UNIVERSIDADE FEDERAL DE RORAIMA
PRÓ-REITORIA DE ENSINO E EXTENSÃO
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA ANÁLISE DE ALGORITMOS 2019.2

2º LISTA DE ANÁLISE DE ALGORITMOS

DISCENTE: Anny Caroline Walker Silva | DOCENTE: Herbert Rocha

[QUESTÃO – 01] Defina e dê exemplos:

(A) Grafos.

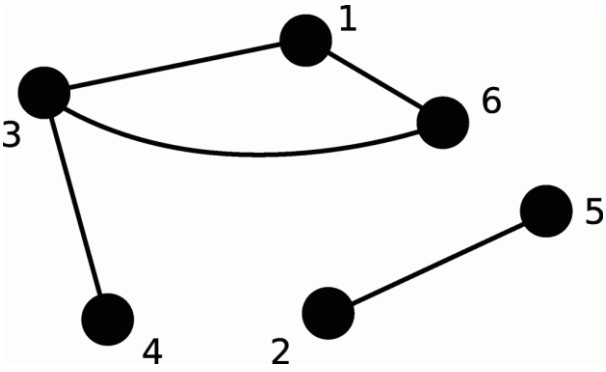
Grafo é um tipo abstrato de dado que representa a relação entre vértices conectadas através de arestas. A principal regra que restringe e dá forma ao grafo é o fato de que uma aresta só conecta um par de vértices.

Definições:

1. Por definição dois vértices conectados através de uma aresta são chamados adjacentes.
2. O nome que se dá para uma aresta que conecta dois vértices é incidente, mas apenas para esses vértices.
3. O nome de arestas incidentes a partir de um vértice define o grau desse grafo. Exemplo: cada vértice V está conectado a 3 outros vértices, indicando a existência de 3 arestas incidindo a partir de V, o que significa que o grafo é de grau 3.
4. Os subconjuntos de vértices e arestas produzidos em um grafo é chamado de sub grafo.
5. Um caminho é o nome que se dá para uma sequência percorrível de arestas entre vértices.
6. Um ciclo é formado quando existe um caminho que percorre diferentes vértices e arestas desde o primeiro vértice até o último. Uma definição mais matemática para o conceito de ciclo seria um caminho que percorre vértices $\{v_0, a_1, v_1, a_2, v_2, \ldots, a_k, v_k\}$, onde todos são diferentes entre si, exceto v_0 e v_k que são iguais, formando ciclo.

A expressão formal de um grafo é dada por G (V, E), onde G é o grafo, V é o número de vértices e E o números de arestas.

Figura 1. Exemplo de grafo G, onde os círculos pretos enumerados são os vértices V e as linhas são as arestas E.



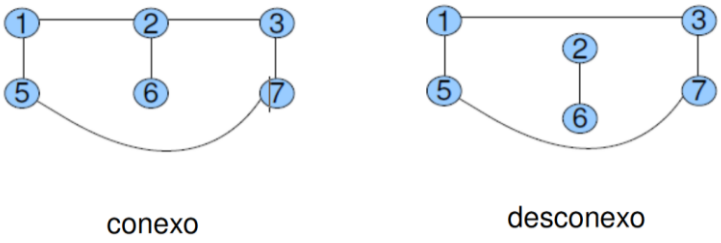
(B) Grafo conexo, acíclico e direcionado.

GRAFO CONEXO:

Um grafo G = (V, E) é dito conexo se existir pelo menos um caminho entre qualquer par de vértices, ou seja, mesmo que não sendo adjacentes os vértices podem ser acessos por qualquer outro vértice a partir de caminhos.

Caso exista um vértice inacessível por outros vértices através de caminhos, o grafo é dito como disconexo, situação ilustrada na Figura 2.

