AED2 - Aula 01
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 a sua disposição, para assim tomar decisões embasadas de projeto".

Mário César DC - UFSCar

Apresentação do curso

Página do curso - http://www2.dc.ufscar.br/~mario/ensino/2020s2/aed2/aed2.php

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: ling. de programação ≠ ser um eng. de alg.
- 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 de palavras,
- Busca em grafos.

e das estrut. de dados

} algs. eficientes p/ probl. fundamentais

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 sobre este.

Ler/estudar por conta:

- Probabilidade e análise combinatória - https://pt.khanacademy.org/math/precalculus/prob-comb
- Séries (progressões aritméticas e geométricas) - https://pt.khanacademy.org/math/precalculus/seq-induction

} matemática

- 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

} código

- Algoritmos e Estruturas de Dados 1 - http://www2.dc.ufscar.br/~mario/previous_courses.php

- recursão
- listas, pilhas e filas
- ordenação elementares
- árvores binárias e heaps

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.

} básicos

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 de Dado Abstrato, 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 escolhe 2) = n (n - 1) / 2 pares são testados.

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.

Ordenação + busca binária pelo complemento de cada elemento.

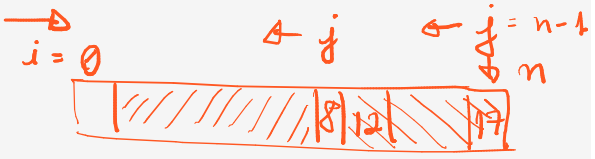
```
int twoSumBinarySearch(int v[], int n, int alvo) {
   int i, j, compl;
   sort(v, n);
   for (i = 0; i < n; i++) {
      compl = alvo - v[i];
      // busca binária pelo complemento
      if (binarySearch(v, n, compl))
          return 1;
   }
   return 0;
}
```

- 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;
}
```

passagem linear



compl = 10
compl = 7

- Quiz: 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;
}
```

Tempo TS | Eficiência Alg
O(n) | O(n²)
O(lg n) | O(n lg n)
O(1) | O(n)

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

Hash Table