Lista 5 — MC458 — Gabarito
Tema: Algoritmos por Indução e Algoritmos Gulosos

1. Potenciação Rápida

Resolução:

a) **Potenciação Linear** ($x^n = x \cdot x^{n-1}$)

Algoritmo em Pseudocódigo:

```
PotenciaLinear(x, n):
  se n == 0 então
    retorne 1
  senão
    retorne x * PotenciaLinear(x, n - 1)
```

Prova de Corretude (por Indução Fraca em n):

- **Base da Indução ($n = 0$):** Para $n = 0$, o algoritmo atinge o caso base e retorna 1. Como $x^0 = 1$ para todo real $x \neq 0$, o caso base está correto.
- **Hipótese de Indução (HI):** Assuma que a chamada `PotenciaLinear(x, k)` devolve corretamente x^k para um inteiro $k = n - 1 \geq 0$.
- **Passo Indutivo ($n \geq 1$):** Para $n > 0$, o algoritmo executa a chamada recursiva `PotenciaLinear(x, n - 1)` e multiplica o resultado por x. Pela Hipótese de Indução, a chamada recursiva devolve x^{n-1}. Logo, o algoritmo retorna:

$$x \cdot x^{n-1} = x^n$$

Por indução fraca, o algoritmo está correto para todo $n \geq 0$.

Contagem de Multiplicações: Seja $M(n)$ o número de multiplicações efetuadas para calcular x^n.

- Para $n = 0$, nenhuma multiplicação é feita: $M(0) = 0$.
- Para $n \geq 1$, realizamos 1 chamada recursiva para $n - 1$ e 1 multiplicação por x:

$$M(n) = M(n-1) + 1$$

Desenrolando a recorrência passo a passo:

$$M(n) = M(n-1) + 1 = M(n-2) + 2 = \cdots = M(n-k) + k$$

Substituindo $k = n$, atingimos a base $M(0)$:

$$M(n) = M(0) + n = 0 + n = n$$

Portanto, o algoritmo realiza exatamente n **multiplicações**.

b) **Potenciação Rápida ($x^n = (x^{\lfloor n/2 \rfloor})^2$ ou $x^n = x \cdot (x^{\lfloor n/2 \rfloor})^2$)**

Algoritmo em Pseudocódigo:

```
PotenciaRapida(x, n):
  se n == 0 então retorne 1
  se n == 1 então retorne x

  k = ⌊n / 2⌋
  y = PotenciaRapida(x, k)

  se n % 2 == 0 então
    retorne y * y
  senão
    retorne x * (y * y)
```

Prova de Corretude (por Indução Forte em n):

- **Casos Base:** Para $n = 0$, retorna $1 = x^0$. Para $n = 1$, retorna $x = x^1$. Ambos corretos.
- **Hipótese de Indução Forte (HI):** Assuma que `PotenciaRapida(x, k)` retorna corretamente x^k para todo inteiro k tal que $0 \leq k < n$.
- **Passo Indutivo ($n \geq 2$):** Seja $k = \lfloor n/2 \rfloor$. Como $n \geq 2$, temos $1 \leq k < n$. Pela HI, a variável y recebe o valor correto $x^{\lfloor n/2 \rfloor}$.
  - **Se n for par ($n = 2k$):** O algoritmo devolve $y \cdot y = (x^k)^2 = x^{2k} = x^n$.
  - **Se n for ímpar ($n = 2k+1$):** O algoritmo devolve $x \cdot (y \cdot y) = x \cdot (x^k)^2 = x \cdot x^{2k} = x^{2k+1} = x^n$.

  Portanto, por indução forte, o algoritmo está correto para todo $n \geq 0$.

Limitante de Multiplicações $M(n) \leq M(\lfloor n/2 \rfloor)+2$: O valor $y =$ `PotenciaRapida`$(x, \lfloor n/2 \rfloor)$ é calculated apenas uma vez. Se n for par, fazemos 1 multiplicação ($y \cdot y$). Se n for ímpar, fazemos 2 multiplicações ($y \cdot y$ e a multiplicação por x). Assim, para $n > 1$:

$$M(n) \leq M(\lfloor n/2 \rfloor) + 2 \qquad \text{com } M(1) = 0$$

Prova por Indução Forte de que $M(n) \leq 2\lfloor \log_2 n \rfloor$ para $n \geq 1$:

- **Base ($n = 1$):** $M(1) = 0$. Pela fórmula: $2\lfloor \log_2 1 \rfloor = 2 \cdot 0 = 0$. Como $0 \leq 0$, vale a base.
- **Hipótese de Indução (HI):** Suponha que $M(k) \leq 2\lfloor \log_2 k \rfloor$ para todo $1 \leq k < n$.
- **Passo Indutivo ($n \geq 2$):** Seja $k = \lfloor n/2 \rfloor$. Aplicando a HI a k:

$$M(k) \leq 2 \left\lfloor \log_2 \left\lfloor \frac{n}{2} \right\rfloor \right\rfloor = 2(\lfloor \log_2 n \rfloor - 1)$$

Substituindo na recorrência:

$$M(n) \leq M\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 \leq 2(\lfloor \log_2 n \rfloor - 1) + 2 = 2\lfloor \log_2 n \rfloor$$

Conclui-se que $M(n) \leq 2\lfloor \log_2 n \rfloor$ para todo $n \geq 1$.

