**DIVISION DE TAREAS**

“La batalla contra reloj”

Grupo “RedBull”

Integrantes:

Alejandro Del Carpio Sanchez - 99655

Julián Montenegro - 109284

Ian Anca – 108924

**TABLA DE CONTENIDO**

1. Grafos

1.1 Algoritmo de Prim

1. Tabla de Hashing
2. Desarrollo de las Listas *Readings* y *Authors*
3. Volcado de información de Archivos a Listas
4. **Grafos**

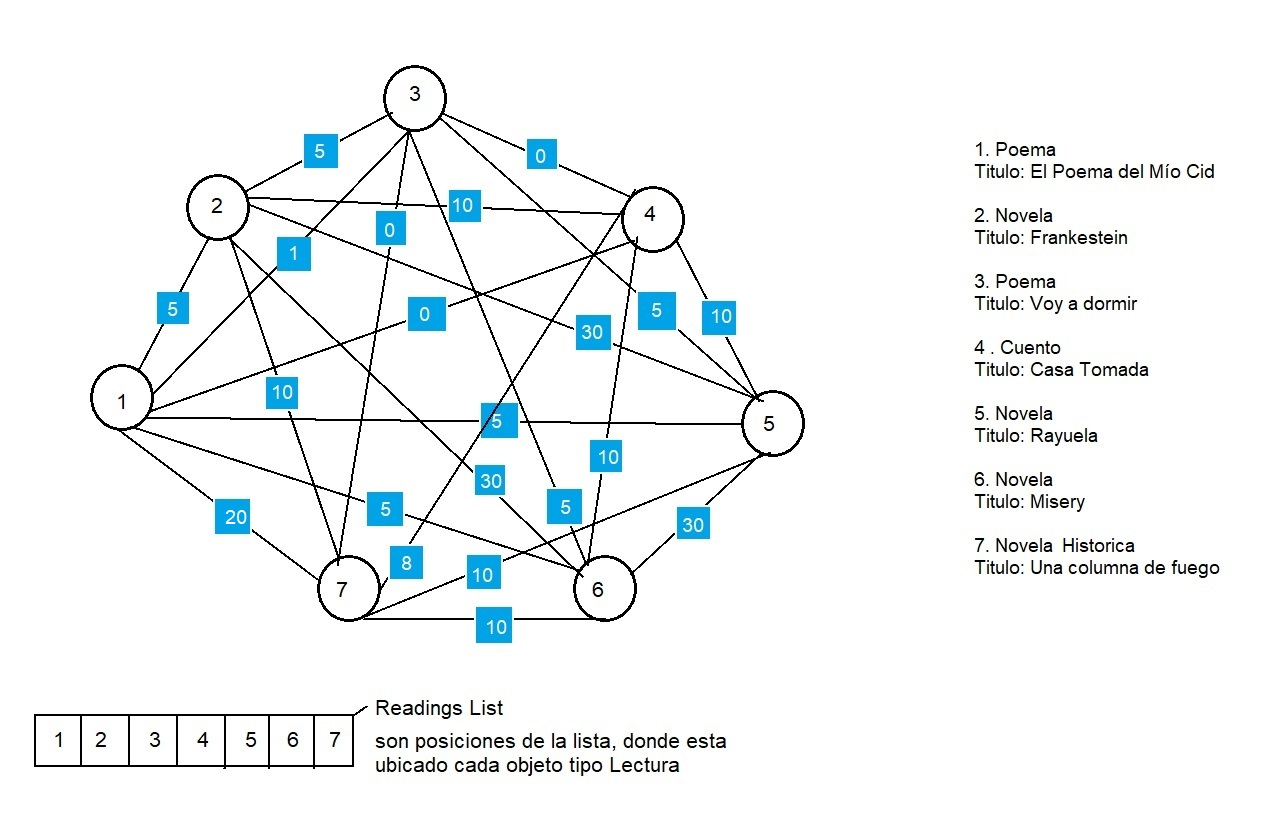
Para la representación del grafo, se implemento las clases Vertex, Graph y MST (Minimun Spanning Tree).

Para encontrar el árbol de expansión se implemento el algoritmo de Prim ya que las aristas iban a ser superiores a la cantidad de nodos, vértices o en este caso las lecturas, esto por enunciado ya que cada lectura iba a estar “conectada” a otra.

