MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2015

Roteiro

1. Fundamentos de análise de algoritmos

2. Cálculo da função de custo

3. Exercícios

Fundamentos de análise de algoritmos

- Como analisar um algoritmo:
  - Contar o número de operações realizadas pelo algoritmo para uma dada entrada.
  - Expressar este número em função do tamanho da entrada (n).
- Exemplos de funções de custo de um algoritmo:
  - $f_1(n) = 5n$
  - $f_2(n) = 2n^2$
  - $f_3(n) = n \log n$

Fundamentos de análise de algoritmos

- A análise é sempre realizada em relação a um dado modelo computacional.
- No nosso caso, vamos considerar um modelo simplificado:
  - O computador tem um único processador.
  - Todos os acessos à memória têm o mesmo custo.
  - Instruções são executadas sequencialmente.
  - Não há instruções nem operações paralelas.
  - Todas as instruções têm custo similar (uma unidade).
- Geralmente estamos interessados em medir o uso dos recursos do sistema (tempo de processamento, uso de memória, largura de banda, consumo de energia, etc).
- Nesta aula vamos considerar como custo computacional apenas o tempo de processamento de um programa.

Fundamentos de análise de algoritmos

- Para exemplificar a diferença entre funções de custo, suponha:
  - Um computador com 1GHz.
  - Uma instrução executada a cada ciclo da máquina (1GHz $= 10^9$ instruções por segundo).
- Considere, por exemplo, que o tamanho da entrada é $n = 1000000$:
  - Algoritmo com custo $f(n) = n$:
    - O computador gastará aproximadamente 1 milissegundo.
  - Algoritmo com custo $f(n) = n \log n$:
    - O computador gastará aproximadamente 6 milissegundos.
  - Algoritmo com custo $f(n) = n\sqrt{n}$:
    - O computador gastará aproximadamente 1 segundo.
  - Algoritmo com custo $f(n) = n^2$:
    - O computador gastará aproximadamente 17 minutos.
  - Algoritmo com custo $f(n) = n^3$:
    - O computador gastará aproximadamente 32 anos.

Cálculo da função de custo

- Considere o número de atribuições realizadas por esta função:

```
void troca(int *x, int *y) {
  int aux;

  aux = *x;
  *x = *y;
  *y = aux;
}
```

- Função de custo:

  $f(n) = 2 + 3 = 5$

Cálculo da função de custo

- Considere o custo de inicialização de um vetor:

```
for (i = 0; i < n; i++)
  v[i] = 0;
```

- Função de custo:

`i = 0`:	$f(n) = 1$
`i < n`:	$f(n) = n + 1$
`i++`:	$f(n) = n$
`v[i] = 0`:	$f(n) = n$
Total:	$f(n) = 3n + 2$

Cálculo da função de custo

- Considere o número de multiplicações realizadas por esta função:

```
int fatorial(int n) {
  int fat = 1;

  for (i = 2; i <= n; i++)
    fat = fat * i;

  return fat;
}
```

- Função de custo:

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

Cálculo da função de custo

- Considere o número de somas entre elementos das matrizes:

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    c[i][j] = a[i][j] + b[i][j];
```

- Função de custo:

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

Cálculo da função de custo

- Considere o número de muliplicações entre elementos das matrizes:

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++) {
    c[i][j] = 0;
    for (k = 0; k < n; k++)
      c[i][j] = c[i][j] + (a[i][k] * b[k][j]);
  }
```

- Função de custo:

$$f(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3$$

Cálculo da função de custo

- Considere o número de chamadas à função `troca` usadas para inverter a ordem dos elementos de um vetor:

```
for (i = 0; i < n/2; i++)
  troca(a[i], a[n - i - 1]);
```

- Função de custo:

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

Cálculo da função de custo

- Considere o número de execuções do comando soma++:

```
soma = 0;
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    soma++;
```

- Função de custo:

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

Cálculo da função de custo

- Considere o número de execuções do comando `soma++`:

```
soma = 0;
for (i = 0; i < n; i++)
  for (j = 0; j < i; j++)
    soma++;
```

- Função de custo:

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

Cálculo da função de custo

- Considere o número de execuções do comando `soma++`:

```
soma = 0;
for (i = 0; i < n; i++)
  for (j = 0; j < n * n; j++)
    soma++;
```

- Função de custo:

$$f(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n^2-1} 1 = \sum_{i=0}^{n-1} n^2 = n^3$$

Cálculo da função de custo

- Considere o número de execuções do comando `soma++`:

```
soma = 0;
for (i = 1, k = 1; i <= n; i++, k = k * 2)
  for (j = 1; j <= k; j++)
    soma++;
```

- Função de custo:

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

Cálculo da função de custo

- Considere o número de execuções do comando soma++:

```
soma = 0;
for (i = 1, k = 2; i <= n; i++, k = k * 2)
  for (j = 1; j <= n/k; j++)
    soma++;
```

- Função de custo:

$$f(n) = \sum_{i=1}^{n} \sum_{j=1}^{\lfloor n/2^i \rfloor} 1 = \sum_{i=1}^{n} \left\lfloor \frac{n}{2^i} \right\rfloor \leq \sum_{i=1}^{n} \frac{n}{2^i} = n \sum_{i=1}^{n} \frac{1}{2^i} \leq n \sum_{i=1}^{\infty} \frac{1}{2^i} = n$$

Cálculo da função de custo

- Considere o número de execuções do comando soma++:

```
soma = 0;
for (i = 2; i <= n; i++)
  for (j = 1; j <= n/i; j++)
    soma++;
```

- Função de custo:

$$f(n) = \sum_{i=2}^{n} \sum_{j=1}^{\lfloor n/i \rfloor} 1 = \sum_{i=2}^{n} \left\lfloor \frac{n}{i} \right\rfloor \leq \sum_{i=2}^{n} \frac{n}{i} = n \sum_{i=2}^{n} \frac{1}{i} \leq n \ln n$$

Exercícios

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++) {
    if (i < j)
      X(n, i, j);
    if (i == j)
      for (k = j + 1; k < n; j++)
        Y(n, i, j);
    else
      Z(n, i, j);
}
```

Considerando o trecho de código acima, responda:

- Quantas vezes a função `X` é executada?
- Quantas vezes a função `Y` é executada?
- Quantas vezes a função `Z` é executada?

Exercícios

```
for (i = 0; i < n; i++)
  for (j = i + 1; j < n; j++) {
    teste(vetor, i, j);

    for (k = j + 1; k < n; k++)
      for (l = k + 1; l < n; l++)
        teste(vetor, k, l);
  }
```

Considerando o trecho de código acima, calcule:

- Um limitante inferior para o número de execuções da função `teste`.
- Um limitante superior para o número de execuções da função `teste`.

Exercícios

```
for (i = 0; i < n; i++) {
  for (j = 0; j < i; j++)
    for (k = j + 1; k < i; k++)
      teste(vetor, j, k);

  for (j = i; j < n; j++)
    for (k = j + 1; k < n; k++)
      teste(vetor, j, k);
}
```

Considerando o trecho de código acima, calcule:

- Um limitante inferior para o número de execuções da função `teste`.
- Um limitante superior para o número de execuções da função `teste`.