Sabrina Araújo da Silva nUSP 12566182

MAC0338 - ANÁLISE DE ALGORITMOS

LISTA 9

1. **(CLRS 17.1-2)** Mostre que se uma operação **Decrementa** for incluída nas operações de manipulação de um contador binário com k bits, n operações podem custar tempo $\Theta(nk)$.

uma operação Decrementa tem a seguinte estrutura

```
Decrementa (A, k)
1   i ← 0
2   enquanto i < k e A[i] = 0 faça
3       A[i] ← 1
4       i ← i + 1
5   se i < n
7       então A[i] ← 0
```

Considere que as operações de manipulação de um contador binário com k bits sejam como abaixo:

```
para i ← 1 até n faça
    Incrementa (A, k)
    Decrementa (A, k)
```

No pior caso, a chamada de Incrementa faz k alterações. Quando isso ocorre, a chamada de Decrementa reverte a operação, também produzindo k alterações.

Por exemplo:

- estado inicial do vetor em um determinado i

0	1	1	1

- após Incrementa

1	0	0	0

- após Decrementa (volta ao estado inicial)

0	1	1	1

É possível observar que se Incrementa produzir K alterações, Decrementa produzirá o mesmo e reverterá o estado do vetor para o estado anterior ao Incrementa, gerando um loop para as próximas operações. No pior caso, cada execução de Incrementa e Decrementa produz K alterações e, portanto, as n operações têm custo $\Theta(nk)$

4. **(CLRS 17.2-3)** Suponha que desejamos não apenas incrementar um contador mas também algumas vezes reinicializá-lo com zero. Mostre como implementar um contador com um vetor binário de maneira que qualquer sequência de n operações `incrementa1` e `zera_contador` consuma tempo $O(n)$, desde que o contador esteja inicialmente com zero. (**Dica:** Mantenha um apontador para o 1 mais significativo do contador.)

Considere as seguintes implementações de incrementa1 e zera_contador

```
incrementa1 (A, K)
1  i ← 0
2  bit ← -1 (bit mais significativo)
3  enquanto i < K e A[i] = 1 faça
4      A[i] ← 0
5      se i == bit
6          então bit ← -1
7      i ← i + 1
8  se i < K
9      então A[i] ← 1
10     se i > bit
11         então bit ← i
```

```
zera_contador (A, bit)
1  se bit ≥ 0
2      então para i ← bit decrescendo até 0 faça
3          se A[i] == 1
4              então A[i] ← 0
```

Para calcular o custo das operações podemos usar a análise por créditos

- atribuímos 3 créditos por incrementa
  - o primeiro é usado para pagar a alteração para 1 na linha 9
  - o segundo é armazenado sobre o bit alterado na linha 9
  - o terceiro é armazenado quando o bit mais significativo muda para

maior na linha 11.

As alterações na linha 4 são pagas por créditos armazenados na linha 9.

As alterações na função zera-contador são pagas por créditos que foram armazenados na linha 11. Para resetar o contador, somente é necessário que o bit mais significativo e os anteriores sejam alterados. Assim, armazenar o número de vezes que o bit mais significativo aumentou de posição corresponde ao número de bits que precisam ser alterados em zera-contador.

Portanto, o custo amortizado por operação é $3 \in O(1)$ e o custo total da sequência de operações é $\leq 3n \in O(n)$.