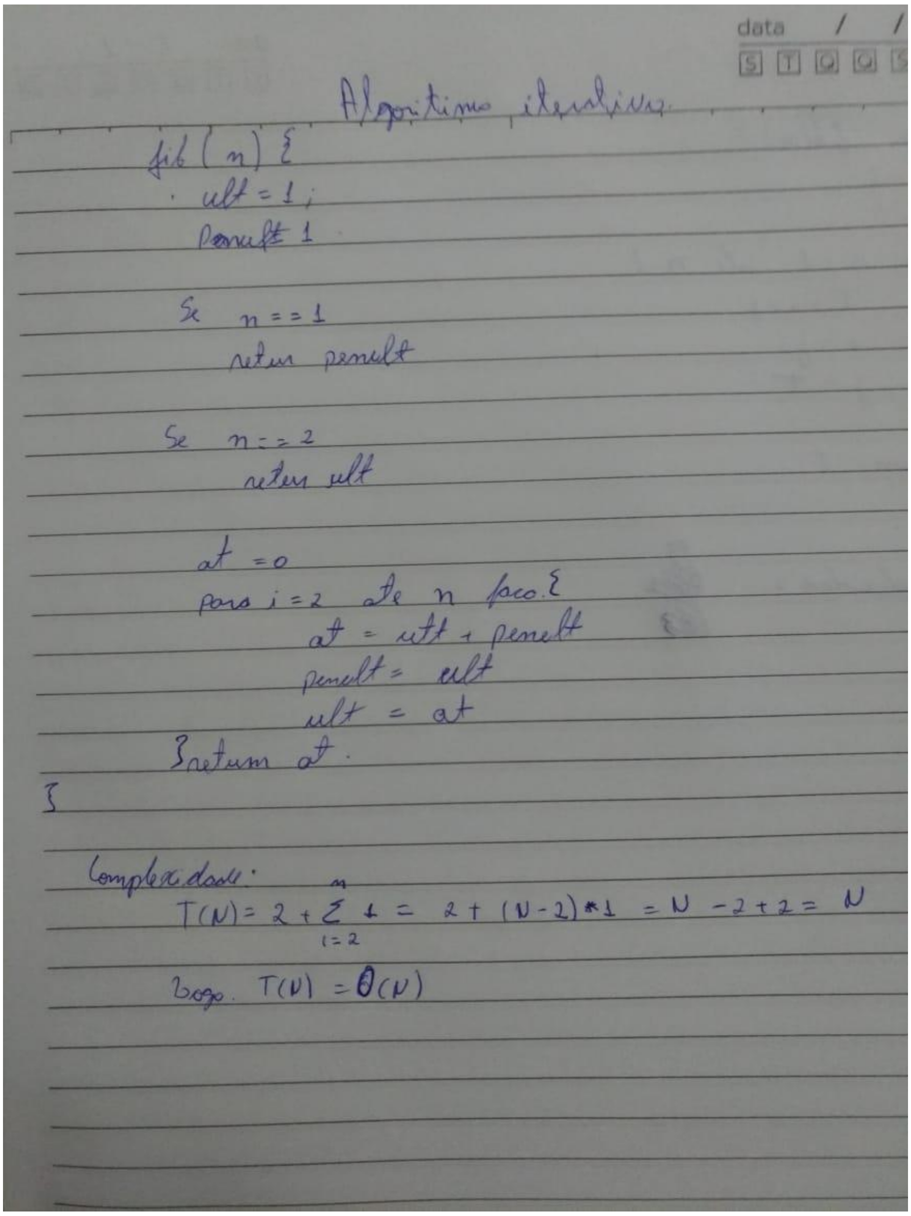
Universidade Federal de Roraima – UFRR
Aluno: Paulo Fábio dos Santos Ramos
Disciplina: Análise de Algoritmos
Professor: Herbert Oliveira Rocha
Data: 18/06/2019

2ª Lista de Exercícios

1) a) O objetivo do algoritmo é encontrar o n-ésimo termo da sequência de Fibonacci portanto não existe relação entre melhor caso, pior caso ou caso médio, sendo possível escrever o algoritmo tanto de maneira recursiva como iterativa. Segue abaixo a analise da complexidade feita para o algoritmo iterativo.



