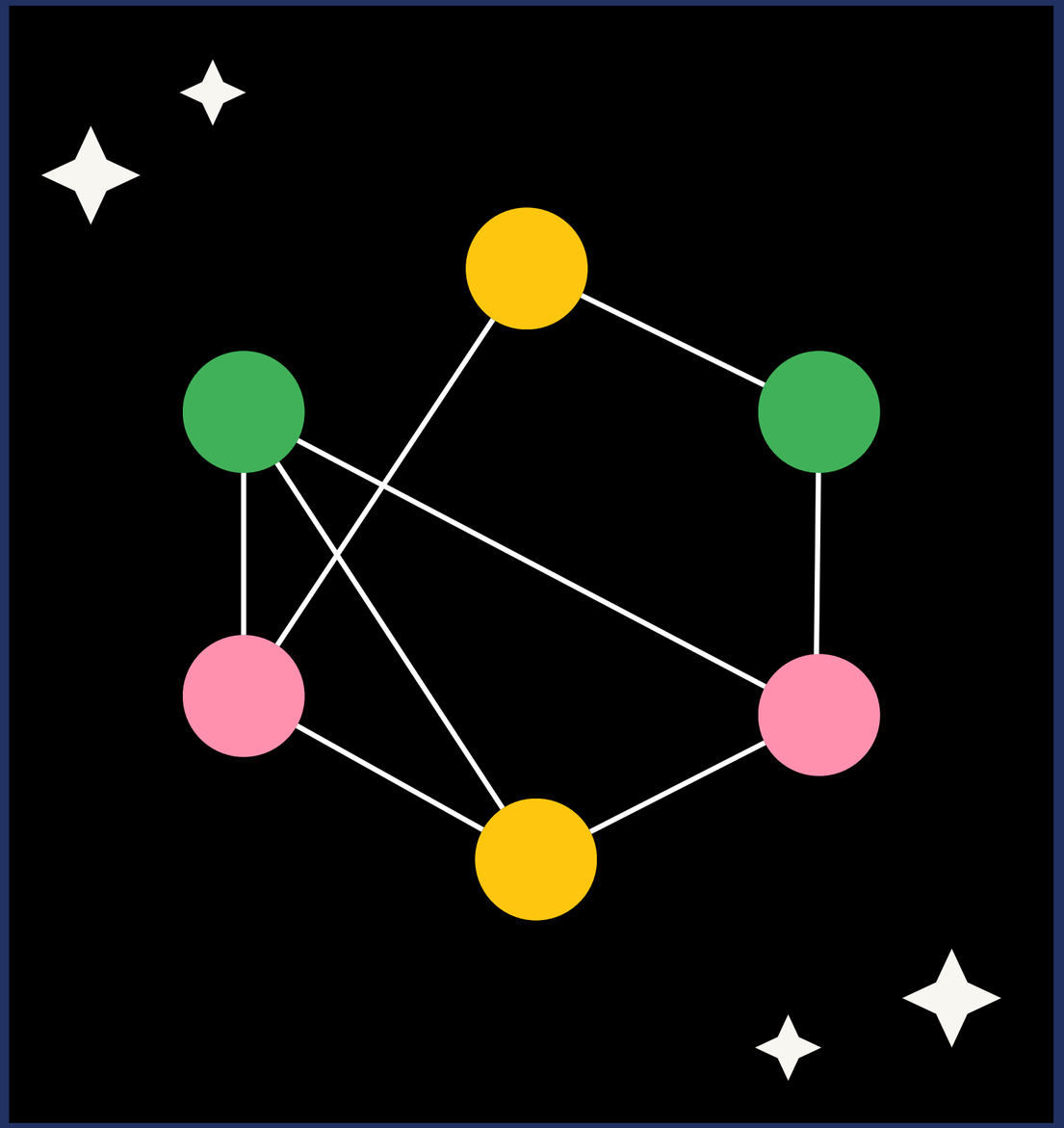
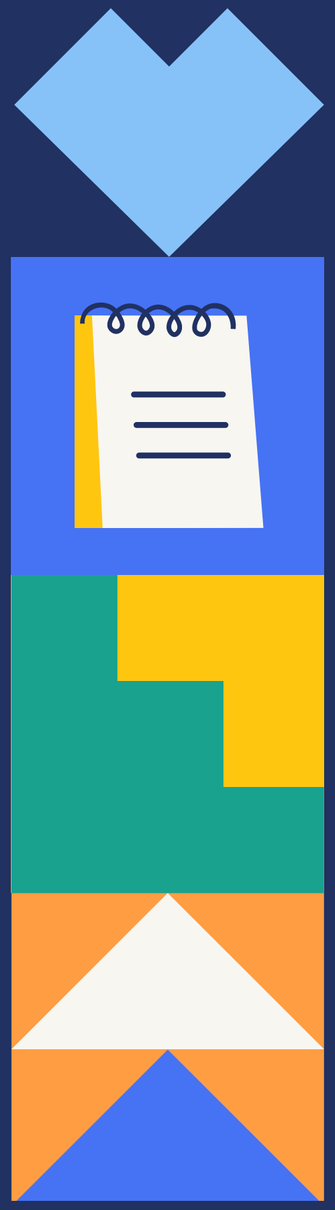
Universidade Federal de Lavras - UFLA
Complexidade e Projetos de Algoritmos

