Matemáticas discretas II

Septiembre 2022

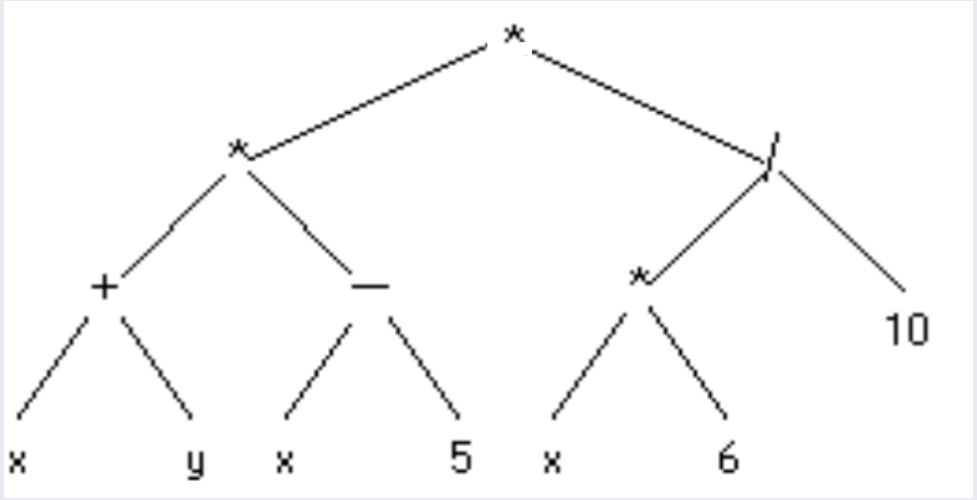
Contenido

Contenido

Recorridos en árboles

Expresiones aritméticas

Las expresiones matemáticas pueden ser representadas usando árboles:



- En preorden: (* (* (+ x y) (- x 5)) (/ (* x 6) 10))
- En inorden: ((x + y) * (x - 5)) * ((x * 6) / 10))
- En postorden: (((x y +) (x 5 -) *) ((x 6 *) 10 /) *)

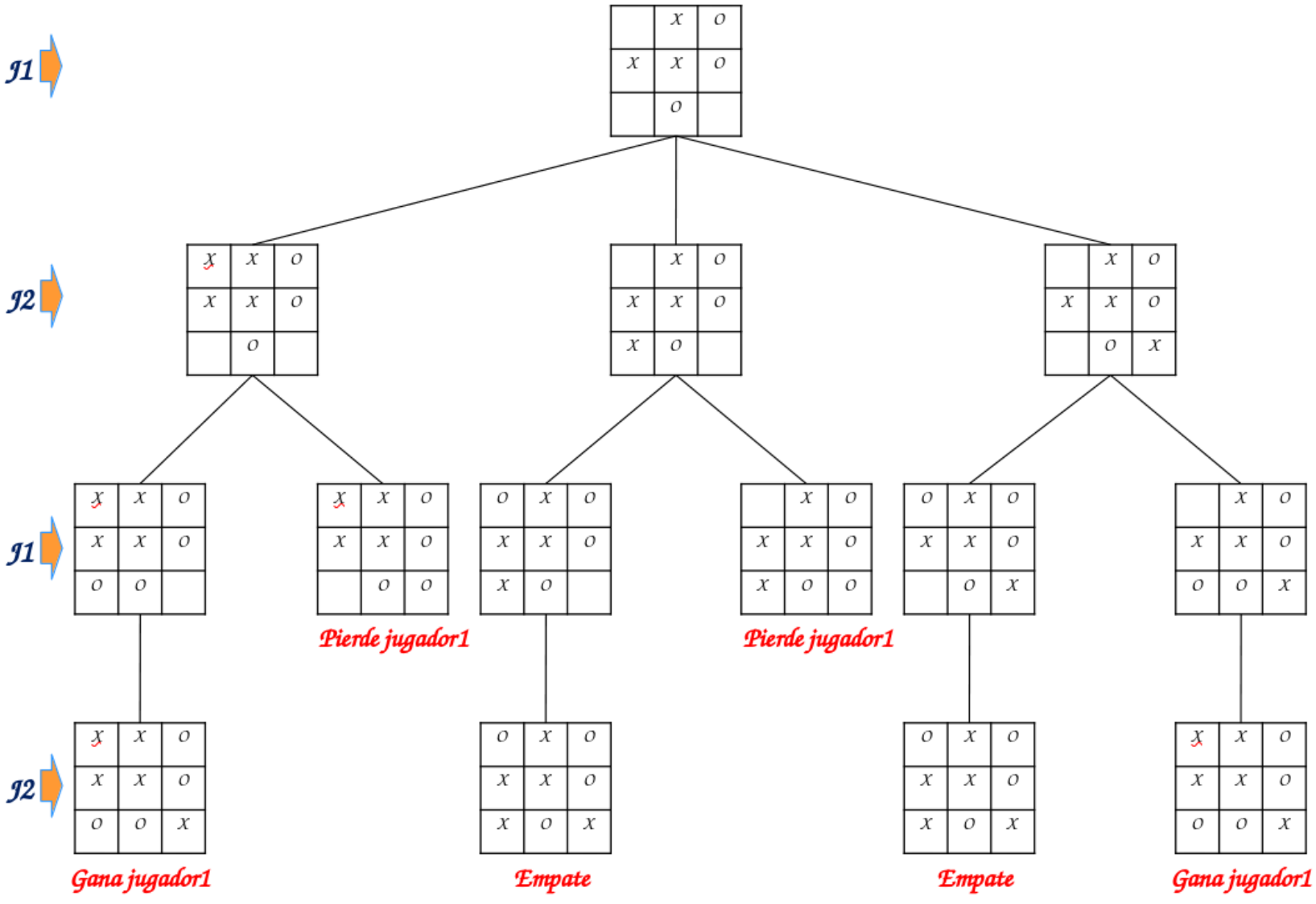
Contenido

Árboles de Juego

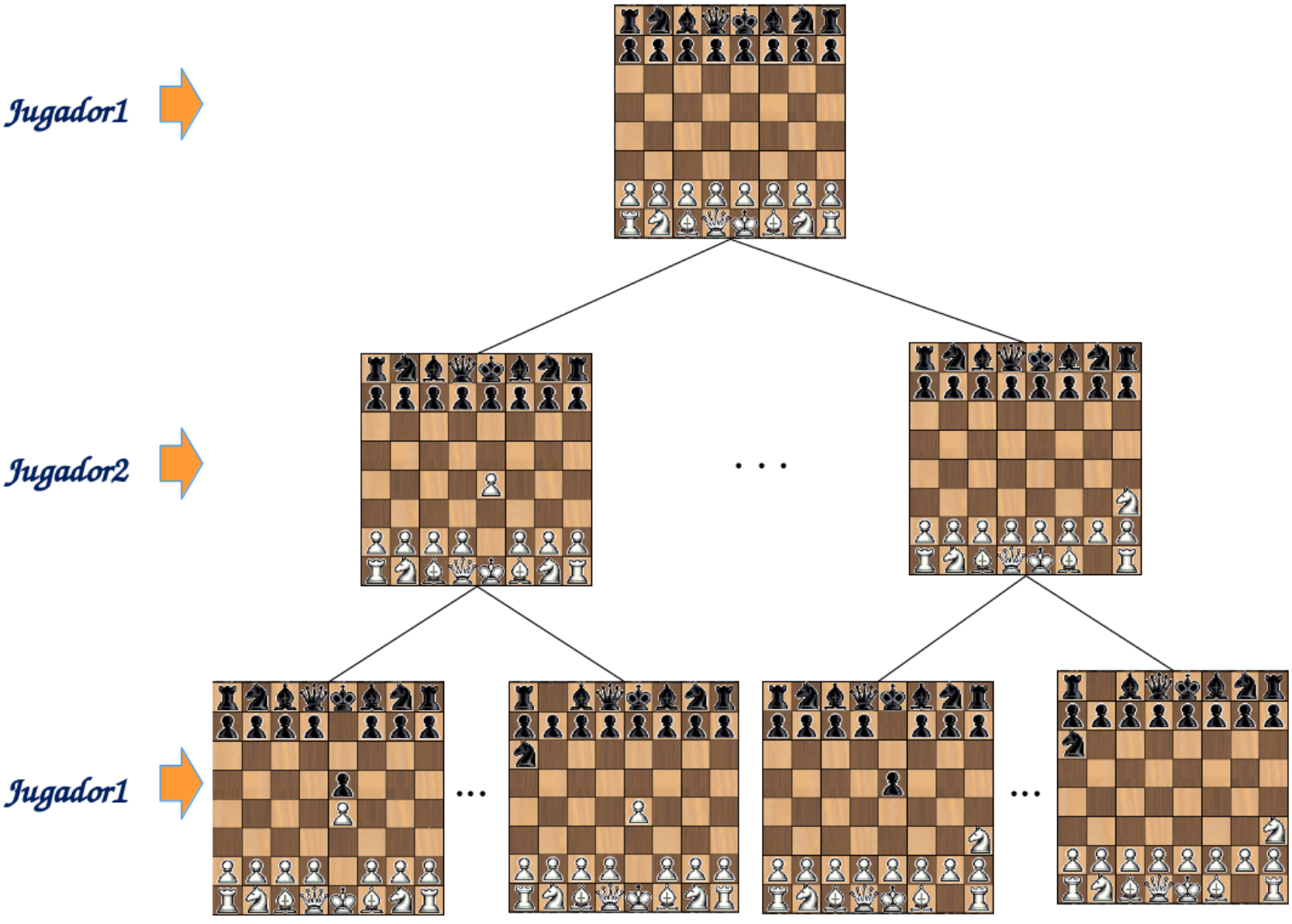
Definición

- Se modelan los posibles movimientos de un jugador en un juego
- Sirve para analizar el efecto de las jugadas
- Considera el cambio de estado de un juego al realizar la jugadas, esta información debe almacenarse en la representación del nodo
- Pueden considerar 1 o más jugadores
- Es útil para generar *jugadores inteligentes* contrincantes de jugadores humanos.

