Universidade Federal de Roraima
Departamento de Ciência da Computação
Análise de Algoritmos

DISCIPLINA: Análise de Algoritmos – DCC606

2ª Lista
Prazo de Entrega: 18/06/2019

ALUNO(A): FRANCISCO PIRES JUNIOR NOTA:_____

ATENÇÃO: Descrever as soluções com o máximo de detalhes possível, inclusive a forma como os testes foram feitos. Todos os artefatos (relatório, código fonte de programas, e outros) gerados para este trabalho devem ser adicionados em um repositório online no github, com o nome do repositório no seguinte formato: **nomeDoAluno_AA_Lista2_rr_2019**. Na resposta para as questões de implementação deve ser apresentado: o modo de compilar/executar o programa; a linha de comando para executar o programa; e um exemplo de entrada/saída do programa.

[QUESTÃO – 01]
Especifique cada problema e calcule o M.C. (melhor caso), P.C. (pior caso), C.M. (caso médio) e a ordem de complexidade para algoritmos (os melhores existentes e versão recursiva e não-recursiva) para problemas abaixo. Procure ainda, pelo L.I. (Limite Inferior) de tais problemas:

(A) **N-ésimo número da seqüência de Fibonacci:** Dada uma sequencia de Fibonacci, o N-ésimo termo da sequencia é definido pela seguinte equação F(n) = F(n-1) + F(n-2) de tal maneira que os dois termos iniciais F(0) = 1, F(1) = 1, assim, temos que por exemplo, uma sequencia seria 1 , 1 , 2 , 3 , 5 , N, onde N seria o N-ésimo termo da sequencia.

Para a função recursiva, temos que o custo para o cálculo de F(n) é :

$O(\varphi^n)$, onde $\varphi = (1 + 51/2)/2 = 1{,}61803$ é a proporção áurea, assim, a complexidade do algoritmo recursivo no melhor e no pior caso é: 2^n

Já na versão interativa, temos que a complexidade de tempo é O(n), e de espaço O(1), assim, o melhor e o pior caso é **n** .

(B) Geração de todas as permutações de um número

Para o caso da geração de todas as permutações de um número, a complexidade no caso médio e pior caso será O(n!) . No melhor caso, quando temos apenas 1 elemento, seria O(1).

Exemplo, dada a entrada 1,2,3, as permutações desses números seriam:

1,2,3
1,3,2
2,1,3
2,3,1
3,1,2
3,2,1

Note que foram geradas 6 permutações possíveis, ou seja, 3! = 6.

[QUESTÃO – 02]
Defina e dê exemplos:

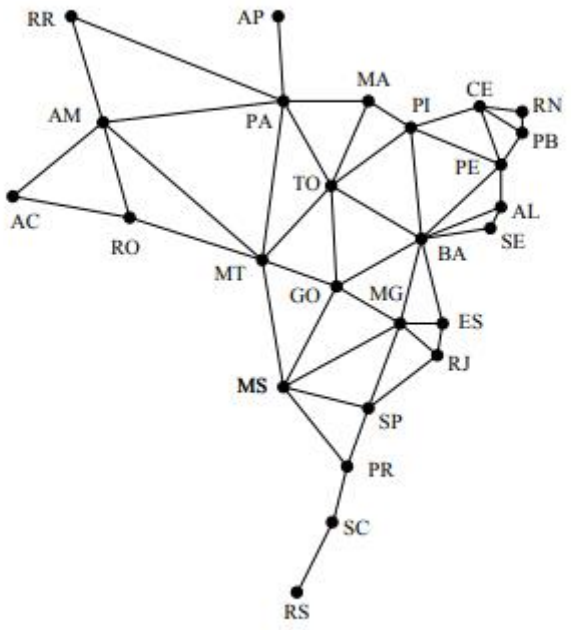
(A) Grafos.

Um grafo é um par (V,A) em que V é um conjunto arbitrário e A é um subconjunto de V. Os elementos de V são chamados de **vértices** e os de A são chamados de **arestas.** Uma aresta {v,w} será denotada simplesmente por vw ou por wv. Diremos que a aresta vw incide em v e em w e que v e w são as **pontas** da aresta. Se vw é uma aresta, diremos que os vértices v e w são vizinhos ou adjacentes.

