Estruturas de Dados

Complexidade de Algoritmos

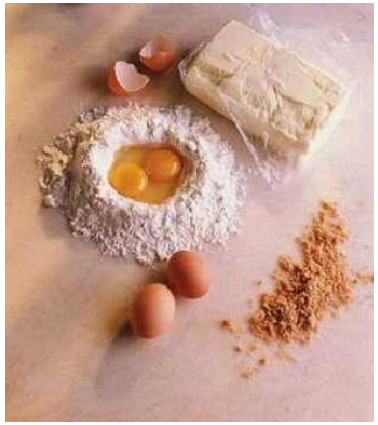
Prof. MSc. Raphael Gomes
raphael@ifg.edu.br

O que veremos hoje...

- *Noções Básicas de Complexidade de Algoritmos*
- *Notação O*
- *Exemplos*

Algoritmos

- *Um algoritmo é um procedimento passo-a-passo para solucionar um problema em um tempo finito.*



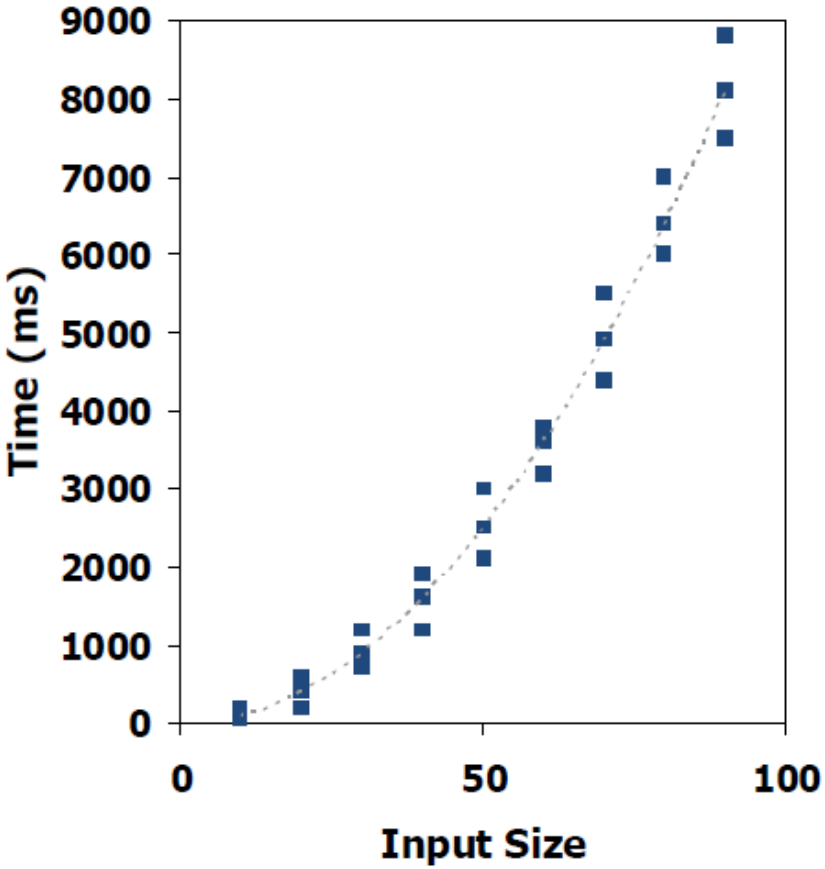
Entrada

Processamento

Saída

Tempo de Execução – Resultados Experimentais

- *Escreva um programa implementando o algoritmo.*
- *Execute o programa com entradas de tamanho e de composição variadas.*
- *Use um método, tal como gettimeofday(), para medir o tempo real de execução do algoritmo.*
- *Apresente os resultados em um gráfico.*



Limitações do Experimento

- *É necessário **implementar** o algoritmo, e isto pode ser uma tarefa difícil.*
- *Os resultados podem não ser **estendidos** para o tempo de execução de outras entradas não incluídas no experimento.*
- *Para comparar dois algoritmos, os **mesmos ambientes** de hardware e de software devem ser usados.*

Análise Teórica

- *Objetivo:*
  - ✓ avaliar a velocidade de um algoritmo, independentemente dos ambientes de hardware e de software.

