Algoritmos con grafos

Contenido

Recapitulando

Definición. Un **grafo** (también llamado grafo no dirigido o grafo simple) G consiste en un conjunto de vértices (o nodos) y un conjunto E de aristas (o arcos) tal que cada arista $e \in E$ se asocia con un **par no ordenado** de vértices. Si la arista e está asociada con los vértices v y w, se escribe $e = (v, w)$ o $e = (w, v)$. Es importante notar que ambas son exactamente la misma arista.

Recapitulando

Definición Se dice que una arista $e = (v, w)$ en una grafo es *incidente* sobre v y w. A su vez, se dice que los vértices v y w son incidentes sobre e y que son **adyacentes** (o *vecinos*) entre ellos.

Recapitulando

Definición Se dice que una arista $e = (v, w)$ en una grafo es *incidente* sobre v y w. A su vez, se dice que los vértices v y w son incidentes sobre e y que son **adyacentes** (o *vecinos*) entre ellos.

Ayuda memoria Siempre que hablamos de incidencia estamos hablando de una relación entre un vértice y una arista, y siempre que hablamos de una relación de adyacencia, estamos hablando de una relación entre dos vértices. Cuando dos vértices son adyacentes, podemos decir coloquialmente que son **vecinos**.

Recapitulando

Definición Llamamos **grafo ponderado** o **grafo con pesos** a un grafo $G = (V, E)$ con una función $w : E \to \mathbb{R}$, es decir, un grafo donde a cada arista se le asigna un peso. Notamos $w(i, j)$ al peso de la arista que va de i a j, si esta existe.

Recapitulando

Definición El **grafo completo** de n vértices, notado como K_n, es el grafo simple con n vértices y una arista entre cada par de vértices distintos. A modo de ejemplo, se muestra el grafo K_4.

Recapitulando

Definición Un grafo $G = (V, E)$ es un **bipartito** si existen subconjuntos V_1 y V_2 de V tales que:

1. $V_1 \cap V_2 = \emptyset$
2. $V_1 \cup V_2 = V$
3. Cada arista en E es incidente en un vértice V_1 y un vértice en V_2.

Recapitulando

Definición Un grafo $G = (V, E)$ es un **bipartito** si existen subconjuntos V_1 y V_2 de V tales que:

1. $V_1 \cap V_2 = \emptyset$
2. $V_1 \cup V_2 = V$
3. Cada arista en E es incidente en un vértice V_1 y un vértice en V_2.

Definición El **grafo bipartito completo** de m y n, denotado $K_{m,n}$ vértices es el grafo simple donde el conjunto de vértices puede particionarse en V_1 con m vértices y V_2 con n vértices, y donde el conjunto de aristas consiste en todas las aristas de la forma (v_1, v_2) con $v_1 \in V_1$ y $v_2 \in V_2$.

Recapitulando

Definición Un **grafo conexo** es un grafo donde, dado cualesquiera dos vértices, siempre existe un camino que comienza en uno y termina en el otro. Todos los ejemplos que grafos que hemos visto hasta ahora son conexos. A continuacion, se da un ejemplo de un grafo **no conexo**.

Recapitulando

Definición Un **grafo conexo** es un grafo donde, dado cualesquiera dos vértices, siempre existe un camino que comienza en uno y termina en el otro. Todos los ejemplos que grafos que hemos visto hasta ahora son conexos. A continuacion, se da un ejemplo de un grafo **no conexo**.

Intuitivamente, un grafo conexo es aquel "de una sola pieza". Mientras que los grafos no conexos parecen, a simple vista, estar formado por varias partes. A cada una de estas "partes" se la llama **componente conexa**.

Recapitulando

Definición Sea $G = (V, E)$. Si $U \subseteq V$, el **subgrafo de G inducido por** U es el subgrafo cuyo conjunto de vértices es U y que contiene todas las aristas de G de la forma (v, w) donde $v, w \in U$.

Recapitulando

Definición. Sea un grafo $G = (V, E)$, un **camino** en dicho grafo es una sucesión de vértices $v_0, v_1, ..., v_n$, y aristas $l_0, l_1, ..., l_{n-1}$, tales que $l_i = (v_i, v_{i+1})$ para todo $i \in \{0, 1, ...n\}$.

