Eficiência de algoritmos e programas

Graziela Araújo

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

Motivação

- Tenho mais de um algoritmo/programa para solucionar um problema computacional. Qual devo escolher?
  - Aquele que gasta *menos* tempo?
  - Aquele que gasta *menos* memória?
  - Aquele que faz *menos* comunicação entre processos?
  - Aquele que usa/acessa *menos* portas lógicas?
- Estamos interessados no “tempo”!

Algoritmos e programas

- Algoritmo ou programa é uma seqüência bem definida de passos (descritos em uma linguagem de programação específica) que transforma um conjunto de valores – a entrada – em um outro conjunto de valores – a saída
- Algoritmo é uma ferramenta para solucionar um problema computacional Se um programa pára com a resposta correta então dizemos que o programa é correto
- Um programa correto soluciona o problema computacional associado
- Um programa incorreto pode sequer parar, para alguma entrada, ou pode parar mas com uma resposta indesejada

Definições

- Analisar um algoritmo significa predizer os recursos computacionais que o algoritmo requer quando da sua execução: memória, tempo de processamento.
- **Em geral**, existem vários algoritmos para solucionar um mesmo problema e a análise é capaz de identificar qual é o mais eficiente.

Definições

- Interesse: obter uma expressão ou fórmula matemática que represente o tempo de execução de um algoritmo.
- Aspectos mais importantes da análise de tempo:
  - Quantidade de elementos a processar (tamanho da entrada).
  - Forma como os elementos estão dispostos na entrada.
- **Tempo de execução de um algoritmo**: uma função $f(n)$, onde n é o tamanho da entrada.
- A função f deve expressar o número de operações básicas, ou *passos*, executados pelo algoritmo.
- A operação básica de maior frequência de execução no algoritmo é denominada *operação dominante* ou *operação fundamental*.

Problema da busca

Dado um número inteiro n, com $1 \leqslant n \leqslant 100$, um conjunto C de n números inteiros e um número inteiro x, verificar se x encontra-se no conjunto C

Algoritmos e programas

```
#include <stdio.h>
#define MAX 100
int main(void)
{
   int n, i, C[MAX], x;

   scanf("%d", &n);
   for (i = 0; i < n; i++)
      scanf("%d", &C[i]);
   scanf("%d", &x);

   for (i = 0; i < n && C[i] != x; i++)
      ;
   if (i < n)
      printf("%d está na posição %d de C\n", x, i);
   else
      printf("%d não pertence ao conjunto C\n", x);
   return 0;
}
```

Análise de algoritmos

- Fixar um modelo de computação: máquina de acesso aleatório

Análise de algoritmos

- O tempo gasto por um algoritmo cresce com o tamanho da entrada, isto é, o número de itens na entrada
- O tempo de execução de um algoritmo sobre uma entrada particular de tamanho n é o número de passos realizados por ele com base em n
  - Procurar um elemento x em um conjunto C com milhares de elementos certamente gasta mais tempo que em um conjunto C com 3 elementos
  - Mesmo para dois conjuntos diferentes mas com a mesma quantidade de elementos, uma busca pode ser mais demorada que a outra

Análise de Algoritmos

- Para calcular o custo do algoritmo de busca visto, devemos multiplicar o custo e o número de vezes que cada linha é executada:
  - Uma linha i de um algoritmo gasta uma quantidade constante de tempo $c_i > 0$
- Melhor e pior caso
- O tempo de execução $T(n)$ do algoritmo é dado pela soma dos tempos para cada instrução executada.

Análise de algoritmos

	Custo	Vezes
#include <stdio.h>	c_1	1
#define MAX 100	c_2	1
int main(void)	c_3	1
{	0	1
int n, i, C[MAX], x;	c_4	1
scanf("%d", &n);	c_5	1
for (i = 0; i < n; i++)	c_6	$n+1$
scanf("%d", &C[i]);	c_7	n
scanf("%d", &x);	c_8	1
for (i = 0; i < n && C[i] != x; i++)	c_9	t_i
;	0	$t_i - 1$
if (i < n)	c_{10}	1
printf("%d está na posição %d de C\n", x, i);	c_{11}	1
else	c_{12}	1
printf("%d não pertence ao conjunto C\n", x);	c_{13}	1
return 0;	c_{14}	1
}	0	1

- O tempo de execução $T(n)$ do algoritmo é dado pela soma dos tempos para cada sentença executada
- Ou seja, devemos somar os produtos das colunas ***Custo*** e ***Vezes***

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} .$$

- Tempo de execução $T(n)$ do algoritmo da busca:

$$\begin{aligned} T(n) = {} & c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ & c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14}\,. \end{aligned}$$

- Melhor caso: ocorre se x encontra-se na primeira posição do vetor C; então, $t_i = 1$ e assim

$$\begin{aligned} T(n) = {} & c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ & c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ = {} & (c_6 + c_7)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ = {} & an + b\,. \end{aligned}$$

