Cursos de Verão 2009 - IME/USP
Tópicos de Programação
Aula 5 - invariantes e notação O
Prof. Alexandre da Silva Freire

1 Correção de algoritmos iterativos

Considere o seguinte problema: dado um vetor $v[0 \ \cdots \ n-1]$ de números inteiros, encontrar um inteiro min tal que $0 \leq min \leq n-1$ e $v[min] \leq v[j]$, para $j = 0, 1, \ldots, n-1$. Em palavras, queremos o índice de um elemento mínimo em um vetor. Eis uma possível solução:

```
     int minimo(int v[], int n) {
(1)     int i, min;
(2)     i = 1; min = 0;
(3)     while (i < n) {
(4)        if(v[i] < v[min]) {
(5)           min = i;
           }
(7)        i++;
        }
(9)     return min;
     }
```

O algoritmo acima está correto? Ou seja, para qualquer entrada válida ele devolve a resposta correta? Para responder a esta pergunta, primeiro precisamos definir o que é uma *entrada válida* para este algoritmo. Em uma entrada válida o valor de n deve ser maior ou igual a 1 (um vetor vazio não possui elemento mínimo) e o vetor v deve conter n posições, cada uma delas contendo um inteiro. Agora que nossa entrada está bem definida, considere o seguinte invariante: (i) *imediatamente antes da execução da linha (3), vale que* $0 \leq min \leq i-1$ *e* $v[min] \leq v[j]$, *para* $j = 0, 1, \ldots, i-1$. Mostraremos que o invariante (i) é válido durante toda a execução do algoritmo:

Inicialização: imediatamente antes da primeira execução da linha (3), vale que $min = 0$ e $i = 1$. Como $i = 1$, o intervalo de 0 a $i-1$ no vetor v contém apenas o elemento $v[0 = min]$, que, portanto, é o elemento mínimo. Ou seja, imediatamente antes da primeira execução da linha (3), temos que $0 \leq min \leq i-1$ e $v[min] \leq v[j]$, para $j = 0, 1, \ldots, i-1$. Portanto, o invariante (i) é válido na inicialização do laço da linha (3).

Manutenção: suponha que o invariante (i) seja válido imediatamente antes de uma iteração na qual $1 \leq i < n$. Mostraremos que (i) continua válido na próxima iteração, na qual devido à linha (7) temos $i' = i+1$ no papel de i. Por hipótese, temos que $0 \leq min \leq i-1$ e $v[min] \leq v[j]$,

para $j = 0, 1, \ldots, i-1$. Se $v[i] < v[min]$, significa que $v[i]$ é o elemento mínimo de v no intervalo $0, 1, \ldots, i$. Logo, a atribuição feita na linha (5) atualiza corretamente a variável min. Isto porque por hipóstese $v[min]$ é um elemento mínimo de v no intervalo $0, 1, \ldots, i-1$. Ou seja, comparando apenas $v[i]$ com $v[min]$ é possível determinar um elemento mínimo de v no intervalo $0, 1, \ldots, i$. Pelo mesmo motivo, se $v[i] \geq v[min]$, temos que $v[min]$ é um elemento mínimo de v no intervalo $0, 1, \ldots, i$. Neste caso, o teste da linha (4) falha e a linha (5) não é executada. Portanto, no final da iteração corrente, vale que $0 \leq min \leq i' - 1$ e $v[min] \leq v[j]$, para $j = 0, 1, \ldots, i' - 1$. Após a linha (7) ser executada, a variável i é incrementada e o invariante (i) é válido imediatamente antes da próxima iteração do laço da linha (3).

Término: no início da execução vale que $i = 1$. Sabemos que $n \geq 1$ e que a cada iteração do laço da linha (3) é acrescentada (na linha (7)) uma unidade ao valor da variável i. Isto implica que o laço da linha (3) pára quando $i = n$. Como o invariante (i) é mantido ao longo de todas as iterações do laço da linha (3), temos que após a última iteração, quando $i = n$, vale que $0 \leq min \leq n-1$ e $v[min] \leq v[j]$, para $j = 0, 1, \ldots, n-1$. Portanto, o índice devolvido na linha (9) corresponde ao índice de um elemento mínimo de v no intervalo $0, 1, \ldots, n-1$ e, consequentemente, o algoritmo está correto.

Outra pergunta que estamos interessados em responder: qual o consumo de tempo do algoritmo `minimo`? Definimos as constantes $C_1, C_2, \ldots, C_9$ para representar quantas instruções são executadas nas linhas $(1), (2), \ldots, (9)$, respectivamente. As linhas (1),(2) e (9) são executadas uma única vez, independentemente da entrada. Já as demais linhas são executadas de forma dependente da entrada. A linha (3) é executada n vezes. As linhas (4) e (7) são executas $n-1$ vezes. A linha (5) é executada no máximo $n-1$ vezes. No total o algoritmo `minimo` executa no máximo $C_1 + C_2 + C_9 + C_3 n + (C_4 + C_7 + C_5)(n-1)$ operações.

2 Notação O, Ω e Θ

Para uma função $g(n)$, denotamos por

- $O(g(n)) = \{f(n)$: existem constantes positivas c e n_0 tais que $0 \leq f(n) \leq c \cdot g(n)$ para todo $n \geq n_0\}$.
- $\Omega(g(n)) = \{f(n)$: existem constantes positivas c e n_0 tais que $0 \leq c \cdot g(n) \leq f(n)$ para todo $n \geq n_0\}$.
- $\Theta(g(n)) = \{f(n)$: existem constantes positivas c_1, c_2 e n_0 tais que $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ para todo $n \geq n_0\}$.

Suponha que $f(n)$ e $g(n)$ sejam funções assintoticamente positivas[1]. Vejamos como comparar assintoticamente estas duas funções:

- **Transitividade**:

  $f(n) = O(g(n))$ e $g(n) = O(b(n))$ implicam $f(n) = O(b(n))$;
  $f(n) = \Omega(g(n))$ e $g(n) = \Omega(b(n))$ implicam $f(n) = \Omega(b(n))$;
  $f(n) = \Theta(g(n))$ e $g(n) = \Theta(b(n))$ implicam $f(n) = \Theta(b(n))$.

- **Reflexividade**:

  $f(n) = O(f(n))$;
  $f(n) = \Omega(f(n))$;
  $f(n) = \Theta(f(n))$.

- **Simetria**:

  $f(n) = \Theta(g(n))$ se e somente se $g(n) = \Theta(f(n))$.

- **Simetria de transposição**:

  $f(n) = O(g(n))$ se e somente se $g(n) = \Omega(f(n))$.

Utilizando esta notação, vejamos qual o consumo de tempo do algoritmo `minimo`. Temos que $C_1 + C_2 + C_9 = \Theta(1)$ e $C_3 n + (C_4 + C_7)(n-1) = \Theta(n)$. A quantidade de operações executadas na linha 5 é no máximo $C_5(n-1) = O(n)$. Logo, o consumo total de tempo é $\Theta(1) + O(n) + \Theta(n) = \Theta(n)$.

[1] uma função $f(n)$ é assintoticamente positiva se existe um n_0 tal que para todo $n \geq n_0$ temos que $f(n)$ é positivo.