20/08/18

Módulo 01:

Vinícius
20 pontos de prova
05 pontos de lista
} 25 pontos

Livro do Cormen:
Cap. 01 a 05 e 17 sobre Análise amortizada

outros livros indicados: (complementar)
- Sedgewick
- Kleinberg, Tardos
- Aho, Hopcroft, Ulman

Avaliando os algoritmos:
- corretude
- eficiência

Laços e recursão ajudam na avaliação!

* ordenação de algoritmos
- Algoritmo de Inserção (Insertion sort)

1ª 12, 8, 5, 19, 13, 4 ...
2ª 8, 12, 5, 19, 13, 4 ...
3ª 5, 8, 12, 19, 13, 4 ...
4ª 5, 8, 12, 19, 13, 4 ... já estava correto
5ª 5, 8, 12, 13, 19, 4 ...
6ª 4, 5, 8, 12, 13, 19 ...

```
InsertionSort (v[], n)        // usando laços
    para i = 2 até n
        j = i
        enquanto j > 1 e v[j-1] > v[j]
            temp = v[j]
```

continua...

v[j] = v[j-1]
v[j-1] = temp
j = j-1

conferindo se o algoritmo está correto:

* Invariante: para o loop externo (para i = 2 até n...)

- Inicialização;
- Manutenção (manter a propriedade verdadeira);
- Terminação.

na inicialização:
v[1 até posição 1] está ordenado v[1..1] ordenado!

na manutenção:
v[1 ... i] está ordenado após "i" iterações

na terminação:
v[1 ... n] está ordenado

* Invariante para o loop interno
na inicialização:
v[1 ... j-1] está ordenado

na manutenção:
v[1 ... j-2] está ordenado
v[j ..... i] está ordenado
v[j-1] < v[j]
j ≥ 1 no final da interação

v[1... j-1, j+1 ... i] (pela manutenção externa)

v[1... j-1, j+1...i] está ordenado v[j] < v[j+1] j ≥ 2

na terminação:

j = 1 ✓

v[j-1] < v[j]

* Recursão:

```
procedimento (n)
    ... // condição de parada
    ...
    procedimento (n-1)
    ...
```

- InsertionSort Recursivo

```
InsertionSort (v[], n)
    Se n = 1 retorne
    InsertionSort (v[], n-1)
        j = n
        enquanto j > 1 e v[j-1] > v[j]
            temp = v[j]
            v[j] = v[j-1]
            v[j-1] = temp
            j = j-1
```

Indução: Se o "loop" interno está correto, o externo também estará por indução, já que ele usa recursão.

22/08/18

* Eficiência dos algoritmos

```
InsertionSort (v[ ], n)
...
para i = 2 até n          // tempo c1
    j = i                 // c2
    enquanto j > 1 e v[j-1] > v[j]      // c3, c4 ← i vezes
        temp = v[j]
        v[j] = v[j-1]
        v[j-1] = temp
        j = j-1
                          // 4c2, i-1 vezes
```

$$\sum_{i=2}^{n} 2c3 - i + 4c2\,(i-1) + c2 + c1$$

$$\sum_{i=2}^{n} c1 + c2 - 4c2 + \sum (4c2 + 2c3)\,i$$

$$(c1 - 3c2)(n-1) + (4c2 + 2c3) \cdot \left(\frac{(n+2)(n-1)}{2}\right)$$

notações:

Θ	teta	limite firme
O	"ó"	limite superior
Ω	ômega	limite inferior
o	"ozinho"	
ω	"omegazinho"	

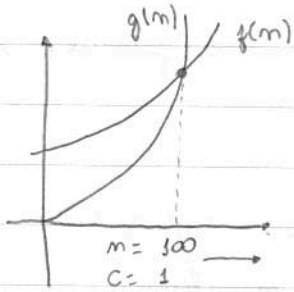
Notação O "ó":

$f(n) = O(g(n))$ apenas notação e não igual. Pertence a uma mesma classe!

$f(n) \in O(g(n))$ pensar como pertence!

