

Algoritmos e Estruturas de Dados 2 (AED2)

Apresentação, estruturas de dados, tabelas de símbolos

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

Apresentação do curso

Detalhes técnicos

- Página do curso - <http://www.aloc.ufscar.br/felice/ensino/2024s1aed2/aed2.php>
- Cronograma e critérios de avaliação,
 - Provas (P1, P2, P3),
 - Trabalhos práticos (TP1, TP2, TP3).
- Listas de exercícios e bibliografia.

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 áreas 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 díspares.
- Eles são interessantes, divertidos e desafiadores, pois o desenvolvimento de algoritmos mistura conhecimento técnico com criatividade.

Neste curso vamos estudar diversos problemas e apresentar ferramentas

- para que vocês possam desenvolver soluções interessantes para eles,
 - bem como avaliar a qualidade destas soluções.
- Observem a importância de conseguir analisar as soluções,
 - caso contrário não temos critério para escolher entre elas.

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.

Habilidades que serão desenvolvidas:

- Tornar-se um melhor programador.
- Melhorar habilidades analíticas.
- Aprender a pensar algoritmicamente.
 - i.e., ser capaz de entender as regras que regem diferentes processos.

Principais tópicos do curso:

- Tabelas de símbolos,
- Ordenação,
- Busca em grafos,
- Busca de palavras.

Esses tópicos serão permeados por análise de corretude e eficiência de algoritmos,

- pois não queremos focar apenas no conteúdo,
 - mas também no desenvolvimento do nosso senso crítico.

Ler/estudar por conta:

- Probabilidade e análise combinatória -
<https://pt.khanacademy.org/math/precalculus/x9e81a4f98389efdf:prob-com-b>
- Séries (progressões aritméticas e geométricas) -
<https://pt.khanacademy.org/math/precalculus/x9e81a4f98389efdf:series>
- 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
- Algoritmos e Estruturas de Dados 1 -
http://www.aloc.ufscar.br/felice/cursos_anteriores.php

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 chamada de dicionário:

- Corresponde a um conjunto de itens,
 - em que cada item possui uma chave e um valor.
- Suporta diversas operações sobre os itens,
 - sendo busca a principal delas.
- Costuma ser dinâmica, isto é,
 - suporta operações de inserção e remoção.
- É um Tipo Abstrato de Dado, pois
 - o foco está no propósito da estrutura, e não em sua implementação.
- Útil para armazenar e acessar dados com facilidade a partir de suas chaves,
 - as quais podem ser nomes, IPs, configurações, etc.
- Possui diversas aplicações, como:
 - manutenção de variáveis conhecidas em compiladores,
 - bloquear tráfego de certos IPs,
 - detectar duplicatas.

Na próxima aula vamos começar a pensar nas estruturas de dados

- que podemos usar para implementar uma tabela de símbolos.

Por ora, como exemplo, vamos considerar o seguinte problema.

Problema 2-sum

Definição:

- Dado um vetor 'v' de inteiros e um valor 'alvo' inteiro,
 - determinar se existe um par de elementos em 'v'
 - cuja soma é igual ao 'alvo'.

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 no valor 'alvo'.
- Também podemos pensar nessa abordagem como
 - verificar o valor da soma de todos os pares de elementos.

```
int twoSumBruteForce(int v[], int n, int alvo) {
    int i, j;
    for (i = 0; i < n; i++) // testa todos os pares
        for (j = i + 1; j < n; j++)
            if (v[i] + v[j] == alvo)
                return 1;
    return 0;
}
```

- Eficiência: $O(n^2)$, i.e., proporcional ao quadrado do tamanho do vetor,
 - pois todos os $(n \text{ escolhe } 2) = n(n - 1) / 2$ pares são testados.
- Quiz1: Por que posso começar o laço interno de $i + 1$? E se fosse subtração?

Busca linear pelo complemento de cada elemento,

- sendo que o complemento de um elemento
 - corresponde ao valor 'alvo' menos o valor do elemento.

```
int twoSumLinearSearch(int v[], int n, int alvo) {
    int i, j, compl;
    for (i = 0; i < n; i++) {
        compl = alvo - v[i];
        for (j = i + 1; j < n; j++) // busca linear pelo complemento
            if (v[j] == compl)
                return 1;
    }
    return 0;
}
```

- Eficiência: $O(n^2)$, pois é realizada uma busca para cada elemento,
 - e cada busca leva tempo linear no tamanho do vetor.
- Quiz2: Preciso ir até $n - 1$ no laço externo?

Ordenação + busca binária pelo complemento de cada elemento.

```
int twoSumBinarySearch(int v[], int n, int alvo) {
    int i, j, compl;
    sort(v, n);
    for (i = 0; i < n; i++) {
        compl = alvo - v[i];
        // busca binária pelo complemento
        if (binarySearch(v, n, compl))
            return 1;
    }
    return 0;
}
```

- Eficiência: $O(n \log n)$, pois é realizada uma ordenação eficiente
 - em tempo $O(n \log n)$
- seguida de uma busca para cada elemento,
 - sendo que cada busca binária leva tempo $O(\lg n)$.

Bônus: ordenação + busca linear esperta pelo complemento de cada elemento.

```
int twoSumSorting(int v[], int n, int alvo) {
    int i, j, compl;
    sort(v, n);
    j = n - 1;
    for (i = 0; i < j; i++) {
        compl = alvo - v[i];
        // continuo a busca de onde parei
        for (; j > i && v[j] >= compl; j--)
            if (v[j] == compl)
                return 1;
    }
    return 0;
}
```

- Quiz3: esse algoritmo está correto? Por que?
 - Dica: pense no que ocorre com o complemento de $v[i]$
 - quando o índice 'i' cresce.
- Eficiência: $O(n \log n)$, pois é realizada uma ordenação eficiente
 - em tempo $O(n \log n)$
 - seguida de uma passagem linear pelo vetor.
- Note que este método é levemente mais rápido que o anterior,
 - pois a segunda parte do algoritmo (após a ordenação)
 - leva tempo linear, i.e., $O(n)$.

Abordagem com Tabela de Símbolos (TS):

- Percorre o vetor inserindo cada elemento em uma TS.
- Em seguida, percorre o vetor consultando, para cada elemento,
 - se o complemento deste está na TS.

```
int twoSumSymbolTable(int v[], int n, int alvo) {
    int i, j, compl;
    symbolTable TS;
    for (i = 0; i < n; i++)
        insertSymbolTable(TS, v[i]);
    for (i = 0; i < n; i++) {
        compl = alvo - v[i];
        if (lookupSymbolTable(TS, compl))
            return 1;
    }
    return 0;
}
```

- Eficiência: $O(n (\text{tempoInserçãoTS}(n) + \text{tempoBuscaTS}(n)))$,
 - sendo que $\text{tempoInserçãoTS}(n)$
 - e $\text{tempoBuscaTS}(n)$ são, respectivamente,
 - tempos para inserir e buscar um elemento numa TS de tamanho 'n'.
- Vale destacar que, boas implementações para TS
 - possuem busca, inserção e remoção com eficiência $O(\lg n)$,
- e existem implementações com tempo $O(1)$ para essas operações.