Avaliação Continuada 03

(Projetando Algoritmos Iterativos)

Questão 1

- **Questão 1 (Partição).** A entrada é uma lista $L[1..n]$ de números, com $n \geq 1$. Lembre-se que os nossos arrays nessa disciplina começam com o índice 1. O elemento $L[1]$ é chamado de pivô. Seja k o número de elementos de L com valor menor ou igual ao pivô. Seu objetivo é especificar um algoritmo de **tempo linear** $\Theta(n)$ para reorganizar os elementos de L de modo que as três condições a seguir sejam satisfeitas:

  - O pivô $L[1]$ seja reposicionado para a posição k.
  - Os elementos com valor menor ou igual ao pivô sejam colocados nas k primeiras posições.
  - Os elementos com valor maior que o pivô sejam colocados nas últimas $n - k$ posições.

  Obs. 1: Este problema poderia ser resolvido simplesmente ordenando a lista, mas neste caso gastaria tempo $\Theta(n \log n)$.

  Obs. 2: Este algoritmo é utilizado pelo algoritmo de ordenação Quicksort.

  Passos Básicos: Utilize dois índices i e j. O índice i vai da posição 2 em direção em final da lista, e o índice j vai da última posição em direção ao início da lista. Em cada iteração, avance o i se $L[i]$ é menor ou igual ao pivô. Caso contrário, avance o j se $L[j]$ é maior que o pivô. Se nenhuma destas condições ocorrer, troque $L[i]$ com $L[j]$ e avance i e j. A repetição termina quando $j < i$. Finalmente, reposicione o pivô de modo que apenas elementos maiores que o pivô estejam à direita dele.

  Invariante de laço:
  INV1: Todos os elementos nas posições menores que i possuem valor menor ou igual ao pivô.
  INV2: Todos os elementos nas posições maiores que j possuem valor maior que o pivô.

  ## Forneça resposta para cada um dos itens abaixo:

  a. Forneça uma medida de progresso, e argumente que o algoritmo termina.
  b. Indique como estabelecer o invariante do laço, ou seja, quais ações no código pré-laço tornam o invariante do laço verdadeiro na

primeira iteração. Justifique.
c. Qual o código do laço? Argumente que ele mantém o invariante do laço.
d. Argumente que o invariante do laço e a condição de saída garantem que, assim que a execução sair do laço teremos $j = i - 1$.
e. Qual o código pós-laço? Argumente que as pós-condições são satisfeitas.
f. Forneça o pseudocódigo.
g. Considere os casos especiais a seguir e indique, caso necessário, quais adaptações devem ser feitas no algoritmo para atendê-los. (i) A lista tem apenas 1 elemento. (ii) Nenhum elemento é maior que o pivô. (iii) Exceto o pivô, todo os outros elementos são maiores que o pivô.
h. Forneça a complexidade de tempo de pior caso em notação O.

Questão 2

- **Questão 2 (Intercalação).** Nesta questão, queremos que você resolva o problema da intercalação de listas.

  Entrada: A entrada são duas listas $L[1..p]$ e $M[1..q]$ de números, com $p \geq 1$ e $q \geq 1$, tal que as duas listas estão ordenadas em ordem crescente. Lembre-se que as nossas listas nessa disciplina começam com o índice 1.

  Objetivo: Seu objetivo é especificar um algoritmo de **tempo linear** $\Theta(n)$ para criar uma terceira lista $N[1..p+q]$ contendo os elementos das listas L e M em ordem crescente. Essa nova lista N deve ser retornada como resultado do seu algoritmo.

  Saída: Lista $N[1..p+q]$ ordenada em ordem crescente.

  Obs. 1: Este problema poderia ser resolvido simplesmente copiando os elementos de L e M em N e ordenando a lista N, mas neste caso gastaria tempo $\Theta(n \log n)$.

  Obs. 2: Este problema é uma leve modificação de um problema semelhante que ocorre como subrotina no algoritmo de ordenação Mergesort.