Dizemos $f(n) = O(g(n))$ se $\exists c$ e $\exists n_0$ tal que:

$\forall n \geq n_0 \quad f(n) \leq c \cdot g(n)$



limite superior

Exemplo:

$f(n) = 2n + 5 \qquad g(n) = c \cdot n \qquad$ para $c = 3$

$f(n) = O(g(n))$

$2n + 5 \leq 3n \qquad$ no caso de $n = 5$ já vale, pois:

$2 \cdot 5 + 5 \leq 3 \cdot 5$

$15 \leq 15$

Logo, para $n = 5, 6, 7 \ldots$ já satisfaz!

Outros exemplos:

	$f(n)$	⋯⋯→	$g(n)$	
1)	$2n + 5$		1	$\exists c, n_0, \forall n \geq n_0 \quad 2n + 5 \leq c \cdot 1$ não é $O(1)$!
2)	$2n + 5$	⋯⋯→	$c \cdot 2^n$	$2n + 5 \leq c \cdot 2^n$ para $c = 1$, $n_0 = 4$ funciona! É $O(c \cdot 2^n)$!
	Provar para outros números!			
3)	$n\sqrt{n}$	⋯⋯→	$n \log n$	$n\sqrt{n} \leq c\, n \log n$ log na base 2
	não é $O(n \log n)$!			$2^{\sqrt{n}} \leq 2^{c \log n} =$ em PAA sempre!
				$2^{\sqrt{n}} \leq n^{\log c} = n^c$
				se $n = m^2 \quad 2^m \leq m^{2c}$

Para qualquer constante em $g(n)$ é $O(1)$!

* Ordem de crescimento das funções:

$1, \log n, \sqrt{n}, n, n \cdot \log n, n^2, n^{2,37}, 2^n, n!$ (fatorial), n^n

notação Ω ômega:

Dizemos que $f(n) = \Omega(g(n))$ se $\exists c, n_0$ talque

$\forall n \geq n_0 \quad f(n) \geq c \cdot g(n)$

$f(n) = 2n + 5$

$g(n) = n$

para $c = 1$ e $n = 0$ já funciona!

$2n + 5 \geq c \cdot n$

$2 \cdot 0 + 5 \geq 1 \cdot 0 \quad 5 \geq 0$

Se $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$?

Prova:

$\exists c, n_0$ tq $n \geq n_0$

$f(n) \leq c \cdot g(n)$

Queremos mostrar que:

$\exists c', n_0'$ tq $n \geq n_0'$

$g(n) \geq c' \cdot f(n)$

$c \cdot g(n) \geq f(n)$

$g(n) \geq \frac{1}{c} \cdot f(n)$

para $c' = \frac{1}{c}$ $n_0' = n^0$

já funciona!

limite inferior

notação Θ "teta":

Dizemos que $f(n) = \Theta(g(n))$ se $\exists c_1, c_2, n_0$ tq

$\forall n \geq n_0 \quad c_1 g(n) \leq f(n) \leq c_2 g(n)$

$f(n) = 2n + 5$ para $c_1 = 1$ e $c_2 = 3$

$g(n) = n$

$1 \cdot g(n) \leq f(n) \leq 3 g(n)$ $\quad n_0 = 10$ funciona!

Limite firme!

Outro exemplo:

$\frac{(n-1)(n+1)}{2} \longrightarrow n^2$

$a_2 \cdot n^2 + a_1 \cdot n + a_0 = \Theta(n^2)$ assumindo que $a^2 > 0$

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$c_1 \cdot n^2 \leq a_2 n^2 + \underline{a_1 n + a_0}$ ignoraremos!

$c_1 = a_2 - 1$ teremos certeza de que a constante esquerda será menor que a constante direita!

$(a_2 - 1) n^2 \leq a_2 n^2 + a_1 n + a_0$

$\cancel{a_2} n^2 - n^2 \leq \cancel{a_2} n^2 + a_1 n + a_0$

$0 \leq n^2 + a_1 n + a_0$ funcionou!