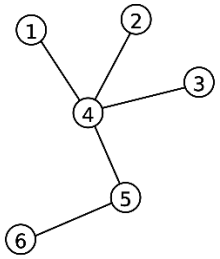
Figura 2. Exemplo de grafo conexo (à esquerda) e grafo disconexo (à direita).



GRAFO ACÍCLICO:

Um grafo é considerado acíclico se não possuir nenhum ciclo como descrito na definição 6 do elemento (A), exemplificado na Figura 3.

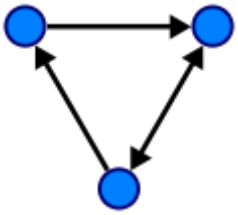
Figura 3. Exemplo de grafo acíclico.



GRAFO DIRECIONADO:

Um grafo direcionado ou orientado é chamado assim quando a aresta que liga um par de vértices deixa de ser uma via de mão dupla, restringindo a forma como são acessados os vértices. Dando um vértice V e um vértice V' de um grafo direcionado temos que ou a aresta entre eles leva de V para V' ou de V' para V. Exemplo ilustrado na Figura 4.

Figura 4. Exemplo de grafo direacionado.



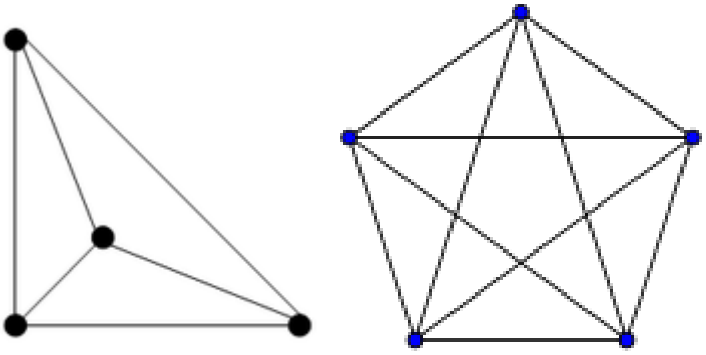
(C) Adjacência x Vizinhança em grafos.

Um vértice é dito adjacente a outro vértice se eles são conectados por uma aresta. Uma vizinhança é então um sub grafo do grafo principal formado pelos vértices adjacentes a um vértice v, assim como as arestas que os ligam. Na Figura 1 temos o vértice 6, que é adjacente aos vértices 1 e 3 e não adjacente aos vértices 2,4 e 5, mas sua vizinhança correspondente aos vértices 1 e 3 e também a aresta entre 1 e 3.

(D) Grafo planar.

Para ser considerado planar um grafo pode ser projetado em um plano sem que suas arestas se cruzem, situação ilustrada na Figura 5.

Figura 5. Grafo planar (à esquerda) e grafo não planar (à direita).



(F) Grafo completo, clique e grafo bipartido.

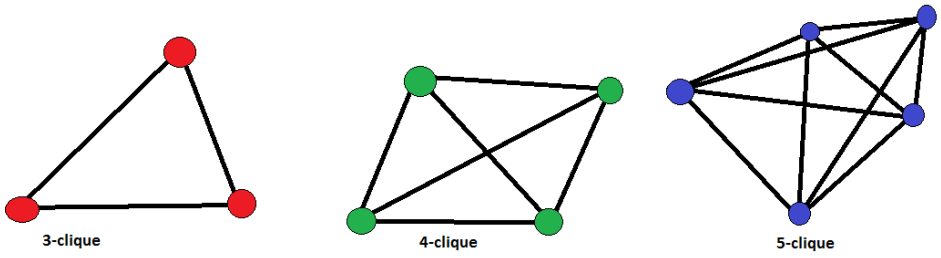
GRAFO COMPLETO:

Um grafo completo é um grafo em que todo vértice é adjacente a todos os outros vértices. A Figura 5 representa grafos completos, onde se observa que todos os vértices são conectados entre si. Um grafo não completo seria por exemplo o grafo da Figura 3.

