Universidad Nacional de General Sarmiento

Programación III

Trabajo práctico 2

Autores:

Elías Espinillo 40.663.777

Federico Farias 36.495.959

Profesores:

Javier Marenco

Patricia Bagnes

Contenidos

1 Introducción

El objetivo del trabajo práctico es implementar el algoritmo de Kruskal con BFS y con la estructura de datos Union-Find y medir los tiempos que toma cada una de estas formas de crear un árbol generador mínimo desde un grafo aleatorio el cual se pasa como parámetros la cantidad de vértices y la cantidad de aristas (siempre respetando la cantidad máxima de aristas con la formula $\frac{n(n-1)}{2}$ con n como la cantidad de vértices)

2 Diseño, especificación e implementación

Se planteó la creación de 2 packages diferentes, en uno se guardan las clases en donde implementaremos la creación de grafos, aristas, grafos aleatorios, BFS, la estructura de datos Union-Find, el algoritmo de Kruskal y por último la medición de tiempos de cada una de las formas del algoritmo de Kruskal sobre grafos aleatorios.
En el 2do package se definen todos los tests unitarios necesarios de las clases para verificar el buen funcionamiento de estas.

Para poder obtener un árbol generador mínimo (mas adelante AGM) el grafo tiene que ser no dirigido y las aristas poseen peso, por lo cual decidimos implementar la clase Grafo con una lista de vecinos y a la clase Arista separada para poder ir creándolas con el peso correspondiente entre los vértices pasados como parámetros.

- **Paquete grafos_and_algoritmos:**
  - **Clase Grafo:**
    - vecinos: Lista de conjuntos de Integers
    - aristas: Lista de Aristas

    Operaciones:
    - public Grafo(int vertices):

      Constructor de la clase en donde se pasa la cantidad de vértices deseada.
    - Getters y setters de aristas y vecinos.
    - public void agregarArista(int s, int t, int peso):

      Agrega aristas con un peso entero desde un vértice inicial al vértice destino.
    - public void agregarArista(int s, int t, int peso):

      Agrega aristas con un peso con punto flotante entre 0 y 1 desde un vértice inicial al vértice destino.
    - public void eliminarArista(int s, int t):

      Elimina la arista que se encuentra desde un vértice inicial al vértice destino.

- public boolean existeArista (int s, int t):

  Verifica si existe la arista entre un vértice inicial al vértice destino.

- public int pesoTotal():

  Indica el valor del peso de todas las aristas del grafo.

- public int grado (int s):

  Devuelve el valor del grado del vértice.

- public int tamanio ():

  Devuelve el tamaño del grafo.

- private void verificarDistintos(int s, int t):

  Verifica si 2 vertices son distintos entre sí.

- private void verificarVertice(int s, Grafo grafo):

  Verifica si el vértice es válido (sea positivo) y que no supere al tamaño del grafo.

- **Clase Arista:**
  - vertice_S, vertice_T y peso: Integer

  Operaciones:

  - public Arista(int vertice_S, int vertice_T, float peso):

    Constructor de la clase donde se pasan como parámetros el vértice inicial, el vértice destino y el peso que tendrá la arista.

  - Getters y setters de vertices y peso.
  - public boolean equals(Object obj):

    Método para determinar si 2 aristas son iguales.

- **Clase BFS:**
  - L: Lista de Integers
  - marcados: Boolean

Operaciones:

- public Set<Integer> alcanzables(Grafo g, int origen):

  Busca desde el vértice tomado como origen y retorna un conjunto de vértices a los cuales alcanza.
- public boolean esConexo(Grafo g):

  Verifica si el grafo es conexo.
- public boolean circuito(Grafo g, int origen, int destino):

  Busca si 2 vertices hacen circuito.
- private void agregarVecinosPendientes(Grafo g, int i):

  Método que utiliza alcanzables() para agregar a los vecinos que faltan.
- private void inicializar(Grafo g, int origen):

  Inicializa el grafo.