Recapitulando

Definición. Sea un grafo $G = (V, E)$, un **camino** en dicho grafo es una sucesión de vértices $v_0, v_1, ..., v_n$, y aristas $l_0, l_1, ..., l_{n-1}$, tales que $l_i = (v_i, v_{i+1})$ para todo $i \in \{0, 1, ...n\}$.

Dos vértices están **conectados o son accesibles** si existe un camino que forma una trayectoria para llegar de uno al otro; en caso contrario, los vértices están desconectados o bien son inaccesible

Recapitulando

Definición. Sea un grafo $G = (V, E)$, un **camino** en dicho grafo es una sucesión de vértices $v_0, v_1, ..., v_n$, y aristas $l_0, l_1, ..., l_{n-1}$, tales que $l_i = (v_i, v_{i+1})$ para todo $i \in \{0, 1, ...n\}$.

Dos vértices están **conectados o son accesibles** si existe un camino que forma una trayectoria para llegar de uno al otro; en caso contrario, los vértices están desconectados o bien son inaccesible

Un **ciclo** es un camino de longitud diferente de cero que empieza y termina en el mismo vértice.

Recapitulando

Definición. Sea un grafo $G = (V, E)$, un **camino** en dicho grafo es una sucesión de vértices $v_0, v_1, ..., v_n$, y aristas $l_0, l_1, ..., l_{n-1}$, tales que $l_i = (v_i, v_{i+1})$ para todo $i \in \{0, 1, ...n\}$.

Dos vértices están **conectados o son accesibles** si existe un camino que forma una trayectoria para llegar de uno al otro; en caso contrario, los vértices están desconectados o bien son inaccesible

Un **ciclo** es un camino de longitud diferente de cero que empieza y termina en el mismo vértice.

Nota Por lo general, estamos interesados en caminos y ciclos **simples** es decir, que no repiten aristas.

Recapitulando

Definición. Sea un grafo $G = (V, E)$, un **camino** en dicho grafo es una sucesión de vértices $v_0, v_1, ..., v_n$, y aristas $l_0, l_1, ..., l_{n-1}$, tales que $l_i = (v_i, v_{i+1})$ para todo $i \in \{0, 1, ...n\}$.

Dos vértices están **conectados o son accesibles** si existe un camino que forma una trayectoria para llegar de uno al otro; en caso contrario, los vértices están desconectados o bien son inaccesible

Un **ciclo** es un camino de longitud diferente de cero que empieza y termina en el mismo vértice.

Nota Por lo general, estamos interesados en caminos y ciclos **simples** es decir, que no repiten aristas.

En un grafo sin pesos, la **longitud de un camino** es la cantidad de aristas por las que pasamos. En un grafo con pesos, la longitud del camino es la suma de los pesos de las aristas por las que pasamos.

Recapitulando

El **grado de un vértice** v, que denotamos como $\delta(v)$ es el número de aristas que inciden en v.

Contenido

Edsger Dijkstra

Edsger W. Dijkstra (1930-2002) fue un científico de la computación holandés que ideo un algoritmo que resuelve el problema de la ruta más corta de forma óptima.
Además, fue de los primeros en proponer la programación como una ciencia.
Ganador del premio Turing en 1972. El premio Turing es "el Nobel de la programación".
Poco después de su muerte recibió también el premio de la ACM.
El premio pasó a llamarse Premio Dijkstra el siguiente año en su honor.

El algoritmo de Dijkstra

Este algoritmo Dijkstra **encuentra la longitud de una ruta más corta** del vértice a al vértice z en un **grafo** $G = (V, E)$ **ponderado y conexo**. El peso de la arista (i, j) es $w(i, j) > 0$ (es decir, requerimos que los pesos sean siempre positivos).

Es importante verificar que estas condiciones se cumplen antes de aplicar el algoritmo.

El algoritmo de Dijkstra

El algoritmo funciona asignando **distancias** a los vértices, que siempre son relativas al vértice inicial. Notaremos a la distancia del vértice a al vértice v como $D(v)$.

El algoritmo de Dijkstra

El algoritmo funciona asignando **distancias** a los vértices, que siempre son relativas al vértice inicial. Notaremos a la distancia del vértice a al vértice v como $D(v)$.

Algunas distancias son temporales y otras son permanentes. Al ilustrar el algoritmo, por lo general se encierran en un círculo, o se pintan de un color distinto los vértices cuya distancia calculada ya es permanente.