Muitas vezes é conveniente dar um nome ao grafo como um todo. Se o nome do grafo for G, o conjunto dos seus vértices será denotado por V (G) e o conjunto das suas arestas por A(G). O número de vértices de G é denotado por n(G) e o número de arestas por m(G); portanto,

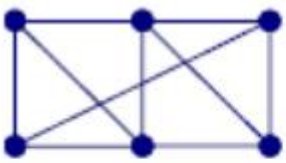
$$n(G) = |V(G)| \text{ e } m(G) = |A(G)| .$$

EXEMPLO: **O grafo dos estados do Brasil**



(B) Grafo conexo, acíclico e direcionado.

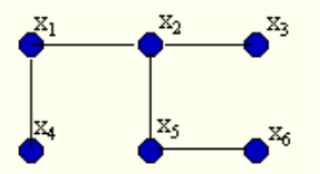
Grafos – Conexo: Um grafo G=(V, E) é conexo se existir um caminho entre qualquer par de vértices. Caso Contrário é desconexo.



G_1 conexo

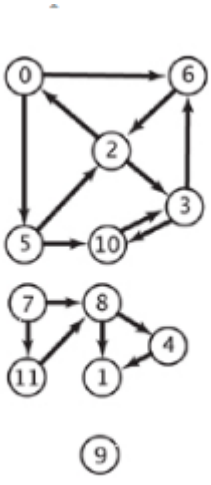
G_2 desconexo

Grafos – Acíclico: Um grafo é acíclico quando não possuir ciclos.



Grafos Direcionado: Um grafo é dito direcionado se a aresta começa na sua ponta inicial e termina na sua ponta final, ou seja, Uma aresta {v,w} será denotada simplesmente por vw, iniciando em v e terminando em w. Quando o grafo é direcionado o inverso não ocorre, ou seja, wv não ocorre, a não seja que seja especificado essa condição.

Exemplo:

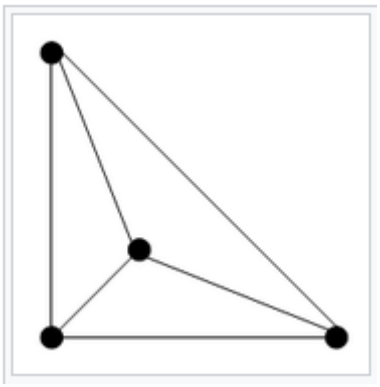


(C) Adjacência x Vizinhança em grafos.

Dizemos que um vértice w é vizinho de um vértice v se v-w é uma aresta do grafo. Dizemos também nessa circunstância que w é adjacente a v. A relação de vizinhança não é simétrica: w pode ser vizinho de v sem que v seja vizinho de w.

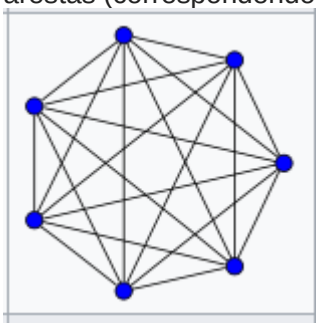
(D) Grafo planar.

Um **grafo planar** é um grafo que *pode ser imerso* no plano de tal forma que suas arestas não se cruzem, esta é uma idealização abstrata de um grafo plano, um **grafo plano** é um grafo planar que *foi* desenhado no plano sem o cruzamento de arestas.

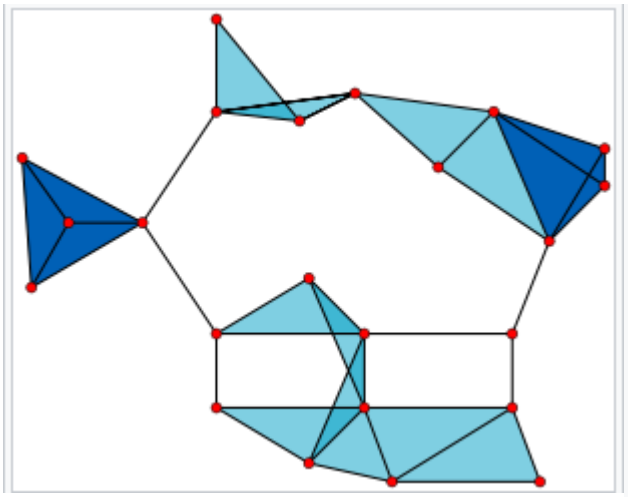


(F) Grafo completo, clique e grafo bipartido.

