Paradigmas de Projeto de Algoritmos

Recursividade

Divisão e Conquista

Equações de Recorrência

Professora:

Fátima L. S. Nunes

SISTEMAS DE INFORMAÇÃO

EACH

Recursividade

- Última aula:
  - Conceitos de recursividade
  - Quando usar e não usar recursividade
  - Questão: como analisar complexidade de algoritmos recursivos?

Indução Matemática

- Algoritmos recursivos podem ser definidos e estudados a partir da indução matemática.
- Seja *T* um teorema a ser provado
  - Consideremos ***T*** como tendo um número natural como parâmetro (***n***)
  - Em vez de tentar provar que *T* é válido para todos valores de ***n***, basta provar duas condições:

  1. *T* é válido para $n = 1$;
  2. Para todo $n > 1$:
     - se *T* é válido para $n-1 \Rightarrow$ *T* é válido para n

SISTEMAS DE INFORMAÇÃO

EACH

Indução Matemática

- Exemplo: cálculo do fatorial de um número inteiro
- Para facilitar a indução, modificaremos um pouco o método usado anteriormente para calcular o fatorial, para que retorne uma constante, facilitando a definição do passo base:

```
int fatorial (int n)
 {
   if (n == 0)
     return 1;
   else
     return n*fatorial (n-1);
 }
```

SISTEMAS DE INFORMAÇÃO

EACH

Recursividade e indução

- Definindo o cálculo do fatorial usando indução:

  - Qual é o passo base?

```
int fatorial (int n)
 {
   if (n == 0)
     return 1;
   else
     return n*fatorial (n-1);
 }
```

Recursividade e indução

- Definindo o cálculo do fatorial usando indução:

  - Qual é o passo base?

    Se n=0, então o fatorial = 1

```
int fatorial (int n)
{
  if (n == 0)
    return 1;
  else
    return n*fatorial (n-1);
}
```

Recursividade e indução

- Definindo o cálculo do fatorial usando indução:

  - Qual é o passo base?

    Se n=0, então o fatorial = 1

  - Qual é o passo indutivo?

```
int fatorial (int n)
{
  if (n == 0)
    return 1;
  else
    return n*fatorial (n-1);
}
```

Recursividade e indução

- Definindo o cálculo do fatorial usando indução:

```
int fatorial (int n)
 {
   if (n == 0)
     return 1;
   else
     return n*fatorial (n-1);
 }
```

- Qual é o passo base?

  Se n=0, então o fatorial = 1

- Qual é o passo indutivo?
  - Devemos expressar a solução para n > 0, supondo que já sabemos a solução para algum caso mais simples (**n-1**, por exemplo)

  n! = n * (n-1)!

Alguns métodos recursivos (Java)

- Busca sequencial

```
int sequencial(int valor, int[] vetor, int n) {
   if(n == 1) {
      if(vetor[0] == valor) {
         return 0;
      }
      else {
         return -1;
      }
   } else {
      int index = sequencial(valor, vetor, n-1);
      if(index < 0) {
         if(vetor[n-1] == valor) {
            index = n-1;
         }
      }
      return index;
   }
}
```

Alguns métodos recursivos (Java)

