Certifique-se de que você tentou fazer por um tempo antes de ver as próximas páginas!

5. Sejam n e m dois inteiros positivos. Seja $\mathcal{C} = \{C_1, \ldots, C_m\}$ uma coleção com m conjuntos de números entre 1 e n, isto é, $C_i \subseteq \{1, 2, \ldots, n\}$ para cada $1 \leq i \leq m$, e tal que $\bigcup_{C_i \in \mathcal{C}} C_i = \{1, 2, \ldots, n\}$.

Descreva um algoritmo guloso para o problema da cobertura de conjuntos. Prove que seu algoritmo não é ótimo.

A ideia do algoritmo é escolher, a cada passo, o conjunto C_i que cubra o maior nº de elementos não cobertos ainda.

```
COBERTURA_GULOSA (n, m, 𝒞)
1   seja S[1..m] um vetor iniciado com zeros
                          // S vai indicar os conj. escolhidos
2   seja U[1..n] um vetor iniciado com zeros
                          // U vai indicar os elementos cobertos
3   seja c = 0    // vai contar o nº de elementos cobertos
4   enquanto c ≠ n
5       escolha Ci ∈ 𝒞 com maior nº de elem. descobertos
6       S[i] = 1
7       para cada u ∈ Ci
8           se U[u] = 0
9               c = c+1
10              U[u] = 1
11  devolve S
```

O algoritmo certamente devolve uma solução viável para o problema porque garante a invariante "Antes da t-ésima iteração, vale que

$$\bigcup_{i, S[i]=1} C_i = \{j : U[j] = 1\}$$

e que $|\{j : U[j] = 1\}| = c$"

Como ao final o laço termina com $c = n$, a invariante mostra que os conjuntos escolhidos cobrem $\{1, \ldots, n\}$.

Esse algoritmo não é ótimo: considere $n = 9$ e $\mathcal{C} = \{C_1 = \{1,2,3,4\}, C_2 = \{5,6,7\}, C_3 = \{8,9\}, C_4 = \{1,2,7,8\}, C_5 = \{4,5\}, C_6 = \{2,3\}, C_7 = \{5,6\}\}$. O algoritmo pode escolher, em ordem, os C_4, C_5, C_6, C_7 e C_3 enquanto a solução ótima é C_1, C_2, C_3

Vamos considerar que cada C_i é um vetor de tamanho $n+1$ cuja posição 0 armazena $|C_i|$ e a j-ésima posição, com $1 \leq j \leq n$, indica se $j \in C_i$.

O tempo do algoritmo é:

$$\Theta(m) + \Theta(n) + \Theta(1) + O(m) + \bigstar + O(m) + O(mn)$$

onde cada termo é o tempo de execução de cada linha entre 1 e 7: linhas 1, 2 e 3 são diretas; como a linha 6 executa no máximo uma vez para cada $C_i \in \mathcal{C}$, as linhas 4, 5, 6 e 7 executam $O(m)$ vezes; uma execução do laço da linha 7 leva tempo $\Theta(n)$ por causa de nossa representação de C_i, e por isso ao todo leva tempo $O(mn)$. Resta descobrir $\bigstar$, que é o tempo da linha 5 ao todo.

Já sabemos que são $O(m)$ execuções da linha 5. Uma execução precisa percorrer todos os m conjuntos C_i verificando quantos elementos são descobertos e guardando o maior valor possível. Para fazer isso, para todo $1 \leq j \leq n$, verificamos se $j \in C_i$ e se $U[j] = 0$ ou 1. Portanto, uma execução da linha 5 leva tempo $O(mn)$. Assim, $\bigstar = O(m^2 n)$.

Com isso, o tempo de execução do algoritmo é

$$\Theta(m) + \Theta(n) + \Theta(1) + O(m) + O(m^2 n) + O(m) + O(mn) = O(m^2 n)$$

ou seja, polinomial no tamanho da entrada.

6. Nesta questão, considere o problema de fazer troco para n centavos usando o menor número total de moedas. Você pode assumir que existe um número infinito de moedas disponíveis para cada valor.

   Forneça um algoritmo guloso ótimo para fazer troco tendo disponíveis moedas de 1, 5, 10, 25 e 50 centavos. *Dica: para provar que seu algoritmo é ótimo, assuma que ele não é. Qual é a diferença de uma solução ótima para a solução do seu algoritmo?*

   O seu algoritmo continua ótimo se o conjunto de moedas disponíveis for 1, 7, 10 e 16? Justifique.