CLIQUE:

Um clique é um subconjunto de vértices de um gráfico não direcionado, de modo que cada dois vértices distintos no clique sejam adjacentes; isto é, um sub grafo completo dentro de um grafo. Exemplo na Figura 6.

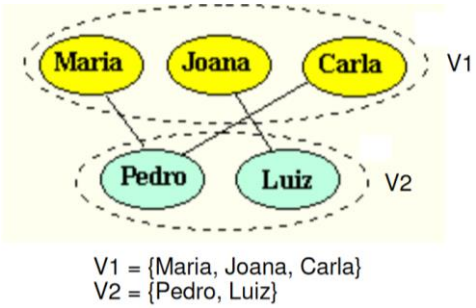
Figura 6. Exemplos de cliques de tamanho 3, 4 e 5 em diferentes grafos.



GRAFO BIPARTIDO:

Para um grafo ser considerado bipartido é preciso que ele seja dividido em dois conjuntos distintos, onde não há aresta entre os vértices do mesmo conjunto. Nesse caso as arestas dos vértices do conjunto A só levam para os vértices do conjunto B e vice-versa, fazendo que vértices no mesmo conjunto não sejam equivalentes. Como ilustra a Figura 7 abaixo.

Figura 7. Grafo bipartido.

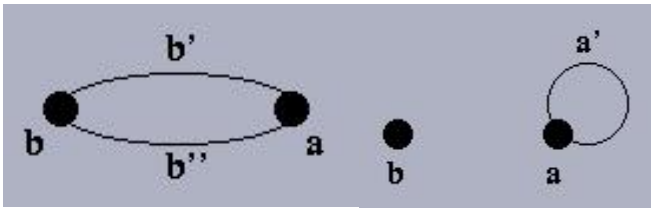


(G) Grafos simples x multigrafo x digrafo.

GRAFO SIMPLES:

Um grafo é simples se não contém ciclos e nem arestas múltiplas, situações mostradas na Figura 8.

Figura 8. Arestas múltiplas b’ e b’’ entre os vértices b e a (à esquerda) e um ciclo a’ no vértice a (à direita).



GRAFO MULTIGRAFO:

Esse tipo de grafo permite a que existam arestas múltiplas, ou seja, mais de uma aresta conectando o mesmo par de vértices, exemplificado à esquerda da Figura 8.

GRAFO DIGRAFO:

Grafo dígrafo é um grafo com flechas nas arestas que define as direções a serem seguidas entre um vértice e outro. É bastante parecido com um grafo dirigido, porém um grafo dirigido pode ser também multigrafo, mas um grafo dígrafo só permite uma única aresta direcional entre dois vértices.

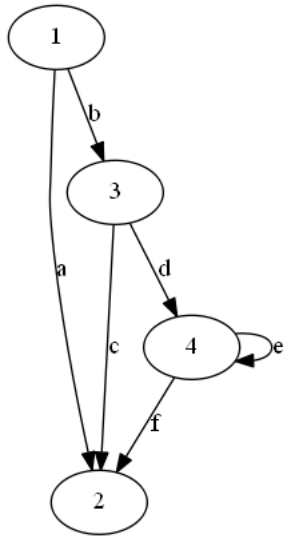
[QUESTÃO – 02] Defina e apresente exemplos de matriz de incidência, matriz de adjacência e lista de adjacência. Adicionalmente, descreva o impacto (vantagens e desvantagens) da utilização de matriz de adjacência e lista de adjacência.

MATRIZ DE INCIDÊNCIA:

Uma matriz de incidência é uma representação bidimensional e matricial de um grafo que usa a definição de incidência de grafos para estrutura-lo. Nesse caso uma dimensão da matriz corresponde as arestas do grafo, e a outra dimensão corresponde aos vértices, a interseção entre uma dimensional e outra da matriz é preenchida com 1 se a aresta é conectada ao vértice, e 0 se não é. Representação na Figura 9.

