Lista 2 - Victor Rocha

[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

Pode ser encontrado com a função **recursiva com complexidade de 2^n**:

```
fib(n)
        if(n >=1)
                return 1
        else
                fib(n-1) + fib(n-2)
```

Cuja análise assintótica gera a equação T(n) = T(n-1) + T(n-2) + c (uma constante).

O limite inferior pode ser calculado aproximando o valor de F(n-1) a F(n-2).

T(n) = T(n-1) + T(n-2) + c
= 2T(n-2) + c
= 2*(2T(n-4) + c) + c
= 4T(n-4) + 3c
= 8T(n-8) + 7c
= 2^k * T(n - 2k) + (2^k - 1)*c

n - 2k = 0
k = n/2

T(n) = 2^(n/2) * T(0) + (2^(n/2) - 1)*c
= 2^(n/2) * (1 + c) - c

T(n) ~ 2^(n/2) Complexidade O(2^(n/2))

Na sua forma **iterativa:**

```
fibo(int n)
 if(n <= 1){
        return n;
 int fibo = 1;
 int fiboPrev = 1;
 for(int i = 2; i < n; ++i)
        int temp = fibo;
        fibo += fiboPrev;
        fiboPrev = temp;
 return fibo;
```

Somando temos 3*n + 3, o que gera uma complexidade de O(n)

(B) Geração de todas as permutações de um número.

O escolhido foi o Heap, pois ele tenta reduzir o número de computações ao trocar somente dois elementos de cada vez e garante chegar a todas as permutações. Ele começa definindo um contador i = 0 e vai iterar sobre até gerar (n - 1)! permutações que terminam com cada número da sequência, se o valor de n for ímpar trocamos o primeiro termo com o último e se for par trocamos o i-ésimo termo com o último.

O pseudo-código: disponível em https://en.wikipedia.org/wiki/Heap%27s_algorithm

```
procedure generate(n : integer, A : array of any):
   if n = 1 then
       output(A)                          1
   else
      for i := 0; i < n - 1; i += 1 do    n -1 vezes
         generate(n - 1, A)               T(n-1)
         if n is even then
            swap(A[i], A[n-1])            1
         else
            swap(A[0], A[n-1])            1
         end if
      end for
   end if
```

if n = 1 { T(n) = 1 }
if n > 1 { T(n) = n*T(n-1) + 2 } Cuja complexidade é O(n!)

Versão iterativa:

```
procedure generate(n : integer, A : array of any):
   c : array of int

   for i := 0; i < n; i += 1 do       n vezes
      c[i] := 0                       1
   end for
   output(A)
   i := 0;                            1
   while i < n do                     n vezes
      if  c[i] < i then               1
         if i is even then            1
            swap(A[0], A[i])          1
         else
            swap(A[c[i]], A[i])
         end if
         output(A)

         c[i] += 1                    1
```

```
        i := 0                  1
      else
        c[i] := 0               1
        i += 1                  1
      end if
    end while
```

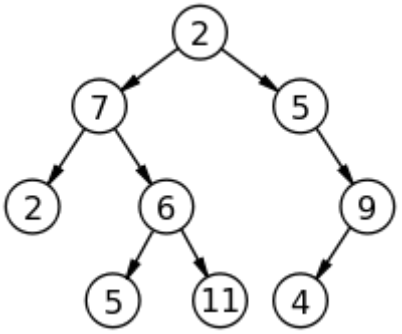
Complexidade de n*1 + n * 4 (levando em consideração if == verdadeiros)
O(n)

[QUESTÃO – 02]
Defina e dê exemplos:
(A) Grafos.
Grafos são estruturas que mostram a relação entre elementos de um mesmo conjunto. Descritos da maneira mais básica como G = (V,E), em que V representa os vértices, os elementos dos conjuntos, e E representa as aresta, as relações entre esses elementos.
Exemplo: Estações de trens, rede de roteadores, Pontos de ônibus.

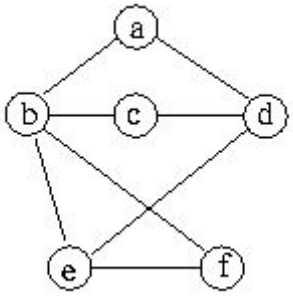
(B) Grafo conexo, acíclico e direcionado.
Um grafo é:
Conexo quando é possível sair de qualquer vértice e chegar a qualquer outro vértice do grafo, tipo em uma Árvore binária.

Acíclico quando não tem um caminho que começa e termina no mesmo vértice, ou seja, um ciclo. Também visto em uma Árvore.