$O(\ ) \leq$

$\Omega(\ ) \geq$

$\Theta(\ ) =$

$o(\ ) <$

$\omega(\ ) >$

A seguir, as notações o e ω:

27/08/18

notação o "ozinho" e ω:

Dizemos que $f(n) = o(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ Essa é a notação o

Dizemos que $f(n) = \omega(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ Essa é a notação ω

Propriedades:

1. $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$
2. $f(n) = O(g(n)),\ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
3. $f(n) = o(g(n)),\ g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$
4. $f(n) = o(g(n)) \quad g(n) = \omega(f(n))$

5. $f(n) = O(\max(g(n), h(n))$ limitado pela maior das duas funções g e h!

se e somente se:

$f(n) = O(g(n) + h(n))$

* Recorrências para algoritmos recursivos

```
InsertionSort (v[], n)
    Se n = 1   - constante
        return
    InsertionSort (v[], n-1)  - T(n-1)
    j = n  - constante
    enquanto j > 1 e v[j-1] > v[j]      } ≤ dn
        temp = v[j]
        v[j] = v[j-1]
        v[j-1] = temp
        j--
```

$T(n)$ = número de passos que InsertionSort executa para um vetor com n elementos.

$$T(n) = \begin{cases} c & , n = 1 \quad \text{base} \\ T(n-1) + dn & , n \neq 1 \quad \text{caso geral} \end{cases}$$

$T(n) \neq O(n^3)$

Há 3 formas de resolver:

- Substituição
- Árvore
- Teorema Mestre

* Substituição

Vamos supor que seja $O(n^3)$:

$T(n) \leq c' n^3$

Base: $T(1) \leq c' \cdot 1^3$ verdadeiro se $c' \geq c$.

Passo indutivo: hipótese de indução

$T(n-1) \leq c'(n-1)^3$

$T(n) = T(n-1) + dn$

$T(n) \leq c'(n-1)^3 + dn = c' \cdot (n^3 - 3n^2 + 3n - 1) + dn$

$= c'n^3 - \underbrace{3c'n^2 + 3c'n - 1c' + dn}_{<0} \leq c'n^3$

$-3c'n^2 + (3c' + d)n - c' < 0$

$c' = d$ substituir

$-3dn^2 + 4dn - 1 < 0$ (função negativa)

para n suficientemente grande ($n \geq n_0$) é sempre verdade!

Vamos melhorar a suposição:

$T(n-1) = T(n-2) + d(n-1)$

$T(n) = T(n-1) + dn$

$T(n) = T(n-2) + d(n-1) + dn$ → ...

$T(n) = T(n-3) + d(n-2) + d(n-1) + dn$ → ...

⋮

$T(n) = T(1) + d(2) + d(3) + \cdots + dn$

$T(n) = c + d\,\frac{(n+2)(n-1)}{2}$

Base:

$T(1) = c + \frac{d\cdot(1+2)(1-1)}{2} \quad T(1) = c$

$\frac{}{2}\ 0$

Passo: hipótese indutiva

H.I. $T(n-1) = c + \frac{d(n+1)(n-2)}{2}$

$T(n) = T(n-1) + dn$

$T(n) \overset{HI}{=} c + \frac{d(n+1)(n-2)}{2} + dn$

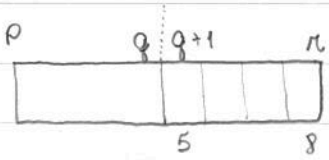
$= c + \frac{d(n+1)(n-2) + 2dn}{2}$

$= c + \frac{d(n^2 - n - 2 + 2n)}{2}$

$= c + \frac{d(n^2+n-2)}{2} = c + \frac{d(n+2)(n-1)}{2}$ Provado!

* Dividir para conquistar

- Dividir
- conquistar
- combinar



$r - p + 1$

$8 - 5 + 1 = 4$ elementos

$q - p + 1$

Mergesort:

```
merge ( v[], p, q, r )        // apenas mescla os elementos - θ(n)
  para i=1 até q-p+1
    L[i] = v[ p+i-1 ]         // vetor left
  para i=1 até r-q
    R[i] = v[ q+1 ]           // vetor right
  L[ q-p+2 ] = ∞              // sentinela
  R[ r-q+1 ] = ∞              // sentinela
  i = 1
  j = 1
  para K=p até r              // copiar de volta para o vetor
    se L[i] ≤ R[j]                ordenado
      v[K] = L[i]
      i++
    senão
      v[K] = R[j]
      j++
```

note que:

se $x \in R$

$\lfloor x \rfloor$ = maior inteiro menor ou igual a n : piso ou chão

$\lceil x \rceil$ = menor inteiro maior ou igual a n = teto

Mergesort (v[], p, r)

se p = r

return

$q = \left\lfloor \frac{p+r}{2} \right\rfloor$

mergesort (v[], p, q)

mergesort (v[], q+1, r)

merge (v[], p, q, r)

