TÉCNICAS DE PROGRAMAÇÃO AVANÇADA

Série 7 – Análise de Algoritmos

Medida de tempo de execução de um programa

- Análise de um algoritmo particular.
  - Qual é o custo de usar um dado algoritmo para resolver um problema específico?
  - Características que devem ser investigadas:
    - análise do número de vezes que cada
    - parte do algoritmo deve ser executada,
    - estudo da quantidade de memória necessária.

Medida de tempo de execução de um programa

- Análise de uma classe de algoritmos.
  - Qual é o algoritmo de menor custo possível para resolver um problema particular?
  - Toda uma família de algoritmos é investigada.
  - Procura-se identificar um que seja o melhor possível.
  - Coloca-se limites para a complexidade computacional dos algoritmos pertencentes à classe.

Medida de tempo de execução de um programa

- Medida de dificuldade para resolver um problema:
  - menor custo possível para resolver problemas de uma dada classe,
- Algoritmo ótimo : Quando o custo de um algoritmo é igual ao menor custo possível para a medida de custo considerada.
- Podem existir vários algoritmos para resolver o mesmo problema.
  - Se a mesma medida de custo é aplicada a diferentes algoritmos, então é possível compará-los e escolher o mais adequado.

Medida de tempo de execução de um programa

- Medida do Custo pela Execução do Programa
  - os resultados são dependentes do hardware, sistema operacional e compilador
- Medida do Custo por meio de um Modelo Matemático
  - Modelo matemático baseado em um computador idealizado.
  - Deve ser especificado o conjunto de operações e seus custos de execuções.

Funções de Complexidade

- A função de **complexidade tempo f(n)** é a medida do tempo necessário para executar um algoritmo para um problema de tamanho n.
- Quando o critério é espaço, temos que a função de **complexidade de espaço f(n)** é a medida da memória necessária para executar um algoritmo em um problema de tamanho n.

Complexidade de tempo

- Utilizaremos f para denotar uma função de complexidade de tempo.
- A complexidade de tempo na realidade não representa tempo diretamente, mas o número de vezes que determinada operação considerada relevante é executada.

Exemplo – maior Elemento

- Considere o algoritmo para encontrar o maior elemento de um vetor de inteiros

```
A[1..n], n >=  1.
function Max (var A: Vetor ) : integer ;
var i , Temp: integer ;
begin
Temp := A[1] ;
for i := 2 to n do
 i f (Temp < A[ i ]) then
    Temp := A[ i ] ;
 Max := Temp;
end;
```

- Seja f uma função de complexidade tal que f(n) é o número de comparações entre os elementos de A, se A contiver n elementos.
- Logo, $f(n) = n-1$, para $n>0$
- Cada um dos n - 1 elementos tem de ser mostrado, por meio de comparações, que é menor do que algum outro elemento.
  - Assim são necessárias pelo menos n-1 comparações para resolver esse problema.
  - Conclusão: o algoritmo apresentado é ótimo tomando como parâmetros o número de comparações.

Dados de Entrada

- É comum considerar o tempo de execução de um programa como uma função do tamanho da entrada.
- Para alguns algoritmos, o custo de execução é uma função da entrada particular dos dados, não apenas do tamanho da entrada.
  - A função Max tem custo uniforme sobre todos os problemas de tamanho n.
  - Já para um algoritmo de ordenação isso não ocorre: o algoritmo pode ter que trabalhar menos.

Melhor caso, caso médio, pior caso

- **Melhor caso: menor tempo de execução** sobre todas as entradas de tamanho n.
- **Pior caso: maior tempo de execução sobre** todas as entradas de tamanho n.
- Se f é uma função de complexidade baseada na análise de pior caso, o custo de aplicar o algoritmo nunca é maior do que f(n).
- **Caso médio (ou caso esperado): média dos** tempos de execução de todas as entradas de tamanho n.

Caso médio

- Na análise do caso esperado, supõe-se uma **distribuição de probabilidades sobre o** conjunto de entradas de tamanho n e o custo médio é obtido com base nessa distribuição.
- É comum supor uma distribuição de probabilidades em que todas as entradas possíveis são igualmente prováveis.
- Na prática isso nem sempre é verdade.

Exemplo

- Recuperação de registros em arquivos desordenados. Algoritmo de busca seqüencial.
- f é uma função de complexidade tal que f(n) é o número de registros consultados no arquivo (número de vezes que a chave de consulta é comparada com a chave de cada registro).
- melhor caso: $f(n) = 1$ (registro procurado é o primeiro consultado);
- pior caso: $f(n) = n$ (registro procurado é o último consultado ou não está presente no arquivo);
- caso médio: $f(n) = (n + 1)/2$.

