**­**



**Materia:** Algoritmos Complejos para Estructuras de Datos Avanzados

**Curso: K3051**

**Docente:** Pablo Augusto Sznajdleder

**Trabajo Práctico:** Monografía de los algoritmos de Prim y Kruskal

**Versión**: 1

**Equipo Nro:**

|  |  |
| --- | --- |
| Apellido y Nombre | Numero de Legajo |
| Baltazar, Cristian | 141505-0 |
| Jordan, Johana Andrea | 138698-0 |
| Mendez, Maximiliano | 143769-0 |
| Mamani Gutierrez, Jeanette | 144375-6 |
|  |  |

**Fecha de Entrega:** 25 / 06 / 2014 **Calificación:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Fecha Devolución:** \_\_\_/\_\_\_/\_\_\_\_ **Firma Docente:**

**ALGORITMO DE KRUSKAL**

El Algoritmo de Kruskal, es un algoritmo de la teoría de grafos. Es uno de los más fáciles de entender y probablemente el mejor para resolver problemas prácticos.

Su objetivo es, a partir de un grafo conexo no dirigido, formar un árbol de mínimo peso, que contenga todos sus vértices (Árbol Recubridor Mínimo o Árbol Expandido Mínimo).

Si el grafo no es conexo, halla el Bosque Recubridor mínimo. Es decir, el Árbol Recubridor Mínimo para cada componente conexa del grafo.

Fue creado en 1956 por el matemático y estadístico estadounidense Joseph B. Kruskal. Se aplica normalmente a problemas de optimización, en la búsqueda del valor óptimo de una cierta función.

**REQUISITOS PARA SU APLICACIÓN**

Para poder aplicar este algoritmo, el grafo debe contar con las siguientes características:

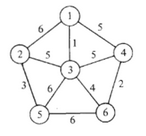
* **Debe ser conexo** (Para cualquier par de vértices existe al menos un camino que los conecte).
* **Debe ser no dirigido** (aristas no orientadas).
* **No debe presentar bucles.**
* **Debe tener todas sus aristas etiquetadas** (Ponderadas).

**PROCEDIMIENTO DE EJECUCIÓN**

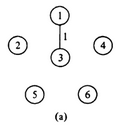
Se procede a elegir las aristas de menor peso, de a una a la vez, y sin repetirlas; con la única salvedad de que no se formen ciclos. Es decir, que en el caso de que la próxima arista de menor peso formara un ciclo en el árbol que se está generando, hay que descartarla, y continuar por la evaluación de la siguiente arista de menor peso.

Este procedimiento se realiza hasta que se hayan seleccionado V - 1 aristas, siendo V la cantidad de Vértices en el grafo dado.

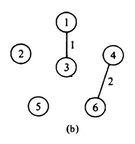
**EJEMPLO PRÁCTICO**



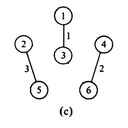
Sea el siguiente grafo conexo, no dirigido y cuyas aristas tienen un peso asignado, hallaremos el árbol recubridor mínimo. Según la cantidad de vértices (6), podemos determinar que necesitaremos elegir 5 aristas.



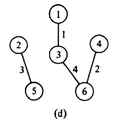
**a**. Elegimos la arista de menor peso (1-3), en el caso de que hubieran varias del mismo peso, se escoge arbitrariamente.



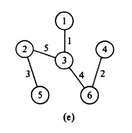
**b**. Elegimos la siguiente arista con menor peso (4-6), teniendo en cuenta que no se formen ciclos cerrados (para que siempre se forme un árbol).



**c**. Escogemos la arista 2-5.



**d**. La siguiente arista de menor peso es la 3-6



**e**. Si bien no quedó ningún vértice aislado, debemos elegir otra arista más, para poder formar el árbol. De esta forma, hemos hallado el Árbol Recubridor Mínimo del grafo dado.

**APLICACIÓN EN LA PRÁCTICA**

Al igual que el Algoritmo de Prim, el Algoritmo de Kruskal se utiliza en diversas áreas, tales como para la distribución de la red eléctrica, el diseño de circuitos algorítmicos complejos, el diseño de circuitos electrónicos, etc.

**COMPLEJIDAD**

La complejidad computacional temporal es de O (nlog n)siendo **n** el número de vértices del grafo.

La complejidad computacional espacial es O (n2) ya que el grafo se implementa mediante su matriz de adyacencia.

**CONCLUSIÓN**

El algoritmo de KRUSKAL es el algoritmo de búsqueda de Árboles Generadores Mínimos de mejor desempeño para grafos dispersos.

**LINKS DE INTERES**

**SIMULADOR:** <http://students.ceid.upatras.gr/~papagel/project/kruskal.htm>

**VIDEO DE RESOLUCIÓN:** <https://www.youtube.com/watch?v=SilS8f8icWY>

**ALGORITMO DE PRIM**

El algoritmo de Prim es un algoritmo perteneciente a la [teoría de los grafos](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos). Este algoritmo permite encontrar el subconjunto de menor peso posible de [aristas](http://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)) que forman un [árbol](http://es.wikipedia.org/wiki/%C3%81rbol_(teor%C3%ADa_de_grafos)) con todos los vértices (Árbol generador mínimo) en un grafo conexo no dirigido.

