Universidade Federal de Roraima
Centro de Ciência e Tecnologia
Departamento de Ciência da Computação

Lista 02 de Análise de Algoritmo
Aluno: William Juan da Silva Melo
Professor: Herbert Rocha
2019

[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 sequência de Fibonacci.

Resposta: é uma sequência de números que começa com 0 e 1 e os números seguintes são a soma dos dois anteriores. Veja abaixo:

0, 1, 2, 3, 5, 8, 13, …

O código recursivo em C é este abaixo:

```
Int fibonacci (int n) {
    if ((n == 1) || (n == 0)) {
        return n;
    }
    if (n == 0) {
        return 0;
    }
    return fibonacci (n - 1) + fibonacci (n - 2);
}
```

Temos uma complexidade que é dada por $O(\phi n)$ onde : ϕ é igual a $\frac{1+\sqrt{5}}{2}$. Nos baseando nisso o melhor caso e o pior caso equivalem a 2^n .

Já o código iterativo é:

```
int fibonacci (int n) {
	int	f, f1, f2;
	if ((n == 1) || (n == 0)) {
		return n;
	}
	else {
		f1 = 0; f2 = 1;
		for (int I = 2; I <= n; I++) {
			f = f1 + f2;
			f1 = f2;
			f2 = f;
		}
		return f;
	}
	return 0;
}
```

Aqui a complexidade é O(n). Com base nisso o pior e o melhor caso equivalem a n.

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

Permutação significa troca, isso porque uma lista de valores, as permutações são todas as possibilidades em que podemos organizar estes números. Exemplo:

Lista 1, 2, 3

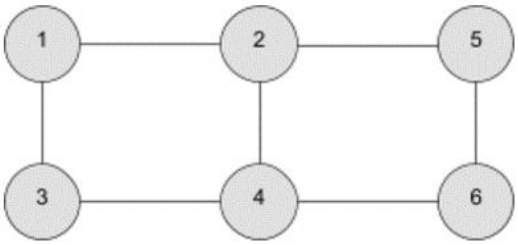
1, 2, 3
1, 3, 2
2, 1, 3
2, 3, 1
3, 1, 2
3, 2, 1

Aqui temos 6 possibilidades. Se observarmos melhor veremos que isso é um fatorial, pois cada vez que colocamos o valor em uma casa, diminuamos em 1 o que a quantidade de valores da próxima casa a ser preenchida. Logo temos que a complexidade é n!

[QUESTÃO – 02]
Define e dê exemplos de:

a) Grafos:

Grafo é uma estrutura baseada em vértices e arestas (V, A), onde as arestas são ligações entre os vértices.

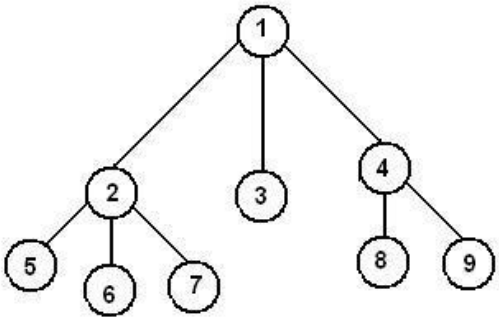


Grafo

b) Grafo conexo, acíclico e direcionado

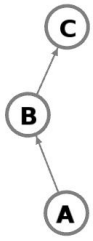
O grafo conexo é quando há ao menos um caminho entre qualquer vértice para todos os outros vértices.

O grafo acíclico é o grafo em que os caminhos entre os vértices não contém vértices repetidos.



Grafo acíclico

O grafo direcionado é um grafo onde as arestas possuem sentido, e por isso não podemos percorrer pelas arestas no sentido contrário.



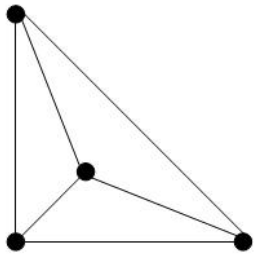
Grafo direcionado

c) Adjacência x Vizinhança em grafos.

Adjacência ou vizinhança são a mesma coisa, é um sub grafo formado por todos os vértices ligados diretamente a um vértice do grafo, ou seja, , um vértice v em um grafo G é um sub grafo induzido de G constituído por todos os vértices adjacentes a v e todas as arestas ligando esses dois vértices. Resumindo, são os vértices diretamente conectados a v.

d) Grafo planar

É o grafo que pode ser desenhado no plano sem que haja arestas se cruzando.



Grafo planar

e) Grafo completo, clique e grafo bipartido.

O grafo completo é um grafo em que cada par de vértices é conectado por uma aresta.

O clique é um conjunto de vértices V tal que, para cada par de vértices de V, existe uma aresta que os conecta, ou seja, é sub grafo completo dentro de um grafo.

Um grafo bipartido é um grafo não dirigido G = (V , E) no qual V pode ser particionado em dois conjuntos V1 e V2 tais que (u,v) ϵ E. Isto é, todas as arestas ficam entre os dois conjuntos V1 e V2.

f) Grafos simples x multigrafo x dígrafo:

O grafo simples é um grafo que não contém laços nem arestas múltiplas.

O mutigrafo os vértices possuem arestas múltiplas.

O dígrafo é um grafo direcionado e simples.

[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: 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 .

2	1	0	1
1	0	0	1
1	1	1	0
0	1	0	1

Matriz de adjacência: Uma matriz de adjacência é uma das formas de se representar um grafo. Se o vertices i esta ligado ao vertice j então na matriz ij recebe 1 caso contrario recebe 0 para indicar que não contem relação.

