Complexidade de Algoritmos

Estruturas de Dados – 2023.2

Prof. Pedro Nuno Moura

Material adaptado da Profa. Adriana Alvim

Complexidade de Algoritmos

- Algoritmo
  - Característica importante
    - Tempo de execução
  - Possível determinar através de métodos empíricos
    - Entradas diversas
  - Alternativamente
    - Ordem de grandeza através de métodos analíticos
    - Objetivo: expressão matemática que traduza o comportamento de tempo de um algoritmo
    - Independente do computador, linguagem e compiladores
    - Tarefa não é simples

Complexidade de Algoritmos

- Expressão matemática : simplificações modelo
  - Supor quantidade grande de dados
  - Comportamento assintótico
    - expressão matemática fornecerá valores de tempo válidos unicamente quando a quantidade de dados crescer suficientemente
  - Não considerar constantes aditivas ou multiplicativas na expressão matemática
  - Exprimir o tempo de execução em função da entrada

Complexidade de Algoritmos

- Execução de um algoritmo
  - Etapas elementares, passos
  - Cada passo
    - número fixo de operações básicas, cujos tempos são considerados constantes
    - operação básica de maior frequência é denominada operação dominante
    - o número de passos de um algoritmo pode ser interpretado com sendo o número de execuções da operação dominante
  - Exemplo
    - Ordenar elementos de uma sequencia
      - cada passo: comparação entre dois elementos

Complexidade de Algoritmos

- Exemplo: inversão de uma sequencia

```
1 for i := 1, ..., ⌊n/2⌋ do
2 |   temp := S[i];
3 |   S[i] := S[n − i + 1];
4 |   S[n − i + 1] := temp;
5 end
```

Algorithm 1: Inversão de uma sequencia.

  - A notação $\lfloor x \rfloor$ significa piso de x e representa o maior inteiro menor ou igual a x
  - Analogamente $\lceil x \rceil$ é o teto de x e corresponde ao menor inteiro maior ou igual a x
  - Assim, piso de 9/2 = 4 e teto de 9/2 = 5

Complexidade de Algoritmos

- Exemplo: inversão de uma sequencia

```
1 for i := 1, ..., ⌊n/2⌋ do
2 |   temp := S[i];
3 |   S[i] := S[n − i + 1];
4 |   S[n − i + 1] := temp;
5 end
```

Algorithm 1: Inversão de uma sequencia.

Cada entrada é uma sequencia que se deseja inverter

  - O algoritmo efetua as mesmas operações para sequencias de mesmo tamanho n
    - cada passo corresponde a troca de posição de dois elementos da sequencia
      - as três instruções de atribuição dentro do loop
  - O número de passos é igual ao número de execuções do bloco para (for), igual a piso de n/2, n > 1

Recursividade

- Procedimento que contém, em sua descrição, uma ou mais chamadas a si mesmo
  - Todo procedimento tem uma chamada externa
- De modo geral, a todo procedimento recursivo corresponde um outro não recursivo que executa, exatamente, a mesma computação
- Exemplo clássico é o cálculo do fatorial de um inteiro $n \geq 0$

Recursividade

- No primeiro algoritmo, a chamada externa é fat(n)
- No segundo exemplo fat representa um vetor, no final fat[n] contém o resultado do fatorial desejado

```
1 Função fat(i);
2 if (i < 1) then
3 |   fat(i) := 1;
4 else
5 |   fat(i) := i * fat(i − 1);
6 end
```

Algorithm 2: Fatorial (recursivo).

```
1 fat[0] := 1;
2 for j := 1, . . . , n do
3 |   fat[j] := j × fat[j − 1];
4 end
```

Algorithm 3: Fatorial (não recursivo).

Complexidade de Algoritmos

- Exemplo: determinar soma C e produto D de duas matrizes $A=(a_{ij})$ e $B=(b_{ij})$, ambas n x n
  - C e D também possuem dimensões n x n e seus elementos c_{ij} e d_{ij} podem ser calculados por

$$c_{ij} = a_{ij} + b_{ij},$$
$$d_{ij} = \sum_{k=1}^{n} a_{ik} + b_{kj}.$$

Complexidade de Algoritmos

- Os Algoritmos 4 e 5 descrevem a computação da soma e do produto de duas matrizes

```
1 for i := 1, ..., n do
2 |   for j := 1, ..., n do
3 |   |   c_ij := a_ij + b_ij;
4 |   end
5 end
```

Algorithm 4: Soma de matrizes.

```
1 for i := 1, ..., n do
2 |   for j := 1, ..., n do
3 |   |   c_ij := 0;
4 |   |   for k := 1, ..., n do
5 |   |   |   c_ij := c_ij + a_ik × b_kj;
6 |   |   end
7 |   end
8 end
```

Algorithm 5: Produto de matrizes.

