****

Universidad de Oriente

Núcleo Anzoátegui

Escuela de Ingeniería

Departamento de Computación y Sistemas

Análisis y Diseño de Algoritmos

Realizado por:

Miguel Rojas C.I.: 18210575

Drusmary Moya C.I: 19839122

**Prof: Claudio Cortinez**

**Bna; 13/07/12**

**Algoritmo de Prim**

El algoritmo de Prim es un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) perteneciente a la [teoría de los grafos](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos) para encontrar un [árbol recubridor mínimo](http://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo) en un [grafo](http://es.wikipedia.org/wiki/Grafo) [conexo](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos#Grafos_conexos),no dirigido y cuyas [aristas](http://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)) están etiquetadas.

En otras palabras, el [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) encuentra un subconjunto 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](http://es.wikipedia.org/wiki/V%C3%A9rtice_(Teor%C3%ADa_de_grafos)), donde el peso total de todas las [aristas](http://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)) en el árbol es el mínimo posible. Si el grafo no es conexo, entonces el algoritmo encontrará el [árbol recubridor mínimo](http://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo) para uno de los componentes conexos que forman dicho grafo no conexo.

El [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) fue diseñado en 1930 por el matemático [Vojtech Jarnik](http://es.wikipedia.org/wiki/Vojtech_Jarnik) y luego de manera independiente por el científico computacional [Robert C. Prim](http://es.wikipedia.org/wiki/Robert_C._Prim) en 1957 y redescubierto por [Dijkstra](http://es.wikipedia.org/wiki/Dijkstra) en 1959. Por esta razón, el algoritmo es también conocido como algoritmo DJP o algoritmo de Jarnik.

**Aplicaciones del algoritmo**

Diseño de Redes Físicas: teléfonos, eléctricas, hidráulicas, TV por cable, computadores, carreteras, etc.

Solución aproximada de problemas NP.

Distribución de mensajes entre agentes.

Aplicaciones indirectas: plegamiento de proteínas, reconocimiento de células cancerosas, etc.

El algoritmo incrementa continuamente el tamaño de un árbol, comenzando por un vértice inicial al que se le van agregando sucesivamente vértices cuya distancia a los anteriores es mínima. Esto significa que en cada paso, las aristas a considerar son aquellas que inciden en vértices que ya pertenecen al árbol.

El árbol recubridor mínimo está completamente construido cuando no quedan más vértices por agregar.

**Demostración**

Sea *G* un grafo [conexo](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos#Grafos_Conexos) y [ponderado](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos#Grafos_ponderados).

En toda iteración del algoritmo de Prim, se debe encontrar una arista que conecte un nodo del subgrafo a otro nodo fuera del subgrafo.

Ya que *G* es conexo, siempre habrá un camino para todo nodo.

La salida *Y* del algoritmo de Prim es un árbol porque las aristas y los nodos agregados a *Y* están conectados.

Sea *Y* el árbol recubridor mínimo de *G*.

Si Y_1 = Y  \Rightarrow Y es el árbol recubridor mínimo.

Si no, sea *e* la primera arista agregada durante la construcción de *Y*, que no está en *Y*1 y sea *V* el conjunto de nodos conectados por las aristas agregadas antes que *e*. Entonces un extremo de *e* está en *V* y el otro no. Ya que *Y*1 es el árbol recubridor mínimo de *G* hay un camino en *Y*1 que une los dos extremos. Mientras que uno se mueve por el camino, se debe encontrar una arista *f* uniendo un nodo en *V* a uno que no está en *V*. En la iteración que *e* se agrega a *Y*, *f* también se podría haber agregado y se hubiese agregado en vez de *e* si su peso fuera menor que el de *e*. Ya que *f* no se agregó se concluye:

P(f) \geq P(e)

Sea *Y*2 el grafo obtenido al remover *f* y agregando e \in Y_1. Es fácil mostrar que *Y*2 conexo tiene la misma cantidad de aristas que *Y*1, y el peso total de sus aristas no es mayor que el de *Y*1, entonces también es un árbol recubridor mínimo de *G* y contiene a *e* y todas las aristas agregadas anteriormente durante la construcción de *V*. Si se repiten los pasos mencionados anteriormente, eventualmente se obtendrá el árbol recubridor mínimo de *G* que es igual a *Y*.

