NOÇÕES INICIAIS DE COMPLEXIDADE DE ALGORITMOS

Prof. Bruno de Castro Honorato Silva

November 30, 2020

1 Introdução

Durante a implementação de um algoritmo, há muitas coisas importantes que devem ser resolvidas: facilidade de uso, modularidade, segurança, manutenção, etc. A questão é que só podemos contemplar todos estes pontos se tivermos desempenho. Desempenho é como uma moeda através da qual podemos contemplar todos estes pontos supracitados.

O desempenho de um algoritmo implica diretamente na escala da tarefas. Imagine um editor de texto que pode carregar 1000 páginas, mas que o seu o algoritmo de correção ortográfica leva 1 minuto para verificar a ortografia de 1 página, ou um editor de imagens que leva 1 hora para rotacionar uma imagem 90 graus para a esquerda. Se um recurso de software não pode lidar com a escala de tarefas que os usuários precisam executar, este recurso está fadado ao desuso.

A principal abordagem para se mensurar o desempenho de um algoritmo é a análise de complexidade. Na análise de complexidade, avaliamos o desempenho de um algoritmo em termos de tamanho de entrada. Algoritmos manipulam dados. A este conjunto de dados, denominaremos entrada. Todo conjunto de dados possuí um tamanho (ou dimensão). A este tamanho, denotaremos por n. Consideremos então que a porção de tempo necessária para se processar a entrada de dados é denominada como tempo de resposta. Existe uma relação direta entre o tempo de resposta, este denotado por T, e o tamanho de uma entrada. É notório que quanto maior o tamanho da entrada, maior será o número de operações para processá-la. Sendo maior o número de operações, invariavelmente será maior o tempo de execução do algoritmo.

Programadores menos experientes tendem a imaginar que o tamanho de entrada afeta proporcionalmente o número de operações, e por consequência, o de tempo de processamento. Por exemplo: se um algoritmo executa 10 operações para uma entrada de tamanho 5, o mesmo tenderá a gastar 20 operações para um entrada de tamanho 10 e consequentemente levará o dobro do tempo. Algoritmos são projetados para resolver problemas. A realidade é que a maioria dos problemas não pode ser resolvido com este tipo de comportamento. A análise de complexidade de algoritmos visa determinar qual o comportamento de um algoritmo. Tal comportamento é denotado por uma expressão matemática que irá definir T em função de n, i. e., $T(n)$.

Quando nos deparamos com cenários em que as entradas tem tamanhos relativamente pequenos, poucos instruções são necessárias e o tempo de resposta é mínimo. Desta forma, a complexidade do algoritmo (ou a relação entre o tempo de resposta e o tamanho da entrada) acaba sendo subestimada. Sendo assim, a forma de classificação da complexidade de algoritmos mais comum é aquela que define o tempo resposta T considerando cenários em que n tendem ao infinito (ou são suficientemente grandes). Em cenários em que n é suficientemente grande, a curva de crescimento de T tende a crescer com proporção vertical. A este comportamento de crescimento vertical, denominamos crescimento assintótico de T em função de n.

A notação matemática utilizada para se definir o comportamento assintótico de um algoritmo é a notação *Big-oh* (ou simplesmente notação O). A notação O, quando aplicada a um algoritmo, nos fornece um limite superior para o tempo de execução desse algoritmo. A notação O é uma notação universal para descrever o desempenho de algoritmos.

Para melhor compreender o volume de conceitos postos até aqui, considere o algoritmo de busca linear (ou sequencial). Este algoritmo recebe como entrada um vetor de tamanho n e uma chave de busca x. Eis o código deste algoritmo em linguagem C:

```
int buscaLinear(char *vetor, int n, char x){
  int i;
  for (i = 0; i < n; i++){
    if (vetor[i] == x){
      return i;
    }
  }
  return -1;
}
```

No melhor caso, o elemento a ser buscado é encontrado logo na primeira tentativa da busca. No pior caso, o elemento a ser buscado encontra-se na última posição e são feitas n comparações, sendo n o número total de elementos do vetor. No caso médio, o elemento é encontrado após $(n+1)/2$ comparações. Desta forma, observa-se que o número máximo (ou limite superior) de instruções a serem processadas para encontrar x não deve ser maior que n, e consequentemente, fica estabelecido que o que o comportamento de assintótico deste algoritmo é $O(n)$.

