Técnicas de programação para Games
Aula02
Bases Matemáticas

Professor Mestre: Adilson Lopes Khouri

4 de novembro de 2018

Sumário

Revisão Matemática

Análise de Algoritmos

Cronograma

Aula	Conteúdo
12/04/2018	XP e banco de dados
17/04/2018	Introdução de estruturas de dados
24/04/2018	Arrays / Matrizes e Ordenação
26/04/2018	Recursão
03/05/2018	Lista Ligada
08/05/2018	Pilha, Fila
10/05/2018	Hash
15/05/2018	Árvore Binária
17/05/2018	Heap
22/05/2018	Grafos
24/05/2018	Prova

Revisão Matemática

- Para compararmos/construirmos algoritmos precisamos de uma base matemática.
- As notações assintóticas exigem algumas operações algébricas e conhecimento geral de funções que serão revistas nessa aula.
- A revisão será prática sem formalismos desnecessários (foco no uso)

Logaritmos

- $\log_a b \Leftrightarrow a^x = b$ onde x é a resposta do logaritmo
- Sendo $b, a > 0$ e $a \neq 1$

Logaritmos - Propriedades mais usadas

- $\log_a 1 = 0$
- $\log_a a = 1$
- $\log_a a^m = m$
- $a^{\log_a b} = b$
- $\log_a b = \log_a c \Leftrightarrow b = c$

Logaritmos - Exercícios

Encontre o valor de x:

- $\log_3 27 = x$
- $\log_{81} x = \frac{3}{4}$
- $\log_4 \sqrt[2]{2} = x$
- $\log_{81} x = \frac{3}{4}$
- $\log_x 8 = 2$
- $\log_4(2x - 1) = \frac{1}{2}$
- $\log_{18} 18 = x$
- $\log_x 1024 = 2$
- $\log_4 0,25 = x$
- $\log_{10} 0,01 = x$

Exponenciação

- Um número é multiplicado por ele mesmo n vezes.
- $a^n = b$ onde a é a base, b o expoente e b é a potência

Exponenciação - Propriedades mais usadas

- $a^0 = 1$
- $a^1 = a$
- $1^n = 1$
- $0^n = 0$ para $n > 0$
- $a^m * a^n = a^{m+n}$
- $\frac{a^m}{a^n} = a^{m-n}$
- $a^{-m} = \left(\frac{1}{a}\right)^m$ para $a \neq 0$
- $(a * b * c)^n = (a)^n * (b)^n * (c)^n$
- $\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$
- $((a)^m)^n = (a)^{m*n}$
- $(\sqrt[n]{a})^m = \sqrt[n]{a^m}$
- $(-3)^2 = 9$
- $-3^2 = -9$

Exponenciação - Exercícios

Calcule:

- 2^3
- $(-2)^3$
- -2^3
- $(0.2)^4$
- $(0.1)^3$
- $(0.2)^3 + (0.16)^2$
- $(5)^{3a} = 64$ então 5^{-a} é?

Radiciação

- É o a operação inversa da potenciação
- $\sqrt[n]{x} = L$ onde L é a raiz n-ésima de x

Radiciação - Propriedades mais usadas

- $\sqrt[n]{x^n} = x$
- $\sqrt[n]{x^m} = \sqrt[np]{x^{mp}}$
- $\sqrt[n]{x^m} = \sqrt[\frac{n}{p}]{x^{\frac{m}{p}}}$
- $\sqrt[n]{a * b} = \sqrt[n]{a} * \sqrt[n]{b}$
- $\sqrt[n]{\frac{a}{b}} = \frac{\sqrt[n]{a}}{\sqrt[n]{b}}$
- $\sqrt[n]{\sqrt[m]{a}} = \sqrt[nm]{a}$
- $\sqrt[n]{a^m} = a^{\frac{m}{n}}$

Radiciação - Exercícios

Resolva a expressão:

- $\sqrt[3]{2(\sqrt[2]{9}+2*\sqrt[2]{25}+1)}$
- $\sqrt[2]{2}*(\sqrt[2]{8}+2\sqrt[2]{6})-\sqrt[2]{3}(\sqrt[2]{27}+3\sqrt[2]{6})$
- $\frac{5\sqrt[12]{64}-\sqrt[2]{18}}{\sqrt[2]{50}-\sqrt[4]{324}}$

Fatoração

- É quando agrupamos fatores comuns em evidência
- $ax + bx = x(a + b)$
- $cx^2 + bx = x(cx + b)$
- $ax + bx + ay + by = (x + y)(a + b)$

Fatoração - Exercícios

Simplifique:

- $x^2 + 14x + 49$
- $x^2 - 14x + 49$
- $\frac{(x^2+14x+49)(x^2-49)}{x^2-14x+49}$

Como construir algoritmos?