El algoritmo de Dijkstra

El algoritmo funciona asignando **distancias** a los vértices, que siempre son relativas al vértice inicial. Notaremos a la distancia del vértice a al vértice v como $D(v)$.

Algunas distancias son temporales y otras son permanentes. Al ilustrar el algoritmo, por lo general se encierran en un círculo, o se pintan de un color distinto los vértices cuya distancia calculada ya es permanente.

Llamaremos T al conjunto de vértices que aún no fueron visitados (y, por lo tanto, tienen una distancia temporal). Esos son los vértices con lo que "tenemos que seguir trabajando". La idea es que, una vez $D(v)$ sea la etiqueta definitiva de un vértice v, $D(v)$ sea la distancia más corta desde a hasta v.

El algoritmo de Dijkstra

El algoritmo funciona asignando **distancias** a los vértices, que siempre son relativas al vértice inicial. Notaremos a la distancia del vértice a al vértice v como $D(v)$.

Algunas distancias son temporales y otras son permanentes. Al ilustrar el algoritmo, por lo general se encierran en un círculo, o se pintan de un color distinto los vértices cuya distancia calculada ya es permanente.

Llamaremos T al conjunto de vértices que aún no fueron visitados (y, por lo tanto, tienen una distancia temporal). Esos son los vértices con lo que "tenemos que seguir trabajando". La idea es que, una vez $D(v)$ sea la etiqueta definitiva de un vértice v, $D(v)$ sea la distancia más corta desde a hasta v.

Al inicio, todos los vértices estarán en la lista de no visitados. Cada iteración del algoritmo visita un nuevo nodo, determinando su distancia definitiva. El algoritmo termina cuando z es visitado. Cuando llega ese momento, $D(z)$ es el valor de la distancia del camino más corto desde a hasta z.

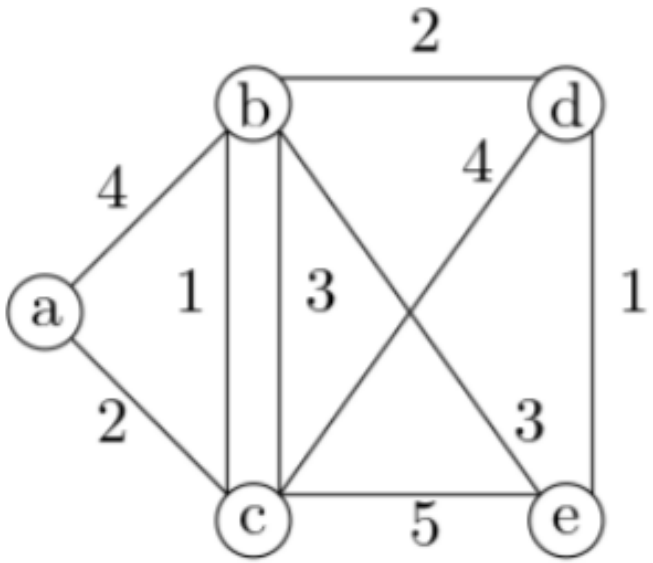
El algoritmo de Dijkstra

Los pasos detallados del **algoritmo de Dijkstra** son los siguientes:

1. Primero, notemos que la ruta más corta del vértice a al vértice a, es la ruta de cero aristas, que tiene peso 0. Así, inicializamos $D(a) = 0$.
2. No sabemos aún el valor de una ruta más corta de a a los otros vértices, entonces para cada vértice $v \neq a$, inicializamos $D(v) = \infty$.
3. Inicializamos el conjunto T como el conjunto de todos los vértices, i.e. $T = V$.
4. Seleccionamos un vértice $v \in T$ tal que $D(v)$ sea mínimo.
5. Quitamos el vértice v del conjunto T: $T = T - v$
6. Para cada $t \in T$ adyacente a v, actualizamos su etiqueta: $D(t) = \text{MIN}\{D(t), D(v) + w(v, t)\}$
7. Si $z \in T$, repetimos desde el paso 4, si no, hemos terminado y $D(z)$ es el valor de la ruta mas corta entre a y z.