Coloração de Grafos (GC)

Gabriela Memento
Giovanna Trindade
Marcelo Rodrigues
Tiago Lage





Conteúdo

- **Definição, explicação do problema e exemplos**
  Tiago Lage
- **Algoritmo backtracking exato para o problema**
  Gabriela Memento
- **Heurística para o problema**
  Giovanna Trindade
- **Demonstração da execução de cada algoritmo**
  Marcelo Rodrigues
- **Análise experimental dos algoritmos**
  Marcelo Rodrigues

Definição

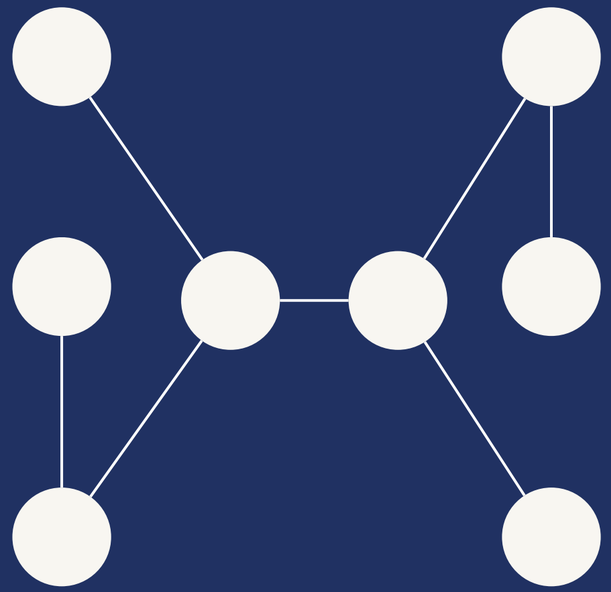
A definição formal para o **Problema de Coloração de Grafos** é:

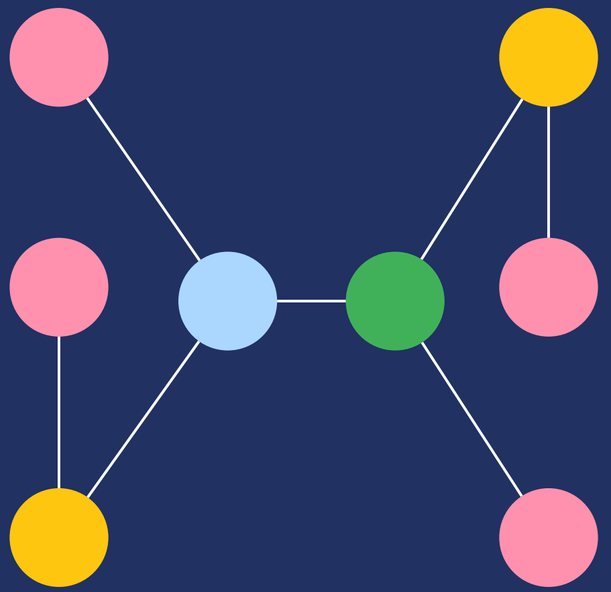
Dado um grafo não-direcionado G = (V, E), com um conjunto V de vértices e um conjunto E de arestas que incidem sobre os vértices de G, o problema de coloração de grafos consiste em atribubir k-cores para os vértices de G, de modo que dois vértices adjacentes não tenham a mesma cor.