Eis outro exemplo: imagine uma sala de aula com 100 (ou $n = 100$) alunos em que você está inserido e perdeu sua caneta. Agora, você quer recuperar essa caneta. Eis algumas maneiras de encontrar a caneta:

- Ir e perguntar a cada aluno individualmente. Neste método, você fará até n perguntas no intuito de checar se alguém achou sua caneta. Se imaginarmos que este método é um algoritmo e a quantidade de perguntas são as operações, então o comportamento assintótico deste algoritmo pode ser matematicamente representado por $O(n)$.

- Perguntar a cada colega sobre a caneta. Além disso, você pede a ajuda de todos os colegas para que eles façam o mesmo. Assim, cada colega pergunta aos outros 99 colegas pela caneta e ainda perguntam se você já achou a bendita caneta. Desta forma, você e cada aluno da sala fará n

perguntas e teremos até n^2 feitas. Fazendo a mesma analogia do item anterior, este método seria um algoritmo com comportamento assintótico denotado por $O(n^2)$;

- Dividir a turma em dois grupos e perguntar: "O colega que encontrou a caneta está no lado esquerdo ou no lado direito da sala de aula?". Um dos colegas responde: "A caneta não está do lado direito". Você então descarta direito e o lado em que pode estar a caneta é subdividido entre esquerdo e direito. O processo é repetido até que sobrem dois colegas e, desta forma, um deles estará com a caneta ou a mesma não estará na sala. A cada pergunta, elimina-se metade do espaço de busca. Assim, até $\log_2 n$ perguntas seriam feitas. Fazendo a mesma analogia do item anterior, este método seria um algoritmo com comportamento assintótico denotado por $O(\log_2 n)$(ou apenas $O(\log n)$);

É válido ressaltar que a complexidade temporal de um algoritmo não exprime o tempo real que requer a execução do código. O tempo gasto para processar uma instrução varia entre computadores.

2 Classes de Complexidade

Por meio da definição do comportamento assintótico de um algoritmo com a notação O, é possível definir classes assintóticas. As mais comuns são enumeradas, em ordem de grandeza, como se segue:

1. Constante: $O(1)$. As instruções são executadas um número fixo de vezes. Não depende do tamanho dos dados de entrada;

2. Logarítmica: $O(\log n)$. Típica de algoritmos que resolvem um problema transformando-o em problemas menores;

3. Linear: $O(n)$. Em geral, uma certa quantidade de operações é realizada sobre cada um dos elementos de entrada;

4. Logarítmica-linear: $O(n \log n)$. Típica de algoritmos que trabalham com particionamento dos dados. Esses algoritmos resolvem um problema transformando-o em problemas menores, que são resolvidos de forma independente e depois unidos;

5. Quadrática: $O(n^2)$. Normalmente ocorre quando os dados são processados aos pares. Uma característica deste tipo de algoritmos é a presença de um aninhamento de dois comandos de repetição;

6. Cubica: $O(n^3)$. É caracterizado pela presença de três estruturas de repetição aninhadas;

7. Polinomial: $O(n^c)$. Geralmente ocorre quando se usa uma solução heurística. São úteis do ponto de vista prático;

8. Exponencial: $O(c^n)$. Geralmente ocorre quando se usa uma solução de força bruta. Não são úteis do ponto de vista prático;

9. Fatorial: $O(n!)$. Também classificada como Exponencial devido a seu polinômio, esta classe de algoritmos geralmente ocorre quando se usa uma solução de força bruta. Não são úteis do ponto de vista prático. Possui um comportamento muito pior que o exponencial;

A Figura 2 ilustra as curvas de crescimento assintóticas destas classes de complexidade algorítmicas.

