Algoritmos Avaros

Miguel Raggi

Algoritmos
ENES
UNAM

10 de abril de 2018

Índice:

1. Introducción
2. Arbol Generador de peso mínimo
   - Aplicaciones
3. Algoritmo de Kruskal
4. Algoritmo de Prim
   - Descripción
   - Ejemplo
   - Prueba
   - Estructuras de datos
   - Pseudo-código

Índice:

1. Introducción
2. Arbol Generador de peso mínimo
   - Aplicaciones
3. Algoritmo de Kruskal
4. Algoritmo de Prim
   - Descripción
   - Ejemplo
   - Prueba
   - Estructuras de datos
   - Pseudo-código

Número más grande

Problema

Dada una serie de dígitos $d_1, d_2, ..., d_n$, encuentra el número más grande que puedes formar acomodándolos.

Número más grande

Problema

Dada una serie de dígitos $d_1, d_2, ..., d_n$, encuentra el número más grande que puedes formar acomodándolos.

Por ejemplo, si los dígitos fueran 7,9,3,2, el más grande que puedo formar es 9732.

Número más grande

Problema

Dada una serie de dígitos $d_1, d_2, ..., d_n$, encuentra el número más grande que puedes formar acomodándolos.

Por ejemplo, si los dígitos fueran 7,9,3,2, el más grande que puedo formar es 9732.

- No es difícil: simplemente tómalos en orden decreciente.

Gasolina

Problema

Tienes un carro y quieres llegar del punto A al punto B, que están a 1,000km. Un tanque de gasolina me alcanza para avanzar 400km. Durante el camino hay varias gasolinerías puestas en distintos puntos p_1, p_2, ... p_n. ¿Cuál es el mínimo número de veces que me tengo que detener para llenar gasolina?

Gasolina

Problema

Tienes un carro y quieres llegar del punto A al punto B, que están a 1,000km. Un tanque de gasolina me alcanza para avanzar 400km. Durante el camino hay varias gasolinerías puestas en distintos puntos p_1, p_2, ... p_n. ¿Cuál es el mínimo número de veces que me tengo que detener para llenar gasolina?

- Otra vez, simplemente hay que detenerse cuando ya no hubiera podido avanzar más.

Índice:

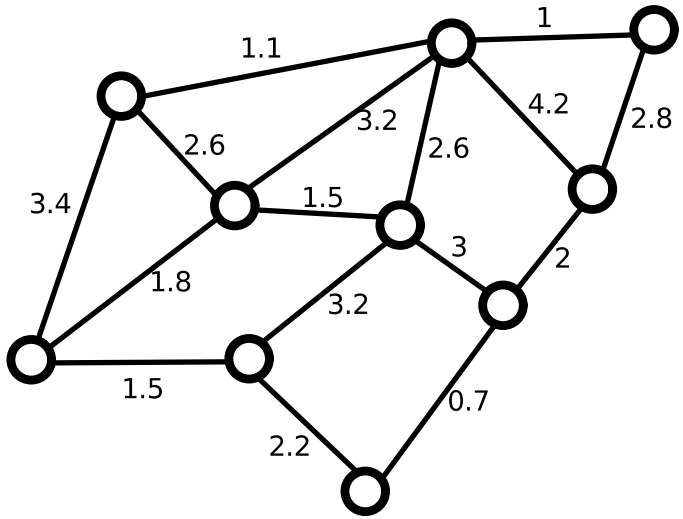
1. Introducción
2. Arbol Generador de peso mínimo
   - Aplicaciones
3. Algoritmo de Kruskal
4. Algoritmo de Prim
   - Descripción
   - Ejemplo
   - Prueba
   - Estructuras de datos
   - Pseudo-código

Árbol Generador Mínimo

- Sea G un grafo simple y conexo con costos en las aristas (positivos o negativos, no importa).

