Complexidade de Algoritmos

Norton T. Roman

Apostila baseada nos trabalhos de Marcos Chaim e Delano M. Beder

Algoritmos

- Seqüência de passos computacionais que transformam a entrada na saída
- É dito correto se, para cada entrada possível, produz a saída correta
- Incorreto pode não parar em algumas instâncias ou dar a resposta errada
    - São até úteis, se pudermos controlar a taxa de erro

Projeto de Algoritmos

- Algoritmos são projetados para resolver problemas
  - Em geral, a especificação do problema especifica em termos gerais a relação entrada/saída.
- Ex:
  - Problema: encontrar a melhor rota, em termos de tempo de entrega, dos produtos das Casas Ceará em Ermelino Matarazzo
  - Solução: algoritmo para descoberta da melhor rota tendo como entrada os locais de entrega

Projeto de Algoritmos

- Projetar algoritmos implica estudar o seu comportamento
  - No tempo: quanto tempo vai demorar para encontrar a solução do problema
  - No espaço: quanto de memória será necessário para encontrar a solução

Análise de Algoritmos

- Na área de análise de algoritmos há dois problemas distintos:
  - Análise de um algoritmo particular
    - Qual é o custo de usar um dado algoritmo para resolver um problema específico?
    - Quantas vezes cada parte desse algoritmo vai ser executada?
    - Quanto de memória será necessária?

Análise de Algoritmos

- Na área de análise de algoritmos há dois problemas distintos:
  - Análise de uma classe de algoritmos
    - Qual é o algoritmo de menor custo possível para resolver um problema específico?
    - Isto implica investigar toda uma família de algoritmos
      - Buscamos identificar um que seja o melhor possível
      - Ao encontrá-lo, seu custo será uma medida da dificuldade inerente para resolver problemas da mesma classe.

Análise de Algoritmos

- Na área de análise de algoritmos há dois problemas distintos:
  - Análise de uma classe de algoritmos
    - Isso significa colocar limites para a complexidade dos algoritmos:
      - Exemplo: podemos estimar o número mínimo de comparações necessárias para ordenar n números por meio de comparações sucessivas.
      - Logo, nenhum algoritmo vai fazer melhor que isto $\Rightarrow$ menor custo possível.
      - Menor custo possível $\Rightarrow$ medida de dificuldade.
    - Se o custo do algoritmo A = menor custo possível $\Rightarrow$ A é ótimo.

Complexidade

- Como medir o custo de um algoritmo?
- Como comparar o custo de vários algoritmos que resolvem um problema?
  - Medição direta do tempo de execução em um computador real.
    - Problemas: depende do compilador; depende do hardware;
    - Medidas de tempo podem se influenciadas pela memória disponível.
  - Computador ideal em que cada instrução tem seu custo determinado (solução de Donald Knuth).

Complexidade

- Como comparar o custo de vários algoritmos que resolvem um problema?
  - Considerar apenas as operações mais significativas (mais usual).
    - Ignora-se o custo de algumas operações envolvidas, como atribuições etc.
    - Exemplo: Ordenação ⇒ número de comparações.

Função de Complexidade

- Para medir o custo de execução de um algoritmo, definimos uma função de custo ou complexidade f(n):
  - f(n) é a medida do tempo ou espaço necessário para executar um algoritmo para uma entrada de tamanho n.
  - Se for de tempo, então f (n) é chamada de função de complexidade de tempo ou temporal do algoritmo.
  - Se for da memória necessária (espaço) para executar o algoritmo, então f (n) é a função de complexidade espacial.

Função de Complexidade

- Se nada for dito, entende-se f(n) como complexidade de tempo.
  - Lembre que isso não representa o tempo diretamente, mas o número de vezes que determinada operação, considerada relevante, é executada.
- Mas e isso funciona?
- Por que não condicionar apenas ao hardware, ou o tempo utilizado?
  - ???

Função de Complexidade

- Se nada for dito, entende-se f(n) como complexidade de tempo.
  - Lembre que isso não representa o tempo diretamente, mas o número de vezes que determinada operação, considerada relevante, é executada.
- Mas e isso funciona?
- Por que não condicionar apenas ao hardware, ou o tempo utilizado?
  - Porque não conseguiremos garantir isso para toda e qualquer entrada

Função de Complexidade

- Exemplo:
  - Suponha dois algoritmos com duas funções de complexidade:
    - $f_1(n) = c_1 n^2$
    - $f_2(n) = c_2 \log_{10}(n)$
  - Suponha dois computadores:
    - A: 1.000.000.000 de instruções por segundo
    - B: 10.000.000 de instruções por segundo (100 vezes mais lento)

Função de Complexidade

- Exemplo:
  - Agora suponha dois compiladores
    - Um ótimo, usado no primeiro algoritmo, resultando em uma constante $c_1 = 2$
    - Um bem pior, usado no segundo algoritmo, resultando em uma $c_2 = 50$
  - Fazemos então o algoritmo f_1 rodar no mais rápido e o f_2 no mais lento
    - f_1 não somente tem a máquina mais rápida, como o melhor compilador
    - f_2 ficou com o pior dos dois mundos

