**Ejercicio 1**

**Inciso 1:**

Partiendo de los conceptos vistos en clase, sabemos que una arista se puede representar como un par ordenado de vértices de un grafo. Con esto en mente, considero que la estructura más sencilla para el archivo que contiene la data es aquella en la cual cada línea del archivo representa un par ordenado de vértices. De esta manera, tendríamos la información de las conexiones entre cada vértice en el grafo. En caso de que haya un vértice sin conexiones, aparecerá como un solo elemento en el archivo. Un ejemplo de este formato utilizando un grafo de 4 vértices es:

**Inciso 2:**

1, 3

4, 1

1, 2

3, 2

Con esta estructura se cumplen los 3 requisitos solicitados. Cualquier humano que lea el archivo, asumiendo que conoce el contexto de la situación, puede entender fácilmente que, por ejemplo, los vértices 1, 3 están conectados. Cuando se expresa la asunción de que la persona conoce el contexto se quiere decir que dicho individuo está consciente de que se está lidiando con grafos. Adicionalmente, es un archivo fácil de editar en el caso de que se requiera modificar una conexión o se quiera agregar una nueva; solo habría que escribir un nuevo par ordenado o modificar alguno existente. Sin embargo, entiendo que se puede hacer un poco tedioso si el grafo es muy grande ya que el archivo sería muy extenso. Finalmente, un programa puede cargar el *file* fácilmente tan solo leyendo cada línea y agregando los vértices que conforman el par ordenado.

**Inciso 3: Algoritmo para cargar el archivo.**

𝑬𝒏𝒕𝒓𝒂𝒅𝒂: 𝑎𝑟𝑐ℎ𝑖𝑣𝑜 𝐹 𝑞𝑢𝑒 𝑐𝑜𝑛𝑡𝑖𝑒𝑛𝑒 𝑙𝑎 𝑑𝑎𝑡𝑎 𝑑𝑒𝑙 𝑔𝑟𝑎𝑓𝑜.

𝑺𝒂𝒍𝒊𝒅𝒂: 𝑒𝑠𝑡𝑟𝑢𝑐𝑡𝑢𝑟𝑎 𝑟𝑒𝑝𝑟𝑒𝑠𝑒𝑛𝑡𝑎𝑡𝑖𝑣𝑎 𝑑𝑒 𝑢𝑛 𝑔𝑟𝑎𝑓𝑜 𝐺. 1. 𝒉𝒂𝒄𝒆𝒓 𝒎𝒊𝒆𝒏𝒕𝒓𝒂𝒔 𝑛𝑜 𝑠𝑒𝑎 𝑒𝑙 𝑓𝑖𝑛𝑎𝑙 𝑑𝑒𝑙 𝑎𝑟𝑐ℎ𝑖𝑣𝑜

2. 𝑒𝑑𝑔𝑒 ← 𝑟𝑒𝑎𝑑𝐿𝑖𝑛𝑒( )

3. 𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[ ] ← 𝑒𝑑𝑔𝑒. 𝑠𝑝𝑙𝑖𝑡𝐵𝑦𝐶𝑜𝑚𝑚𝑎𝑠( )

4. 𝒔𝒊 𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠. 𝑠𝑖𝑧𝑒( ) < 2 𝒆𝒏𝒕𝒐𝒏𝒄𝒆𝒔 𝒉𝒂𝒄𝒆𝒓

5. 𝒔𝒊 𝐺. 𝑒𝑥𝑖𝑠𝑡𝑠(𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[0]) 𝑒𝑠 𝑓𝑎𝑙𝑠𝑜 𝒆𝒏𝒕𝒐𝒏𝒄𝒆𝒔 𝒉𝒂𝒄𝒆𝒓

6. 𝐺. 𝑎𝑑𝑑𝑉𝑒𝑟𝑡𝑒𝑥(𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[0])

7. 𝒇𝒊𝒏 𝒔𝒊

8. 𝒔𝒊 𝒏𝒐 𝒉𝒂𝒄𝒆𝒓

9. 𝒔𝒊 𝐺. 𝑒𝑥𝑖𝑠𝑡𝑠(𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[0]) 𝑒𝑠 𝑓𝑎𝑙𝑠𝑜 𝒆𝒏𝒕𝒐𝒏𝒄𝒆𝒔 𝒉𝒂𝒄𝒆𝒓

10. 𝐺. 𝑎𝑑𝑑𝑉𝑒𝑟𝑡𝑒𝑥(𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[0])

11. 𝒇𝒊𝒏 𝒔𝒊

12. 𝒔𝒊 𝐺. 𝑒𝑥𝑖𝑠𝑡𝑠(𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[1]) 𝑒𝑠 𝑓𝑎𝑙𝑠𝑜 𝒆𝒏𝒕𝒐𝒏𝒄𝒆𝒔 𝒉𝒂𝒄𝒆𝒓

13. 𝐺. 𝑎𝑑𝑑𝑉𝑒𝑟𝑡𝑒𝑥(𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[1])

14. 𝒇𝒊𝒏 𝒔𝒊

12. 𝐺. 𝑎𝑑𝑑𝐸𝑑𝑔𝑒(𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[0], 𝑣𝑒𝑟𝑡𝑖𝑐𝑒𝑠[1])

13. 𝒇𝒊𝒏 𝒎𝒊𝒆𝒏𝒕𝒓𝒂𝒔

14. 𝒓𝒆𝒕𝒐𝒓𝒏𝒂 𝐺

**Inciso 4.**

Tomando en cuenta los datos que recibimos, tendríamos que:

# 𝑇(𝑚, 𝑛, 𝑎, 𝑣) = (𝑚 ∗ 𝑣) + (𝑛 ∗ 𝑎)