introducao

February 9, 2015

1 Introdução

O entendimento da complexidade de algoritmos é uma tarefa chave no desenvolvimento de sistemas e algoritmos capazes não apenas de fornecer a saída correta, mas de acordo com restrições impostas pelo contexto ou pelo domínio. Para exemplificar esta afirmação, tomemos como exemplo um algoritmo que precisa encontrar um número n em um vetor D. O algoritmo deve retornar um par (a, b) onde:

- a : Um valor booleano indicando se o número n foi encontrado; e
- b : Um número inteiro indicando a posição de n no vetor.

Uma possível implementação deste algoritmo é a seguinte (em pseudo-código).
Algoritmo 1.1:

```
int i = 0;
foreach (d in D) {
      if (d == n) {
          return {True, d};
      }
      i++;
}
return {False, 0};
```

Chama-se **operação fundamental** a operação principal do algoritmo. Neste caso, digamos que a operação fundamental é a igualdade (comparação) da linha 3.

É possível perceber que a quantidade de **operações fundamentais** varia conforme o tamanho de D ($|D|$). Podemos analisar das seguintes formas:

1. Considerando $n \in D$:
    1. Se $|D| = 1$, então a operação fundamental será executada 1 vez; e
    2. Se $|D| = 10$, então a operação fundamental será executada, no máximo, 10 vezes.
2. Considerando $n \notin D$:
    1. Independentemente $|D|$, a operação fundamental será executada $|D|$ vezes.

** Exercício 1.1:** Considerando as afirmações anteriores, qual a diferença entre n estar no início ou no final de D? Demonstre e justifique a afirmação 1B.

A análise destes casos e a representação disso seguindo um formalismo é o que chamamos **Análise de Algoritmos**. Existem diversas formas de analisar algoritmos, ou a computação deles, como considerar o tempo de execução e o espaço (a memória necessária para a execução do algoritmo). Neste curso estamos mais interessados na primeira forma, a análise do tempo de execução.

De modo geral, a análise do comportamento de algoritmos é estudada pela **Complexidade de Algoritmos**. Portanto, **Complexidade de Algoritmos** é a análise do esforço computacional (tempo ou memória) necessário para executar um algoritmo.

É importante ter a noção de que, para um determinado problema, podem haver diversas implementações.

** Exercício 1.2:** Este exercício será feito em grupo. Cada grupo deve escolher uma linguagem ou plataforma diferente (ex.: Java, C++, C#, Javascript, Python etc.). Seu programa deverá ler um arquivo com o seguinte formato:

- A primeira linha contém o número n;
- A segunda linha possum um número inteiro que corresponde à quantidade de números do conjunto D; e
- Da terceira linha em diante, estão os números do conjunto D.

Após ler o arquivo, seu programa deve procurar o número n no conjunto D e gerar um arquivo texto contendo três números, um em cada linha:

- A palavra True ou False, conforme a busca de n em D;
- Um número inteiro que corresponde à posição de n em D. Se $n \notin D$, então o valor pode ser 0 (zero); e
- Um número real que corresponde ao tempo de execução do programa.

Para verificar corretitude do seu programa, você deve usar os três arquivos a seguir:

1. dataset-1-a.csv
2. dataset-1-b.csv
3. dataset-1-c.csv

1.1 Complexidade e Desempenho de Algoritmos

Como você pode perceber, o tempo de execução do algoritmo pode variar conforme questões como a plataforma de programação e o ambiente de execução (a máquina, em si). É importante entender que este formato não pode ser usado como ferramenta para uma análise mais criteriosa e geral de algoritmos. Ainda, podemos dizer que a execução do algoritmo depende do conjunto de entradas e da sequência de operações fundamentais necessárias para o seu funcionamento. Portanto, vamos a alguns formalismos:

- D é conjunto de dados de entrada do algoritmo; e
- E é o conjunto de operações fundamentais para o funcionamento do algoritmo.

Assim, podemos definir um algoritmo como a função $a : D \to E$.

Execução resulta a sequência de execuções de operações fundamentais realizadas durante a execução do algoritmo a. É representada como $exec(a) : D \to E$.

Custo dá o comprimento da sequência $e \in E$. É representado como $custo : E \to \mathbb{R}_+$. Ou seja, **custo** é um real positivo que representa a quantidade de operações fundamentais. Obviamente, quanto menor a quantidade de operações, menor o custo.

Desempenho dá o custo (em termos das operações fundamentais) da execução de a com a entrada $d \in D$. É representado como $desempenho(a, d) := custo(exec(a, d))$. Isso nos permite analisar um algoritmo considerando as diversas entradas possíveis e, portanto, o desempenho de um algoritmo a com uma entrada $d \in D$ mede o custo da execução do algoritmo sobre esta entrada.

Se você preferir, também pode usar a notação a seguir:

- Algoritmo: $a(d) = e$. Ou seja, um algoritmo a é uma função que recebe uma entrada d e resulta em uma sequência de operações fundamentais e.

É importante perceber que o conceito de *desempenho* indica a quantidade de operações fundamentais executadas pelo algoritmo quando a entrada é d. Portanto, não se pode, necessariamente, julgar o algoritmo como melhor ou pior com base nesta quantiade. Um elemento é o fator *tempo*. Considerando, por exemplo, dois algoritmos: a_1 e a_2. Ambos executam sobre a mesma entrada d. O algoritmo a_1 executa 10 operações fundamentais em 1 segundo. O algoritmo a_2 executa a mesma quantidade de operações em 0.5 segundos. Portanto, o conceito de *desempenho* visto até o momento é dissociado do fator *tempo*. Mais detalhes sobre isso serão vistos posteriormente.

