Escuela de Ingeniería
Departamento de Computación
Análisis y Diseño de Algoritmos (TC2017)
Profesor: Dr. Vicente Cubells Nonell

Tecnológico de Monterrey

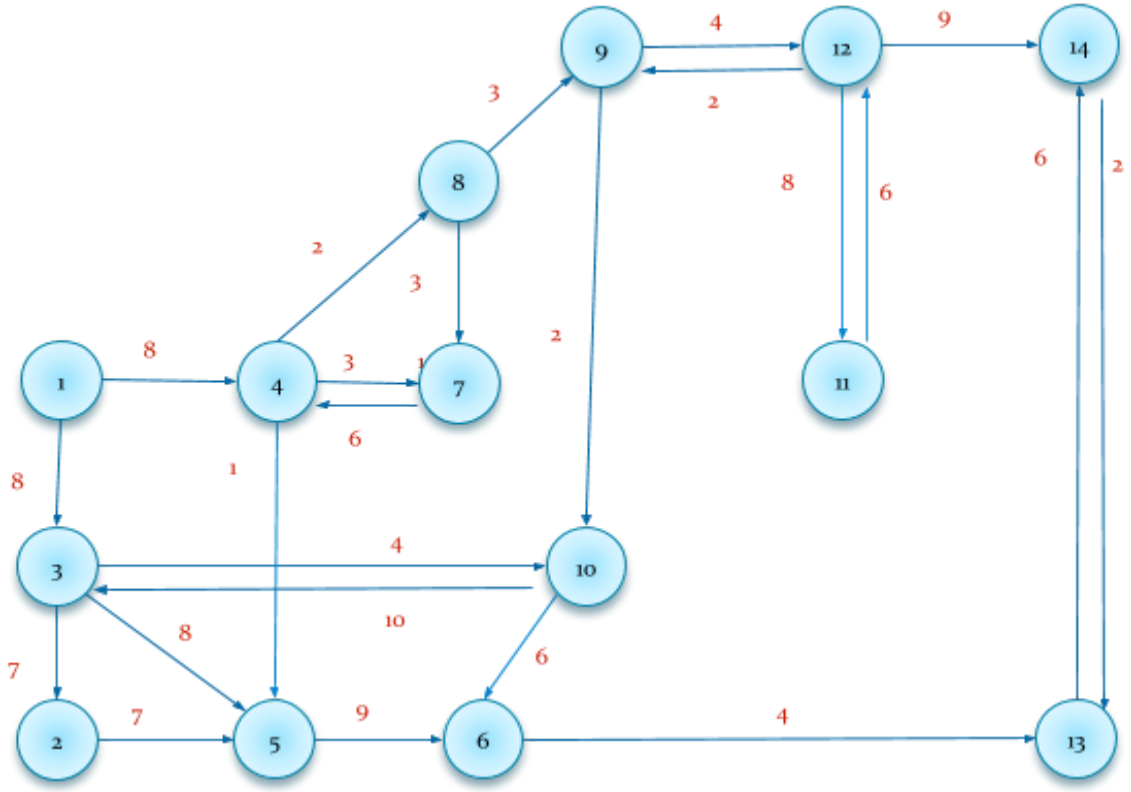
Problema Práctico No. 2

Título	**Implementación de algoritmos de grafos**
Aprendizaje esperado (objetivo)	El alumno aprenderá a utilizar dos de las bibliotecas existentes para la programación de grafos y la utilización de los algoritmos de recorridos en profundidad y en amplitud, obtención de un árbol de expansión mínimo mediante Kruskal y Prim, así como también los algoritmos de caminos mínimos Dijkstra y Floyd-Warshall.
Instrucciones	Busque en Internet las bibliotecas para la implementación de grafos (Boost Graph Library y SNAP) e implemente un programa que haciendo uso de cada biblioteca permita: • Insertar vértices en el grafo • Insertar arista en el grafo • Eliminar vértices del grafo • Eliminar aristas del grafo • Realizar un recorrido en profundidad (DFS) • Realizar un recorrido en amplitud (BFS) • Obtener el árbol de recubrimiento mínimo correspondiente utilizando el algoritmo de Prim • Obtener el árbol de recubrimiento mínimo correspondiente utilizando el algoritmo de Kruskal • Determinar la ruta mínima para llegar de un vértice origen a todos los demás vértices del grafo (Dijkstra) • Determinar la ruta mínima para llegar de cualquier vértice origen a todos los demás vértices del grafo (Floyd-Warshall) Posteriormente modele en cada programa el grafo que aparece a continuación y mida los tiempos de ejecución de la implementación de cada algoritmo (de los anteriores) utilizando cada una de las bibliotecas tanto en su laptop como en una RPi. Para cada uno de los algoritmos utilizados, analice su complejidad temporal y espacial. Finalmente, genere un reporte como resultado de la investigación donde analice los resultados obtenidos y en caso de haber diferencias entre los tiempos de ejecución utilizando una biblioteca u otra, realice una reflexión personal donde mencione con sus palabras cuáles son los factores que ocasionan dichas diferencias de tiempo. Como parte de la respuesta de este ejercicio debe incluir en el reporte de investigación, lo siguiente: ● Tabla con los tiempos de ejecución de cada algoritmo utilizado de cada biblioteca, tanto en la RPi como en la laptop / escritorio. ● Para cada algoritmo, una gráfica comparativa con sus tiempos de ejecución en la RPi y en la laptop / escritorio para cada biblioteca. ● Análisis e interpretación de los resultados (con sus palabras). Suba a la plataforma el archivo en Google Docs e incluya en el mismo la liga al repositorio de GitHub dentro de la clase (https://classroom.github.com/g/cQ0EZq64) que contenga todos los códigos programados.