Grafo completo é o grafo simples em que, para cada vértice do grafo, existe uma aresta conectando este vértice a cada um dos demais. Ou seja, todos os vértices do grafo possuem mesmo grau. O grafo completo de n vértices é frequentemente denotado por K_n. Ele tem $n(n-1)/2$ arestas (correspondendo a todas as possíveis escolhas de pares de vértices).



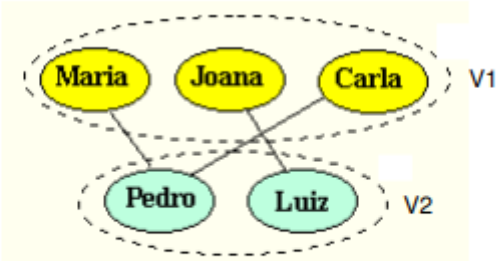
Clique em um grafo é um subgrafo que também é um grafo completo.



Um grafo com 23 cliques de 1-vértice (seus vértices), 42 cliques de 2-vértices (suas arestas), 19 cliques de 3-vértices (os triângulos em azul claro), e 2 cliques de 4-vértices (azul escuro). Seis das arestas e 11 dos triângulos formam cliques maximais. As duas 4-cliques em azul escuro são tanto máximas quanto maximais, e o número de clique do grafo é 4.

Grafos – Bipartido: Os vértices V do grafo são divididos em dois conjuntos V1 e V2. Não há arestas entre elementos do mesmo conjunto. Toda aresta de G une um vértice de V1 a outro de V2

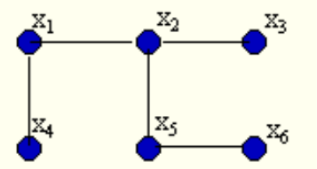
Seja o Grafo G:



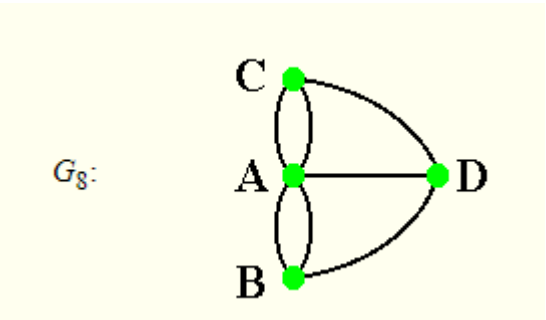
V1 = {Maria, Joana, Carla}
V2 = {Pedro, Luiz}

(G) Grafos simples x multigrafo x digrafo.

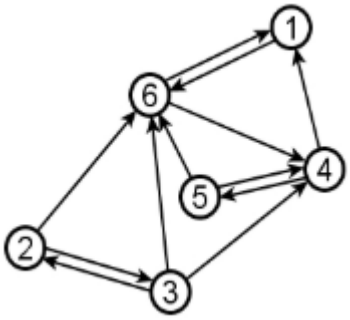
- **Grafo Simples:** é um grafo não direcionado, sem laços e existe no máximo uma aresta entre quaisquer dois vértices (sem arestas paralelas). Para um grafo simples, o número de vizinhos de um vértice é igual à sua valência.



- **Multigrafo** é um grafo que permite múltiplas arestas ligando os mesmos vértices (arestas paralelas).



- **Um digrafo** é um objeto formado por dois conjuntos: um conjunto de vértices e um conjunto de arcos. Cada arco é um par ordenado de vértices; o primeiro vértice do par é a ponta inicial do arco e o segundo é a ponta final do arco. Você pode imaginar que um digrafo é uma espécie de mapa rodoviário: os vértices são cidades e os arcos são estradas de mão única.

