INE5451 - Tarefa 1

Frank Paulo Filho

Tarefa

Fazer os exercícios 2.1-4, 2.2-2, 2.3-6 e Problema 2.2 (letras b e c) do capítulo 2 do livro do Cormen.

Exercício 2.1-4

Enunciado

Consider the problem of adding two n-bit binary integers, stored in two n-element arrays arrays A and B. The sum of the two integers should be stored in binary form in an (n+1)-element array C. State the problem formally and write pseudocode for adding the two integers.

Resposta

Descrição formal

Input Duas sequências de tamanho N A = (a_1, a_2, ..., a_n) e B: (b_1, b_2, ..., b_n) representando dois números inteiros em binário, do bit menos significante para o mais significante.

Output Uma sequência de tamanho N+1 C = (c_1, c_2, ..., c_n) representando a soma binária de A e B.

Pseudocódigo

```
SOMA-BINÁRIA(A, B)
  N = A.length
  Alocar array C de tamanho N + 1
  carry = 0
  for i = 1 to A.length
    C[i] = A[i] ^ B[i] ^ carry
    carry = (A[i] & B[i]) | (carry & (A[i] ^ B[i]))
  C[N+1] = carry
```

Nota: Estão sendo usadas as convenções do livro de indexar arrays começando em 1.

Exercício 2.2-2

Enunciado

Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in `A[1]`. Then find the second smallest element of A, and exchange it with `A[2]`. Continue in this manner for the first n - 1 elements of A. Write pseudocode for this algorithm, which is known as **selection sort**. What loop invariant does this algorithm maintain? Why does it need to run for only the first n-1 elements, rather than for all n elements? Give the best-case and worst-case running times of selection sort in -notation.

Resposta

Pseudocódigo

```
FIND-SMALLEST(A, startAt)
  // Assume que o último elemento é o menor
  smallest = A.length

  // Procura até o penúltimo elemento para descobrir se algum elemento é menor que smallest
  for i = startAt to A.length-1
    if A[i] < A[smallest]
      smallest = i

  return smallest

SELECTION-SORT(A)
  for i = 1 to A.length-1
    smallest = FIND-SMALLEST(A, i)
    tmp = A[i]
    A[i] = A[smallest]
    A[smallest] = tmp
```

Loop invariante

No começo de cada iteração do loop `for` externo, a subarray A[1..i-1] consiste dos elementos originalmente em A[1..i-1], porém ordenados.

Por que só precisa rodar para n-1 elementos?

Na última iteração do `SELECTION-SORT`, existem dois elementos ainda não ordenados: `A[n-1]` e `A[n]`. Se `A[n]` for menor que `A[n-1]`, os dois serão trocados e

`A` estará, portanto, ordenada. Se `A[n]` não for menor, `A` já está ordenada.

Tempos de execução

No **melhor caso**, a array já está ordenada. O `if` de `FIND-SMALLEST` sempre é falso, portanto seu corpo não é executado. O tempo de execução é (n^2).

No **pior caso**, a array está ordenada ao contrário (ordem decrescente). O `if` de `FIND-SMALLEST` é sempre verdadeiro. O tempo de execução é (n^2).

Exercício 2.3-6

Enunciado

Observe that the while loop of lines 5-7 of the `INSERTION-SORT` procedure in Section 2.1 uses a linear search to scan (backward) through the sorted subarray A[i..j-1]. Can we use a binary search (see Exercise 2.3-5) instead to improve the overall worst-case running time of insertion sort to $(n \lg n)$?

Resposta

Uma busca binária pode ser utilizada pois a subarray `A[i..j-1]` está ordenada. O número de comparações para encontrar o elemento predecessor será, no pior caso, $O(\lg n)$, enquanto com busca linear seria $O(n)$. Porém, ainda é necessário mover cada elemento depois do predecessor um “slot” pra frente, portanto cada iteração do loop externo continua fazendo trabalho $O(n)$ e, assim, **o tempo de execução do pior caso ainda é O(n)**.

Problema 2.2 - Corretude do bubblesort

Enunciado

Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeadtly swapping adjacent elements that are out of order.

```
BUBBLESORT(A)
i = 1
  for i = 1 to A.length - 1
    for j = A.length downto i + 1
      if A[j] < A[j-1]
        exchange A[j] with A[j-1]
```

a. Let A' denote the output of `BUBBLESORT(A)`. To prove that `BUBBLESORT` is correct, we need to prove that it terminates and that

`A'[1] <= A'[2] <= ... <= A'[n]` (2.3)

where n = A.length. In order to show that `BUBBLESORT` actually sorts, what else do we need to prove?

The next two parts will prove inequality (2.3).

b. State precisely a loop invariant for the `for` loop in lines 2-4, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in this chapter.

c. Using the termination condition of the loop invariant proved in part (b), state a loop invariant for the `for` loop in lines 1-4 that will allow you to prove inequality (2.3). Your proof should use the structure of the loop invariant proof presented in this chapter.

Resposta

Letra b

No começo de cada loop `for` interno, a subarray `A[j..n]` consiste dos elementos originalmente em `A[j..n]`, porém possivelmente em ordem diferente. O primeiro elemento (`A[j]`) é o menor da subarray `A[j..n]`.

Inicialização na primeira iteração, `A[j..n]` contém apenas um elemento e, portanto, este elemento é o menor da subarray.

Manutenção em cada iteração, `A[j]` é trocado com `A[j-1]` se for menor. Assim, `A[j-1]` será o menor dentre os dois e o primeiro elemento será o menor da subarray.

Término a condição de término do loop é j >= i + 1. Logo, temos j = i quando o loop termina. Assim, `A[i]` é o menor elemento da subarray `A[i..n]`

Letra c

No começo de cada loop `for` externo, a subarray `A[1..i-1]` consiste de elementos ordenados, todos menores do que os de `A[i..n]`.

Inicialização na primeira iteração, a subarray `A[1..i-1]` está vazia e portanto ordenada.

Manutenção em cada iteração, a partir da letra b) temos que `A[i]` é o menor elemento da subarray `A[i..n]`. Também sabemos que a subarray `A[1..i-1]` está ordenada. Portanto, a subarray `A[i..n]` estará ordenada também.

Término quando o loop termina, i = n+1. Logo, a array `A[1..n]` estará ordenada.