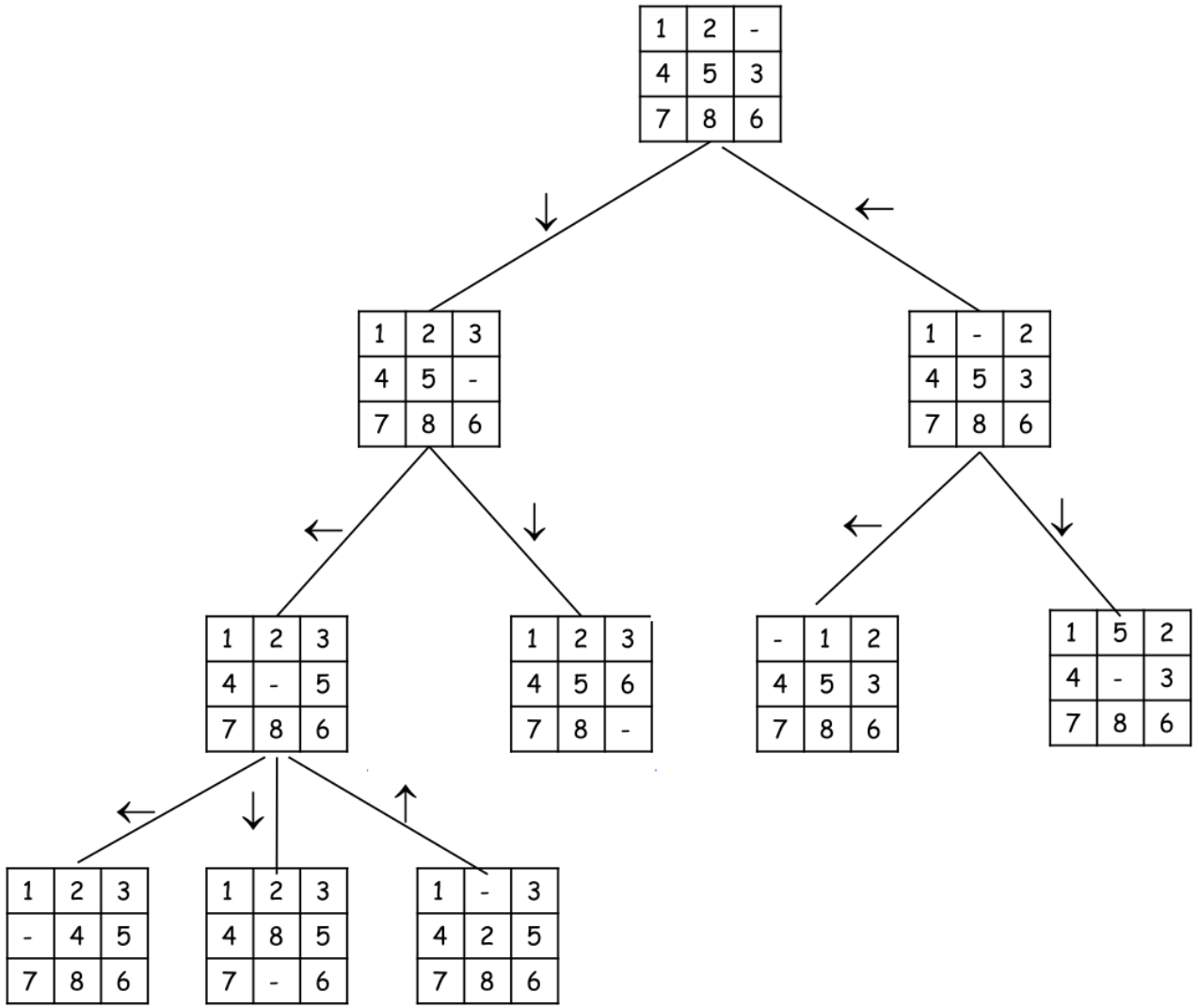
Árboles de Juego



Árboles de Juego



Árboles de Juego



Contenido

Contenido

Árboles de expansión

Definición

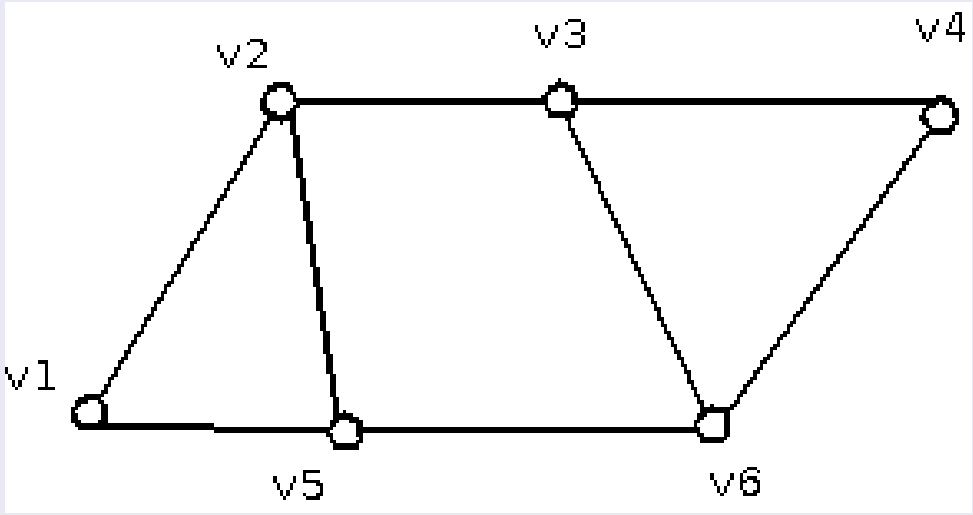
Es un problema que está asociado a como obtener un árbol expansión para un grafo. Este árbol contiene todos los nodos del grafo y algunas de sus aristas para asegurar conectividad Un árbol

de expansión de un **grafo conexo** G=(V,E) es un árbol que tiene el conjunto de nodos N y es subgrafo de G. Esto es, un árbol de expansión es conexo, a cíclico y tiene a todo N y a parte de A como un conjunto de aristas.

Árboles de expansión

Obtención del árbol

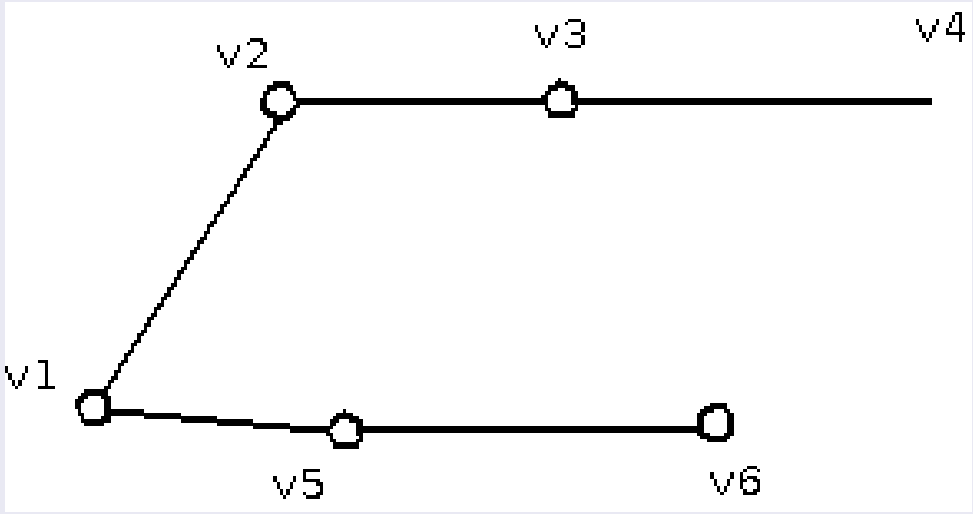
Hay muchas formas de obtener un árbol de expansión. Al empezar a borrar aristas para borrar los ciclos, por ejemplo:



