AED1 - Aula 01
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 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 começar 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:

- Recursão.
- Ordenação.
- Listas.
- Pilhas.
- Filas.
- Árvores.

- Backtracking.

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

Laços aninhados

Um laço - busque um elemento em um vetor:

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

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

Dois laços - busque um elemento em dois vetores:

```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;
}
```

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

Dois laços aninhados - verifique se os vetores A e B têm algum elemento em comum:

```
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;
}
```

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

Dois laços aninhados - verifique se um vetor tem algum elemento repetido:

```
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;
}
```

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

Logaritmos

O logaritmo na base 2 de um número N é o expoente a que 2 deve ser elevado para produzir N. O logaritmo na base 2 de N é denotado por lg N. 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 e piso podemos projetar a seguinte função para resolver o problema:

```
// A função lg recebe um inteiro N > 0 e devolve o piso de log N, ou seja,
```

```
// o único inteiro i tal que 2^i <= N < 2^(i+1).
int lg (int N)
{
  int i, n;
  i = 0;
  n = 1;
  while (n <= N/2) {
     n = 2 * n;
     i += 1;
  }
  return i;
}
```

Podemos pensar numa definição equivalente para piso de lg N, como sendo o máximo 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 código, que é baseado em divisões sucessivas no lugar das multiplicações sucessivas:

```
int lg(int N)
{
   int i = 0;
   int n = N;
   while (n > 1)
   {
       n = n / 2;
       i += 1;
   }
   return i;
}
```

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

Análise de corretude:

Considere o processo iterativo da primeira versão da função lg (aquela das multiplicações sucessivas por 2). Digamos que o início de cada iteração fica imediatamente antes do teste n <= N/2. Então as seguintes relações entre as variáveis n, i e N valem no início de cada iteração:

$n = 2^i$ e $n \leq N$.

Verifique que essas relações valem no início da primeira iteração, numa iteração intermediária qualquer e na última iteração. Essas relações são, portanto, invariantes. A validade das invariantes no início da última iteração garante que o

algoritmo está correto. Note que, quando o algoritmo termina temos n > N/2 e, portanto, N < 2n. Juntando ao segundo invariante obtemos n <= N < 2. Pelo primeiro invariante 2^i <= N < 2^(i+1). Logo, i <= lg N < i+1. Como essas inequações garantem que i é o maior inteiro que não supera lg N, temos que $i = \lfloor \log N \rfloor$.

Análise de eficiência:

Quantas iterações os algoritmos anteriores executam para encontrar o piso de lg N? (Dica: conte quantas vezes i é incrementado e responda em função do valor de N)