Exercício 1.3: O desempenho de um algoritmo sempre cresce com o tamanho da entrada? Por quê?

Exercício 1.4: Considere o problema de encontrar o maior valor em um conjunto de dados. Conside, também, que haja diversos conjuntos de dados, em arquivos texto que contêm os elementos destes conjuntos, um em cada linha. Crie um programa que lê cada um dos conjuntos de dados a seguir, procura o maior valor e gera um arquivo de saída contendo, em cada linha: o maior valor encontrado e o tempo de execução do algoritmo. Depois de executar estes experimentos, lendo os arquivos de dados e encontrando o maior valor, plote os tempos de execução um gráfico que representa a evolução do desempenho (a execução do algoritmo conforme a entrada). O que se pode perceber? Comente.

Os conjuntos de dados:

- dataset-2-a.csv
- dataset-2-b.csv
- dataset-2-c.csv
- dataset-2-d.csv
- dataset-2-e.csv ou sua versão compactada (com ˜3,8MB): dataset-2-e.rar

1.2 Complexidade de algoritmo

Embora a função *complexidade*() vista anteriormente seja útil como um critério para se entender a complexidade de algoritmos, ela não é suficiente. A análise do algoritmo 1.1 demonstrou justamente isso: a quantidade de operações fundamentais varia conforme a entrada e, portanto, a noção atual de desempenho precisa ser complementada.

A função **tamanho** é definida como: $tamanho(d) = n$, com $d \in D$. Esta função retorna n: o tamanho (a quantidade de elementos) da entrada d.

No caso de a entrada ser uma lista, $tamanho(d)$ resulta na quantidade de elementos da lista. No caso de ser um grafo, a informação pode ser composta pela quantidade de vértices e arestas.

Exercício 1.5: Por que é importante a utilização da função *tamanho*() no estudo da complexidade de um algoritmo?

Utilizando a função $tamanho(d)$, a função $desempenho(a, d)$ pode ser condensada na função **avalia**, definida como: $avalia(a, n)$ resulta em um número real que representa uma medida de complexidade do algoritmo com base no tamanho da entrada. Este valor pode ser entendido como um valor máximo.

Seguindo este raciocínio, a definição de $avalia(a, n)$ envolve um conjunto: todos os desempenhos para entradas de tamanho n. Dado $n \in N$, considere D_n o conjunto das entradas como tamanho n:

$D_n = \{d \in D : tamanho(d) = n\}$.

No geral, o desempenho de um algoritmo depende [do tamanho] da entrada. Assim, é razoável considerar o tamanho máximo e médio da entrada.

1.2.1 Complexidade média (ou esperada)

Considere um algoritmo a possa receber 100 entradas d_j de tamanho 10 ($n = 10$). Cada entrada d_j possui desempenho r_j. Assim, o desempenho esperado é o valor médio dado por:

$c_M(a, n) = avalia_M(a, n) = \frac{r_1+r_2+...+r_{100}}{100}$

ou

$avalia_M(a, n) = \frac{desempenho(a,d_1)+desempenho(a,d_2)+...+desempenho(a,d_{100})}{100}$.

Em outras palavras, o desempenho esperado considera a média esperada dos desempenhos do algoritmo. Neste caso, considera-se que cada uma das d_j entradas possui a mesma probabilidade de ocorrer, ou seja, segue uma *distribuição uniforme*.

De maneira geral, a *complexidade média (ou esperada)* de um algoritmo é dada por:

$avalia_M(a, n) = \sum_{d \in D_n} (prob(d) \times desempenho(a, d))$

onde $prob(d)$ dá a probabilidade de ocorrer a entrada d no conjunto D_n.

Exercício 1.6: Considere um algorimto a cujo desempenho sobre cada entrada é o tamanho da entrada. Determine sua complexidade média $avalia_M(a, 50)$ considerando distribuição uniforme.

1.2.2 Complexidade pessimista (pior caso)

O algoritmo do **Exercício 1.6** possui uma característia interessante: o desempenho do alogoritmo sobre cada entrada é o tamanho da entrada. Assim, o seu pior desempenho sobre entradas com tamanho até 20 será 20:

$avalia(a, [1, 2, ..., 20]) = \max\{1, 2, ..., 20\} = 20$

A *complexidade pessimista* de um algoritmo fornece o seu desempenho no pior caso (ou o pior desempenho que se pode esperar) e é definida como:

$avalia_P(a, n) = \max\frac{desempenho(a,d)\in(R)}{d\in D_n}$.

Em outras palavras, a *complexidade pessimista* de um algoritmo a é o valor máximo de seus desempenhos sobre todas as entradas com tamanho n.

Exercício 1.7: Considere o algoritmo abaixo:

Algoritmo 1.2: encontrar o máximo valor na lista

```
n = tamanho(lista);
max = lista[1];
for(i = 2; i < n; i++) {
    if (max < lista[i]) {
        max = lista[i];
    }
}
return max;
```

Determine a *complexidade pessimista* $avalia_P(a, 20)$.

Exercício 1.8: Como poderia ser definida a *complexidade otimista*? Discuta a sua utilidade.

Exercício 1.9: Determine as complexidades média e pessimista do **Algoritmo 1.1**.