- Tempo de execução $T(n)$ do algoritmo da busca:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14}\,.$$

- Pior caso: ocorre se x não se encontra no vetor C; então, $t_i = n + 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ &\quad c_9(n+1) + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7 + c_9)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b\,. \end{aligned}$$

Análise de algoritmos

- Estamos interessados no tempo de execução de pior caso de um algoritmo

Ordem de crescimento de funções

- Estamos interessados na taxa de crescimento ou ordem de crescimento de uma função que descreve o tempo de execução de um algoritmo
- Consideramos apenas o *maior* termo da fórmula e ignoramos constantes
- Dizemos assim que o algoritmo da busca tem tempo de execução de pior caso $O(n)$
- Um algoritmo é mais eficiente que outro se seu tempo de execução de pior caso tem ordem de crescimento menor
- Eficiência assintótica de um algoritmo: como a função que descreve seu tempo de execução cresce com o tamanho da entrada *no limite*

Ordem de crescimento de funções

Definição

Para uma dada função $g(n)$, denotamos por $O(g(n))$ o conjunto de funções

$$O(g(n)) = \{f(n) \colon \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leqslant f(n) \leqslant cg(n) \text{ para todo } n \geqslant n_0\} \,.$$

GRÁFICO!

- $f(n)$ pertence ao conjunto $O(g(n))$ se existe uma constante positiva c tal que $f(n)$ "não seja maior que" $cg(n)$, para n suficientemente grande
- Usamos a notação O para fornecer um limitante assintótico superior sobre uma função, dentro de um fator constante
- Usamos $f(n) = O(g(n))$ para $f(n) \in O(g(n))$

$$O(n^2) = \{\tfrac{3}{2}n^2 - 1,\ 2n^2, 3n^2 - 4, 7n^2 + 5n - 2, ...\}$$

Será que $4n + 1 = O(n)$?

Existem constantes positivas c e n_0 tais que

$$4n + 1 \leqslant cn$$

para todo $n \geqslant n_0$. Tome $c = 5$ e então

$$\begin{aligned} 4n + 1 &\leqslant 5n \\ 1 &\leqslant n\,, \end{aligned}$$

ou seja, para $n_0 = 1$, a desigualdade $4n + 1 \leqslant 5n$ é satisfeita para todo $n \geqslant n_0$ e assim, $4n + 1 = O(n)$

Análise da ordenação por trocas sucessivas

```
void trocas_sucessivas(int n, int v[MAX])
{
   int i, j, aux;

   for (i = n-1; i > 0; i--)
      for (j = 0; j < i; j++)
         if (v[j] > v[j+1]) {
            aux = v[j];
            v[j] = v[j+1];
            v[j+1] = aux;
         }
}
```

Análise da ordenação por trocas sucessivas

	Custo	Vezes
`void trocas_sucessivas(int n, int v[MAX]`	c_1	1
`{`	0	1
`   int i, j, aux;`	c_2	1
`   for (i = n-1; i > 0; i--)`	c_3	n
`      for (j = 0; j < i; j++)` **j=0,1,...,i**	c_4	$\sum_{i=1}^{n-1}(i+1)$
`         if (v[j] > v[j+1]) {`	c_5	$\sum_{i=1}^{n-1} i$
`            aux = v[j];`	c_6	$\sum_{i=1}^{n-1} t_i$
`            v[j] = v[j+1];`	c_7	$\sum_{i=1}^{n-1} t_i$
`            v[j+1] = aux;`	c_8	$\sum_{i=1}^{n-1} t_i$
`         }`	0	$\sum_{i=1}^{n-1} i$
`}`	0	1

Análise da ordenação por trocas sucessivas

- Melhor caso: quando a seqüência de entrada com n números inteiros é fornecida em ordem crescente ($t_i = 0$ para todo i)

$$
\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} 0 \\
&= n + 2\,\frac{n(n-1)}{2} + n - 1 \\
&= n^2 + n - 1
\end{aligned}
$$

Análise da ordenação por trocas sucessivas

- Melhor caso: $T(n) = n^2 + n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$n^2 + n - 1 \leqslant cn^2 , \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 2$ temos:

$$\begin{aligned} n^2 + n - 1 &\leqslant 2n^2 \\ n - 1 &\leqslant n^2 \end{aligned}$$

ou seja,

$$n^2 - n + 1 \geqslant 0 .$$

- A inequação $n^2 - n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$

Análise da ordenação por trocas sucessivas

- Assim, escolhendo $c = 2$ e $n_0 = 1$, temos que

$$n^2 + n - 1 \leqslant cn^2$$

para todo $n \geqslant n_0$, onde $c = 2$ e $n_0 = 1$

- Portanto, $T(n) = O(n^2)$

Análise da ordenação por trocas sucessivas

- Assim, escolhendo $c = 2$ e $n_0 = 1$, temos que

$$n^2 + n - 1 \leqslant cn^2$$

  para todo $n \geqslant n_0$, onde $c = 2$ e $n_0 = 1$
- Portanto, $T(n) = O(n^2)$