2. Parênteses Balanceados

Resolução:

a) **Insuficiência da Hipótese Fraca**

Se a hipótese de indução mantiver apenas a informação de se a sequência de tamanho $n-1$ é balanceada ou não, o retorno é apenas um valor booleano (`Verdadeiro` ou `Falso`).

Considere as duas sequências:

- Sequência $A =$ `"("` (tamanho 1)
- Sequência $B =$ `"(("` (tamanho 2)

Demonstração:

1) Ambas as sequências A e B **não são balanceadas**. A hipótese devolve `Falso` para ambas.
2) Acrescentando o símbolo `)` ao final de cada uma delas para formar o tamanho n:
   - $A' = A +$ `")"` $=$ `"()"` $\implies$ **É balanceada** (`Verdadeiro`).
   - $B' = B +$ `")"` $=$ `"(())"` $\implies$ **Não é balanceada** (`Falso`).

Duas entradas que tinham a mesma resposta no tamanho $n-1$ (`Falso`) geraram respostas diferentes no tamanho n após a mesma alteração. Isso prova que a hipótese fraca é insuficiente.

b) **Algoritmo com Hipótese Reforçada, Prova e Complexidade**

Algoritmo em Pseudocódigo:

```
EhBalanceada(A, n):
  s = 0                                    // s:  saldo de '(' menos ')'
  v = Falso       // v:  indica se algum prefixo teve saldo negativo

  para i de 1 até n faça
    se A[i] == '(' então
      s = s + 1
    senão se A[i] == ')' então
      s = s - 1

    se s < 0 então
      v = Verdadeiro

  retorne (v == Falso) e (s == 0)
```

Prova de Corretude por Indução no Tamanho da Sequência: Seja $A[1..k]$ o prefixo dos primeiros k símbolos da sequência.

- **Invariante de Laço (Hipótese de Indução):** Após processar o k-ésimo símbolo:
  i) $s_k =$ (nº de `(` em $A[1..k]$) $-$ (nº de `)` em $A[1..k]$).
  ii) $v_k =$ `Verdadeiro` s.s.p. existe algum $j \in \{1, \ldots, k\}$ com $s_j < 0$.
- **Base ($k = 0$):** Na sequência vazia $A[1..0]$: $s_0 = 0$ (pois $0 - 0 = 0$) e $v_0 =$ `Falso` (nenhum prefixo lido). Correto.

- **Passo Indutivo ($k \geq 1$):** Suponha o invariante válido para $k-1$. Ao ler $A[k]$:
  - Se $A[k] =$ `'('`, o saldo atualiza para $s_k = s_{k-1} + 1$.
  - Se $A[k] =$ `')'`, o saldo atualiza para $s_k = s_{k-1} - 1$.

  Assim, s_k mantém a diferença exata entre `(` e `)`. A flag atualiza para $v_k = v_{k-1} \vee (s_k < 0)$. Pela HI, v_{k-1} registrava falhas prévias. Ao testar $s_k < 0$, v_k passa a indicar corretamente se houve violação até k.
- **Conclusão:** Ao final ($k = n$), o algoritmo retorna `Verdadeiro` se e somente se nenhum prefixo teve saldo negativo ($v_n ==$ `Falso`) e o saldo final é nulo ($s_n == 0$).

Complexidade de Tempo: O laço executa exatamente n iterações. Como cada passo realiza apenas operações aritméticas $O(1)$, o tempo total de execução é $O(n)$.

3. Cobertura de Pontos por Antenas de Raio Fixo

Resolução:

a) **Contraexemplo da Estratégia Gulosa Incorreta**

Instância do Contraexemplo: Raio $R = 1$ (uma antena no ponto p cobre o intervalo $[p-1, p+1]$) e vilas nas posições $x = [0, 1, 2]$.