Esto demuestra que *Y* es el árbol recubridor mínimo de *G*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Image** | **Descripción** | **No visto** | **En el grafo** | **En el árbol** |
| [Prim Algorithm 0.svg](http://es.wikipedia.org/wiki/Archivo:Prim_Algorithm_0.svg) | Este es el grafo ponderado de partida. No es un árbol ya que requiere que no haya ciclos y en este grafo los hay. Los números cerca de las aristas indican el peso. Ninguna de las aristas está marcada, y el vértice **D** ha sido elegido arbitrariamente como el punto de partida. | C, G | A, B, E, F | D |
| [Prim Algorithm 1.svg](http://es.wikipedia.org/wiki/Archivo:Prim_Algorithm_1.svg) | El segundo vértice es el más cercano a **D**: **A** está a 5 de distancia, **B** a 9, **E** a 15 y **F** a 6. De estos, 5 es el valor más pequeño, así que marcamos la arista**DA**. | C, G | B, E, F | A, D |
| [Prim Algorithm 2.svg](http://es.wikipedia.org/wiki/Archivo:Prim_Algorithm_2.svg) | El próximo vértice a elegir es el más cercano a **D** o **A**. **B** está a 9 de distancia de **D** y a 7 de **A**, **E** está a 15, y **F** está a 6. 6 es el valor más pequeño, así que marcamos el vértice **F** y a la arista **DF**. | C | B, E, G | A, D, F |
| [Prim Algorithm 3.svg](http://es.wikipedia.org/wiki/Archivo:Prim_Algorithm_3.svg) | El algoritmo continúa. El vértice **B**, que está a una distancia de 7 de **A**, es el siguiente marcado. En este punto la arista **DB** es marcada en rojo porque sus dos extremos ya están en el árbol y por lo tanto no podrá ser utilizado. | null | C, E, G | A, D, F, B |
| [Prim Algorithm 4.svg](http://es.wikipedia.org/wiki/Archivo:Prim_Algorithm_4.svg) | Aquí hay que elegir entre **C**, **E** y **G**. **C** está a 8 de distancia de **B**, **E** está a 7 de distancia de **B**, y **G** está a 11 de distancia de **F**. **E** está más cerca, entonces marcamos el vértice **E** y la arista **EB**. Otras dos aristas fueron marcadas en rojo porque ambos vértices que unen fueron agregados al árbol. | null | C, G | A, D, F, B, E |
| [Prim Algorithm 5.svg](http://es.wikipedia.org/wiki/Archivo:Prim_Algorithm_5.svg) | Sólo quedan disponibles **C** y **G**. **C** está a 5 de distancia de **E**, y **G** a 9 de distancia de **E**. Se elige **C**, y se marca con el arco **EC**. El arco **BC** también se marca con rojo. | null | G | A, D, F, B, E, C |
| [Prim Algorithm 6.svg](http://es.wikipedia.org/wiki/Archivo:Prim_Algorithm_6.svg) | **G** es el único vértice pendiente, y está más cerca de **E** que de **F**, así que se agrega **EG** al árbol. Todos los vértices están ya marcados, el [árbol de expansión mínimo](http://es.wikipedia.org/wiki/%C3%81rbol_de_expansi%C3%B3n_m%C3%ADnimo) se muestra en verde. En este caso con un peso de 39. | null | null | A, D, F, B, E, C, G |

**Algoritmo de Prim**

Variables

A= conjunto de aristas  
N= conjunto de vértices

u=es como un vértice auxiliar

T= árbol generado con el algoritmo

Función Prim (G, peso)

Entrada grafo G=<N,A>

Realpeso [A]

T←vacio

V←un elemento cualquiera de N

Mientras V≠N hacer

Encontrar {u, v} de peso mínimo, tal que u,v∈N , v∈V y u∉V

T←T∪ {u, v}

V←V∪ {u}

Fin mientras

Devolver T

Finfunción