PROJETO E ANÁLISE DE ALGORITMOS

PARTE II

ALGORITMOS ITERATIVOS

1. Análise de Algoritmos Não-recursivos

- Determinar o tempo de execução de um programa pode ser um problema matemático complexo.

- Determinar a ordem do tempo de execução, sem preocupação com o valor da constante envolvida, pode ser uma tarefa mais simples.

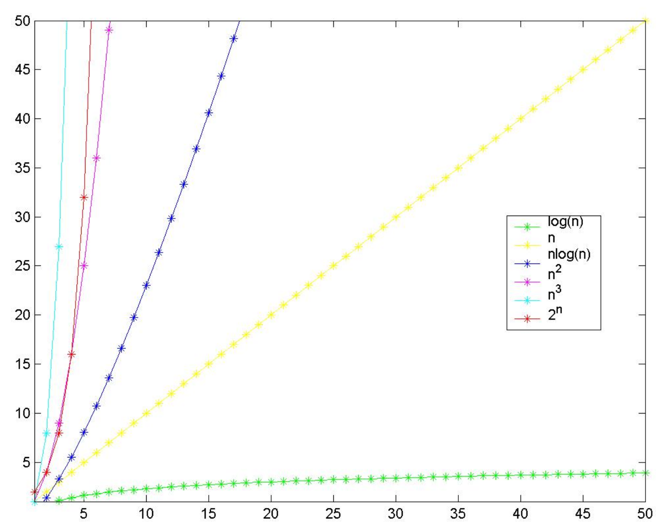
a) Hierarquia de funções

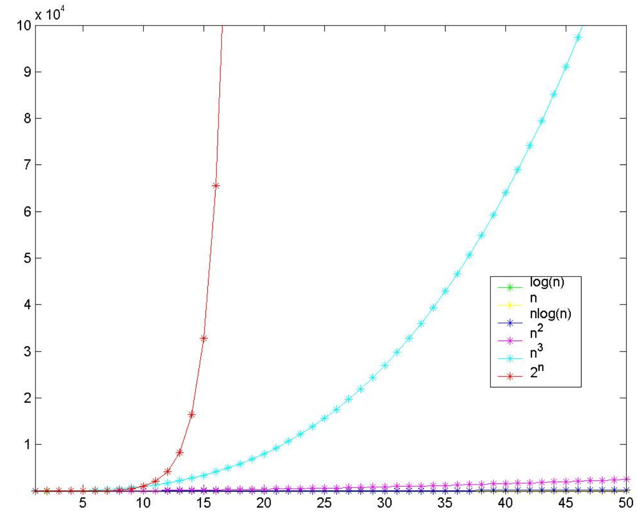
- A seguinte hierarquia de funções pode ser definida do ponto de vista assintótico:

$$1 \prec \log\log n \prec \log n \prec n^{\varepsilon} \prec n \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n}$$

onde ε e c são constantes arbitrárias com $0 < \varepsilon < 1 < c$

$$f(n) \prec g(n) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$





b) Relações importantes

- Logaritmos e Exponenciais:

$$\log_a a^x = x$$

$$a^0 = 1 \implies \log_a 1 = 0$$

$$a^{x+y} = a^x \times a^y \implies \log_a p + \log_a q = \log_a pq$$

$$a^{x-y} = \frac{a^x}{a^y} \implies \log_a \frac{p}{q} = \log_a p - \log_a q$$

$$(a^x)^y = a^{xy} \implies \log_a x^y = y \log_a x$$

$$(a^x)^y = a^{xy} \implies \log_a x = \frac{\log_b x}{\log_b a}$$

- Aproximação de Stirling:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

c) Somatórios

Um somatório é a soma de uma sequência de números (série).

Ex: $\sum_{i=a}^{b} i = a + a + 1 + a + 2 + ... + b$

Propriedades

a) $\sum_{i} f(i) + g(i) = \sum_{i} f(i) + \sum_{i} g(i)$

b) $\sum_{i}\sum_{j} f(i)g(j) = (\sum_{i} f(i))(\sum_{j} g(j))$

c) $\sum_{i} kf(i) = k\sum_{i} f(i)$

d) $\sum_{i=a}^{b} f(i) = \sum_{i=a}^{c} f(i) + \sum_{i=c+1}^{b} f(i), a < c < b$

- número de termos = limite superior - limite inferior +1. Se o somatório variar de forma decrescente, inverter os limites.