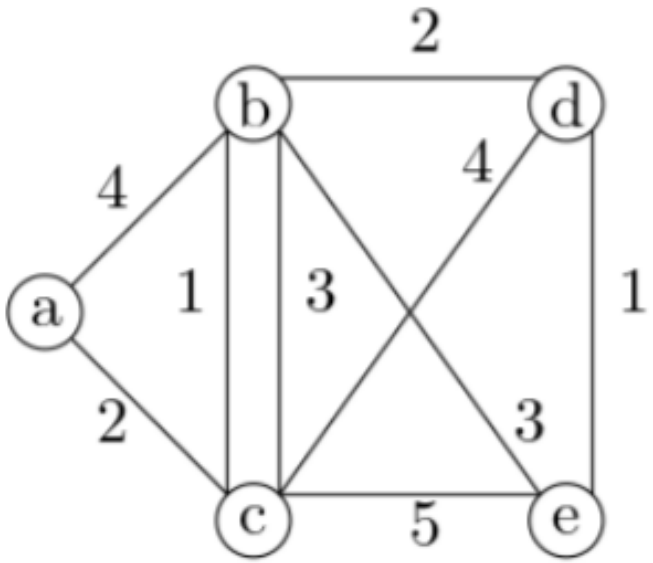
El algoritmo Dijkstra - ejemplo

Utilizaremos el algoritmo de Dijkstra para encontrar el tamaño de la ruta más corta de *a* a *e* en el siguiente grafo:



El algoritmo Dijkstra - ejemplo

Utilizaremos el algoritmo de Dijkstra para encontrar el tamaño de la ruta más corta de *a* a *e* en el siguiente grafo:



Distancias

$a = 0$

$b = 3$

$c = 2$

$d = 5$

$e = 6$

Contenido

Arboles

Los **árboles** son un tipo especial de **grafo**. La definición que dimos en unidades anteriores se puede expresar ahora en términos de teoría de grafos.

Arboles

Los **árboles** son un tipo especial de **grafo**. La definición que dimos en unidades anteriores se puede expresar ahora en términos de teoría de grafos.

Definición. Un **árbol** (libre) T es una grafo simple que satisface lo siguiente: si v y w son vértices en T existe un camino único de v a w.

Arboles

Los **árboles** son un tipo especial de **grafo**. La definición que dimos en unidades anteriores se puede expresar ahora en términos de teoría de grafos.

Definición. Un **árbol** (libre) T es una grafo simple que satisface lo siguiente: si v y w son vértices en T existe un camino único de v a w.

Recordemos también que llamamos **árbol con raíz** a un árbol donde se ha designado un nodo especial, llamado raíz.

Arboles

Los **árboles** son un tipo especial de **grafo**. La definición que dimos en unidades anteriores se puede expresar ahora en términos de teoría de grafos.

Definición. Un **árbol** (libre) T es una grafo simple que satisface lo siguiente: si v y w son vértices en T existe un camino único de v a w.

Recordemos también que llamamos **árbol con raíz** a un árbol donde se ha designado un nodo especial, llamado raíz.

Ahora que sabemos teoría de grafos podemos, además, dar algunas propiedades adicionales de árboles.

Arboles

Los **árboles** son un tipo especial de **grafo**. La definición que dimos en unidades anteriores se puede expresar ahora en términos de teoría de grafos.

Definición. Un **árbol** (libre) T es una grafo simple que satisface lo siguiente: si v y w son vértices en T existe un camino único de v a w.

Recordemos también que llamamos **árbol con raíz** a un árbol donde se ha designado un nodo especial, llamado raíz.

Ahora que sabemos teoría de grafos podemos, además, dar algunas propiedades adicionales de árboles.

Propiedad Un grafo conexo y sin ciclos es un árbol. En efecto, como el grafo es conexo entre dos vértices cualesquiera siempre hay un camino.

Arboles

Los **árboles** son un tipo especial de **grafo**. La definición que dimos en unidades anteriores se puede expresar ahora en términos de teoría de grafos.

Definición. Un **árbol** (libre) T es una grafo simple que satisface lo siguiente: si v y w son vértices en T existe un camino único de v a w.

Recordemos también que llamamos **árbol con raíz** a un árbol donde se ha designado un nodo especial, llamado raíz.

Ahora que sabemos teoría de grafos podemos, además, dar algunas propiedades adicionales de árboles.

Propiedad Un grafo conexo y sin ciclos es un árbol. En efecto, como el grafo es conexo entre dos vértices cualesquiera siempre hay un camino.