Complexidade de Algoritmos

- Os algoritmos de soma e produto efetuam as mesmas operações sempre que A e B forem matrizes de mesmas dimensões n x n
  - A variável independente é o parâmetro n
  - Cada passo do Algoritmo 4 corresponde à execução de uma soma $a_{ij} + b_{ij}$
  - Cada passo do Algoritmo 5 corresponde à execução de do produto $a_{ik} * b_{kj}$
  - O número total de passos é igual ao número total de somas $a_{ik} + b_{kj}$ e produtos
  - O Algoritmo 4 efetua n^2 passos e o Algoritmo 5 efetua n^3 passos

Complexidade de Algoritmos

- Noção de complexidade de tempo
  - Seja A um algoritmo, $\{E_1, ..., E_n\}$ o conjunto de todas as entradas possíveis de A
  - Denote por t_i o número de passos efetuados por A, quando entrada for E_i
  - Definem-se
  - Complexidade de pior caso $= \max_{E_i \in E}\{t_i\},$
  - Complexidade de melhor caso $= \min_{E_i \in E}\{t_i\},$
  - Complexidade de caso médio $= \sum_{1 \le i \le m} p_i t_i.$
    - onde p_i é a probabilidade de ocorrência da entrada E_i

Complexidade de Algoritmos

- Noção de complexidade de tempo
  - A complexidade tem por objetivo avaliar a eficiência de tempo
  - A complexidade de tempo de pior caso corresponde ao número de passos que o algoritmo efetua no seu pior caso de execução
    - para a entrada mais desfavorável
  - É a mais importante entre as três
  - Fornece limite superior para o número de passos que o algoritmo pode efetuar, em qualquer caso
  - É a mais utilizada
  - O termo complexidade será empregado com o significado de complexidade de pior caso

Complexidade de Algoritmos

- Noção de complexidade de tempo
  - A complexidade de melhor caso, é de uso bem menos frequente
    - empregada em situações específicas
  - A complexidade de caso médio, embora importante, é menos utilizada do que a de pior caso por dois motivos
    - Exige prévio conhecimento de distribuição de probabilidade das diferentes entradas do algoritmo
    - Em diversos casos, esta distribuição é desconhecida
  - Além disso o cálculo de seu valor é frequentemente de tratamento matemático complexo

Complexidade de Algoritmos

- Notação O
  - Simplificações
    - Constantes aditivas e multiplicativas
      - 3n passos, aproximado para n passos
    - Interesse assintótico, termos de menor grau podem ser desprezados
      - $n^2 + n$ será aproximado para n^2
      - $6n^3 + 4n - 9$ será aproximado para n^3

Complexidade de Algoritmos

- Notação O (“O grande” – O)
  - sejam f, h funções reais positivas de variável inteira n
  - diz-se que f é O(h), escrevendo-se f = O(h) quando existir uma constante c > 0 e um valor inteiro n_0 tal que

$$n > n_0 \Rightarrow f(n) \leq c * h(n)$$

  - ou seja, a função atua como um limite superior para valores assintóticos da função f

Complexidade de Algoritmos

- Exemplo

$$f = n^2 - 1 \Rightarrow f = O(n^2).$$
$$f = n^2 - 1 \Rightarrow f = O(n^3).$$
$$f = 403 \Rightarrow f = O(1).$$
$$f = 5 + 2\log n + 3\log^2 n \Rightarrow f = O(\log^2 n).$$
$$f = 5 + 2\log n + 3\log^2 n \Rightarrow f = O(n).$$
$$f = 3n + 5\log n + 2 \Rightarrow f = O(n).$$
$$f = 5 \times 2^n + 5n^{10} \Rightarrow f = O(2^n).$$

- As seguintes propriedades são úteis para manipular expressões em notação O
- Sejam g, h funções reais positivas e k uma constante
- Então
  - O(g + h) = O(g) + O(h),
  - O(k * g) = k O(g) = O(g)

Complexidade de Algoritmos

- A notação O será utilizada para exprimir complexidades dos Algoritmos de 1 a 5
  - Todos apresentam a propriedade de o número de passos manter-se o mesmo quando aplicados a entradas diferentes de mesmo tamanho
  - Para um mesmo valor de n, o número de passos mantém-se constante
  - Como a variável independente é o valor de n, conclui-se que as complexidades de pior, melhor e caso médio são todas iguais entre si para cada algoritmo

Complexidade de Algoritmos

- O algoritmo Inversão de uma sequencia efetua sempre piso de n/2 passos, logo sua complexidade é O(n)

```
1 for i := 1, ..., ⌊n/2⌋ do
2 |   temp := S[i];
3 |   S[i] := S[n − i + 1];
4 |   S[n − i + 1] := temp;
5 end
```

Algorithm 1: Inversão de uma sequencia.

