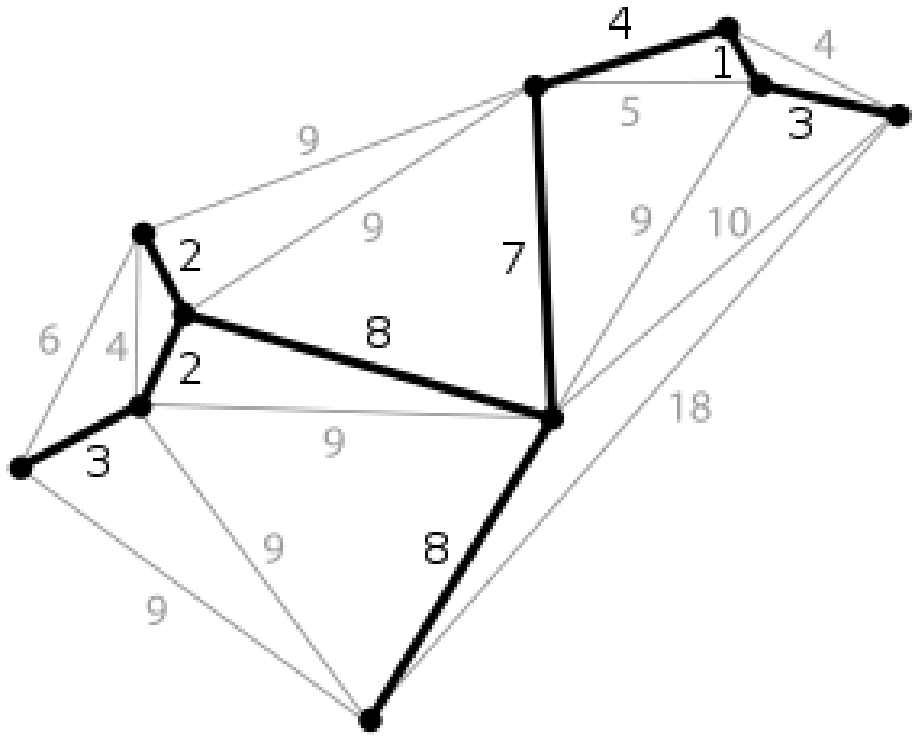
Algoritmo de Prim, parte 1

Árboles de mínima expansión

Consiste en “conectar” todos los nodos de un grafo de la manera menos costosa posible en términos de la sumatoria de longitudes (o costos para ser mas generales) de las aristas del árbol resultante.



Fuente: http://en.wikipedia.org/wiki/File:Minimum_spanning_tree.svg

Árboles de mínima expansión

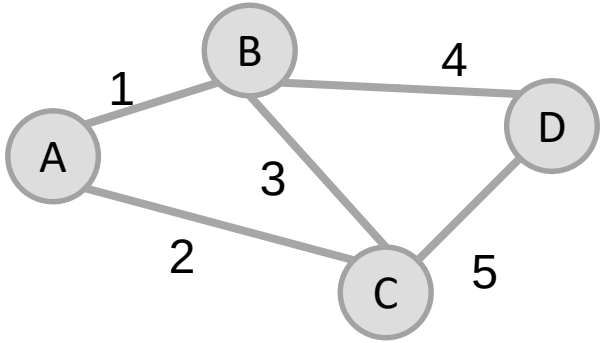
Entrada: Un grafo no dirigido $G = (V, E)$ completamente conectado donde cada arista (u, v) tiene un costo c (incluso puede ser negativo).

Salida: Un árbol T de mínimo costo que “abarque” todos los nodos de V. Es decir, el sub-grafo (V, T) debe ser completamente conectado.

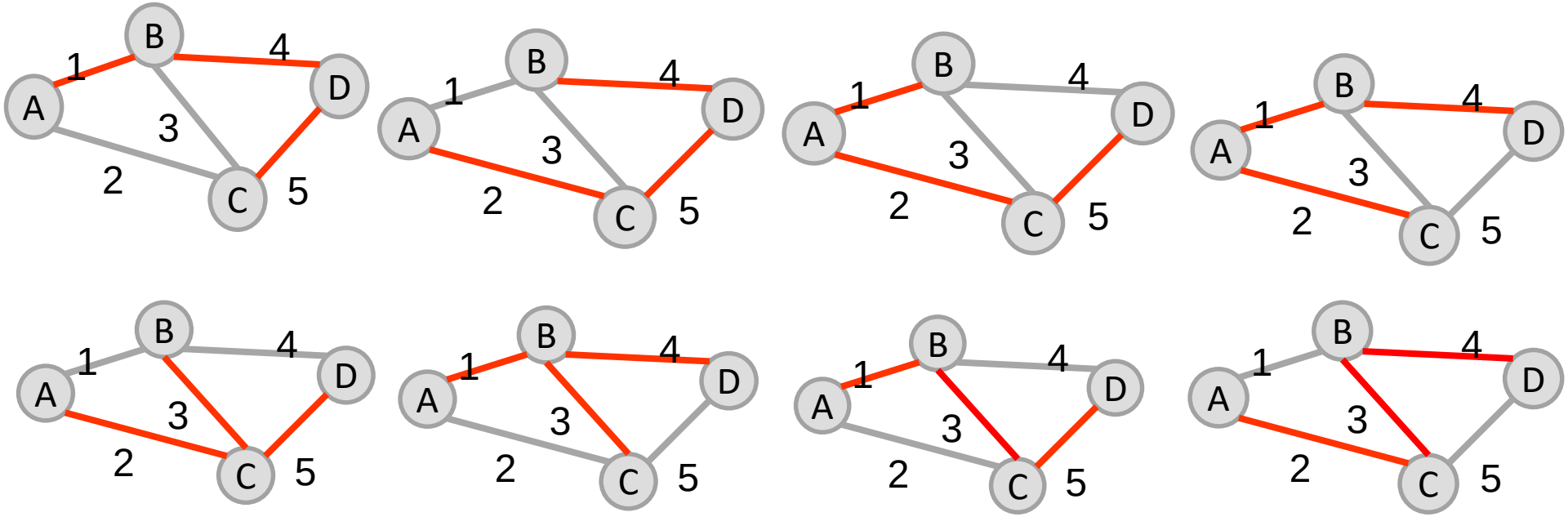
Si G no es completamente conectado, lo cual se puede verificar fácilmente usando DFS por ejemplo, el problema se puede convertir fácilmente a uno de “Bosques de mínima expansión”, que consiste en encontrar el conjunto de árboles de mínima expansión que abarcan todos los nodos.

Árboles de mínima expansión

Ejemplo:



¿Cuántos árboles diferentes tiene este grafo?



Y en general ¿Cuántos árboles diferentes puede tener un grafo?

En un árbol conectado (todos con todos) la cantidad de árboles es tan grande como N^{N-2} (http://en.wikipedia.org/wiki/Cayley%27s_formula)

Algoritmo de Prim

```
function Prim(grafo G){
   a = cualquier nodo ∈ V
   C = {a} //Nodos ya revisados
   T = NULL
   while C ≠ V:
      entre todas las aristas (v, w) ∈ V con v ∈ C y w ¬ ∈ C,
      escoger (v,’ w’) de menor costo
      C.add(w’)
      T.add(v,’ w’)
```

¿Cuál es la complejidad?

Haciendo el mismo análisis que con Dijkstra llegamos a que la complejidad resultante es *O*(*N*M)*, sin duda mejor que exponencial.

Sin embargo, igual que con Dijkstra, podemos mejorar esta eficiencia empleando una cola con prioridad implementada mediante un montículo binario.