João Vitor Fernandes Dias

21h45
27/01/25

#Revisão: Recursão e Indução

② Rosen 4.3.3:

ⓐ $f(n+1) = f(n) + 2$

n	0	1	2	3	4	5
f(n)	-1	2	4	6	8	10

$f(5) = f(4) + 2$ → $8 + 2 = 10 = f(5)$

$f(4) = f(3) + 2$ → $6 + 2 = 8 = f(4)$

$= f(2) + 2$ → $4 + 2 = 6 = f(3)$

$= f(1) + 2$ → $2 + 2 = 4 = f(2)$

$f(1) = 2$

ⓑ $f(n+1) = 3f(n)$

n	0	1	2	3	4	5
f(n)	-1	2	$2 \cdot 3^1$	$2 \cdot 3^2$	$2 \cdot 3^3$	$2 \cdot 3^4$

$f(5) = 3f(4) = 3 \cdot (3^3 \cdot 2) = 3^4 \cdot 2$

$f(4) = 3f(3) = 3 \cdot (3^2 \cdot 2) = 3^3 \cdot 2$

$f(3) = 3f(2) = 3 \cdot (3 \cdot 2) = 3^2 \cdot 2$

$f(2) = 3f(1) = 3 \cdot 2$

$f(1) = 2$

ⓒ $f(n+1) = 2 \wedge f(n)$

n	0	1	2	3	4	5
f(n)	-1	2	$2 \wedge 2$	$2 \wedge (2 \wedge 2)$	$2 \wedge (2 \wedge (2 \wedge 2))$	$2 \wedge (2 \wedge (2 \wedge (2 \wedge 2)))$

$f(5) = 2 \wedge f(4) = 2 \wedge (2 \wedge (2 \wedge (2 \wedge 2)))$

$f(4) = 2 \wedge f(3) = 2 \wedge (2 \wedge (2 \wedge 2))$

$f(3) = 2 \wedge f(2) = 2 \wedge (2 \wedge 2)$

$f(2) = 2 \wedge f(1) = 2 \wedge 2$

$f(1) = 2$

22h14
17/01/25

d) $f(n+1) = f(n)^2 + f(n) + 1$

n	0	1	2	3	4	5
f(n)	-1	2	7	57	3257	10679546

$f(5) = f(4)^2 + f(4) + 1 = 3257^2 + 3257 + 1 = 10679546$
$f(4) = f(3)^2 + f(3) + 1 = 57^2 + 57 + 1 = 3257$
$f(3) = f(2)^2 + f(2) + 1 = 7^2 + 7 + 1 = 57$
$f(2) = f(1)^2 + f(1) + 1 = 2^2 + 2 + 1 = 7$
$f(1) = 2$

// 1

3) Rosen 4.3.3

a) $f(n+1) = f(n) + 3f(n-1)$

n	0	1	2	3	4	5
f(n)	-1	2	-1	5	2	17

$f(5) = f(4) + 3f(3) = 2 + 3\cdot(5) = 17$
$f(4) = f(3) + 3f(2) = 5 + 3\cdot(-1) = 2$
$f(3) = f(2) + 3f(1) = -1 + 3\cdot(2) = 5$
$f(2) = f(1) + 3f(0) = 2 + 3\cdot(-1) = -1$
$f(1) = 2$
$f(0) = -1$

d) $f(n+1) = \frac{f(n-1)}{f(n)}$

n	0	1	2	3	4	5
f(n)	-1	2	1/2	-4	1/8	-32

$f(5) = f(3) / f(4) = (-4)/(1/8) = -32$
$f(4) = f(2) / f(3) = (1/2)/(-4) = 1/8$
$f(3) = f(1) / f(2) = 2/(-1/2) = -4$
$f(2) = f(0) / f(1) = -1/2$
$f(1) = 2$
$f(0) = -1$

// 2

~~9h46~~ 10hos
18/01/25

④ Rosen 4.3.7

ⓐ $a_n = 6n$ →

a_1	a_2	a_3	a_4	a_5
6	12	18	24	30

$\begin{cases} A(0) = 0 \\ A(x+1) = 6 + A(x) \end{cases}$

ⓑ $a_n = 2n+1$

a_1	a_2	a_3	a_4	a_5
3	5	7	9	11

$\begin{cases} B(0) = 1 \\ B(x+1) = 2 + B(x) \end{cases}$

ⓒ $a_n = 10^n$

a_1	a_2	a_3	a_4	a_5
10^1	10^2	10^3	10^4	10^5

$\begin{cases} C(0) = 1 \\ C(x+1) = 10C(x) \end{cases}$

ⓓ $a_n = 5$

a_1	a_2	a_3	a_4	a_5
5	5	5	5	5

$\begin{cases} D(0) = 5 \\ D(x+1) = D(x) \end{cases}$

//

③

⑨ Rosen 4.3.35

- $\lambda^R = \lambda$; $|\lambda| = 0$
- $\omega = x + y$; $|\omega| = n+1$; $|x| = n$; $|y| = 1$
- $\omega^R = y + x^R$