Complexidade de Algoritmos

- No algoritmo Fatorial (não recursivo) o número de passos é igual ao número de produtos j x fat(j-1), que é n, logo sua complexidade é O(n)

```
1 fat[0] := 1;
2 for j := 1, ..., n do
3 |   fat[j] := j × fat[j − 1];
4 end
```

Algorithm 3: Fatorial (não recursivo).

Complexidade de Algoritmos

- A complexidade do algoritmo de soma de matrizes é igual a $O(n^2)$ e o de produto é igual a $O(n^3)$

```
1 for i := 1, ..., n do
2 |   for j := 1, ..., n do
3 |   |   c_ij := a_ij + b_ij;
4 |   end
5 end
```

Algorithm 4: Soma de matrizes.

```
1 for i := 1, ..., n do
2 |   for j := 1, ..., n do
3 |   |   c_ij := 0;
4 |   |   for k := 1, ..., n do
5 |   |   |   c_ij := c_ij + a_ik × b_kj;
6 |   |   end
7 |   end
8 end
```

Algorithm 5: Produto de matrizes.

Complexidade de Algoritmos

- Para encontrar a complexidade de algoritmos recursivos aplica-se a seguinte técnica
  - Determina-se o número total de chamadas ao procedimento recursivo
  - Calcula-se a complexidade da execução correspondente a uma única chamada
    - Sem considerar as chamadas recursivas
  - A complexidade total será o produto do número de chamadas pela complexidade da computação de uma chamada isolada

Complexidade de Algoritmos

- Exemplo

```
1 Função fat(i);
2 if (i < 1) then
3 |   fat(i) := 1;
4 else
5 |   fat(i) := i * fat(i − 1);
6 end
```

Algorithm 2: Fatorial (recursivo).

  - Para calcular o fatorial de n > 0, o algoritmo efetua um total de n chamadas ao procedimento fat
  - A complexidade de uma chamada é constante, O(1)
    - Para n > 1, apenas um produto é efetuado e quando n ≤ 1 apenas uma atribuição é efetuada
  - Logo, complexidade final é O(n)

Complexidade de Algoritmos

- Até agora os algoritmos efetuam exatamente os mesmos passos para entradas diferentes de mesmo tamanho
  - Este não é o caso geral

Complexidade de Algoritmos

- Suponha um problema cuja entrada consiste de duas matrizes A e B de dimensões n x n e um parâmetro binário x com valores possíveis 0 e 1
  - dependendo do valor de x, deve-se calcular a soma ou o produto das matrizes
  - se x = 0, calcular a matriz A + B, ou, se x = 1, calcular a matriz A * B
  - um algoritmo para efetuar tarefa é simples
  - sua entrada consiste em $2n^2 + 1$ informações e possui tamanho $O(n^2)$
  - em seguida verifica-se o valor de x obtido na entrada

Complexidade de Algoritmos

- Continuação
  - se x = 0 executa-se o algoritmo da soma de matrizes e se x = 1 executa-se o algoritmo do produto de matrizes
  - para cada valor de n, o algoritmo é sensível a duas entradas distintas, correspondendo aos valores 0 ou 1 de x, respectivamente
  - a complexidade de melhor caso corresponde a x = 0, o pior caso quando x = 1
  - para determinar a complexidade de caso médio, seja q a probabilidade de que o valor de x, da entrada, seja igual a 0
    - então a expressão de complexidade de caso médio é

$$q\ n^2 + (1-q)\ n^3$$

Complexidade de Algoritmos

- A notação Θ (“Téta”) é útil para exprimir limites superiores justos
  - sejam f, g funções reais positivas da variável inteira n
  - diz-se que f é Θ(g), escrevendo-se f = Θ(g) quando ambas as condições f = O(g) e g = O(f) forem verificadas
  - a notação Θ exprime o fato de que as duas funções possuem a mesma grandeza assintótica

