Programación II

Tecnicatura Universitaria en Inteligencia Artificial

2023

Teoría de grafos

1. Algoritmo de Dijkstra

Un problema muy recurrente es encontrar el camino más corto entre dos vértices dados (es decir, un camino que tiene la longitud mínima entre todas los posibles caminos entre esos dos vértices.)

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[1]. El premio pasó a llamarse Premio Dijkstra el siguiente año en su honor.

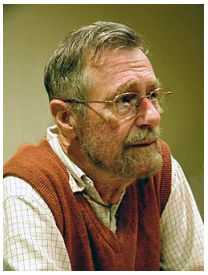


Figura 1: Edsger Wybe Dijkstra

Este algoritmo 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.

Como el objetivo es encontrar la ruta de longitud mínima, el algoritmo Dijkstra funciona asignando **distancias** a los vértices, que siempre son relativas al vértice inicial. Notaremos a la distancia del vértice a al vértice v como $D(v)$ (no es necesario mencionar a a porque es fijo durante una ejecución del algoritmo). 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. Los pasos detallados 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: $T = T - v$

[1] Association for Computing Machinery

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.

Ejemplo. En la Figura 2 se muestra un ejemplo de la ejecución del algoritmo de Dijkstra en 10 pasos. El objetivo es encontrar la longitud de la ruta más corta desde el vértice a hacia el vértice e (en este caso, 6). En violeta se marcan los nodos que son visitados en cada paso, mientras que en verde se señalan las aristas que se consideran para (quizás) actualizar las distancias hacia los nodos adyacentes.