O problema pode ser formulado de duas formas. Primeiro como problema de decisão, em que o objetivo é identificar se um grafo é k-colorível. Na segunda formulação o objetivo é encontrar um valor de k que seja mínimo.

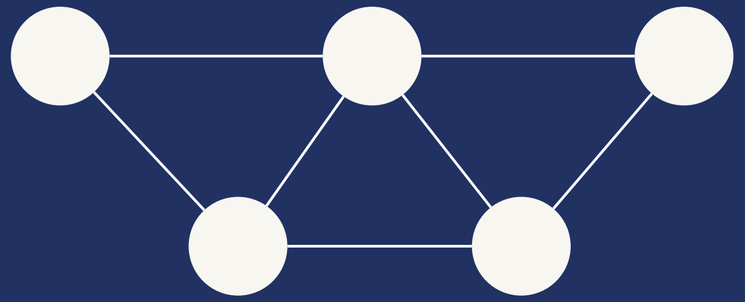
Exemplos

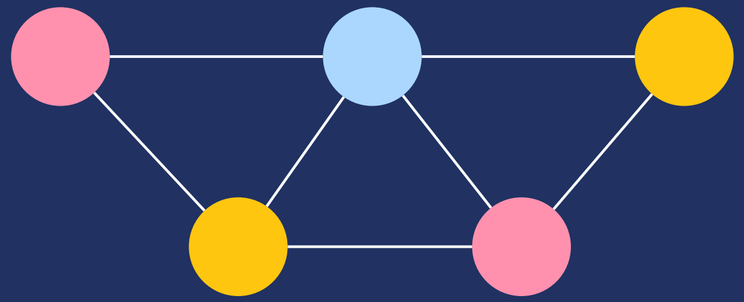
Exemplos





Exemplos





Problema das 4 cores

Suponha a tarefa de colorir os países da América Sul de tal maneira que quaisquer dois países com uma fronteira em comum tenham que ser coloridos com cores distintas. Quantas cores, no mínimo, seriam necessárias para execução desta tarefa? A despeito de saber-se que o número mínimo de cores necessário para este fim é quatro, pode-se afirmar que isto é verdadeiro para qualquer mapa?

Problema das 4 cores

Embora possa parecer somente uma curiosidade, este problema despertou por muito tempo o interesse e a atenção de muitos pesquisadores, desta forma, as tentativas e iniciativas em solucioná-lo contribuíram de modo significativo para o desenvolvimento de uma área da matemática conhecida como Teoria dos Grafos.

O problema das quatro cores é um problema de decisão que busca determinar se é possível colorir um mapa com quatro cores ou menos, de modo que países vizinhos tenham cores diferentes. Proposto em 1852 por Francis Guthrie, o problema foi resolvido apenas em 1976 por Kenneth Appel e Wolfgang Haken, utilizando uma prova por computador. Eles demonstraram que qualquer mapa pode ser colorido com no máximo quatro cores, criando cerca de 1.700 instâncias do problema e verificando todas com sucesso. Esse resultado é conhecido como o Teorema das Quatro Cores.

Algoritmo Exato: Backtracking

O algoritmo de **backtracking** é uma técnica usada para encontrar soluções, explorando todas as possibilidades e voltando atrás quando encontra uma solução inválida. No contexto da coloração de grafos, o algoritmo tenta colorir o grafo vértice por vértice, retrocedendo sempre que uma atribuição de cor não pode ser validada.

Para encontrar a quantidade mínima de cores necessárias, o algoritmo começa testando com um número mínimo de cores, incrementando gradualmente até encontrar a quantidade mínima que permita uma coloração válida.

Funcionamento (pt. 1)

1 - Escolha do número de cores: O algoritmo começa testando com k = 1 e incrementa k até encontrar a menor quantidade de cores necessárias.

2 - Escolha do vértice: Para cada número de cores, o algoritmo seleciona um vértice do grafo para colorir. Normalmente, os vértices são escolhidos de forma sequencial.

3 - Tentativa de colorir: Para cada vértice, tenta-se atribuir uma cor disponível que ainda não tenha sido usada nos vértices adjacentes.

4 - Validação: Após atribuir uma cor a um vértice, verifica-se se a coloração é válida (ou seja, se nenhum vértice adjacente tem a mesma cor).

