Algoritmos e Estruturas de Dados 1 (AED1)

Apresentação, análise de algoritmos intuitiva, laços aninhados e logaritmos

"É esperado de um projetista 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/2022s2aed1/aed1.php
- Cronograma e critérios de avaliação,
    - Provas (P1, P2, Sub),
    - Trabalhos práticos (TP1, TP2, TP3, TP4).
- 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 bem definida e não ambígua 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 um problema computacional normalmente têm diversas soluções, e usar soluções mais eficientes torna viável atacar instâncias maiores e em novos contextos.
- 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:

- Recursão,
- Busca,
- Vetores e Listas Ligadas,
- Pilhas e Filas,
- Ordenação,
- Árvores,
- Backtracking.

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:

- 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

Laços aninhados

Vamos aquecer analisando as seguintes funções iterativas.

Um laço:

```c
int function1(int vector[], int tam, int element)
{
   int i = 0;
   while (i < tam)
   {
       if (element == vector[i])
           return 1;
       i++;
   }
   return 0;
}
```

O que faz a função anterior?

- Busca um elemento em um vetor e indica se o encontrou.

Qual o número de operações em função do tamanho do vetor?

- Da ordem do tamanho do vetor, ou O(tam).

Dois laços:

```c
int function2(int vectorA[], int tamA, int vectorB[], int tamB, int element)
{
    int i;
    for (i = 0; i < tamA; i++)
    {
        if (element == vectorA[i])
            return 1;
    }
    for (i = 0; i < tamB; i++)
    {
        if (element == vectorB[i])
            return 1;
    }
    return 0;
}
```

O que faz a função anterior?

- Busca um elemento em dois vetores e indica se o encontrou em algum deles.

Qual o número de operações em função dos tamanhos dos vetores?

- Proporcional ao tamanho dos vetores, ou O(tamA + tamB).

Dois laços aninhados:

```c
int function3(int vectorA[], int tamA, int vectorB[], int tamB)
{
    int i, j;
    for (i = 0; i < tamA; i++)
        for (j = 0; j < tamB; j++)
            if (vectorA[i] == vectorB[j])
                return 1;
    return 0;
}
```

O que faz a função anterior?

- Verifica se os vetores A e B têm algum elemento em comum.

Qual o número de operações em função dos tamanhos dos vetores?

- Proporcional ao produto entre os tamanhos, ou O(tamA * tamB).

Dois laços aninhados:

```
int function4(int vector[], int tam)
{
   int i, j;
   for (i = 0; i < tam; i++)
       for (j = i + 1; j < tam; j++)
           if (vector[i] == vector[j])
               return 1;
   return 0;
}
```

O que faz a função anterior?

- Verifica se um vetor tem algum elemento repetido.

Qual o número de operações em função do tamanho do vetor?

- Da ordem do tamanho do vetor ao quadrado, ou O(tam^2).

Logaritmos

Depois dos polinômios, como

- função linear (N), quadrática (N^2) e cúbica (N^3)

as funções que aparecem com maior frequência,

- quando estudamos o comportamento de algoritmos,
    - são as exponenciais (2^N) e os logaritmos (lg N),
        - que, aliás, são intimamente relacionadas.

O logaritmo na base 2 de um número N,

- denotado por (log2 N) ou (lg N),

é o expoente a que 2 deve ser elevado para produzir N,

- i.e., lg N = x ⇔ N = 2^x.

Note que, lg N só está definido se N é estritamente positivo.

Problema: dado um inteiro estritamente positivo N, calcular o piso de lg N.

Lembre que, o piso de um número K é

- o maior número inteiro i menor ou igual a K,
    - ou seja, i <= K < i+1.

Da definição de lg N podemos derivar um algoritmo

- que começa no valor 1,
    - vai dobrando esse valor até chegar em N,
- e contando quantas multiplicações por 2 foram realizadas,
    - i.e., qual o expoente a que 2 foi elevado.

Juntando essa ideia, com a definição de piso,

    - para saber quando parar,
- podemos projetar a seguinte função para resolver o problema:

```c
// A função lgProd recebe um inteiro N > 0
// e devolve o piso de lg N, ou seja,
// o único inteiro x tal que 2^x <= N < 2^(x+1).
int lgProd(int N) {
    int x, prod;
    x = 0;
    prod = 1;
    while (prod <= N / 2) {
        prod = 2 * prod;
        x += 1;
    }
    return x;
}
```

Invariantes e corretude:

- Determinar o que um trecho de código faz é relativamente simples,
    - basta verificar a sequência de instruções.
- Mas um laço torna isso mais complexo,
    - pois após inúmeras iterações pode ser difícil
        - determinar o resultado de tudo que foi feito.
- Invariantes são relações/propriedades
    - entre as variáveis de um algoritmo iterativo
        - que se mantém verdadeiras ao longo de todas as iterações.
- Observe que, no início de cada iteração do laço da função lgProd,
    - valem os seguintes invariantes envolvendo prod, x e N:

        prod = 2^x

        prod ≤ N
- Quando o algoritmo termina o laço, temos que

    prod > N/2
- Juntando essa desigualdade com as propriedade invariantes temos

$$N/2 < acum \leq N \qquad acum = 2^x$$

$$N/2 < 2^x \leq N$$

$$2\times \qquad N < 2^{x+1} \leq 2N$$

$$\lg N < x+1 \leq \lg 2N$$

$$\lg N < x+1 \leq \lg N + \lg 2$$

$$\lg N < x+1 \leq \lg N + 1$$

$$x \leq \lg N < x+1$$

- Como x é o maior inteiro que não supera lg N,
    - pela definição de piso, temos que x = ⌊lg N⌋.

Bônus: Podemos pensar numa definição equivalente para piso de lg N,

- como sendo o maior número de vezes que N pode ser dividido por 2
  - antes que o resultado fique menor ou igual a 1.

Essa definição sugere o seguinte algoritmo, que é baseado

- em divisões sucessivas no lugar das multiplicações sucessivas:

```
// A função lgDiv recebe um inteiro N > 0
// e devolve o piso de lg N, ou seja,
// o único inteiro x tal que 2^x <= N < 2^(x+1).
int lgDiv(int N)
{
   int x = 0;
   int resid = N;
   while (resid > 1)
   {
       resid = resid / 2;
       x += 1;
   }
   return x;
}
```

Observe que a expressão resid / 2

- corresponde à divisão inteira por 2,
  - só devolvendo objetos do tipo int.
- De fato, o valor da expressão é o piso de resid / 2.

Eficiência de tempo:

- Quiz: Quantas iterações os algoritmos anteriores executam
  - para encontrar o piso de lg N?
- Dica: conte quantas vezes x é incrementado
  - e responda em função do valor de N.