- **Clase Kruskal:**

Operaciones:

- public Grafo kruskalBFS(Grafo g):

  Crea un árbol generador mínimo usando el algoritmo de Kruskal con BFS.
- public Grafo kruskalUF (Grafo g):

  Crea un árbol generador mínimo usando el algoritmo de Kruskal con estructura de datos Union-Find.
- private Arista seleccionarMenorBFS(ArrayList<Arista> aristaList, Grafo g):

  Método que utiliza KruskalBFS para seleccionar la arista de menor peso.
- private Arista seleccionarMenorUF(ArrayList<Arista> aristaList):

  Método que utiliza elegirArista para seleccionar la arista de menor peso.
- private Arista elegirArista(ArrayList<Arista> aristaList, UnionFind uf):

  Método que utiliza KruskalUF para elegir la arista y comenzar con el algoritmo.

- **Clase UnionFind:**
  - A: Array de Integers

  Operaciones:

  public UnionFind(int N):

  Constructor de la estructura de datos Union-Find.

  - public int root (int i):

    Raíz del vértice.

  - public boolean find(int i, int j):

    Verifica si 2 vertices están en la misma componente conexa.

  - public void union(int i, int j):

    Union de 2 componentes conexas.

  - public boolean esConexo():

    Verifica si es conexo.

- **Clase GrafoAleatorio:**

  Operaciones:

  - public Grafo nuevoGrafoAleatorio(int vertices, int aristas):

    Crea un grafo aleatorio con vertices y aristas ingresadas como parámetros. Se le indica al usuario con excepciones si se ingresan vértices o aristas negativas y también por teoría de grafos las aristas no pueden superar la cantidad máxima según la formula $\frac{n(n-1)}{2}$ con n como la cantidad de vértices.

- **Clase Principal:**

  Operaciones:

  - public static long medicionDeTiempoKruskalBFS():

    Mide el tiempo que tarda el algoritmo de Kruskal con BFS.

  - public static long medicionDeTiempoKruskalUF():

    Mide el tiempo que tarda el algoritmo de Kruskal con la estructura de datos Union-Find.

  - public static void main (String[]args):

    Main donde por medio de un StringBuilder se muestra por consola al usuario el resultado de los tiempos que se tomaron con el algoritmo de Kruskal con las diferentes formas, BFS y con la estructura de datos Union-Find.

3 Muestra de tiempos del algoritmo de Kruskal

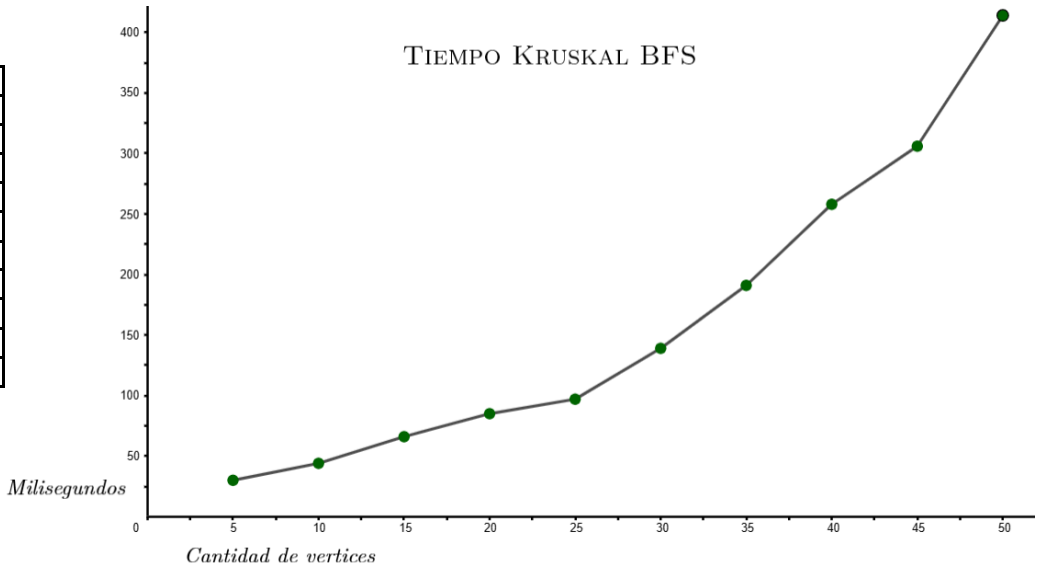
Para poder realizar las muestras de lo que tardaba el algoritmo de Kruskal sobre grafos aleatorios decidimos hacer las pruebas en nuestras computadoras personales las cuales tienen diferentes especificaciones y nos permitiría obtener resultados mas realistas sobre como funciona el algoritmo según nuestra implementación del mismo y así también probar la implementación de los grafos aleatorios.