Funcionamento (pt. 2)

5 - Recursão e retrocesso: Se a coloração for válida, o algoritmo continua para o próximo vértice. Se a coloração não for válida ou se o algoritmo atinge um ponto onde não há cores disponíveis que respeitem as restrições, o algoritmo retrocede (remove a cor do vértice atual e tenta a próxima cor).

6 - Incremento de cores: Se o algoritmo não conseguir colorir o grafo com o número atual de cores, incrementa o número de cores e tenta novamente até encontrar a coloração válida.

7 - Conclusão: O algoritmo termina quando todos os vértices estão coloridos de forma válida com o menor número de cores possível ou quando todas as possibilidades são exploradas sem sucesso.

Algoritmo Exato **Backtracking**

Pseudocódigo do backtracking
Seja G um grafo com n vértices e m arestas.

1. Inicialização:
 - colors[v] = -1 para todos os vértices v (nenhuma cor atribuída)
 - k = 1 (iniciar com 1 cor)

2. Tente colorir o grafo com k cores:
 a. Para cada vértice v de G:
 - Tente atribuir a menor cor i do conjunto {0, 1, 2, …, k-1} a v, que não esteja sendo usada por nenhum vizinho de v.
 - Se uma cor i for atribuída com sucesso:
 - colors[v] = i
 - Se nenhuma cor puder ser atribuída, retroceda (backtrack) e tente outra cor para o vértice anterior.

 b. Se todos os vértices forem coloridos com sucesso:
 - Terminar (solução encontrada com k cores).
 c. Caso contrário:
 - Incrementar k (k = k + 1) e repetir o passo 2.

3. Continue até que todos os vértices estejam coloridos com sucesso.

Heurística para o problema

O algoritmo DSATUR (Degree of Saturation) é uma abordagem heurística para a coloração de grafos que visa encontrar uma solução aproximada de maneira eficiente. A heurística DSATUR combina a estratégia de escolher o vértice com o maior grau de saturação e o menor grau de cor para realizar a coloração. A saturação de um vértice é o número de cores diferentes usadas em seus vértices adjacentes.

O objetivo do algoritmo é minimizar a quantidade de cores usadas e encontrar uma coloração que seja, na maioria dos casos, próxima da solução ótima. O algoritmo é eficiente para grafos grandes e complexos e é amplamente utilizado devido à sua simplicidade e bom desempenho na prática.

Funcionamento (pt. 1)

1 - Inicialização: Inicialmente, todos os vértices são descoloridos, e a saturação é calculada com base no número de cores diferentes presentes nos vizinhos de cada vértice.

2 - Escolha do vértice: Em cada passo, o algoritmo seleciona o vértice com a maior saturação. Em caso de empate, o vértice com o maior grau (número de arestas) é escolhido.

3 - Atribuição de cor: Para o vértice escolhido, tenta-se atribuir a menor cor possível que ainda não tenha sido usada pelos seus vizinhos.

Funcionamento (pt. 2)

4 - Atualização da saturação: Após a atribuição de uma cor, atualiza-se a saturação dos vértices adjacentes.

5 - Repetição: O processo é repetido até que todos os vértices estejam coloridos.

6 - Conclusão: O algoritmo termina quando todos os vértices são coloridos e retorna a coloração encontrada.

Heurística

Pseudocódigo da heurística
Seja G um grafo com n vértices e m arestas.
Assuma que usaremos os rótulos de cores 0, 1, 2, …, n-1.

1. Inicialização:
   - colors[v] = -1 para todos os vértices v (nenhuma cor atribuída)
   - saturation[v] = 0 para todos os vértices v (saturação inicial)
   - degrees[v] = grau do vértice v no grafo
   - uncoloured = conjunto de todos os vértices em G

2. Enquanto existirem vértices não coloridos:
   a. Escolha o vértice v não colorido com o maior grau de saturação.
      - Em caso de empate, selecione o vértice com o maior grau no subgrafo induzido pelos vértices não coloridos.
      - Empates adicionais podem ser resolvidos arbitrariamente.

   b. Determine a menor cor i do conjunto {0, 1, 2, …, n-1} que não seja usada por nenhum vizinho de v.