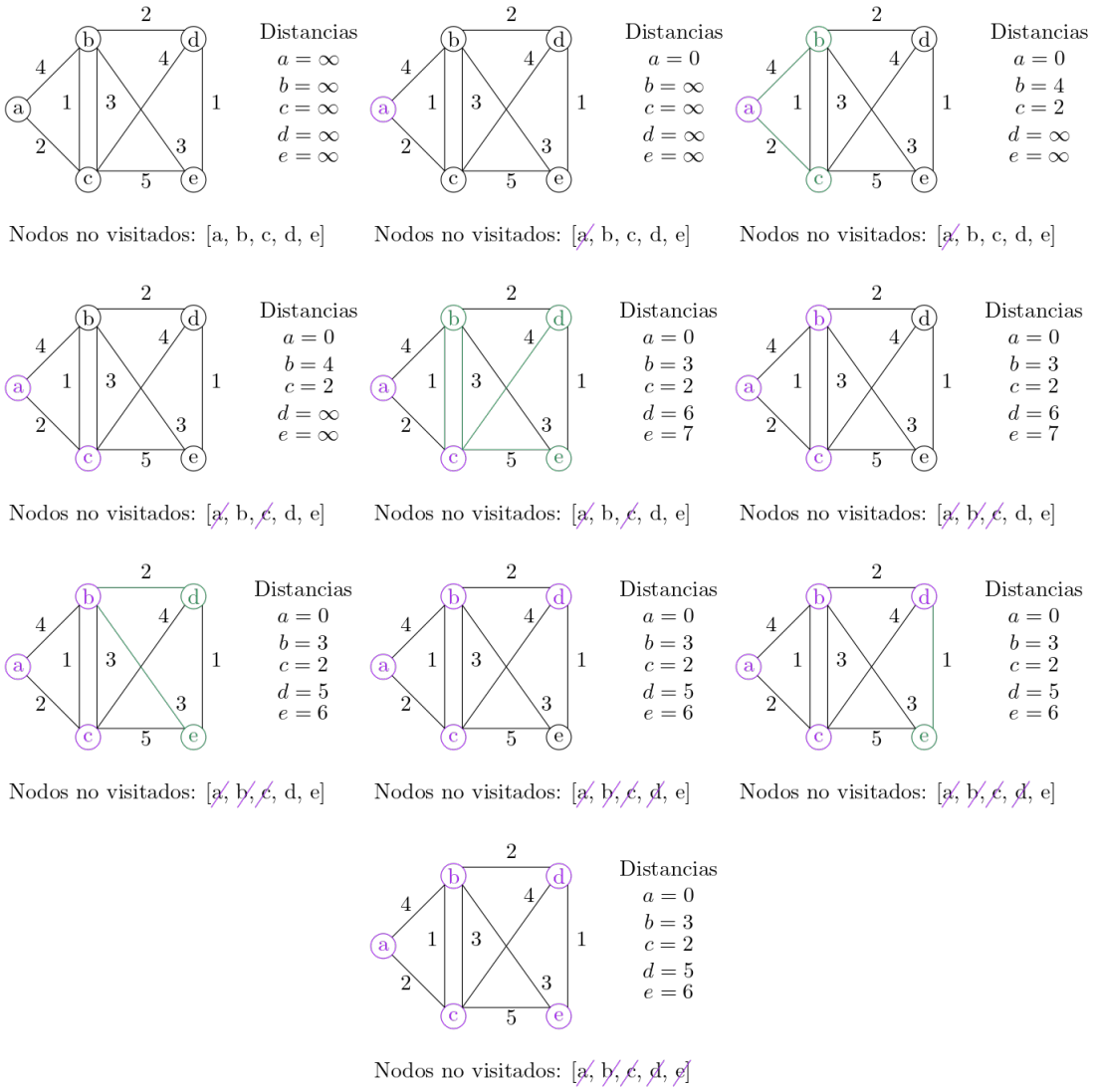


Figura 2: Ejecución del algoritmo de Dijkstra

2. Árboles

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.

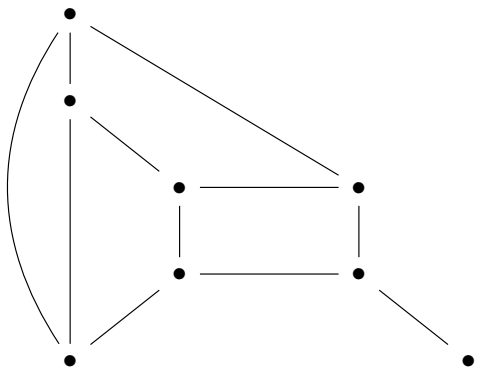
Propiedad Si un grafo con n vértices y $n-1$ aristas, no contiene ciclos entonces necesariamente es un árbol.

2.1. Árboles de expansión

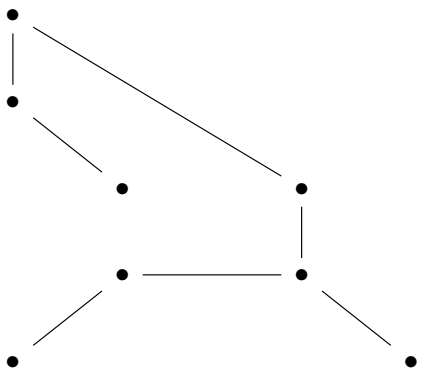
Se dice que árbol T es un **árbol de expansión** de un grafo G si:

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

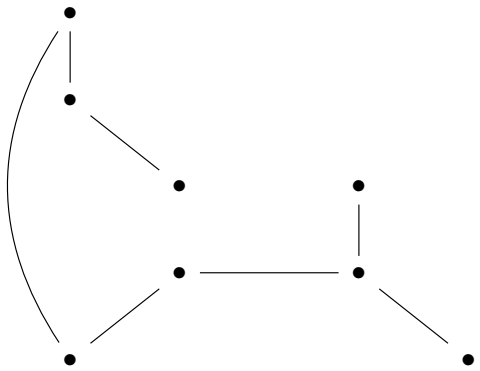
Por ejemplo, dado este grafo



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 si y solo si el grafo es conexo.

En efecto, todos los arboles 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 arboles 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. 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. Dado un orden $v_1, v_2, ..., v_n$ de los vértices, el algoritmo puede expresarse formalmente como sigue:

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:

   a) Elegimos la arista (w, v_k) con k mínimo tal que al agregarla a T no genera un ciclo.

   b) Agregamos la arista (w, v_k) a E'.