Árboles de expansión

Obtención del árbol

Al eliminar $\{(v1, v5), (v3, v6)(v4, v6)\}$



Contenido

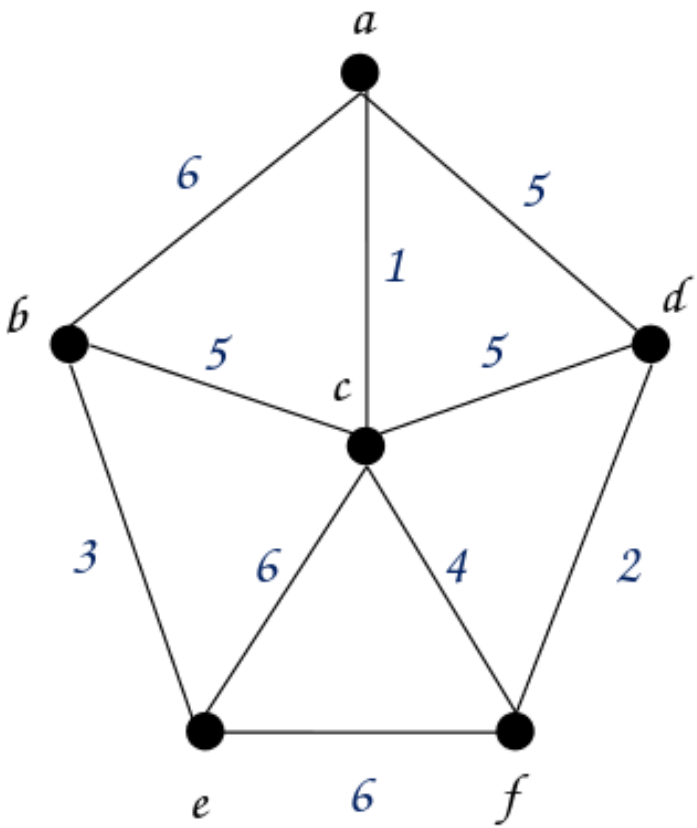
Algoritmo de Prim

Definición

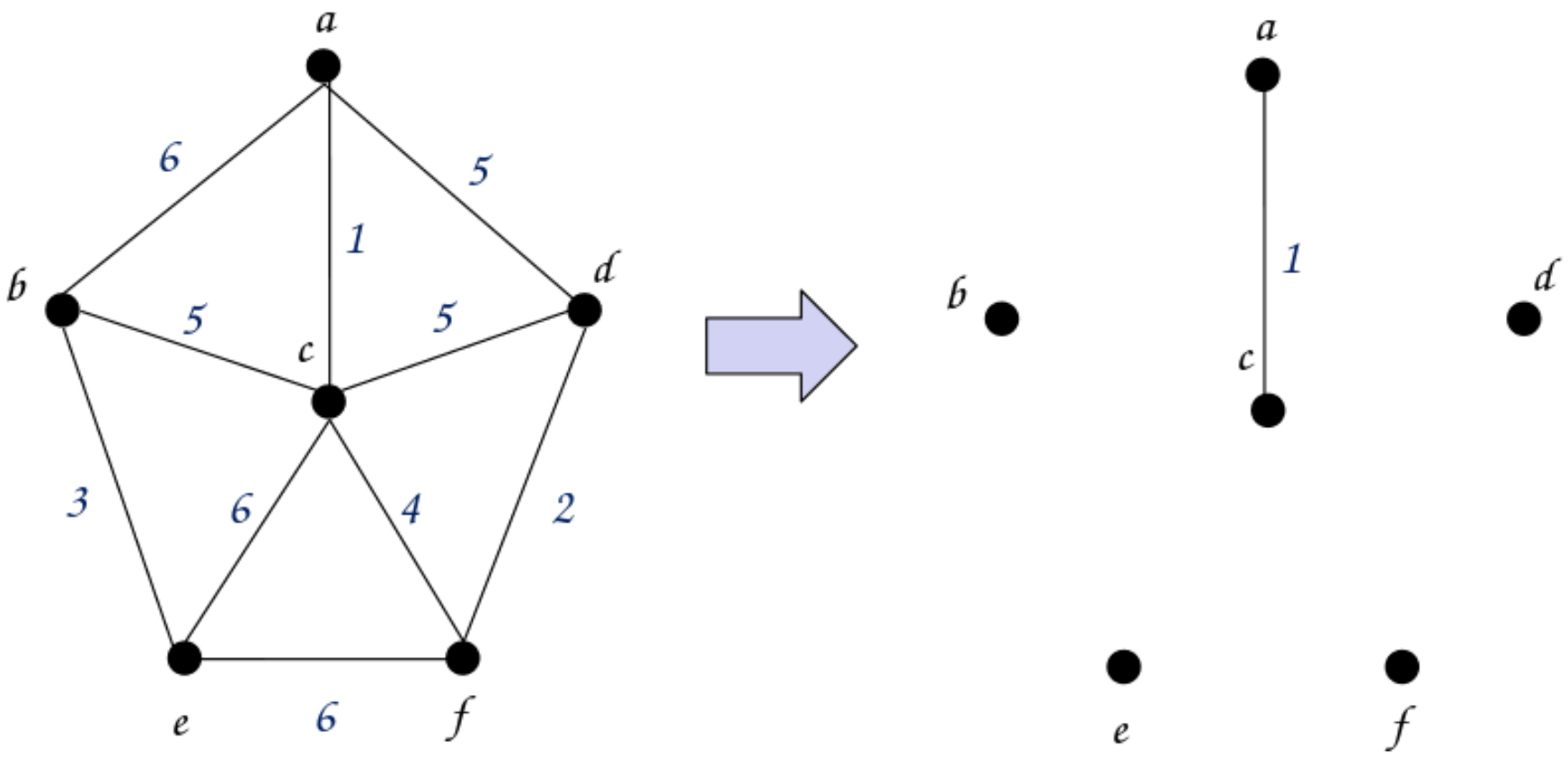
1. Escoja la arista de menor peso y adicionela al árbol recubridor
2. Seleccione la mejor arista que sea incidente al árbol recubridor y que no cree un circuito. Adicionela al árbol
3. Repita el proceso hasta cuando el árbol tenga $n-1$ aristas (n es el número de vértices)

