Tarea Examen 1

Julio César Lozano Garnica. No. de cuenta UNAM: 420095390

04 de septiembre del 2025

Esta es la primer tarea examen de la materia *Estructuras de Datos* impartida por el profesor Erick Quintero Villeda, el ayudante Diego Jesus Vidal Aguilar, y la ayudante de laboratorio Sandra Valeria Rivera Lara.

1. TDA Persona.

TDA Persona (VALORES: Las personas mexicanas;
OPERACIONES: *devPL1Apellido*, *devPL2Apellido*, *devPLNombre*, *devFechaNacimiento*, *devGenero*, *devEntidadFederativa*, *devConsonantes*, *devUltimosDigitos*, *calcularCURP*)

Descripción de operaciones:

1. *devPL1Apellido*: Devuelve las primeras dos letras del primer apellido de la persona.
   - *devPL1Apellido(Persona)* $\rightarrow$ *Cadena*
2. *devPL2Apellido*: Devuelve la primera letra del segundo apellido de la persona.
   - *devPL2Apellido(Persona)* $\rightarrow$ *Cadena*
3. *devPLNombre*: Devuelve la primera letra del primer nombre de la persona.
   - *devPLNombre(Persona)* $\rightarrow$ *Cadena*
4. *devFechaNacimiento*: Devuelve una representación de la fecha de nacimiento de la persona.
   - *devFechaNacimiento(Persona)* $\rightarrow$ *Cadena*
5. *devGenero*: Devuelve la representación del género de la persona.
   - *devGenero(Persona)* $\rightarrow$ *Cadena*
6. *devEntidadFederativa*: Devuelve la representación de la entidad federativa donde vive la persona.
   - *devEntidadFederativa(Persona)* $\rightarrow$ *Cadena*
7. *devConsonantes*: Devuelve la primera consonante después de las primeras dos letras del primer apellido, seguida de la primera consonante después de la primera letra del segundo apellido, y la primera consonante después de la primera letra del primer nombre.
   - *devConsonantes(Persona)* $\rightarrow$ *Cadena*
8. *devUltimosDigitos*: Devuelve los dos últimos números de mi número de cuenta UNAM.
   - *devUltimosDigitos(Persona)* $\rightarrow$ *Cadena*
9. *calcularCURP*: Devuelve el CURP de la persona.

- *calcularCURP(Persona)* → *Cadena*

Axiomas:

1. La operación *devPL2Apellido* devolverá la cadena "X" si la persona no tiene segundo apellido.
2. La operación *devFechaNacimiento* deberá devolver una cadena con el formato AAMMDD, donde los dos primeros dígitos corresponden a los últimos dos dígitos del año de nacimiento, el tercer y cuarto dígito corresponden al mes de nacimiento, y los últimos dos dígitos corresponden al día de nacimiento.
3. La operación *devGenero* deberá devolver la cadena "H", si la persona es hombre y "M", si la persona es mujer.
4. La operación *devEntidadFederativa* deberá devolver la cadena de dos letras mayúsculas correspondientes al estado de residencia de la persona.
5. La operación *devConsonantes* deberá devolver la cadena de tres letras consonantes mayúsculas correspondientes a la primera consonante después de las primeras dos letras del primer apellido de la persona, seguida de la primer consonante después de la primera letra del segundo apellido de la persona, luego la primer consonante que sigue de la primera letra del nombre de la persona, respectivamente.

2. Implementación del TDA Persona.

NOMBRE DEL TIPO DE DATO: PersonaCURP que implementa *TDA Persona.*

ATRIBUTOS:

1. primerApellido : Cadena.
2. segundoApellido : Cadena.
3. nombre : Cadena.
4. fechaNacimiento : Cadena.
5. genero : Cadena.
6. entidadFederativa : Cadena.
7. numeroCuentaUNAM : "420095390".