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2) + \theta(n), & \text{caso contrário} \end{cases}$$

$n = 2^K$, $K \in N$

$T(n) \leq 2T\left(\frac{n}{2}\right) + dn$

$T(n) \leq 2\left[2T\left(\frac{n}{4}\right) + d\frac{n}{2}\right] + dn$

$\leq 4T\left(\frac{n}{4}\right) + 2dn$

$\leq 4\left[2T\left(\frac{n}{8}\right) + d\frac{n}{4}\right] + 2dn$

$\frac{n}{2^i} = 1$

$n = 2^i$

$\log n = i$

$\leq 8T\left(\frac{n}{8}\right) + 3dn$

$T(n) \leq nT(1) + n \log n$

$T(n) \leq n + dn \log n$

29/08/18

Base: $n=1 \quad T(1) \leq 1 + d \cdot 1 \log 1$ ok!

Passo: $T(n) \leq 2T\left(\frac{n}{2}\right) + dn$

$$T(n) \overset{HI}{\leq} 2\left[\frac{n}{2} + d\frac{n}{2}\log\left(\frac{n}{2}\right)\right] + dn$$

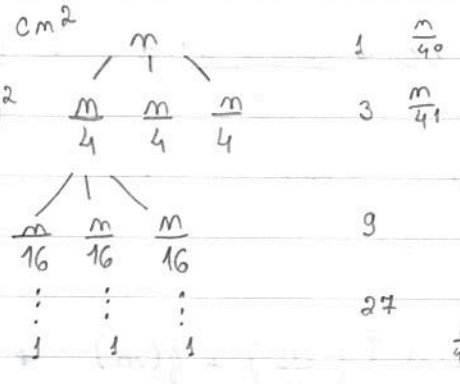
$$= (d+1)n + dn \underbrace{\log\left(\frac{n}{2}\right)}_{\log n - \log 2^1}$$

$$= (d+1)n + dn \log n - dn$$

$$= n + d\, n \log n \quad \text{Provado!}$$

* Árvore

$T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$



$T(1) = 1$

$\frac{n}{4^i} = 1 = \log_4 n$

no i-ésimo nível: 3^i nós da árvore

último nível: $3^{\log_4 n}$

Em um nó no i-ésimo nível gastamos o tempo $c\left(\frac{n}{4^i}\right)^2$

$$T(n) = cn^2 + 3c\left(\frac{n}{4}\right)^2 + 3^2 c\left(\frac{n}{4^2}\right)^2 + 3^3 c\left(\frac{n}{4^3}\right)^2 + \dots + 3^{\log_4 n - 1} c\left(\frac{n}{4^{\log_4 n - 1}}\right)^2 + \underbrace{3^{\log_4 n}}$$

$$T(n) = cn^2 \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i + \left(3^{\log_3 n}\right) \cdot \log_4 3$$

$\log_3 n = \frac{\log_4 n}{\log_4 3} =$

$\log_4 3 = \log_3 n \cdot \log_4 3$

$$\frac{cn^2\left(1-\left(\frac{3}{16}\right)^{\log_4 n}\right)}{1-\frac{3}{16}} + n^{\log_4 3} \leq c'n^2$$

$\log_4 3 < 1$, pois $\log_4 4 = 1$

Substituição:

Base: $n=1$

$T(1) = 1 \leq c'n^2$ ok!

