Princípios de Análise de Algoritmos

Marcelo K. Albertini

10 de Março de 2020

Aula de hoje

Nesta aula veremos:

- Apresentação da disciplina
- Tipo abstrato de dados
- Princípios de Análise de Algoritmos

A disciplina GSI011 - Estruturas de Dados 2

- Prof. Marcelo Keese Albertini
- Site: `http://www.facom.ufu.br/~albertini/ed2`

Aulas GSI011

- Objetivo

  "Aplicar algoritmos e estrutura de dados adequados à solução eficiente de problemas."
- Aulas
  - Teóricas: exposição de conceitos
  - Práticas: resolução de problemas

Conteúdo segundo a ementa

- Princípios de Análise de Algoritmos
- Recursão: Algoritmos recursivos
- Programação Dinâmica
- Ordenação
- Grafos
- Árvores
- Busca: Texto, Linear, Binária
- Árvore de busca binária (ABB)
- Hashing.
- Coleta de Lixo (garbage collection).

Avaliação

- Avaliação teórica: 60 pontos divididos em 3 provas (20 + 20 + 20 pontos) + 1 substitutiva (conteúdo completo e substitui prova com nota mais baixa)
- Avaliação prática: 20 pontos de aulas práticas (pontos divididos igualmente pelo número de práticas concluídas **durante a aula**)
- Avaliação prática: 20 pontos de 2 trabalhos

Tipo Abstrato de Dados (TAD) – *Abstract Data Types* (ADT)

- **Definição:** TAD é uma coleção de dados e as operações que operam neles.
  - TAD $\approx$ dados $+$ algoritmos que implementam operações
  - TAD $\approx$ classe em uma linguagem orientada a objetos
- **Exemplos:**
  - Queue, Stack, Set, List (Estruturas de Dados 1)
  - Map, Graph (Estrutura de Dados 2)

Exemplo TAD: `Stack`

- Uma pilha de números inteiros: `Stack<Integer>`
- **Dados:** números inteiros
- **Operações:**
  - `void push(Integer i)` : *coloca inteiro no topo da pilha*
  - `Integer pop()`: *remove inteiro do topo da pilha e retorna esse valor*
  - `Integer peek()`: *ler mas não remover topo da pilha*

Exemplo TAD: `Queue`

- Uma fila de Object: `Queue<Object>`
- **Dados:** objetos do tipo `Object`
- **Operações:**
  - `enqueue(Object o)`: *insere o no fim da fila*
  - `Object dequeue()`: *remove e retorna objeto no começo da fila*

Exemplo TAD: `Set`

- Um conjunto de Dado: `Set<Dado>`
- **Dados:** elementos sem ordem definida e sem elementos repetidos
- **Operações:**
  - `void add(Dado d)`:*insere d no conjunto sem criar repetições*
  - `boolean contains(Dado d)`: *verifica se d está no conjunto*
  - `boolean remove(Dado d)`: *remove d do conjunto e retorna se foi bem sucedido*
  - `Iterator<Dado> iterator()`: *retorna um iterador para percorrer os elementos do conjunto*

Exemplo TAD: `List`

- Uma fila de Object: `List<Element>`
- **Dados:** objetos do tipo `Element`
- **Operações:**
  - `put(Element e, int i)`: *insere e na posição i da lista*
  - `get(int i)`: *retorno Element na posição i da lista*
  - `remove(int i)`: *remove e retorna elemento na posição i da lista*

Exemplo TAD: `Map`

- Tabela de associações (Dicionário)
- **Dados:** pares de chaves e valores
- **Operações:**
  - `void put(Key k, Value v`: *insere associação entre k e v, substituindo v se preciso*
  - `Value get(Key k)`: *obtém valor associado à chave k*

Exemplo TAD: `Graph`

- Representa relações entre entidades
- **Dados:** pares de chaves e valores
- **Operações:**
  - `boolean adjacent(int u, int v)`: *retorna se entidades u e v estão conectadas*
  - `Iterator neighbours(int u)`: *obtém iterador dos vizinhos de u*
  - ...