Figura 9. Matriz de incidência do grafo G (à esquerda) e grafo G (à direita).

	a	b	c	d	e	f
1	1	1	0	0	0	0
2	1	0	1	0	0	1
3	0	1	1	1	0	0
4	0	0	0	1	2	1

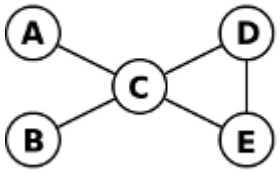


Essa representação pode ter desvantagem em grafo grandes (com muitos vértices) e espaçados (com poucas arestas), isso geraria uma matriz de incidência majoritariamente ocupada por zeros, um grande desperdício de memória. Armazenamento necessita Ω (V^2) de espaço, então ler ou examinar a matriz nos custa O (V^2), valor muito alto para grafos gigantes.

MATRIZ DE ADJACÊNCIA:

Seguindo a definição de adjacência da teoria de grafos, uma matriz de adjacência é a representação de um grafo em duas dimensões representando o conjunto de vértices do grafo. A interseção entre essas dimensões indica se existe ou não conexão (aresta) entre os vértices. Ilustração desse caso é a Figura 10.

Figura 10. Grafo G à esquerda e matriz adjacente de G à direita.



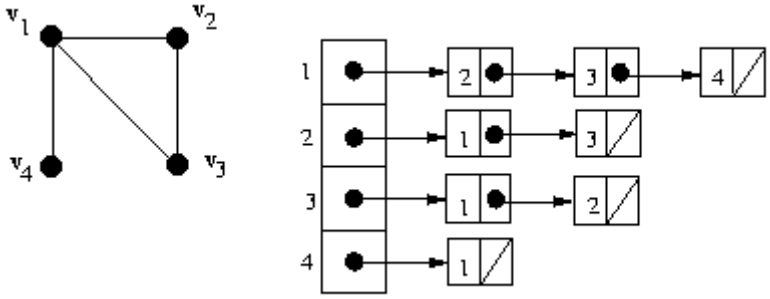
	A	B	C	D	E
A	0	0	1	0	0
B	0	0	1	0	0
C	1	1	0	1	1
D	0	0	1	0	1
E	0	0	1	1	0

São duas as desvantagens para esse tipo de representação, a primeira é o desperdício de memória com excessos de 0 em situação de grafo com muitos vértices e poucas arestas, a segunda é que para verificar quais vértices são adjacentes ao vértice V devemos percorrer toda a coluna de V, desperdício de processamento, pois se ormazenamento necessita Ω (V^2) de espaço, então ler ou examinar a matriz nos custa O (V^2), valor muito alto para grafos gigantes.

LISTA DE ADJACÊNCIA:

Uma lista de adjacência é uma representação de grafo baseado na estrutura lista a partir das adjacências entre os vértices do grafo, onde cada vértice vira um elemento da lista principal, e sub listas encadeadas aos vértices da lista principal indicam os vértices adjacentes a ele. Na Figura 11 podemos ver visualmente essa representação.

Figura 11. Grafo G à esquerda e lista de adjacência de G à direita.



Pode custar O (V) para verificar se existe uma aresta entre dois vértices, pois no pior caso podem existir o total de V vértices adjacentes ao verificado.

[QUESTÃO – 03] Defina, explicando as principais características e exemplifique:

(A) Enumeração explícita x implícita.

Métodos de enumeração explícita são métodos de pesquisa de operações que são usados para resolver um problema de otimização, onde não existem algoritmos de solução analítica e o espaço da solução é finito. Ele determina todas as soluções viáveis. A solução ideal será encontrada por comparação. Devido à alta complexidade computacional (em n variáveis com k valores possíveis, surgem soluções k*n), esse método é aplicável apenas a problemas muito pequenos. Em geral, você passará para enumeração limitada, ramificação e limite ou otimização dinâmica.