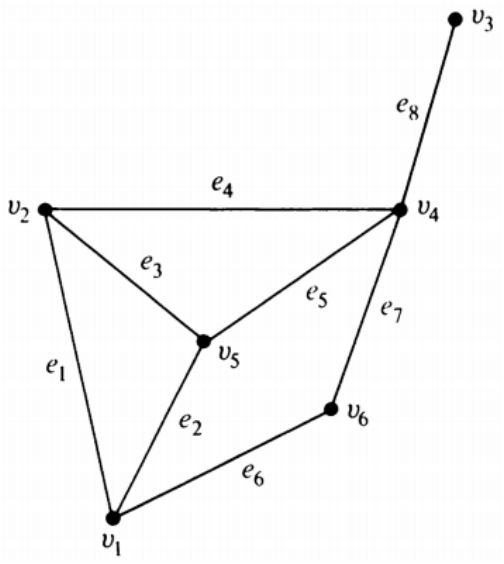


[QUESTÃO – 03]
Defina e apresente exemplos de matriz de incidência, matriz de adjacência e lista de adjacência. Adicionalmente, descreva o impacto (vantagens e desvantagens) da utilização de matriz de adjacência e lista de adjacência.

Matriz de Adjacência

É uma matriz nxn, denotada por $X = [x_{ij}]$ e definida como :

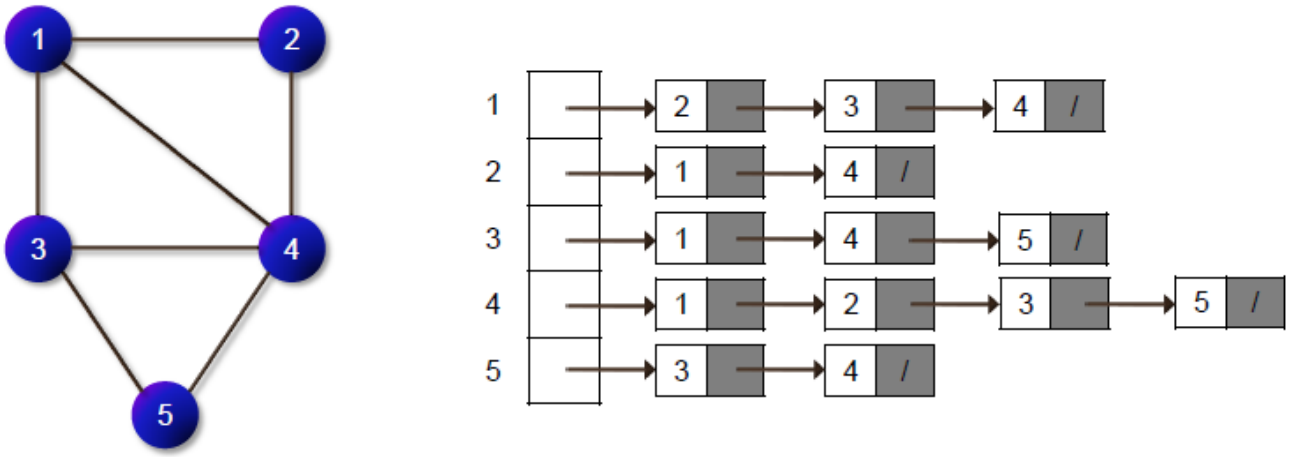
$X_{ij} =$ 1 se existe uma aresta entre os vértices vi e vj
0 caso contrário



$$X = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Lista de adjacência

Nesta representação as *n* linhas da matriz de adjacência são representadas como *n* listas encadeadas. Por definição, o vetor de listas de adjacência de uma grafo tem uma lista encadeada associada com cada vértice do grafo. A lista associada com um vértice v contém todos os vizinhos de v.