Algoritmo iterativo

```
fib (n) {
    ult = 1;
    penult 1

    Se  n == 1
        retun penult

    Se  n == 2
        retun ult

    at = 0
    para i = 2 até n faça {
        at = ult + penult
        penult = ult
        ult = at
    } return at.
}
```

Complexidade:

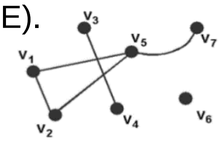
$T(N) = 2 + \sum_{i=2}^{n} 1 = 2 + (N-2)*1 = N - 2 + 2 = N$

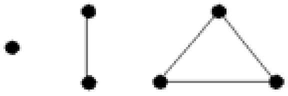
Logo. $T(N) = \Theta(N)$

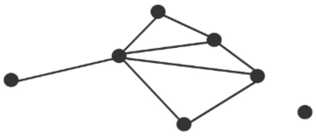
Onde podemos ver que tem sua complexidade dada pro O(n).

2) a) **Grafo:** É um modelo matemático que representa relações entre objetos, consiste de um conjunto de vértices V, ligados por um conjuntos de arestas ou arcos G = (V, E).

Exemplos.

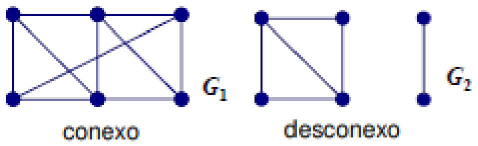


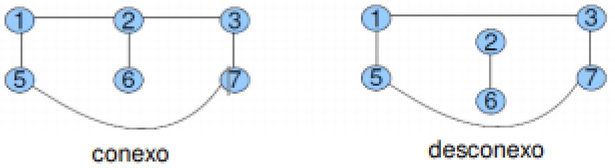




b) **Conexo:** Um grafo é conexo se existir um caminho entre qualquer par de vértices, se existir pelo menos 1 par de vértices que não está ligado a nenhum caminho, ele é desconexo.

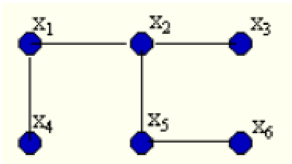
Exemplos.

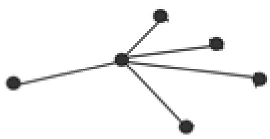




Acíclico: É considerado acíclico quando não possuir ciclos. Uma arvore é um grafo acíclico conexo.

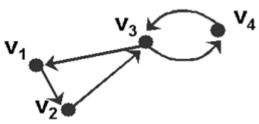
Exemplos.

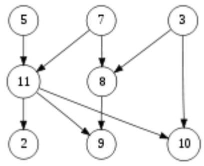




Direcionado: Cada aresta possui uma única direção de x para y.

Exemplos:

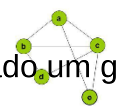




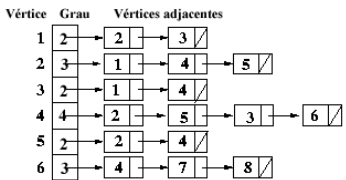
c) A vizinhança de um vértice V consiste em todos os vértices que são adjacentes a V, e pode ser representado tanto pela matriz de adjacência como pela lista de adjacência.

Matriz de adjacência: Dado um grafo G, a matriz de adjacências M é uma matriz n x n tal que:

n = número de vértices e M[i,j] = 1, se existir aresta de i a j e M[i,j] = 0, se não existir aresta de i a j.