Propiedad Un grafo conexo con n vértices y $n-1$ aristas es necesariamente un árbol.

Arboles

Los **árboles** son un tipo especial de **grafo**. La definición que dimos en unidades anteriores se puede expresar ahora en términos de teoría de grafos.

Definición. Un **árbol** (libre) T es una grafo simple que satisface lo siguiente: si v y w son vértices en T existe un camino único de v a w.

Recordemos también que llamamos **árbol con raíz** a un árbol donde se ha designado un nodo especial, llamado raíz.

Ahora que sabemos teoría de grafos podemos, además, dar algunas propiedades adicionales de árboles.

Propiedad Un grafo conexo y sin ciclos es un árbol. En efecto, como el grafo es conexo entre dos vértices cualesquiera siempre hay un camino.

Propiedad Un grafo conexo con n vértices y $n-1$ aristas es necesariamente un árbol.

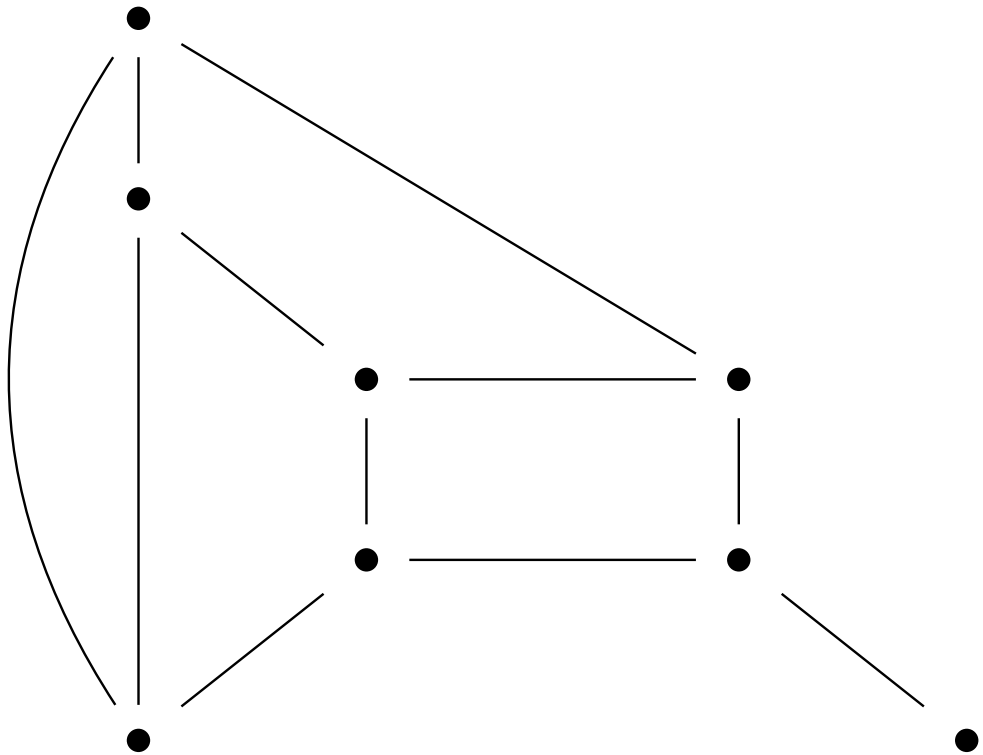
Propiedad Si un grafo con n vértices y $n-1$ aristas, no contiene ciclos entonces necesariamente es un árbol.

