Teoría de Grafos y Algoritmia
Parte 2: Teoría de Arboles

Topicos

- Árboles de expansión
  - Búsqueda a lo ancho (BFS)
  - Búsqueda en profundidad (DFS)
- Árboles de expansión mínimos
  - Prim
  - Kruskal

Teoría de Árboles – Árboles de expansión

Ahora consideraremos el problema de determinar un subgrafo T de un grafo G de modo que T sea un árbol con todos los vértices de G; es decir, un árbol de expansión.

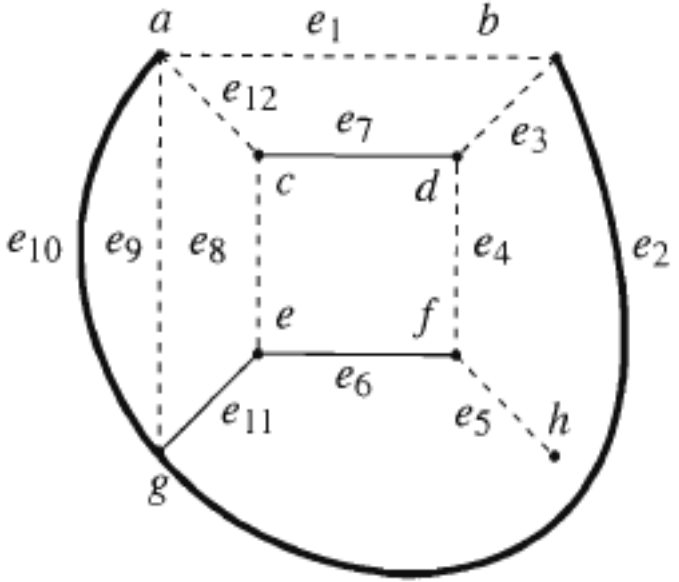
Veremos que los métodos para determinar árboles de expansión se pueden aplicar también a otros problemas.

Definimos entonces que un árbol T es un Árbol de expansión de un grafo G si T es una subgrafo de G que contiene a todos los vértices de G.

Vemos que un grafo $G = (V, E)$ tiene un árbol de expansión $T = (V', E')$ si y sólo si G es conexo.

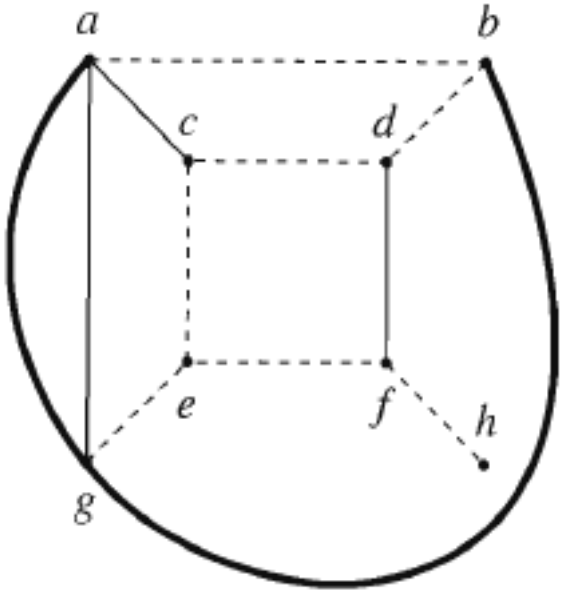
Teoría de Árboles – Árboles de expansión

Un árbol de expansión del grafo G de la siguiente figura aparece en líneas punteadas.



Teoría de Árboles – Árboles de expansión

En general, un grafo tendrá varios árboles de expansión posibles. Por ejemplo, otro árbol de expansión de la misma figura anterior está marcado en líneas punteadas en la siguiente figura.



Teoría de Árboles – Árboles de expansión

Vamos a ver dos algoritmos para poder encontrar un árbol de expansión.

Lo primero que tenemos que considerar como entrada para estos algoritmos es un grafo conexo G y un orden entre los vértices, si no se indica explícitamente se usará el orden usual numérico o lexicográfico y, su salida sería un árbol de expansión T sobre G.

Nuestro primer algoritmo se denomina Búsqueda a lo ancho, también llamado BFS.

Es un algoritmo que consiste en partir de un vértice (la raíz del árbol obtenido) e ir agregando a los adyacentes a la raíz en cada iteración, es decir, en la primera iteración agregaremos a los vértices que están a una distancia de 1 de la raíz elegida, en la segunda iteración se agregarán los vértices que están a una distancia de 2 de la raíz y, así sucesivamente hasta que todos los vértices formen parte del árbol.