Passo:

$$T(n) \underset{\text{Hi}}{\leq} 3 \cdot c'\left(\frac{n}{4}\right)^2 + c \cdot n^2$$

$$\left(\frac{3c'}{16} + c\right) n^2 \leq c'n^2$$

$$\frac{3c'}{16} + c \leq c'$$

$$c' - \frac{3}{16}c' \geq c \rightarrow \frac{16c' - 3c'}{16} \geq c \rightarrow \frac{13c'}{16} \geq c$$

$$c' \geq \frac{16c}{13}$$ ok!

no Mergesort:

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ → $T(n) = 2T\left(\frac{n}{2}\right) + cn$

$a = 2$

$b = 2$

$f(n) = cn$

* Teorema Mestre

Sejam $a, b \geq 1$

$T(n) = \underbrace{aT\left(\frac{n}{b}\right)}_{} + \underbrace{f(n)}_{}$ comparar os dois termos

1º caso: se $f(n)$ for menor

se $f(n) = O(n^{\log_b a - \epsilon})$, para algum $\epsilon > 0$

então $T(n) = \Theta(n^{\log_b a})$

2º caso: Se forem iguais

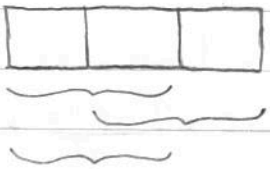
Se $f(n) = \Theta(n^{\log_b a})$, então $T(n) = \Theta(n^{\log_b a} \cdot \log n)$

3º caso: se $f(n)$ for maior

Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para algum $\epsilon > 0$

e se $a f\left(\frac{n}{b}\right) \leq c f(n)$ para algum $c < 1$, para todo n suficientemente grande, então $T(n) = \Theta(f(n))$

Exemplo: Ordenar um vetor dividido em 3 partes



3 ordenações

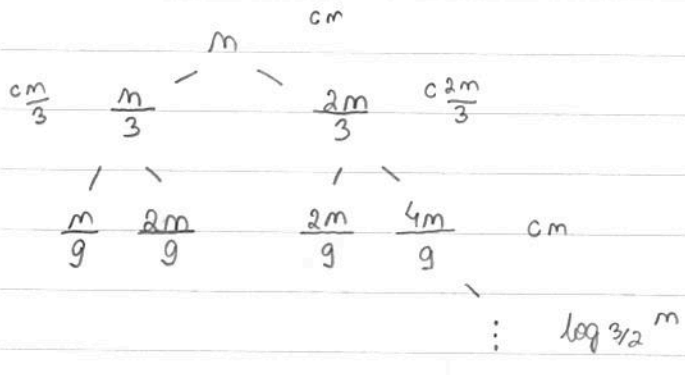
$a = 3 \quad b = 3/2 \quad f(n) = c$

$\Theta(n^{\log_{3/2} 3})$

$$T(n) = \begin{cases} 3T\left(\frac{2}{3}n\right) + c, & n \geq 3 \\ d, & n \leq 2 \end{cases}$$

Outro exemplo:

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$



$\log_{3/2} n = \frac{\log_2 n}{\log_2 3/2}$

$\Theta(\log n)$

$T(n) \leq c' n \log n + c'n$

$T(n) \leq c' n \log_{3/2} n + 1$

vamos provar! →

$T(m) \leq c' m \log m$

Caso funciona!

Passo:

$$T(m) \overset{HI}{\leq} c'\left(\frac{m}{3}\right)\log\left(\frac{m}{3}\right) + c'\left(\frac{m}{3}\right) + c'\left(\frac{2m}{3}\right)\log\left(\frac{2m}{3}\right) + c'\left(\frac{2m}{3}\right) + cm$$

$$T(m) \leq c'm + cm + c'\left(\frac{m}{3}\right)\left[\log m + \log\frac{1}{3}\right] + c'\left(\frac{2m}{3}\right)\left[\log m + \log 2/3\right]$$

$$T(m) \leq c'm + cm + c'm\log m + c'm\left(\frac{\log 1/3}{3} + \frac{2\log 2/3}{3}\right)$$

$$\cancel{cm} + c'\cancel{m}\underbrace{\left(\frac{\log 1/3}{3} + \frac{2\log 2/3}{3}\right)}_{d>0} \leq 0$$

