SIN 211 - Algoritmos e Estruturas de Dados
(Análise de Algoritmos)

Prof^{o}: Joelson Antônio dos Santos

Universidade Federal de Viçosa
Instituto de Ciências Exatas e Tecnológicas
Campus de Rio Paranaíba - MG

joelsonn.santos@gmail.com
Sala: BBT 233

20 de março de 2018

Aula de Hoje

1. Análise de Algoritmos
2. Comportamento Assintótico de Funções
3. Notação Big O

Análise de Algoritmos

- Área cujo foco é o estudo do comportamento dos algoritmos.

- Nesta área existem algumas perguntas a serem respondidas:
  - Qual a necessidade de elaborar algoritmos eficientes?
  - Como avaliar o custo computacional dos algoritmos?

Análise de Algoritmos

- Diferentes algoritmos podem resolver o mesmo problema de maneiras distintas.

Análise de Algoritmos

- Diferentes algoritmos podem resolver o mesmo problema de maneiras distintas.

- Esses algoritmos não trabalham necessariamente com a mesma **eficiência** sobre o mesmo problema.

Análise de Algoritmos

- Diferenças de eficiência:
  - **Irrelevante:** para um pequeno número de elementos processado.
  - **Proporcionalmente crescente:** em relação ao número de elementos processados.

Análise Matemática

- Utiliza um modelo matemático baseado em um computador idealizado.

- Permite entender o comportamento de um algoritmo a partir de um conjunto de entradas.

  Exemplo: Dado um algoritmo de ordenação de elementos, quais os tempos de execução para entradas iguais a: 10 e 1 milhão de elementos?

Análise Matemática

- É comum considerar na análise, o custo de operações mais significativas de um determinado algoritmo.

    Exemplo: Em um algoritmo de ordenação: Pode ser considerado apenas o número de comparações entre os elementos do conjunto e ignorados operações de atribuição ou manipulação de índices, entre outras.

Função de Complexidade

- Para medir o custo de execução de um algoritmo é comum definir uma função de custo ou **função de complexidade** f.

- A **função de complexidade** pode ser representada de duas maneiras:
    - Função de **complexidade de tempo**: $f(n)$ mede o tempo necessário para executar um algoritmo submetido a uma entrada de dados de tamanho n.
    - Função de **complexidade de espaço**: $f(n)$ mede a memória necessária para executar um algoritmo a partir de uma entrada de dados de tamanho n.

Análise Matemática

- O **tempo** gasto por um algoritmo corresponde à **contagem** de instruções executadas pelo mesmo.
- Tipos de instruções:
  - Atribuição
  - Comparação
  - Incremento/Decremento
  - Operações aritméticas básicas (adição, subtração, etc).
  - Acesso à elementos de um *array*.
- Assume-se que todas as instruções citadas possuem o mesmo custo computacional.

Contagem de Instruções

- **Exemplo:** Procurar o maior valor inteiro de um vetor.

```c
#include<stdio.h>
#include<stdlib.h>

int procura_maximo(int* vetor, int n){
    int i;
    int max = vetor[0];
    for(i = 0; i < n; i++){
        if(max < vetor[i]){
            max = vetor[i];
        }
    }
  return max;
}
```

Contagem de Instruções

- **Exemplo:** Contagem de instruções do algoritmo **procura_maximo(...)**
- Linha 6: 1 atribuição
- Linha 7: 1 atribuição
- Linha 7: 2 instruções ($\{n+1\}$ comparações) e ($\{n\}$ incrementos)
- Linha 8: $\{n\}$ comparações
- Linha 9: $\{n\}$ (**pior caso**) e 0 (**melhor caso**) atribuições

Contagem de Instruções (Cont.)

- É comum considerar o **pior caso** ao analisar qualquer algoritmo.

- Logo, ao fim da contagem de instruções do algoritmo **procura_maximo(...)**, tem-se a função de complexidade $f(n) = 4n + 3$ para o **pior caso**.

Melhor Caso, Pior Caso e Caso Médio

- **Pior caso:** maior tempo de execução sobre todas as entradas de tamanho n.
- **Melhor caso:** menor tempo de execução sobre todas as entradas de tamanho n.
- **Caso médio** (ou caso esperado): média dos tempos de execução de todas as entradas de tamanho n.

Melhor Caso, Pior Caso e Caso Médio (Cont.)

- Na análise de **caso médio**, uma distribuição de probabilidades sobre o conjunto de entradas de tamanho n é suposta e o custo médio é obtido a partir dessa distribuição.
- A análise de **caso médio** é geralmente muito mais difícil de obter/ocorrer do que as análises de melhor e pior caso.
- Portanto, é comum assumir uma distribuição de probabilidades em que todas as entradas possíveis são igualmente prováveis.