Algoritmo $bfs(V, E)$

$S := (v_1)$

$V' := \{v_1\}$

$E' := \emptyset$

$Mientras\ V' \neq V\ entonces$

$Para\ cada\ x \in S$, en orden, $hacer$

$Para\ cada\ y \in V - V'$, en orden, $hacer$

$Si\ (x, y) \in E\ Entonces$

$E' := E' \cup \{(x, y)\}$

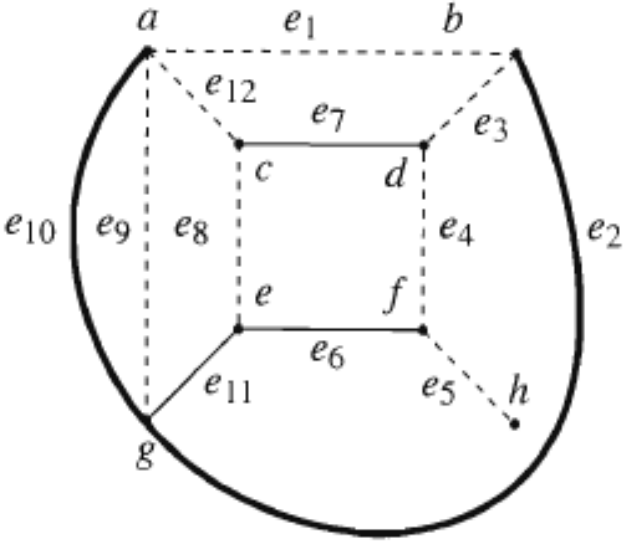
$V' := V' \cup \{y\}$

$Retornar\ \ T = (V', E')$

Apliquemos el algoritmo sobre el grafo que vimos en el slide 3.

Primero, elegimos un orden, digamos $abcdefgh$, de los vértices de G.

Elegimos el primer vértice a y lo etiquetamos como la raíz. Sea $T = (V', E')$ el árbol que comienza siendo $V' = \{a\}$ y $E' = \emptyset$.



Agregamos a T todas las aristas de la forma (a, x) y los vértices sobre los cuales son incidentes, desde $x = b$ hasta h, que no produzcan un ciclo al agregarse a T.

Así, agregamos a T las aristas (a, b), (a, c) y (a, g) (Podemos usar cualesquiera de las aristas paralelas incidentes en a y g).

Repetimos este procedimiento con los vértices del nivel 1, examinándolos en orden:

- b, incluye el lado (b, d);
- c, incluye el lado (c, e);
- g, ninguno.

Nuestro segundo algoritmo se denomina Búsqueda en profundidad, también llamado DFS.

Es un algoritmo que consiste en partir de un vértice (la raíz del árbol obtenido) e ir agregando un vértice y pasar a los niveles sucesivos apenas se pueda.

Esto hace que cuando no se puede seguir avanzando, tengamos que volver al padre del vértice que estábamos examinando.

Algoritmo $dfs(V, E)$

$w := v_1$

$V' := \{v_1\}$

$E' := \emptyset$

$Mientras\ V' \neq V\ entonces$

$Mientras$ existe (w, v) con $v \in V - V'\ entonces$

sea v_k el mínimo que cumple con esta condición

$E' := E' \cup \{(w, v_k)\}$

$V' := V' \cup \{v_k\}$

$w := v_k$

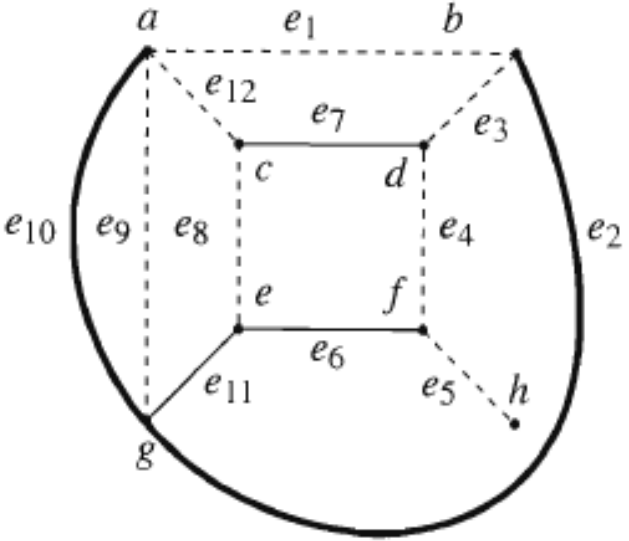
$Si\ V' \neq V\ Entonces\ \ w :=$ padre de v_k

$Retornar\ \ T = (V', E')$

Apliquemos el algoritmo sobre el grafo que vimos en el slide 3.

Primero, elegimos un orden, digamos $abcdefgh$, de los vértices de G.

Elegimos el primer vértice a y lo etiquetamos como la raíz. Sea $T = (V', E')$ el árbol que comienza siendo $V' = \{a\}$ y $E' = \emptyset$.



A continuación, agregamos a nuestro árbol la arista de la forma (a, x) con x mínimo. En nuestro caso agregamos la arista (a, b).

Repetimos este proceso y agregamos, en orden, las aristas (b, d), (d, c), (c, e), (e, f) y (f, h).

En este momento no podemos agregar una arista de la forma (h, x), así que retrocedemos al padre de h que es f e intentamos agregar una arista de la forma (f, x).

De nuevo, no podemos lograr esto, de modo que vamos al padre de f que es e.

Esta vez podemos agregar la arista (e, g). Por lo que $V' = V$ y termina el algoritmo.