	No se aceptan trabajos fuera de fecha ni por correo electrónico. En ambos casos la calificación de la actividad será 0 puntos.
Lugar en que se llevará a cabo	Casa
Forma de trabajo	En equipos de 2 estudiantes
Recursos	Notas de clases Foros de información en Internet Wikipedia (http://www.wikipedia.org) Códigos de grafos vistos en la materia Estructura de Datos Computadora
Tiempo estimado	10 horas
Criterios de evaluación	La evaluación se realizará de la siguiente manera: Programar cada algoritmo correctamente y medir el tiempo de ejecución (70 puntos, 7 puntos por cada algoritmo) Implementación correcta del grafo (5 puntos) Programar el ejemplo que demuestre el uso de cada algoritmo (15 puntos) Obtener la complejidad de cada algoritmo (5 puntos). Obtener la técnica de diseño de cada algoritmo (5 puntos).
Valor de la actividad	20% de la calificación del segundo parcial

Grafo a modelar

Aparece a continuación



Respuestas

Repositorio de GitHub:

https://github.com/tec-csf/tc2017-p2-otono-2019-equipo-algoritmos

Tabla con los resultados de las mediciones:

Tabla de resultados a completar (*tiempo en ms*)				
No.	**Algoritmo**		**Boost**	**SNAP**
1	**Insertar vértice**	*RPi*	100	128
		PC	0	0
2	**Insertar arista**	*RPi*	100	175
		PC	0	1
3	**Eliminar vértice**	*RPi*	200	129
		PC	0	0
4	**Eliminar arista**	*RPi*	200	78
		PC	0	0
5	**DFS**	*RPi*	280	67
		PC	30	98
6	**BFS**	*RPi*	200	198
		PC	20	87
7	**Prim**	*RPi*	1000	1503
		PC	310	500
8	**Kruskal**	*RPi*	1200	782
		PC	330	562
9	**Dijkstra**	*RPi*	850	190
		PC	240	102
10	**Floyd-Warshall**	*RPi*	800	1483
		PC	100	243

Interpretación de los resultados:

- **DFS:** Se realiza un recorrido en profundidad de los vértices de un grafo de manera iterativa. Comienza desde el nodo inicial e intenta ver hasta dónde puede llegar. La técnica de algoritmos es ávido pues realiza un "greedy choice" siempre que llega a un nuevo nodo para decidir a dónde se va a mover después. Su complejidad es lineal pues simplemente se consideran el número de vértices y de aristas que se tienen dentro del grafo; O(|V| + |E|).
- **BFS:** Se recorren en amplitud todos los nodos (nivel por nivel) de un grafo de manera iterativa. Comienza en el nodo inicial y explora todos los nodos adyacentes al nodo actual antes de moverse al siguiente nivel. Igual que DFS, es un algoritmo ávido pues en cada momento considera cuál es la mejor solución y también tiene una complejidad O(|V| + |E|); aunque en nuestro caso varió terriblemente.
- **Prim:** Encuentra el árbol de expansión mínima de un grafo no dirigido. Así como los dos algoritmos anteriores, también utiliza una técnica ávida, pues busca el peso más pequeño entre un nodo A y B para obtener el peso más pequeño global. Su complejidad puede variar dependiendo de la implementación aunque, idealmente, utilizando un heap binario y una lista de adyacencia, se puede obtener una complejidad de O(|E| log|V|).
- **Kruskal:** Se utilizan clusters (o conjuntos) para almacenar los vértices. Es un algoritmo ávido, pues todas sus decisiones son de manera local. Su complejidad es de O(|E| log|E|).
- **Dijkstra:** Te permite obtener el camino más corto del nodo origen a cualquier otro nodo de un grafo dirigido. Su implementación es muy similar al de PRIM variando únicamente en la manera de registrar el peso; pues ahora sí realiza la suma total de los nodos anteriores para llegar a un nodo B. Su complejidad utilizando una lista de prioridad es de $O(|V|^2)$ y la técnica de algoritmo es ávida.
- **Floyd-Warshall:** Se utiliza una matriz de adyacencia para "mapear" los costos de los caminos más cortos entre todo par de vértices. En este caso, el algoritmo puede trabajar muy bien con un grafo dirigido y uno no dirigido. Su complejidad es $O(|V^3|)$ ya que utiliza tres ciclos for anidados para poder recorrer y llenar correctamente la matriz. Funciona utilizando programación dinámica para dividir el problema en subproblemas para finalmente combinar la solución de los subproblemas para tener una solución global.

Nos podemos dar cuenta, a partir de los resultados observados en las gráficas y tablas comparativas, que los resultados son realmente similares. Hay una que otra ocasión donde, sorprendentemente, la RPi fue más rápida que la PC usando SNAP (Gráfica 5); realmente no sabría explicar por qué fue que estos resultados salieron así.

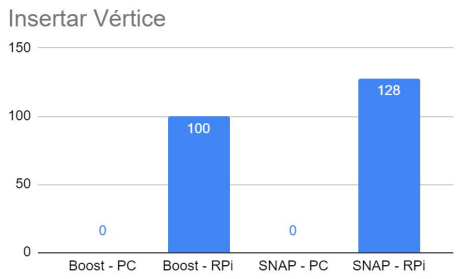
Por otro lado; podemos observar que la implementación del algoritmo de Prim utilizando SNAP y BOOST es realmente lento comparado con el resto (Gráfica 7); y al revisar el código con la librería de SNAP, por ejemplo, rápidamente nos damos cuenta que se está utilizando un triple "for" anidado, obteniendo así una complejidad de $O(|V|^3)$. Esto último se hizo porque el grafo que tuvimos que programar era dirigido, entonces, para asegurar que los pesos del nodo A al nodo B y viceversa fueran los mismos se tuvo que recorrer los nodos adyacentes de A (nodo padre), los nodos adyacentes de B y que de éstos últimos, ninguno fuera el nodo padre. Algo muy interesante que notar, es que el algoritmo de Floyd-Warshall (Gráfico 10), aunque también tiene una complejidad de $O(|V|^3)$, fue considerablemente más rápido que el de Prim. Esto probablemente se debe a que en la implementación de SNAP cada vez se hace una revisión de los nodos adyacentes del hijo, sin importar que se hayan visitado antes, lo cual implica trabajo doble (o ¡triple!) completamente innecesario.

La programación del algoritmo de Dijkstra y de Kruskal con Boost quizás no fueron las más eficientes y por eso es que los resultados están tan diferentes al compararlos con SNAP (Gráficas 8 y 9). En la gráfica 10 se puede apreciar lo contrario, la programación del algoritmo de Floyd-Warshall utilizando SNAP es considerablemente más lenta que utilizando Boost (1483 ms vs 800 ms). Asimismo, las "operaciones básicas" (Gráficas 1, 2, 4) de insertar/eliminar un vértice o un arista son más lentas en SNAP, quizás es porque la librería de SNAP es bastante grande y como nosotros utilizamos una Máquina Virtual con especificaciones reducidas, es probable que haya influído mucho en la ejecución de estos algoritmos.

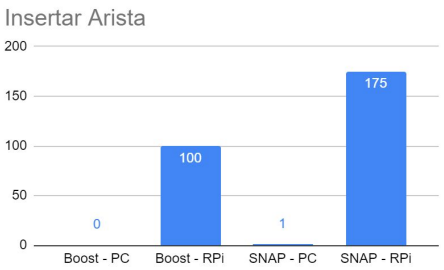
De manera general, se llegó a la conclusión que ejecutar algoritmos en la RPi es realmente lento, pues sus especificaciones son muy limitadas comparadas con las de una computadora normal. Asimismo, los tiempos de ejecución pudieron variar porque las librerías se ejecutaron en diferentes computadoras y, como se mencionó anteriormente, utilizando sistemas operativos diferentes (Windows y Ubuntu)

Fue muy interesante aprender a utilizar estos algoritmos con una librería completamente nueva. Considero que en general se puede hacer un mejor trabajo para implementar todos los algoritmos si se aprende a utilizar correctamente las funciones que están dentro de las librerías.