//

④

Divisão e Conquista

Cardoso, Cap. 5, Exercício 1, Pág. 246

19h 15
19/01/25

Divisão e Conquista

Tardos, Cap 5, Ex. 1, Pág. 246

```
def get_mean_by_index_query(size):
    left_a = left_b = 0
    right_a = right_b = size-1
    while (right_a > left_b) or (right_b > left_b):
        mean_value_a = (left_a + right_a)//2
        elem_a = A(mean_value_a)
        mean_value_b = (left_b + right_b)//2
        elem_b = B(mean_value_b)
        min_offset = min(mean_value_a - left_a, \
                         mean_value_b - left_b)
        min_offset = 1 if min_offset == 0 else min_offset
        if elem_a < elem_b:
            left_a += min_offset
            right_b -= min_offset
        else:
            left_b += min_offset
            right_a -= min_offset
    return min(A(left_a), B(left_b))
```

5

Cormen, Cap 9, Ex. 9.3-8, Pág. 223

A resolução desta questão é basicamente a anterior, apenas trocando as chamadas ~~à função~~ às funções A() e B(), por acesso aos vetores A[] e B[], que poderiam ser passados como parâmetro da função.

6

11h33
13/01/2025

Módulo 3 - Paradigma: Programação Dinâmica

Tardos, Cap 6, Ex 7, Pág 318-319

- Conceito do algoritmo: percorre de 1 a n mantendo o índice i e o valor P(i) do menor P(i) encontrado até o i verificado. Guarda também a diferença entre o P(i) atual e o menor P(i) encontrado até então se for maior que a última maior diferença encontrada. Sempre que atualizar o menor P(i) anterior, atualiza o i de retorno. Quando atualizar a diferença, atualiza o i e j de retorno.

```
def get_best_pair_report_DP(P):
    n = len(P)
    i = j = min_i = max_diff = 0
    for k in range(1,n):
        if P[k] < P[min_i]:
            min_i = k
        diff = P[k] - P[min_i]
        if diff > max_diff:
            max_diff = diff
            i = min_i
            j = k
    if i == j:
        return f'There was no way to make money during these {n} days'
    return f'To buy on {i+1}, sell on {j+1}'
```

//

Cap 11 : ~ ⓪

21/01/25

Tardos, Cap. 6, Ex. 13, Pág. 324

O que precisamos encontrar é um ciclo tal que a sequência das razões $r_{i,j}$ seja maior que 1. Então teremos um ciclo de oportunidade se: $r_{a_1,a_2} \cdot r_{a_2,a_3} \cdot \ldots \cdot r_{a_k,a_1} > 1$.

Um dos algoritmos usados para se encontrar ciclos negativos é o algoritmo de Bellman-Ford-Moore, mas, para isso, seria necessário considerarmos um somatório de pesos que, ao final, deveria ser menor que 0. Desse modo, para que o ~~prob~~ problema em questão se enquadre nos ~~param~~ parâmetros do algoritmo, podemos fazer a operação de logaritmo aos dois lados da equação, de modo que o ciclo de oportunidade será encontrado se: $\log(r_{a_1,a_2}) + \log(r_{a_2,a_3}) + \cdots + \log(a_k, a_1) > 0$.

O algoritmo de Bellman-Ford-Moore assegura a existência de ciclo negativo caso, após $n-1$ iterações, ainda haja atualizações nos pesos. Para isso, definiremos que todos os pesos em questão serão $-\log(r_{i,j})$. Assim, caso o algoritmo encontre um ciclo negativo, podemos afirmar que há um ciclo de oportunidade, visto que seu somatório será menor que 0.

21h08

(8)

21h10
22/01/25

Clea

22h10 # Módulo 4 - NP e Intratabilidade Computacional

Tardos, Cap. 8, Ex. 1, Pág. 505

1. Sim. O problema de Interval Scheduling é um problema de cobertura, onde precisamos encontrar um conjunto de intervalos não sobrepostos de tamanho pelo menos K. O problema de vertex cover também é um problema de cobertura, onde precisamos encontrar um conjunto de vértices que cubra todas as arestas do grafo.

22h33 Eu manipulamos o Interval Scheduling de tal forma que cada uma das tarefas seja representada por um vértice, e façamos com que todas as ~~arestas~~ tarefas em horários sobrepostos tenham arestas entre si, podemos transformar o problema de interval scheduling em um problema de Independent Set. E esse problema, por sua vez, é redutível para Vertex Cover, assim tornando o problema de Interval Scheduling redutível para Vertex Cover.

23h39 2. 2 Desconhecido. Isso se dá pois o problema de Independent Set é um problema NP-Completo, já o problema de Interval Scheduling é um problema polinomial. Por isso, não podemos afirmar que Independent Set é redutível para Interval Scheduling.

23h41 //

Tardos, Cap. 8, Ex. 3, Pág 505-506

3) O problema Efficient Recruiting (ER) é um problema de cobertura, onde precisamos encontrar um conjunto de Conselheiros(C) que cubra todas as áreas de Esporte (E). ~~Def~~ De forma similar, no problema de Set Cover precisamos encontrar um Conjunto de Conjuntos (C) que cubram todos os Elementos (E).

22/01/25

23h45

Sendo assim, podemos reduzir o problema de ER para o problema de Set Cover Cover, tornando o n n ∈ R NP-Completo.

//

(10)