Para poder graficar nuestros resultados utilizamos GeoGebra creando puntos con los resultados en base a la cantidad de vértices utilizadas en los grafos aleatorios, siempre con la cantidad máxima de aristas según la formula $\frac{n(n-1)}{2}$. El eje x es equivalente a la cantidad de vértices del grafo y el eje y es equivalente a la cantidad de tiempo en milisegundos. También en los gráficos debajo de cada uno mostraremos las especificaciones de nuestras computadoras como PC_1 y PC_2.

Por una mejor visualización de los gráficos se han separado en las próximas hojas (10 y 11) de modo que pueda observarse mejor las diferencias

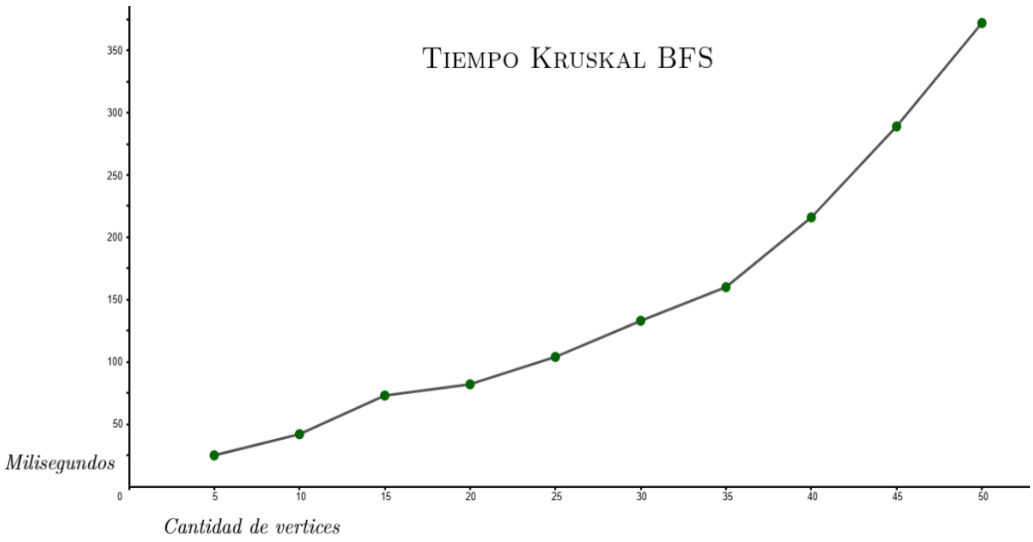
Kruskal BFS:

Vértices	Tiempo(ms)
5	30
10	44
15	66
20	85
25	97
30	139
35	191
40	258
45	306
50	414



Especificaciones de PC_1 : AMD R52600 y 16GB de RAM.