Complexidade de Algoritmos

- A notação Θ é útil para exprimir limites superiores justos
  - por exemplo, se $f = n^2 - 1$, $g = n^2$ e $h = n^3$, então
  - f é O(g) , f é O(h), g é O(f), mas h não é O(f)
  - consequentemente, f é Θ(g), mas f não é Θ(h)
  - da mesma forma, se $f = 5 + 2 \log n + \log^2 n$ e $g = n$, então f é O(g), mas f não é Θ(g)
  - no caso, f é $\Theta(\log^2 n)$

Complexidade de Algoritmos

- Assim como a notação O é útil para descrever limites superiores assintóticos, a notação Ω ("Ômega") é empregada para limites inferiores assintóticos
  - sejam f, h funções reais positivas da variável inteira n
  - diz-se que f é Ω(g), escrevendo-se f = Ω(g) quando existir uma constante c > 0 e um valor inteiro n_0 tal que

$$n > n_0 \Rightarrow f(n) \geq c * h(n)$$

  - por exemplo, se f = n^2 - 1, então
  - São válidas as igualdades f = Ω(n^2), f = Ω(n), f = Ω(1), mas não vale f = Ω(n^3)

Complexidade de Algoritmos

- Algoritmos ótimos
  - Complexidade está relacionada a um dado algoritmo
    - não leva em consideração existência de outros algoritmos para mesmo problema
  - Seja P um problema
    - um limite inferior para P é uma função I tal que a complexidade de pior caso de qualquer algoritmo que resolva P é Ω(I)
      - todo algoritmo que resolve P efetua pelo menos Ω(I) passos
    - se existir um algoritmo A cuja complexidade seja O(I) então A é denominado algoritmo ótimo para P
      - nesse caso o limite Ω(I) é o melhor (maior) possível

Complexidade de Algoritmos

- Algoritmos ótimos
  - Intuitivamente, um algoritmo ótimo é aquele que apresenta a menor complexidade dentre todos os possíveis algoritmos existentes para resolver o mesmo problema
  - Assim como a notação O é conveniente para exprimir complexidades, a notação Ω é útil para limites inferiores

Complexidade de Algoritmos

- Algoritmos ótimos
  - Existem limites inferiores naturais, como, por exemplo, o tamanho da entrada
    - todo possível algoritmo para o problema considerado deverá, necessariamente, efetuar a leitura de entrada
  - Algoritmo inversão de sequência
    - entrada sequencia de n elementos
    - limite inferior para um algoritmo para o problema de inversão de sequencias é Ω(n)
    - a complexidade do Algoritmo 1 é O(n)
    - então ele é um algoritmo ótimo.

Complexidade de Algoritmos

- Algoritmos ótimos
  - Poucos problemas para os quais existem limites inferiores não triviais conhecidos, como por exemplo, ordenar uma sequência de n elementos
    - O limite trivial, extraído da literatura, é Ω(n)
    - Há uma prova matemática de que Ω(n log n) é um limite inferior
      - existem algoritmos de ordenação com complexidade O(n log n)
    - Tais algoritmos são ótimos
  - Para uma quantidade grande de problemas a distância entre o melhor (maior) limite inferior conhecido e o algoritmo de melhor (menor) complexidade é grande

Complexidade de Algoritmos

- Algoritmos ótimos
  - De modo geral, o interesse é determinar a função que represente o maior limite inferior possível para um problema
  - Analogamente, para um certo algoritmo, o interesse é encontrar a função representativa da menor complexidade de pior caso do algoritmo

Complexidade de Algoritmos

- Algoritmos ótimos
  - Para uma quantidade grande de problemas a distância entre o melhor (maior) limite inferior conhecido e o algoritmo de melhor (menor) complexidade é grande

Complexidade de Algoritmos

- Implementação de pilha usando vetor
- Operações
  - Vazia – O(1)
  - Cheia – O(1)
  - Empilha – O(1)
  - Desempilha – O(1)

Complexidade de Algoritmos

- Implementação de fila circular usando vetor
- Operações
  - Vazia – O(1)
  - Cheia – O(1)
  - Insere no fim (FIFO) – O(1)
  - Remove do início – O(1)
  - Insere em uma dada posição – O(n)

Referências

- Jayme Luiz Szwarcfiter e Lilian Markenzon, Estruturas de Dados e Seus Algoritmos, LTC Livros Técnicos e Científicos Ed, 1994.
- Capítulo 1 do livro do Robert Sedgewick e Kevin Wayne, Algorithms. 4ª ed. Addison Wesley, 2016.

FIM