Para $c' \geq \underbrace{\frac{c}{-\log}}_{>0} \leq 0$ e, portanto,

$$T(m) \leq c'm + c'm\log m$$

$$c + c'd \leq 0$$

$$c \leq c'd$$

$$c' \geq \frac{c}{d}$$

03/09/18

* Análise Probabilística

- Algoritmos probabilísticos:

q [ ] probabilidade de aparecer $\frac{1}{n!}$

contrata (1) — contratação de melhores professores

melhor = 1

para i = 2 até n — Algoritmo Las Vegas

se q[i] > q[melhor] // se qualidade do próximo for melhor

contrata (i) // dispensa anterior e contrata o

melhor = i // próximo

$$O(dn + cn) \rightarrow O(dn + c.\log n)$$

1º	100%	1
2º	50%	1/2
3º	33%	1/3
...		1/n

} Probabilidade da contratação ser a melhor

$$X_i = \begin{cases} 1, & \text{se } i\text{-ésimo for contratado} \\ 0, & \text{caso contrário} \end{cases}$$

$$X = \sum_{i=1}^{n} X_i$$

Esperança:

$$E[X] = \sum x \cdot P(X = x) \rightarrow E[X_i] = 1 \cdot \frac{1}{i} + 0 \cdot \left(1 - \frac{1}{i}\right) = \frac{1}{i}$$

linearidade: $E\left[\sum a_i X_i\right] = \sum a_i E[X_i]$

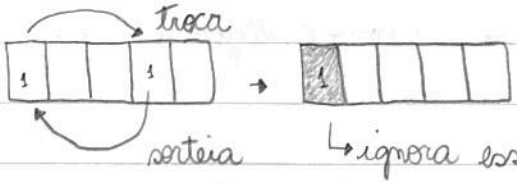
$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{i} = \log n + O(1)$$

Afinando o algoritmo:

- Embaralhar para garantir que as permutações fiquem uniformes.

```
embaralha (q[ ])        // garante que seja probabilístico
contrata (1)
melhor = 1
para i = 2 até n
    se q[i] > q[melhor]
        contrata (i)
        melhor = i
```

- Implementando o método embaralha ( ):




```
para i = 1 até n-1
    j = sorteia (i, n)
    troca (i, j)
```

Outro exemplo:

* Verifica se um dado número é primo ou composto:

```
P ∈ [2, n/2]
sorteia ( )
se n/p não é inteiro
    retorna primo
senão
    retorna composto
```

n/saída	% primo	% composto	% correto
primo	1	0	1
composto	$\leq \frac{n-2}{2}$	$\geq \frac{2}{n}$	$\geq \frac{2}{n}$

$\lim_{x \to \infty} \left(1 + \frac{1}{x}\right)^x = e$

Probabilidade de insucesso:

$\leq \left(1 - \frac{2}{n}\right)^k \leq e^{-\frac{2k}{n}} \leq e^{-1} = \frac{1}{e} < 1$

$e^x \geq 1 + x$ → propriedades de cálculo

$x = -\frac{2}{n}$

primalidade (n)

rode teste(n) $\frac{n}{2}$ vezes

se retornar composto alguma vez

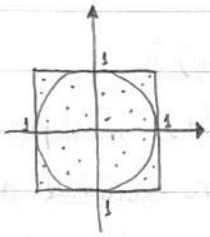
retorna composto

senão

retorna primo

Algoritmo Monte Carlo

Esses exemplos são chamados de algoritmos Monte Carlo e Las Vegas.



$A\square = 4$

$AO = \pi r^2 = \pi$ já que $r = 1$

$$\frac{\text{pontos} \in O}{\text{pontos} \in \square} \approx \frac{AO}{A\square} = \frac{\pi}{4}$$

Para sabermos se um ponto está dentro ou fora do círculo, medimos a distância do centro ao ponto.

se $\angle 1$ está no círculo senão está fora do círculo

05/09/18

* Análise Amortizada (cap. 17 cormen)

- olhar o contexto em que as operações acontecem;
- operações baratas compensam operações mais caras.