Análise da ordenação por trocas sucessivas

- Pior caso: quando a seqüência de entrada com n números inteiros é fornecida em ordem decrescente ($t_i = i$ para todo i)

$$
\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} t_i \\
&= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} i \\
&= n + 5\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 \\
&= n + 5\,\frac{n(n-1)}{2} + n - 1 \\
&= \frac{5}{2}\,n^2 - \frac{5}{2}\,n + 2n - 1 = \frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1
\end{aligned}
$$

Análise da ordenação por trocas sucessivas

- Pior caso: $T(n) = (5/2)n^2 - (1/2)n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$\frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 \leqslant cn^2\,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 5/2$ temos:

$$\begin{aligned} \frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 &\leqslant \frac{5}{2}\,n^2 \\ -\frac{1}{2}\,n - 1 &\leqslant 0 \end{aligned}$$

ou seja,

$$\frac{1}{2}\,n + 1 \geqslant 0$$

Análise da ordenação por trocas sucessivas

- A inequação $(1/2)n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$
- Assim, escolhendo $c = 5/2$ e $n_0 = 1$, temos que

$$\frac{5}{2}n^2 - \frac{1}{2}n - 1 \leqslant cn^2$$

  para todo $n \geqslant n_0$, onde $c = 5/2$ e $n_0 = 1$
- Portanto, $T(n) = O(n^2)$

Exemplo

- Somar duas matrizes A e B, ambas $n \times n$.
- Como?

void somaMatriz(int matA[MAX][MAX], int matB[MAX][MAX], int matC[MAX][MAX])

```
{
 for (int i = 0; i < n; i++)
     for (int j = 0; j < n; j++)
         matC[i][j] = matA[i][j] + matB[i][j];
}
```

Complexidade? $O(n^2)$

Exemplo

- Multiplicar duas matrizes A e B, ambas $n \times n$.
- Como?

void multMatriz(int matA[MAX][MAX], int matB[MAX][MAX], int matC[MAX][MAX])

```
{
 for (int i = 0; i < n; i++)
 {
      for (j = 0; j < n; j++)
      {
           matC[i][j] = 0;
           for (k = 0; k < n; k++)
                matC[i][j] = matC[i][j] + matA[i][k] * matB[k][j];
      }
 }
}
```

Complexidade? $O(n^3)$

Moral da história

- Algoritmos diferentes para resolver um mesmo problema em geral diferem dramaticamente em eficiência
- Essas diferenças podem ser muito mais significativas que a diferença de tempos de execução em um supercomputador e em um computador pessoal

Moral da história

- Algoritmos diferentes para resolver um mesmo problema em geral diferem dramaticamente em eficiência
- Essas diferenças podem ser muito mais significativas que a diferença de tempos de execução em um supercomputador e em um computador pessoal

Moral da história

Problema da ordenação

- Algoritmo A
  - Tempo de execução de pior caso $O(n^2)$
  - Supercomputador: 100 milhões de operações por segundo
  - Programador: hacker, codificação com tempo de execução $2n^2$
- Algoritmo B
  - Tempo de execução de pior caso $O(n \log_2 n)$
  - Computador pessoal: 1 milhão de operações por segundo
  - Programador: mediano, codificação com tempo de execução $50n \log_2 n$
- Ordenar 1 milhão de números ($n = 10^6$)

Moral da história

Problema da ordenação

- Algoritmo A
  - Tempo de execução de pior caso $O(n^2)$
  - Supercomputador: 100 milhões de operações por segundo
  - Programador: hacker, codificação com tempo de execução $2n^2$
- Algoritmo B
  - Tempo de execução de pior caso $O(n \log_2 n)$
  - Computador pessoal: 1 milhão de operações por segundo
  - Programador: mediano, codificação com tempo de execução $50n \log_2 n$
- Ordenar 1 milhão de números ($n = 10^6$)

Moral da história

Problema da ordenação

- Algoritmo A
  - Tempo de execução de pior caso $O(n^2)$
  - Supercomputador: 100 milhões de operações por segundo
  - Programador: hacker, codificação com tempo de execução $2n^2$
- Algoritmo B
  - Tempo de execução de pior caso $O(n \log_2 n)$
  - Computador pessoal: 1 milhão de operações por segundo
  - Programador: mediano, codificação com tempo de execução $50n \log_2 n$
- Ordenar 1 milhão de números ($n = 10^6$)

Moral da história

Problema da ordenação

- Algoritmo A

$$\frac{2 \cdot (10^6)^2 \text{ operações}}{10^8 \text{ operações/segundo}} = 20.000 \text{ segundos} \approx 5{,}56 \text{ horas}$$

- Algoritmo B

$$\frac{50 \cdot 10^6 \log_2 10^6 \text{ operações}}{10^6 \text{ operações/segundo}} \approx 1.000 \text{ segundos} \approx 16{,}67 \text{ minutos}$$