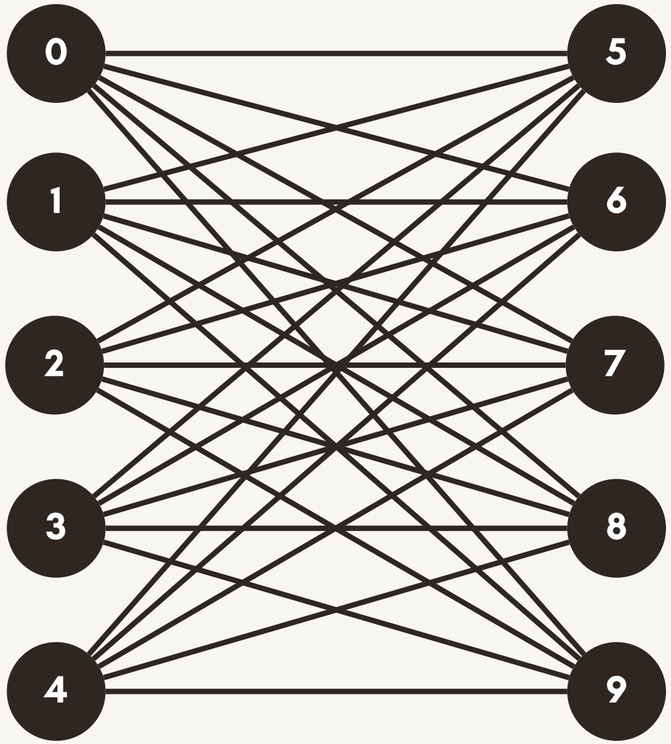
   c. Atribua a cor i ao vértice v.

   d. Atualize o grau de saturação de todos os vizinhos não coloridos de v:
      - Para cada vizinho não colorido u de v:
         - saturation[u] = número de cores diferentes atribuídas aos vizinhos de u

   e. Remova v do conjunto de vértices não coloridos.

3. Finalize quando todos os vértices estiverem coloridos.

Demonstração da execução de cada algoritmo



tests > in.txt

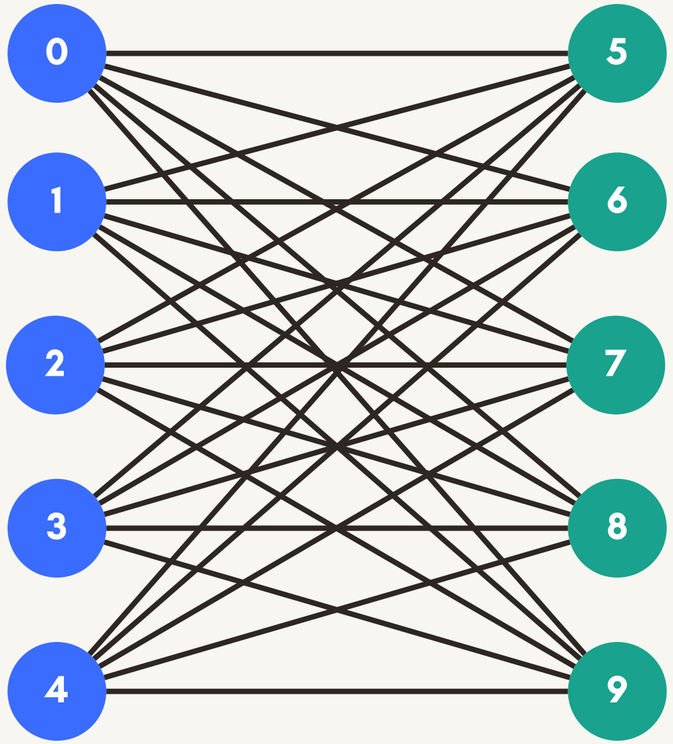
```
10
0 5
0 6
0 7
0 8
0 9
1 5
1 6
1 7
1 8
1 9
2 5
2 6
2 7
2 8
2 9
3 5
3 6
3 7
3 8
3 9
4 5
4 6
4 7
4 8
4 9
```

Demonstração da execução de cada algoritmo: Backtracking

tests > in.txt

```
10
0 5
0 6
0 7
0 8
0 9
1 5
1 6
1 7
1 8
1 9
2 5
2 6
2 7
2 8
2 9
3 5
3 6
3 7
3 8
3 9
4 5
4 6
4 7
4 8
4 9
```

```
./backtracking "tests/in.txt"
Vertex 0-> Color 0
Vertex 1-> Color 0
Vertex 2-> Color 0
Vertex 3-> Color 0
Vertex 4-> Color 0
Vertex 5-> Color 1
Vertex 6-> Color 1
Vertex 7-> Color 1
Vertex 8-> Color 1
Vertex 9-> Color 1
k colors: 2
Time: 0 ms
```