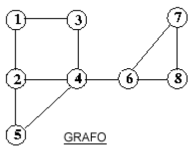


Lista de adjacência: Dado um grafo G, a estrutura de adjacência é um vetor de n elementos que são capazes de apontar, cada um, para uma lista linear. O



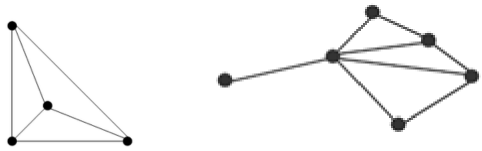
iesimo elemento do vetor aponta para a lista linear das arestas que incidem no vértice i.

Exemplo.



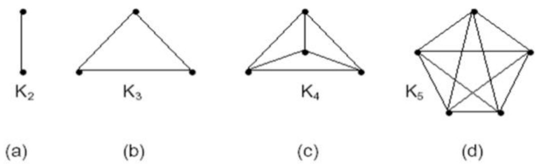
D) **Grafo planar:** É considerado um grafo planar, aqueles que são submetidos no plano de tal forma que suas arestas não se cruzem.

Exemplo.



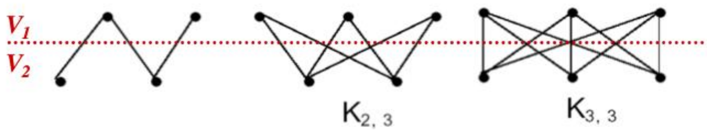
F) **Grafo completo:** Um grafo é completo se todos os seus vértices forem adjacentes.

Exemplos.



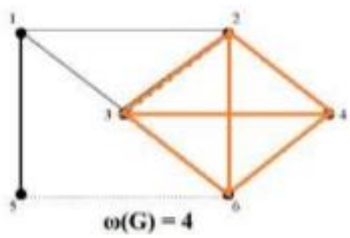
Grafo bipartido: Um grafo G = (V,E) é bipartido quando o seu conjunto de vertices V pode ser dividido em dois subconjuntos V1 e V2 tais quais todas as arestas do cojunto E une um vertice de V1 a outro vertice de V2.

Exemplo.



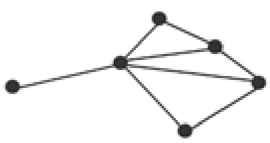
Clique: Em um grafo não-dirigido, um clique é um conjunto de vértices dois a dois adjacentes, ou seja, se tiver a seguinte propriedade: para todo par (v,w) de vértices distintos em C, existe uma aresta com pontas (v e w).

Exemplo.

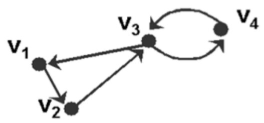


G)**Grafo simples.** Um grafo simples é um grafo que não possui arestas múltiplas.

Exemplo.

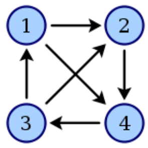


Multigrafo. Quando possui mais de uma aresta interligando os mesmos dois vertces.



Digrafo: O mesmo que um grafo direcionado, cada aresta possui uma única direção de x para y.

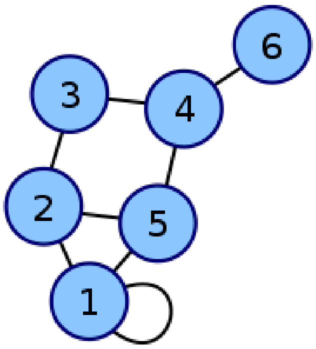
Exemplo.



3) Uma **matriz de incidência** representa computacionalmente um grafo através de uma matriz bidimensional, onde uma das dimensões são vértices e a outra dimensão são arestas. Para representar um grafo sem pesos nas arestas e não direcionado, basta que as entradas da matriz M contenham 1 se a aresta incide no vértice, 2 caso seja um laço (incide duas vezes) e 0 caso a aresta não incida no vértice.

Exemplo.

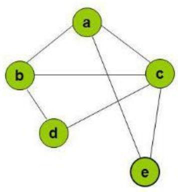
$$\begin{bmatrix} 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$



Matriz de adjacência: Dado um grafo G, a matriz de adjacências M é uma matriz n x n tal que:

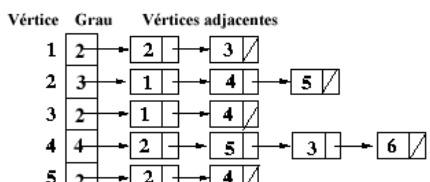
n = número de vértices e M[i,j] = 1, se existir aresta de i a j e M[i,j] = 0, se não existir aresta de i a j.