Algoritmo de Prim

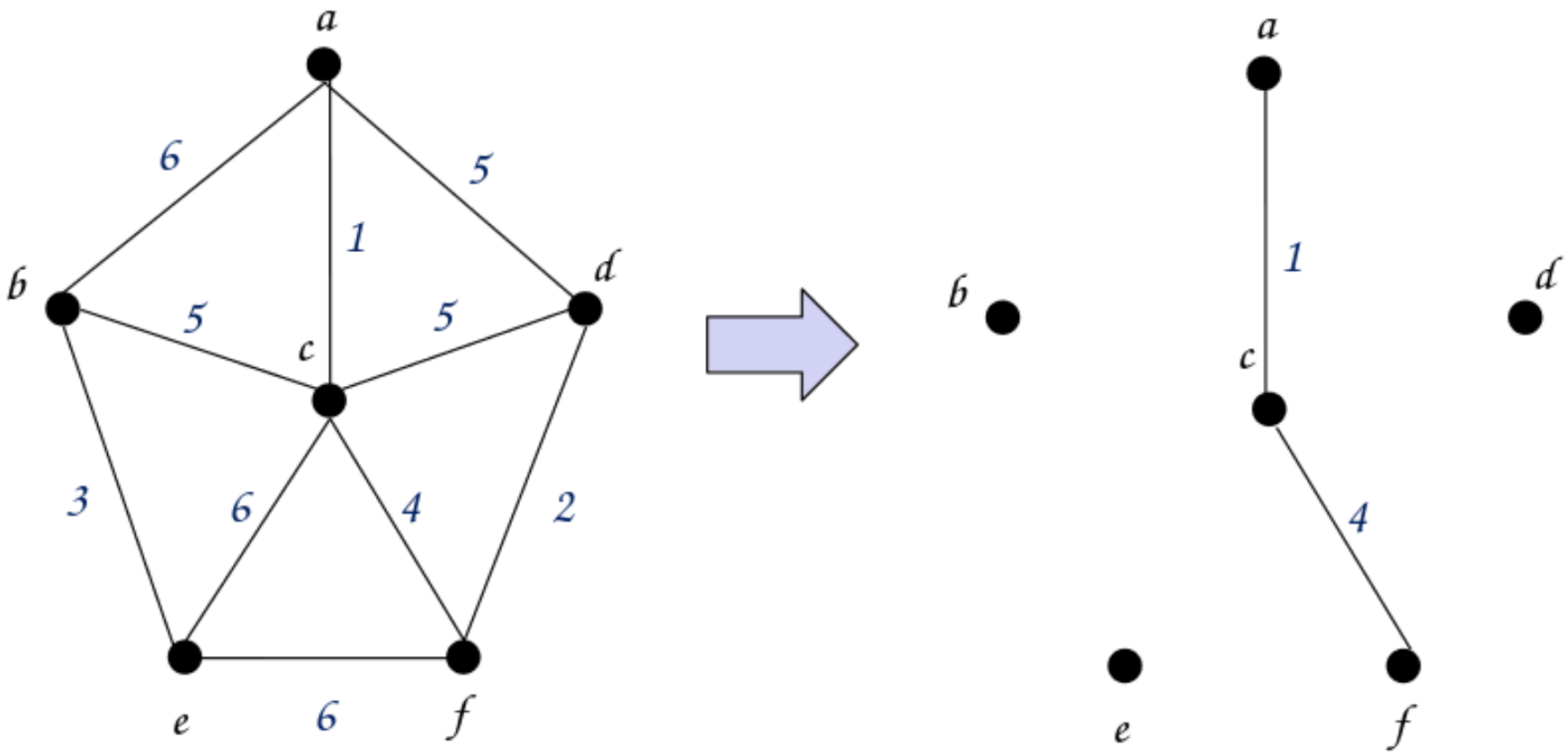
Aplicar el algoritmo de Prim



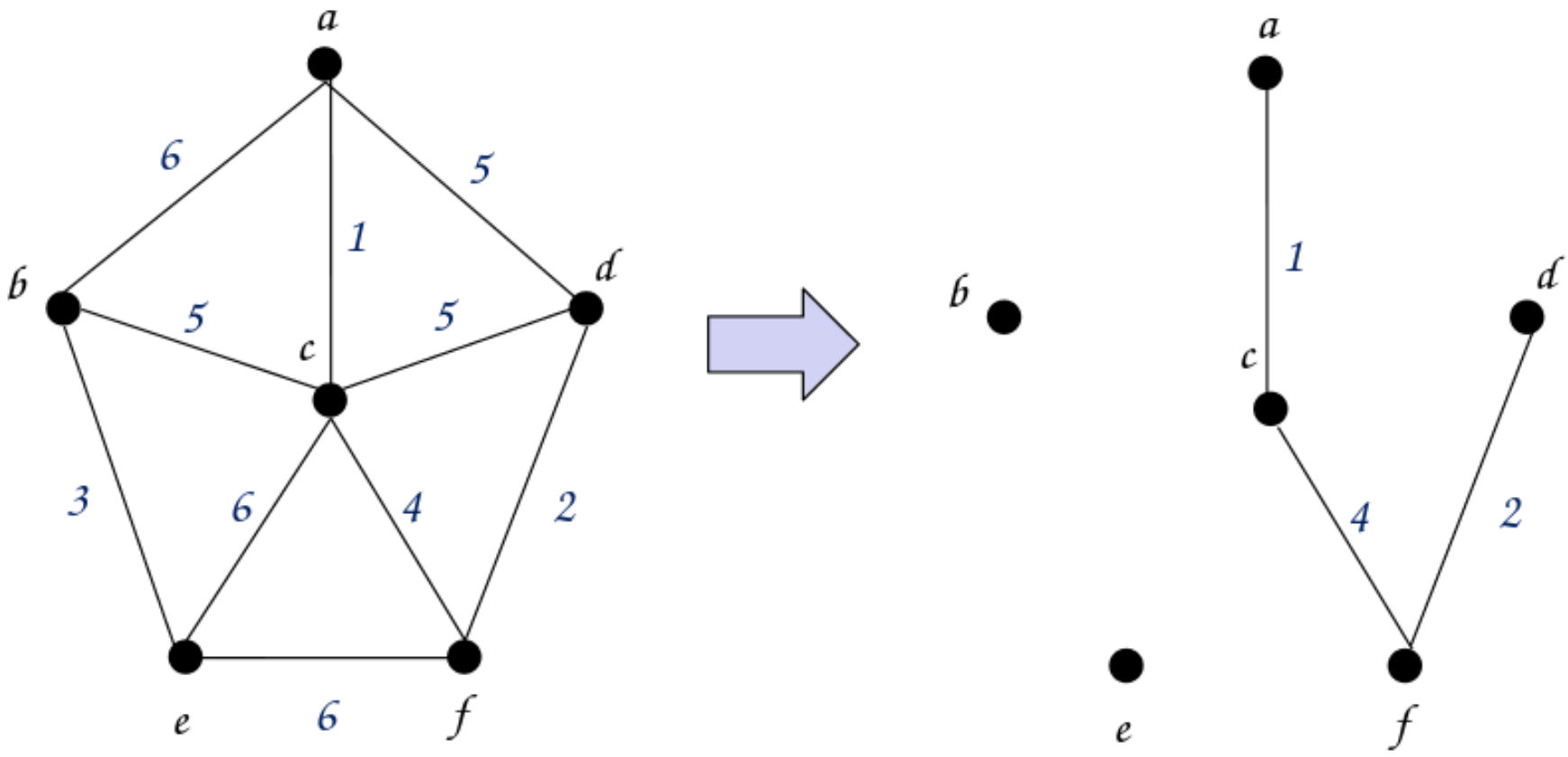
Algoritmo de Prim



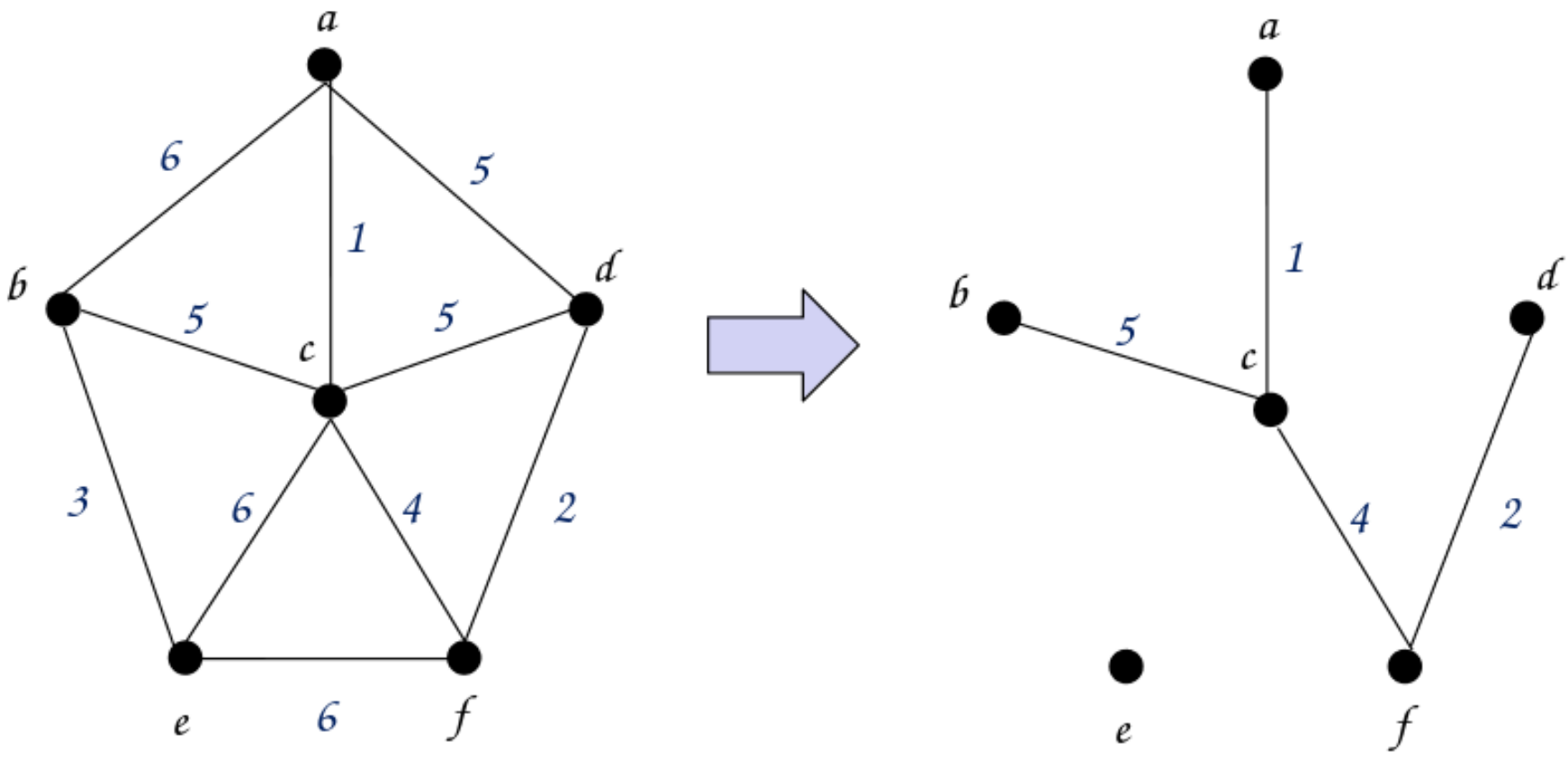
Algoritmo de Prim



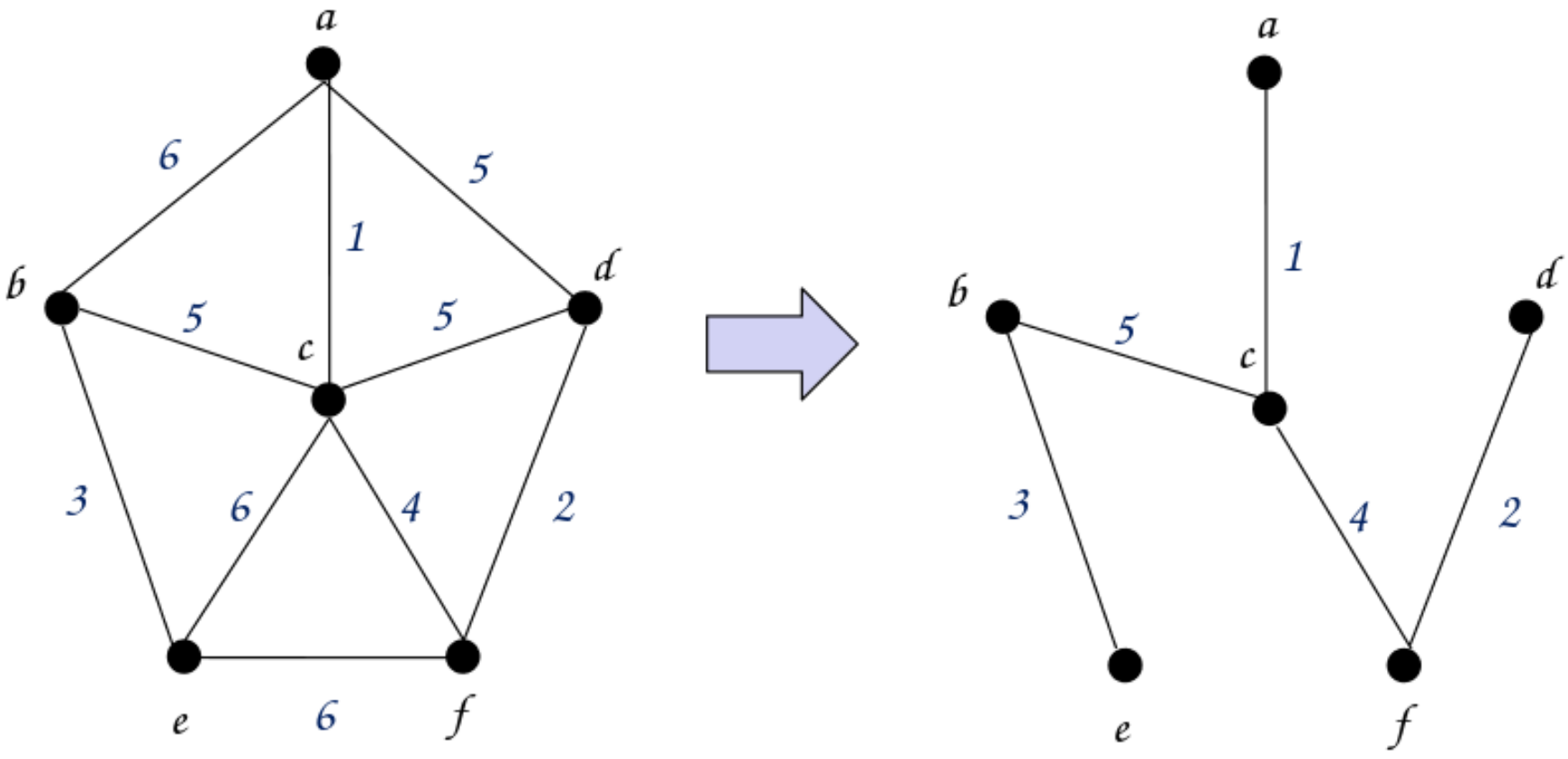
Algoritmo de Prim



Algoritmo de Prim



Algoritmo de Prim



Contenido

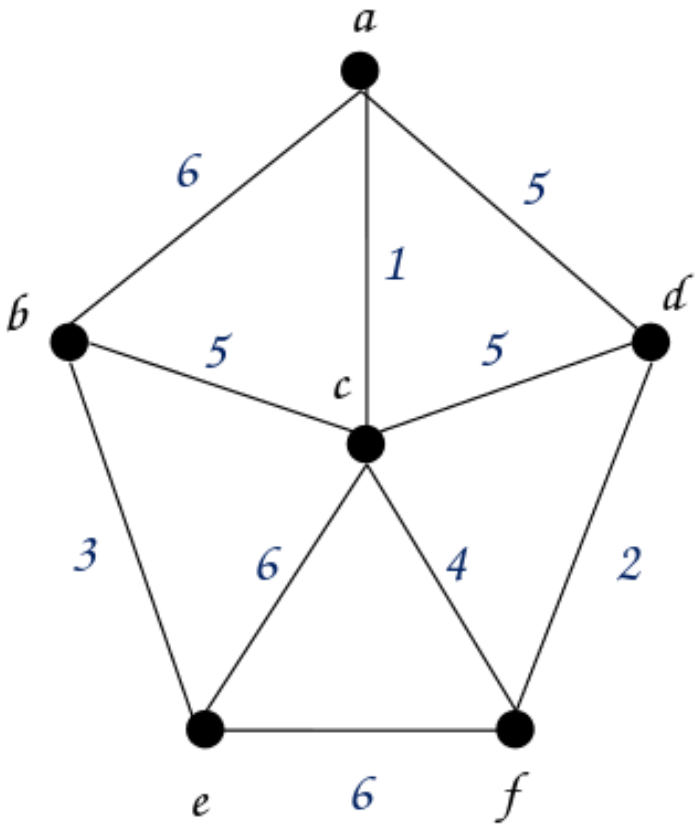
Algoritmo de Kruskal

Definición

1. Escoja la arista de menor peso y adiciónela al árbol recubridor
2. Seleccione la arista con menor peso y adiciónela al árbol recubridor
3. Repita el proceso hasta cuando el árbol tenga $n-1$ aristas (n es el número de vértices)