Direcionado, também chamado de dígrafo ou quiver, é aquele cujo as arestas possuem propriedades que indicam de onde elas começam e terminam.



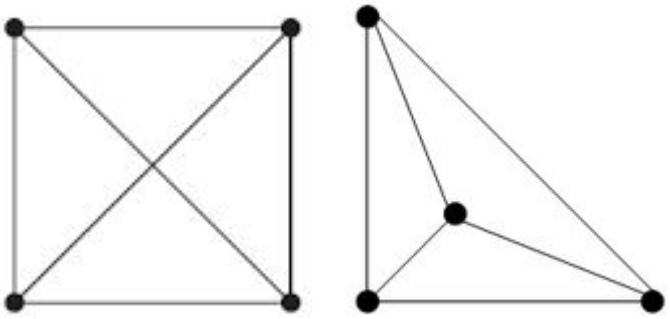
(C) Adjacência x Vizinhança em grafos.



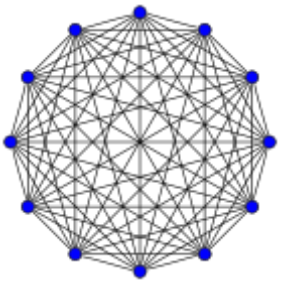
Adjacência em grafos exprime que dois vértices são conectados por uma aresta.
Exemplo: a é adjacente à b e d.

Vizinhança de um vértice é o conjunto de todos os outros vértices adjacentes a ele.
Exemplo: a vizinhança de b é {a, c, e}

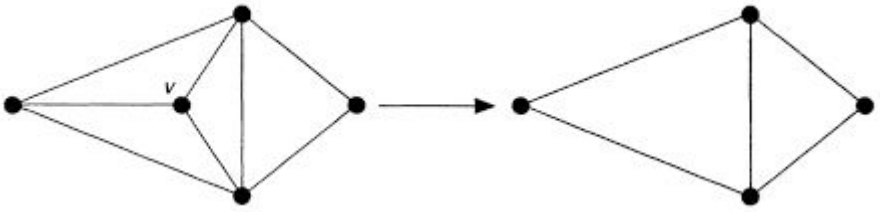
(D) Grafo planar.
São grafos que podem ser representados num plano sem que haja sobreposição de arestas. Não é possível representar grafos completos com número de vértices maior que 4 de forma planar.
Exemplo: Grafo (4, 6) em forma não planar e planar.



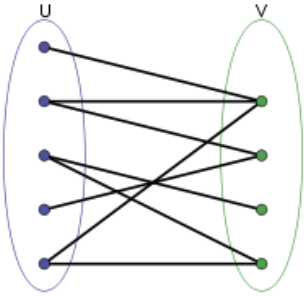
(F) Grafo completo, clique e grafo bipartido.
Um grafo é:
Completo quando todos vértices possuem arestas para os demais vértices. Todos os vértices possuem o mesmo grau.



Clique é um subgrafo completo dentro de um grafo.



Bipartido quando os vértices puderem ser divididos em dois grupo (U, V). Não possuem ciclos com números ímpar de vértices e podem ser coloridos com somente duas cores.



(G) Grafos simples x multigrafo x dígrafo.

Um grafo é:

Simples se não é direcionado, não possui laços e no máximo uma aresta entre dois vértices.

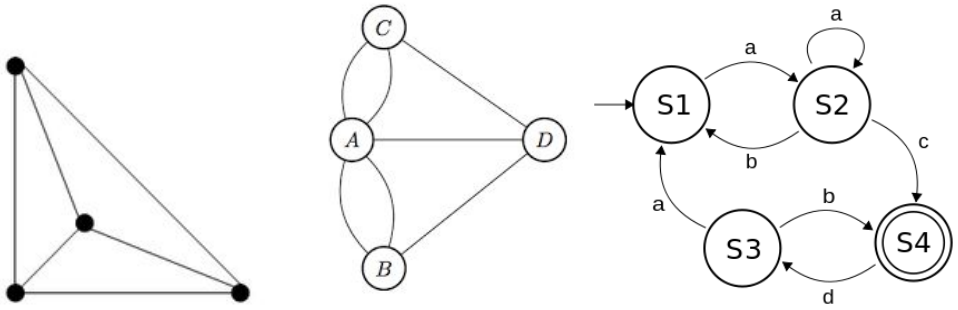
Exemplo: A

Multigrafo quando há mais de uma aresta ligando os mesmos dois vértices.

Exemplo: B

Dígrafo quando suas arestas são orientadas, exemplo nas máquinas de estados finitos.