  Passos Básicos: Utilize dois índices i e j. O índice i indica a posição do menor elemento da lista L que ainda não foi copiado para a lista de saída, e o índice j indica a posição do menor elemento da lista M que ainda não foi copiado para a lista de saída. Em cada iteração, avançe o i se $L[i] \leq M[j]$. Caso contrário, avançe o j. A repetição termina quando $(i > p)$ ou $(j > q)$. Neste momento, uma das listas ficou vazia, mas a outra ainda pode conter elementos. É preciso copiar a lista resultante no vetor de saída.

Forneça resposta para cada um dos itens abaixo:

a. Forneça uma invariante de laço para o laço principal do seu algoritmo.

b. Forneça uma medida de progresso, e argumente que o algoritmo termina.
c. Indique como estabelecer o invariante do laço, ou seja, quais ações no código pré-laço tornam o invariante do laço verdadeiro na primeira iteração. Justifique.
d. Qual o código do laço? Argumente que ele mantém o invariante do laço.
e. Argumente que o invariante do laço e a condição de saída garantem que, assim que a execução sair do laço teremos $(i > p)$ ou $(j > q)$.
f. Qual o código pós-laço? Argumente que as pós-condições são satisfeitas.
g. Forneça o pseudocódigo.
h. Considere os casos especiais a seguir e indique, caso necessário, quais adaptações devem ser feitas no algoritmo para atendê-los. (i) As listas dadas como entrada têm apenas 1 elemento. (ii) Os elementos de uma das listas são todos iguais.
i. Forneça a complexidade de tempo de pior caso em notação O.

Respostas

Questão 1

A.

Medida de progresso: Podemos usar como métrica o número de pares (i, j) ainda não explorados, ou simplesmente a soma dos deslocamentos de i e j: a cada iteração ou i avança ou j recua (ou ambos, em caso de *swap*).

Argumento de término:

- Inicialmente, $i = 2$ e $j = n$;
- A cada iteração, pelo menos um dos ponteiros move-se em direção ao outro;
- O laço executa enquanto $i \leq j$ e termina assim que $i > j$;
- Como ambos percorrem no máximo $n - 1$ posições, temos $\Theta(n)$ iterações no pior caso.

B.

INV1: Todos os elementos nas posições menores que i possuem valor menor ou igual ao pivô.
INV2: Todos os elementos nas posições maiores que j possuem valor maior que o pivô.

Inicialização:

- Antes do laço, definimos $i = 2$ e $j = n$;
- Os subvetores $[2..\,i-1]$ e $[j+1..\,n]$ estão vazios, logo vacuamente satisfazem INV1 e INV2, respectivamente.

C.

Pseudocódigo:

```
while i ≤ j do
  if L[i] ≤ pivo then
    i++
  else if L[j] > pivo then
    j++
  else
    swap L[i], L[j]
    i++
    j--
end
```

Manutenção:

- Se avançamos i porque $L[i] \leq pivo$, preservamos que tudo menor que i segue menor ou igual ao $pivo$;
- Se recuamos j porque $L[j] > pivo$, preservamos que tudo maior que j segue maior que $pivo$;
- Se trocamos $L[i]$ com $L[j]$, trazemos um valor menor ou igual ao $pivo$ para a região à esquerda de i e um valor maior que $pivo$ para a região à direita de j, antes de atualizar i e j;

D.

O laço termina exatamente quando $i > j$, ou seja quando $j - i + 1 \leq 0$. Como $j - i + 1$ é sempre inteiro e começa em $n - 1$, a primeira vez que $j - i + 1$ deixa de ser um positivo não-nulo é quando $j - i + 1 = 0$.

Logo, na saída do laço, temos

$$j - i + 1 = 0 \longrightarrow j = i - 1$$

E.