- *Usar uma descrição de alto nível do algoritmo, em vez de uma implementação (ex. Pseudocódigo).*

- *Caracterizar o tempo de execução como uma função do tamanho* **n** *da entrada.*
  - ✓ Levar em conta todas as entradas possíveis.

Análise Teórica

- *Análise de um algoritmo particular*
  - Qual é o custo de usar um dado algoritmo para resolver um problema específico?
  - Características que devem ser investigadas:
    - *1. análise do número de vezes que cada parte do algoritmo deve ser executada,*
    - *2. estudo da quantidade de memória necessária.*

Análise Teórica

- *Análise de uma classe de algoritmos.*
  - ✓ Qual é o algoritmo de menor custo possível para resolver um problema particular?
  - ✓ Toda uma família de algoritmos é investigada.
  - ✓ Procura-se identificar um que seja o melhor possível.

Custo de Algoritmos

- *Determinando o menor custo possível para resolver problemas de uma dada classe, temos a medida da dificuldade inerente para resolver o problema.*
- *"menor custo possível" ?*
  - ✓ custo da solução mais simples para resolver o problema.
- *Encontrar a solução mais simples para resolver um problema é, por si só, um problema muito difícil.*
- *Para alguns problemas, sempre poderá haver um método mais eficiente, à espera de ser descoberto.*
  - ✓ a complexidade de muitos problemas ainda é desconhecida.

Função de complexidade

- *Para medir o custo de execução de um algoritmo é comum definir uma função de custo ou função de complexidade;*
- ***Função de complexidade de tempo**: f(n), mede o tempo necessário para executar um algoritmo em um problema de tamanho n.*
- ***Função de complexidade de espaço**: f(n), mede a memória necessária para executar um algoritmo em um problema de tamanho n.*
- *A complexidade de tempo na realidade não representa tempo diretamente, mas o **número de vezes** que determinada operação considerada relevante é executada.*
  - ✓ Exemplo: algoritmos de ordenação
    - *considerar o número de comparações e de movimentações;*
    - *ignorar as operações aritméticas, de atribuição e de manipulações de índices.*

Tamanho da entrada de dados

- *A medida do custo de execução de um algoritmo depende principalmente do tamanho da entrada dos dados.*
  - ✓ indexa a complexidade, independemente da representação escrita do algoritmo e, tipicamente, do custo de cada operação (apenas contagem de operações).

Melhor caso, pior caso e caso médio

- ***Melhor caso**: menor tempo de execução sobre todas as entradas de tamanho n.*

- ***Pior caso**: maior tempo de execução sobre todas as entradas de tamanho n.*
  - ✓ Se f é uma função de complexidade baseada na análise de pior caso, o custo de aplicar o algoritmo nunca é maior do que f(n).

- ***Caso médio** (ou caso esperado): média dos tempos de execução de todas as entradas de tamanho n.*

Complexidade de Algoritmos

- *Exemplo – pesquisa sequencial*
  - Seja f uma função de complexidade tal que f(n) é o número de registros consultados no arquivo (número de vezes que a chave de consulta é comparada com a chave de cada registro).
    - *melhor caso:*
      - f(n) = 1 (registro procurado é o primeiro consultado);
    - *pior caso:*
      - f(n) = n (registro procurado é o último consultado ou não está presente no arquivo);
    - *caso médio:*
      - f(n) = (n + 1)/2.

Complexidade de Algoritmos

- *Exemplo – pesquisa binária.*
  - ✓ Dados ordenados; divisão sucessiva do espaço de busca;
  - ✓ caso o valor procurado não exista no vetor (pior caso):
    - *f(n) = $log_2(n)$;*
  - ✓ caso o valor procurado exista no vetor:
    - *melhor caso: f(n) = 1, valor na mediana do vetor;*
    - *caso médio: f(n) = $log_2(n)$.*

Comportamento Assintótico de Funções

- *A escolha do algoritmo não é um problema crítico para problemas de tamanho pequeno.*

- *O comportamento assintótico de f(n) representa o limite do comportamento do custo quando n cresce.*
  - ✓ a medida de custo ou medida de complexidade relata o crescimento assintótico da operação considerada.