Considerando os valores das moedas em ordem decrescente, o algoritmo sempre escolhe a maior quantidade possível de moedas daquele valor.

Seja $M = (50, 25, 10, 5, 1)$ um vetor que armazena os valores das moedas. A solução do algoritmo será um vetor $P[1..5]$ tal que $P[i]$ é a quantidade de moedas de valor $M[i]$ que ele pegou.

```
TROCO (n, M)        // n é o valor do troco a ser feito
    seja P[1..5] um vetor
    para i = 1 até 5
        P[i] = ⌊n / M[i]⌋
        n = n % M[i]
    devolva P
```

Note que o algoritmo tem tempo $\Theta(1)$ (constante!).

Vamos mostrar que a solução do algoritmo é ótima.
Suponha que ela não seja ótima e seja P^* uma solução ótima.
Então $\sum_{i=1}^{5} P^*[i] < \sum_{i=1}^{5} P[i]$ (a ótima usa menos moedas que o alg.)
Em particular, deve haver alguma posição i, $1 \le i \le 5$, com $P[i] \neq P^*[i]$.

Note que $n = \sum_{i=1}^{5} P[i] \cdot M[i] = \sum_{i=1}^{5} P^*[i] \cdot M[i]$. (ambos fazem o troco p/ n)

Seja j, $1 \le j \le 5$, o menor índice tal que $P[j] \neq P^*[j]$ (já que há algum, pegamos o menor).
Então vale que $\sum_{i=1}^{j-1} P[i] \cdot M[i] = \sum_{i=1}^{j-1} P^*[i] \cdot M[i] = z$ (de 1 a $j-1$, são iguais e fazem o mesmo troco)

Portanto, $P[j..5]$ é uma solução para fazer o troco de $n-z$.
Além disso, $P^*[j..5]$ é uma solução ótima para fazer o troco de $n-z$.

Como $P[j] \neq P^*[j]$, deve valer que $P[j] > P^*[j]$, pois o algoritmo sempre pega a maior quantidade de moedas

Assim, $P[j] = P^*[j] + x$, com $x > 1$.
Mas então existem $x \cdot M[j]$ centavos descritos com moedas de menor valor em P^*, e isso vale especificamente porque temos moedas de valores 50, 25, 10, 5, 1. → Para os curiosos, vejam os Lemas 1 e 2 ao fim.
Então há um conjunto com mais do que x moedas em $P^*[j+1..5]$ cuja soma de valores é $x \cdot M[j]$ e que poderiam ser removidas dali e substituídas por x moedas em $P^*[j]$, diminuindo a quantidade de moedas em P^*, o que é um absurdo pois P^* já é ótima!

Então nossa suposição inicial é falsa e, portanto, P é solução ótima.
CQD

Não é ótimo para as moedas 1, 7, 10 e 16. Tome $n = 14$, por exemplo. O algoritmo devolveria 1 moeda de 10 e 4 de 1, totalizando 5 moedas. Claramente a solução ótima são 2 moedas.

7. Um *grafo caminho* G é um grafo que consiste apenas de um caminho, isto é, $V(G) = \{v_1, v_2, \ldots, v_n\}$ e $E(G) = \{v_i v_{i+1} : 1 \leq i < n\}$ e seja $w : V(G) \to \mathbb{R}^+$ uma função de peso sobre **os vértices** de G. Assim, para qualquer conjunto $X \subseteq V(G)$ de vértices, o peso de X, $w(X)$, é naturalmente definido como $w(X) = \sum_{v \in X} w(v)$.

   Um conjunto S de vértices de um grafo qualquer é *independente* se para quaisquer dois vértices $u, v \in S$, não existe aresta uv no grafo.

   O problema do *Conjunto Independente de Peso Máximo em Caminhos* consiste em receber um grafo caminho G e uma função w de peso nos vértices e o objetivo é encontrar um subconjunto S de vértices tal que S é um conjunto independente e $\sum_{v \in S} w(v)$ é máximo. Responda:

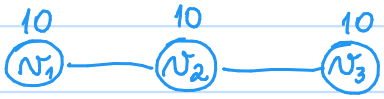
   (a) Explique por que o algoritmo a seguir é guloso e não é ótimo.

   1: seja $S = \emptyset$
   2: **Enquanto** $V(G) \neq \emptyset$ **faça**
   3: escolha $v \in V(G)$ com peso $w(v)$ máximo
   4: $S = S \cup \{v\}$
   5: remova v e seus vizinhos de G
   6: **Devolve** S

   (b) Explique por que o algoritmo a seguir não é ótimo.