- Podemos implementar o primeiro pensamento que vier em nossas mentes...
- Como saber se esse pensamento foi 'bom' como comparar dois algoritmos que resolvem o mesmo problema e determinar que o primeiro é melhor que o segundo de forma sistemática?
- Sugestões?

Como comparar algoritmos?

- Precisamos estudar como algoritmos se comportam no tempo (velocidade de processamento) e no espaço (memória gasta)
- Um exemplo clássico é pesquisar um número dentro de um conjunto de números de forma sistemática
- Podemos usar uma busca sequencial (que tem um custo de execução linear no pior caso, usamos a notação O(n)[lê-se big Oh de 'n'])
- Podemos usar uma busca binária (que tem custo temporal, no pior caso, logarítmica O(log n))
- Obs: $\log n < n$

Como comparar algoritmos no tempo?

- Devemos contabilizar o tempo gasto no computador para executar dois algoritmos distintos? Por exemplo, rodamos a busca binária e a sequencial na mesma máquina, contabiliza o tempo e compara? Seria boa essa abordagem?
- Ou podemos contar o número de operações que um algoritmo realiza e encontrar um função que relacione esse número com o tamanho da entrada do algoritmo.

Como comparar algoritmos no tempo?

- Devemos contabilizar o tempo gasto no computador para executar dois algoritmos distintos? Por exemplo, rodamos a busca binária e a sequencial na mesma máquina, contabiliza o tempo e compara? Seria boa essa abordagem?
- Ou podemos contar o número de operações que um algoritmo realiza e encontrar um função que relacione esse número com o tamanho da entrada do algoritmo.

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
$i = j - 1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

Figura: Análise do Insertion Sort [1]

Uma aproximação do tempo de execução

- Para valores grandes de entrada, quando n tende ao infinito, por exemplo. As constantes e termos de baixa ordem podem ser ignoradas.
- Uma proposta de aproximação é a notação assintótica que usa o termo de mais alta ordem para determinar o melhor, pior e caso médio de um algoritmo.
- Notação assintótica não depende da máquina onde o algoritmo vai executar apenas da contagem de passos internos do algoritmo. Vamos analisar o InsertionSort com a notação assintótica.

Notação assintótica

- Não se assuste! É mais simples do que parece!
- Pior caso: $O(g(n))$ existem constantes positivas c, n_0 tal que $0 \leq f(n) \leq cg(n)$ para todos $n \geq n_0$
- Melhor caso: $\Omega(g(n))$ existem constantes positivas c, n_0 tal que $0 \leq cg(n) \leq f(n)$ para todos $n \geq n_0$
- Caso médio: $\Theta(g(n))$ existem constantes positivas c_1, c_2, n_0 tal que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todos $n \geq n_0$

Notação Assintótica

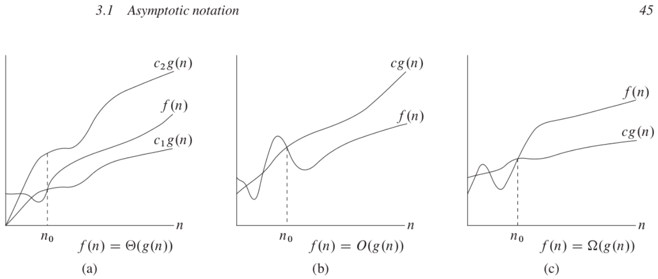

Figura: Notação Assintótica [1]

Análise Busca Sequencial

```
public int sequencialSearch(int arrayA[], int chave){

int indice = -1;
 for (int i = 0; i < arrayA.length; i++) {

 if(chave ==  arrayA[i]){
 indice = i;
 break;
 }
}
return indice;
}
```

Notação assintótica

- Qual o melhor caso?
- Qual o pior caso?
- Qual o caso médio?

Análise encontra mínimo

```
public int encontraMinimo(int arrayA[]){

int minimo = arrayA[0];
for (int i = 1; i < arrayA.length; i++) {

 if(minimo < arrayA[i]){
         minimo = arrayA[i];
 }
}

return minimo;
}
```

Dúvidas...

Alguma dúvida?

Contato

- E-mail: $0800dirso@gmail.com$ (alunos SENAC)
- E-mail: $adilson.khouri.usp@gmail.com$
- Phone: $+55119444-26191$
- Linkedin
- Lattes
- GitHub

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[2] A. V. Aho, J. E. Hopcroft, and J. Ullman, *Data Structures and Algorithms*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983.

[3] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[4] Beck, *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[5] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999.

[6] Geeks. (2018) A computer science portal for geeks. [Online]. Available: https://www.geeksforgeeks.org