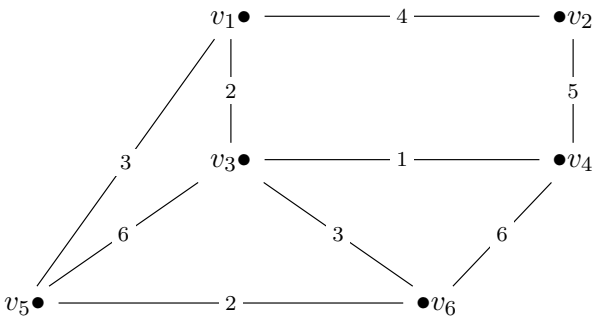
   c) Agregamos v_k a V'

   d) 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.

2.2. Árbol de expansión mínima

Ejemplo 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, sería redundante tener dos caminos entre ciudades. Entonces lo que necesitamos es el árbol de expansión del grafo que sea de peso mínimo.



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 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.

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 puede ser descrito formalmente como sigue:

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:

   a) Definimos la variable temporal $min = \infty$.

   b) 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.

   c) Al finalizar el for, agregamos el vértice candidato al árbol actualizando el diccionario agregado, y además agregamos la arista candidata al conjunto de aristas.

Ejemplo. En la Figura 3 se muestra un ejemplo de la ejecución del algoritmo Prim en 12 pasos. El objetivo es encontrar encontrar un árbol de expansión mínimo. En violeta se marcan los nodos y aristas que son agregados al árbol en cada paso, mientras que en verde se señalan las aristas candidatas que van surgiendo.

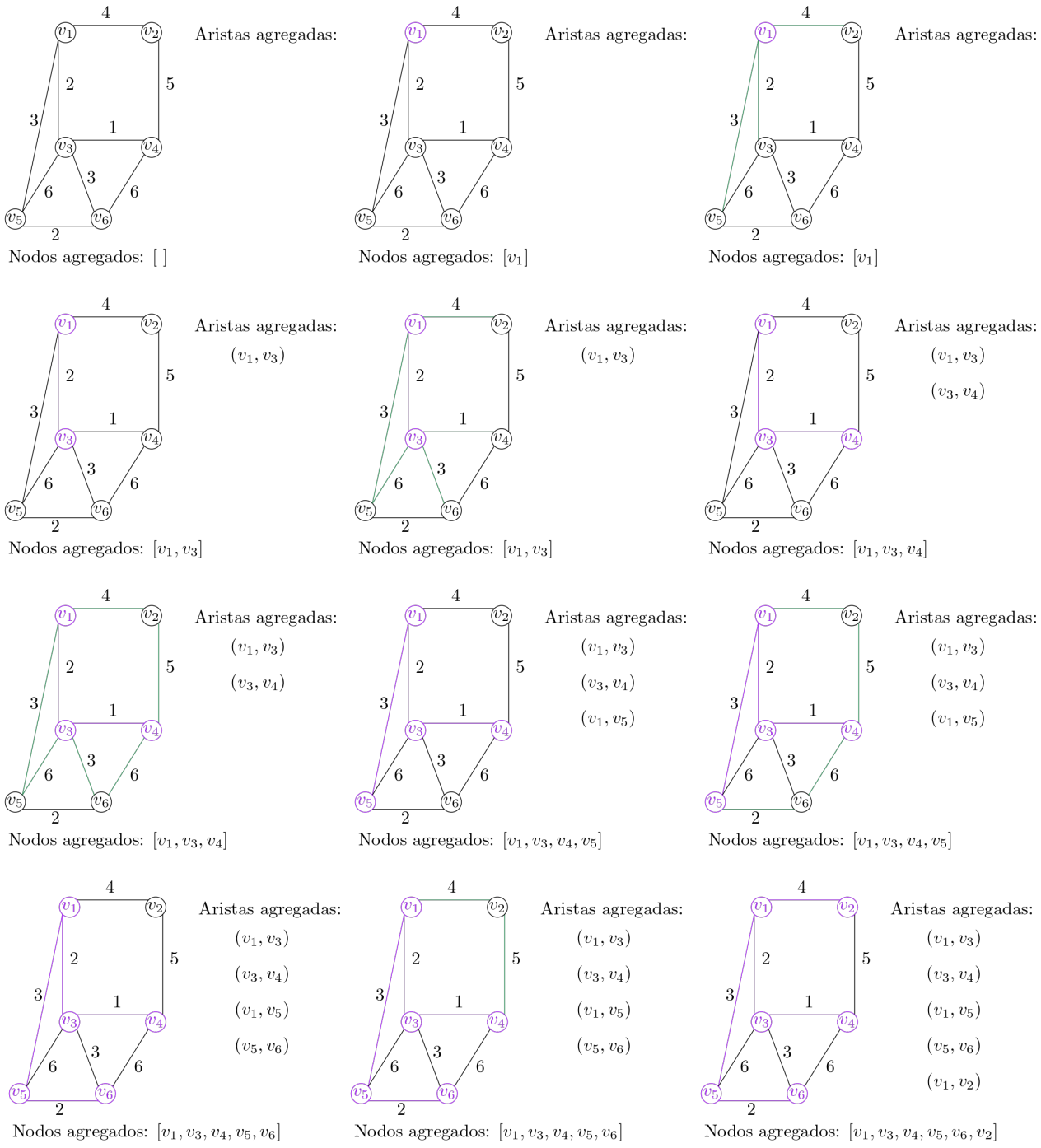
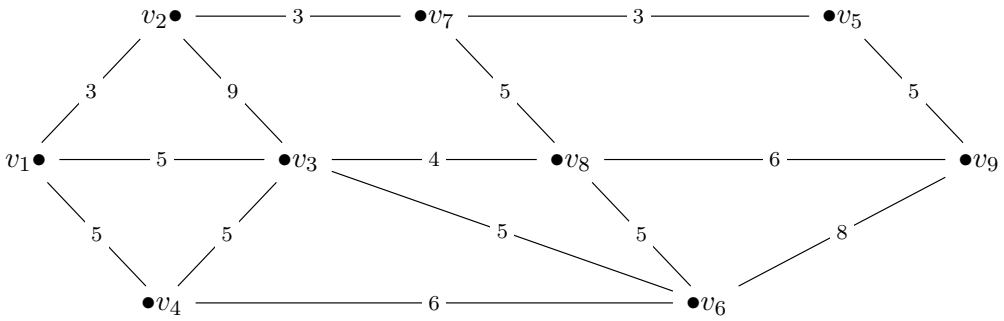