- Busca Binária

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
         esq = meio + 1;
         return binaria(valor, vetor, esq, dir);
      }  else if(valor < vetor[meio]) {
         dir = meio - 1;
         return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
         return -1; // retorna -1 se o valor nao for
encontrado
   }
}
```

Paradigma Dividir e conquistar

- Em geral, algoritmos recursivos seguem a abordagem *dividir e conquistar:*
  - desmembram o problema em vários subproblemas semelhantes, mas menores em tamanho;
  - resolvem os subproblemas recursivamente;
  - combinam soluções dos subproblemas para criar solução para o problema original.

Paradigma Dividir e conquistar

- Três passos em cada nível de recursão:
  - Dividir – o problema em um determinado número de subproblemas.
  - Conquistar – os subproblemas, resolvendo-os recursivamente. Se os tamanhos dos subproblemas forem pequenos o suficiente, resolvê-los diretamente.
  - Combinar – as soluções dadas aos subproblemas para formar solução procurada para problema original.

Paradigma Dividir e conquistar

- Três passos em cada nível de recursão: dividir, conquistar e combinar

- Como se calcula a complexidade geral, dados esses 3 passos, considerando entrada de tamanho ***n***?

Paradigma Dividir e conquistar

- Três passos em cada nível de recursão: dividir, conquistar e combinar

- Como se calcula a complexidade geral, dados esses 3 passos, considerando entrada de tamanho ***n***?
  - T(n) = T(dividir(n)) + T(conquistar(n)) + T(combinar(n))

Paradigma Dividir e conquistar

- Três passos em cada nível de recursão: dividir, conquistar e combinar

- Como se calcula a complexidade geral, dados esses 3 passos, considerando entrada de tamanho ***n***?
  - $T(n) = T(dividir(n)) + T(conquistar(n)) + T(combinar(n))$
  - Para entradas pequenas ($n \leq c$, c pequeno), podemos assumir $T(n) = O(1)$

- Problemas resolvidos com o paradigma dividir em conquistar têm T(n) expressa em função da própria T(n).
  - Nesses casos, dizemos que T(n) é uma equação de recorrência.
  - T(n) da etapa de ***conquistar*** é expresso pelo tamanho do subproblema.
  - Exemplo: se temos ***a*** chamadas recursivas e em cada chamada o problema é dividido em um tamanho ***b***:
  - $T(n) = aT(n/b)$

Paradigma Dividir e conquistar

- Chamando o tempo de **dividir** e **combinar** de D(n) e C(n), respectivamente:
- T(n) = aT(n/b) + D(n) + C(n)

- Considerando que para ***n*** suficiente pequeno T(n) = O(1)

$$T(n) = \begin{cases} O(1), \;\; n \le c \\ aT(n/b) + D(n) + C(n), caso\ contrário \end{cases}$$

$$T(n) = \begin{cases} O(1), n \le c \\ aT(n/b) + f(n), caso\ contrário \end{cases}$$

Paradigma Dividir e conquistar

- Chamando o tempo de **dividir** e **combinar** de D(n) e C(n), respectivamente:
- T(n) = aT(n/b) + D(n) + C(n)

- Considerando que para ***n*** suficiente pequeno T(n) = O(1)

$$T(n) = \begin{cases} O(1), \quad n \le c \\ aT(n/b) + D(n) + C(n), caso\ contrário \end{cases}$$

$$T(n) = \begin{cases} O(1), n \le c \\ aT(n/b) + f(n), caso\ contrário \end{cases}$$

f(n)=D(n)+C(n)

Equações de recorrência

- Forma geral e uma recorrência que usa paradigma *dividir e conquistar* :

$$T(n) = aT(n/b) + f(n)$$

onde :

$a \geq 1$;

$b > 1$;

$f(n)$ é uma função assintoticamente positiva.

SISTEMAS DE INFORMAÇÃO

EACH

Equações de recorrência – Método Mestre

- Forma geral e uma recorrência que usa paradigma *dividir e conquistar* :

$$T(n) = aT(n/b) + f(n)$$

O que é **a** ?

onde :

$a \geq 1$;

$b > 1$;

$f(n)$ é uma função assintoticamente positiva.

Equações de recorrência – Método Mestre

- Forma geral e uma recorrência que usa paradigma *dividir e conquistar* :

$$T(n) = aT(n/b) + f(n)$$

O que é **a** ?
Quantidade de subproblemas

onde :

$a \geq 1$;

$b > 1$;

$f(n)$ é uma função assintoticamente positiva.

Equações de recorrência – Método Mestre

- Forma geral e uma recorrência que usa paradigma *dividir e conquistar* :

$$T(n) = aT(n/b) + f(n)$$

O que é **n/b** ?

onde :

$a \geq 1$;

$b > 1$;

$f(n)$ é uma função assintoticamente positiva.

Equações de recorrência – Método Mestre

- Forma geral e uma recorrência que usa paradigma *dividir e conquistar* :

$$T(n) = aT(n/b) + f(n)$$

O que é **n/b** ?
Tamanho dos subproblemas

onde :

$a \geq 1$;

$b > 1$;

$f(n)$ é uma função assintoticamente positiva.

Equações de recorrência – Método Mestre

- Forma geral e uma recorrência que usa paradigma *dividir e conquistar* :

$$T(n) = aT(n/b) + f(n)$$

O que é **f(n)**?

onde :

$a \geq 1$;

$b > 1$;

f(n) é uma função assintoticamente positiva.

Equações de recorrência – Método Mestre

- Forma geral e uma recorrência que usa paradigma *dividir e conquistar* :

$$T(n) = aT(n/b) + f(n)$$

O que é **f(n)**?
Função que fornece a complexidade das etapas de divisão e conquista.

onde :

$a \geq 1$;

$b > 1$;

$f(n)$ é uma função assintoticamente positiva.

Complexidade de algoritmos recursivos

- Como calcular a complexidade temporal de algoritmos recursivos?

```
int binaria(int valor, int[] vetor, int esq, int dir) {
  int meio = (esq + dir)/2;
  if(esq <= dir) {
     if(valor > vetor[meio]) {
        esq = meio + 1;
        return binaria(valor, vetor, esq, dir);
     }  else if(valor < vetor[meio]) {
        dir = meio - 1;
        return binaria(valor, vetor, esq, dir);
     } else {
       return meio;
     }
     } else {
        return -1; // retorna -1 se o valor nao for encontrado
  }
}
```

SISTEMAS DE INFORMAÇÃO

EACH

Complexidade de algoritmos recursivos

- Como calcular a complexidade temporal de algoritmos recursivos?
  - devemos considerar as chamadas para o próprio método
  - já sabemos que a abordagem *dividir e conquistar* gera T(n) como uma equação de recorrência.
  - precisamos analisar o algoritmo e definir a forma de T(n), de acordo com este conceito.

Complexidade de algoritmos recursivos

- Como calcular a complexidade temporal de algoritmos recursivos?
  - devemos considerar as chamadas para o próprio método
  - já sabemos que a abordagem *dividir e conquistar* gera T(n) como uma equação de recorrência.
  - precisamos analisar o algoritmo e definir a forma de T(n), de acordo com este conceito.