Função de Complexidade

- Exemplo:
  - Para uma entrada de 1.000 elementos:
    - $F_1 = 2 \times (1.000)^2 =$ **2.000.000** de instruções
      - 2.000.000 / 1.000.000.000 = 2 ms
    - $f_2 = 50 \times \log_{10}(1.000) =$ **150** instruções
      - 150 / 10.000.000 = 0,015 ms
  - Mesmo tendo o pior compilador e a pior máquina, para apenas 1.000 elementos já rodou ≈ 133 vezes mais rápido

Cálculo da Função de Complexidade

- Exemplo
  - Encontrar o maior elemento de um arranjo (vetor) de inteiros A

```
int maxArray(int [] A) {
  int i, max;
  max = A[0];
  for(i=1; i < A.length; i++) {
    if(max < A[i]) {
      max = A[i];
    }
  }
  return max;
}
```

Exemplo

```
int maxArray(int [] A) {
  int i, max;
  max = A[0];
  for(i=1; i < A.length; i++) {
    if(max < A[i]) {
       max = A[i];
    }
  }
  return max;
}
```

- Seja f(n) uma função de complexidade
  - f(n) é o número de comparações para um vetor A de tamanho n
- Como seria f(n)?

Exemplo

```
int maxArray(int [] A) {
  int i, max;
  max = A[0];
  for(i=1; i < A.length; i++) {
    if(max < A[i]) {
       max = A[i];
    }
  }
  return max;
}
```

- Seja f(n) uma função de complexidade
  - f(n) é o número de comparações para um vetor A de tamanho n
- Como seria f(n)?
  - $f(n) = n - 1$, para $n > 0$.
- Será que o algoritmo apresentado é ótimo?

Prova

- Teorema:
  - Qualquer algoritmo para encontrar o maior elemento de um conjunto de n elementos, $n \geq 1$, faz ao menos $n - 1$ comparações.
- Prova:
  - Cada um dos $n-1$ elementos tem que ser verificado, por meio de comparações, que é menor do que algum outro elemento. Logo, $n-1$ comparações são necessárias.
- Como maxArray() possui complexidade igual ao limite inferior de custo, então seu algoritmo é ótimo.

Função de Complexidade

- Normalmente, a medida de custo de execução depende do tamanho da entrada.
  - Mas este não é o único fato que influencia o custo.
- O tipo de entrada pode também influenciar o custo.
  - No caso de maxArray() a entrada não influencia.
- Considere um outro método para obter o máximo e o mínimo de um arranjo.

Exemplo

```
void maxArray1(int [] A) {
  int i, max, min;
  max = min = A[0];
  for(i=1; i < A.length; ++i) {
    if(max < A[i]) {
       max = A[i];
    }
    else if(A[i] < min) {
          min = A[i];
    }
  }
  System.out.print("Mínimo = " + min);
  System.out.print(", Máximo = " + max);
}
```

- Qual a função de complexidade de maxMin1?
- ????

Exemplo

```
void maxArray1(int [] A) {
  int i, max, min;
  max = min = A[0];
  for(i=1; i < A.length; ++i) {
    if(max < A[i]) {
       max = A[i];
    }
    else if(A[i] < min) {
          min = A[i];
    }
  }
  System.out.print("Mínimo = " + min);
  System.out.print(", Máximo = " + max);
}
```

- Qual a função de complexidade de maxMin1?
- Depende:
  - Se o arranjo já estiver ordenado em ordem crescente
    - ????

Exemplo

```
void maxArray1(int [] A) {
  int i, max, min;
  max = min = A[0];
  for(i=1; i < A.length; ++i) {
    if(max < A[i]) {
       max = A[i];
    }
    else if(A[i] < min) {
          min = A[i];
    }
  }
  System.out.print("Mínimo = " + min);
  System.out.print(", Máximo = " + max);
}
```

- Qual a função de complexidade de maxMin1?
- Depende:
  - Se o arranjo já estiver ordenado em ordem crescente
    - $f(n) = n - 1$

Exemplo

```
void maxArray1(int [] A) {
  int i, max, min;
  max = min = A[0];
  for(i=1; i < A.length; ++i) {
    if(max < A[i]) {
       max = A[i];
    }
    else if(A[i] < min) {
           min = A[i];
    }
  }
  System.out.print("Mínimo = " + min);
  System.out.print(", Máximo = " + max);
}
```

- Qual a função de complexidade de maxMin1?
- Depende:
  - Se o arranjo já estiver ordenado em ordem decrescente
    - ???