Código pós-laço:

```
swap L[1] com L[j]
```

Argumento: Após o loop, `j = i - 1`. O subvetor `L[2..j]` contém elementos ≤ pivô e `L[i..n]` > pivô. A troca posiciona o pivô na posição `j` (exatamente `k`), garantindo:

- `L[1..j-1]`: elementos ≤ pivô
- `L[j]`: pivô
- `L[j+1..n]`: elementos > pivô

F.

```
def particao(L):
    pivo = L[1]
```

```
    i = 2
    j = len(L)
    while i <= j:
        if L[i] <= pivo:
            i += 1
        elif L[j] > pivo:
            j -= 1
        else:
            swap L[i] e L[j]
            i += 1
            j -= 1
    swap L[1] e L[j]
    return L
```

G.

- **1 elemento:** Loop não executa, troca `L[1]↔L[1]` mantém invariantes
- **Nenhum > pivô:** `j` permanece em `n`, pivô vai para `L[n]`
- **Todos > pivô:** `j` chega a 1, pivô fica em `L[1]`

H.

No pior caso (como quando o pivô acaba em uma extremidade da lista), os ponteiros percorrem praticamente todo o vetor.

- Loop de avanço de i até $i = n + 1$.
  Total: aproximadamente $2(n - 1)$ operações.
- Loop de recuo de j até $j = 1$.
  Total: aproximadamente $2(n - 1)$ operações.
- Laço externo (swaps e comparações) executa no máximo n vezes.
  Total: aproximadamente $3n$ operações.
- *Swap* final do pivô, custo constante.

Total: 2 operações.

Polinômio total:

$$f(n) = 2(n-1) + 2(n-1) + 3n + 2$$
$$= 2n - 2 + 2n - 2 + 3n + 2$$
$$= 7n - 2$$

Prova de *f(n) = O(n)*

Desejamos provar que existe uma constante `c > 0` e um `n₀` tal que:

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

Escolhendo **c = 8** e **n_0 = 1**, temos:

$f(n) = 7n - 2 \leq 7n \leq 8n = c \cdot n$

Logo, pela definição formal:

$f(n) = O(n)$

Questão 2

A.

INV: `N[1..k-1]` contém os `k-1` menores elementos ordenados. `L[i..p]` e `M[j..q]` contêm elementos restantes não copiados.

B.

Medida: $(p - i) + (q - j)$

Término: Medida reduz em 1 por iteração, atinge 0 em $p + q$ passos

C.

Inicialização: $i = j = k = 1$, `N[1..0]` vazio satisfaz INV vacuamente

D.

```
while i ≤ p e j ≤ q:
    if L[i] ≤ M[j]:
        N[k] = L[i]
        i += 1
    else:
        N[k] = M[j]
        j += 1
    k += 1
```

E.

Condição de saída $(i > p) \vee (j > q)$ ocorre quando uma lista é esgotada, garantido pela medida de progresso

F.

```
# Copia restantes de L
while i ≤ p:
  N[k] = L[i]
  i += 1
  k += 1

# Copia restantes de M
while j ≤ q:
  N[k] = M[j]
  j += 1
  k += 1
```

G.

```
merge(L, M):
  p, q = len(L), len(M)
  N = nova_lista(p+q)
  i = j = k = 1
  while i ≤ p e j ≤ q:
    if L[i] ≤ M[j]:
      N[k] = L[i]
      i += 1
    else:
      N[k] = M[j]
      j += 1
    k += 1
```

```
# Copia restantes
while i ≤ j do
  if L[i] ≤ pivo then
    i++
  else if L[j] > pivo then
    j++
  else
    swap L[i], L[j]
    i++
    j--
end
return N
```

H.

- **1 elemento:** Funciona normalmente com comparação única;
- **Elementos iguais:** Mantém ordem estável com condição menor ou igual.

I.

Complexidade: $O(p + q)$

- Cada elemento processado exatamente 1 vez;
- 2 loops adicionais de cópia: $O(p + q)$.