Exemplo

- Procurar o maior e menor elemento de um vetor.

```
void maxMin(int* vetor, int* max, int* min, int n){
      int i;
      *max = vetor[0];
      *min = vetor[0];
      for(i = 1; i < n; i++){
         if(*max < vetor[i]){
            *max = vetor[i];
         }else if(*min > vetor[i]){
            *min = vetor[i];
         }
      }
}
```

Se considerarmos função de custo $f(n)$ o número de comparações entre elementos do vetor neste algoritmo, temos?

Exemplo (cont.)

- **Melhor caso:** quando os elementos estão em ordem crescente.
  - $f(n) = n - 1$
- **Pior caso:** quando os elementos estão em ordem decrescente.
  - $f(n) = 2(n - 1)$
- **Caso médio:** o vetor[i] é maior do que *máximo* metade das vezes.
  - $f(n) = \frac{(n-1)+2(n-1)}{2} = \frac{3n-3}{2}$, para $n > 0$.

Comportamento Assintótico de Funções

- O parâmetro n fornece uma medida da dificuldade para se resolver um problema.

- Para valores relativamente pequenos de n, qualquer algoritmo custa pouco para ser executado, mesmo os ineficientes.

- Portanto, a **escolha do algoritmo** não é problema crítico para problemas de tamanho pequeno.

Comportamento Assintótico de Funções

- Logo, a análise de algoritmos está relacionada a valores "grandes" de n. Ou seja, o estudo do comportamento assintótico das **funções de custo**.

- O **comportamento assintótico** de $f(n)$ representa o limite do comportamento do custo quando n cresce.

- Neste contexto, é importante considerarmos todos os termos de $f(n)$ para obter o custo do algoritmo?

Comportamento Assintótico de Funções

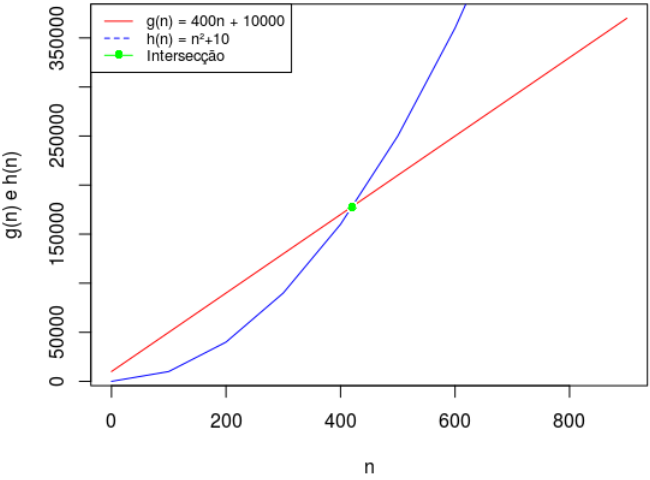
- Considere dois algoritmos diferentes para a solução de um problema **A**. As funções abaixo definem o custo de cada algoritmo.

  - **Algoritmo 1:** $g(n) = 400n + 10000$

  - **Algoritmo 2:** $h(n) = n^2 + 10$

Comportamento Assintótico de Funções

- Qual algoritmo possui menor custo se considerarmos que o valor de *n* cresce?



Comportamento Assintótico de Funções

- Apenas os termos de **maior** crescimento devem ser mantidos para a obtenção do comportamento assintótico de um algoritmo.

- Podemos definir que o maior termo domina o comportamento da função.

Comportamento Assintótico de Funções

- Logo, para determinar o maior valor de um vetor (primeiro exemplo), pode-se dizer que:
  - $f(n) = 4n + 3 \rightarrow f(n) = n$

- O comportamento assintótico é o melhor para todas as entradas?

Notação *Big O*

- Mais **conhecida** e mais **utilizada**.
- Utilizada para representar o custo no pior caso para todas as entradas n.
- **Limite superior** do custo, baseando-se na entrada.