Vantagens e desvantagens da Lista de Adjacência e Matriz de adjacência

Ao comparar ambos as técnicas de representação de grafos, em termos de complexidade de espaço de memória de um algoritmo que use uma matriz de adjacência para armazenar o grafo é O (n^2). A desvantagem dessa matriz em relação a lista de adjacência é que questões como: “quantas arestas existem no grafo”? ou “o grafo é conexo?”, essas

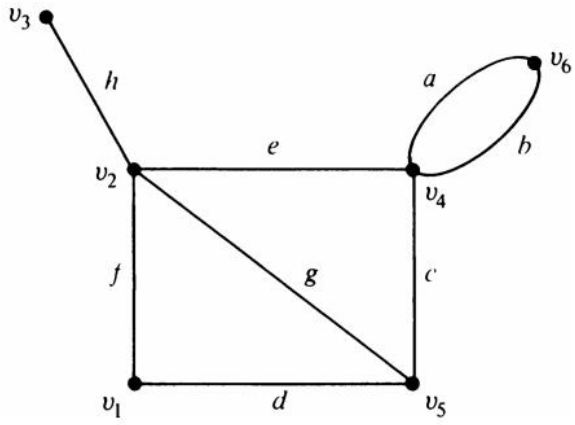
questões serão respondidas em tempo de $O(n^2)$. Assim, se grafo possui poucas arestas, este tempo pode ser melhorado utilizando-se listas de adjacência. Quando falamos sobre as Listas de Adjacência, No caso de um grafo não dirigido com *n* vértices e *e* arestas, O grau de qualquer vértice *i* em um grafo não dirigido pode ser determinado pelo número de nós na lista do vértice *i*. O número total de arestas no grafo pode ser determinado em tempo *O(n+e), o que o torna mais vantajoso em relação a matriz de adjacência*.
Temos como vantagens que, ao utilizar lista de adjacência, é muito fácil de se obter os vizinhos de um vértice i, pois não necessita percorrer os outros vértices, tendo um custo O(n) para vértice em questão e matriz de adjacência a implementação da mesma é mais fácil.

Lista de Incidência

Seja G um grafo com n vértices e m arestas. Sua matriz de incidência é uma matriz de ordem n × m, denotada por A = [a_{ij}], definida como

xij = 1 se a aresta a j é incidente no v_i ,
0 caso contrário.

OBS: A complexidade em termos de espaço de memória de um algoritmo que use uma matriz de adjacência para armazenar o grafo é O (nm).



	a	b	c	d	e	f	g	h
v_1	0	0	0	1	0	1	0	0
v_2	0	0	0	0	1	1	1	1
v_3	0	0	0	0	0	0	0	1
v_4	1	1	1	0	1	0	0	0
v_5	0	0	1	1	0	0	1	0
v_6	1	1	0	0	0	0	0	0

[QUESTÃO – 04]
Defina, explicando as principais características e exemplifique:

(A) Enumeração explícita x implícita.

A enumeração explícita consiste em testar todas soluções possíveis para um determinado problema, basicamente, são construídos algoritmos de tentativa e erro. Essa técnica busca escolher sempre a melhor. Um exemplo bastante conhecido e é um exemplo que caracteriza a enumeração explícita é o problema do caixeiro viajante, onde geramos todas as permutações possíveis, onde cada permutação é um caminho e verificamos qual delas é a melhor solução para o prolema. Já a implícita, baseia-se na idéia de desenvolver uma enumeração inteligente das soluções candidatas à solução ótima inteira de um problema. iniciamos com uma solução qualquer, ou seja, uma solução parcial, e dai buscamos a melhor solução e partir da geração de todas soluções que podemos explorar usando a enumeração explicita. Um exemplo de enumeração implícita, é a aplicação da técnica especialmente em problemas de optimização discreta e combinatorial.

(B) Programação Dinâmica.

PD é basicamente um **esquema de enumeração** de soluções que visa, através de uma abordagem de **divisão-e-conquista**(**decomposição**), minimizar o montante de computação a ser feito.

Exemplo:
Qual a melhor maneira de se fazer o produto
entre n matrizes?

Resolver esse problema por tentativa e erro, isto é, testar
todas as ordens possíveis de fazer o produto das matrizes
para se obter qual é a melhor (f (n)), é um processo
exponencial em n (f (n) _ 2n>=2 [1]).
Usando programação dinâmica é possível obter um
algoritmo $O(n^3)$!

(C) Algoritmo Guloso.