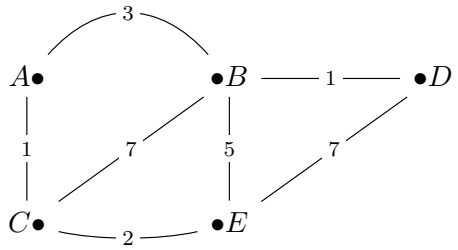
Figura 3: Ejecución del algoritmo Prim

3. Ejercicios - algoritmos

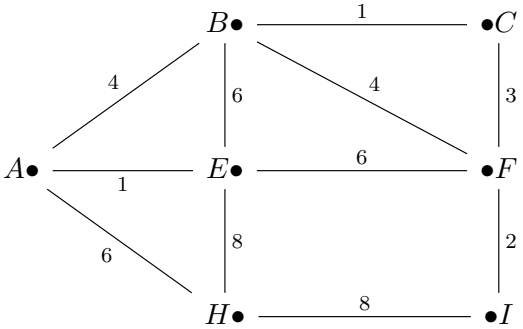
1. Para el siguiente grafo, encontrar la longitud de la ruta más corta del vértice v_1 al vértice v_9 utilizando el algoritmo de Dijkstra y el árbol de expansión mínimo utilizando el algoritmo de Prim. Describir de manera detallada los algoritmos y sus pasos.



2. Para el siguiente rgafo, encuentre la longitud de la ruta más corta del vértice C al vértice E utilizando el algoritmo de Dijkstra y el árbol de expansión mínimo utilizando el algoritmo de Prim. Describir de manera detallada los algoritmos y sus pasos.



3. (1 pt.) Para el siguiente grafo, encontrar la longitud del camino más corto del vértice A al vértice I, utilizando el algoritmo de DIjkstra. Describir de manera detallada el algoritmo y sus pasos.



4. Describa que es un árbol de expansión de un grafo. ¿Que propiedad debe cumplirse para que el árbol de expansión exista? ¿Cuando podemos decir que el árbol de expansión del grafo es mínimo?