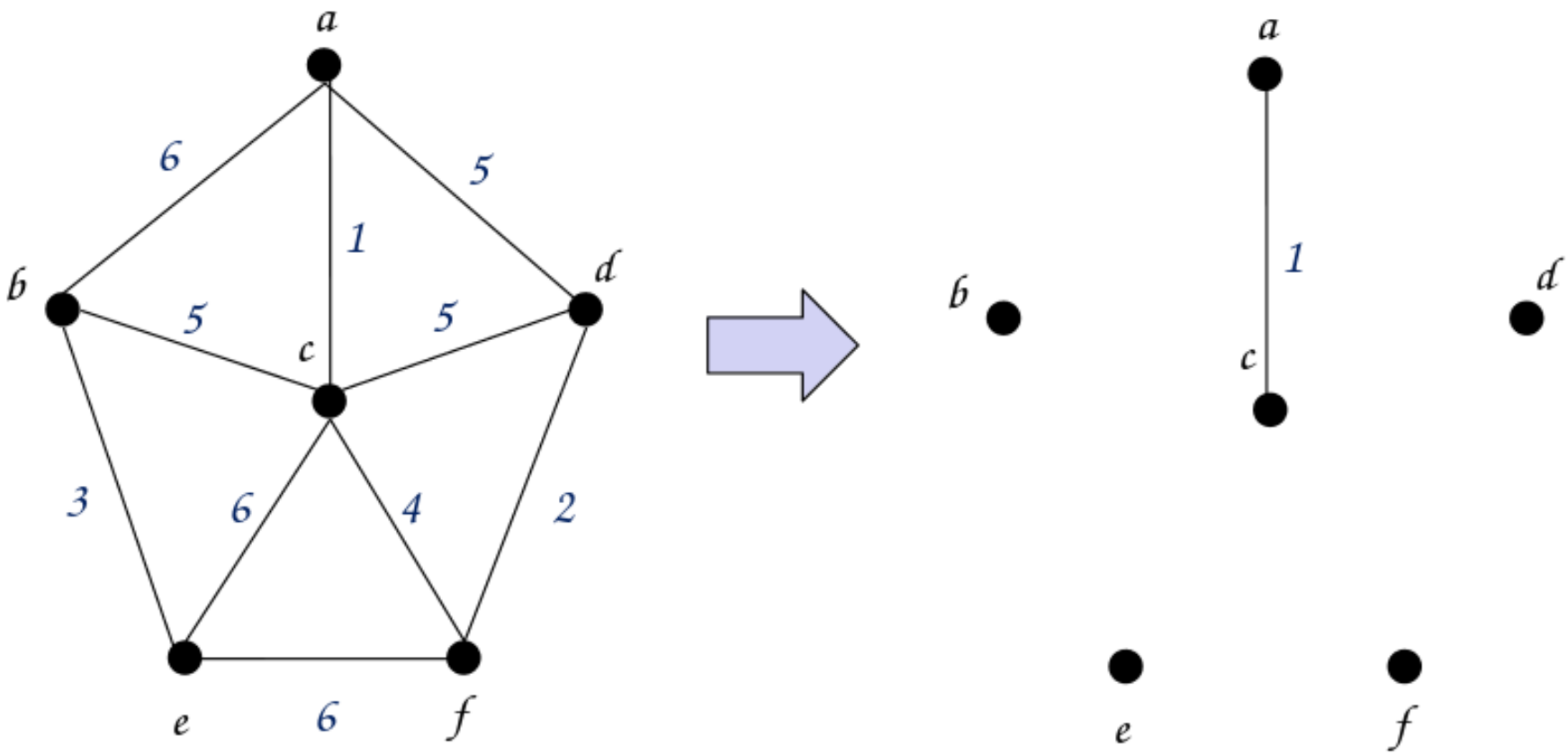
Algoritmo de Kruskal

Aplicar el algoritmo de Kruskal



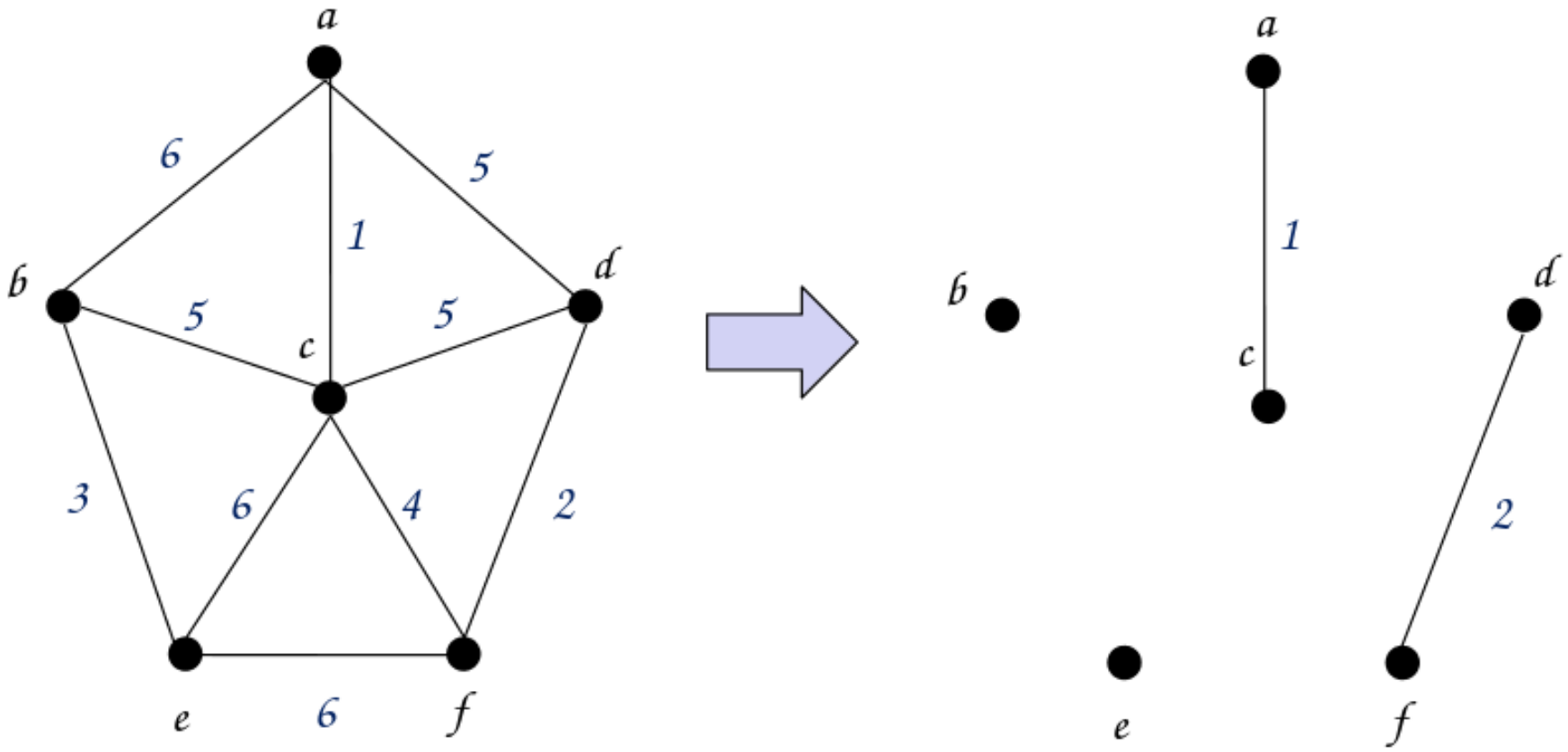
Algoritmo de Kruskal

Aplicar el algoritmo de Kruskal



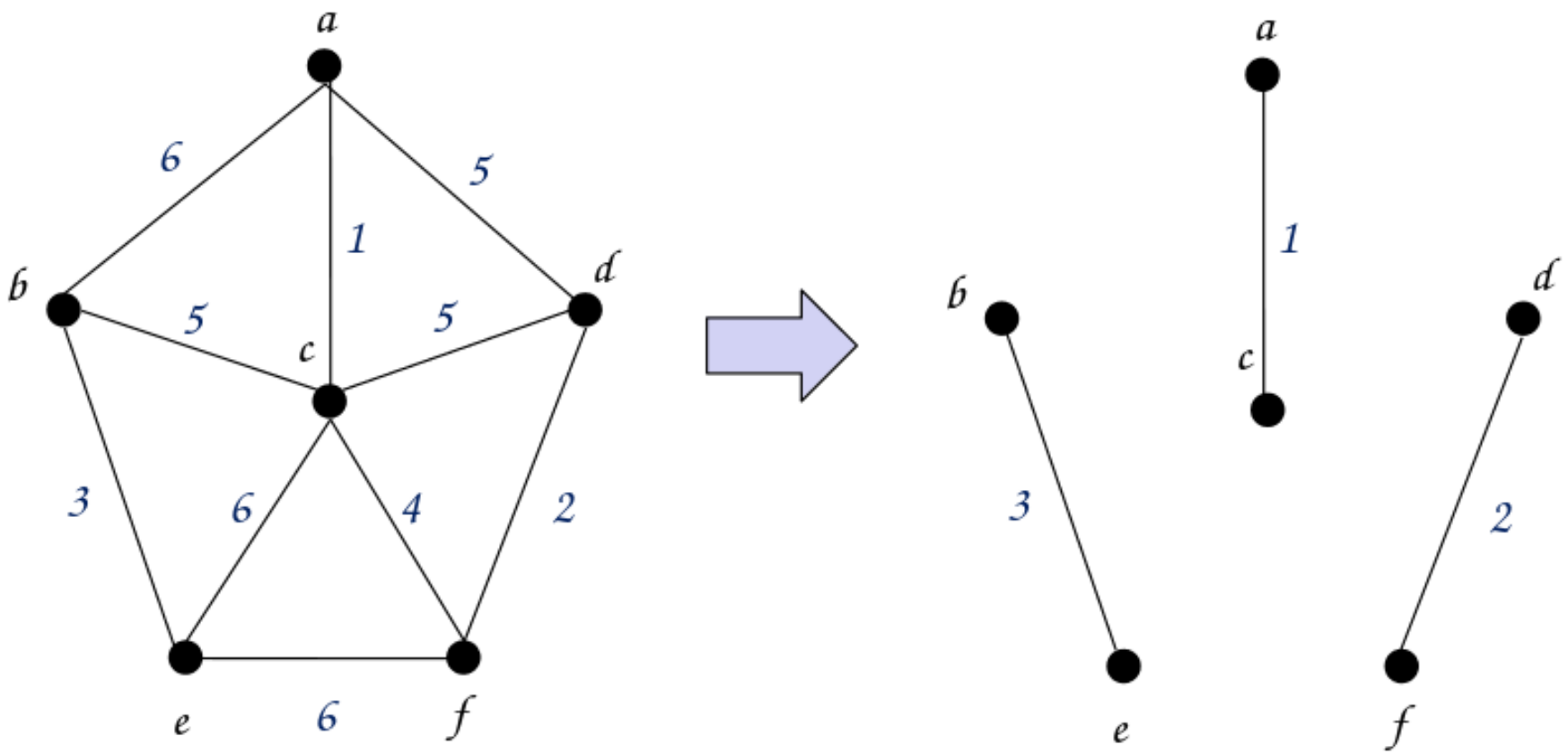
Algoritmo de Kruskal

Aplicar el algoritmo de Kruskal



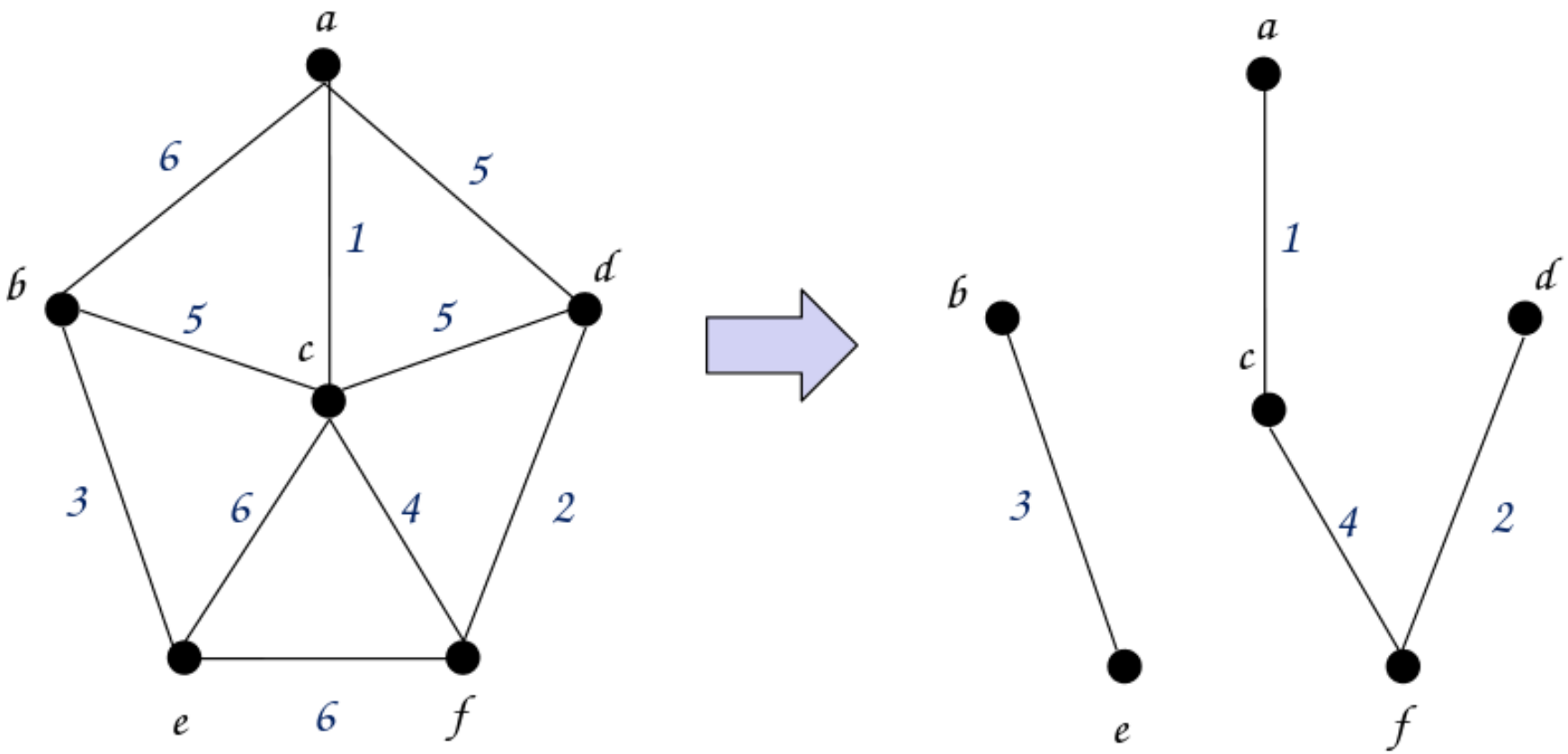
Algoritmo de Kruskal

Aplicar el algoritmo de Kruskal



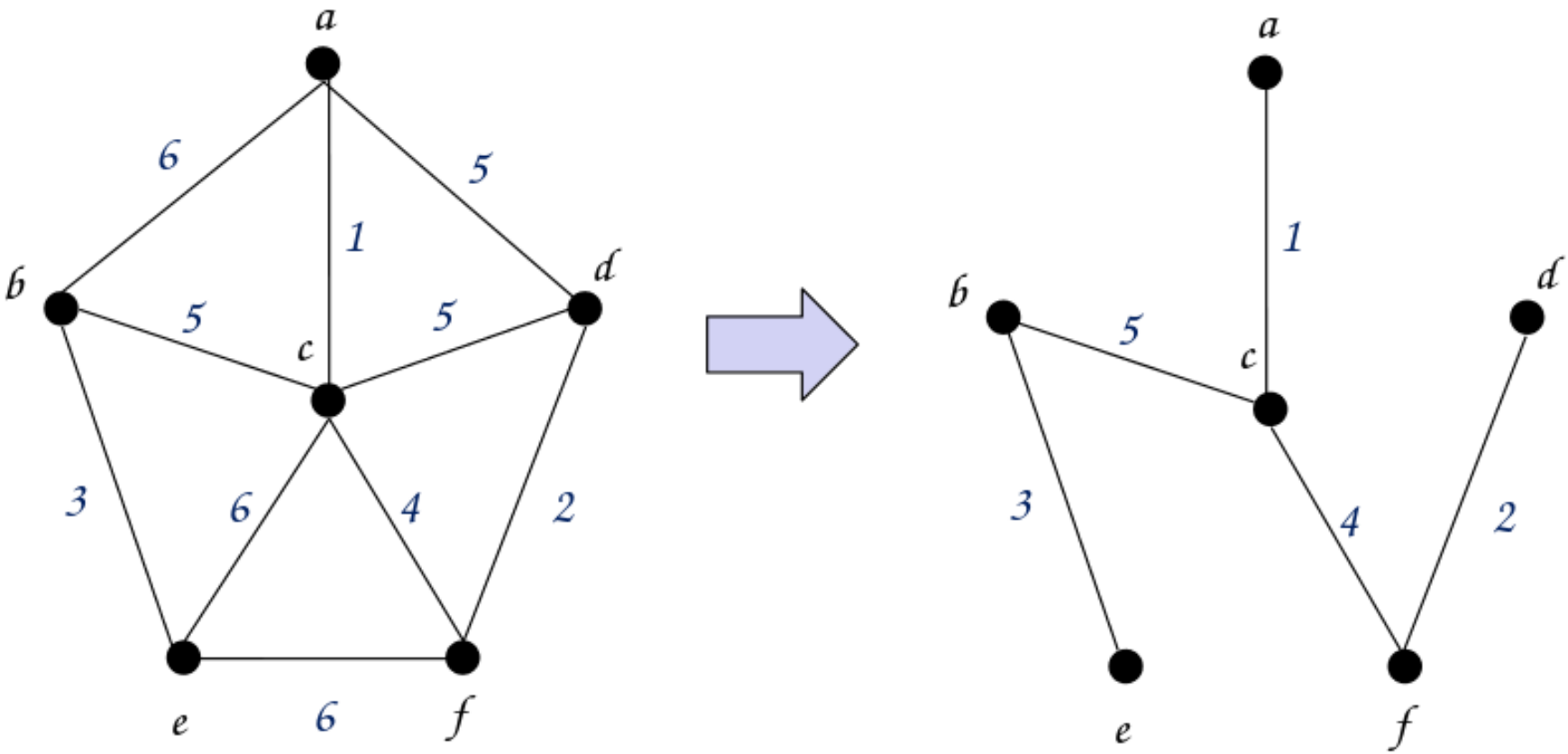
Algoritmo de Kruskal

Aplicar el algoritmo de Kruskal

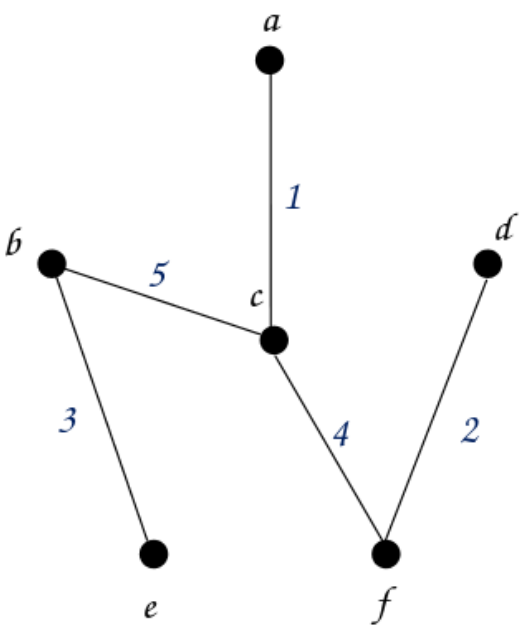


Algoritmo de Kruskal

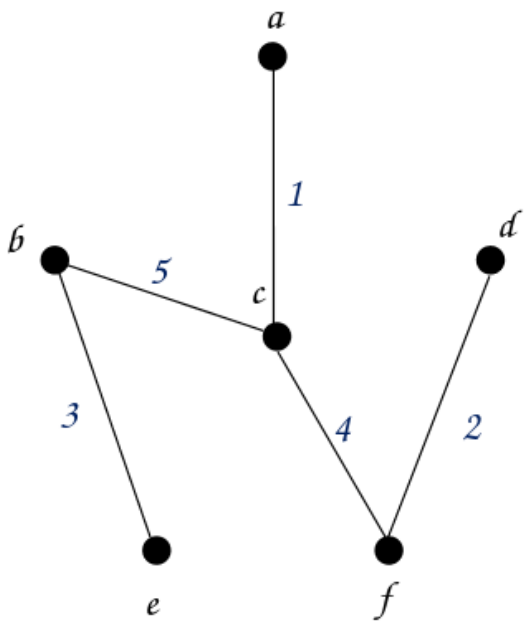
Aplicar el algoritmo de Kruskal



Comparación Kruskal y Prim



Árbol recubridor mínimo obtenido con el algoritmo de Prim



Árbol recubridor mínimo obtenido con el algoritmo de Kruskal