Exemplo

- Caso médio – considerando que toda pesquisa recupera um registro.
- Se p_i for a probabilidade de que o i-ésimo registro seja procurado, e considerando que para recuperar o i-ésimo registro são necessárias i comparações, então

$$f(n) = 1 \times p_1 + 2 \times p_2 + 3 \times p_3 + \cdots + n \times p_n.$$

Exemplo

- Para calcular f(n) basta conhecer a distribuição de probabilidades p_i.
- Se cada registro tiver a mesma probabilidade de ser acessado que todos os outros, então

$$p_i = 1/n, 1 \leq i \leq n.$$

- e

$$f(n) = \frac{1}{n}(1+2+3+\cdots+n) = \frac{1}{n}\left(\frac{n(n+1)}{2}\right) = \frac{n+1}{2}.$$

Regras para determinar o tempo de execução de um programa

- Comando de atribuição, de leitura ou de escrita: O(1).
- Sequência de comandos:
  - maior tempo de execução de qualquer comando da seqüência.
- Comando de decisão:
  - tempo dos comandos dentro do comando condicional,
  - mais tempo para avaliar a condição, que é O(1).
- Anel: soma do tempo de execução do corpo do anel
  - mais o tempo de avaliar a condição para terminação (geralmente O(1)),
  - multiplicado pelo número de iterações.

Comportamento Assintótico de funções

- O parâmetro n fornece uma medida da dificuldade para se resolver o problema.
- Para valores suficientemente pequenos de n, qualquer algoritmo custa pouco para ser executado, mesmo os ineficientes.
- A **escolha do algoritmo não é um problema** crítico para problemas de tamanho pequeno.
- Logo, a análise de algoritmos é realizada para valores grandes de n.

Exercício

- Identificar pelos menos três funcionalidades presentes em sistemas de informação que operam sobre grandes entradas de dados e cujos desempenhos são altamente influenciados pelos algoritmos empregados.

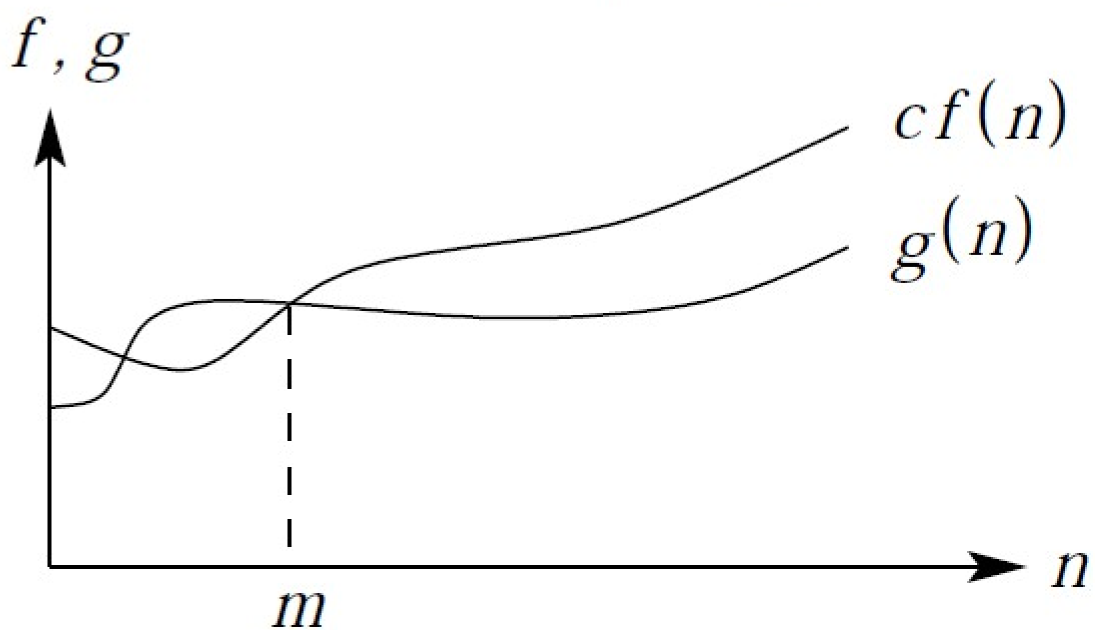
Comportamento Assintótico de funções

- Estuda-se o comportamento assintótico das **funções de custo (comportamento de suas** funções de custo para valores grandes de n)
- O comportamento assintótico de f(n) representa o limite do comportamento do custo quando n cresce.