O método de enumeração implícita é frequentemente usado para resolver modelos de 0 a 1. A enumeração implícita se baseia no fato de que cada variável deve ser igual a 0 ou 1 para simplificar os componentes de ramificação e de limitação dos componentes de ramificação e limite e componentes de ligação do processo de ramificação e limite e determinar com eficiência quando um nó é inviável.

A árvore usada no método de enumeração implícita é semelhante à da marca e do método encadernado. No entanto, a ramificação na enumeração implícita é específica. 0 ou 1.

(B) Programação Dinâmica.

Esse paradigma de programação é utilizado por muitos algoritmos. Ele consiste em uma estratégia de projeto de algoritmos, baseado na recursão e pode ser descrito como uma recursão auxiliada por tabela.

Como utiliza o recurso de recursividade, cada instância do problema que se tenta resolver com programação dinâmica é resolvida a partir da solução de instâncias menores, particionadas a partir da instância original. O diferencial da programação dinâmica é a tabela auxiliar que armazena as soluções das várias de instâncias menores. Normalmente o tempo de consulta é proporcional ao número de entrada.

Primeiro é preciso que o problema determinar se o problema é recursível, e se for então é possível aplicar o paradigma de programação dinâmica. Para determinar essa recursividade do problema, deve ser avaliado se a solução de toda instância do problema contém soluções de instância particionadas do mesmo problema.

Diferentemente dos algoritmos recursivos comuns, o algoritmo produzido pela programação dinâmica não realiza mais de uma vez a solução da mesma instância particionada, pois a tabela auxiliar está lá justamente para isso, é ela que armazena as soluções já encontradas e assim torna possível passar recuperar a solução de uma instância particionada sem ter que executá-la novamente.

Um exemplo desse tipo de paradigma é o algoritmo de Dijkstra descrito abaixo quando aplicado ao problema de encontrar menor distância entre um caminho e outro a partir do peso dos caminhos. O conceito de programação dinâmica está aplicado na fila criada na linha 4, que adicionada numa estrutura enquanto que verifica se a solução para o problema verificado já foi encontrada ou não, evitando assim o processamento extra daquilo que já foi determinado.

Tabela 1. Exemplo do algoritmo Dijkstra para solucionar distância com pesos usando programação dinâmica.

```
Dijkstra (n, Adj, f, r)
1  para  u ← 1  até  n  faça
2  dist[u] ← ∞
3  dist[r] ← 0
4  Q ← Cria-Fila-Vazia ( )
5  para v crescendo de 1 até n faça
```

```
6  Insere-na-Fila (v, Q)
7  enquanto  Q  não está vazia faça
8  u ← Extrai-Min (Q)
9  para cada  v  em  Adj[u]  faça
10 se  dist[u]+f(uv) < dist[v]
11 então  Diminui-Chave (v, Q, dist[u]+f(uv))
12 devolva  dist[1..n]
```

Esse algoritmo resolve o problema de distância com pesos quando não há arcos de peso negativo. O algoritmo recebe um digrafo com vértices 1, 2, ..., *n* e arcos definidos por listas de adjacência *Adj*[1..*n*]. Recebe também um vértice *r* e uma função *f* que atribui um peso positivo ou nulo a cada arco. O algoritmo devolve a *f*-distância de *r* a cada um dos vértices do dígrafo.

(C) Algoritmo Guloso.

Esse algoritmo tenta resolver problemas a partir da estratégia de estágios, onde durante cada estágio é feita uma decisão localmente ótima, que ao chegar no estágio final proporciona a soma total de decisões localmente ótimas, produzindo uma solução para o problema (EBERT, 2019).

Nem sempre a soma total das decisões do algoritmo guloso resulta na solução ótima para o problema, então essa solução é do tipo aproximada, ou heurística.

A vantagem desse algoritmo é que sua implementação é simples e mesmo assim oferece uma solução aproximada, que pode ser útil em casos de problema que a exatidão da solução é dispensável.