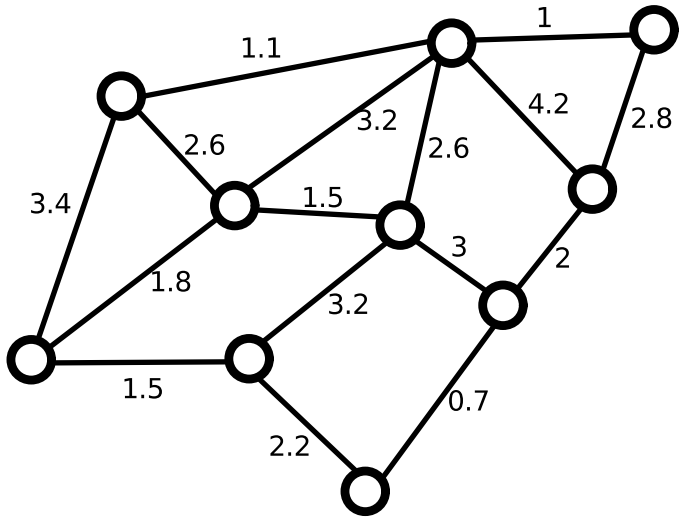
Árbol Generador Mínimo

- Sea G un grafo simple y conexo con costos en las aristas (positivos o negativos, no importa).
- Queremos encontrar el árbol con menos costo.



Árbol Generador Mínimo

- Sea G un grafo simple y conexo con costos en las aristas (positivos o negativos, no importa).
- Queremos encontrar el árbol con menos costo.



- Puedo sumar una constante a todos los pesos, o multiplicar por una constante positiva todos los pesos y no afecto el problema.

Aplicaciones

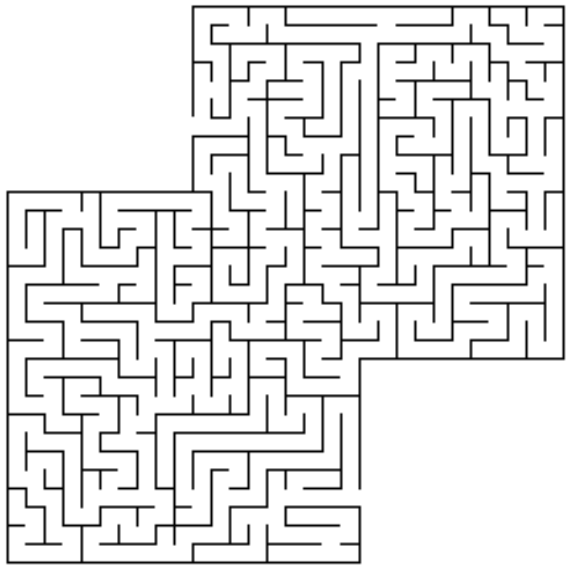
Aplicaciones:

- Conectar ciudades (circuitos, casas, etc) con calles (cables, tubos, etc) de manera que quede todo conectado.

Aplicaciones

Aplicaciones:

- Conectar ciudades (circuitos, casas, etc) con calles (cables, tubos, etc) de manera que quede todo conectado.
- ¡Generar un laberinto! (cuadrícula con pesos al azar)



Índice:

1. Introducción
2. Arbol Generador de peso mínimo
   - Aplicaciones
3. Algoritmo de Kruskal
4. Algoritmo de Prim
   - Descripción
   - Ejemplo
   - Prueba
   - Estructuras de datos
   - Pseudo-código

Algoritmo de Kruskal

- Es un algoritmo avaro:

Algoritmo de Kruskal

- Es un algoritmo avaro:
  - Empieza ordenando el conjunto de aristas por peso.

Algoritmo de Kruskal

- Es un algoritmo avaro:
  - Empieza ordenando el conjunto de aristas por peso.
  - En cada iteración, toma la arista de menor peso que no forme ciclo con las que tienes.

Algoritmo de Kruskal

- Es un algoritmo avaro:
  - Empieza ordenando el conjunto de aristas por peso.
  - En cada iteración, toma la arista de menor peso que no forme ciclo con las que tienes.
- Para saber si una arista la puedes agregar, sólo necesitas saber si conecta dos componentes conexas diferentes.