```
int binaria(int valor, int[] vetor, int esq, int dir) {
  int meio = (esq + dir)/2;
  if(esq <= dir) {
     if(valor > vetor[meio]) {
        esq = meio + 1;
        return binaria(valor, vetor, esq, dir);
     }  else if(valor < vetor[meio]) {
        dir = meio - 1;
        return binaria(valor, vetor, esq, dir);
     } else {
       return meio;
     }
     } else {
        return -1; // retorna -1 se o valor nao for encontrado
  }
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
      esq = meio + 1;
      return binaria(valor, vetor, esq, dir);
      }         else if(valor < vetor[meio]) {
      dir = meio - 1;
      return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
      return -1; // retorna -1 se o valor nao for encontrado
   }
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
      esq = meio + 1;
      return binaria(valor, vetor, esq, dir);
      }         else if(valor < vetor[meio]) {
      dir = meio - 1;
      return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
      return -1; // retorna -1 se o valor nao for encontrado
   }
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
      esq = meio + 1;
      return binaria(valor, vetor, esq, dir);
      }         else if(valor < vetor[meio]) {
      dir = meio - 1;
      return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
      return -1; // retorna -1 se o valor nao for encontrado
   }
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com v[meio]

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
      esq = meio + 1;
      return binaria(valor, vetor, esq, dir);
      }         else if(valor < vetor[meio]) {
      dir = meio - 1;
      return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
      return -1; // retorna -1 se o valor nao for encontrado
   }
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com v[meio]
  - Quando n = 1, quanto vale T(n)?

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
      esq = meio + 1;
      return binaria(valor, vetor, esq, dir);
      }        else if(valor < vetor[meio]) {
      dir = meio - 1;
      return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
      return -1; // retorna -1 se o valor nao for encontrado
   }
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com v[meio]
  - Quando n = 1, quanto vale T(n)?
    - T(n) = O(2)   >>>> Por quê?

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
      esq = meio + 1;
      return binaria(valor, vetor, esq, dir);
      }         else if(valor < vetor[meio]) {
      dir = meio - 1;
      return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
      return -1; // retorna -1 se o valor nao for encontrado
   }
}
```

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com v[meio]
  - Quando n = 1, quanto vale T(n)?
    - T(n) = O(2) >>>> Por quê?
  - Quando n > 1, quanto vale T(n)?

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
      esq = meio + 1;
      return binaria(valor, vetor, esq, dir);
      }         else if(valor < vetor[meio]) {
      dir = meio - 1;
      return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
      return -1; // retorna -1 se o valor nao for encontrado
   }
}
```

EACH

Equações de recorrência

- Custos da busca binária:
  - Qual o pior caso?
    - Quando o valor não está no vetor
  - Qual é a operação que mais ocorre?
    - inspeções do elemento – comparações de valor com v[meio]
  - Quando n = 1, quanto vale T(n)?
    - T(n) = O(2)   >>>> Por quê?
  - Quando n > 1, quanto vale T(n)?
    - T(n) = T(n/2) + O(2)   >>>> Por quê?

```
int binaria(int valor, int[] vetor, int esq, int dir) {
   int meio = (esq + dir)/2;
   if(esq <= dir) {
      if(valor > vetor[meio]) {
      esq = meio + 1;
      return binaria(valor, vetor, esq, dir);
      }        else if(valor < vetor[meio]) {
      dir = meio - 1;
      return binaria(valor, vetor, esq, dir);
      } else {
        return meio;
      }
      } else {
      return -1; // retorna -1 se o valor nao for encontrado
   }
}
```

EACH

Equações de recorrência

- Como encontrar a complexidade desta equação?

$$T(n) = \begin{cases} O(2), se\ n = 1 \\ T(n/2) + O(2), caso\ contrário \end{cases}$$

Equações de recorrência

- Como encontrar a complexidade desta equação?

$$T(n) = \begin{cases} O(2), se \ \ n = 1 \\ T(n/2) + O(2), caso \ \ contrário \end{cases}$$

- Vamos tentar expandi-la: $T(n) = T(n/2) + O(2)$

$$T(n) = T(n/4) + O(2) + O(2)$$

$$T(n) = T(n/8) + O(2) + O(2) + O(2)$$

$$\ldots$$

$$T(n) = T(n/2^i) + i * O(2)$$

onde *i* é tal que $n/2^i=1$. Isto é, $i = \log_2 n$

Equações de recorrência

$$T(n) = T(n/2) + O(2)$$
$$T(n) = T(n/4) + O(2) + O(2)$$
$$T(n) = T(n/8) + O(2) + O(2) + O(2)$$
...
$$T(n) = T(n/2^i) + i * O(2)$$

onde *i* é tal que $n/2^i=1$. Isto é, $i = \log_2 n$

Substituindo $T(n/2^i)$ por T(1), temos: T(n) = i*O(2) +T(1)

Mas T(1) = O(2)

T(n) = (i+1) * O(2)

Usando operação f(n) * O(g(n)) = O(f(n)g(n)) da notação O, temos:

$$T(n) = O(2 * i) + O(2)$$

que é igual a:

$$T(n) = O(i)$$

Lembrando que $i = \log_2 n$:

$$T(n) \in O(\log_2 n)$$

Equações de recorrência

Busca sequencial recursiva

- Pior caso?
- Operação mais realizada?
- T(n) para n=1?
- T(n) para n > 1?

```
int sequencial(int valor, int[] vetor, int n) {
   if(n == 1) {
      if(vetor[0] == valor) {
         return 0;
      }
      else {
         return -1;
      }
   } else {
      int index = sequencial(valor, vetor, n-1);
      if(index < 0) {
         if(vetor[n-1] == valor) {
            index = n-1;
         }
      }
      return index;
   }
}
```