Estruturas de Dados

- Estruturas de Dados são usadas para implementar TADs

  "EDs são o motor de TADS"
- Uma **estrutura de dados** de lista ligada pode ser usada para implementar (partes de):
  - Queue, Set, List, Graph

Análise de Algoritmos

Objetivo da Análise de Algoritmos

Compreender os custos de execução de um algoritmo.

Fruto da análise de um algoritmo

Produto frequente é um modelo matemático, equação, que permite:

- predizer o tempo de execução;
- comparar com outros algoritmos;
- melhorar o algoritmo etc.

Custo de operações

Custos concretos

- Quantos bytes de memória o algoritmo vai consumir?
- Quanto segundos levar o algoritmo?
- Quantos Watts da bateria vai ser gasto?
- Quantas comparações entre números serão necessários?

Custos abstratos

- Quantas comparações entre números serão necessários?
- Quantas transmissões em rede serão usadas?
- Qual é o número mínimo de acessos a disco feito?

Exemplo: avaliação de custos em um algoritmo

```
// ler inteiro positivo "n"...
if (n > 10)
    System.out.println("Isto pode demorar...");

for (int i = 1; i < 10; i++)
    for (int j = 1; j < i; j++)
        System.out.println(i * j);
```

Exemplo de Análise: Ordenação

- Problema de ordenação
  - Como ordenar cartas de baralho?
  - Como ordenar depósitos em um banco?
- Perguntas:
  - Como estimar a duração?
  - Qual é o modelo de aumento de custo com o aumento de entradas?
  - Como extrapolar o resultado?

Definição formal de um problema

Na definição de um problema são necessários

- **Entrada**: descrição e formato de dados fornecidos ao algoritmo
- **Saída**: condições estabelecidas ao dados que o algoritmo deve produzir

Instância de um problema

Uma instância do problema é definição concreta dos dados de um problema. Podemos escolher uma característica dos dados de **entrada** para definir o tamanho da instância do problema. Em um problema de ordenação, o tamanho da instância de um problema pode ser a quantidade de elementos a serem ordenados N.

Exemplo: problema de ordenação crescente

Ordenação de números

- Entrada: uma sequência de n números $a_1, a_2, a_3, \ldots, a_n$
  - Neste problema, a quantidade n de número representa o tamanho da instância
- Saída: uma reordenação $b_1, b_2, \ldots, b_n$, tal que $b_i < b_j$ se $i < j$

A solução: um algoritmo

- correto: se para cada instância de entrada, o algoritmo conclui com a saída correta
- se o é algoritmo é correto então ele resolve o problema

Análise empírica de algoritmos

Custos

- Quanta memória o algoritmo vai usar?
- Quanto tempo o algoritmo vai levar?

Como atribuir custo com estimativas de tempo de operações

- uniforme - todas as operações tem o mesmo custo fixo T
  - atribuições de valores, cópias de variáveis, chamadas de funções
  - operações lógico-aritméticas
- proporcional - operações tem custo proporcional ao número de bits sendo operados

Avaliação empírica

A avaliação empírica permite

- avaliação do desempenho em uma determinada configuração de computador/linguagem
- considerar custos não aparentes: custo de alocação de memória, sobrecarga de chamada de funções
- comparar computadores
- comparar linguagens
- reconhecer modelo de desempenho para casos difíceis de avaliação matemática de algoritmos

Avaliação empírica

Dificuldades

- necessário implementar o algoritmo
- resultados dependem do computador utilizado e de eventos ocorridos no momento de avaliação
- difícil de generalizar

Avaliação matemática

A avaliação matemática

- uso de algoritmo em um **modelo computacional teórico**
  - Exemplo de modelo: “Máquina de Acesso Aleatório”
- simplificação para considerar somente os custos dominantes