   1: seja $S_1 = \{v_i : i \text{ é ímpar}\}$
   2: seja $S_2 = \{v_i : i \text{ é par}\}$
   3: **Devolve** o conjunto de maior peso dentre S_1 e S_2

(a) O algoritmo é guloso porque a cada passo escolhe um vértice de maior peso, o que faz sentido com o objetivo de ter peso total máximo.
Ele não é ótimo, no entanto, por causa da seguinte instância:



O algoritmo escolheria $\{v_2\}$ como solução, com peso 10, sendo que $\{v_1, v_3\}$ é solução ótima, com peso 20.

(b) O algoritmo não é ótimo por causa da seguinte instância:

A resposta do algoritmo é $\{v_1, v_3\}$, com peso 6, sendo que a solução ótima é $\{v_1, v_4\}$, com peso 10.

11. Forneça um algoritmo que recebe a matriz totalmente preenchida pelo algoritmo de programação dinâmica para o problema do Alinhamento de Sequências e constrói um alinhamento para as sequências da entrada. Um alinhamento pode ser representado por dois vetores, um para cada sequência. Não é necessário provar a corretude deste algoritmo.

```
CONSTROI ALINHAMENTO (X, m, Y, n, α, M)
    Sejam X'[1..m+n] e Y'[1..m+n] dois vetores
    k = 1   // vai indicar a posição do alinhamento
    i = m   // vai percorrer X
    j = n   //  "      "     Y
    enquanto i ≥ 1 e j ≥ 1
        se M[i][j] == M[i-1][j-1] + α(x_i, y_j)
            X'[k] = x_i ; i--
            Y'[k] = y_j ; j--
        senão se M[i][j] == M[i-1][j] + α(gap)
            X'[k] = x_i ; i--
            Y'[k] = '_'
        senão
            X'[k] = '_'
            Y'[k] = y_j ; j--
        k++
    enquanto i ≥ 1
        X'[k] = x_i ; i--
        Y'[k] = '_'
        k++
    enquanto j ≥ 1
        X'[k] = '_'
        Y'[k] = y_j ; j--
        k++
    devolve X', Y', k
```

OBS: X' e Y' estão invertidos

12. Nesta questão, considere o problema de fazer troco para n centavos usando o menor número total de moedas (mesmo da Questão 6). Você pode assumir que existe um número infinito de moedas disponíveis para cada valor. Forneça um algoritmo de programação dinâmica que seja ótimo para fazer troco tendo disponíveis moedas de valores $m_1, m_2, \ldots, m_t$ centavos, para $t \geq 1$, onde algum $m_k = 1$.

Denote por $T_{n,t}$ a quantidade mínima de moedas para o problema (n,t), que é o problema de fazer troco para o valor n tendo t valores de moedas disponível.

Vamos mostrar que

$$T_{n,t} = \begin{cases} \min\{T_{n-m_t,t} + 1,\ T_{n,t-1}\} & \text{se } n \geq m_t \\ T_{n,t-1} & \text{caso contrário} \end{cases}$$

Seja S uma sequência que contém as moedas usadas em uma solução ótima para (n,t). Então $m_t \in S$ ou $m_t \notin S$.

Se $m_t \in S$, então $S-(m_t)$ é solução ótima para $(n-m_t, t)$ (se não fosse, uma solução ótima para $(n-m_t,t)$ com uma moeda m_t seria melhor para (n,t)).

Se $m_t \notin S$, então S já é solução ótima para $(n,t-1)$.

Casos base: para $n=0$, podemos usar zero moedas $\therefore T_{0,\_} = 0$.

Para $t=0$, é impossível fazer troco (se $n>0$) $\therefore T_{\_,0} = \infty$.

Para o algoritmo, vamos manter uma matriz $T[0..n][0..t]$ que vai guardar $T_{i,j}$ em $T[i][j]$.

```
PD_TROCO(n, t)
1   Seja T[0..n][0..t] matriz
2   para j = 0 a t
3       T[0][j] = 0
4   para i = 1 a n
5       T[i][0] = ∞
6   para i = 1 a n
7       para j = 1 a t
8           se i ≥ m_j
9               T[i][j] = min{T[i-m_j][j]+1, T[i][j-1]}
10          senão
11              T[i][j] = T[i][j-1]
12  devolve T[n][t]
```

O algoritmo tem inicialmente dois laços de tempo $\Theta(t)$ e $\Theta(n)$ cada e o terceiro laço leva tempo $\Theta(nt)$, já que para cada valor de i entre 1 e n o laço interno executa t vezes e todas as operações feitas são constantes $\therefore$ total $= \Theta(nt)$

13. Considere o problema *Conjunto Independente de Peso Máximo em Caminhos*, apresentado na Questão 7. Forneça um algoritmo de programação dinâmica que resolve esse problema otimamente.