SISTEMAS DE INFORMAÇÃO

EACH

Equações de recorrência

Busca sequencial recursiva

- Pior caso? valor não está no *array*
- Operação mais realizada? inspeção
- T(n) para n=1? O(1)
- T(n) para n > 1? T(n-1) + O(1)

```
int sequencial(int valor, int[] vetor, int n) {
   if(n == 1) {
      if(vetor[0] == valor) {
         return 0;
      }
      else {
         return -1;
      }
   } else {
      int index = sequencial(valor, vetor, n-1);
      if(index < 0) {
         if(vetor[n-1] == valor) {
            index = n-1;
         }
      }
      return index;
   }
}
```

Equações de recorrência

Busca sequencial recursiva

$$T(n) = \begin{cases} O(1), se \ n = 1 \\ T(n-1) + O(1), caso \ contrário \end{cases}$$

$$\boxed{\begin{array}{l} T(n) = T(n-1) + O(1) \\ T(n) = T(n-2) + O(1) + O(1) \\ \ldots \\ T(n) = T(n-i) + i * O(1) \end{array}}$$

onde i é tal que n – i = 1

T(n) = T(1) +(n-1) * O(1)

Mas T(1) = O(1)

T(n) = O(1) +(n-1)*O(1)

T(n) = n*O(1)

Portanto:

$$T(n) \in O(n)$$

```
int sequencial(int valor, int[] vetor, int n) {
   if(n == 1) {
      if(vetor[0] == valor) {
         return 0;
      }
      else {
         return -1;
      }
   } else {
      int index = sequencial(valor, vetor, n-1);
      if(index < 0) {
         if(vetor[n-1] == valor) {
            index = n-1;
         }
      }
      return index;
   }
}
```

Equações de recorrência

Torre de Hanói recursiva

$$T(n) = \begin{cases} O(1), se\ n = 1 \\ 2T(n-1) + O(1), caso\ contrário \end{cases}$$

$$T(n) = 2T(n-1) + O(1)$$
$$T(n) = 4T(n-2) + 3 * O(1)$$
$$\ldots$$
$$T(n) = 2^i T(n-i) + 2^i * O(1)$$

onde i é tal que n – i = 1

$T(n) = 2^{n-1}*T(1) + (2^{n-1} - 1)*O(1)$

Mas $T(1) = O(1)$

$T(n) = (2^{n-1+1} - 1) *O(1)$

$T(n) = (2^n - 1) *O(1)$

Portanto:

$$T(n) \in O(2^n)$$

```
void hanoi(char ori, char dst, char aux, int nro) {
   if(nro == 1) {
      System.out.print("Move de " + ori);
      System.out.println(" para " + dst);
   }
   else {
      hanoi(ori, aux, dst, nro-1);
      hanoi(ori, dst, aux, 1);
      hanoi(aux, dst, ori, nro-1);
   }
}
```

Equações de recorrência

Fatorial recursivo

- Operação mais realizada?
- T(m) para m=1?
- T(m) para m > 1?

```
int fatorial (int m)
 {
   if (m == 0)
     return 1;
   else
     return m*fatorial (m-1);
 }
```

Equações de recorrência

Fatorial recursivo

- Operação mais realizada? multiplicação
- T(m) para m=1? assumindo que ocorre uma multiplicação: return 1+1 $\Rightarrow$T(m)=O(1)
- T(m) para m > 1? T(m)=O(1) + T(m-1)

$$T(m)=\begin{cases} O(1), se \;\; m=1 \\ T(m-1)+O(1), caso \;\; contrário \end{cases}$$

```
int fatorial (int m)
 {
   if (m == 0)
     return 1;
   else
     return m*fatorial (m-1);
 }
```

Equações de recorrência

Fatorial recursivo

- Problema: o que é m, neste caso?

```
int fatorial (int m)
 {
   if (m == 0)
     return 1;
   else
     return m*fatorial (m-1);
 }
```

Equações de recorrência

Fatorial recursivo

```
int fatorial (int m)
 {
  if (m == 0)
    return 1;
  else
    return m*fatorial (m-1);
 }
```

- Problema: o que é m, neste caso?
  - Não é um tamanho de entrada, mas a própria entrada...
  - Ou seja: a complexidade assintótica é proporcional à própria entrada...
  - Como podemos escrever a complexidade em termo do tamanha da entrada?
  - Ou seja: como definir o tamanho ***da entrada***?
- Se considerarmos n como um número implementado com ***n*** bits?
  - Tamanho da entrada = ***n***
  - Como representar a entrada em termo do tamanho de bits da entrada?
    - $T(n) \in O(f(n))$

SISTEMAS DE INFORMAÇÃO

EACH

Equações de recorrência

Fatorial recursivo

```
int fatorial (int m)
 {
   if (m == 0)
     return 1;
   else
     return m*fatorial (m-1);
 }
```