Exemplo: avaliação de custos em um algoritmo

```java
// Entrada: inteiro positivo n
// Saida: imprimir multiplicacoes de 0 a n
if (n > 10)
    System.out.println("Isto pode demorar...");

for (int i = 0; i < n; i++)
    for (int j = 0; j < i; j++)
        System.out.println(i * j);
```

Exemplo: avaliação de custos em um algoritmo

```
// Entrada: inteiro positivo "n"
// Saida: imprimir multiplicacoes de 0 a "n"
if (n > 10)    // 1 vez, custo $c_1$
 System.out.println("Isto pode demorar..."); $c_2$

for (int i = 0; i < n; i++)  // n vezes,  $c_3 + nc_1 + nc_4$
 for (int j = 0; j < i; j++)  //  $m = \sum_{j=1}^{n} j$ vezes,
    $c_3 + mc_1 + mc_4$
  System.out.println(i * j); // $c_2 + c_5$
```

Custos

comparação c_1, impressão c_2, atribuição c_3, incremento c_4, multiplicação c_5

Exemplo: avaliação de custos em um algoritmo

```
// Entrada: inteiro positivo "n"
// Saida: imprimir multiplicacoes de 0 a "n"
if (n > 10)  // 1 vez, custo c1
 System.out.println("Isto pode demorar..."); c2

for (int i = 0; i < n; i++)  // n vezes,  c3 + nc1 + nc4
 for (int j = 0; j < i; j++)  //  m = ∑_{j=1}^{n} j vezes,
    c3 + mc1 + mc4
  System.out.println(i * j); // c2 + c5
```

Custos: pior caso – quando o algoritmo executar tudo

$$T(n) = c_1 + c_2 + (c_3 + nc_1 + nc_4) + (c_3 + mc_1 + mc_4) + m * (c_2 + c_5)$$

Exemplo: avaliação de custos em um algoritmo

Custos: pior caso – quando o algoritmo executar tudo

- $T(n) = c_1 + c_2 + (c_3 + nc_1 + nc_4) + (c_3 + mc_1 + mc_4) + m * (c_2 + c_5))$
- se todos custos forem considerados iguais a "c":
- $T(n) = c + c + (c + nc + nc) + (c + mc + mc) + m * (c + c))$
- $T(n) = 2c + (c + 2nc) + (c + 2mc) + 2mc$
- $T(n) = 4c + 2nc + 4mc$

Exemplo: avaliação de custos em um algoritmo

Custo do algoritmo é

$T(n) = 4c + 2nc + 4mc$

Quanto é m ?

$m = \sum_{j=1}^{n} j =$

- $1 + 2 + 3 + 4 + 5 + \ldots + (n-3) + (n-2) + (n-1) + n =$
- $(n+1) + (2 + (n-1)) + (3 + (n-2)) + \ldots + (n+1)$
- Somamos $n/2$ pares de números $(n+1)$
    - $m = \frac{n}{2}(n+1)$

Custo do algoritmo é

$T(n) = 4c + 2nc + 4(\frac{n}{2}(n+1))c$

Exemplo: avaliação de custos em um algoritmo

Custo do algoritmo é

- $T(n) = 4c + 2nc + 4(\frac{n}{2}(n+1))c$
- $T(n) = 4c + 2nc + 2n(n+1)c$
- $T(n) = 4c + 2nc + 2n^2c + 2nc$
- $T(n) = 4c + 4cn + 2cn^2$

Qual é o custo dominante do algoritmo?

Complexidade de tempo

Custo do algoritmo de multiplicação de números

A função que descreve o custo do nosso algoritmo é $T(n) = 4c + 4cn + 2cn^2$.

Complexidade de tempo

Para descrever a complexidade usamos somente o custo dominante: $2cn^2$.

Porquê?

Existe um número $n = a$ a partir do qual o custo $2ca^2$ é sempre superior a $4ca$, tornando-o pouco importante.

Ordem de crescimento

Dizemos que a complexidade de *pior caso* do nosso algoritmo é $O(n^2)$.