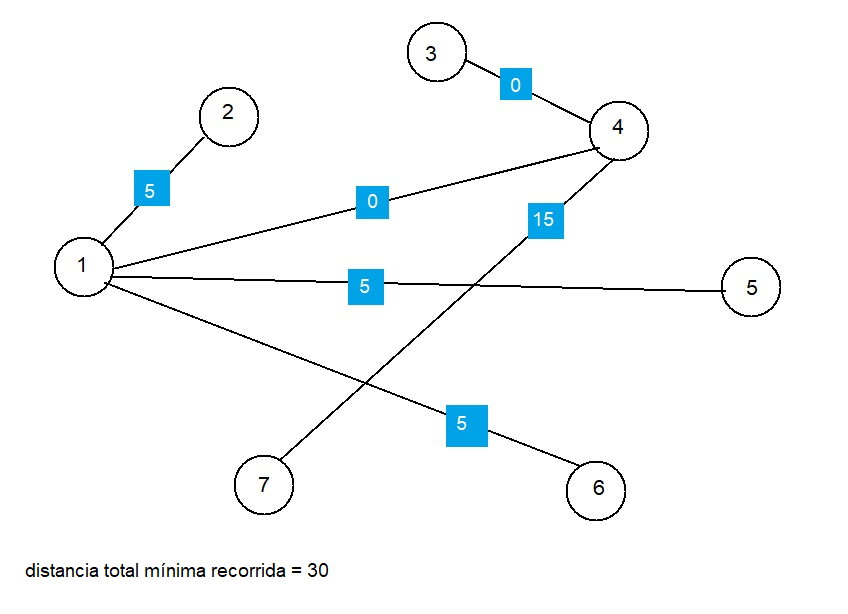
* 1. **Algoritmo de Prim**

Este algoritmo a diferencia de Kruskal va insertando los nodos y encontrando las “conexiones” evitando formar caminos, para lograr esto, se creo 3 vectores dinámicos del cual su tamaño va a depender de la cantidad de nodos que exista en el grafo, estos son : parent, visited y weigth, además de estos vectores se generó desde Graph.h/cpp la matriz de adyacencia para encontrar si ese nodo fue ya visitado o tiene una “conexión” con otro y encontrando además el peso menor.

Una cosa que aclarar es que al primer nodo, es decir a la primera posición de parent se le asigna un -1 para identificar cual es el primer nodo.

Ejemplo de lecturas.txt , considerando lo pedido en el enunciado.   
  
Grafo

Árbol de expansión mínima



Matriz de adyacencia

1. **Hashing**

**2.1 Orden de alta: O(1)**

Todo nuevo elemento va a ser añadido al final de la lista, irrelevantemente del tamaño de la misma.

Por lo que no se puede definir claramente un mejor o un peor caso en particular.

**2.2 Orden de consulta: O(n)**

Considerando n como la cantidad de objetos en la lista analizada, se puede definir los siguientes casos:

* **Mejor caso**

Si se consulta por un elemento que se encuentra en una lista de tamaño 1. (Ninguna colisión con esa clave)

Si se consulta por el primer elemento de una lista (Irrelevantemente del tamaño de la lista)

* **Mejor caso (en términos generales)**

Si se consulta por un elemento cuya clave no colisiono al ser añadido a tabla.

* **Peor caso**

Si se consulta un elemento que se encuentra al final de la lista con más elementos de toda la tabla.

(La tabla tiene la mayor cantidad de colisiones en esa posición)

* **Caso promedio**

Varía según la cantidad de elementos que colisionado cuando se añadió el elemento a la tabla.

(A menor colisiones cuando se lo añadió, más rápido será la consulta)

* 1. **Orden de baja: O(n)**

Considerando n como la cantidad de objetos en la lista analizada, se puede definir los siguientes casos:

* **Mejor caso (en términos generales)**

Si se quiere eliminar un elemento cuya clave no colisiono al ser añadido a tabla.

* **Peor caso**

Si se quiere eliminar un elemento que se encuentra al final de la lista con más elementos de toda la tabla.

(La tabla tiene la mayor cantidad de colisiones en esa posición).

* **Caso promedio**

Varía según la cantidad de elementos que colisionado cuando se añadió el elemento a la tabla.

(A menor colisiones cuando se lo añadió, más rápido será eliminar el elemento)