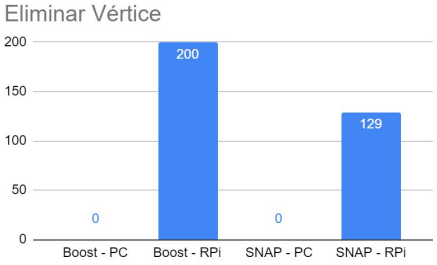
Gráficas comparativas



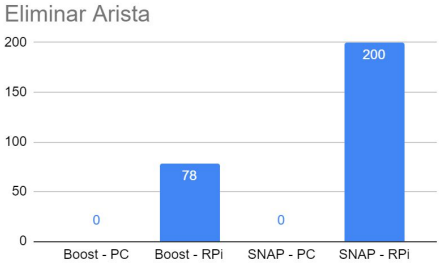
Gráfica 1



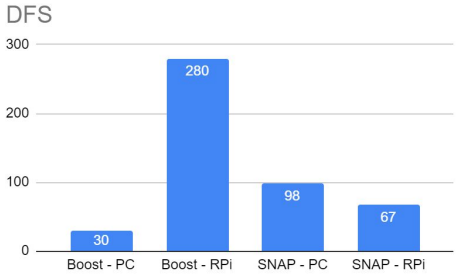
Gráfica 2



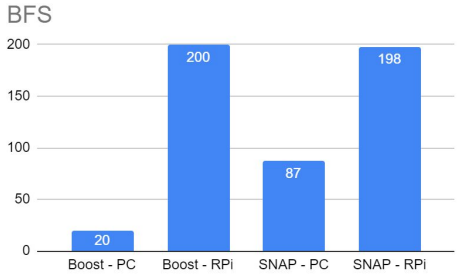
Gráfica 3



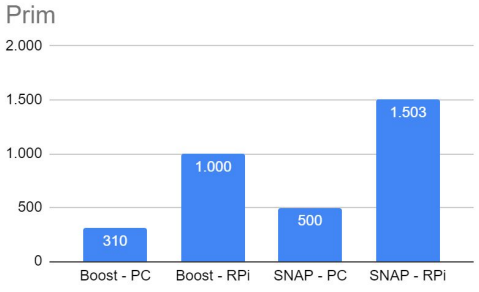
Gráfica 4



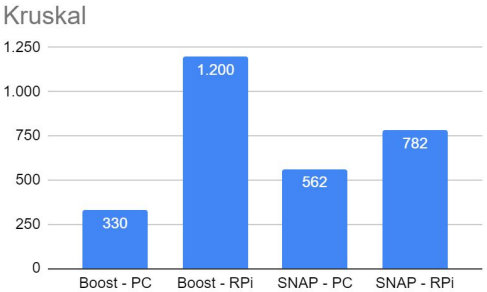
Gráfica 5



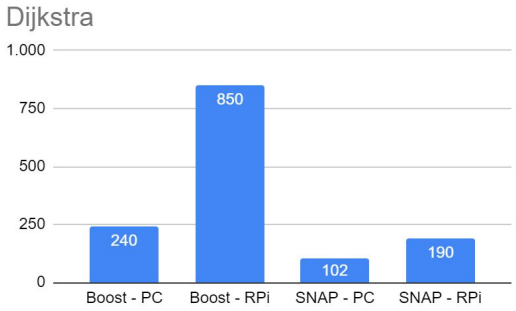
Gráfica 6



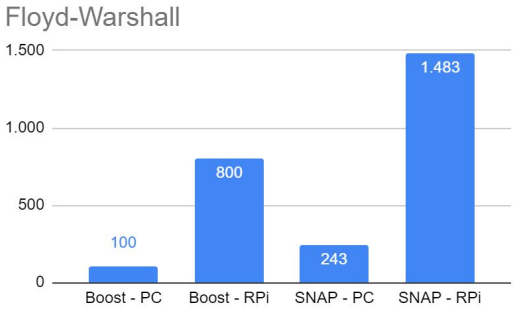
Gráfica 7



Gráfica 8



Gráfica 9



Gráfica 10