Progressão aritmética

$$\sum_{i=a}^{b} i = a + a + 1 + ... + b - 1 + b = (b - a + 1)(\frac{b + a}{2})$$

Ex: $\sum_{i=0}^{n} 2i$

Solução:

$$\sum_{i=0}^{n} 2i = 2\sum_{i=0}^{n} i = 2(n+1)n/2 = n2 + n$$

<u>Progressão geométrica</u>

$$\sum_{i=a}^{b} k^i = k^a + k^{a+1} + \ldots + k^{b-1} + k^b = k^a\left(\frac{k^{b-a+1}-1}{k-1}\right)$$

Ex: $\sum_{i=1}^{n} 2^{-i}$

Solução:

$$\sum_{i=1}^{n} 2^{-i} = \sum_{i=1}^{n} (\tfrac{1}{2})^i = \tfrac{1}{2}((\tfrac{1}{2})^n - 1)/(\tfrac{1}{2} - 1) = 1 - (\tfrac{1}{2})^n = 1 - \frac{1}{2^n}$$

<u>Quadrados e Cubos</u>

$$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$$

$$\sum_{i=0}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

Expansão Binomial e Combinatórios

$$(x+y)^n = \binom{n}{n} x^n + \binom{n}{n-1} x^{n-1}y + \binom{n}{n-2} x^{n-2}y^2 + \cdots + \binom{n}{1} xy^{n-1} + \binom{n}{0} y^n$$

0	1
1	1 1
2	1 2 1
3	1 3 3 1
4	1 4 6 4 1
5	1 5 10 10 5 1
6	1 6 15 20 15 6 1
7	1 7 21 35 35 21 7 1
8	1 8 28 56 70 56 28 8 1

$$\binom{n}{r} = \frac{n!}{(n-r)!\,r!} = \binom{n}{n-r}$$

$$\binom{n}{r} = \binom{n-1}{r} + \binom{n-1}{r-1}, \quad n>=2$$

Mudança de variável

- Se o contador do laço não variar de uma em uma unidade é necessário fazer uma mudança de variável para escrever o somatório.

  Ex:

  ```
  x=0;
  enquanto x<=n faça
     opCritica;
     x=x+2;
  fim;
  ```

  x varia de 0 a n com passo 2: 0,2,4...n. Fazer um somatório com índice y=x/2, variando de 0 a n/2 com passo 1.

- Ex:

```
x=1;
enquanto x<=n faça
   opCritica;
   x=x*2;
fim;
```

x varia de 1 a n de forma exponencial: 1,2,4,8...n. Fazer um somatório com índice y=lgx, variando de 0 a lgn.

Observe que toda ocorrência da variável original dentro do laço deve ser substituída pela nova.

- Ex:

```
x=1;
enquanto x<=n faça
   opCritica;
   repita para z=0 até x
         opCritica;
   x=x*2;
fim;
```

- x varia de 1 a n de forma exponencial: 1,2,4,8...n. Fazer um somatório com índice y=lgx, variando de 0 a lgn.
- z varia de 0 a x. Fazer um somatório com z variando de 0 a 2^y.

d) Análise do Tempo de Execução

- Comando de atribuição, de leitura ou de escrita: $\Theta(1)$.

- Sequência de comandos: determinado pelo maior tempo de execução de qualquer comando da sequência.

- Comando de decisão: tempo dos comandos dentro do comando condicional, mais tempo para avaliar a condição.

- Anel: soma do tempo de execução do corpo do anel mais o tempo de avaliar a condição para terminação (geralmente $\Theta(1)$), multiplicado pelo número de iterações.

- Procedimentos não recursivos: cada um deve ser computado separadamente um a um, iniciando com os que não chamam outros procedimentos. Avaliam-se então os que chamam os já avaliados (utilizando os tempos destes). O processo é repetido até chegar no programa principal.

Exemplo 1:

Considerando que a operação relevante seja o número de atribuições à variável a, qual é a função de complexidade da função exemplo1? Qual sua ordem de complexidade?

```
void exemplo1 (int n)
{
   int i, a;
   a=0;
   for (i=0; i<n; i++)
            a+=i;
}
```

Solução:

$$f(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1 = \Theta(n)$$

<u>Exemplo 2:</u>

Considerando que a operação relevante seja o número de atribuições à variável a, qual é a função de complexidade da função exemplo2? Qual sua ordem de complexidade?

```
void exemplo2 (int n)
{
   int i,j,a;
   a=0;
   for (i=0; i<n; i++)
      for (j=n; j>i; j--)
         a+=i+j;
   exemplo1(n);
}
```

Solução:

$$f(n) = 1 + \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 1 + n + 1 = n + 2 + \sum_{i=0}^{n-1} (n - i)$$

$$f(n) = n + 2 + \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i = n + 2 + n^2 - n(n-1)/2$$

$$f(n) = n + 2 + n^2 - \frac{n^2}{2} + \frac{n}{2} = \frac{n^2}{2} + \frac{3n}{2} + 2 = \Theta(n^2)$$