Execução da Estratégia Proposta ($p = x_i$):

1) Primeira vila não coberta é $x_1 = 0$. Instala antena em $p_1 = 0$.
   - Alcance: $[-1, 1] \implies$ cobre as vilas $x_1 = 0$ e $x_2 = 1$.
2) Vila $x_3 = 2$ continua sem sinal. Instala segunda antena em $p_2 = 2$.
   - Alcance: $[1, 3] \implies$ cobre a vila $x_3 = 2$.
3) **Total de antenas da heurística:** 2 antenas.

Solução Ótima: Instalar **1 única antena** em $p = 1$. O alcance será $[0, 2]$, cobrindo as três vilas ($x_1 = 0, x_2 = 1, x_3 = 2$) simultaneamente.

b) **Estratégia Gulosa Correta, Prova e Complexidade**

Estratégia Gulosa Correta: Encontre a primeira vila não coberta mais à esquerda x_i e instale a antena no ponto mais à direita possível que ainda a cubra:

$$p = x_i + R$$

Prova de Corretude (Argumento "O Guloso Não Fica Para Trás"): Seja $G = \{g_1, g_2, \ldots, g_k\}$ as antenas do algoritmo guloso e $O = \{o_1, o_2, \ldots, o_m\}$ as antenas de uma solução ótima qualquer.

Provaremos por indução que $g_j \geq o_j$ para todo $j \geq 1$:

- **Base ($j = 1$):** Para cobrir a vila x_1, a ótima deve ter $o_1 \leq x_1 + R$. Como $g_1 = x_1 + R$, temos $g_1 \geq o_1$.
- **Passo Indutivo:** Suponha $g_j \geq o_j$. Como g_j está mais (ou igualmente) à direita que o_j, a primeira vila não coberta pelo guloso $x_{i'}$ estará na mesma posição ou à direita da primeira não coberta por o_j. Assim, a próxima antena $g_{j+1} = x_{i'} + R$ satisfará $g_{j+1} \geq o_{j+1}$.

Como o guloso cobre toda a rodovia mantendo suas antenas mais à direita, ele requer no máximo o mesmo número de antenas que a ótima ($k \leq m$). Logo, o algoritmo é ótimo.

Complexidade de Tempo: Com o vetor de vilas já ordenado, faz-se uma única varredura linear nas n vilas. O tempo de execução é $O(n)$.

4. Minimização do Atraso Máximo em Tarefas

Resolução:

a) **Contraexemplos para as Heurísticas Flutuantes**

i) **Menor Duração Primeiro (p_i crescente):**
- Tarefa 1: $p_1 = 1$, $d_1 = 100$.
- Tarefa 2: $p_2 = 10$, $d_2 = 10$.

Ordem da Heurística (1, 2): $C_1 = 1 \implies A_1 = 0$; $C_2 = 11 \implies A_2 = 11 - 10 = 1$. Atraso máximo = **1**.
Ordem Ótima (2, 1): $C_2 = 10 \implies A_2 = 0$; $C_1 = 11 \implies A_1 = 0$. Atraso máximo = **0**.

ii) **Menor Folga Primeiro ($d_i - p_i$ crescente):**
- Tarefa 1: $p_1 = 1$, $d_1 = 2 \implies$ folga $= 1$.
- Tarefa 2: $p_2 = 10$, $d_2 = 10 \implies$ folga $= 0$.

Ordem da Heurística (2, 1): $C_2 = 10 \implies A_2 = 0$; $C_1 = 11 \implies A_1 = 11 - 2 = 9$. Atraso máximo = **9**.
Ordem Ótima (1, 2): $C_1 = 1 \implies A_1 = 0$; $C_2 = 11 \implies A_2 = 11 - 10 = 1$. Atraso máximo = **1**.

b) **Prova da Otimidade da Regra EDD**

i) **Análise da Troca de uma Inversão Adjacente:** Considere duas tarefas consecutivas i e $j = i + 1$ com uma inversão de prazo ($d_i > d_{i+1}$). Seja s o tempo de início da tarefa i.
- Antes da troca: $C_i = s + p_i$ e $C_{i+1} = s + p_i + p_{i+1}$.
- Após a troca: $C'_{i+1} = s + p_{i+1}$ e $C'_i = s + p_{i+1} + p_i = C_{i+1}$.

Após a troca, a tarefa $i + 1$ termina mais cedo ($C'_{i+1} < C_{i+1}$), reduzindo seu atraso. O novo atraso de i é $A'_i = \max(0, C_{i+1} - d_i)$. Como $d_i > d_{i+1}$, temos $C_{i+1} - d_i < C_{i+1} - d_{i+1} = A_{i+1}$. Logo, $A'_i < A_{i+1}$. Trocar o par invertido **nunca aumenta o atraso máximo** do sistema.

ii) **Conclusão de Otimidade por Eliminação de Inversões:** Dada qualquer solução ótima S^*, se ela possuir tarefas fora da ordem EDD, conterá inversões adjacentes. Eliminando sucessivamente essas inversões por trocas (como no *Bubble Sort*), reduzimos as inversões sem aumentar o atraso máximo. Ao final das trocas, obtemos a sequência EDD, provando que o algoritmo EDD é ótimo.

