INF152 Estructuras Discretas - Tarea 2

Profesores: Claudio Lobos, Jorge Díaz Sebastián Gallardo, María Paz Vergara y Miguel Guevara.
Ayudantes: Valentina Aróstica, Bryan González, Sofía Mañana, Sofía Riquelme, Carla Herrera, Álvaro Gaete, Constanza Alvarado, Maciel Ripetti, Maureen Gavilán, Ignacio Jorquera, Fernanda Avendaño y Luis González.
Universidad Técnica Federico Santa María
Departamento de Informática – SJ - CC

Tarea 2

1. Reglas generales

- Para resolver cada pregunta, debe hacer uso de los contenidos, algoritmos y métodos aprendidos en el curso. Si su respuesta final es correcta, pero se ha utilizado un método distinto al enseñado en clases no se asignará puntaje. Ídem si entrega resultados sin desarrollo.
- La tarea debe realizarse en grupos de hasta 3 integrantes **DEL MISMO PARALELO**. Esta vez no habrán excepciones.
- Se descontará 5 puntos por warnings en el `.tex` (Excepto los warnings azules de Overleaf *Underfull \hbox (badness 10000)*, estos no tendrán descuentos).
- Si su proyecto `.tex` **no** compila tendrá 100 puntos de descuento sin opción a recorrección, inclusive si existe un `.pdf` con el desarrollo de la tarea.
- Tiene hasta las 23:59 hrs del día **9 de mayo** para entregar esta tarea vía Aula.
- No inserte su desarrollo en fotografías. Debe estar todo desarrollado en LaTeX.
- Se restarán 10 puntos de su nota sucesivamente por cada hora de atraso.

1.1. Entrega

- Sólo un integrante del grupo debe realizar la entrega.
- Se debe entregar un solo archivo `.zip` el cual debe contener:
  1. El o los archivos `.tex` para compilar su tarea.
  2. Las imágenes adjuntas que permiten compilar el enunciado
  3. Un archivo `README.txt` que contenga los nombres de los integrantes del grupo, y sus roles.
- El archivo `.zip` debe tener como nombre: `nombre_apellido.zip`, donde el nombre es el de el estudiante que hace la entrega.
- No es necesario incluir el `.pdf` en el `.zip`

2. Aprendiendo Pseudocodigo en latex

1. Resuelva el siguiente desafío programando en pseudocódigo. La solución debe escribirse en LaTeX, usando los paquetes *algorithmic* y *algorithm* (los cuales ya están inluidos en este `.tex`).

   Se da un String *line* conteniendo una lista de enteros, separados por uno o más espacios. Dado un int givenNumber, retornar el menor entero en line que sea estrictamente mayor que givenNumber. Retornar -1 si no existe tal número en line. [25 pts].

   Método:

   int getLeast(String line, int givenNumber)

   Ejemplos:

   a) `line = "1 2 3 4 5", givenNumber = 2`
   Retorna: 3
   3 es el número más pequeño estrictamente mayor que 2.

   b) `line = "120 450 780", givenNumber = 1000`
   Retorna: -1
   No hay números estrictamente mayores que 1000.

   c) `line = "45 253 645 400 676 567", givenNumber = 1`
   `Retorna: 45`

   d) `line = "45 253 645 400 676 567", givenNumber = 400`
   Retorna: 567

   e) `line = "568 769 436 432 457 563 567 311 34 3 2 9", givenNumber = 460`
   Retorna: 563

   Aquí tiene un pseudocódigo de ejemplo, que puede utilizar como plantilla para su solución:

Algorithm 1 Titulo del algoritmo

```
1: Method String nombreFuncion(String text)
2: Empiece a programar aquí
3: hola mundo
4: string variable ← "hola"
5: for i in lista do
6:     haz algo
7: end for
8: while True do
9:     print  Hola
10: end while
```

   Solución:

Algorithm 2 Find the minimum value larger than the parameter

```
1: Method int getLeast(String line, int givenNumber)
2: int x ← 0
3: int y ← 0
4: int list ← list()
5: int largo ← len(line)
6: for item in line do
7:    if item equals ' ' and y > 0 then
8:       list.append(stringToInt(line[x:y]))
9:       x ← y + 1
10:   end if
11:   if  y equals largo -1 then
12:      list.append(stringToInt(line[x:y+1]))
13:   end if
14:   y ← y + 1
15: end for
16: int maxValue ← 0
17: for element1 in list do
18:   if element1 > maxValue then
19:      maxValue ← element1
20:   end if
21: end for
22: if maxValue ≤ givenNumber then
23:   return -1
24: end if
25: for element2 in list do
26:   if element2 > givenNumber and element2 < maxValue  then
27:      maxValue ← element2
28:   end if
29: end for
30: return maxValue
31: // list() puede ser reemplazado por lista[], pero debe ser una estructura iterable
32: // y se asume la existencia de un metodo append() o parecido que agrege elementos a lista
33: // se asume la existencia de una funcion stringToInt() que transforme strings a enteros
```