Denote por I_n o custo de uma solução ótima para o problema (n), isto é, o peso de um conj. indep. mínimo em um grafo caminho com n vértices.
Vamos mostrar que

$$I_n = \min \{ I_{n-2} + w(v_n), I_{n-1} \}.$$

Seja S solução ótima para (n).
Se $v_n \in S$, então $v_{n-1} \notin S$ e $S \setminus \{v_n\}$ é ótima para o problema $(n-2)$ (se não, uma ótima para $(n-2)$ com o vértice v_n seria melhor que S para (n)).
Se $v_n \notin S$, então S já é ótima para $(n-1)$.
Caso base: se $n = 0$, não há vértices e $I_0 = 0$.
Se $n = 1$, só há um vértice e $I_1 = w(v_1)$.

Para o algoritmo, vamos ter um vetor $I[0..n]$ que vai guardar, em $I[i]$ o valor I_i.

```
CONJ_IND_CAM (G, w, n)
1    Seja I[0..n] um vetor
2    I[0] = 0
3    I[1] = w(v_1)
4    para i = 2 até n
5    :      I[i] = min{ I[i-2] + w(v_i) , I[i-1] }.
6    devolva I[n]
```

O algoritmo tem um único laço que executa aprox. n vezes fazendo operações constantes.
Logo, o tempo é $\Theta(n)$.

Extra: Por que a prova do Ex. 6 não funciona para qualquer conjunto de moedas?

No meio da prova, usamos o fato de que dentre as moedas usadas pela solução ótima sempre há um conjunto de moedas que podem ser removidas e substituídas por menos moedas de maior valor. E isso funciona bem para 50, 25, 10, 5, 1 mas não para 16, 10, 7, 1. Veja o porquê a seguir.

Lema 1: Sejam $a_1, a_2, a_3, a_4, x \in \mathbb{N}$ tais que $25a_1 + 10a_2 + 5a_3 + a_4 \geq 50x$. Então existem $b_1, b_2, b_3, b_4 \in \mathbb{N}$ tais que $b_i \leq a_i$, $1 \leq i \leq 4$, $25b_1 + 10b_2 + 5b_3 + b_4 = 50x$ e $b_1 + b_2 + b_3 + b_4 > x$.

Demonstração: Por indução em x.

Quando $x = 1$, $25a_1 + 10a_2 + 5a_3 + a_4 \geq 50$ (★)

Se $a_1 \geq 2$, tome $b_1 = 2$, $b_2 = b_3 = b_4 = 0$.

Se $a_2 \geq 5$, tome $b_2 = 5$, $b_1 = b_3 = b_4 = 0$

Se $a_3 \geq 10$, tome $b_3 = 10$, $b_1 = b_2 = b_4 = 0$.

Se $a_4 \geq 50$, tome $b_4 = 50$, $b_1 = b_2 = b_3 = 0$.