No caso do problema de coloração de grafo, a implementação de uma solução aproximada com uso do algoritmo guloso tem como pior caso a situação em que o algoritmo deve visitar todas as arestas e vértices, onde O ($V^2 + E$), onde V corresponde ao total de vértices e E o total de arestas de um grafo G.

Um exemplo de algoritmo guloso é representado abaixo, onde a estratégia gulosa é aplicada para tentar colorir um grafo. A estratégia é primeiramente designer a primeira cor para o primeiro vértice e a partir dele colorir os próximos vértices designando cores de menor índice possível de tal forma que os adjancetes desse vértice não repitam essa cor.

Tabela 2. Exemplo do algoritmo guloso aplicado em coloração aproximada de grafo.

```
Entrada: G = (V, E)
Saída: G colorido sem vértices adjancentes com mesma cor
Início
Seja A := {v_1, v_2, ... , v_n} uma sequência de vértices V de G.
Seja C := {v_1, v_2, ... , c_n} uma sequência de cores.

Cor de A[v_1] := C[1]
Para 2 até n faça
Processa todas as cores usadas pelos adjacentes de A[v_n]
Procura a primeira cor em C que não seja usada pelos adjacentes de A[v_n]
```

Fim Para Fim

(D) Backtracking.

Esse algoritmo tenta resolver problemas a partir da estratégia de busca em profundidade, que utiliza o recurso de recursão para retornar depois de chegar em uma extremidade da entrada, e assim buscar por outro caminho, até que a condição de satisfação seja encontrada e o melhor caminho verificado.

Para o problema de coloração de grafos o algoritmo do backtracking procura uma solução a partir de um número m de cores onde existe uma solução que siga a restrição de não repetição de cor em adjacentes.

No caso do problema de coloração de grafo, a implementação de uma solução aproximada com uso do algoritmo backtracking tem como pior caso a situação em que o algoritmo deve visitar todos os vértices a partir de um vértice para escolher a melhor combinação de cores, ou seja, a que use menor quantidade de cores, sendo O (2^V).

Um exemplo de algoritmo backtracking é expressado abaixo, onde tenta-se aplicar a técnica backtracking para realizar coloração de grafo com mínima combinação de cores. A estratégia é combinar todas as cores disponíveis em um determinado vértice em seus adjacentes, e nos adjacentes de seus adjacentes de forma que use a menor quantidade de cores possível.

Tabela 3. Exemplo algoritmo backtracking aplicado em coloração aproximada de grafo.

Entrada: G = (V, E) Saída: G colorido sem vértices adjancentes com mesma cor Início Seja A := $\{v_1, v_2, \dots, v_n\}$ uma sequência de vértices V de G. Seja C := $\{v_1, v_2, \dots, c_m\}$ uma sequência de cores. Para 0 até n faça Recursão Fim Para Recursão Para 1 até m faça Se todo vértice possuir cor então Saia da recursão Senão Checa se a cor designada disponível e designa ela para o vértice Recursão Fimpara Fim

[QUESTÃO – 04] Implemente uma solução para multiplicação de matrizes utilizando programação dinâmica, visando determinar uma ordem em que as matrizes sejam multiplicadas, de modo a minimizar o número de multiplicações envolvidas.

Algoritmo disponível em:

https://github.com/silvandante/atividades_AnaliseAlgoritmos/tree/master/MinimizacaoMultiplicaoMatrizes

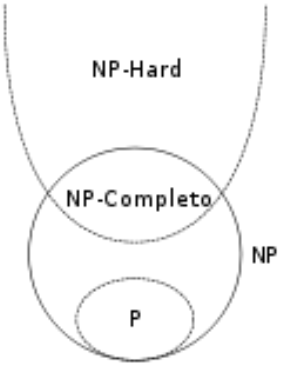
[QUESTÃO – 05] Defina e exemplifique:

(A) Problema SAT x Teoria da NP-Completude.