Exemplo: Procedimento não Recursivo

Algoritmo para ordenar os n elementos de um conjunto A em ordem ascendente:

```
void Ordena (int n, int * A) {
   int i , j , min, x;
(1)  for (i=0; i< n – 1; i++) {
(2)     min = i;
(3)     for (j= i +1; j<n ; j++)
(4)        if (A[ j ] < A[min])
(5)           min = j ;
(6)      x = A[min] ;     // troca A[min] e A[ i ]
(7)      A[min] = A[ i ] ;
(8)      A[ i ] = x;
     }
}
```

- Seleciona o menor elemento do conjunto.
- Troca este com o primeiro elemento A[1].
- Repita as duas operações acima com os n -1 elementos restantes, depois com os n - 2, até que reste apenas um.

Análise do Procedimento não Recursivo – Pior caso:

- Anel Interno:
    - Contém um comando de decisão, com um comando apenas de atribuição. Ambos levam tempo constante para serem executados.

- Quanto ao corpo do comando de decisão, devemos considerar o pior caso, assumindo ser sempre executado.
- O tempo para incrementar o índice do anel e avaliar sua condição de terminação é $\Theta(1)$.
- O tempo combinado para executar uma vez o anel é $\Theta(\max(1, 1, 1)) = \Theta(1)$, conforme regra da soma para a notação.
- Como o número de iterações é n – i – 1, o tempo gasto no anel é $\Theta((n - i) \times 1) = \Theta(n - i)$, conforme regra do produto para a notação.

• Anel Externo

- Contém, além do anel interno, quatro comandos de atribuição. $\Theta(\max(1, (n - i), 1, 1, 1)) = \Theta(n - i)$.
- A linha (1) é executada n -1 vezes, e o tempo total para executar o programa está limitado ao produto de uma constante pelo **somatório** de (n – i-1):

$$\sum_{i=0}^{n-2}(n-i-1) = n^2/2 - n/2 = \Theta(n^2)$$

- Se considerarmos o número de comparações como a medida de custo relevante, o programa faz $(n^2)/2 - n/2$ comparações para ordenar n elementos.
- Se considerarmos o número de trocas, o programa realiza exatamente n - 1 trocas.

Exercício:

- Como seria a análise do melhor caso e caso médio?

- Qual a ordem de complexidade do algoritmo de modo geral?

- Considerando como crítico o comando para seleção do índice mínimo (min=...), como seria a análise de melhor caso, pior caso e caso médio?

Exercícios:

1. O que faz essa função ? Qual é a operação relevante? Qual a sua ordem de complexidade?

```
void p1 (int n)
{
  int i, j, k;

  for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
          C[i][j]=0;
            for (k=n-1; k>=0; k--)
              C[i][j]=C[i][j]+A[i][k]*B[k][j];
    }
}
```

2. O que faz essa função ? Qual é a operação relevante? Qual a sua ordem de complexidade?

```
void p2 (int n)
{
  int i, j, x, y;

      x = y = 0;
      for (i=1; i<=n; i++) {
         for (j=i; j<=n; j++)
              x = x + 1;
         for (j=1; j<i; j++)
              y = y + 1;
  }
}
```

3. Qual é a função de complexidade para o número de atribuições ao vetor x?

```
void Exercicio3(int n){
   int i, j, a;
   for (i=0; i<n; i++){
      if (x[i] > 10)
         for (j=i+1; j<n; j++)
            x[j] = x[j] + 2;
      else {
         x[i] = 1;
         j = n-1;
         while (j >= 0) {
            x[j] = x[j] - 2;
            j = j - 1;
         }
      }
   }
}
```

2. Técnica de projeto: Força bruta

É a mais simples das técnicas de projeto.

- Solução direta, geralmente baseada no enunciado do problema. Pode ser recursiva, mas na maioria das vezes é iterativa.

- É fácil de aplicar e muitas vezes surge como idéia intuitiva e pouco elaborada.

- Pode exigir grande esforço computacional, mas os algoritmos são fáceis de entender.

- Muitas vezes são uma primeira versão para soluções mais elaboradas.

- Aplicável a uma ampla variedade de problemas. Exemplos: cálculo do fatorial de um número, busca sequencial, ordenação pelo método da bolha, multiplicação de matrizes.

- Útil para o desenvolvimento rápido de algoritmos que operem sobre uma entrada pequena ou que serão executados poucas vezes.