Contenido

Contenido

Árboles de expansión

Definición

- Los algoritmos de búsqueda permiten encontrar nodos en un árbol.
- Se utilizan para resolver problemas de búsqueda en problemas basados en árboles y grafos
- Son especialmente útiles para el desarrollo de *jugadores inteligentes* en juegos

Estrategias de búsqueda

Introducción

Existen dos formas de recorrer un árbol. Búsqueda por amplitud (**BFS**) y búsqueda por profundidad (**DFS**). **BFS** se puede utilizar para hallar la distancia más corta entre algún nodo inicial y los nodos restantes de un grafo. La distancia más corta es el mínimo número de aristas que hay que recorrer entre un par vértices.

Contenido

Búsqueda en amplitud

Introducción

La idea de esta búsqueda es iniciar un nodo concreto del grafo e ir examinando todos los nodos adyacentes a este, repitiendo este proceso hasta encontrar el nodo deseado. Para mejorar esta búsqueda se pueden marcar como visitados los nodos ya recorridos, para evitar tener que recorrerlos de nuevo.

Búsqueda en amplitud

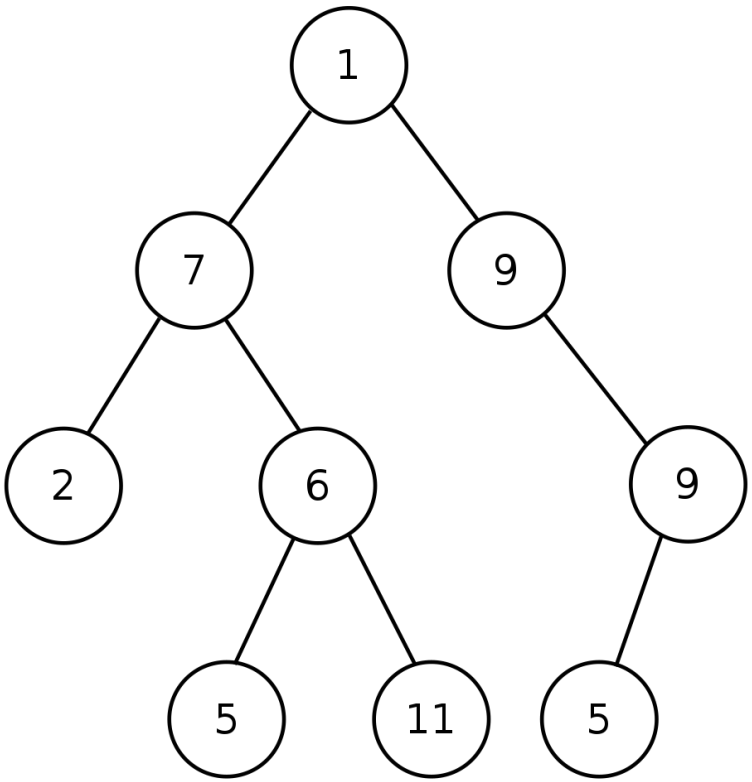
Introducción

El enfoque general de **BFS** es:

1. Inicie en un nodo s
2. Observe todos los hijos
3. Almacene los caminos encontrados desde nodo inicial s al resto de nodos
4. Repita el proceso
5. Para programar esta búsqueda puede usar un cola

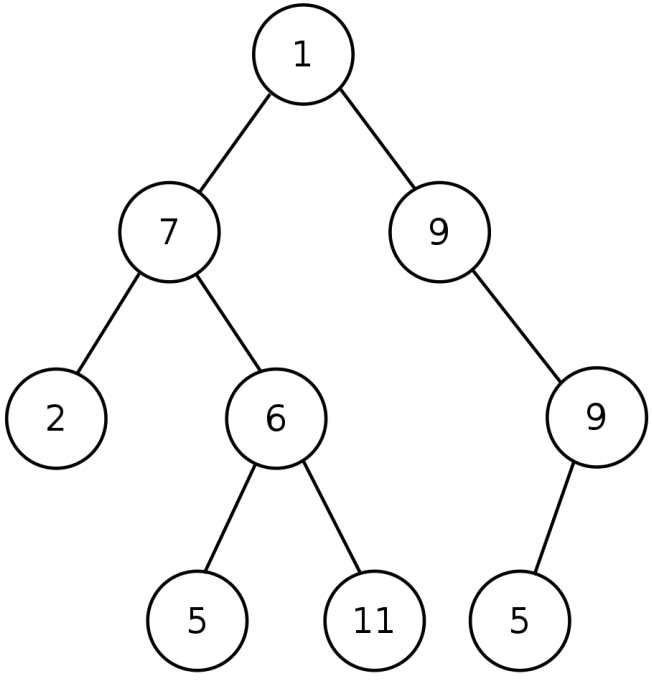
Búsqueda en amplitud

Tomemos como ejemplo el siguiente árbol:



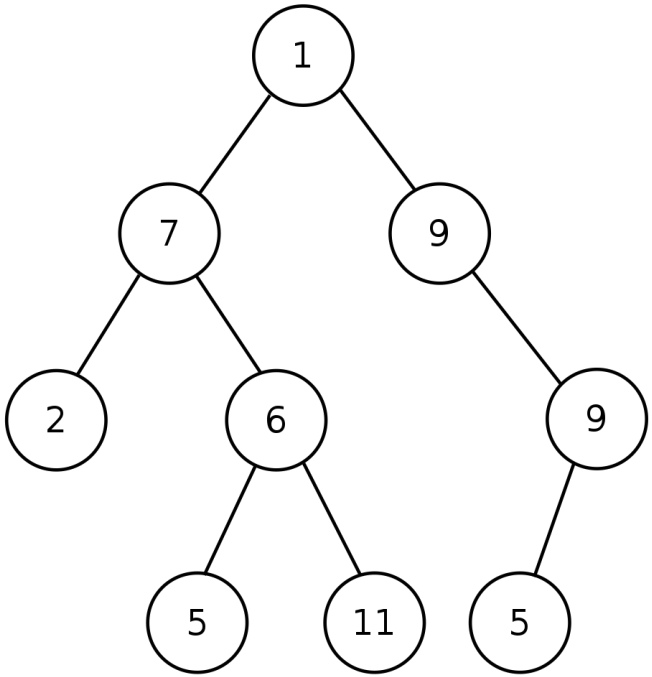
Búsqueda en amplitud

Arrancamos en la raíz, agregamos a la lista y continuamos con los hijos.