Arboles

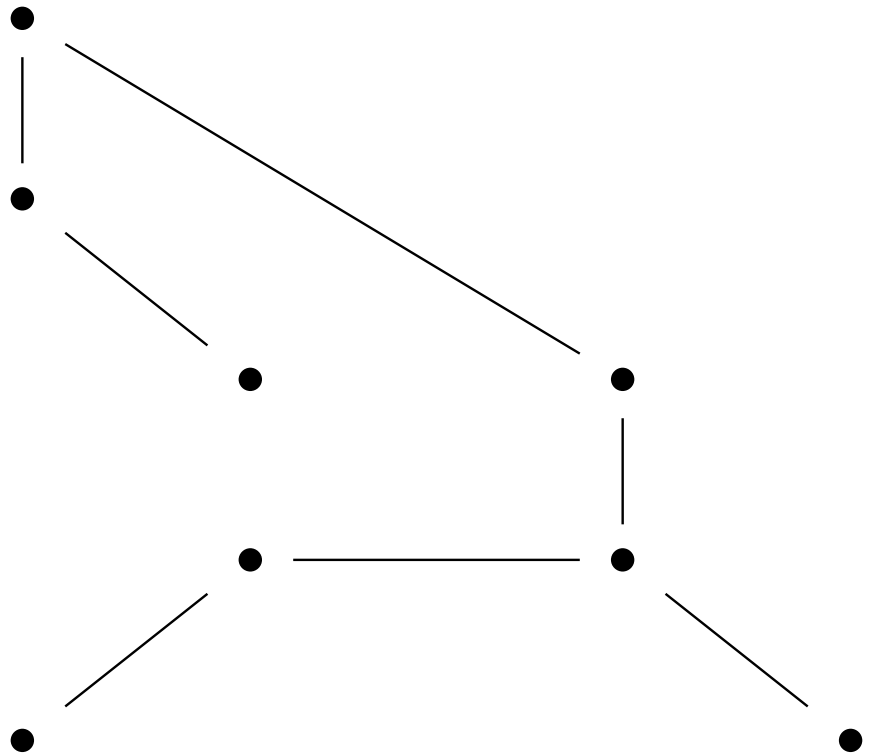
Se dice que árbol T es un **árbol de expansión** de un grafo G si:

1. T es un subrafo de G.
2. T es un árbol.
3. T contiene todos los vértices de G.

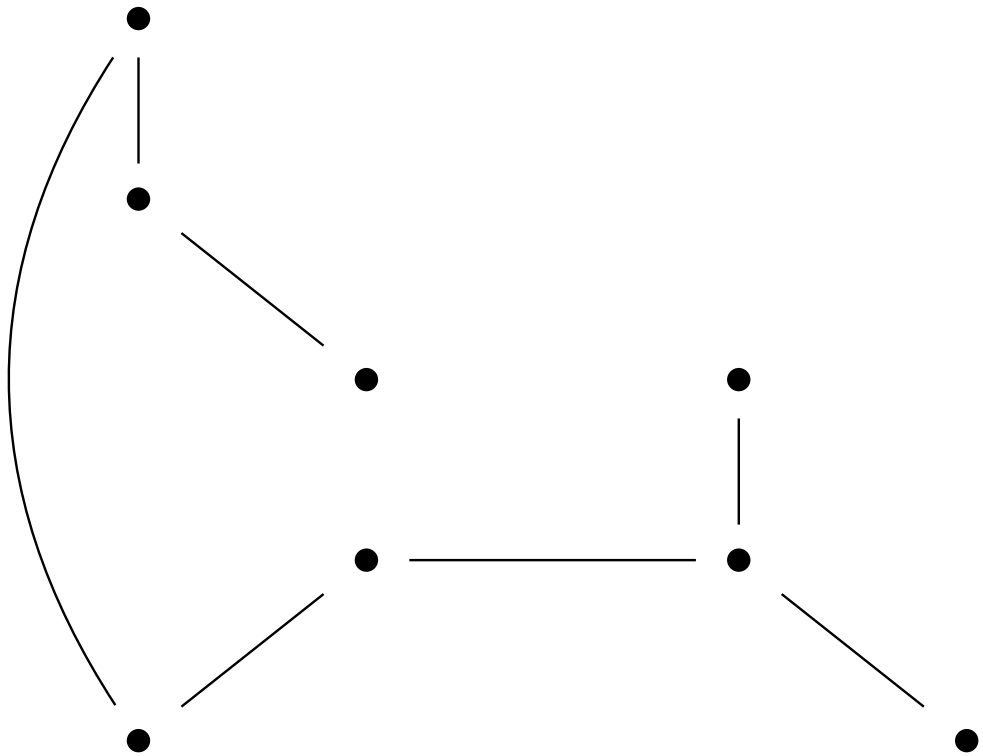
Por ejemplo, dado este grafo



Este es un árbol de expansión de ese grafo:



Nota El árbol de expansión de un grafo en general no es único, por ejemplo, acá mostramos otro árbol de expansión distinto del mismo grafo.



Un árbol de expansión para un grafo existe sí y sólo sí el grafo es conexo.