- Podemos fazer:

$$n = 2^{n-1}d_{n-1} + 2^{n-2}d_{n-2} + \ldots + 2^0 d_0$$

$$onde\ \ d_i \in \{0,1\}\ \ e\ \ 0 \leq i \leq n$$

- Pior caso: d_i=1, então m=$2^{n-1}+2^{n-2}+\ldots+2^0$
- Mas $2^n > 2^{n-1}+2^{n-2}+\ldots+2^0$ (provado em aula anterior por indução)
- Então, podemos escrever que n é $O(2^n)$

$$T(n) \in O(2^n)$$

Equações de recorrência

- Já vimos como escrever T(n) na forma de uma equação de recorrência, a partir da análise do algoritmo.
- Também já sabemos resolver equações de recorrência para algumas casos, considerando sua expansão, isto é, obter limites assintóticos para T(n)

- O problema é:
  - É trivial expandir equações de recorrência?

SISTEMAS DE INFORMAÇÃO

EACH

Equações de recorrência

- Já vimos como escrever T(n) na forma de uma equação de recorrência, a partir da análise do algoritmo.
- Também já sabemos resolver equações de recorrência para algumas casos, considerando sua expansão, isto, obter limites assintóticos para T(n).

- O problema é:
  - É trivial expandir equações de recorrência? NÃO!!!
  - Sempre é possível resolver equações de recorrência com expansão?

Equações de recorrência

- Já vimos como escrever T(n) na forma de uma equação de recorrência, a partir da análise do algoritmo.
- Também já sabemos resolver equações de recorrência para algumas casos, considerando sua expansão, isto, obter limites assintóticos para T(n).

- O problema é:
  - É trivial expandir equações de recorrência? NÃO!!!
  - Sempre é possível resolver equações de recorrência com expansão? NÃO!!!
  - Como fazer para encontrar a complexidade assintótica de equações de recorrência?

Equações de recorrência

- Há basicamente 3 métodos para resolver equações de recorrência
  - Método de substituição
  - Árvore de recursão
  - Método mestre

Equações de recorrência

- Há basicamente 3 métodos para resolver equações de recorrência
  - Método de substituição
  - Árvore de recursão – não veremos formalmente este (ver expansão anterior. Sugestão: livro Cormen)
  - Método mestre

Equações de recorrência - Método de substituição

- Passos a serem seguidos:
  1. Supor um limite hipotético
  2. Usar indução matemática para provar que suposição está correta

- Exemplo:

$$T(n) = \begin{cases} O(2), se\ n = 1 \\ 2T(n/2) + O(2), caso\ contrário \end{cases}$$

Equações de recorrência - Método de substituição

- Passos a serem seguidos:
  1. Supor um limite hipotético
  2. Usar indução matemática para provar que suposição está correta

- Exemplo:

$$T(n) = \begin{cases} O(2), se\ n = 1 \\ 2T(n/2) + O(2), caso\ contrário \end{cases}$$

- Supor que $T(n) \in O(n \log n)$
- O passo 2 consiste em provar que $T(n) \leq c(n \log n)$ para $c > 0$.

Equações de recorrência - Método de substituição

- Exemplo:

$$T(n) = \begin{cases} O(2), se \quad n = 1 \\ 2T(n/2) + O(2), caso \quad contrário \end{cases}$$

  - Supor que T(n) ∈ O(n logn)
  - O passo 2 consiste em provar que T(n) ≤ c(n logn) para c > 0.
  - Passo indutivo: assumindo que T(n/2) ≤ c(n/2 log n/2)

$$T(n) \leq 2(c(n/2 \log n/2)) + n$$

$$T(n) \leq cn(\log n/2) + n$$

  - Manipulando somente o lado direito da inequação:

$$= cn \log n - cn \log 2 + n$$

$$= cn \log n - cn + n$$

$$= cn \log n - n(c-1)$$

  - Se escolhermos c≥1, a parcela (n(c-1)) vai diminuir a parcela (cn log n), então podemos afirmar que:

$T(n) \leq cn \log n$ E fica provado o passo indutivo!!!

SISTEMAS DE INFORMAÇÃO

EACH

Equações de recorrência - Método de substituição

- Exemplo:

$$T(n) = \begin{cases} O(2), se \ n = 1 \\ 2T(n/2) + O(2), caso \ contrário \end{cases}$$

- Falta provar o passo base: temos que provar que T(1) $\leq$ c (1 log 1)

- Mas, conforme definido acima, T(1) =1
- Portanto, não conseguimos provar o passo base T(1)

- Isso significa que precisamos encontrar um outro passo base.

Equações de recorrência - Método de substituição

- Exemplo:

$$T(n) = \begin{cases} O(2), se \ n = 1 \\ 2T(n/2) + O(2), caso \ contrário \end{cases}$$

- A notação assintótica exige que provemos T(n) ≤ c(n log n) para n maior ou igual que uma constante n_0 de nossa escolha.
- Se escolhermos novo passo base, com n_0 = 2, então: n ≥ 2 e o novo passo passe será: $T(2) \le c(2 \log 2), \ isto \ é, \ T(2) \le 2c$

- A constante ***c*** escolhida tem que valer para o passo indutivo e para o passo base.

Prova:

- Pela equação de recorrência 1, T(1) = 1. Logo, T(2) = T(1) + 2 = 3.
- Escolhendo c ≥ 2 temos que T(2) = 3 ≤ 2c, ou seja, o passo base foi provado (note que c ≥ 2 vale para o passo base e o passo indutivo).
- Portanto:

$$T(n) \in O(n \log n)$$

Equações de recorrência – Método Mestre

- Fornece um processo de "livro de receitas" para resolver recorrências da forma:

  $T(n) = aT(n/b) + f(n)$

  onde :

  $a \geq 1$;

  $b > 1$;

  $f(n)$ é uma função assintoticamente positiva.

- Método mestre exige memorização de três casos, mas a partir deles permite descobrir facilmente soluções de muitas recorrências.

Equações de recorrência – Método Mestre

- **Teorema Mestre**

Sejam a ≥ 1 e b > 1 constantes, seja f(n) uma função e seja T(n) definida sobre os inteiros não negativos pela recorrência

T(n) = aT(n/b) + f(n)

onde interpretamos n/b com o significado de $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$

Então, T(n) pode ser limitado assintoticamente como a seguir:

1. Se $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta\left(n^{\log_b a}\right)$

SISTEMAS DE INFORMAÇÃO

EACH

Equações de recorrência – Método Mestre

- **Teorema Mestre**

Sejam a ≥ 1 e b > 1 constantes, seja f(n) uma função e seja T(n) definida sobre os inteiros não negativos pela recorrência

T(n) = aT(n/b) +f(n)

onde interpretamos n/b com o significado de $\lfloor n/b \rfloor \; ou \; \lceil n/b \rceil$

Então, T(n) pode ser limitado assintoticamente como a seguir:

1. Se $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta\left(n^{\log_b a}\right)$

2. Se $f(n) = \Theta\left(n^{\log_b a}\right)$, então $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

Equações de recorrência – Método Mestre

- **Teorema Mestre**

Sejam a ≥ 1 e b > 1 constantes, seja f(n) uma função e seja T(n) definida sobre os inteiros não negativos pela recorrência

T(n) = aT(n/b) +f(n)

onde interpretamos n/b com o significado de $\lfloor n/b \rfloor$ *ou* $\lceil n/b \rceil$

Então, T(n) pode ser limitado assintoticamente como a seguir:

1. Se $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta\left(n^{\log_b a}\right)$

2. Se $f(n) = \Theta\left(n^{\log_b a}\right)$, então $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

3. Se $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ para alguma constante $\epsilon > \mathbf{0}$,
   e se ***af(n/b) ≤ cf(n)*** para alguma constante ***c < 1*** e para todo ***n*** suficientemente grande, então $T(n) = \Theta\left(f(n)\right)$

Equações de recorrência – Método Mestre

O que significa o Teorema Mestre?

Então, T(n) pode ser limitado assintoticamente como a seguir:

1. Se $f(n) = O\left(n^{\log_b a-\epsilon}\right)$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta\left(n^{\log_b a}\right)$

2. Se $f(n) = \Theta\left(n^{\log_b a}\right)$, então $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

3. Se $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ para alguma constante $\epsilon > \mathbf{0}$,
   e se ***af(n/b) ≤ cf(n)*** para alguma constante ***c < 1*** e para todo ***n*** suficientemente grande, então $T(n) = \Theta\left(f(n)\right)$

- Estamos comparando a função ***f(n)*** com a fun $n^{\log_b a}$
- Intuitivamente, a solução para a recorrência é determinada pela maior das duas funções.

Caso 1: se a função $n^{\log_b a}$ for maior, a solução será $T(n) = \Theta\left(n^{\log_b a}\right)$

Caso 3: se a função ***f(n)*** for maior, a solução será $T(n) = \Theta\left(f(n)\right)$

Caso 2: se 2 funções tiverem o mesmo tamanho, faz-se a multiplicação por um fator logarítmico e a solução será $T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta\left(f(n) \lg n\right)$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

$T(n) = aT(n/b) + f(n)$
onde :
$a \geq 1$; $b > 1$;
$f(n)$ é uma função assintoticamente positiva.

Exemplo 1:

$T(n) = 9T(n/3) + n$

a= ?

b= ?

f(n) = ?

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 1:

T(n) = 9T(n/3) + n

a= 9

b= 3

f(n) = n

$$f(n) = O\left(n^{\log_b a - \in}\right), \textbf{algum} \in > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$$

$$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$$

$$f(n) = \Omega\left(n^{\log_b a + \in}\right), \textbf{algum} \in > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 1:

T(n) = 9T(n/3) + n

a= 9, b= 3, f(n) = n

Portanto, temos : $n^{\log_b a} = n^{\log_3 9} = n^2$

Como decidir se a expressão aparecer nos 3 casos???

$f(n) = O\left(n^{\log_b a - \epsilon}\right), \textbf{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \textbf{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

  Exemplo 1:

  T(n) = 9T(n/3) + n

  a= 9, b= 3, f(n) = n

  Portanto, temos : $n^{\log_b a} = n^{\log_3 9} = n^2$

  Como decidir se a expressão aparecer nos 3 casos???

  - Testar os 3 casos!

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

$$f(n) = O\left(n^{\log_b a - \epsilon}\right), \textbf{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$$

$$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$$

$$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \textbf{algum } \epsilon > 0 \Rightarrow T(n) = \Theta(f(n))$$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

  Exemplo 1:

  T(n) = 9T(n/3) + n

  a= 9, b= 3, f(n) = n Portanto, temos : $n^{\log_b a} = n^{\log_3 9} = n^2$

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

- Caso 1, considerando ε =1

$$n \in O\left(n^{\log_3 9-\varepsilon} = n^{2-1} = n\right)?$$

$f(n) = O\left(n^{\log_b a-\epsilon}\right)$, ***algum*** $\epsilon > 0$ ⇒ $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ ⇒ $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$, ***algum*** $\epsilon > 0$ ⇒ $T(n) = \Theta\left(f(n)\right)$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

Exemplo 1:

T(n) = 9T(n/3) + n

a= 9, b= 3, f(n) = n   Portanto, temos : $n^{\log_b a} = n^{\log_3 9} = n^2$

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

- Caso 1, considerando ε =1

$$n \in O\left(n^{\log_3 9-\varepsilon} = n^{2-1} = n\right)?$$

SIM!!!

Então: $T(n) = \Theta\left(n^2\right)$

$f(n) = O\left(n^{\log_b a-\epsilon}\right)$, **algum** $\epsilon > 0$ ⇨ $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ ⇨ $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$, **algum** $\epsilon > 0$ ⇨ $T(n) = \Theta\left(f(n)\right)$

Já encontramos! Não precisamos testar os outros casos!

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a= ?

b= ?

f(n) = ?

$$f(n) = O\left(n^{\log_b a - \epsilon}\right), \textbf{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$$

$$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$$

$$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \textbf{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a= 1, b= 3/2, f(n) = 1 $\quad n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

$f(n) = O\left(n^{\log_b a - \epsilon}\right), \textbf{\textit{algum}} \ \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \textbf{\textit{algum}} \ \epsilon > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

Caso 1 , considerando ε =1

$$1 \in O\left(n^{0-1} = n^{-1} = \frac{1}{n}\right)?$$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a= 1, b= 3/2, f(n) = 1 $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Caso 1 , considerando ε =1: $1 \in O\left(n^{0-1} = n^{-1} = \frac{1}{n}\right)$?

NÃO!!!

$f(n) = O\left(n^{\log_b a - \epsilon}\right), \textbf{\textit{algum}} \in > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \textbf{\textit{algum}} \in > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

SISTEMA
INFORMA

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a= 1, b= 3/2, f(n) = 1 $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Caso 1 , considerando ε =1: $1 \in O\left(n^{0-1} = n^{-1} = \frac{1}{n}\right)?$

NÃO!!!

Caso 2: $1 \in \Theta(1)?$

$f(n) = O\left(n^{\log_b a - \epsilon}\right), \textbf{algum } \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \textbf{algum } \epsilon > 0 \Rightarrow T(n) = \Theta(f(n))$

SISTEMA
INFORMA

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 2:

T(n) = T(2n/3) + 1

a= 1, b= 3/2, f(n) = 1 $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Caso 1 , considerando ε =1: $1 \in O\left(n^{0-1} = n^{-1} = \frac{1}{n}\right)?$

NÃO!!!

Caso 2: $1 \in \Theta(1)?$

SIM!!!

Então: $T(n) = \Theta(n^0 \lg n) = \Theta(\lg n)$

$f(n) = O(n^{\log_b a - \epsilon}), \boldsymbol{algum}\ \epsilon > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$

$f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$

$f(n) = \Omega(n^{\log_b a + \epsilon}), \boldsymbol{algum}\ \epsilon > 0 \Rightarrow T(n) = \Theta(f(n))$

SISTEMA
INFORMA

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3

b= 4

f(n) = n lg n

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

$$f(n) = O\left(n^{\log_b a - \epsilon}\right), \boldsymbol{algum}\ \epsilon > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$$

$$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$$

$$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \boldsymbol{algum}\ \epsilon > 0 \Rightarrow T(n) = \Theta(f(n))$$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$

Caso 1, considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$?

NÃO!!! (porque leva a nlgn=O(n^c), c<1)

$f(n) = O\left(n^{\log_b a - \epsilon}\right), \textbf{\textit{algum}} \in > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \textbf{\textit{algum}} \in > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

SISTE INFOR

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$

Caso 1, considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$?

NÃO!!! (porque leva a nlgn=O(n^c), c<1)

Caso 2: $n \lg n \in \Theta\left(n^{0,793}\right)$?

$f(n) = O\left(n^{\log_b a - \epsilon}\right), \boldsymbol{algum} \in > 0 \Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \boldsymbol{algum} \in > 0 \Rightarrow T(n) = \Theta\left(f(n)\right)$

SISTE
INFOR

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$

Caso 1, considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$? NÃO!!! (porque leva a nlgn=O(n^c), c<1)

Caso 2: $n \lg n \in \Theta\left(n^{0,793}\right)$? NÃO!!! (porque leva a nlgn=O(n^c), c<1)

$f(n) = O\left(n^{\log_b a-\epsilon}\right)$, $\boldsymbol{algum}\ \epsilon > 0$ ⇨ $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ ⇨ $T(n) = \Theta\left(n^{\log_b a} \lg$

$f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$, $\boldsymbol{algum}\ \epsilon > 0$ ⇨ $T(n) = \Theta\left(f(n)\right)$

SISTEMAS D
INFORMAÇÃ

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$

Caso 1, considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$? NÃO!!! (porque leva a nlgn=O(n^c), c<1)

Caso 2: $n \lg n \in \Theta\left(n^{0,793}\right)$? NÃO!!! (porque leva a nlgn=O(n^c), c<1)

Caso 3, considerando ε ≈0,2:

$n \lg n \in \Omega\left(n^{0,793+0,2}\right)$? ou seja: $n \lg n \in \Omega(n)$?