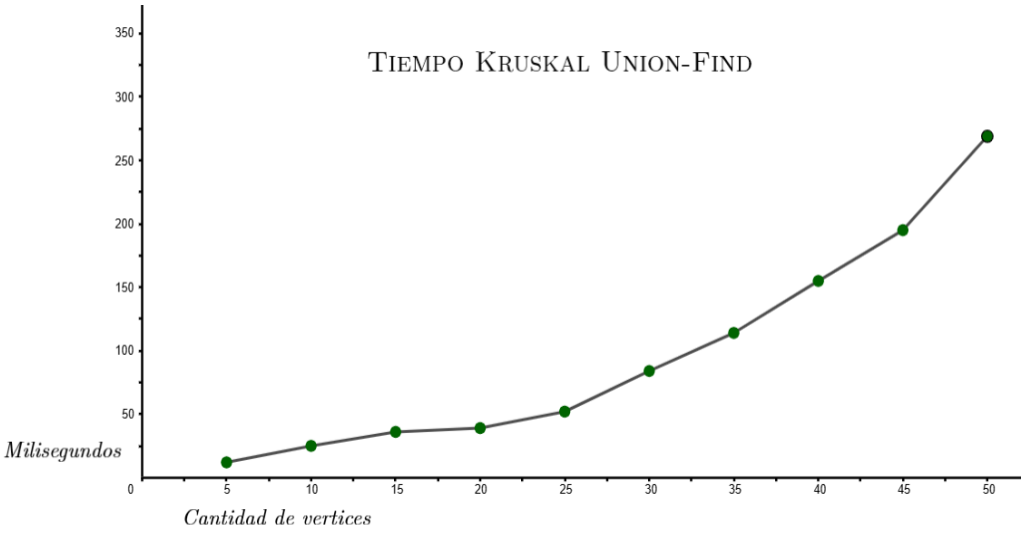
Vértices	Tiempo(ms)
5	28
10	49
15	64
20	78
25	102
30	134
35	162
40	207
45	283
50	363



Especificaciones de PC_2 : Intel i5 8va Gen y 8GB de RAM.

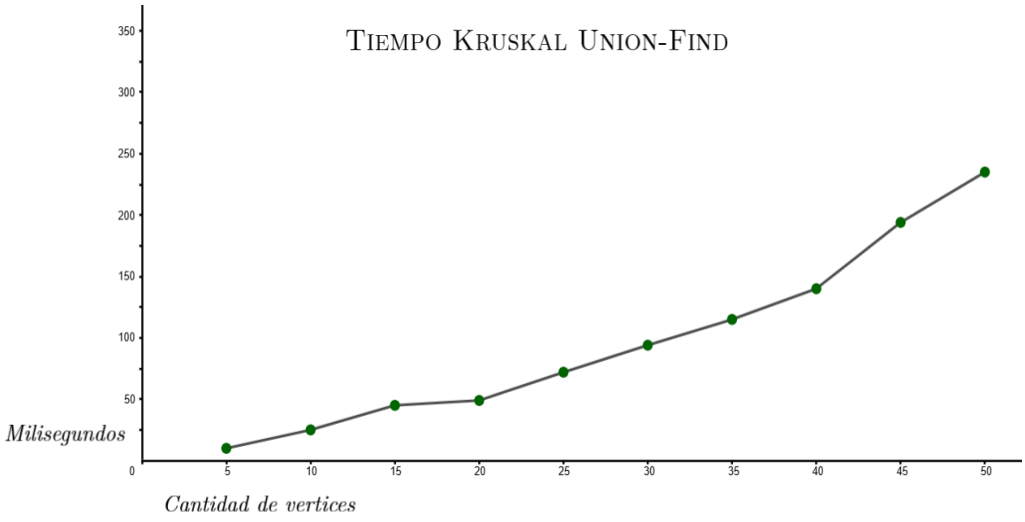
Kruskal Union-Find:

Vértices	Tiempo(ms)
5	12
10	25
15	36
20	39
25	52
30	84
35	114
40	155
45	195
50	269



Especificaciones de PC_1 : AMD R52600 y 16GB de RAM.

Vértices	Tiempo(ms)
5	15
10	26
15	49
20	54
25	76
30	103
35	124
40	146
45	191
50	232



Especificaciones de PC_2 : Intel i5 8va Gen y 8GB de RAM.