Propiedad

En efecto, todos los árboles pueden pensarse como grafos conexos sin ciclos. Si un grafo G tiene un árbol de expansión T, entonces entre dos vértices cualesquiera existe un camino en el árbol que los une (pues todos los árboles son conexos). Ahora bien, como T es subgrafo de G, necesariamente debe existir ese mismo camino en el grafo G.

Propiedad

En efecto, todos los árboles pueden pensarse como grafos conexos sin ciclos. Si un grafo G tiene un árbol de expansión T, entonces entre dos vértices cualesquiera existe un camino en el árbol que los une (pues todos los árboles son conexos). Ahora bien, como T es subgrafo de G, necesariamente debe existir ese mismo camino en el grafo G.

El algoritmo de búsqueda en profundidad, normalmente llamado DFS por sus siglas en inglés *Depth First Search* permite encontrar el árbol de expansión de un grafo conexo. Este algoritmo puede verse como versión generalizada del recorrido en pre-orden.

Propiedad

La estrategia consiste en partir de un vértice determinado v y a partir de allí, cuando se visita un nuevo vértice, explorar cada camino que salga de él. Un camino deja de explorarse cuando se llega a un vértice ya visitado. Si existen vértices no alcanzables, el recorrido queda incompleto; entonces, se debe "volver hacia atrás" hasta encontrar un vértice donde hayamos dejado un camino sin explorar, y repetir el proceso.

Propiedad

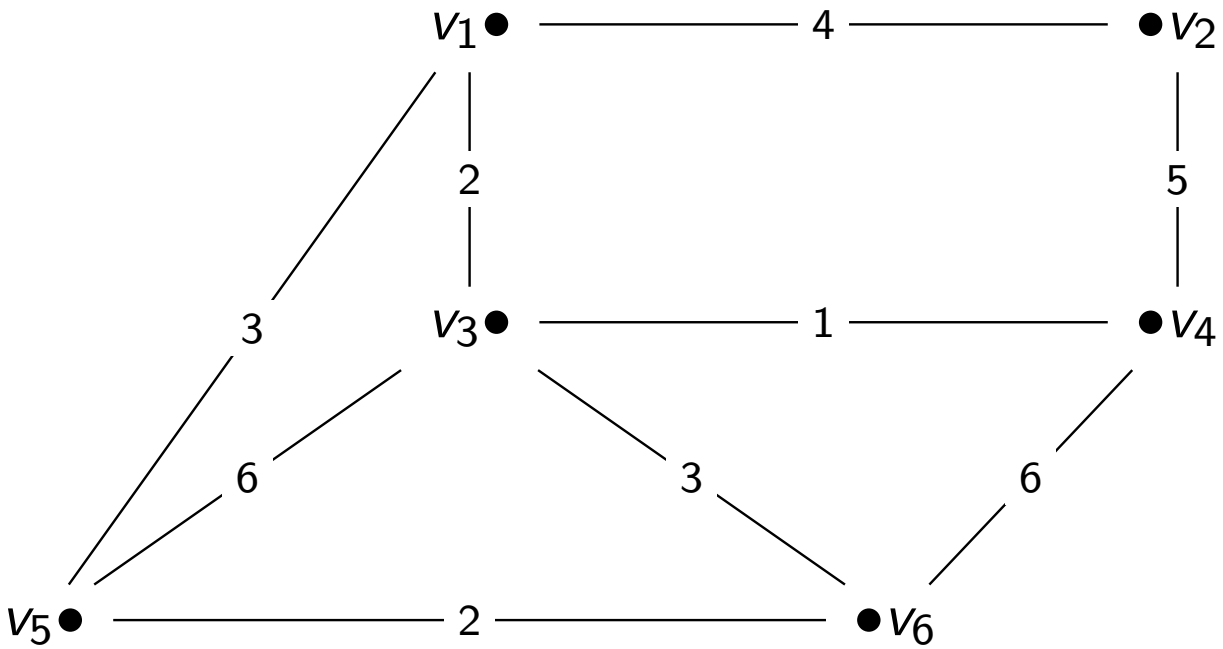
La estrategia consiste en partir de un vértice determinado v y a partir de allí, cuando se visita un nuevo vértice, explorar cada camino que salga de él. Un camino deja de explorarse cuando se llega a un vértice ya visitado. Si existen vértices no alcanzables, el recorrido queda incompleto; entonces, se debe "volver hacia atrás" hasta encontrar un vértice donde hayamos dejado un camino sin explorar, y repetir el proceso.