Exemplo

```
void maxArray1(int [] A) {
  int i, max, min;
  max = min = A[0];
  for(i=1; i < A.length; ++i) {
    if(max < A[i]) {
       max = A[i];
    }
    else if(A[i] < min) {
         min = A[i];
    }
  }
  System.out.print("Mínimo = " + min);
  System.out.print(", Máximo = " + max);
}
```

- Qual a função de complexidade de maxMin1?
- Depende:
  - Se o arranjo já estiver ordenado em ordem decrescente
    - $f(n) = 2(n - 1)$

Exemplo

```
void maxArray1(int [] A) {
  int i, max, min;
  max = min = A[0];
  for(i=1; i < A.length; ++i) {
    if(max < A[i]) {
      max = A[i];
    }
    else if(A[i] < min) {
          min = A[i];
    }
  }
  System.out.print("Mínimo = " + min);
  System.out.print(", Máximo = " + max);
}
```

- Qual a função de complexidade de maxMin1?
- Depende:
  - Se o A[i] for maior que max metade das vezes
    - ???

Exemplo

```
void maxArray1(int [] A) {
  int i, max, min;
  max = min = A[0];
  for(i=1; i < A.length; ++i) {
    if(max < A[i]) {
       max = A[i];
    }
    else if(A[i] < min) {
           min = A[i];
    }
  }
  System.out.print("Mínimo = " + min);
  System.out.print(", Máximo = " + max);
}
```

- Qual a função de complexidade de maxMin1?
- Depende:
  - Se o A[i] for maior que max metade das vezes
    - $$f(n) = \frac{(n-1)+2(n-1)}{2}$$

      $= 3n/2 - 3/2$ para $n > 0$

Função de Complexidade

- Três situações podem ser observadas:
  - Melhor caso: já ordenado crescentemente
    - Menor tempo de execução dentre as entradas de tamanho n
  - Pior caso: já ordenado decrescentemente
    - Maior tempo de execução dentre as entradas de tamanho n
  - Caso médio: um elemento A[i] tem 50% de chances de ser maior ou menor que max
    - Média dos tempos de execução de todas as entradas de tamanho n

Caso Médio

- Supõe uma distribuição de probabilidades sobre o conjunto de entradas de tamanho n.
  - O custo médio é obtido com base nessa distribuição.
- Normalmente, o caso médio é muito mais difícil de determinar do que o melhor e o pior caso.
  - Comumente, supõe-se que todas as entradas têm a mesma chance de ocorrer ⇒ equiprováveis.
  - Nem sempre isto é verdade, por isto, o caso médio é determinado apenas se fizer sentido.

Exemplo

- Busca seqüencial em um vetor de tamanho n.
- Qual o melhor caso?

```
int busca(int x, int[] A) {
    for (int i=0; i<A.length; i++) {
        if (A[i] == x)
            return(i);
    }
    return(-1);
}
```

Exemplo

- Busca seqüencial em um vetor de tamanho n.
- Qual o melhor caso?
  - O valor procurado é o primeiro → tempo constante
  - Representado como f(n) = 1

```
int busca(int x, int[] A) {
    for (int i=0; i<A.length; i++) {
        if (A[i] == x)
            return(i);
    }
    return(-1);
}
```

Exemplo

- Busca seqüencial em um vetor de tamanho n.
- Qual o pior caso?

```
int busca(int x, int[] A) {
    for (int i=0; i<A.length; i++) {
        if (A[i] == x)
            return(i);
    }
    return(-1);
}
```

Exemplo

- Busca seqüencial em um vetor de tamanho n.
- Qual o pior caso?
  - O valor procurado é o último
  - $f(n) = n$

```
int busca(int x, int[] A) {
    for (int i=0; i<A.length; i++) {
        if (A[i] == x)
            return(i);
    }
    return(-1);
}
```

Exemplo

- Melhor caso:
  - $f(n) = 1$
- Pior caso:
  - $f(n) = n$
- Caso médio:
  - ?????

```
int busca(int x, int[] A) {
    for (int i=0; i<A.length; i++) {
        if (A[i] == x)
            return(i);
    }
    return(-1);
}
```

Exemplo

- Melhor caso:
  - f(n) = 1
- Pior caso:
  - f(n) = n
- Caso médio:
  - $f(n) = \frac{1+n}{2}$

```
int busca(int x, int[] A) {
	for (int i=0; i<A.length; i++) {
		if (A[i] == x)
			return(i);
	}
	return(-1);
}
```

Função de Complexidade

- Cálculo da função de complexidade:
  - Identifique o tempo de execução de cada comando no algoritmo
  - Selecione apenas os comandos relacionados com o tamanho da entrada
  - A soma dos tempos de execução de cada um desses comandos corresponde ao tempo de execução do algoritmo

Exercícios

- Determine a função de complexidade do algoritmo de ordenação por inserção no pior caso para um vetor de tamanho n. (Ver [1] Seção 2.2, páginas 16-21)

Resumo

- Problemas requerem algoritmos que os solucione.
- Algoritmo adequado depende do seu comportamento
  - Complexidade temporal e espacial.
- Algoritmo ótimo
  - Soluciona o problema com o menor custo possível.
- Função de complexidade
  - Melhor caso, pior caso e caso médio.

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - Tradução da 2a. Edição Americana. Editora Campus, 2002.
- Nívio Ziviani. Projeto de Algoritmos com implementações em C e Pascal. Editora Thomson, 2a. Edição, 2004.