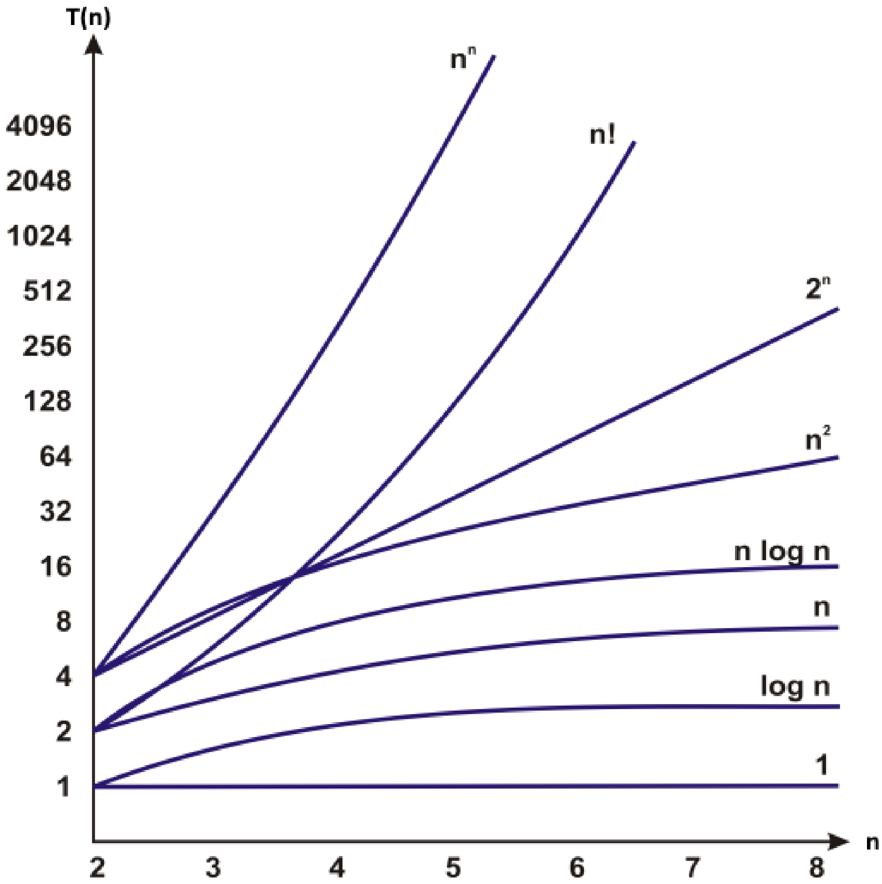

Figure 1: Ilustração das curvas de crescimento das classes assintóticas.

Quando projetamos e implementamos um algoritmo, deve-se buscar sempre que este algoritmo tenha a menor classe assintótica. Problemas que tenham algoritmos para solucioná-los a um custo expresso por um polinômio matemático são chamados P. A única classe não polinomial é a exponencial. Todas as outras classes são caracterizadas como polinomiais. Chamamos de NP os problemas que não tenham algoritmos polinomiais para resolvê-los.

3 Exercícios

1. Qual é a classe assintótica do algoritmo 3?

```
void algoritmo3(int n, int m){
  int a = 0;
  int b = 0;
  int i = 0;
  for (i = 0; i < n; i++)
      a = a + rand();
  for (i = 0; i < m; i++)
      b = b + rand();
  printf(”%i %i”, a, b);
}
```

2. Qual é a classe assintótica do algoritmo 4?

```
void algoritmo4(int n){
  int a = 0;
  int i = 0;
  int j = 0;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      a += i + j;
  printf(”%i”, a);
}
```

3. Qual é a classe assintótica do algoritmo 5?

```
void algoritmo5(int n){
  int a = 0;
  int i = 0;
  int j = 0;
  for (i = n/2; i < n; i++)
    for (j = 2; j < n; j*= 2 )
      a += n/2;
  printf(”%i”, a);
}
```

4. Qual é a classe assintótica do algoritmo 6?

```
void algoritmo6(int n){
  int a = 0;
  int i = n;
  while (i > 0) {
    a += i;
    i /= 2;
  }
  printf("%i", a);
}
```