A classe de problemas NP-Completo fica entre a interseção dos conjuntos de problemas NP-Difícil e NP. Onde NP é uma classe de problemas que podem ser resolvidos e verificados com tempo polinomial não determinístico a partir de algoritmos não determinísticos, que são algoritmos que encontram a solução exata e ótima em tempo polinomial.

Já os problemas do tipo NP-Difícil são problemas que são pelo menos tão difíceis quanto os problemas mais difíceis da classe NP. Na Figura 12 é apresentado um diagrama dos conjuntos dos problemas NP, NP-Difícil, NP-Completo e P

Figura 12. Diagrama de classes de problema P, NP, NP-Completo e NP-Difícil.



Um problema é dito como NP-Completo se é verificado que está em NP, e se todo problema em NP-Difícil é redutível em tempo polinomial.

Quanto o problema SAT ou problema da satisfabilidade booleana é o problema base da classe de problemas NP-Completo. Esse problema consiste em verificar e determinar se existe uma determinada atribuição de valores para variáveis de uma fórmula booleana, de tal forma que esses valores satisfaçam a fórmula. Uma instância de SAT é o problema 3-SAT que procura encontrar satisfabilidade de valoração para uma equação booleana que combina 3 variáveis. O SAT pode ser redutível em clique (QUESTÃO - 06) e assim ser equiparado a outros problemas da classe NP-Completo, pois clique é NP-Completo.

(B) Classes P, NP, NP-Difícil e NP-Completo.

Item (A) da [QUESTÃO – 05]

[QUESTÃO – 06] Descreva a redução (prove a NP-Completude) do problema do SAT ao Clique. Apresente o pseudo-código do algoritmo NP e mostre graficamente as instâncias e soluções, no processo de redução.

Para fazer essa redução usaremos a instância de SAT chamada 3-SAT citada na [QUESTÃO – 05] item (A). Faremos o passo a passo da redução de 3-SAT em um problema Clique em tempo polinomial.

Dada uma de fórmula booleana com combinações de 3 variáveis, o problema 3-SAT é verificar e definir qual valoração para essas 3 variáveis é satisfazível com relação as restrições da fórmula.

Dado um grafo G = (V, 3) e inteiro K, o problema do Clique é encontrar se G contém um clique de tamanho maior ou igual a K.

Assumimos que fórmula ϕ de K cláusulas com 3 literais pode ser reduzida em um problema K-Clique.

Constrói-se um grafo G de K grupos com um máximo de 3 nós em cada grupo.

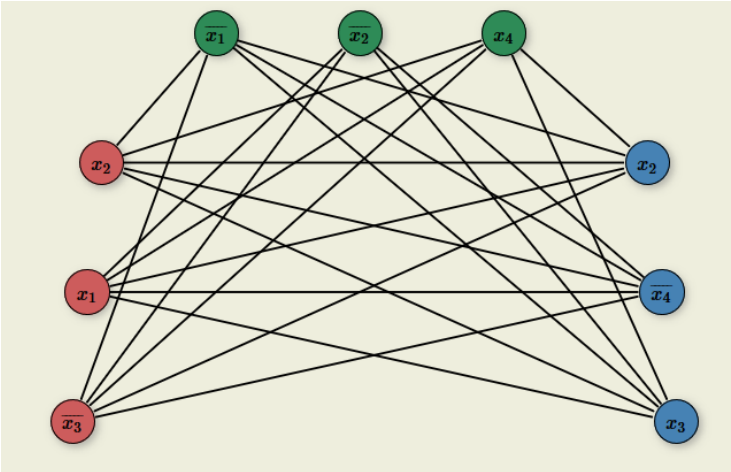
Cada grupo corresponde a uma cláusula literal em ϕ.

Cada nó do grupo é rotulado com um literal da clásula.

É adicionada uma aresta entre todos os pares de nós em grupos diferentes, exceto no par da forma (x, x').