Dominação Assintótica

- Uma função f(n) **domina assintoticamente outra função g(n) se** existem duas constantes positivas c e m tais que, para n>= m, temos

  $|g(n)| <= c \times |f(n)|.$



Exemplo

- Sejam $g(n) = (n+1)^2$ e $f(n) = n^2$.
- As funções $g(n)$ e $f(n)$ dominam assintoticamente uma a outra, desde que $|(n+1)^2| \leq 4|n^2|$ para $n \geq 1$ e $|n^2| \leq |(n+1)^2|$ para $n \geq 0$.

Notação *O*

- $g(n) = O(f(n))$ significa que $f(n)$ domina assintoticamente $g(n)$. Lê-se $g(n)$ é da ordem no máximo $f(n)$.
- Exemplo: quando dizemos que o tempo de execução $T(n)$ de um programa é $O(n^2)$, significa que existem constantes c e m tais que, para valores de n m, $T(n) = cn^2$.
- Genericamente, Uma função $g(n)$ é $O(f(n))$ se existem duas constantes positivas c e m tais que $g(n)$ $cf(n)$, para todo n >= m.

Exercícios

- Verificar se
  - $g(n) = 3n^3 + 2n^2 + n$ é $O(n^3)$.
  - $g(n) = \log_5 n$ é $O(\log n)$.
  - $g(n) = (n + 1)^2$ é $O(n^2)$.

Operações com a notação O

$$
\begin{array}{rcll}
f(n) & = & O(f(n)) & \\
c \times O(f(n)) & = & O(f(n)) & c = constante \\
O(f(n)) + O(f(n)) & = & O(f(n)) & \\
O(O(f(n)) & = & O(f(n)) & \\
O(f(n)) + O(g(n)) & = & O(max(f(n), g(n))) & \\
O(f(n))O(g(n)) & = & O(f(n)g(n)) & \\
f(n)O(g(n)) & = & O(f(n)g(n)) &
\end{array}
$$

Regras para determinar o tempo de execução de um programa

- **Procedimentos não recursivos:**
  - cada um deve ser computado separadamente um a um, iniciando com os que não chamam outros procedimentos. Avalia-se então os que são
- chamam os já avaliados (utilizando os tempos desses). O processo é repetido até chegar no programa principal.
- **Procedimentos recursivos:**
  - associada uma função de complexidade f(n) desconhecida, onde n mede o tamanho dos argumentos.

Exemplo- Procedimento Não recursivo

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

```
procedure Ordena (var A: Vetor ) ;
var i , j , min, x : integer ;
begin
(1) for i := 1 to n-1do
begin
(2) min := i ;
(3) for j := i +1 to n do
(4) i f A[ j ] < A[min]
(5) then min := j ;
{ troca A[min] e A[ i ] }
(6) x := A[min] ;
(7) A[min] := A[ i ] ;
(8) A[ i ] := x;
end;
end;
```

Exemplo- Procedimento Não recursivo

- 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 que será sempre executado.
  - O tempo para incrementar o índice do anel e avaliar sua condição de terminação é O(1).

Exemplo- Procedimento Não recursivo

- Anel interno
- O tempo combinado para executar uma vez o anel é $O(max(1; 1; 1)) = O(1)$, conforme regra da soma para a notação O.
- Como o número de iterações é n -i, o tempo
- gasto no anel é $O((n - i) \times 1) = O(n - i)$,
- conforme regra do produto para a notação O.

Exemplo- Procedimento não recursivo

- Anel externo
  - Contém, além do anel interno, quatro comandos de atribuição:
    - O(max(1; (n - i); 1; 1; 1)) = O(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:

$$\sum_1^{n-1}(n-i) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

Exemplo- Procedimento não recursivo

- 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.

Exercícios

- Determine a ordem de complexidade para os algoritmos:
- Busca de padrão
- Método da bolha
- Método da seleção

Método da Bolha

```
Algoritmo Bubble(V, n)
k = n-1
    para i = 1 até n faça
        j = 1
        enquanto j <= k faça
            se V[j] > V[j+1] então
                aux = V[j]
                V[j] = V[j+1]
                V[j+1] = aux
            j = j+1
```

Busca ordem normal

```
void busca(char *t, char *p){
    int i, j, k, m,n;
    n=strlen(t);
    m=strlen(p);
    for (i=0;i<n-m+1;i++){
       k=i;
       j=0;
       while((j<=m) && (t[k]==p[j])){
          j++;
          k++;
      }
      if (j>=m){
          printf("casamento na posição %d", i);
          break;
      }
   }
}
```

Método da Inserção

```
void insertionSort(int V[], int tam) {
  int i, j, aux;
  for(i = 1; i < tam; i++){
      j = i;
      while((j != 0) && (V[j] < V[j - 1])) {
         aux = V[j];
         V[j] = V[j - 1];
         V[j - 1] = aux; j--;
      }
   }
}
```