# Estrutura de Dados

José Gildásio Freitas do Ó - 473901

Maio 2023

1. Escreva um algoritmo de remoção que, dada uma Lista Sequencial L e um valor de entrada x, remove de L todos os valores menores que x.

   - **Resposta:**
     - Algoritmo: RemoverMenores(L, tam, x)
     - Entrada: Vetor L, tam é o tamanho da lista, x é o valor a ser comparado e removido os menores.
     - Saida: Vetor L com todos os elementos menores que x removidos e tamanho alterado da lista.

```
1– int tam2 = tam
2– int L2[tam2]
3– int j = 0
4– para int i = 0 até i < tam então
5– |     se L[i] >= x então
6– |     |   L2[j] = L[i]
7– |     |   j++
8– |   i++
9– para int j = 0 até j <tam2 faça
10– |     se L2[j] < x então
11– |     |   remove(L2, tam2, L2[j])
12– |     |   tam2 - -
13– |   j++
14– para int i = 0 até i < tam faça
15– |     remove(L, tam, L[i])
16– |   i++
17– tam2 = j
18– para int i = 0 até i < tam2 faça
19– |     L[i] = L2[i]
20– |   i++
21– tam = tam2
```

2. Refaça o algoritmo da Questão 1 de forma a garantir que ele passe pelos elementos do vetor apenas uma vez ao longo de todo o processo. (Em outras palavras, não usar mais que um laço).

   - **Resposta:**
     - Algoritmo: RemoverMenores(L, tam, x)
     - Entrada: Vetor L, tam é o tamanho da lista, x é o valor a ser comparado e removido os menores.
     - Saida: Vetor L com todos os elementos menores que x removidos e tamanho alterado da lista.

```
1– para int i = 0 até i <= a tam faça
2– |     se L[ i ] < x então
3– |     |   remove(L, tam, L[ i ])
4– |     |   tam - -
5– |   i++
6– se L[ 0 ] < x então
7– |   removemenores(L, tam, x)
```

3. Escreva uma versão de algoritmo de inserção que garanta não gerar elementos duplicados na Lista informada. Isso quer dizer que, caso o valor a inserir já exista, a inserção deve falhar.

   - **Resposta:**
     - Algoritmo:
     - Entrada:
     - Saida:

```
1– para int i = 0 até i < tam faça
2– |     se L[ i ] == x então
3– |     |   return
4– tam++
5– L[tam-1] = x
```

4. Escreva um algoritmo para eliminar todas as ocorrências repetidas dos elementos em uma lista. Isso significa que, ao final da execução, cada elemento aparece uma única vez.

   - **Resposta:**
     - Algoritmo:
     - Entrada:
     - Saida:

```
1– para int i = 0 até i < tam faça
2– |     para int j = i+1 até j < tam faça
3– |     |   se L[ i ] == L[ j ] então
4– |     |   |   para int k = j até k < tam faça
5– |     |   |   |     L[ k ] = L[ k+1 ]
6– |     |   |   |     k++
7– |     |   |   tam - -
8– |     |   |   j - -
9– |     j++
10– i++
```

5. Podemos reescrever o algoritmo da Questão 4 de uma maneira mais eficiente caso a lista passada esteja ordenada? Justifique e, em caso afirmativo, escreva essa nova versão.

   - **Resposta:**
     - Algoritmo:
     - Entrada:
     - Saida:

```
1–  int tam2 = tam
2–  L2[tam2]
3–  int j = 0
4–  para int i = 0 até i < tam faça
5–  |     se L[ i ] == L[i+1] então
6–  |     |   L2[ j ] = L[ i ]
7–  |     |   j++
8–  |   i++
9–  para int i = 0 até i < tam faça
10–  |   remove(L, tam, L[ i ])
11–  |   i++
12–  tam2 = j
13–  para int i = 0 até i < tam2 faça
14–  |   L2[ i ] = L2[ i ]
15–  |   i++
16–  tam = tam2
```

6. Considere que estivemos mantendo nossa Lista Sequencial L sempre disposta em ordem crescente. Para facilitar nosso trabalho, desejamos usar uma operação de inserção que garanta sempre preservar essa propriedade na nossa lista, independentemente do novo elemento a ser inserido. Escreva essa versão de inserção.