Exemplo: C



(A) (B) (C)

[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 incidência

A matriz de incidência utiliza uma matriz m x n, onde n é o número de vértices e m é o número de arestas. Para essas matrizes, os vértices são as linhas e as arestas são as colunas e cada elemento da matriz indica se aresta incide sobre o vértice. Consegue proporcionar um acesso constante, porém a utilização de matrizes demanda muito espaço.

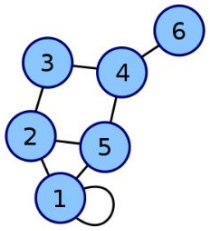
	a	b	c	d	e	f
1	1	1	0	0	0	0
2	1	0	1	0	0	1
3	0	1	1	1	0	0
4	0	0	0	1	1	1

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Matriz de adjacência:

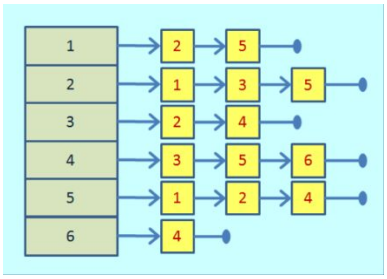
A matriz de adjacência de um grafo com |V| vértices é uma matriz |V|x|V| de 0s e 1s, na qual a entrada na linha i e coluna j é 1 se e somente se a aresta (i,j)(i,j) estiver no grafo. Uma vantagem da utilização de matriz de adjacência está no tempo constante de acesso (O(n)), e uma das desvantagens é a grande necessidade de espaço (Θ(V^2)).

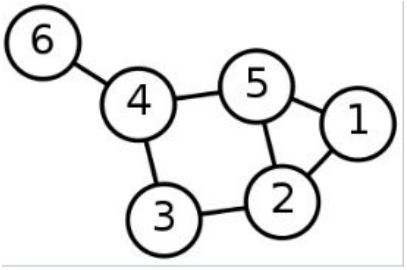
$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$



Lista de adjacência:

A representação de um grafo com listas de adjacência combina as matrizes de adjacência com as listas de arestas. Para cada vértice do grafo, existe uma lista encadeada que armazena os vértices adjacentes. Uma vantagem é a economia de espaço, porém existe um aumento no tempo de acesso.





[QUESTÃO – 04]

Defina, explicando as principais características e exemplifique:

(A) Enumeração explícita x implícita.

O método de enumeração explícita, faz uma enumeração de todas as possíveis soluções, já a enumeração implícita, os resultados são filtrados por limites superiores e inferiores gerando resultados mais significativos.

(B) Programação Dinâmica.

É quando resolvemos problemas que vão ter operações recorrentes na sua computação então resolvemos cada subproblema uma vez e guardamos as soluções dessas operações para não gerar gastos redundantes de poder computacional.
Exemplo, Problema da partição: para checar se dada uma sequência de números é possível dividi-la em duas com a mesma soma.

(C) Algoritmo Guloso.

Sempre fazer a melhor escolha local para ter uma melhor escolha global, geralmente atingindo soluções bem próximas da ótima, porém em alguns casos não encontra soluções tão ótimas. Um exemplo de utilização de algoritmos gulosos é encontrar uma solução do problema da mochila fracionária, no qual queremos encher uma mochila com objetos com pesos e valores diferentes no intuito de maximizar o valor.

(D) Backtracking.
É uma técnica usada por algoritmos para resolver problemas, geralmente de restrição de satisfabilidade, que utiliza a ideia de tentar um caminho de resolução possível até que esse caminho se mostre inviável de gerar uma resolução satisfazível, nesse caso ele retorna, "backtracks", para uma estado anterior e tenta outro caminho. Um exemplo clássico sendo o das 8 rainhas no qual queremos colocar 8 rainhas no tabuleiro sem que elas se matem.

[QUESTÃO – 06]
Defina e exemplifique:

(A) Problema SAT x Teoria da NP-Completude.
Problemas SAT ou Problemas de satisfabilidade booleana, é o primeiro problema classificado com a complexidade NP-Completo, estar nessa classe de complexidade diz duas coisas sempre, primeiro, que ele está na classe NP e segundo, todos os outros problemas na classe NP poderiam ser reduzidos à esse problema em tempo polinomial, ao ponto que se for possível resolvê-lo, seria possível resolver todos os outros problemas nessa classe em tempo polinomialmente definido.
O problema de Satisfabilidade diz que, dado uma sequência de elementos booleanos (ex.: A ∧ ¬ B) procura-se saber se existem valores possíveis para substituir as variáveis de modo a gerar uma solução verdadeira (TRUE), nesse caso ela é satisfatória. Caso não exista tal combinação, ela retorna falso (FALSE) e é insatisfatória.
Esse problema poderia ser pensando na forma de um grafo se adaptássemos as sentenças para uma forma que exprima implicância, exemplo (a v b) pode escrito como (~a -> b v ~b -> a). Daqui podemos escrever toda uma sentença em forma de grafo e fazer uma pesquisa nesse grafo para achar

uma inconsistência, como ~b -> b, que classificaria o problema como insatisfatório e, caso contrário, satisfatório.