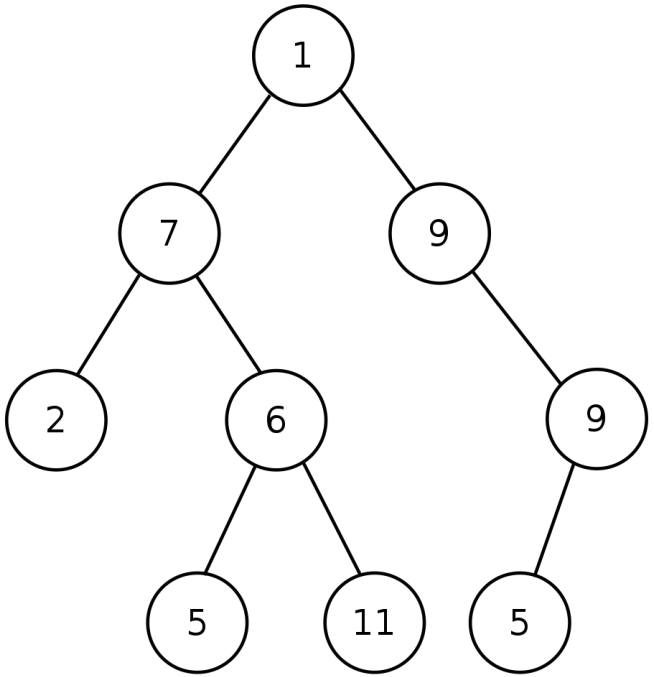

```
//Primer iteración
cola = []

//Agregar raiz
cola = [1]
salida = [1]

//Continuar con los hijos
cola = [7 9]
salida = [1]
```

Búsqueda en amplitud




```
cola = [9 2 6]
salida = [1 7]

cola = [2 6 9]
salida = [1 7 9]

cola = [6 9]
salida = [1 7 9 2]

cola = [9 5 11]
salida = [1 7 9 2 6]
```

Búsqueda en amplitud




```
cola = [5 11 5]
salida = [1 7 9 2 6 9]

cola = [11 5]
salida = [1 7 9 2 6 9 5]

cola = [5]
salida = [1 7 9 2 6 9 5 11]

cola = []
salida = [1 7 9 2 6 9 5 11 5]
```

Contenido

Búsqueda en profundidad

Introducción

Suponga que una persona se encuentra en un sistema de cuevas interconectadas y quiere encontrar la salida.

- Una estrategia es comenzar a explorar la cueva más a la izquierda, hasta encontrar una intersección.
- En esta seleccionamos la cueva más a la izquierda y a así sucesivamente hasta encontrar la salida.
- Si en un camino dado situado más a la izquierda no tiene éxito, ya que es un camino sin salida, se devuelve hasta la ultima intersección y toma el camino situado al lado de la ultima elección.
- Este proceso puede aplicarse a la derecha de forma análoga.

Búsqueda en profundidad

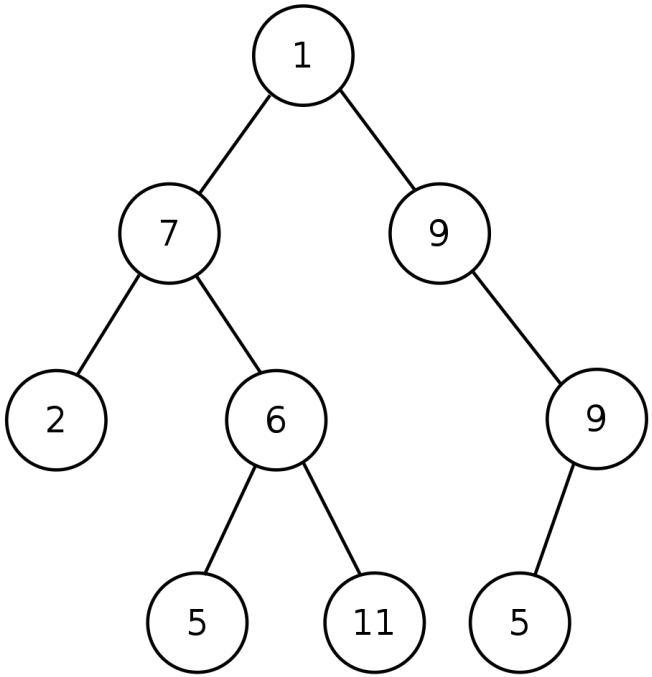
Introducción

La rutina asociada una búsqueda DFS es:

1. Iniciar por la raiz
2. Recorrer los nodos priorizando el orden indicado
3. Una vez no se pueda recorrer más, recorrer los nodos faltantes
4. Para implementar este algoritmo se usa una pila

Búsqueda en amplitud

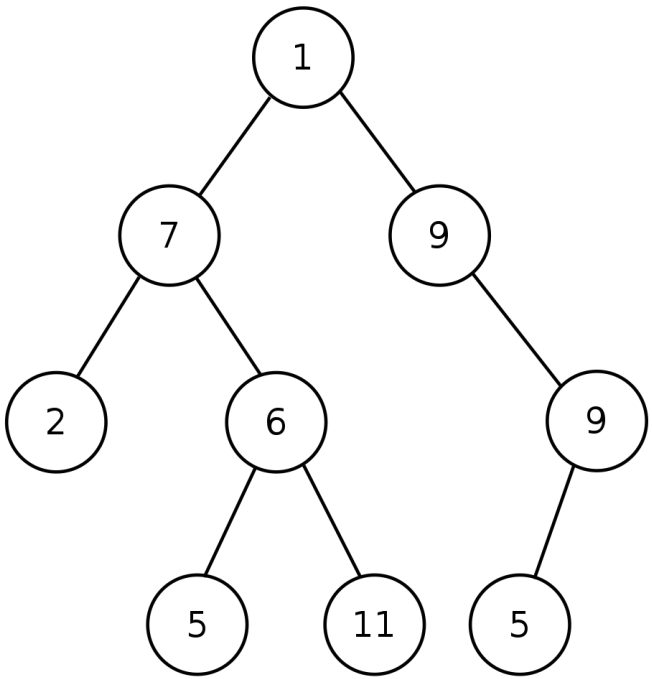
Ejemplo priorizando a la izquierda




```
//Primer iteración
pila = []

//Agregar raiz
pila = [1]
salida = [1]

//Continuar con los hijos
pila = [7 9]
salida = [1]
```

Búsqueda en amplitud

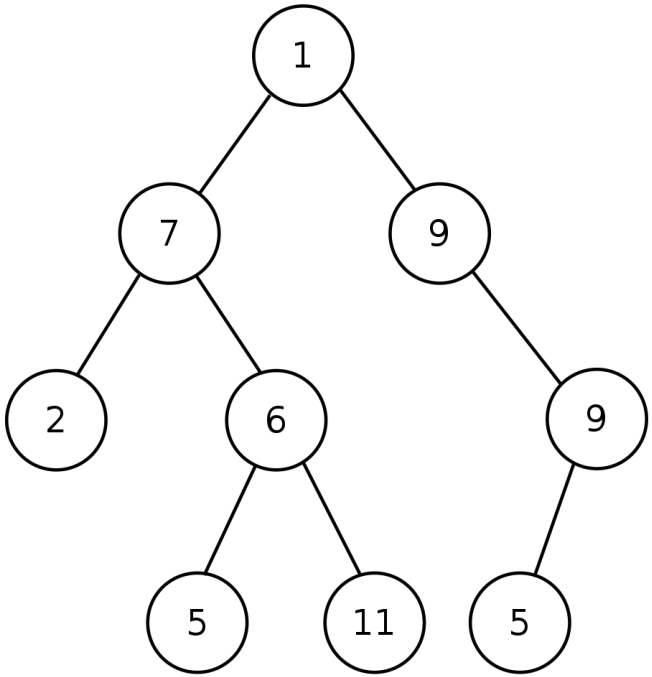
Ejemplo priorizando a la izquierda




```
pila = [2 6 9]
salida = [1 7]

pila = [6 9]
salida = [1 7 2]

pila = [5 11 9]
salida = [1 7 2 6]

pila = [11 9]
salida = [1 7 2 6 5]
```

Búsqueda en amplitud

Ejemplo priorizando a la izquierda




```
pila = [9]
salida = [1 7 2 6 5 11]

pila = [9]
salida = [1 7 2 6 5 11 9]

pila = [5]
salida = [1 7 2 6 5 11 9 9]

pila = []
salida = [1 7 2 6 5 11 9 9 5]
```

Referencias


Kenneth H. Rosen.
Discrete Mathematics and Its Applications.
McGraw-Hill Higher Education, 7th edition, 2011.
Chapter 11. Graphs.

Gracias

Próximo tema:
Lenguajes y gramáticas