A) Exemplo: BubbleSort

Para i:=1 até n –1
    Para j:=1 até n – i
        Se v [j +1] < v [j] então troque v [j] com v [j+1]

Análise do algoritmo em função do número de comparações:

(Melhor=pior caso)

$$
\begin{aligned}
f(n) &= \sum_{i=1}^{n-1}\sum_{j=1}^{n-i} 1 \\
&= \sum_{i=1}^{n-1}(n-i) = n\sum_{i=1}^{n-1} 1 - \sum_{i=1}^{n-1} i = n(n-1) - \frac{(n-1)n}{2} \\
&= \frac{n^2-n}{2} = \Theta(n^2)
\end{aligned}
$$

O algoritmo é ótimo?

B) Exemplo: Casamento de padrões em strings

String s[1..n], padrão p[1..m], m<n:

```
Para i:=1 até n – m +1
        j:=1
        Enquanto j<=m e p [ j ] = s [ j + i -1]
                j:= j + 1
        Se j > m retorne i
```

Análise do algoritmo em função do número de comparações entre elementos dos strings:

(Pior caso)

$$f(n) = \sum_{i=1}^{n-m+1} \sum_{j=1}^{m} 1 = m \sum_{i=1}^{n-m+1} 1 = m(n-m+1) = O(mn)$$

O algoritmo é ótimo?

Existem algoritmos de força bruta que são ótimos?

C) Exemplo: Busca exaustiva

- Aplicado a problemas de otimização com número de soluções exponencial. Todas as possíveis soluções são geradas e a melhor é selecionada.
- Caixeiro viajante, problema da mochila, preenchimento de containers, etc.

3. Técnica de projeto: Transformar e conquistar

Esta técnica compreende dois estágios:

a) No estágio de transformação, a instância do problema é transformada para ser mais fácil encontrar uma solução.
b) No segundo estágio, a instância transformada é resolvida.

Existem 3 variações da técnica:

- **Simplificação**: transformação para uma instância mais simples ou conveniente do mesmo problema. Exemplo: pré-ordenação.
- **Mudança de representação**: transformação para uma representação diferente, na qual o problema é mais facilmente resolvido. Exemplos: heapsort, transformada rápida de Fourier.
- **Redução**: transformação para uma instância de um problema diferente para o qual já existe um algoritmo eficaz. Exemplo: problemas de grafos.

A técnica pode ser usada para o projeto de algoritmos recursivos e não recursivos.

Exemplo: Pré-ordenação

Muitos problemas envolvendo listas são mais simples de serem resolvidos quando a lista já está ordenada.

Exemplo: verificar se existem elementos repetidos em um arranjo.

- Por força bruta, o algoritmo é $\Theta(n^2)$ no pior caso ($O(n^2)$ no caso geral).
- Alternativa: ordenar o arranjo e verificar elementos adjacentes $\Theta(n \log n + n) = \Theta(n \log n)$ no pior caso.

Exercício: escrever uma solução para o problema de encontrar a moda de uma lista, utilizando a técnica de:

a) força bruta
b) transformação

Fazer a análise das soluções

4. Técnica de projeto: Decrementar e conquistar

A técnica, também chamada indutiva ou incremental, se baseia na seguinte estratégia:

- Reduzir a instância do problema para uma instância menor do mesmo problema
- Resolver a instância menor
- Estender a instância menor para obter a solução para o problema original

A técnica pode ser usada para o projeto de algoritmos recursivos e não recursivos.

Exemplo: Ordenação por inserção

Problema: Ordenar um conjunto de $n \geq 1$ inteiros.

Hipótese de Indução: Sabemos ordenar um conjunto de $n-1 \geq 1$ inteiros.

- Caso base: $n = 1$. Um conjunto de um único elemento está ordenado.
- Passo da Indução: Seja S um conjunto de $n \geq 2$ inteiros e x um elemento qualquer de S. Por hipótese de indução, sabemos ordenar o conjunto $S - x$, basta então inserir x na posição correta para obtermos S ordenado.

Inserção (A, n)

- Entrada: Vetor *A* de *n* números inteiros.
- Saída: Vetor *A* ordenado.

```
se n ≥2 faça
    Inserção (A, n − 1)
    v := A[n]
    j := n − 1
    enquanto (j >= 1) e (A[j ] > v) faça
        A[j + 1] := A[j ]
        j := j − 1
    A[j + 1] := v
```

É fácil eliminar o uso de recursão simulando-a com um laço:

Inserção (A)

- Entrada: Vetor *A* de *n* números inteiros.
- Saída: Vetor *A* ordenado.

```
para i := 2 até n faça
    v := A[i ]
    j := i −1
    enquanto (j >= 1) e (A[j ] > v) faça   //compara
        A[j + 1] := A[j ]  //troca
        j := j −1
    A[j + 1] := v  //troca
```

<u>Exercício</u>: Quantas comparações e quantas trocas o algoritmo executa no pior caso?