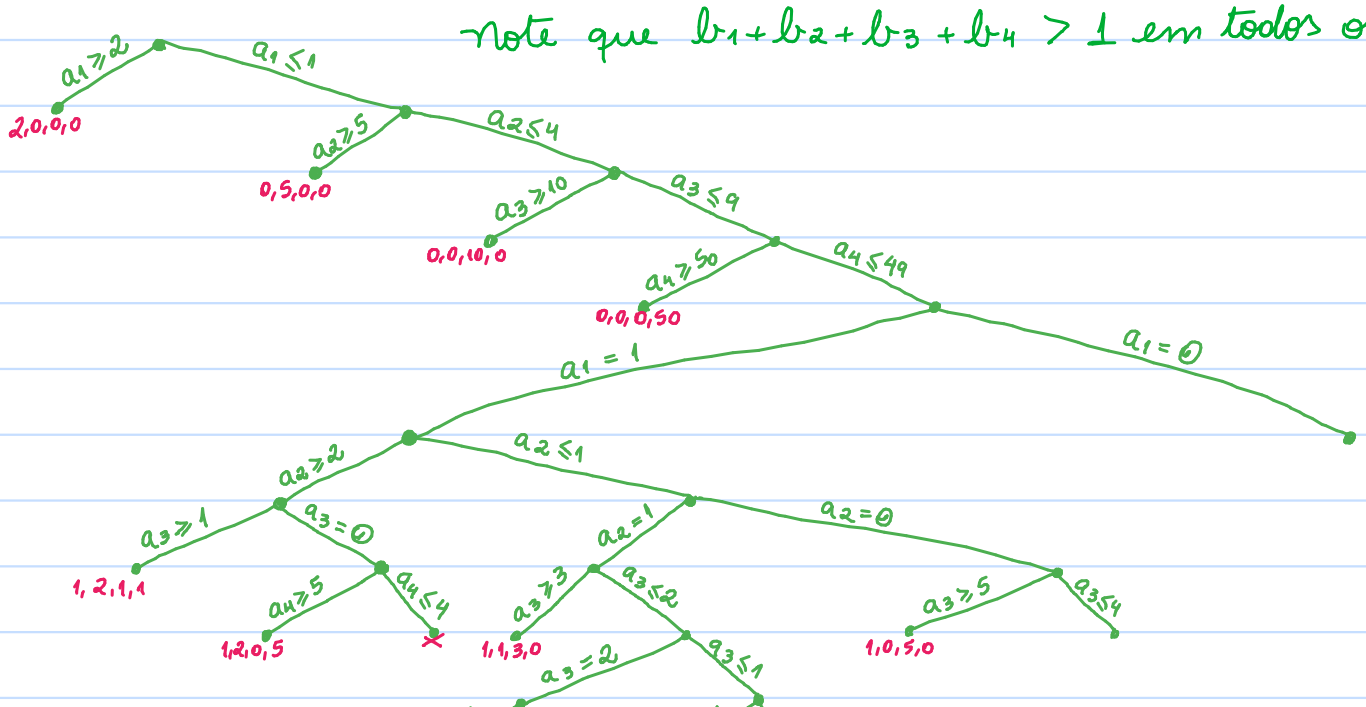
Então $a_1 \leq 1$, $a_2 \leq 4$, $a_3 \leq 9$ e $a_4 \leq 49$.

Se $a_1 = 1$, $a_2 \geq 2$ e $a_3 \geq 1$, então tome $b_1 = 1$, $b_2 = 2$, $b_3 = 1$ e $b_4 = 0$

Se $a_1 = 1$, $a_2 \geq 2$ e $a_3 = 0$, então $a_4 \geq 5$ pois vale ★. Tome $b_1 = 1$, $b_2 = 2$ $b_3 = 0$ e $b_4 = 5$.

Se $a_1 = 1$, $a_2 = 1$ e $a_3 \geq 1$ ....

Note que $b_1 + b_2 + b_3 + b_4 > 1$ em todos os casos.



Então seja $x > 1$. Logo, $50x > 50$

Tome b_1, b_2, b_3, b_4 como no caso base tais que $25b_1 + 10b_2 + 5b_3 + b_4 = 50$. Note que $b_1 + b_2 + b_3 + b_4 > 1$.

Note que $25(a_1 - b_1) + 10(a_2 - b_2) + 5(a_3 - b_3) + (a_4 - b_4) \geq 50(x-1)$.

Por HI, existem $c_1, c_2, c_3, c_4 \in \mathbb{N}$ tais que $c_1 + c_2 + c_3 + c_4 > x - 1$ e $25c_1 + 10c_2 + 5c_3 + c_4 = 50(x-1)$, com $c_i \leq (a_i - b_i)$, $1 \leq i \leq 4$.

Então vale que

$$25(c_1 + b_1) + 10(c_2 + b_2) + 5(c_3 + b_3) + (c_4 + b_4) = 50x$$

e $c_1 + b_1 + c_2 + b_2 + c_3 + b_3 + c_4 + b_4 > x - 1 + 1 = x$.

CQD

"Lema 2": Sejam $a_1, a_2, a_3, x \in \mathbb{N}$ tais que $10a_1 + 7a_2 + a_3 \geq 16x$.
Então existem $b_1, b_2, b_3 \in \mathbb{N}$ tais que $b_i \leq a_i$, $1 \leq i \leq 3$,
$10b_1 + 7b_2 + b_3 = 16x$ e $b_1 + b_2 + b_3 > x$.

Falso! Tome $a_1 = 2$, $a_2 = a_3 = 0$ e $x = 1$.
Vale que $10 \cdot 2 + 7 \cdot 0 + 0 = 20 \geq 16 = 16 \cdot 1$
Deveríamos ter $b_1 \leq 2$, $b_2 = b_3 = 0$ e $10b_1 = 16$,
o que é impossível.