DEFINICIÓN DE ALGORITMOS:

ALGORITMO 1.1 *devMayusculas*

Entrada: primerApellido : Cadena; segundoApellido : Cadena; nombre : Cadena; genero : Cadena; entidadFederativa : Cadena

Salida: primerApellido en mayúsculas; segundoApellido en mayúsculas; nombre en mayúsculas; genero en mayúsculsa; entidadFerderativa en mayúsculas

1. primerApellido = primerApellido.toUpperCase()
2. segundoApellido = segundoApellido != null ? segundoApellido.toUpperCase() : " ";
3. nombre = nombre.toUpperCase()

4. genero = genero.toUpperCase()
5. entidadFederativa = entidadFederativa.toUpperCase()

ALGORITMO 1.2 *devPL1Apellido*

Entrada: primerApellido : Cadena

Salida: Devuelve las primeras dos letras del primer apellido de la persona.

1. return primerApellido.substring(0 , 2)

ALGORITMO 1.3 *devPL2Apellido*

Entrada: segundoApellido : Cadena

Salida: Devuelve la primera letra del segundo apellido de la persona.

1. return segundoApellido.isEmpty() ? “X” : segundoApellido.substring(0 , 1)

ALGORITMO 1.4 *devPLNombre*

Entrada: nombre : Cadena

Salida: Devuelve la primera letra del nombre de la persona.

1. return nombre.substring(0 , 1)

ALGORITMO 1.5 *devFechaNacimiento*

Entrada: fechaNacimiento : Cadena

Salida: Devuelve la fecha de nacimiento de la persona en el formato AAMMDD.

1. Cadena[ ] partes = fechaNacimiento.split(“/”)
2. dia : cadena = partes[0]
3. mes : cadena = partes[1]
4. año = partes[2].substring(2)
5. return año + mes + dia

ALGORITMO 1.6 *devGenero*

Entrada: genero : Cadena

Salida: Devuelve “H” si el genero de la persona es hombre y “M” si es mujer.

1. if (genero.startsWith(“H”)) hacer:
2. return “H”
3. end if
4. if (genero.startsWith(“M”)) hacer:
5. return “M”
6. end if

ALGORITMO 1.7 *devEntidadFederativa*

Entrada: entidadFederativa : Cadena

Salida: Devuelve la cadena representativa de la entidad federativa.

1. switch (entidadFederativa)
2. case “AGUASCALIENTES”
3. return “AS”
4. case “BAJA CALIFORNIA”
5. return “BC”
6. case “BAJA CALIFORNIA SUR”
7. return “BS”
8. case “CAMPECHE”
9. return “CC”
10. case “COAHUILA”
11. return “CL”
12. case “COLIMA”
13. return “CM”
14. case “CHIAPAS”
15. return “CS”
16. case “CIUDAD DE MEXICO”
17. case “DISTRITO FEDERAL”

18. return “DF”
19. case “DURANGO”
20. return “DG”
21. case “GUANAJUATO”
22. return “GT”
23. case “GUERRERO”
24. return “GR”
25. case “HIDALGO”
26. return “HG”
27. case “JALISCO”
28. return “JC”
29. case “MEXICO”
30. return “MC”
31. case “MICHOACAN”
32. return “MN”
33. case “MORELOS”
34. return “MS”
35. case “NAYARIT”
36. return “NT”
37. case “NUEVO LEON”
38. return “NL”
39. case “OAXACA”
40. return “OC”
41. case “PUEBLA”
42. return “PL”
43. case “QUERETARO”
44. return “QT”
45. case “QUINATANA ROO”
46. return “QR”
47. case “SAN LUIS POTOSI”
48. return “SP”
49. case “SINALOA”
50. return “SL”
51. case “SONORA”

52. return “SR”
53. case “TABASCO”
54. return “TC”
55. case “TAMAULIPAS”
56. return “TS”
57. case “TLAXCALA”
58. return “TL”
59. case “VERACRUZ”
60. return “VZ”
61. case “YUCATAN”
62. return “YN”
63. case “ZACATECAS”
64. return “ZS”
65. case “NACIDO EN EL EXTRANJERO”
66. return “NE”
67. default
68. return “NE”

ALGORITMO 1.8 *devConsonantes*

Entrada: primerApellido : Cadena; segundoApellido : Cadena; nombre : Cadena

Salida: Devuelve la primera consonante después de las primeras dos letras del primer apellido, seguida de la primera consonante después de la primera letra del segundo apellido, y la primera consonante después de la primera letra del primer nombre.