3. Preguntas de Grafos y Árboles

1. Considere los siguientes grafos, dos de ellos representados en tablas de adyacencia, y conteste las preguntas relacionadas.

	n	m	l	o	p	q
n	0	0	1	1	1	0
m	0	0	1	0	0	0
l	1	1	0	1	0	0
o	1	0	1	0	1	0
p	1	0	0	1	0	1
q	0	0	0	0	1	0

Tabla 1: Matriz de Adyacencia de Grafo 1

	a	b	c	d	e	f	g	h
a	0	1	0	0	1	0	0	0
b	1	0	1	1	0	0	1	0
c	0	1	0	1	0	0	0	0
d	0	1	1	0	1	0	0	1
e	1	0	0	1	0	1	1	0
f	0	0	0	0	1	0	1	0
g	0	1	0	0	1	1	0	1
h	0	0	0	1	0	0	1	0

Tabla 2: Matriz de Adyacencia de Grafo 2

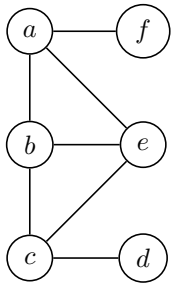


Figura 1: Grafo 3.

a) Utilizando el paquete Tikz de LaTeX, dibuje los Grafos 1 y 2 expresados como matriz de adyacencia en las Tablas 1 y 2. No se aceptarán imágenes *(El package ya está incluido en el header)* [10 pts].
Solución:

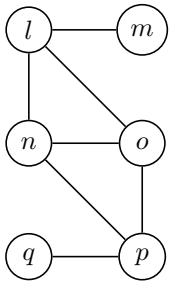


Figura 2: Grafo 1.

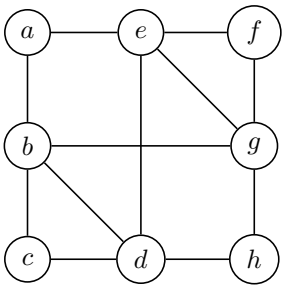


Figura 3: Grafo 2.

b) Demuestre o refute la existencia de un circuito de Euler en el grafo 2. ¿Existe Camino de Euler? En caso de que exista, nombre uno. [15 pts].
Solución:
El camino de Euler tiene una condición que lo define, la cual es $\{\exists\ 2\ v \in V \mid \delta(v)$ es impar$\}$, lo que en el caso del grafo 2 no se cumple pues todos sus nodos son de grado par, lo que da lugar a la definición del Circuito de Euler.

c) Demuestre o refute: Grafo **1** y Grafo **3** son isomorfos [20 pts]
Solución:
Podemos expresar ambos grafos tal que $G_1 = \{V_1, E_1\}$ y $G_3 = \{V_3, E_3\}$, entonces uno de los requisitos para que sean isomorfos es que el grado de V sea igual, entonces:

$$|V_1| = 6 \quad ; \quad |V_3| = 6$$

Una vez sabemos que son grafos con la misma cantidad de nodos, debemos definir la función biyectiva que relacione estos nodos según su grado.

$$\psi(m) = d,\ \psi(l) = c,\ \psi(n) = b,\ \psi(o) = e,\ \psi(p) = a,\ \psi(q) = f$$

y como $\psi()$ debe ser biyectiva, tambien hay que definir su inversa:

$$\psi^{-1}(d) = m,\ \psi^{-1}(c) = l,\ \psi^{-1}(b) = n,\ \psi^{-1}(e) = o,\ \psi^{-1}(a) = p,\ \psi^{-1}(f) = q$$

Y así la función $\psi()$ define el isomorfismo entre los grafos 1 y 3.

2. Los sistemas operativos de las computadoras modernas organizan las carpetas y los archivos usando una estructura de árbol. Una carpeta contiene otras carpetas y archivos. La figura 4 muestra el explorador de Windows con el despliegue de carpetas. La raíz se llama INF152.