Algoritmo de Kruskal

- Es un algoritmo avaro:
  - Empieza ordenando el conjunto de aristas por peso.
  - En cada iteración, toma la arista de menor peso que no forme ciclo con las que tienes.
- Para saber si una arista la puedes agregar, sólo necesitas saber si conecta dos componentes conexas diferentes.
- Puedes hacer esto simplemente marcando todos los vértices de cada componente conexa con un numerito que indique la componente en la que están, y al agregar aristas, actualizar estos numeritos.
- También se pueden utilizar disjoint sets,

Índice:

1. Introducción
2. Arbol Generador de peso mínimo
   - Aplicaciones
3. Algoritmo de Kruskal
4. Algoritmo de Prim
   - Descripción
   - Ejemplo
   - Prueba
   - Estructuras de datos
   - Pseudo-código

Algoritmo de Prim: Árbol Generador Mínimo

Aquí está el algoritmo:

Algoritmo de Prim: Árbol Generador Mínimo

Aquí está el algoritmo:

- Empieza en un nodo cualquiera escogido al azar y márcalo como "explorado".

Algoritmo de Prim: Árbol Generador Mínimo

Aquí está el algoritmo:

- Empieza en un nodo cualquiera escogido al azar y márcalo como "explorado".
- En cada paso del algoritmo, repite lo siguiente:

Algoritmo de Prim: Árbol Generador Mínimo

Aquí está el algoritmo:

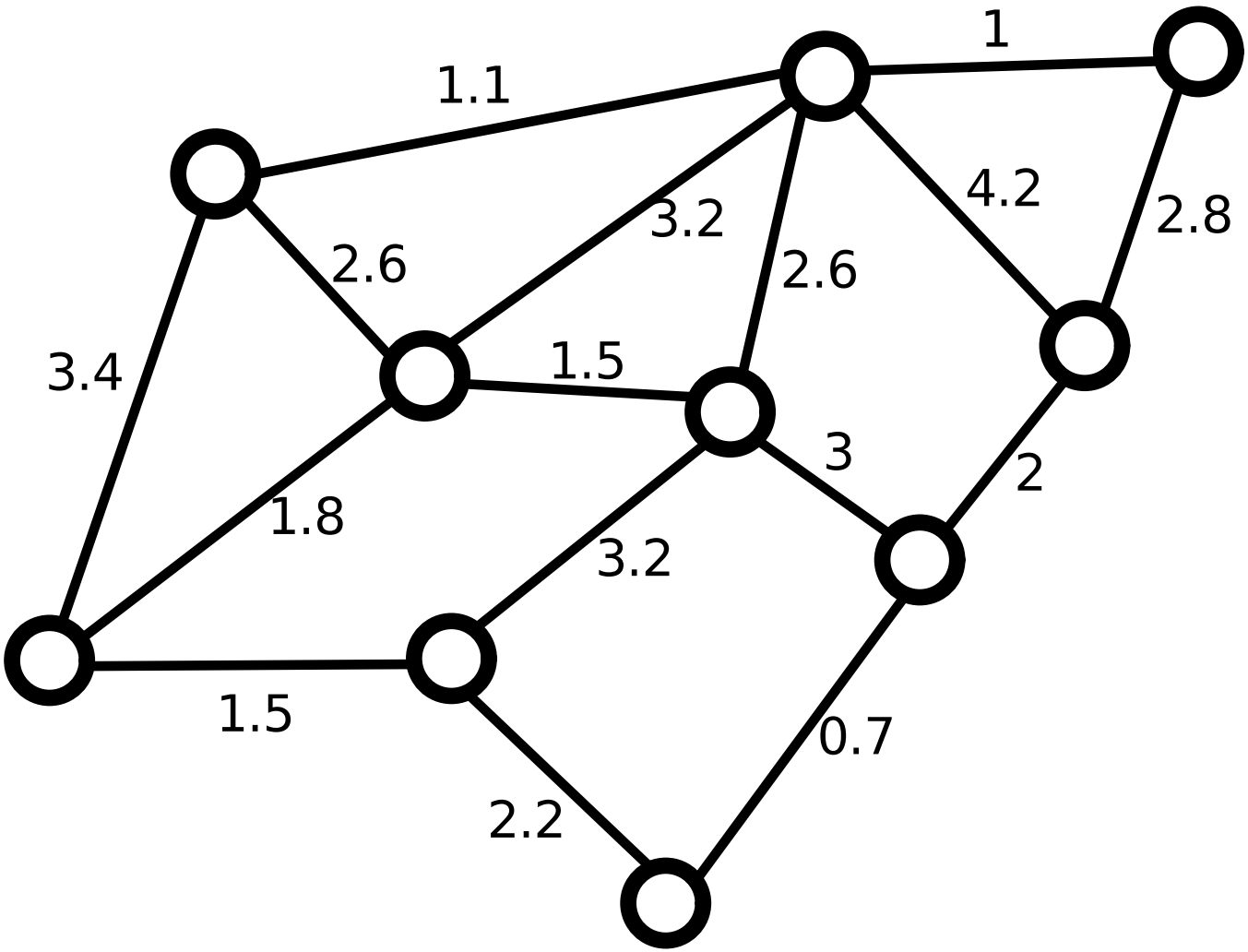
- Empieza en un nodo cualquiera escogido al azar y márcalo como "explorado".
- En cada paso del algoritmo, repite lo siguiente:
- Escoge la arista de menor costo que salga de alguno de los nodos explorados, pero que no vaya a un nodo explorado.