Exemplo: ordenação de transações bancárias

Problema

- Um banco tem 10 milhões clientes.
- Cada cliente faz em média 4 transações diárias.

Relatório

O banco precisa entregar um relatório diário para o Banco Central, com uma relação ordenada de todas as transações.

Infraestrutura

O computador do banco é dedicado ao problema e resolve operações lógico-aritméticas na taxa de 2^{30} (~ 1 bilhão) operações por segundo.

Exemplo: o problema da ordenação interna

ordenar em memória

- Pré-condições: vetor em memória principal, inicializado com elementos
- Pós-condições: vetor com elementos em ordem crescente (ou decrescente)

Posições no vetor

Seis elementos

0	1	2	3	4	5
3	1	6	2	4	8

Como fica ordenado?

0	1	2	3	4	5
1	2	3	4	6	8

Exemplo: análise de um algoritmo de ordenação

Definição de ordenação

Sequência de comparações e trocas de posição entre elementos para obter vetor ordenado.

Análise

Quantas trocas, comparações (complexidade de tempo) e variáveis auxiliares (de espaço) são necessárias?

Exemplo: algoritmo de ordenação interna bubblesort

Como programar um algoritmo de ordenação simples?

Ideia

Comparar pares consecutivos de elementos e trocá-los de posição caso o primeiro seja maior que o segundo.

```
if (vetor[i] > vetor[i+1]) {
    aux = vetor[i];
    vetor[i] = vetor[i+1];
    vetor[i+1] = aux;
}
```

Variável auxiliar aux é custo extra de memória .

Exemplo: algoritmo bubblesort simplificado

"Um dos piores algoritmos de ordenação desde o tempo dos Faraós."

```
bubblesort (int vetor[], int nelem) {
    int i, iteracao, aux;

    /*controle do numero de iteracoes (n - 1)*/
    for (iteracao = 0; iteracao < nelem-1; iteracao++){
        /*repeticao interna, percorre vetor (n - 1)*/
        for (i = 0; i < nelem - 1; i++){
            if (vetor[i] > vetor[i+1]){
                /*necessaria uma troca*/
                aux = vetor[i];
                vetor[i] = vetor[i+1];
                vetor[i+1] = aux;
            }
        }
    }
}
```

Exemplo: análise de uma solução para o problema de ordenação de transações

Se o algoritmo é o Bubblesort, o banco:

- a) conseguirá tranquilamente fazer o relatório a tempo;
- b) deverá comprar um computador melhor;
- c) deverá contratar um programador melhor.

Exemplo: aplicação de análise de algoritmos ao caso bancário

Avaliação:

- Ordem do Bubblesort: $O(n^2)$
- Elementos a ordenar: número de transações $n = 4 \times 10^7$
- Operações necessárias: $(4 \times 10^7)^2$
- Capacidade do computador: 2^{30} operações por segundo
- Tempo necessário em minutos: $(4 \times 10^7)^2/2^{30}$

$=$ 17 dias, 4 horas, 55 minutos e 16 segundos

Notação de complexidade

Função de complexidade (de tempo ou espaço)

Para um problema de tamanho n, buscamos uma função $f(n)$ que descreve o componente principal do nosso custo

Conflito de interesses

Frequentemente, quando a complexidade de tempo diminui, a complexidade de espaço aumenta

Notação assintótica: formal

Limitante inferior $\Omega(f(n))$

O custo de um algoritmo se torna sempre **maior ou igual** que $c \cdot f(n)$, para uma constante arbitrária $c > 0$, para todas as instâncias de um problema de tamanho $n > n_0$ a partir de algum $n_0 > 0$.

Limitante superior $O(f(n))$

O custo de um algoritmo se torna sempre **menor ou igual** que $c \cdot f(n)$, para uma constante arbitrária $c > 0$, para todas as instâncias de um problema de tamanho $n > n_0$ a partir de algum $n_0 > 0$.

Notação Θ

Uma função $t(n)$ é $\Theta(f(n))$ se, e somente se, $t(n) = O(f(n))$ e também $t(n) = \Omega(f(n))$.