Se utiliza, además, un orden sobre los vértices para desempatar en casos donde tengamos más de una opción. Normalmente, como convención interna, utilizaremos el orden alfabético cuando los nodos lleven letras por nombres, o el orden natural cuando los vértices sean numerados.

1. Dado el grafo $G = (V, E)$, inicializamos el árbol $T = (V', E')$ con $V' = \{v_1\}$ y $E' = \emptyset$. La idea sera ir agregando aristas a E' a medida que lo necesitemos. Definimos además una variable $w = v_1$ que llevará el nodo donde estamos parados actualmente.
2. Mientras exista v tal que (w, v) es una arista que al agregarla a T no genera un ciclo, realizamos lo siguiente:
    1. Elegimos la arista (w, v_k) con k mínimo tal que al agregarla a T no genera un ciclo.
    2. Agregamos la arista (w, v_k) a E'.
    3. Agregamos v_k a V'
    4. Actualizamos $w = v_k$
3. Si $V' = V$ hemos terminado y T y T es un árbol de expansión del grafo G. Si $w = v_1$, el grafo es disconexo, y por lo tanto jamás podremos encontrar un árbol de expansión para el mismo. Si no se da ninguna de las dos situaciones, actualizamos el valor de w para que sea el padre de w en el árbol T, y repetimos desde el paso 2. Dar este paso hacia atrás nos obligará a explorar otros caminos.

Árboles de expansión mínima

El grafo con pesos de la figura muestra seis ciudades y los costos de construir carreteras entre ellas. Se desea construir el sistema de carreteras de menor costo que conecte a las seis ciudades. La solución debe necesariamente ser un árbol de expansión ya que debe contener a todos los vértices y para ser de costo mínimo, seria redundante tener dos caminos entre ciudades. Entonces lo que necesitamos es el árbol de expansión del grafo que sea de peso mínimo.



Árboles de expansión mínima

Definición Sea G un grafo con pesos. Un árbol de expansión mínima de G es un árbol de expansión de G que tiene peso mínimo entre todos los posibles.

Nota El algoritmo DFS no asegura que el árbol encontrado sea de peso mínimo.

El algoritmo de Prim

El **algoritmo de Prim** permite encontrar un **árbol de expansión mínimo** para un grafo con pesos conexo de vértices $v_1, v_2, ...v_n$. El algoritmo incrementa continuamente el tamaño de un árbol, de manera similar al algoritmo DFS, 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 está completamente construido cuando no quedan más vértices por agregar.

El algoritmo de Prim

Definimos $w(i, j)$ como el peso de la arista que une los vértices i, j si existe, o como ∞ si la misma no existe. Además, llevamos la cuenta en un diccionario *agregado* cuyas claves son los vértices y cuyas valores son *True* si el vértice fue agregado al árbol de expansión mínima, y *False* si aún no ha sido agregado. También iremos actualizando el diccionario E' de las aristas del árbol.

El algoritmo de Prim

1. Inicializamos el diccionario *agregado*, seteando todos los vértices a *False* (es decir, ningún vértice ha sido agregado aún.)
2. Agregamos el primer vértice al árbol $agregado[v_1] = True$.
3. Inicializamos la lista de aristas que compondrán el árbol como un conjunto vacio: $E = \emptyset$.
4. Para cada i en el rango $1, ..., n-1$, agregamos la arista de peso mínimo que tiene un vértice que ya fue agregado y un vértice que aún no fue agregado, esto lo hacemos del siguiente modo:
    1. Definimos la variable temporal $min = \infty$.
    2. Para cada j en el rango $(1, ...n)$:
        1. Si $agregado[v_j] == True$, el vértice v_j ya esta en el árbol:
        2. Para cada k en el rango de $(1, ..., n)$:

           Si $agregado[v_k] == False$ y además $w(j,k) < min$, el vértice v_k será el *candidato* a ser agregado al árbol, y la arista (j,k) sera la arista candidata a agregar al árbol.
    3. Al finalizar el for, agregamos el vértice candidato al árbol actualizando el diccionario agregado, y además agregamos la arista candidata al conjunto de aristas.

El algoritmo de Prim

Utilizaremos el algoritmo de Prim para encontrar el árbol de expansión mínimo para el siguiente árbol: