

Análise de Algoritmos Recursivos

- Para cada procedimento recursivo é associada uma função de complexidade $f(n)$ desconhecida, onde **n mede o tamanho dos argumentos para o procedimento.**
- Por se tratar de um algoritmo recursivo, $f(n)$ vai ser obtida através de uma equação de recorrência.
- **Equação de recorrência:** maneira de definir uma função por uma expressão envolvendo a mesma função.

Análise de Algoritmos Recursivos

- Tempo de execução de um algoritmo recursivo: tamanho, número de subproblemas e o tempo para decomposição.
- Capturado pela *equação de recorrência*.

$$\begin{cases} T(n) = aT(n/b) + f(n), \text{ para } n > c \\ T(n) = k, \text{ para } n \leq c \end{cases}$$

Equação de Recorrência

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

Equação de Recorrência

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

- Equação de recorrência para o Fatorial

$$\begin{cases} T(n) = c + T(n-1), & \text{para } n > 0 \\ T(n) = d. & \text{para } n \leq 0 \end{cases}$$

Equação de Recorrência

```
void max(int *v, int e, int d){  
    int u, v;  
    int m = (e+d)/2;  
  
    if (e == d) return v[e];  
    u = max(v, e, m);  
    v = max(v, m+1, d);  
    if (u > v) return u;  
    else return v;  
}
```

Equação de Recorrência

```
void max(int *v, int e, int d){  
    int u, v;  
    int m = (e+d)/2;  
  
    if (e == d) return v[e];  
    u = max(v, e, m);  
    v = max(v, m+1, d);  
    if (u > v) return u;  
    else return v;  
}
```

$$\begin{cases} T(n) = 2T(n/2) + 1, & \text{para } n > 1 \\ T(n) = 0. & \text{para } n \leq 1 \end{cases}$$

Exercício 1

- Determine a equação de recorrência para a função abaixo (operação relevante: atribuição para vetor X).

```
void ex(vetor X, int n) {  
    int i;  
  
    if (n > 0) {  
        for(i=0; i<n-1; i++)  
            X[i] = 0;  
        X[n] = n;  
        ex(X, n-1);  
    }  
}
```

Exercício 1

- Determine a equação de recorrência para a função abaixo (operação relevante: atribuição para vetor X).

```
void ex(vetor X, int n) {  
    int i;  
  
    if (n > 0) {  
        for(i=0; i<n-1; i++)  
            X[i] = 0;  
        X[n] = n;  
        ex(X, n-1);  
    }  
}
```

$$\begin{cases} T(n) = n + T(n-1), & \text{para } n > 0 \\ T(n) = 0, & \text{para } n \leq 0 \end{cases}$$

Outro Exemplo

- Considere a seguinte função:

Pesquisa(n)

```
{  
  (1) if (n <= 1)  
  (2)   'inspecione elemento' e termine;  
    else  
    {  
  (3)   para cada um dos n elementos 'inspecione elemento';  
  (4)   Pesquisa(n/3) ;  
    }  
}
```

Outro Exemplo

- Considere a seguinte função:

Pesquisa(n)

```
{  
  (1) if (n <= 1)  
  (2)   'inspecione elemento' e termine;  
    else  
    {  
  (3)   para cada um dos n elementos 'inspecione elemento';  
  (4)   Pesquisa(n/3) ;  
    }  
}
```

$$\begin{cases} T(n) = n + T(n/3), & \text{para } n > 1 \\ T(n) = 1, & \text{para } n \leq 1 \end{cases}$$

Análise de Algoritmos Recursivos

- Abordagem para solução da recorrência:
 - ❑ Expande a árvore de recursão
 - ❑ Calcula o custo em cada nível da árvore
 - ❑ Soma os custos de todos os níveis
 - ❑ Calcula a fórmula fechada do somatório

Equação de Recorrência

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

- Equação de recorrência para o Fatorial

$$\begin{cases} T(n) = c + T(n-1), & \text{para } n > 0 \\ T(n) = d. & \text{para } n \leq 0 \end{cases}$$

Equação de Recorrência

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

- Se $n \leq 0$ faz uma operação de custo constante
- Se $n > 0$ faz uma operação de custo constante e chama recursivamente a função

Equação de Recorrência

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

- Equação de recorrência para o Fatorial

$$\begin{cases} T(n) = c + T(n-1), & \text{para } n > 0 \\ T(n) = d. & \text{para } n \leq 0 \end{cases}$$

Resolvendo a Equação de Recorrência

- Existem várias formas de se resolver uma equação de recorrência
- A mais simples é expandir a equação e depois fazer uma substituição dos termos
- Ex. Fatorial:

$$T(n) = c + T(n-1)$$

$$T(n-1) = c + T(n-2)$$


$$T(n-2) = c + T(n-3)$$

...

$$T(1) = c + T(0)$$

$$T(0) = d$$

$$T(n) = c + c + c + \dots + c + d$$


n vezes

$$T(n) = n.c + d \rightarrow O(n)$$

Equação de Recorrência

```
void max(int *v, int e, int d){  
    int u, v;  
    int m = (e+d)/2;  
  
    if (e == d) return v[e];  
    u = max(v, e, m);  
    v = max(v, m+1, d);  
    if (u > v) return u;  
    else return v;  
}
```

$$\begin{cases} T(n) = 2T(n/2) + 1, & \text{para } n > 1 \\ T(n) = 0. & \text{para } n \leq 1 \end{cases}$$

Exercício 1

- Determine a equação de recorrência para a função abaixo (operação relevante: atribuição para vetor X). Qual a sua complexidade?

```
void ex(vetor X, int n) {  
    int i;  
  
    if (n > 0) {  
        for(i=0; i<n-1; i++)  
            X[i] = 0;  
        X[n] = n;  
        ex(X, n-1);  
    }  
}
```

$$\begin{cases} T(n) = n + T(n-1), & \text{para } n > 0 \\ T(n) = 0, & \text{para } n \leq 0 \end{cases}$$

Outro Exemplo

- Equação :

$$T(n) = 2T(n/2) + n;$$

$$T(1) = 1;$$

$$\begin{array}{l} T(n) = 2T(n/2) + n; \\ T(1) = 1; \end{array}$$

- Essa equação lembra que tipo de problema?
- Como resolver essa equação?

Resolvendo a Equação

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

$$T(1) = 1$$

Expandindo a equação :

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

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

$$4T(n/4) = 8T(n/8) + n$$

?

$$2^{i-1}T(n/2^{i-1}) = 2^i T(n/2^i) + n$$

Substituindo os termos :

$$T(n) = 2^i T(n/2^i) + i.n$$

Para colocar a Condição de Parada :

$$T(n/2^i) \rightarrow T(1)$$

$$n/2^i = 1 \rightarrow i = \log_2 n$$

Logo :

$$T(n) = 2^i T(n/2^i) + i.n = 2^{\log_2 n}.1 + (\log_2 n).n = n + n.\log_2 n = O(n.\log_2 n)$$

Exercício

- Considere a seguinte função:

Pesquisa(n)

```
{  
  (1)  if (n <= 1)  
  (2)    'inspecione elemento ' e termine;  
      else  
      {  
  (3)    para cada um dos n elementos 'inspecione elemento';  
  (4)    Pesquisa(n/3) ;  
      }  
}
```

Análise da Função Recursiva

- Equação de recorrência

$$\begin{cases} T(n) = n + T(n/3), & \text{para } n > 1 \\ T(n) = 1, & \text{para } n \leq 1 \end{cases}$$

- Resolva a equação de recorrência

- Dicas:

- Pode fazer a simplificação de n será sempre divisível por 3
 - Somatório de uma PG finita:

$$\sum_{i=0}^n r^i = \frac{1 - r^{n+1}}{1 - r}$$

Resolvendo a equação

- Substitui-se os termos $T(k)$, $k < n$, até que todos os termos $T(k)$, $k > 1$, tenham sido substituídos por fórmulas contendo apenas $T(1)$.

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

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

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

$$\vdots$$
$$\vdots$$

$$1 \rightarrow n/3^K = 1 \rightarrow n = 3^K$$

$$T(n/3/3 \cdots /3) = n/3/3 \cdots /3 + \underbrace{T(n/3 \cdots /3)}$$

Resolvendo a equação

Considerando que $T(n/3^K) = T(1)$ temos:

$$T(n) = \sum_{i=0}^{k-1} \frac{n}{3^i} + T(1) = n \sum_{i=0}^{k-1} \frac{1}{3^i} + 1$$

Aplicando a fórmula do somatório de uma PG finita temos:

$$1 + n(1 - (1/3)^k) / (1 - 1/3)$$

$$1 + n(1 - (1/n)) / (2/3)$$

$$1 + (n - 1) / (2/3)$$

$$3n/2 - 1/2$$

$$\left(\sum_{i=0}^n r^i = \frac{1 - r^{n+1}}{1 - r} \right)$$

O(n)