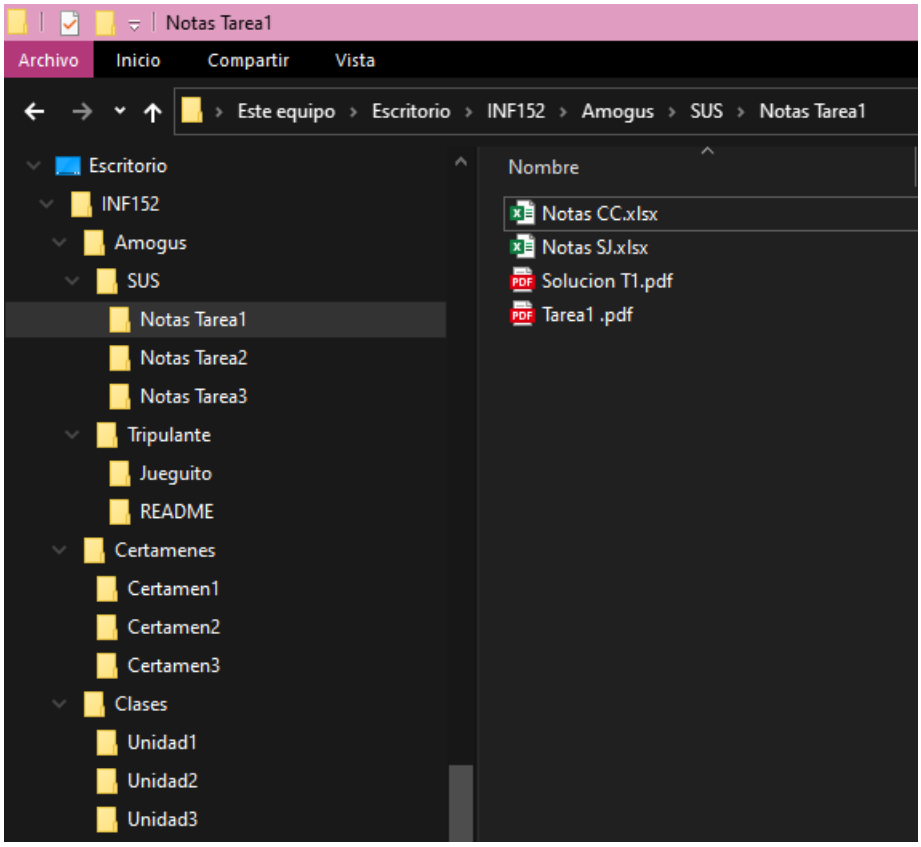
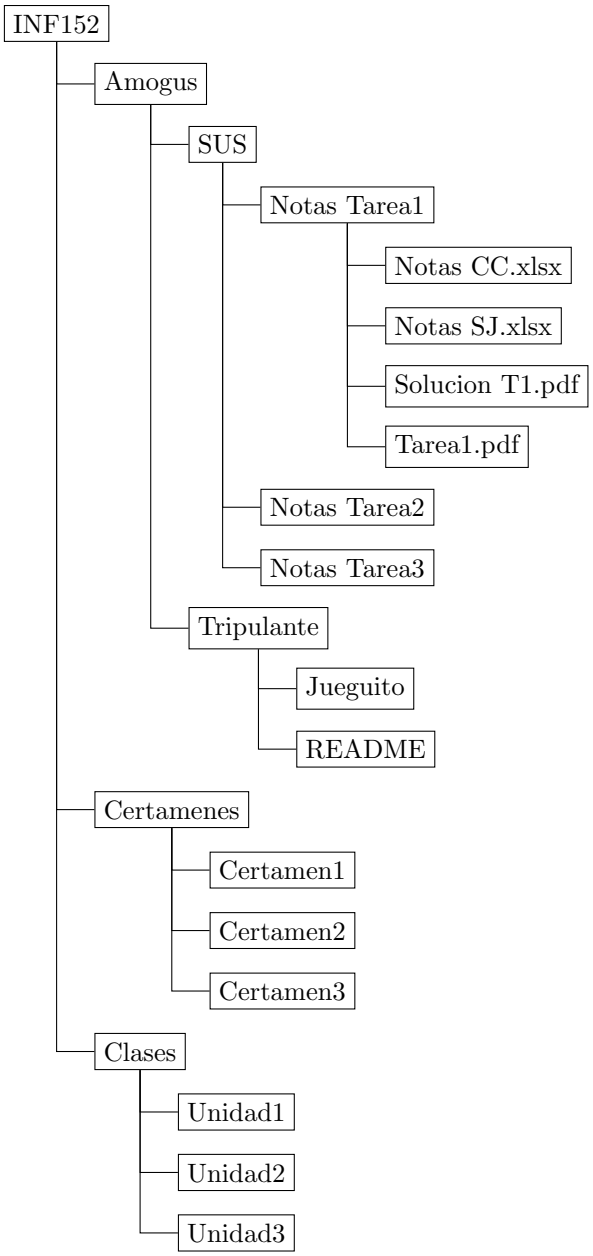