3 Paradigmas:

- Análise agregada (olhar o algoritmo como um todo)
- Método do contador / banqueiro
- Método da energia potencial

* Análise agregada

Exemplo 1:

operação de inserir numa pilha - push $O(1)$

operação de retirar elementos - pop $O(1)$ multipop com $K = 1$

multipop (K) - $O(\min(n, K))$ n é uma variável do problema e θ número de operações

$$\sum_{j=1}^{i} c(\theta_j) = \sum_{\substack{j \in [1,j] \\ \theta_j = m_j}} c(m_j) + \sum_{\substack{j \in [1,j] \\ \theta_j = p_j}} c(p_j) \leq M + 2P \leq 2(M+P) = 2n$$

θ = operações

multipops + pushes

M parcelas P parcelas

$$\sum_{\substack{j \in [1,j] \\ \theta_j = m_j}} c(m_j) \leq M + \sum_{\substack{j \in [1,j] \\ \theta_j = p_j}}$$

"P"

Exemplo 2: Somando bits + 1

	0 0 0 0 +1	i-ésimo bit muda a cada 2^i incrementos
	0 0 0 1 +1	Em n operações i-ésimo bit muda $\lfloor \frac{n}{2^i} \rfloor$
	0 0 1 0 +1	
$n=3 \rightarrow$	0 0 1 1 +1	4 bits $O(K \cdot n)$
	0 1 0 0 +1	
	b_3 b_2 b_1 b_0 ...	

Total de mudanças de bits: $\sum_{i=0}^{K-1} \left\lfloor \frac{m}{2^i} \right\rfloor < \sum_{i=0}^{K-1} \frac{m}{2^i} = m \sum_{i=0}^{K-1} \frac{1}{2^i} < m \sum_{i=0}^{\infty} \frac{1}{2^i} = 2m$

* Método do banqueiro

Custo amortizado como planos de saúde, por exemplo.

$o_1, \ldots, o_m$ (operações)

$c(o_i)$ custo real

$\hat{c}(o_i)$ custo amortizado

$$\sum_{j=1}^{i} \hat{c}(o_i) \geq \sum_{j=1}^{i} c(o_i)$$

$c(o_i)$	1	1	1	5	gasto
$\hat{c}(o_i)$	2	2	2	2	ganho
	sobra 1	sobra 1	sobra 1		

usaremos as sobras + 2 para a operação mais cara!

não importa muito o custo por operação, mas sim cumprir o tempo que foi prometido!

Push / multipop

c custo real: 1 $\min(n, K)$

$\hat{c}$ amortizado: 2 0 (se a pilha estiver vazia)

O saldo guarda o número de elementos na pilha

Assim como em planos de saúde, você cobra mais, para, quando for usar, o saldo ser suficiente e o plano não falir.

* Método da energia potencial (método do físico)

- operações baratas adicionam energia ao sistema;
- Quando houver uma operação cara, usamos a energia acumulada.

$\Phi(E_i)$

$E_0 \xrightarrow{\theta 1} E_1 \xrightarrow{\theta 2} E_2 \xrightarrow{\theta 3} \dots \xrightarrow{\theta m} E_m$ $\quad \Phi$ quantidade de energia

$\hat{c}(\theta i) \geq c(\theta i) + \Phi(E_i) - \Phi(E_{i-1})$

Armazena energia para gastar em operações caras!

$\Phi(E_i)$ = número de elementos na pilha

$\Phi(E_i) = 0$

$\Phi(E_i) = \Phi(E_{i-1}) + 1$ $\qquad E_i - E_{i-1} = -\min(n, K)$

$\hat{c}(\text{Push}_i) = 1 + 1 = 2$ $\qquad \hat{c}(\text{multipop}_i) = \min(n, K) - \min(n, K) = 0$

$$\hat{c}(o1) \geq c(\theta 1) + \cancel{\Phi(E1)} - \Phi(E_0)$$
$$+ \quad \hat{c}(\theta 2) \geq c(\theta 2) + \cancel{\Phi(E2)} - \cancel{\Phi(E1)}$$
$$\vdots \qquad\qquad \vdots$$
$$\hat{c}(\theta m) \geq c(\theta m) + \Phi(Em) - \cancel{\Phi(Em-1)}$$

$$\sum \hat{c}(\theta m) \geq \sum c(\theta m) + \Phi(Em) - \Phi(E_0)$$

* Redimensionamento de vetores

n - número de elementos do vetor

N - tamanho real da memória

Nós dobramos o vetor a cada vez que ele se completa.

	Inserção	Inserção + redimensionamento	Leitura
C	1	N+1	1
$\hat{C}$	3	3	1
		⋮	
		3m	