Notação *Big O*

- Podemos dizer que uma função $g(n)$ é $O(f(n))$ se existem duas constantes positivas c e m tais que:
  - $g(n) \leq c \times f(n)$, para $n \geq m$

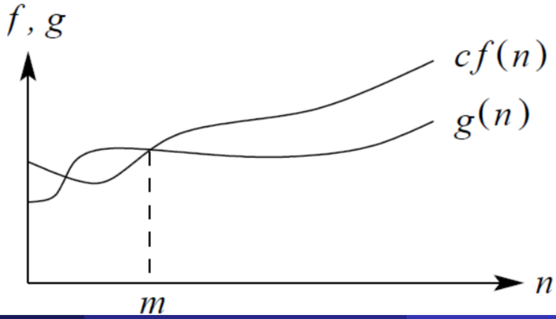
Exemplo:

- $g(n) = (n+1)^2 \rightarrow g(n)$ é $O(n^2)$, para $m = 1$ e $c = 4$.
- Isso porque $(n+1)^2 \leq 4n^2$, para $n \geq 1$.

Notação *Big O*

- A função $f(n)$ **domina assintoticamente** $g(n)$, quando esta possui duas constantes m e c, tais que, para $n \geq m$, temos $g(n) \leq c \times f(n)$.

- $g(n)$ está limitado ao $c \times f(n)$, ou seja, o maior valor que $g(n)$ pode assumir é $c \times f(n)$.



Notação *Big O*

- Considerando o trecho de código:

```
for(i = 0; i < n; i++){
    for(j = 0; j < n; j++){
        // instruções...
    }
}
```

- Como podemos contabilizar o número de instruções?
- Qual seria a notação *Big O* para esse trecho?

- $f(n) = O(1)$: Complexidade constante
  - O algoritmo não depende do tamanho da entrada n. Todas as execuções são executadas um número fixo de vezes.

Classes de Crescimento Assintótico

- $f(n) = O(logn)$: Complexidade logarítmica.
  - Divide o problema em problemas menores.
  - Algoritmos com essa complexidade são bons por conseguirem resolver problemas grandes.

  Exemplo: $n = 1000 = log_2 n \approx 10$

Classes de Crescimento Assintótico

- $f(n) = O(n)$: Complexidade linear.
  - Trabalho realizado sobre cada elemento da entrada.

  - Melhor situação para algoritmos que tem que processar n elementos ou produzir n elementos de saída.

Classes de Crescimento Assintótico

- $f(n) = O(nlogn)$: Complexidade linearítmica.
  - Ocorre geralmente em programas do tipo divisão e conquista.
  - Ocorre tipicamente em algoritmos que resolvem um problema quebrando-o em problemas menores, resolvendo-os e depois juntando suas soluções.
  - Um algoritmo eficiente de ordenação com complexidade média $(nlogn)$ é o **Quicksort**.

Classes de Crescimento Assintótico

- $f(n) = O(n^2)$: Complexidade quadrática.
  - Ocorrem quando os itens de dados são processados aos pares, muitas vezes em um anel dentro de outro.
  - Quando n é mil, o número de operações é da ordem de 1 milhão.
  - Sempre que n dobra, o tempo de execução é multiplicado por 4.
  - Úteis para resolver problemas de tamanhos relativamente pequenos.

- $f(n) = O(n^3)$: Complexidade cúbica.
  - Úteis apenas para resolver pequenos problemas.
  - Quando n é 100, o número de operações é da ordem de 1 milhão.
  - Sempre que n dobra, o tempo de execução fica multiplicado por 8.

Classes de Crescimento Assintótico

- $f(n) = O(2^n)$: Complexidade exponencial.
  - Um algoritmo de complexidade $O(2^n)$ é dito ter complexidade exponencial.
  - Geralmente não são úteis sob o ponto de vista prático.
  - Ocorrem na solução de problemas quando se usa força bruta para resolvê-los.
  - Quando n é 20, o tempo de execução é cerca de 1 milhão. Quando n dobra, o tempo fica elevado ao quadrado.

Classes de Crescimento Assintótico

- $f(n) = O(n!)$: Complexidade exponencial.
  - Piores do que $O(2^n)$
  - Tentam todas as possibilidades para problemas de otimização combinatória.
  - Quando $n = 20$, o tempo de execução é de cerca de 2432902008176640000 ou aproximadamente 2.4×10^18.

Exercício

- Desenvolva um algoritmo que receba como entrada N números inteiros. O algoritmo deve ter uma função que retorne o número de triplas cuja soma é exatamente zero.
    - **Caso teste:**
        - **Entrada:** $\{30, -30, -20, -10, 40, 0, 10, 5\}$
        - **Resultado:** Resultado: "A entrada possui 4 triplas."
- Qual a ordem de grandeza do algoritmo criado?

Bibliografia Básica

- ZIVIANI, Nivio. Projeto de algoritmos: implementações em Java e C++. *Thomsom Learning*. 2006.
  - Capítulo 1