   - **Resposta:**
     - Algoritmo:
     - Entrada:
     - Saida:

```
1–  int i = 0
2–  enquanto i < tam && L[ i ] < x faça
3–  |   i++
4–  para int j = tam até j > i faça
5–  |   L[ i ] = L[ j-1 ]
6–  |   j - -
7–  L[ i ] = x
8–  tam++
```

7. Escreva um algoritmo que receba duas Listas Sequenciais L1 e L2 , que estarão garantidamente com seus elementos dispostos em ordem crescente, e remova de L1 quaisquer ocorrências dos elementos encontrados em L2 .

   - **Resposta:**
     - Algoritmo:
     - Entrada:
     - Saida:

```
1–  int i = 0
2–  int j = 0
3–  para int k = 0 até k < tam faça
4–  |       se i >= tam então
5–  |       |   break
6–  |       se L[ i ] < L2[ j ] então
7–  |       |   L[ k ] = L[ i ]
8–  |       |   i++
9–  |        senão se L[ i ] > L2[ j ] então
10–  |       |   j++
11–  |       senão
12–  |       |   i++
13–  |       |   tam - -
14–  k++
```

8. Refaça o algoritmo da Questão 7 de forma a garantir que ele passe pelos elementos de ambas as listas apenas uma vez ao longo de todo o processo. (De uma certa forma, não usar mais que um laço por lista).

   - **Resposta:**
     - Algoritmo:
     - Entrada:
     - Saida:

```
1–  int i = 0
2–  int j = 0
3–  int k = 0
4–  enquanto i < tam faça
5–  |  se L1[ i ] < L2[ j ] então
6–  |  |  L1[ k ] = L1[ i ]
7–  |  |  i++
8–  |  |  k++
9–  |  senão se L1[ i ] > L2[ j ]
10– |  |  j++
11– |  senão
12– |  |  i++
13– |  |  tam - -
```

9. Considere que estivemos usando Listas Sequenciais para representar conjuntos, de forma que precisa- mos garantir que nenhum elemento ocorra mais de uma vez em uma lista. Escreva algoritmos para representar as operações de União e de Inserseção entre duas dessas listas, de forma que o resultado seja retornado como uma nova lista.

   - **Resposta:**
     - Algoritmo:
     - Entrada:
     - Saida:

```
1– int* uniao = neu int[tamL1 + tamL2]
2– int i = 0
3– int j = 0
4– int k = 0
5– enquanto i < tamL1 && j < tamL2 faça
6– se L1[ i ] < L2[ j ] então
7– uniao[k] = L1[ i ]
8– i++
9– k++
10– senão se L1[ i ] > L2[ j ] então
11– uniao [k] = L2[ j ]
12– j++
```

13– k++
14– **senão**
15– uniao[k] = L1[ i ]
16– i++
17– j++
18– k++
19– tamanhoUniao = k
20– retornar uniao

– Algoritmo:
– Entrada:
– Saida:

1– int* intersecao = new int[tamL1]
2– int i = 0
3– int j = 0
4– int k = 0
5– **enquanto** i < tamL1 && j < tamL2 **faça**
6– **se** L1[ i ] < L2[ j ] **então**
7– i++
8– **senão se** L1[ i ] > L2[ j ]
9– j++
10– **senão**
11– intersecao[k] = L1[ i ]
12– i++
13– j++
14– k++
15– tamanhoIntersecao = k
16– retornar intersecao

10. Escreva um algoritmo que receba duas Listas Sequenciais, ambas com elementos dispostos em ordem crescente, e retorna uma nova lista contendo os elementos de ambas as listas de entrada, também dispos- tos em ordem crescente. Seu algoritmo deve funcionar sem a necessidade de alocar memória adicional (a não ser para a nova lista) e sem a necessidade de executar um procedimento de ordenação após a movimentação.

    - **Resposta:**
        - Algoritmo:
        - Entrada:
        - Saida:

```
1– int i = 0
2– int j = 0
3– int k = 0
4– enquanto i < tamL1 && j < tamL2
5– se L1[ i ] < L2[ j ] então
6– novaLista[k] = L1[ i ]
7– i++
8– senão
9– novaLista[k] = L2[ j ]
10– j++
11– k++
12– enquanto i < tamL1 faça
13– novaLista[k] = L1[ i ]
14– i++
15– k++
16– enquanto j < tamL2 faça
17– novaLista[k] = L2[ j ]
18– j++
19– k++
```