Exemplos de uso de notação assintótica

Casos

- Para "o custo pelo menos é...", usar notação Ω
  - Se $t(n) = 2$ operações, então $t(n) = \Omega(1)$
    i.e., $t(n) \geq c * 1$ para $\mathbf{n} \geq \mathbf{1}$ e $\mathbf{c} = \mathbf{2}$
- Para "o custo é até ... ", usar notação O
  - Se $t(n) = 2n + 1$ operações
    então $t(n) = O(n)$, i.e., $t(n) \leq c * n$ para $\mathbf{c} = \mathbf{3}$ e $\mathbf{n} \geq \mathbf{1}$
- Para "o custo cresce na ordem de magnitude de ... ", usar notação Θ:
  - Se $t(n) = 3n^2 - n + 1$,
    então $t(n) = \Theta(n^2)$
    pois $t(n) = O(n^2)$, $t(n) \geq c_1 n^2$ e $t(n) \leq c_2 n^2$ para $\mathbf{c} = \mathbf{3}$ e $\mathbf{n} \geq \mathbf{1}$

Exemplos: avaliação assintótica

Exemplo 1

- $t(n) = n^2 + n + 2$
- $t(n) = O(n^3)$

  pois com $n^2 + n + 2 \leq cn^3$ para $\mathbf{c} = \mathbf{4}$ e $\mathbf{n} \geq \mathbf{1}$

Exemplo 2

- $f(n) = n + log(n) + 1$
- $f(n) = O(n)$

  pois $n + log(n) + 1 \leq cn$ para $\mathbf{c} = \mathbf{3}$ e $\mathbf{n} \geq \mathbf{1}$

Exemplo 3

- $f(n) = n! + n^2$
- $f(n) = O(n!)$

  pois $n! + n^2 \leq cn!$ para $\mathbf{c} = \mathbf{2}$ e $\mathbf{n} \geq \mathbf{1}$

Exemplo: o problema de busca em um vetor

Problema

Verificar se um dado número está em um vetor.
Entrada: vetor com n números.
Saída: Afirmativo, se número buscado está entre os n números do vetor. Negativo, caso contrário.

Algoritmo

Comparar cada número do vetor com o número buscado. Dar a resposta afirmativa assim que encontrá-lo.

- Seja $t(n)$ a função que descreve o custo desse algoritmo.
- Qual é a função $f(n)$ tal que esse algoritmo é $t(n) = O(f(n))$?
- Qual é a função $f(n)$ tal que esse algoritmo é $t(n) = \Omega(f(n))$?

Orientações

- Um dos problemas em usar somente notação assintótica é desprezar constantes
  - $f(n) = 100000000 * n + 999999999$
  - $f(n) = O(n)$
- Desprezar termos de menor grandeza

Atenção

Complexidade assintótica menor não indica que o algoritmo sempre é o melhor. Se as constantes forem significativas, na prática é melhor considerá-las.

Classes de problemas

$$O(1) \in O(log(n)) \in O(n) \in O(nlog(n)) \in O(n^2) \in O(n^3) \in O(2^n)$$

- $O(1)$: ordem constante
- $O(log(n))$: ordem logaritmica
- $O(n)$: ordem linear
- $O(nlog(n))$: ordem típica de soluções em "pedaços", e.g., árvores
- $O(n^2)$: ordem quadrática. "um laço dentro de outro"
- $O(2^n)$: ordem exponencial. Na maior parte das vezes, impraticável.
- $O(n!)$: ordem fatorial. Pior que exponencial.

Exercício: maior valor de um vetor

Qual é a função que descreve o custo e a ordem de crescimento de pior caso do seguinte algoritmo?

```
// Entrada: vetor com inteiros
// Saida: maior valor do vetor
int max (int a[]){
    int i, maior;
    maior = a[0];

    for (i = 1; i < a.length; i++){
        if (maior < a[i]) {
            maior = a[i];
        }
    }
    return(maior);
}
```