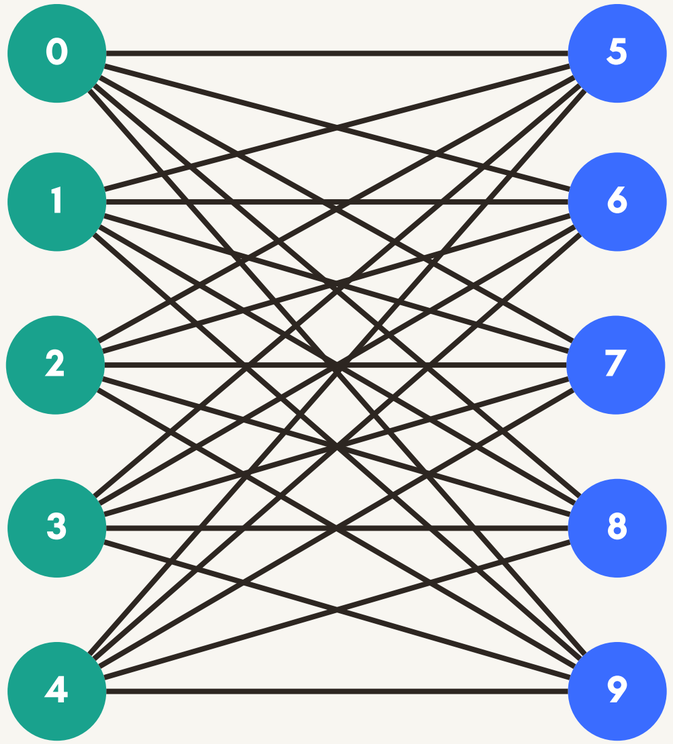


Demonstração da execução de cada algoritmo: Heurística

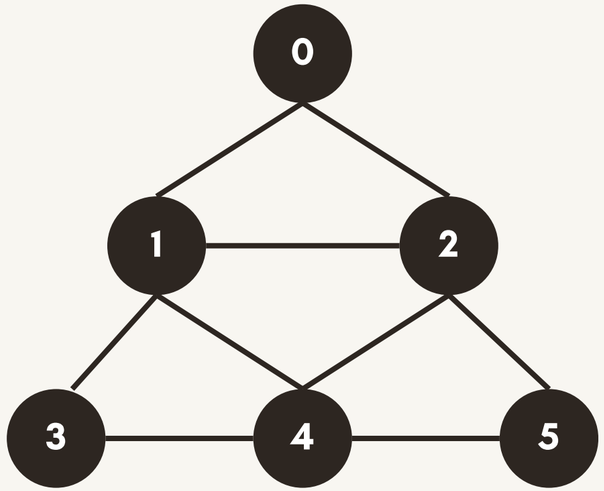
tests > in.txt

```
10
0 5
0 6
0 7
0 8
0 9
1 5
1 6
1 7
1 8
1 9
2 5
2 6
2 7
2 8
2 9
3 5
3 6
3 7
3 8
3 9
4 5
4 6
4 7
4 8
4 9
```

```
./dsatur "tests/in.txt"
Vertex 0-> Color 1
Vertex 1-> Color 1
Vertex 2-> Color 1
Vertex 3-> Color 1
Vertex 4-> Color 1
Vertex 5-> Color 0
Vertex 6-> Color 0
Vertex 7-> Color 0
Vertex 8-> Color 0
Vertex 9-> Color 0
k colors: 2
Time: 0 ms
```



Demonstração da execução de cada algoritmo

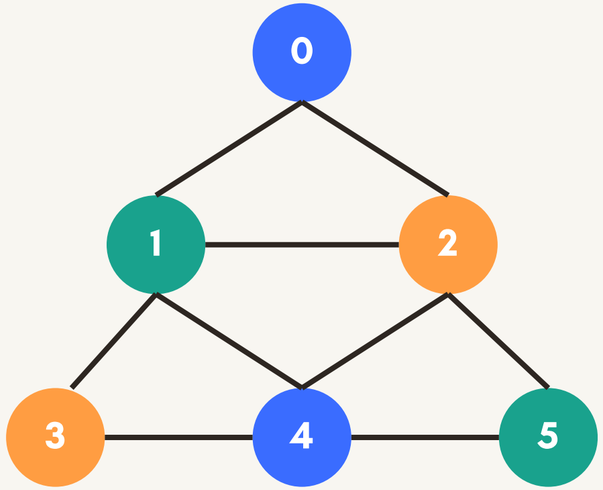



```
no elements
    1   6
    2   0 1
    3   0 2
    4   1 2
    5   1 3
    6   1 4
    7   2 4
    8   2 5
    9   3 4
   10   4 5
```