Algoritmo **Guloso** é uma solução comum para problemas de otimização. São tipicamente usados para resolver problemas esses tipos de problemas . Um exemplo de problema onde aplicação de um algoritmo guloso pode ser utilizado é em encontrar o caminho mais curto entre duas cidades. Um algoritmo guloso escolhe a estrada que parece mais promissora no instante atual e nunca muda essa decisão, independentemente do que possa acontecer depois.

(D) Backtracking.

Backtracking refere-se a um tipo de algoritmo para encontrar todas (ou algumas) soluções de um problema computacional, que incrementalmente constrói candidatas de soluções e abandona uma candidata parcialmente

construída tão logo quanto for possível determinar que ela não pode gerar uma solucão válida. Backtracking pode ser aplicado para problemas que admitem o conceito de "solução candidata parcial" e que exista um teste relativamente rápido para verificar se uma candidata parcial pode ser completada como uma soluçãao válida. Alguns exemplos famosos de uso de backtracking: **I Problema das Oito Rainhas I Passeio do Cavalo I Labirinto.**

[QUESTÃO – 06]
Defina e exemplifique:

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

Na teoria da complexidade computacional, o **problema da satisfatibilidade booleana (SAT)** é reconhecidamente um dos primeiros NP-completo. O problema visa encontrar uma solução verdade para uma dada fórmula booleana, e caso encontrada, a fórmula é considerada "satisfatível". Por exemplo:

Se x1, x2, x3 e x4 são variáveis booleanas de uma fórmula booleana qualquer como:

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$$

Caso seja possível encontrar valores para as quatro variáveis que torne a dada fórmula verdadeira, o problema SAT é resolvido. O problema do SAT é frequentemente utilizado para provar a NP-Completude de outros problemas.

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

P: Denota o conjunto de problemas que podem ser resolvidos em tempo polinomial $O(n^k)$, onde **k** é uma constante. Podemos ter a classe **P** como a classe de problemas que são solúveis para um computador real e um ótimo exemplo de um problema que pertence a essa classe é um exemplo de um problema de ordenação que é utilizado o algoritmo de ordenação Bubble Sort, cuja a complexidade é $O(n^2)$.

NP: Os problemas da Classe NP são problemas que podem ser resolvidos em tempo polinomial, mas NP significa "nondeterministic polynomial time", um termo que remonta às raízes da teoria da complexidade. Um exemplo de problema pertencente a essa classe é a do problema do ciclo longo, que está na classe NP, pois é possível verificar em tempo polinomial se uma dada sequência de vértices é um ciclo do grafo e tem comprimento maior que *k*.

NP-dificil: NP-difícil (ou **NP-hard**, ou **NP-complexo**) na teoria da complexidade computacional, é uma classe de problemas que são, informalmente, "Pelo menos tão difíceis quanto os problemas mais difíceis em NP". Um exemplo de um problema NP-difícil é o problema de otimização de se encontrar a rota de menor custo através de todos os nós de um grafo ponderado. Isto é comumente conhecido como o Problema do caixeiro viajante.

NP-completo: Na teoria da complexidade computacional, a classe de complexidade é o subconjunto dos problemas NP de tal modo que todo problema em NP se pode reduzir, com uma redução de tempo polinomial, a um dos problemas NP-completo. Pode-se dizer que os problemas de NP-completo são os problemas mais difíceis de NP e muito provavelmente não formem parte da classe de complexidade P.
O problema de isomorfismo de subgrafos é **NP-completo**. O problema do isomorfismo do grafo é suspeito de não estar em **P** nem NP-completo, ainda que obviamente está em **NP**. Este é um exemplo de um problema que se imagina difícil, mas não tanto para estar em NP-completo.

[QUESTÃO – 7]
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.

A redução do problema do SAT ao Clique consiste em transformar uma dada instância do SAT em uma instância do clique. A satisfabilidade dessa transformação é atingida se para a condição da instância em do SAT for verdadeira, a do clique também terá que ser verdadeira.