1	0	0	1
0	1	1	0
0	0	0	1
1	1	1	1

lista de adjacência: é a representação de todas arestas ou arcos de um grafo em uma lista, onde as setas indicam as ligações e null significa fim da lista daquele vertice.

[1] → 2 → 3 → null
[2]→ null
[3] → 4 → 5 → null
[4] → null
[5] → 1 → null
[6] → 2 → null

Algumas vantagens e desvantagens da matriz adjacência e lista de adjacência estão:

- Na matriz de adjacência a velocidade de leitura e escritura é constante.
- A lista de adjacência ocupa menos espaço na memória que a matriz de adjacência.
- Quando a quantidade de vértices é igual ao número de arestas, utilizar a matriz de adjacência é mais viável.
- Quando o número de arestas á bem inferior ao número de vértices pode ser melhor utilizar lista de adjacência.

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

A) Enumeração explícita x implícita:

Uma enumeração explicita é quando vamos dizer criamos uma enumeração dos meses do ano e explicitamos quantos dias cada mês tem. Já na enumeração implícita vamos dizer que criamos uma enumeração dos dias zero , um, dois ,… ele receberá implicitamente os valores por esta em uma sequencia crescente.

B) Programação Dinâmica:

Programação dinâmica é uma técnica de programação onde salva-se os resultados calculados a fim de não precisar recalcular contas já calculadas, com isso somente precisa pegar o resultado caso precise. Ou seja, se aplica quando os subproblemas se sobrepõem, isto é, quando os subproblemas compartilham sub-subproblema. Exemplo fibonacci usando pd.

C) Algoritmo Guloso:

Um algoritmo guloso é aquele que faz escolhas locais ótimas, a fim de encontrar um resultado. Mesmo não sendo a melhor escolha global. Exemplo busca gulosa em grafo ou árvore.

D) Backtracking:

Backtracking é uma técnica onde se o algoritmo perceber que não tem mais para onde ele volta os passos. Exemplo na busca em profundidade se ele chegar no final da subárvore ele volta fazendo backtracking.

[QUESTÃO – 06]
Defina e exemplifique:

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

Problema SAT: O problema de satisfatibilidade booleana é um problema de decisão, cuja instância é uma expressão booleana escrita somente com operadores AND, OR, NOT, variáveis, e parênteses. A questão é: dada uma expressão, será que há alguma atribuição de valores VERDADEIROS e FALSOS para as variáveis que torne a expressão inteira verdadeira? Uma fórmula da lógica proposicional é dita satisfazível-ou seja, avaliada como VERDADEIRA se for possível atribuir valores lógicos a suas variáveis de tal maneira que eles tornem a fórmula verdadeira. A classe de fórmulas satisfatíveis proposicionais é NP-completa. O Problema de satisfazibilidade Proposicional (SAT), que decide se uma dada fórmula proposicional é satisfazível, é de fundamental importância em várias áreas da ciência da computação, incluindo a Teoria da Computação, algoritmos, inteligência artificial, projeto de hardware e verificação.

NP-Completude: NP-completo é um subconjunto de NP, o conjunto de todos os problemas de decisão os quais suas soluções podem ser verificadas em tempo polinomial; NP pode ser equivalentemente definida como o conjunto de problemas de decisão que podem ser solucionados em tempo polinomial em uma Máquina de Turing não determinística. Um problema p em NP também está em NPC Se e somente se todos os outros problemas em NP podem ser transformados em p em tempo polinomial.

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

P: Na teoria da complexidade computacional, P é o acrônimo em inglês para Tempo polinomial determinístico (Deterministic Polynomial time) que denota o conjunto de problemas que podem ser resolvidos em tempo polinomial por uma máquina de Turing determinística. Qualquer problema deste conjunto pode ser resolvido por um algoritmo com tempo de execução O(n^{k}), (com k constante). Podemos ter a classe P como a classe de problemas que são solúveis para um computador real.

NP: Na teoria da complexidade computacional, NP é o acrônimo em inglês para Tempo polinomial não determinístico (Non-Deterministic Polynomial time) que denota o conjunto de problemas que são decidíveis em tempo polinomial por uma máquina de Turing não-determinística. Uma definição equivalente é o conjunto de problemas de decisão que podem ter seu certificado verificado em tempo polinomial por uma máquina de Turing determinística.

NP-Difícil: 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 problema H é NP-difícil se e somente se existe um problema NP-completo L que é Turing-redutível em tempo polinomial para H. Em outras palavras, L pode ser resolvido em tempo polinomial por uma Máquina de Turing não determinística com um oráculo para H. Informalmente, podemos pensar em um algoritmo que pode chamar tal Máquina de Turing Não-Determinística como uma sub-rotina para resolver H, e resolver L em tempo polinomial, se a chamada da sub-rotina leva apenas um passo para computar. Problemas NP-difíceis podem ser de qualquer tipo: problemas de decisão, problemas de pesquisa ou problemas de otimização.

NP-Completo: NP-completo é um subconjunto de NP, o conjunto de todos os problemas de decisão os quais suas soluções podem ser verificadas em tempo polinomial; NP pode ser equivalentemente definida como o conjunto de problemas de decisão que podem ser solucionados em tempo polinomial em uma Máquina de Turing não determinística. Um problema p em NP também está em NPC Se e somente se todos os outros problemas em NP podem ser transformados em p em tempo polinomial.