1. cons1Apellido : Cadena = obtenerPrimeraConsonante(primerApellido , 2)
2. cons2Apellido : Cadena = segundoApellido.isEmpty() ? “X” : obtenerPrimeraConsonante(segundoApellido, 1)
3. consNombre : Cadena = obtenerPrimeraConsonante(nombre , 1)
4. return cons1Apellido + cons2Apellido + consNombre

ALGORITMO 1.9 *obtenerPrimeraConsonante*

Entrada: palabra : Cadena; indice : Entero

Salida: Devuelve la primera consonante de la palabra : Cadena.

1. vocales : Cadena = “AEIOU”
2. Para cada i con indice<= i <palabra.length hacer:
3. letra : char = palabra.ChatAt(i)
4. if (!vocales.contains (String.valueOf(letra) y Character.isLetter(letra)) hacer:
5. return String.valueOf(letra)
6. end if
7. return “X”

ALGORITMO 1.10 *devUltimosDigitos*

Entrada: numeroCuentaUNAM : Cadena

Salida: Devuelve los últimos dígitos de mi número de cuenta UNAM.

1. return numeroCuentaUNAM.substring(numeroCuentaUNAM.length() - 2)

ALGORITMO 1.11 *calcularCURP*

Entrada: primerApellido : Cadena; segundoApellido : Cadena; nombre : Cadena; fechaNacimiento : Cadena; genero : Cadena; entidadFederativa : Cadena; numeroCuentaUNAM : Cadena

Salida: Devuelve la CURP de la persona.

1. return devPL1Apellido + devPL2Apellido + devPLNombre + devFechaNacimiento + devGenero + devEntidadFederativa + devConsonantes + devUltimosDigitos

---

3. Definición formal del 1er Algoritmo.

Entrada: Un arreglo de objetos del tipo PersonaCURP.

Salida: Un arreglo que tenga como elementos las CURP´s de los objetos del arreglo de entrada.

1. if personaCURP == null then
2. return null
3. end if
4. Sea CURPs un arreglo de tamaño personaCURP.lenght
5. Para cada i con 0<=i<CURPs.length hacer:
6. CURPs[i] = personaCURP[i].calcularCURP()
7. return CURPs

4. Cálculo de operaciones elementales y complejidad del algoritmo *calcularCURP*.

Entrada: Una cadena de la representación del primer apellido de la persona, una cadena de la representación del segundo apellido de la persona, una cadena de la representación del primer nombre de la persona, una cadena de la representación de la fecha de nacimiento de la persona, una cadena de la representación del género de la persona, una cadena de la representación de la entidad federativa donde vive la persona, una cadena de la representación de las consonantes del primer y segundo apellido y el nombre, una cadena de la representación de mi número de cuenta UNAM.

Salida: Una cadena que sea el resultado de la concatenación de las cadenas de entrada.

1. return *devPL1Apellido* + *devPL2Apellido* + *devPLNombre* + *devFechaNacimiento* + *devGenero* + *devEntidadFederativa* + *devConsonantes* + *devUltimosDigitos*

Número de operaciones elementales. $T(n) = 16$. Su complejidad es $O(1)$.

1. 1 operación (return) + 8 operaciones (Invocación de los métodos) + 7 operaciones (Concatenaciones) = 16 operaciones

5. Cálculo de operaciones elementales y complejidad del algoritmo de la sección 3.

Número de operaciones elementales. $T(n) = 5n + 6$. Su complejidad es $O(n)$.

1. 1 operación
2. 1 operación
4. 2 operaciones
5. 1 operación + 3 operaciones por iteración
6. 2 operaciones por iteración
7. 1 operación

Proposición (Complejidad del algoritmo de la sección 3)**.** *Sea* $c = 11$ *y* $k = 1$. *Demostraremos que:*

$$5n + 6 \leq 11n, \quad \forall n \in \mathbb{N}, n \geq 1$$

Demostración. (Inducción)

Base: $n = 1$

$$5(1) + 6 = 11 \leq 11 = 11(1)$$

Por lo tanto, la desigualdad se cumple para $n = 1$.

