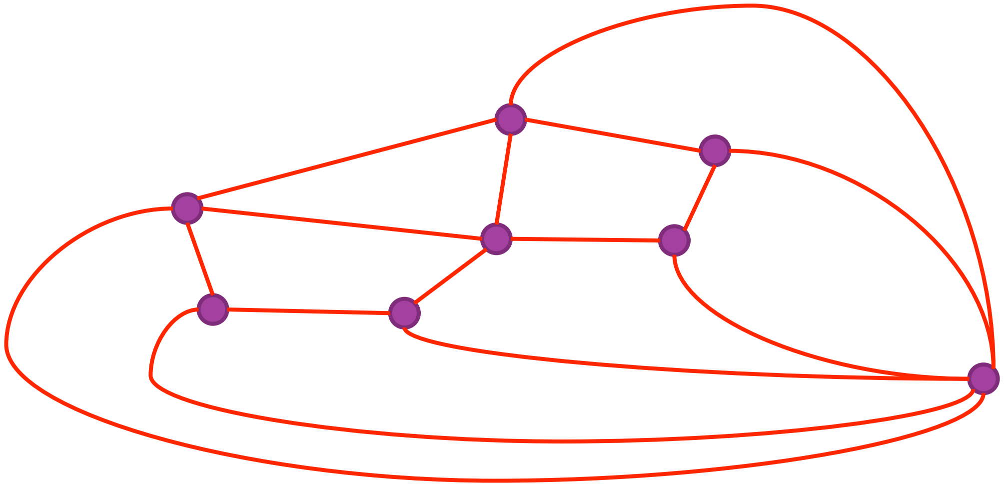
Coloração de grafos planares

- Considerando o dual de G (G*), a coloração de faces de G equivale à coloração de vértices de G*.



Coloração de grafos planares

- G é K-cromático de faces se, e somente se, G* for K-cromático de vértices

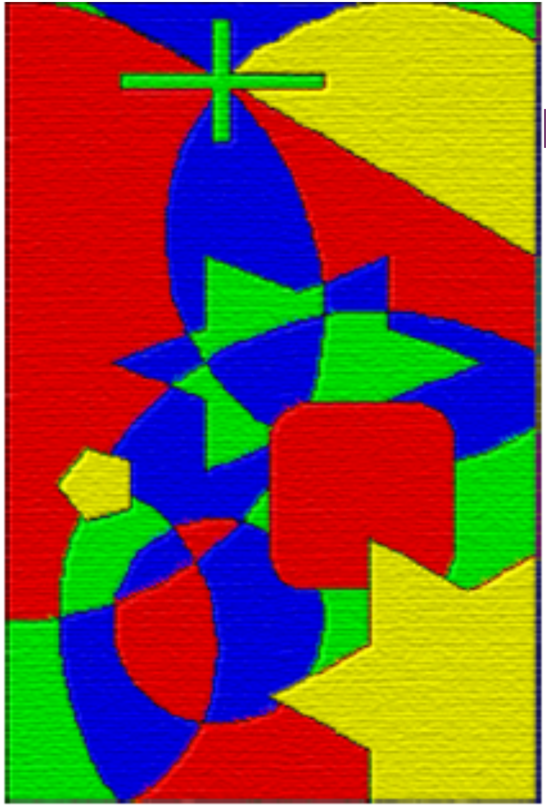


Coloração de grafos planares

- TEOREMA DAS QUATRO CORES: Dado um mapa plano, dividido em regiões, quatro cores são suficientes para colorir, de forma a que regiões vizinhas não partilhem a mesma cor. (Appel e Haken, 1976)

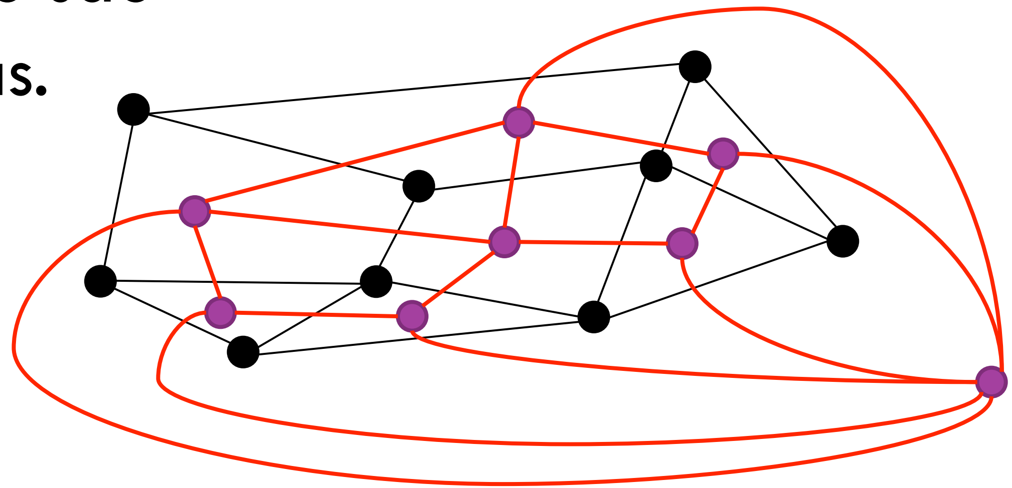
Teorema das quatro cores

- Condição: as regiões que só se tocam num ponto não são consideradas vizinhas.



Teorema das quatro cores

- Condição: as regiões que só se tocam num ponto não são consideradas vizinhas.

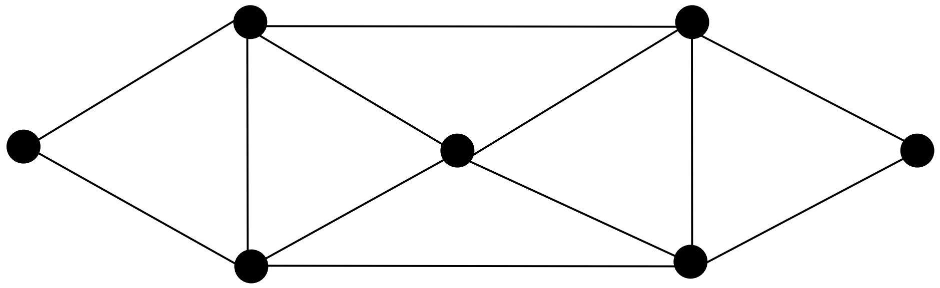


Teorema das quatro cores

- Considerando tal condição, o teorema foi demonstrado pela primeira vez em 1976 por Appel e Haken, utilizando um computador IBM 360.
- Em 1994 foi produzida uma prova simplificada por Raul Seymour, Neil Roberton, Daniel Sanders e Robin Thomas, mas continua a ser impossível demonstrar o teorema sem recorrer a um computador.

Coloração de grafos planares

- TEOREMA: Um grafo planar G pode ter as faces coloridas com 2 cores se, e somente se, G for Euleriano.



Coloração de grafos planares

- TEOREMA: Um grafo planar G pode ter as faces coloridas com 2 cores se, e somente se, G for Euleriano.