5. O Problema do Motorista e Postos de Combustível

Resolução:

a) **Contraexemplo da Regra do Meio Tanque**

Instância do Contraexemplo: Tanque $C = 10$ km (gatilho $< C/2 = 5$ km), distância $D = 15.2$ km e postos em $x_1 = 5.1$ km e $x_2 = 10.0$ km.

Simulação:

1) **Em $x_1 = 5.1$ km:** Restam $10 - 5.1 = 4.9$ km. Como $4.9 < 5$, o algoritmo **reabastece** (Parada 1).
2) **Em $x_2 = 10.0$ km:** Restam $10 - 4.9 = 5.1$ km. Como $5.1 \geq 5$, o algoritmo **não para**.
3) **Destino $D = 15.2$ km:** A distância desde a parada em x_1 é $15.2 - 5.1 = 10.1$ km > 10 km. **O carro para sem combustível a 100m do destino**.

Solução Ótima: Não parar em x_1 ($5.1 \leq 10$) e fazer **1 única parada** no posto $x_2 = 10.0$ km.

b) **Algoritmo Guloso Correto, Pseudocódigo e Prova**

Pseudocódigo do Algoritmo:

```
MenorNumeroParadas(x, n, C, D):
  paradas = 0
  pos_atual = 0
  i = 0

  enquanto pos_atual + C < D faça
    prox = i
    enquanto prox + 1 <= n e x[prox + 1] <= pos_atual + C faça
      prox = prox + 1

    pos_atual = x[prox]
    paradas = paradas + 1
    i = prox

  retorne paradas
```

Prova por Indução ("O Guloso Não Fica Para Trás"): Seja g_k a k-ésima parada do guloso e o_k a da solução ótima.

- **Base ($k = 0$):** $g_0 = o_0 = 0$.
- **HI:** Assuma $g_k \geq o_k$.
- **Passo Indutivo ($k+1$):** A autonomia a partir de g_k alcança até $g_k + C \geq o_k + C$. A ótima deve escolher $o_{k+1} \leq o_k + C \leq g_k + C$. Como o guloso escolhe o **último posto alcançável** dentro de $g_k + C$, sua escolha satisfaz $g_{k+1} \geq o_{k+1}$.

Por indução, o guloso está sempre mais à frente. Logo, atinge o destino D com o menor número de paradas.

6. Organização Ótima de Arquivos em Fita Magnética

Resolução:

a) **Cálculo Comparativo para** $L = [2, 5, 10]$

i) **Ordem Decrescente** $[10, 5, 2]$**:** $T_1 = 10$, $T_2 = 15$, $T_3 = 17 \implies T_{\text{total}} = 10 + 15 + 17 = \mathbf{42}$.

ii) **Ordem Crescente** $[2, 5, 10]$**:** $T_1 = 2, T_2 = 7, T_3 = 17 \implies T_{\text{total}} = 2{+}7{+}17 = \mathbf{26}$.

b) **Prova de Otimidade por Análise de Contribuição de Pesos**

i) **Dedução da Soma Ponderada:** Para qualquer ordem π, o tempo de acesso ao k-ésimo arquivo é $T_k = \sum_{j=1}^{k} L_{\pi(j)}$. Somando todos os T_k:

- $L_{\pi(1)}$ é somado n **vezes**.
- $L_{\pi(2)}$ é somado $(n-1)$ **vezes**.
- O arquivo na posição k é somado $(n-k+1)$ **vezes**.

Reescrevendo a soma total:

$$T_{\text{total}} = n \cdot L_{\pi(1)} + (n-1) \cdot L_{\pi(2)} + \cdots + 1 \cdot L_{\pi(n)}$$

ii) **Minimização pela Desigualdade do Rearranjo:** Na fórmula acima, os multiplicadores/pesos são fixos e decrescentes: $W = [n, n-1, \ldots, 1]$. Pela Desigualdade do Rearranjo, para minimizar a soma dos produtos entre um conjunto de pesos e valores positivos, devemos associar os **maiores pesos aos menores valores**. O maior peso (n) deve multiplicar o menor arquivo, o segundo maior peso $(n-1)$ o segundo menor, e assim por diante. Portanto, a ordem que minimiza T_{total} é a ordem crescente de tamanho: $L_{\pi(1)} \leq L_{\pi(2)} \leq \cdots \leq L_{\pi(n)}$.