Hipótesis inductiva: Supongamos que para $n = k$ se cumple

$$5k + 6 \leq 11k$$

Paso inductivo: Demostraremos que se cumple para $n = k + 1$:

$$5(k + 1) + 6 = 5k + 5 + 6 = 5k + 11$$

Por hipótesis inductiva, $5k + 6 \leq 11k$, y además $5 \leq 11$, por lo que

$$5k + 11 \leq 11k + 11 = 11(k + 1)$$

Luego:

$$5(k+1)+6 \leq 11(k+1)$$

$$\therefore 5n+6 \leq 11n, \quad \forall n \in \mathbb{N}, n \geq 1$$

□

6. Definición formal, cálculo de operaciones elementales y demostración de la complejidad del 2do algoritmo.

Entrada: Un arreglo A que tiene como elementos números reales.

Salida: Un arreglo B del tamaño del arreglo de la entrada con elementos $(\lfloor A[i] \rfloor, y)$ con $y = A[i] - \lfloor A[i] \rfloor$

1. Sea B un arreglo de tamaño A.length
2. Para cada i con 0<=i<B.length hacer:
3. entero = A[i].piso()
4. fraccion = A[i] - entero
5. B[i] = (entero, fraccion)
6. return B

Número de operaciones elementales. $T(n) = 13n + 4$. Su complejidad es $O(n)$.

1. 2 operaciones
2. 1 operación + 3 operaciones por iteración
3. 4 operaciones por iteración
4. 4 operaciones por iteración
5. 2 operaciones por iteración
6. 1 operación

Proposición (Complejidad del algoritmo de la sección 6)**.** *Sea $c = 17$ y $k = 1$. Demostraremos que:*

$$13n + 4 \leq 17n, \quad \forall n \in \mathbb{N}, n \geq 1$$

Demostración. (Inducción)

Base: $n = 1$

$$13(1) + 4 = 17 \leq 17 = 17(1)$$

Por lo tanto, la desigualdad se cumple para $n = 1$.

Hipótesis inductiva: Supongamos que para $n = k$ se cumple

$$13k + 4 \leq 17k$$

Paso inductivo: Demostraremos que se cumple para $n = k + 1$:

$$13(k+1) + 4 = 13k + 13 + 4 = 13k + 17$$

Por hipótesis inductiva, $13k + 4 \leq 17k$, y además $13 \leq 17$, por lo que

$$13k + 17 \leq 17k + 17 = 17(k + 1)$$

Luego:

$$13(k + 1) + 4 \leq 17(k + 1)$$

$$\therefore 13n + 4 \leq 17n, \quad \forall n \in \mathbb{N}, n \geq 1$$

□

7. Determina cuales de las siguientes afirmaciones son verdaderas, si son falsas da un contrajemplo, y si son verdaderas justifica por qué lo son:

a. Todo problema tratable es decidible.

- Verdadera. Si un problema es tratable, existe un algoritmo que lo resuelve en tiempo polinómico. Todo algoritmo que termina en tiempo polinómico siempre termina, por lo que el problema es decidible.

b. Todo problema indicidible es intratable.

- Verdadera. Un problema indecidible no tiene ningún algoritmo que lo resuelva. Si no hay algoritmo , no puede resolverse en tiempo polinómico ni en ningún tiempo finito. Por definición, es intratable.

c. Todo problema intratable es indecidible.

- Falsa. El problema del viaje del vendedor con peso máximo es intratable, pero es decidible, porque se puede enumerar todas las soluciones posibles y decidir si existe una que cumpla la condición.

d. Todo problema decidible es tratable.

- Falsa. Problemas del tipo de satisfiabilidad booleana son decidibles, pero no se conoce ningún algoritmo polinómico, por lo que son decidibles pero intratables.

8. Invariantes fuertes y débiles para los algoritmos definidos en las secciones 3 y 6.

Algoritmo de la sección 3.

- Antes de la primera iteración (i = 0), no hay elementos procesados, lo que significa condición vacía, consistente.
- Al inicio de cada iteración i, para todo j con $0 \leq$ j <i, se tiene: CURPs[j] = personaCURP[j].calcularCURP().

Algoritmo de la sección 6.

- Antes de la primera iteración (i = 0), no hay elementos procesados, lo que significa condición vacía, consistente.
- Al inicio de cada iteración i, para todo j con $0 \leq$ j <i, se cumple: $B[j] = (\lfloor A[j] \rfloor, A[j] - \lfloor A[j] \rfloor)$