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

A classe P de complexidade abrange problemas cujo as soluções podem ser encontradas em tempo polinomial determinístico.
Exemplo, o problema de determinar se um número é primo.

NP (polinomial não determinista) é a classe de problemas cuja solução pode ser verificada a partir de uma entrada e retornar um "sim" ou "não" em um tempo limitado por um polinômio, assim chamados problemas polinomialmente verificáveis.
Exemplo, o problema do isomorfismo de subgrafos: dados dois grafos A e B como input, quer-se determinar se o B é subgrafo de A.

Também dentro de NP existe uma classe chamada NP-Completo cujos elementos atendem a duas condições:

- estar dentro da classe NP, como já dito.
- todo problema em NP é redutível para esse problema em tempo polinomial. De modo que se conseguíssemos achar uma solução em tempo polinomial para esse problema, poderíamos resolver todos os outros problemas em NP em tempo polinomial.

O próprio problema de satisfabilidade, assim como o problema de Coloração de grafos que se propõe a descobrir como colorir os vértices de um grafo baseado em alguma restrição (geralmente que duas cores não podem estar justapostas).

NP-Difícil é a classe dos problemas "tão difíceis quanto NP-completo" por que mesmo não estando na classe NP e mesmo sem nem precisar ser problemas de decisão, estes problemas são reduzíveis a problemas NP.
Exemplo, o problema da parada que é um problema de decisão que pergunta se dado um programa ele vai terminar ou rodar para sempre.

[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.

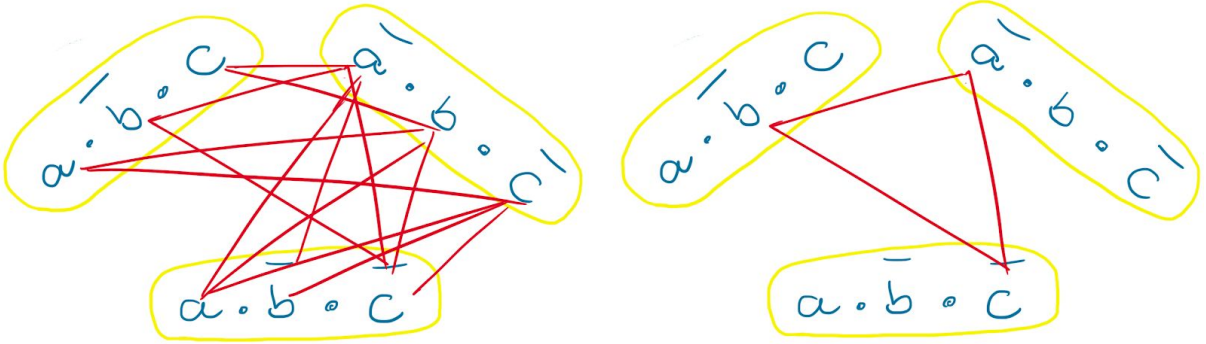
Para provar a NP-completude de um problema é preciso provar que esse problema está em NP, ou seja, gastaria tempo polinomial para checar uma possível solução, e se todo problema em NP é redutível para tal problema em tempo polinomial.
No caso ao reduzir SAT em um problema de clique, estaríamos provando que achar uma solução Y para SAT estaria na mesma ordem de dificuldade do que achar uma solução X para o Clique.
Suponhamos que temos a fórmula booleana a qual queremos provar sua satisfabilidade.

$$(a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee \neg b \vee \neg c)$$

Poderíamos transformar um grafo bidirecionado que atendesse a restrição de não possuir arestas que causem contradição, por exemplo "a" ligando com¬a.

SAT ⟶ Clique

$$(a \vee \bar{b} \vee c) \wedge (\bar{a} \vee b \vee \bar{c}) \wedge (a \vee \bar{b} \vee \bar{c})$$



$$(a \vee \bar{b} \vee c) \wedge (\bar{a} \vee b \vee \bar{c}) \wedge (a \vee \bar{b} \vee \bar{c})$$

a = 0 0 ⌟ 0 ⌟ 0 ⌟ 0 ⌟ ⌟
b = 0 ⌟ ∧ ⌟ ∧ ⌟ = Satisfeitz
c = 0