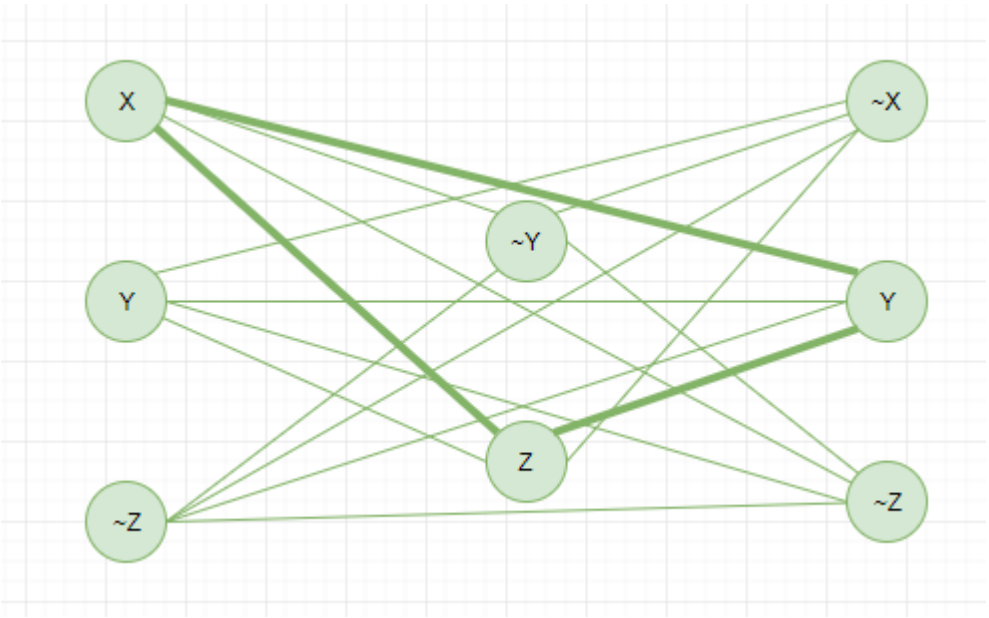
Por exemplo: Dada a seguinte expressão booleana (instância do SAT):

Φ = (X v Y v ~Z) ^ (~Y v Z) ^ (~X v Y v ~Z)

Aplicando ao problema SAT, ao fazer os devidos cálculos necessário, concluímos que essa expressão booleana é TRUE, ou seja, a satisfabilidade é atingida.

Reduzir esse problema ao clique, consiste em demostrar que ao convertermos essa expressão a um grafo G(V,E), encontraremos uma instância TRUE pertencente ao clique para k<= |V|.

Assim, para gerarmos o grafo G(V,E) e k, temos:



Prova

Pela definição, φ ∈ 3SAT se somente se (G, k) ∈ a CLIQUE, sendo que o CLIQUE é definido como: CLIQUE = {(G, k) | G é um grafo não direcionado com k-clique}. Logo temos:

- Lema: se φ ∈ à 3SAT então (G, M) ∈ a CLIQUE
- Prova: Dado uma atribuição SAT A de φ, para cada cláusula C existe pelo menos um conjunto literal verdadeiro por A. Seja Vc o vértice em G correspondente ao primeiro literal de C satisfeito por A.
- Afirmação: S = {Vc : C ∈ φ} é um m-clique
- Prova: Se (Vc, Vc') ∉ E então Vc e Vc', deve rotular literais inconsistente, x ¬x. Portanto (Vc, Vc') ∈ E para todo Vc, Vc' ∈ S. Então, S é um m-clique e (G, m) ∈ CLIQUE.

satclique.c - https://github.com/pires28/FranciscoPiresJunior_AA_Lista2_rr_2019/blob/master/satclique.c

reference: Site : https://www.geeksforgeeks.org/2-satisfiability-2-sat-problem/

Referências

Site 1: http://theory.stanford.edu/~trevisan/cs154-12/np-reductions.pdf

Site 2: http://www.cs.nthu.edu.tw/~wkhon/toc07-lectures/lecture21.pdf

Site 3: https://www.youtube.com/watch?v=u3xNH6a1oa0

Site 4: https://www.geeksforgeeks.org/2-satisfiability-2-sat-problem/