Figura 4: Despliegue de explorador de archivos de Windows

a) Dibuje utilizando LaTeX el árbol que representa las carpetas de la Figura 4 [5 pts].
Solución:



b) Utilice los algoritmos de búsqueda en árboles para encontrar la solución del certamen 1 de la carpeta **Certamen1** y luego, realice los mismos pasos para encontrar un archivo dentro de la carpeta **Notas Tarea1**. ¿Puede determinar cuál búsqueda es más rápida en cada caso? Justifique. *(Puede dibujar con Tikz el paso a paso en los grafos y marcar con colores las rutas de búsqueda. Otra forma es explicar con pseudocódigo o puede escribir un ruteo de los algoritmos. Lo importante es que explique formalmente cómo ejecuta los algoritmos.)* [25 pts].

Solución:
Utilizando Busqueda en Profundidad:

Caso 1: INF152 → Clases → Unidad2 → Unidad3 → Unidad1 → Certamenes → Certamen3 → Certamen1;

En este caso se parte desde la raíz y se visita el nodo 'Clases', luego se visitan las hojas Unidad1, Unidad2 y Unidad3, luego visita el vecino de 'Clases', 'Certamenes' y visita las hojas de este, donde encuentra Certamen3 y luego Certamen1.

Caso 2: INF152 → Clases → Unidad2 → Unidad3 → Unidad1 → Certamenes → Certamen3 → Certamen1 → Certamen2 → Amogus → Tripulante → Jueguito → README → SUS → Notas Tarea2 → Notas Tarea1→ Notas SJ.xlsx;

Siguiendo desde el caso 1, se visita la hoja Certamen2 que faltó y luego se visita el ultimo vecino de 'Clases' y 'Certamenes', que es 'Amogus', luego se visita uno de los hijos de este nodo, 'Tripulante' y se visitan sus hojas, Jueguito y README, después se visita el nodo vecino de 'Tripulante', el nodo 'SUS', donde se visita el nodo 'Notas Tarea2' y su vecino 'Notas Tarea1' con su hoja 'Notas SJ.xlsx'

Notese que la busquedas **INF152 → Certamenes → Certamen1;** y **INF152 → Certamenes → Certamen1 → Certamen2 → Certamen3 → Amogus → SUS → Notas Tarea1 → Solucion T1.pdf;** tambien son soluciones validas ya que el algoritmo de Busqueda en profundidad toma un nodo al azar y este puede ser justamente el camino más directo.

Utilizando Busqueda a lo Ancho:

Caso 1: INF152 → Amogus → Certamenes → Clases → SUS → Tripulante → Certamen1;

En este caso se visitan primero todos los hijos de la raiz, 'Amogus', 'Certamenes' y 'Clases', luego se visitan los hijos de 'Amogus', 'SUS' y 'Tripulante', despues los hijos de 'Certamenes', donde se encuentra a Certamen1.

Caso 2: INF152 → Amogus → Certamenes → Clases → SUS → Tripulante → Certamen1 → Certamen2 → Certamen3 → Unidad1 → Unidad2 → Unidad3 → Notas Tarea1 → Notas Tarea2 → Notas Tarea3 → Jueguito → README → Notas SJ.xlsx;

Siguiendo desde el caso 1, se siguen visitando los hijos de 'Certamenes', 'Certamen2' y 'Certamen3', para luego seguir la busqueda visitando los hijos del nodo 'Clases', los nodos 'Unidad1', 'Unidad2' y 'Unidad3', después se visitan los hijos del nodo 'SUS', los nodos 'Notas Tarea1', 'Notas Tarea2' y 'Notas Tarea3',luego los hijos del nodo 'Tripulantes', que son 'Jueguito' y 'README', para luego visitar los hijos de 'Notas Tarea1' y se encuentra la hoja 'Notas SJ.xlsx'.

La busqueda más rápida para el caso 1 seria la de búsqueda a lo ancho, y la más rápida para el caso 2 seria la de búsqueda en profundidad