$f(n) = O\left(n^{\log_b a - \epsilon}\right)$, **algum** $\epsilon > 0$ ⇨ $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ ⇨ $T(n) = \Theta\left(n^{\log_b a} \lg\right.$

$f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$, **algum** $\epsilon > 0$ ⇨ $T(n) = \Theta(f(n))$

SISTEMAS D
INFORMAÇ

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$

Caso 1 , considerando ε =1: $n \lg n \in O\left(n^{0,793-1}\right)$? NÃO!!! (porque leva a nlgn=O(n^c), c<1)

Caso 2: $n \lg n \in \Theta\left(n^{0,793}\right)$? NÃO!!! (porque leva a nlgn=O(n^c), c<1)

Caso 3, , considerando ε ≈0,2:

$n \lg n \in \Omega\left(n^{0,793+0,2}\right)$? ou seja: $n \lg n \in \Omega(n)$? SIM, mas ainda temos que provar a expressão de regularidade.

Se $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ para alguma constante $\epsilon > \boldsymbol{0}$,
e se ***af(n/b) ≤ cf(n)*** para alguma constante ***c < 1*** e para todo ***n*** suficientemente grande, então $T(n) = \Theta\left(f(n)\right)$

Equações de recorrência – Método Mestre

Como usar o Teorema Mestre?

- Para usar o método mestre:
  1. Determinar qual caso (se houver algum) se aplica
  2. Anotar a resposta

T(n) = aT(n/b) + f(n)
onde :
a ≥ 1; b > 1;
f(n) é uma função assintoticamente positiva.

Exemplo 3:

T(n) = 3T(n/4) + n lg n

a= 3, b= 4, f(n) = n lg n

Portanto, temos $n^{\log_b a} = n^{\log_4 3} = O\left(n^{0,793}\right)$

Caso 3, considerando ε ≈0,2:

$n \lg n \in \Omega\left(n^{0,793+0,2}\right)$? ou seja: $n \lg n \in \Omega(n)$? SIM, mas ainda temos que provar a expressão de regularidade af(n/b) ≤ cf(n), para alguma constante c < 1.

$$af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n \lg n = cf(n) \quad para \;\; c = 3/4$$

Então: $T(n) = \Theta(n \lg n)$

Se $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ para alguma constante $\epsilon > \boldsymbol{0}$, e se ***af(n/b) ≤ cf(n)*** para alguma constante ***c < 1*** e para todo ***n*** suficientemente grande, então $T(n) = \Theta\left(f(n)\right)$

Teorema Mestre

- Exemplos de aplicação:
  - (Caso 1) $T(n) = 4T(n/2) + n \log(n)$, $T(1) = 1$.
  - (Caso 2) $T(n) = 2T(n/2) + n$, $T(1) = 1$.
  - (Caso 3) $T(n) = T(n/2) + n \log(n)$, $T(1) = 1$.

- Exemplos nos quais não se aplica:
  - $T(n) = T(n-1) + n \log(n)$, $T(1) = 1$.
  - $T(n) = T(n-a) + T(a) + n$, $T(b) = 1$.
    (para inteiros $a \geq 1$, $b \leq a$, a e b inteiros)
  - $T(n) = T(\alpha n) + T((1-\alpha)n) + n$, $T(1) = 1$.
    (para $0 < \alpha < 1$)
  - $T(n) = T(n-1) + \log(n)$, $T(1) = 1$.
  - $T(n) = 2\,T(n/2) + n \log(n)$, $T(1) = 1$.

Exercícios

- Use o Teorema Mestre para encontrar solução para as seguintes recorrências:

1. $T(n) = 4T(n/2) + n \log(n)$
2. $T(n) = 2T(n/2) + n$
3. $T(n) = T(n/2) + n \log(n)$
4. $T(n) = T(n/2) + \Theta(1)$
   [Recorrência da exponenciação - divisão e conquista]
5. $T(n) = 4T(n/2) + n$
6. $T(n) = 4T(n/2) + n^2$
7. $T(n) = 4T(n/2) + n^3$

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - Tradução da 2a. Edição Americana. Editora Campus, 2002 (texto base)
- Nívio Ziviani. Projeto de Algoritmos com implementações em C e Pascal. Editora Thomson, 2a. Edição, 2004.
- Notas de aula – Prof. Delano Beder – EACH-USP
- Notas de aula – Prof. Norton Roman – EACH-USP

Paradigmas de Projeto de Algoritmos

Recursividade

Divisão e Conquista

Equações de Recorrência

Professora:

Fátima L. S. Nunes

SISTEMAS DE INFORMAÇÃO

EACH