Prog. Dinâmica

Wednesday, June 24, 2015 10:30 AM

- Das tres tecnicas estudadas, é a "mais prática".
- Nome ~~"retrado"~~. errado.
  - ↳ estratégia de venda do inventor
- Nome melhor: tabelação sistemática.

Fibonacci:

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & n \geq 2 \\ 1 & n = 1 \\ 1 & n = 0 \end{cases}$$

$F[i] = 1,\ i \in [0,1],\ = \text{undef},\ i \in [2, \dots$

fib(n) =

if F[n] return F[n]

~~if n ≤ 1 return 1~~

F[n] = ~~return~~ fib(n-1) + fib(n-2).

return F[n]

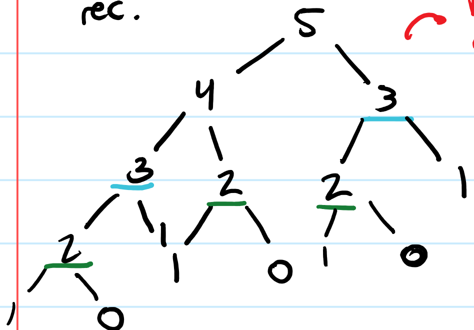
↳ complexidade exponencial

$T(n) = T(n-1) + T(n-2) + O(1).$

$\in \Theta(2^n).$

↳ Akra-Bazzi ñ resolve

Árvore de rec.



Muitos cálculos repetidos

n	0	1	2	3	4	5	...
	1	1	2	3	5	8	..

Memorização (memorization)*

(cache)

Fib2(n) :=
  F[0] = F[1] = 1;
  $\forall i \in [2 \ldots n]$
    f[i] = f[i-1] + f[i-2]
  return F[n].

↳ O(n)

Redução de $O(2^n)$ para $O(n)$!!!

Fib3(n) :=
  f = g = 1.
  while (n ≥ 2)
    g = f
    f = f + g
    g = f
    n++

Ideia: poupar espaço na tabela.

Ex: Subset sum

$C = \{a_1, \ldots, a_n\}, a_i \in \mathbb{Z}$

multi-set (com rep)

Questão:

$\exists\, S \subseteq C$ tq $\sum_i a_i = 0$?

O problema é NP-Completo.

OBS: a ordem de incluir ou excluir ñ importa.

ex: se

$a_1 + a_2 = -4 \Rightarrow$ Subproblema:

encontra em $a_3 \dots a_n$

um subc. que soma +4.

OBS2: cada subproblema é definido por

i) primeiro índice do elemento

ii) Soma a ser alcançada.

$P(i, \Sigma)$. ← subproblema

- $P(1,0) = ?$ V/F ?
  - $P(2,-a_1)$
    - $P(3,-(a_1+a_2))$
    - $P(3,-a_1)$
  - $P(2,0)$
    - $P(3,-a_2)$
    - $P(3,0)$

⋮

$$P(1,0) = P(2,-a_1) \lor P(2,0)$$

$$P(2,-a_1) = P(3,-(a_1+a_2)) \lor P(3,-a_1).$$

$$P(i,\Sigma) = \begin{cases} P(i+1, \Sigma - a_i) \lor P(i+1, \Sigma), & 1 \le i \le n,\ m \le \Sigma \le M \\ V & i > n,\ \Sigma = 0 \end{cases}$$

[illegible]

$$
\begin{cases} V & i > n,\ \Sigma = 0 \\ f & i > n,\ \Sigma \neq 0 \\ f & \Sigma < m \vee \Sigma > M. \end{cases}
$$

$m = \sum_{a<0} a$

$M = \sum_{a>0} a$

Mesmo template do Fibonacci.

P	m	m+1	m+2	...	M-1	M
1						
2						
3						
⋮						
n						

Tamanho: $(M-m)\cdot n$

Sem recursão: preenche a tabela na direção ↑.

```
SubsetSum (c) :=
    aloca uma tabela P
    for i ∈ [m, M] {
        if i = 0 then P[n+1, i] = V
               else P[n+1, v] = f
    }for i in [n, n-1, ..., 1] {
        for j in [m, M]
            P[i, j] = P[i+1, j-c_i] ∨ P[i+1, j].
```

Espaço: $O(n \cdot D)$

Tempo: $O(n \cdot D)$ (para a sol: dá ou ñ dá)

$D = M - m + 1$.

Tempo p/ recuperar a solução: $O(n)$.

OBS: não é necessário toda a tabela; bastam 2 linhas. (p/ o problema de decisão).

Linear? Exponencial?

OBS: Tamanho da entrada

$$\underbrace{\sum^{n} \log_2 |a_i| + 1}_{= n'} \quad \text{bits.}$$

ainda:

$n' = \log_2 a_1$: $m = 0$, $M = a_1$; $D = a_1 + 1$
$= 2^{n'} + 1$.

maior número p/ armazenar a soma

OBS: $O(nD)$ é exponencial no tamanho da entrada

→ Alg. pseudo-polinomial

"Um algoritmo é pseudo-polinomial caso ele é polinomial para entradas com uma representação unitária."

↳ Se aplica p/ algoritmos
numéricos

Partition

Dado C, $\exists S \subseteq C$ tal que

$$\sum_{c_i \in S} c_i = \sum_{c_i \in \bar{S}} c_i \text{ ¿}$$