Notação O

- *Se f é uma função de complexidade de um algoritmo F, então O(f) é considerada a complexidade assintótica (ou comportamento assintótico) de F.*
- *Considere dois algoritmos, F e G, para uma mesma classe de problemas, se f(n) = 3g(n), então O(f(n)) = O(g(n)).*
- *um algoritmo O(n) é melhor que outro algoritmo $O(n^2)$?*
  - ✓ depende da constante de proporcionalidade e do tamanho do problema (exemplo: 100n e $2n^2$).

Classes de Comportamento Assintótico

- *Complexidade constante.*
  - f(n) = O(1), uso do algoritmo independe do tamanho de n.
- *Complexidade logarítmica.*
  - f(n) = O(log n), tipicamente, o algoritmo resolve um problema transformando-o em problemas menores.
- *Complexidade linear.*
  - f(n) = O(n), tipicamente, um pequeno trabalho é realizado sobre cada elemento de entrada.
- *Complexidade (linear e logarítmica ?).*
  - f(n) = O(n log n), tipicamente, o algoritmo resolve um problema transformando-o em problemas menores, resolvendo-os de forma independente e depois juntando as soluções.

Classes de Comportamento Assintótico

- *Complexidade quadrática.*
  - ✓ f(n) = O(n^2), itens de dados são processados aos pares; um laço dentro de outro; úteis para resolver problemas de tamanho relativamente pequenos.
- *Complexidade cúbica.*
  - ✓ f(n) = O(n^3), úteis apenas para resolver pequenos problemas.
- *Complexidade exponencial.*
  - ✓ f(n) = O(2^n), não são úteis do ponto de vista prático; uso de força bruta para se resolver problemas.
- *Complexidade fatorial*
  - ✓ f(n) = O(n!), não são úteis do ponto de vista prático; complexidade superior à complexidade exponencial; 20 ! é um número com 19 dígitos; 40 ! é um número com 48 dígitos.

Comparação de Funções de Complexidade

Função de custo	Tamanho n					
	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0.35 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,64 s	0,125 s	0.316 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13 min
2^n	0,001 s	1 s	17,9 min	12,7 dias	35,7 anos	366 séc.
3^n	0,059 s	58 min	6,5 anos	3855 séc.	10^8 séc.	10^{13} séc.

Algoritmos Polinomiais × Algoritmos Exponenciais

- *Algoritmo exponencial.*
  - Tem função de complexidade $O(c^n)$, $c > 1$.
  - São geralmente simples variações de pesquisa exaustiva.

- *Algoritmo polinomial.*
  - Tem função de complexidade $O(p(n))$, onde $p(n)$ é um polinômio.
  - São geralmente obtidos mediante entendimento mais profundo da estrutura do problema.

Algoritmos Polinomiais × Algoritmos Exponenciais

- *Comparação.*
  - A distinção entre estes dois tipos de algoritmos torna-se significativa quando o tamanho do problema a ser resolvido cresce.
  - Um problema é considerado:
    - *intratável: se não existe um algoritmo polinomial para resolvê-lo.*
    - *bem resolvido: quando existe um algoritmo polinomial para resolvê-lo.*

Encontrar menor elemento de um vetor

- *Algoritmo.*
  - 1. Comece supondo que o menor é o elemento da primeira posição.
  - 2. Compare este elemento com o elemento da segunda posição. Se o segundo for menor ele assume o lugar do primeiro.
  - 3. Repita essa operação com os n−2 itens restantes.

Encontrar menor elemento de um vetor

- *Análise*
  - ???

Verificar se um vetor contém um determinado número

- *Algoritmo*.
  - 1. Comece verificando o elemento da primeira posição.
  - 2. Se não encontrou, verifique os demais até encontrar ou até terminar o vetor.

Vetor com 4 posições (0 .. 3)

2	4	6	8
pos 0	pos 1	pos 2	pos 3

Verificar se um vetor contém um determinado número

- *Análise*
  - ✓ ???

Bibliografia

- *M. T. GOODRICH, R. TAMASSIA, Estrutura de Dados e Algoritmos em Java, 2a Ed. Bookman, 2002.*