

Análise Amortizada dos Algoritmos

Análise amortizada

Introdução

- Uma ferramenta de análise de complexidade (desempenho) dos algoritmos
- Permite estudar o custo necessário para se efetuar uma sequência de operações
 - o custo é uma grandeza não negativa, em geral inteira, com o qual se pretende medir os gastos dum determinado recurso como o tempo ou o espaço de armazenamento dos dados
- Considera-se uma sequência de operações e divide-se o custo total da sequência pelo número de operações efetuadas, obtendo-se um custo *médio por operação*
 - algumas operações podem ser mais dispendiosas do que outras, pelo que podem ser atribuídos custos diferentes a cada uma delas
- O caso mais interessante é quando é aplicada à análise do comportamento de uma **estrutura de dados**
 - a sequência de operações é sobre uma estrutura de dados

Introdução

- O objetivo é calcular o custo médio, por operação, de uma sequência de **n** operações sobre uma estrutura de dados
- A ideia é considerar a sequência de **n** operações mais desfavorável, o que dá origem a uma análise no pior caso
 - diferente da análise normal do pior caso, a qual analisa o custo mais elevado de uma operação (a mais dominante ou a mais dispendiosa)

Definições

- O **custo amortizado por operação** para uma sequência de **n** operações é o custo total (soma dos custos) de todas operações dividido por **n**
 - seja uma sequência de **n** operações: $op_1, op_2, \dots, op_n$
e os respectivos custos: $c_1, c_2, \dots, c_n$
 - **custo máximo de uma operação**:
$$C_{\max} = \max \{ c_i : i = 1, 2, \dots, n \}$$
 - **custo amortizado por operação** (ou custo médio por operação):
$$E(c) = (c_1 + c_2 + \dots + c_n) / n$$
- A **complexidade amortizada** de uma sequência de **n** operações, é a complexidade da sequência no pior caso (isto é, o custo total máximo possível entre todas as possíveis sequências de **n** operações) dividido por **n**
 - custo de execução da sequência no pior caso: $T(n)$
 - custo amortizado por operação no pior caso: $T(n) / n$

Métodos para realizar a Análise Amortizada

- **Método agregado** (custo total)
 - estuda o custo total das operações e considera a média
- **Método contabilístico** (ponto de vista do banqueiro)
 - impõe um custo amortizado extra às operações de menor custo, economizando para futuras operações de maior custo
- **Método do potencial** (ponto de vista do físico)
 - define uma função potencial (não negativa) sobre o estado da estrutura de dados e usa esta função para limitar o custo amortizado
- Para qualquer um dos métodos, existem limites amortizados

$$\sum_{\text{operação}} \text{"custo amortizado"} \geq \sum_{\text{operação}} \text{"custo real"}$$

Método Agregado

Princípios

- Trata-se de uma aproximação direta
- Para determinar o custo médio de uma operação
 - 1) calcular o custo total de uma sequência de operações no pior caso
 - 2) dividir o valor anterior pelo número de operações
- Ou seja,
 - se para qualquer **n**, uma sequência de **n** operações, no **pior caso**, tem um custo de **$T(n)$** , então o **custo amortizado** por operação é: **$T(n) / n$**
- Pode-se considerar que todas as operações têm o **mesmo custo amortizado**

Exemplo (contador binário)

- Seja A um array 1D de k bits

$$A[j] \in \{0, 1\} \text{ e } j = 0, \dots, k-1$$

- Então

$$x = A[k-1] 2^{k-1} + A[k-2] 2^{k-2} + \dots + A[1] 2^1 + A[0] 2^0$$

- Suponha-se que o contador
 - começa com $x = 0$, e
 - é incrementado de 1 unidade **n** vezes

Exemplo (contador binário)

- Função em C

```
void incrementa (int *A, int k)  
{  
    int j;  
    j = 0;  
    while (j < k && A[j] == 1) {  
        A[j] = 0;  
        j = j + 1;  
    }  
    if (j < k)  
        A[j] = 1;  
}
```


Exemplo (contador binário)

	k-1	k-2			...	3	2	1	0	decimal
Incremento 00:	0	0	0	0	...	0	0	0	0	0
Incremento 01:	0	0	0	0	...	0	0	0	1	1
Incremento 02:	0	0	0	0	...	0	0	1	0	2
Incremento 03:	0	0	0	0	...	0	0	1	1	3
Incremento 04:	0	0	0	0	...	0	1	0	0	4
Incremento 05:	0	0	0	0	...	0	1	0	1	5
Incremento 06:	0	0	0	0	...	0	1	1	0	6
Incremento 07:	0	0	0	0	...	0	1	1	1	7
Incremento 08:	0	0	0	0	...	1	0	0	0	8
Incremento 09:	0	0	0	0	...	1	0	0	1	9
Incremento 10:	0	0	0	0	...	1	0	1	0	10
Incremento 11:	0	0	0	0	...	1	0	1	1	11
Incremento 12:	0	0	0	0	...	1	1	0	0	12
...		...			...			...		...

Exemplo (contador binário)

- Numa sequência de n execuções de incrementa(A, k)
 - $A[0]$ muda a cada 1 incremento ($1 = 2^0$)
 - $A[1]$ muda a cada 2 incrementos ($2 = 2^1$)
 - $A[2]$ muda a cada 4 incrementos ($4 = 2^2$)
 - ...
 - $A[i]$ muda a cada 2^i incrementos
 - ...
 - $A[k-1]$ muda a cada 2^{k-1} incrementos
- Total de vezes que os bits são alterados em n incrementos
 - $A[0]$ muda n vezes (1 alteração a cada incremento)
 - $A[1]$ muda $n/2$ (1 alteração a cada 2 incrementos)
 - $A[2]$ muda $n/4$ (1 alteração a cada 4 incrementos)
 - ...
 - $A[k-1]$ muda $n/2^{k-1}$ (1 alteração a cada 2^{k-1} incrementos)

Exemplo (contador binário)

- Total de vezes que os bits são alterados em **n** incrementos (operações)

$$n + n/2 + n/4 + \dots + n/2^{k-1} =$$

$$n \times (1 + 1/2 + 1/4 + \dots + 1/2^{k-1}) =$$

$$1 + 1/2 + 1/4 + \dots + 1/2^{k-1} \rightarrow \text{progressão geométrica de } q = 1/2 \text{ e } a_1 = 1$$

$$S = (a_1 \times (1 - q^{k-1})) / (1 - q) =$$

$$(1 \times (1 - (1/2)^{k-1}) / (1 - 1/2) = (1 - 1/2^{k-1}) / (1/2) =$$

$$((2^{k-1} - 1) / 2^{k-1}) / (1/2) = (2 \times (2^{k-1} - 1)) / (2^{k-1}) =$$

$$(2^k / 2^{k-1}) - (2 / 2^{k-1}) = 2 - 2 / 2^{k-1} =$$

$$2 - 1/2^{k-1} < 2$$

$$n \times (1 + 1/2 + 1/4 + \dots + 1/2^{k-1}) = n \times S < 2 \times n$$

- Portanto,

- custo total da sequência de **n** operações (incrementos): $T(n) < 2 \times n \in O(n)$ (linear)

- **custo amortizado por operação: $T(n) / n < (2 \times n) / n = 2 \in O(1)$ (constante)**