Algoritmos são sequências de passos para se realizar uma tarefa, o exemplo mais famoso é uma receita de bolo, onde se tem o passo a passo de como preparar e fazer um bolo, da mesma forma na ciência da computação tem a sequência de passos que é escrito no código para se definir alguma ação a ser feita por um programa, podendo receber um valor de entrada e retornar um valor de saída.

Exemplos de problemas que envolvem algoritmos no dia a dia:

* Trocar uma lâmpada.
* Preparar um bolo.
* Organizar a lista de contatos por ordem alfabética.
* Calcular a rota mais curta entre duas ruas.
* Calcular a rota mais curta entre duas ruas

De forma geral, algoritmos são usados como ferramentas para decifrar um problema, as suas características são: finitas, bem definidas e efetivas.

As principais técnicas são:

* Algoritmo guloso
* Programação Dinâmica
* Dividir e Conquistar
* Busca e Ordenação
* Backtracking

Também são utilizadas as **Estrutura de Dados** juntamente com essas técnicas para ajudar numa melhor eficácia dos algoritmos. Como por exemplo: Grafos, Árvores, Heaps, Tabelas Hash, Pilhas e Filas.

**1.2 Eficiência dos Algoritmos**

Há vários métodos para se analisar a performance e eficácia de algum programa, os famosos “Profilers”. Contudo, embora tenha uma boa eficácia, eles não são tão conveniente para analisar a complexidade de algoritmos.A complexidade dos algoritmos avaliam um algoritmo em “nível de idealização/definição”, ou seja, desconsiderando qualquer implementação de linguagens específicas ou hardware.

Comparar o tempo que um algoritmo leva para executar determinada tarefa em milissegundos em relação a outro, não é características para dizer que um algoritmo é melhor do que o outro.

Pode-se dizer que o melhor algoritmo para decifrar um problema é aquele que tem a menor complexidade de tempo e espaço. Ou seja, é aquele algoritmo em que, de acordo com que a entrada cresce visando o infinito, é aquele que mostra o menor tempo de variação e a menor memória utilizada para terminar.

Um algoritmo é capaz de ser melhor do que outro quando processa pouco dados, mas é capaz de ser muito pior de acordo com que os dados crescem.

A**Análise de complexidade** nos permite medir o quão rápido um programa executa seus cálculos.

**1.3 Comportamento Assintótico**

Seria um trabalho muito cansativo ficar contando a quantidade de informação para cada trecho de código que fosse escrito. Além disso, a quantidade de informações varia muito de linguagem, compilador e até mesmo o hardware da máquina que está sendo utilizada.

Na **Análise de Complexidade** podemos apenas nos interessar pelo termo que mais cresce de acordo com a entrada. Para atingirmos esse termo, podemos remover todas as constantes e manter apenas o termo que mais cresce.

No método **f(n) = 6n + 4**, evidentemente, 4 continua com o mesmo valor independente da entrada, mas **6n** aumenta cada vez. Retirando ele, encontramo-nos com o método **f(n) = 6n**.

Visto que 6 é uma constante, conseguimos retirá-lo e obtermos o método **f(n) = n**. Dessa forma, facilita demais a análise da complexidade do algoritmo.

**1.4 Complexidade dos Algoritmos e Notação Big-O**

Para apresentar o comportamento assintótico de algum algoritmo, foi adotado a Notação Big-O pelo cientistas da computação. Ela é usada para demarcar assintoticamente o aumento(tempo ou espaço) superior do algoritmo.

Usando o algoritmo para encontrar o maior elemento como forma de demonstração, somos capazes de encontrar caso de entrada que fará com que ele execute um número menor de cálculos. Não será para todo caso de entrada que seu método para a quantidade de instruções seja **f(n)**.

Usando a notação Big-O, conseguimos dizer que a complexidade do algoritmo é **“Big-O de O(n)**”, ou seja, no pior caso cresce em ordem de **n.**

Em algoritmos simples, é fácil identificar a complexidade do mesmo. Normalmente, se o algoritmo tem apenas 1 laço de repeitção, sua complexiade é **O(n)**, se possui 2 laços de repetição encadeados **O(n²)** e se não possuir nenhum laço **O(1)**.

**1.5 Complexidade de espaço**

Toda os estudos feitos até o momento foram em função da quantidade de operações que os algoritmos pedem, que é semelhante à **Complexidade de Tempo**.

A complexidade de espaço de um algoritmo não é muito distinto da complexidade de tempo em questão de análise, que também é utilizado a notação **Big-O**.

Para conseguir analisar a complexidade de espaço de algum algoritmo, deve-se identificar a quantidade de memória que o algoritmo precisa alocar para resolver o problema no pior dos casos.

**2 Métodos de Ordenação**