Algoritmo $bfs(V, E)$

$w \coloneqq v_1$

$V' \coloneqq \{v_1\}$

$E' \coloneqq \emptyset$

$Mientras\ V' \neq V\ entonces$

$Mientras$ existe (w, v) con $v \in V - V'\ entonces$

sea v_k el mínimo que cumple con esta condición

$E' \coloneqq E' \cup \{(w, v_k)\}$

$V' \coloneqq V' \cup \{v_k\}$

$w \coloneqq v_k$

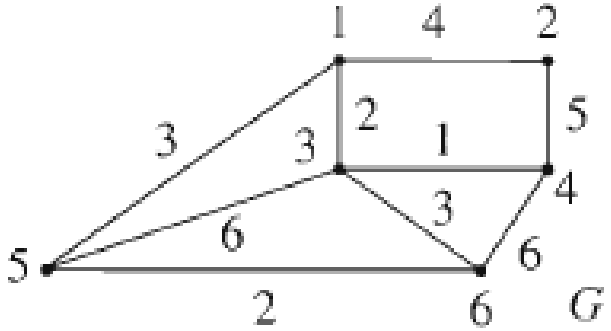
$Si\ V' \neq V\ Entonces$

$w \coloneqq$ padre de v_k

$Retornar\ \ T = (V', E')$

Teoría de Árboles – Árboles de expansión mínimos

El grafo con pesos de la siguiente figura muestra seis ciudades y los costos de construcción de carreteras entre ciertos pares de ellas. Queremos construir el sistema de carreteras de menor costo que una estas seis ciudades.



Teoría de Árboles – Árboles de expansión mínimos

La solución a este problema se puede representar mediante un subgrafo el cual debe ser un árbol de expansión porque:

- debe tener todos los vértices (cada ciudad debe estar en el sistema de carreteras),
- debe ser conexa (para que podamos llegar de una ciudad a cualquier otra),
- debe tener un único camino simple entre cada par de vértices (para que sea de costo mínimo).

Así, lo que estamos necesitando es un árbol de expansión cuya suma de pesos sea mínima. Un árbol de esas características se llama Árbol de expansión Mínimo.

Para resolver esto, vamos a ver dos algoritmos los cuales difieren en la forma en la que se construye el árbol.

Uno comienza, como vienen siendo los anteriores, con el conjunto de vértices del árbol generado como vacío mientras que en el otro comienza con todos los vértices y, sólo va agregando lados. La dificultad que este último trae es que se debe analizar que el lado agregado no forme circuito.

Ambos, al igual que los dos anteriores, toma un orden de vértices.

Veremos cada uno de estos algoritmos en detalle: uno se denomina Prim y, el otro, Kruskal.

Teoría de Árboles – Árboles de expansión mínimos – Prim

El algoritmo de Prim es una variante del algoritmo de Dijkstra que vimos que utiliza etiquetas al igual que las usaba Dijkstra teniendo dos cambios con el anterior:

- La etiqueta **no** representa el costo del camino más corto desde un vértice inicial **sino** el costo del lado más barato, encontrado hasta el momento, que es incidente en ese vértice.
- Necesitamos que el algoritmo devuelva el árbol formado por lo que es preciso que sepamos **quién** fue el que **actualizó** la etiqueta de un vértice dado. Para esto usamos la función $\boldsymbol{A}$ que toma un vértice y nos devuelve el vértice que lo actualizó.

Teoría de Árboles – Árboles de expansión mínimos – Prim

Algoritmo Prim(V, E, v_1)

$w \coloneqq v_1$,
$V' \coloneqq \{v_1\}$
$E' \coloneqq \emptyset$
$T \coloneqq V$
$L(v_1) \coloneqq 0$,
$A(v_1) \coloneqq$ ""
Para todos los vértices $x \neq v_1$ *Hacer* $L(x) \coloneqq \infty$
Mientras $T \neq \emptyset$ *Hacer*
    Elegir $v \in T$ con $L(v)$ mínimo
    $T \coloneqq T - \{v\}$
    $V' \coloneqq V' \cup \{v\}$
    Si $A(v) \neq$ "" *entonces* $E' \coloneqq E' \cup \{(v, A(v))\}$
    Para cada $x \in T$ adyacente a v *Hacer*
        Si $L(x) > w(v,x)$ *entonces*
            $L(x) \coloneqq w(v,x)$
            $A(x) \coloneqq v$
Retornar $T = (V', E')$

El algoritmo de Kruskal tiene como idea la siguiente: el árbol, para que sea mínimo, debe estar formado por los lados para baratos por lo que, se van agregando lados desde los más económicos hasta los más costosos mientras no formen circuito hasta llegar a tener $n-1$ (siendo n la cantidad de vértices del grafo).

Algoritmo Kruskal(V, E, v_1)

$w \coloneqq v_1$

$V' \coloneqq V$

$E' \coloneqq \emptyset$

$Mientras\ |E'| < n - 1 \quad Hacer$

$Elegir\ e \in E$ con $w(e)$ mínimo

$T \coloneqq T - \{v\}$

$V' \coloneqq V' \cup \{v\}$

$Si\ e$ no forma ciclo en $E'\ entonces\ E' \coloneqq E' \cup e$

$Retornar\ \ T = (V', E')$