Demonstração da execução de cada algoritmo: Backtracking

```
no elements
1   6
2   0 1
3   0 2
4   1 2
5   1 3
6   1 4
7   2 4
8   2 5
9   3 4
10  4 5
```

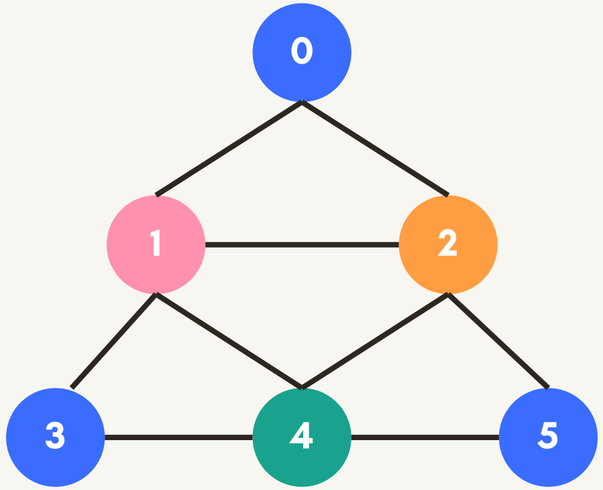
```
./backtracking tests/subop
Vertex 0-> Color 0
Vertex 1-> Color 1
Vertex 2-> Color 2
Vertex 3-> Color 2
Vertex 4-> Color 0
Vertex 5-> Color 1
k colors: 3
Time: 5 ms
```



Demonstração da execução de cada algoritmo: Heurística

```
no elements
    6
    0 1
    0 2
    1 2
    1 3
    1 4
    2 4
    2 5
    3 4
    4 5
```

```
./dsatur tests/suboptimal
Vertex 0-> Color 0
Vertex 1-> Color 3
Vertex 2-> Color 2
Vertex 3-> Color 0
Vertex 4-> Color 1
Vertex 5-> Color 0
k colors: 4
Time: 0 ms
```



Análise experimental dos algoritmos

Id	V	E	T(Sa)	C(Sa)	T(S*)	C(S*)	Q
P10	16	32	1 ms	4	1 ms	4	1
P11	18	48	1 ms	5	2 ms	5	1
P17	37	161	1 ms	7	1 ms	7	1
P06	16	38	1 ms	5	2 ms	4	1,25
P07	24	92	1 ms	6	1 ms	5	1,2

Análise experimental dos algoritmos

Id	V	E	T(Sa)	C(Sa)	T(S*)	C(S*)	Q
P09	25	100	1 ms	8	1 ms	5	1,6
P15	34	136	1 ms	7	264 ms	6	1,16
P13	34	160	1 ms	7	698 ms	6	1,16
P23	44	204	1 ms	8	13,9 s	7	1,14
P41	13	78	1 ms	13	21 min	13	1

Análise experimental dos algoritmos

A qualidade da solução da heurística em relação à ótima depende da organização da fila de prioridade no DSATUR; se mal organizada, pode resultar em uma solução não tão próxima da ótima. No entanto, em muitos casos, a razão entre a solução da heurística e a solução ótima é próxima de 1, indicando eficácia.

A heurística mostrou-se particularmente eficiente para grafos completos, bipartidos, ciclos e grafos rodas, proporcionando soluções rápidas e de alta qualidade. Ainda assim, deve-se ter cautela com a organização da fila de prioridade no DSATUR para garantir a melhor solução possível.

Referências bibliográficas

NETO, Alfredo Silveira Araújo; GOMES, Marcos José Negreiros. PROBLEMA E ALGORITMOS DE COLORAÇÃO EM GRAFOS - EXATOS E HEURÍSTICOS. **Revista de Sistemas e Computação**, Salvador, v. 4, n. 2, p. 101-115, 25 ago. 2024.

GRELIER, C. gc_instances. 2022. Disponível em: <https://github.com/Cyril-Grelier/gc_instances>. Acesso em: 25 ago. 2024.