Nenhuma aresta é adicionada em nós no mesmo grupo.

Seja ϕ = (x2 + x1 + !x3) . (!x1 + !x2 + x4) . (x2 + !x4 + x3) e o grafo G construído com K grupos e 3 nós em cada grupo:



Constatações sobre o grafo G:

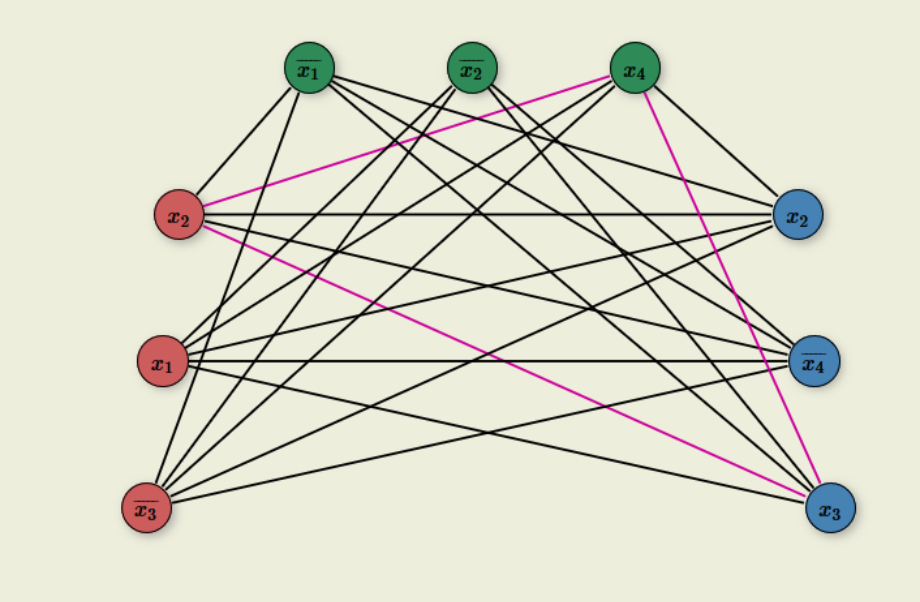
- Se dois nós no grafo estão conectados, os literais correspondentes podem ser simultaneamente designados como sendo TRUE (já que não existe arestas entre variáveis do mesmo grupo).
- Se dois literais de cláusulas diferentes podem ser designados como TRUE simultaneamente, então os nós correspondentes a esses literais no grafo estão conectados.
- A construção desse grafo pode então ser performada em tempo polinomial.

Assumimos que G tem um clique K se e somente se a instância 3-SAT ϕ é satisfazível.

Então podemos concluir que:

- Se o grafo G tem um clique K, então o clique tem exatamente um nó para cada grupo (já que nós no mesmo grupo não são conectados).
- Todos nós de um clique estão conectados, logo todos os literais correspondentes podem ser designados como TRUE simultaneamente.
- Se ϕ é satisfazível, então A é uma valoração satisfatória. Selecionamos então toda cláusula literal que é do tipo TRUE em A para construir S. Onde ||S|| = K.
- Já que nenhum par de literais em A são do mesma cláusula e todos os termos são simultaneamente TRUE, então todos os nós correspondentes restantes no grafo estão conectados uns aos outros, formando assim um clique de tamanho K, já que sabemos que o grafo G possui um clique K.

Exemplo de um clique K em G destacado de lilás:
A valoração correspondente é: x2 é TRUE, x3 é TRUE e x4 é TRUE.



Encontramos o clique K no grafo G equivalente a satisfabilidade para as cláusulas da fórmula booleana ϕ, logo, provamos que 3-SAT é NP-Completo.

SOLUÇÃO ENCONTRADA NO SITE DO LIVRO OpenDSA Data Structures and Algorithms Modules Collection. Disponível em: < https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/threeSAT_to_clique.html>.