# AED2 - Aula 01
# Apresentação, estruturas de dados, tabelas de símbolos e hash tables

“É esperado de um projetista de algoritmos que ele entenda o problema a resolver e compreenda as ferramentas a sua disposição, para assim tomar decisões embasadas de projeto”.

**Apresentação do curso**

Princípios de projeto de algoritmos e estrutura de dados, com ênfase no porquê das coisas.

O que é um algoritmo?

- É uma receita para resolver um problema.

Por que estudar algoritmos?

- São importantes para inúmeras área da computação, como roteamento de redes, criptografia, computação gráfica, bancos de dados, biologia computacional, inteligência artificial, otimização combinatória, etc.
- Relevantes para inovação tecnológica pois, para resolver um problema computacional normalmente existem diversas soluções viáveis, por vezes com características e desempenho muito dispare. Neste curso vamos apresentar ferramentas para que vocês possam desenvolver estas variadas soluções, bem como continuar a desenvolver suas capacidades para avaliar a qualidade das mesmas.
- Eles são interessantes, divertidos e desafiadores, pois o desenvolvimento de algoritmos mistura conhecimento técnico com criatividade.

Habilidades que serão desenvolvidas:

- Tornar-se um melhor programador.
- Melhorar habilidades analíticas.
- Aprender a pensar algoritmicamente.

Principais tópicos do curso:

- Tabelas de espalhamento (hash tables).
- Árvores de busca.
- Ordenação.
- Busca de palavras.
- Busca em grafos.

Esses tópicos serão permeados por noções de análise de corretude e eficiência de algoritmos.

Comparação com outras áreas:

- Literatura, pensem na diferença entre ser alfabetizado e ser capaz de escrever um romance.
- Construção civil, pensem na diferença entre projetar uma casa e projetar pontes, edifícios, estradas, portos.

Ler/estudar por conta:

- Leiaute - apêndice A do livro “Algoritmos em linguagem C” ou www.ime.usp.br/~pf/algoritmos/aulas/layout.html
- Documentação - capítulo 1 do livro “Algoritmos em linguagem C” ou www.ime.usp.br/~pf/algoritmos/aulas/docu.html

**Estruturas de dados**

Visão geral:

- usadas para organizar dados permitindo acesso rápido aos mesmos.
- não existe estrutura perfeita, cada uma é eficiente para algumas operações e ineficiente para outras.
- parcimônia, escolha a estrutura de dados mais simples que suporta todas as operações requisitadas pela sua aplicação.

Objetivos:

- conhecer uma variedade de estruturas de dados.
- entender os pontos fortes e fracos de cada uma, permitindo escolher onde utilizá-las.
- saber como implementar e modificar as estruturas de dados para atender a necessidades específicas que surjam em suas aplicações.

Motivação para escolha de certa estrutura de dados:

- meu algoritmo realiza muitas operações que são baratas na estrutura de dados em questão?

**Tabelas de símbolos**

Uma tabela de símbolos:

- também é chamada de dicionário
- corresponde a um conjunto de itens,
    - em que cada item possui uma chave e um valor.
- é uma estrutura dinâmica, isto é,

    - suporta operações de inserção e remoção, além da busca.
- trata-se de um Tipo de Dado Abstrato, pois
    - o foco está no propósito da estrutura, e não em sua implementação.

Vamos começar a pensar na implementação de uma tabela de símbolos
- e nas estruturas de dados que podemos usar para tanto

Queremos armazenar e acessar dados com facilidade a partir de suas chaves
- as chaves podem ser nomes, IPs, configurações, etc.

Como exemplo, vamos supor que queremos criar uma lista de telefones.
- uma ideia é utilizar um vetor indexado pelas chaves
    - ou seja, pelos nomes das pessoas
- a vantagem dessa abordagem é que o tempo de acesso é constante (O(1)).
- se as chaves estiverem num intervalo bem comportado, por exemplo, inteiros entre 1 e 1000, isso é viável.
    - infelizmente (ou não), nomes raramente são inteiros entre 1 e 1000.
- insistindo nessa ideia, podemos converter/considerar a representação numérica/binária de cada nome
    - e usar esse número para indexar o vetor
    - qual o problema dessa abordagem? Qual será o tamanho do vetor resultante?
        - no caso dos nomes teríamos uma posição para cada nome possível
        - considerando que cada caracter tem 26 possibilidades e um nome pode ter até 30 caracteres
            - o vetor teria 26^30 ~= 2,81*10^42 ~= 2^141 posições.
        - como comparação, o armazenamento total disponível na Terra é da ordem de 10^25 bits.
    - em geral, o tamanho serial da ordem de 2^n, sendo n o número de bits da chave.

O tamanho do vetor no exemplo anterior deixa claro que essa abordagem é inviável.
- mas existe uma estrutura de dados que
    - suporta busca, inserção e remoção em tempo constante (O(1))
    - e ocupa espaço proporcional ao número de elementos armazenados.

**Tabelas de espalhamento (hash tables)**

Trata-se de uma implementação bastante popular e eficiente para tabelas de símbolos.

Hash tables propriamente implementadas suportam operações de consulta, inserção e remoção muito eficientes, i.e., tempo constante (O(1)) por operação.

- a eficiência das operações depende da hash table ter tamanho adequado e uma boa função de espalhamento (hash function)
    - vale destacar que é fácil implementar uma função de espalhamento problemática.
- além disso, a hash table não dá garantia de eficiência de pior caso,
    - pois sempre podem existir conjuntos de dados patológicos.

Aplicações:

- manutenção de variáveis conhecidas em compiladores.
- bloquear tráfego de certos IPs.
- detectar duplicatas.
- 2-sum problem:
    - dado um vetor v de tamanho n e um inteiro t, encontrar os pares de elementos em v cuja soma é igual a t.

Abordagens para o 2-sum problem:

- busca exaustiva, i.e., para cada elemento em v verificar se cada um dos n-1 outros elementos somados ao primeiro resulta em t. Eficiência \Theta(n^2)

```
int twoSumBruteForce(int v[], int n, int t)
{
   int i, j;
   for (i = 0; i < n; i++)
       for (j = i + 1; j < n; j++)
           if (v[i] + v[j] == t)
               return 1;
   return 0;
}
```

- ordenação + buscas binárias. Eficiência \Theta(n log n)
    - bônus: é possível substituir a parte das buscas binárias por uma passagem linear no vetor ordenado. Como?

```
int twoSumBinarySearch(int v[], int n, int t)
{
   int i, j, y;
   sort(v, n);
   for (i = 0; i < n; i++)
   {
       y = t - v[i];
       if (binarySearch(v, n, y))
           return 1;
```

```
    }
    return 0;
}
```

- percorrer o vetor inserindo cada elemento numa hash table e, em seguida, percorrê-lo novamente consultando, para cada elemento e, se seu complemento (t - e) está na hash table. Eficiência \Theta(n)

```
int twoSumHashTable(int v[], int n, int t)
{
    int i, j, y;
    int h[n];
    for (i = 0; i < n; i++)
        insertHashTable(h, n, v[i]);
    for (i = 0; i < n; i++)
    {
        y = t - v[i];
        if (lookupHashTable(h, n, y))
            return 1;
    }
    return 0;
}
```