Fue diseñado inicialmente en 1930 por el matemático Vojtech Jarnik y luego, en 1957, de manera independiente por el científico computacional Robert C. Prim (de allí su nombre). Posteriormente fue redescubierto en 1959 por El científico computacional Edsger Dijkstra en 1959. Es por estos motivos que el algoritmo es también conocido como **algoritmo DJP** o **algoritmo de Jarnik**.

**REQUISITOS PARA SU APLICACIÓN**

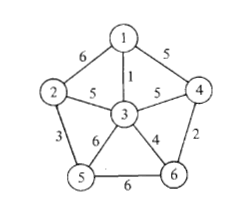
Para poder aplicar el algoritmo de Prim el grafo analizado debe cumplir con 4 requisitos fundamentales:

* **Debe ser conexo** (Para cualquier par de vértices existe al menos un camino que los conecte).
* **Debe ser no dirigido** (sus aristas no están orientadas, gráficamente son curvas sin flechas).
* **No debe presentar ciclos** (Para cualquier camino del grafo no se debe repetir ningún vértice excepto el primero).
* **Debe tener todas sus aristas etiquetadas** (De esta forma podremos evaluar el peso de cada camino entre nodos).

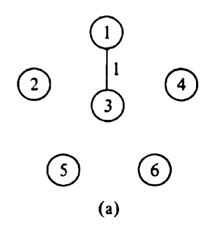
**PROCEDIMIENTO DE EJECUCIÓN**

El procedimiento de ejecución del algoritmo es relativamente sencillo, se avanza con generación del árbol a partir de un vértice inicial definido al cual se le van agregando sucesivamente vértices que contemplen una distancia (peso de la arista) mínima incidente a cualquiera de los vértices que ya pertenecen al Árbol. Finalmente el árbol generador mínimo estará completamente construido cuando no quedan más vértices por agregar.

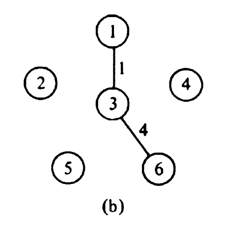
**EJEMPLO PRÁCTICO**



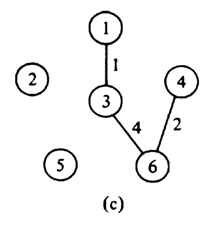
Sea el siguiente grafo conexo, no dirigido y cuyas aristas tienen un peso asignado, hallaremos el árbol generador mínimo. Tomaremos arbitrariamente al vértice **1** como punto de partida.



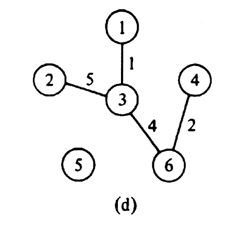
**a**. Evaluamos la cercanía de los vértices incidentes a 1**:** el vértice 2 tiene un peso de 6, el vértice 3 un peso de 1 y el vértice 4 un peso de 5 por lo que optaremos por el vértice 3.



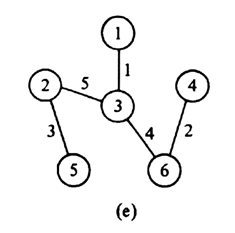
**b**. Ya tenemos 2 vértices en nuestro árbol, a continuación evaluamos la cercanía de los vértices incidentes a los vértices ya contenidos dentro de nuestro árbol (1 y 3): El vértice 2 tiene un peso de 6 (desde 1) y 5 (desde 3), el vértice 4 un peso de 1 y el vértice 4 un peso 5. El vértice 6 un peso 4 y el vértice 5 un peso 6 por lo que iremos por el vértice 6



**c.** Tal como realizamos los pasos anteriores ahora podemos identificar que el nodo 4 será el próximo nodo a considerar en nuestro árbol debido a que su peso es el menor entre los nodos incidentes.



**d.** Continuando con la lógica establecida notamos que la arista de menor peso es de valor 5 y se presenta tanto entre los vértices 4 y 3 como entre 4 y 1 y 3 y 2 pero como los vértices 1 y 3 ya están incluidos en nuestro árbol y debemos evitarlos los ciclos avanzamos con la incorporación de 2.



**e.** Finalmente incorporando el último vértice restante mediante la arista de menor peso habremos conformado el árbol generador mínimo.

**APLICACIÓN EN LA PRÁCTICA**

El algoritmo de Prim se utiliza en una extensa variedad de campos del conocimiento, es utilizado normalmente para ahorrar recursos y sus aplicaciones más comunes se pueden encontrar en la implementación de cables de redes, de servidores y postes de luz. También son utilizados dentro de la programación para el diseño de circuitos algorítmicos complejos, en el campo de la electrónica para el diseño de circuitos electrónicos y hasta en administración para el diseño de los organigramas de jerarquías.

**COMPLEJIDAD**

Tanto la complejidad computacional temporal como la espacial son del orden de O(n2) representando **n** el número de vértices del grafo.

**CONCLUSIÓN**

Podemos destacar que el algoritmo de PRIM es el algoritmo de búsqueda de MST (Árboles Generadores Mínimos) más sencillo de implementar y el de mejor desempeño para grafos densos.

**LINKS DE INTERES**

**SIMULADOR:** <http://students.ceid.upatras.gr/~papagel/project/prim.htm>

**JAVA CODE:** <http://pastecode.org/index.php/view/5907833>

**VIDEO DE RESOLUCIÓN:** <https://www.youtube.com/watch?v=-2NlTulDUq4>

**LABERINTO**: <http://upload.wikimedia.org/wikipedia/commons/b/b1/MAZE_30x20_Prim.ogv>