Exemplo.



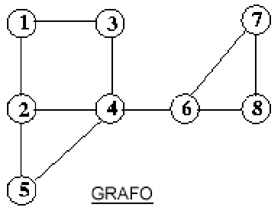
	a	b	c	d	e
a	0	1	1	0	1
b	1	0	1	1	0
c	1	1	0	1	1
d	0	1	1	0	0
e	1	0	1	0	0

Lista de adjacência: Dado um grafo G, a estrutura de adjacência é um vetor de n elementos que são capazes de apontar, cada um, para uma lista linear. O



iesimo elemento do vetor aponta para a lista linear das arestas que incidem no vértice i.

Exemplo.



4) a) **Enumeração explicita**: é a famosa resolução por força bruta, onde é feita todas as comparações possíveis em um conjunto de dados para se obter a resposta desejada.

Exemplo: algoritmos de força bruta.

Enumeração implícita: Quando apenas uma parte dos dados é realmente analisada, sem necessidade de se analisar todos os casos possíveis.

Exemplo: Algoritmos gulosos.

b) **Programação dinâmica** é um método para a construção de algoritmos para a resolução de problemas computacionais, em especial os de otimização combinatória. Ela é aplicável a problemas nos quais a solução ótima pode ser computada a partir da solução ótima previamente calculada e memorizada - de forma a evitar recálculo - de outros subproblemas que, sobrepostos, compõem o problema original.

Exemplos: algoritmo de Dijkstra, algoritmo para o problema da mochila booleana.

b) **Algoritmo guloso,** toma decisões com base nas informações disponíveis na iteração corrente, sem olhar as consequências que essas decisões terão no futuro. Um algoritmo guloso jamais se arrepende ou volta atrás: as escolhas que faz em cada iteração são definitivas, ou seja, ele sempre busca a melhor solução que aparecer no momento, sem ver onde isso pode dar.

Exemplos: problema da mochila fracionaria, problema do escalonamento de intervalos

c) **Backtracking** é um algoritmo que busca, por força bruta, soluções possíveis para problemas computacionais. Ele busca por candidatos à soluções e abandona cada candidato parcial C quando C não pode resultar em uma solução válida. Quando sua busca chega a uma extremidade da estrutura de dados, como um nó terminal de uma árvore, o algoritmo realiza um retrocesso tipicamente implementado através de uma recursão.

Exemplos: N-rainhas, Caixeiro Viajante.

5) Pseudocódigo da **multiplicação de matrizes** usando programação dinâmica

MATRIXCHAINORDER (p,n)

```
para i ← 1 até n faça
    m[i,i] ← 0
para l ← 2 até n faça
     para i ← 1 até n − l + 1 faça
          j ← i + l − 1
          m[i, j] ← ∞
          para k ← i até j − 1 faça
               q ← m[i, k] + p[i − 1] p[k]p[j] + m[k+1, j]
               se q < m[i, j]
                    então m[i, j] ← q
devolva m[1, n]
```

6) a) **Problema SAT**: é o problema de determinar se existe uma determinada valoração para as variáveis de uma determinada fórmula booleana tal que esta valoração satisfaça esta fórmula em questão. Por exemplo, tomando x1, x2, x3, x4 como as variáveis booleanas e a expressão $(X1 \vee \neg X3 \vee X4) \wedge (\neg X2 \vee X3 \vee \neg X4)$ caso exista uma atribuição de valores de verdade para as variáveis da fórmula que torne a fórmula avaliada VERDADEIRA, esta fórmula é considera satisfazível, em contrapartida se nenhuma atribuição levou a uma avaliação da fórmula como verdadeira, ela é considerada insatisfazível.

b)A classe **P** é a classe de todos os problemas de busca que são resolvidos em tempo polinomial.(2sat, arvore geradora minima, mochila fracionaria..)

A classe **NP** é a classe de todos os problemas de busca, seja ele resolvido em tempo polinomial ou não.

A classe **NP-Dificil** é um subconjunto de NP, que são informalmente definidos como “Pelo menos tão dificeis quanto os problemas mais dificeis em NP”. Um

problema H é NP-difícil se e somente se existe um problema NP-completo L que é Turing-redutível em tempo polinomial para H.
NP-Completo: Um problema de decisão X é completo em NP (ou NP-completo) se X está em NP e qualquer outro problema em NP pode ser polinomialmente reduzido a X.

7) Uma redução (ou redução polinomial) é reduzir um problema x em um problema y onde um algoritmo 1 que resolve x usando uma subrotina hipotética algoritmo 2 que resolve y, tal que, se algoritmo 2 é um algoritmo polinomial, então algoritmo 1 é um algoritmo polinomial também.
A notação: x <=p y. Significa que existe uma redução de x a y.
Se x <=p y e y está em P, então x está em P.
Para mostrar que SAT <=p Clique primeiro mostraremos que SAT <=p 3-SAT e que 3-SAT <=p Clique.

SAT <=p 3-SAT
Quando se diz 3 literais por clausula significa isso:
Ø = (x1 v !x1 v !x2) ^ (x3 v x2 v x4)
Um literal são as variáveis x1,x2.. e sua negação é !x1,!x2...
Uma clausula é (x1 v !x1 v !x2)...
Descreveremos um algoritmo polinomial T que recebe uma fórmula booleana Ø e devolve uma fórmula booleana Ø’ com exatamente 3 literais por cláusulas tais que:
Ø é satisfazível se e somente se Ø’ é satisfazível

A transformação consiste em substituir cada clausula de Ø por uma coleção de cláusulas com exatamente 3 literais cada e equivalente a Ø;
Seja ($l_1 \vee l_2 \vee \cdots \vee l_k$) uma cláusula de Ø .
Caso 1. k = 1
Troque (l_1)
por ($l_1 \vee y_1 \vee y_2$) ($l_1 \vee \neg y_1 \vee y_2$) ($l_1 \vee y_1 \vee \neg y_2$) ($l_1 \vee \neg y_1 \vee \neg y_2$) onde y_1 e y_2 são variáveis novas.
Caso 2. k = 2

Troque ($l1 \vee l2$) por ($l1 \vee l2 \vee y$) ($l1 \vee l2 \vee \neg y$). onde y é uma variáveis nova.

Caso 3. $k = 3$

Mantenha ($l1 \vee l2 \vee l3$).

Caso 4. $k > 3$

Troque ($l_1 \vee l_2 \vee \cdots \vee l_k$) por

($l_1 \vee l_2 \vee y_1$)

($\neg y_1 \vee l_3 \vee y_2$) ($\neg y_2 \vee l_4 \vee y_3$) ($\neg y_3 \vee l_5 \vee y_4$) . . .

($\neg y_{k-3} \vee l_{k-1} \vee l_k$)

onde $y_1, y_2, \ldots, y_{k-3}$ são variáveis novas

Verifique que Ø é satisfazível se e somente se nova fórmula é satisfazível. O tamanho da nova cláusula é O(m), onde m é o número de literais que ocorrem em Ø (contando-se as repetições).

Agora para 3-SAT <=p Clique

Descreveremos um algoritmo polinomial T que recebe uma fórmula booleana Ø com k clausulas e exatamente 3 literais por clausula e devolve um grafo G tais que

Ø é satisfativel se e somente se G possui um clique >= k

Para cada clausula o grafo G terá 3 vértices, um correspondente a cada literal da clausula, Logo G terá 3k vértices. Teremos arestas ligando vértices u e v se u e v são vértices que correspondem a literais em diferentes clausulas;

e se u corresponde a um literal x então v não corresponde ao literal !x.