Algoritmo de Prim: Árbol Generador Mínimo

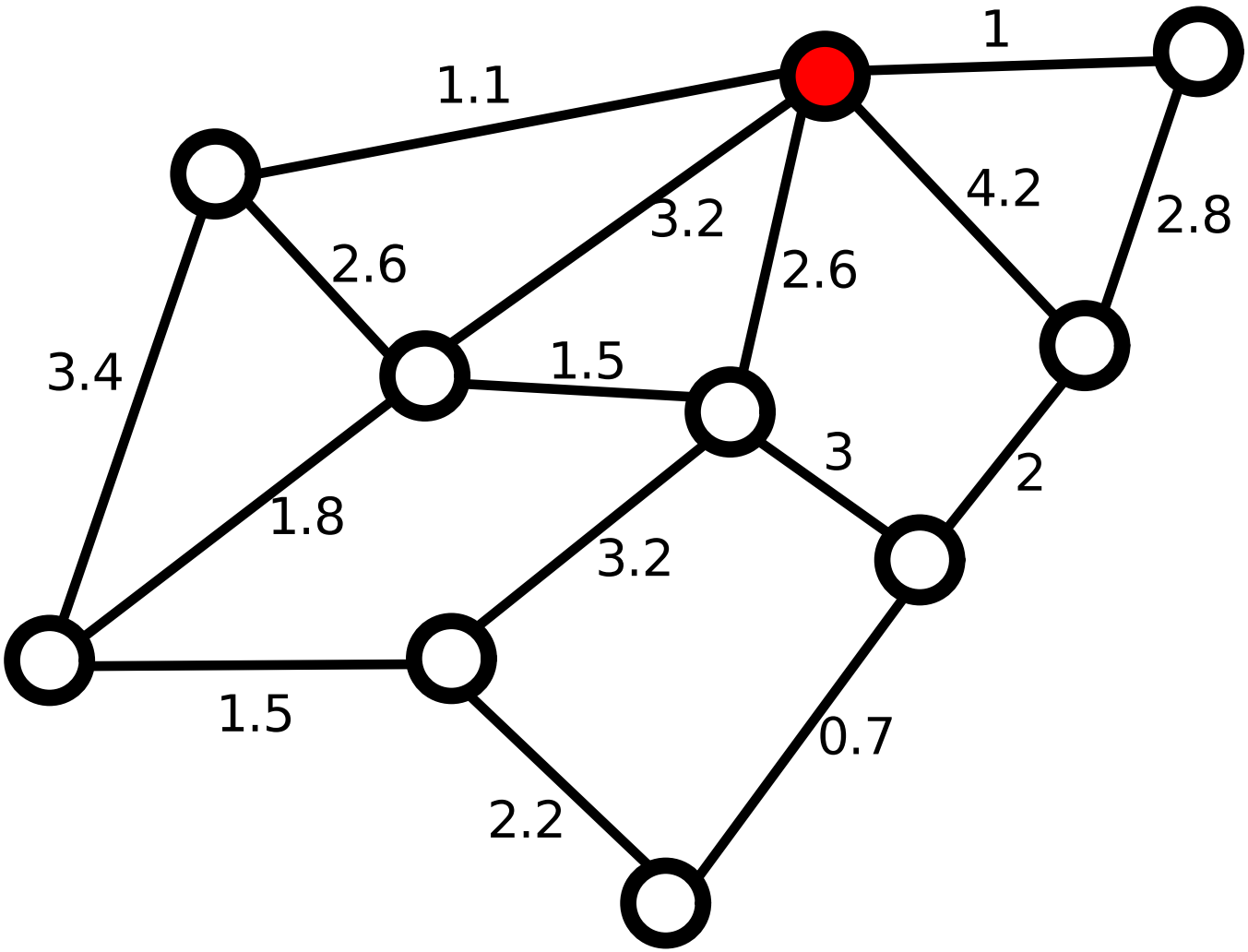
Aquí está el algoritmo:

- Empieza en un nodo cualquiera escogido al azar y márcalo como "explorado".
- En cada paso del algoritmo, repite lo siguiente:
- Escoge la arista de menor costo que salga de alguno de los nodos explorados, pero que no vaya a un nodo explorado.
- Repite, sólo fijándote en las aristas que no tienen del otro lado a un nodo "explorado".

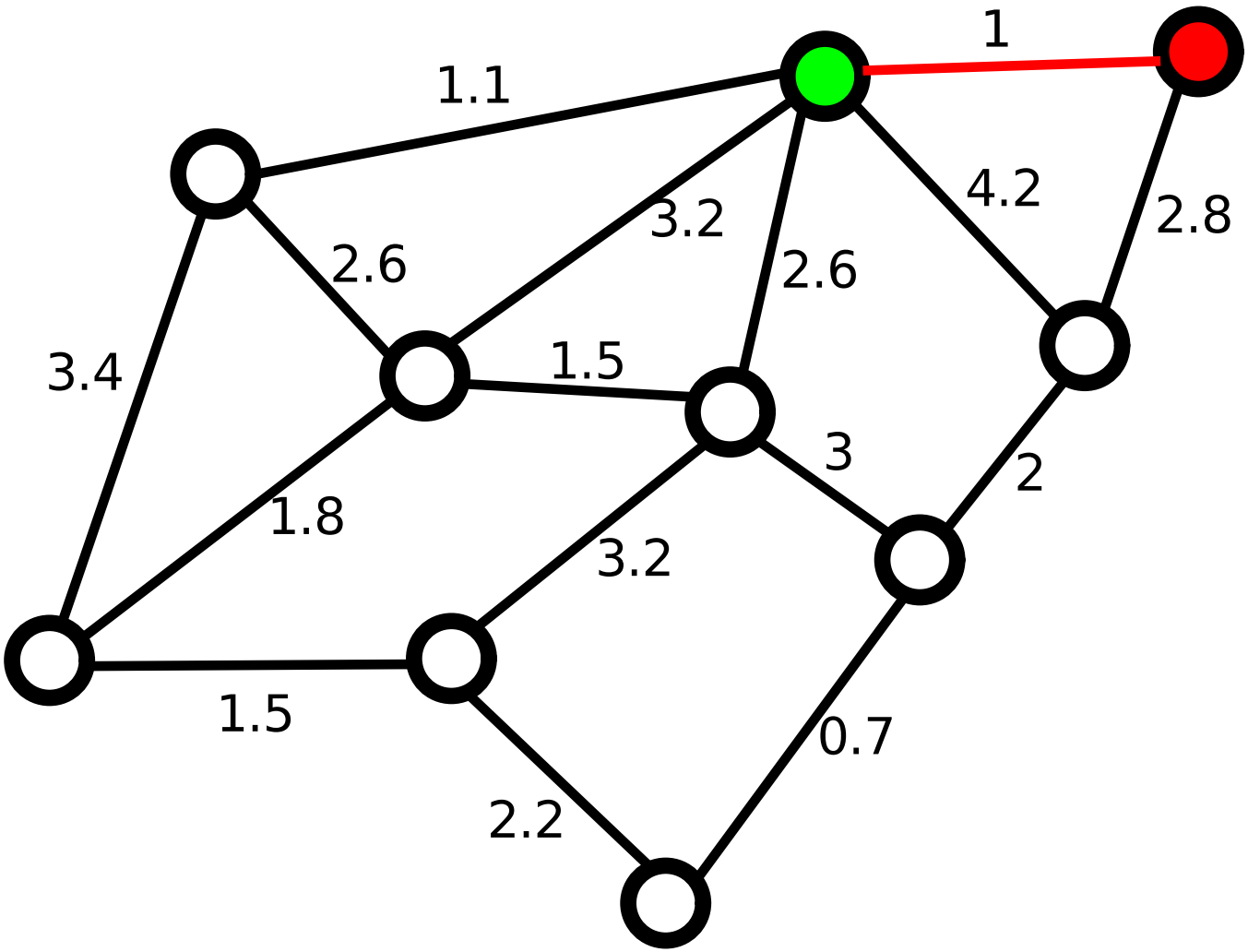
Algoritmo de Prim



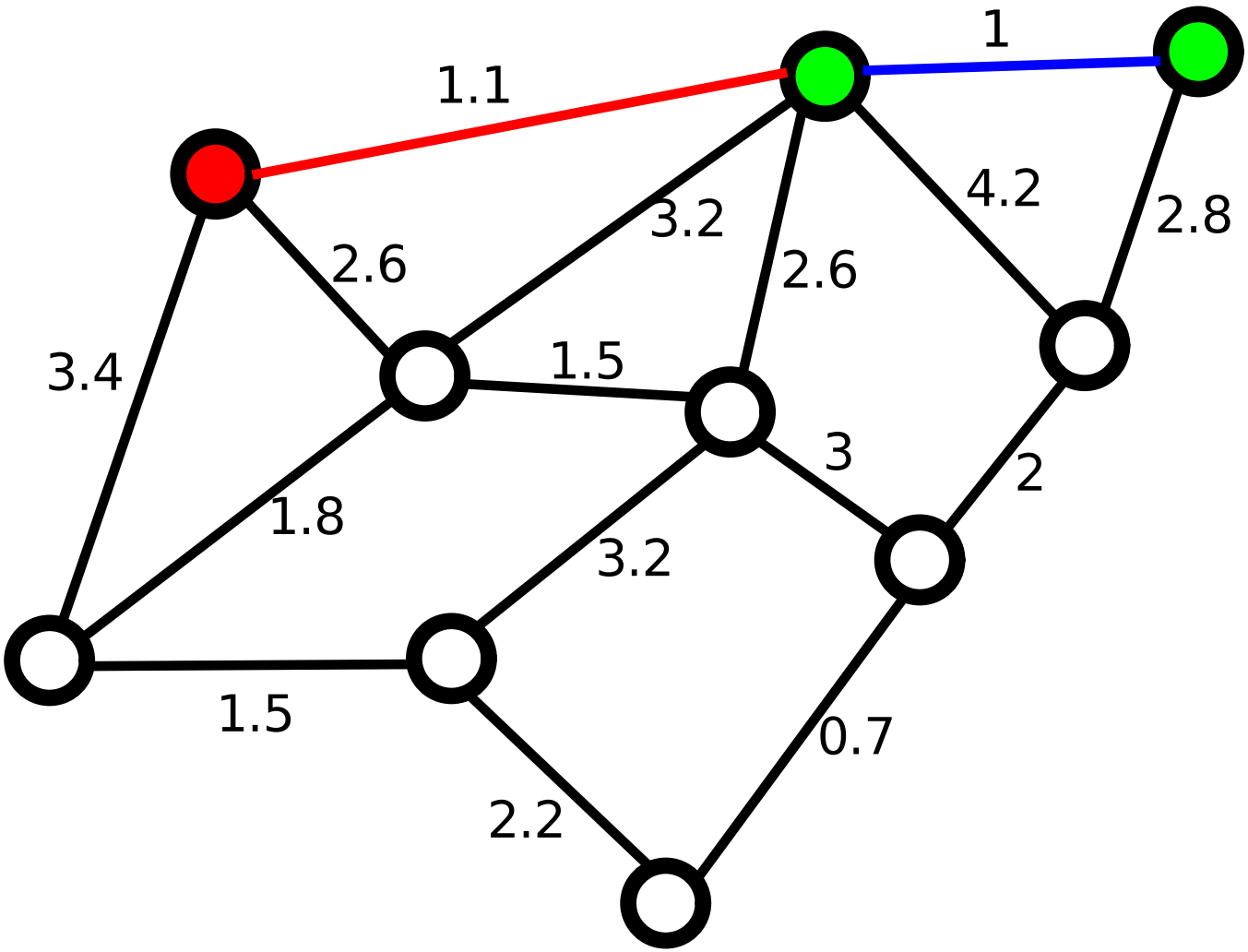
Algoritmo de Prim



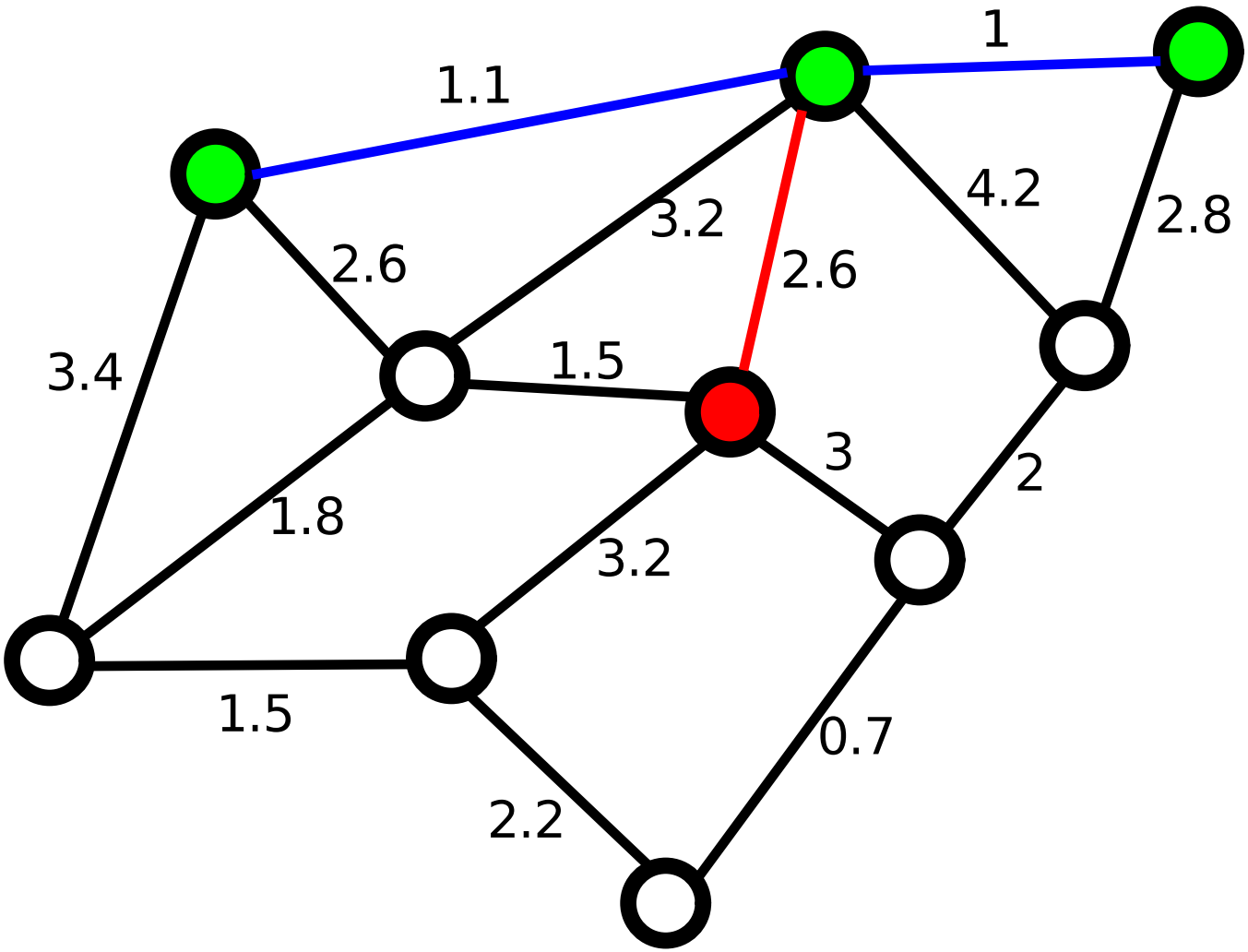
Algoritmo de Prim



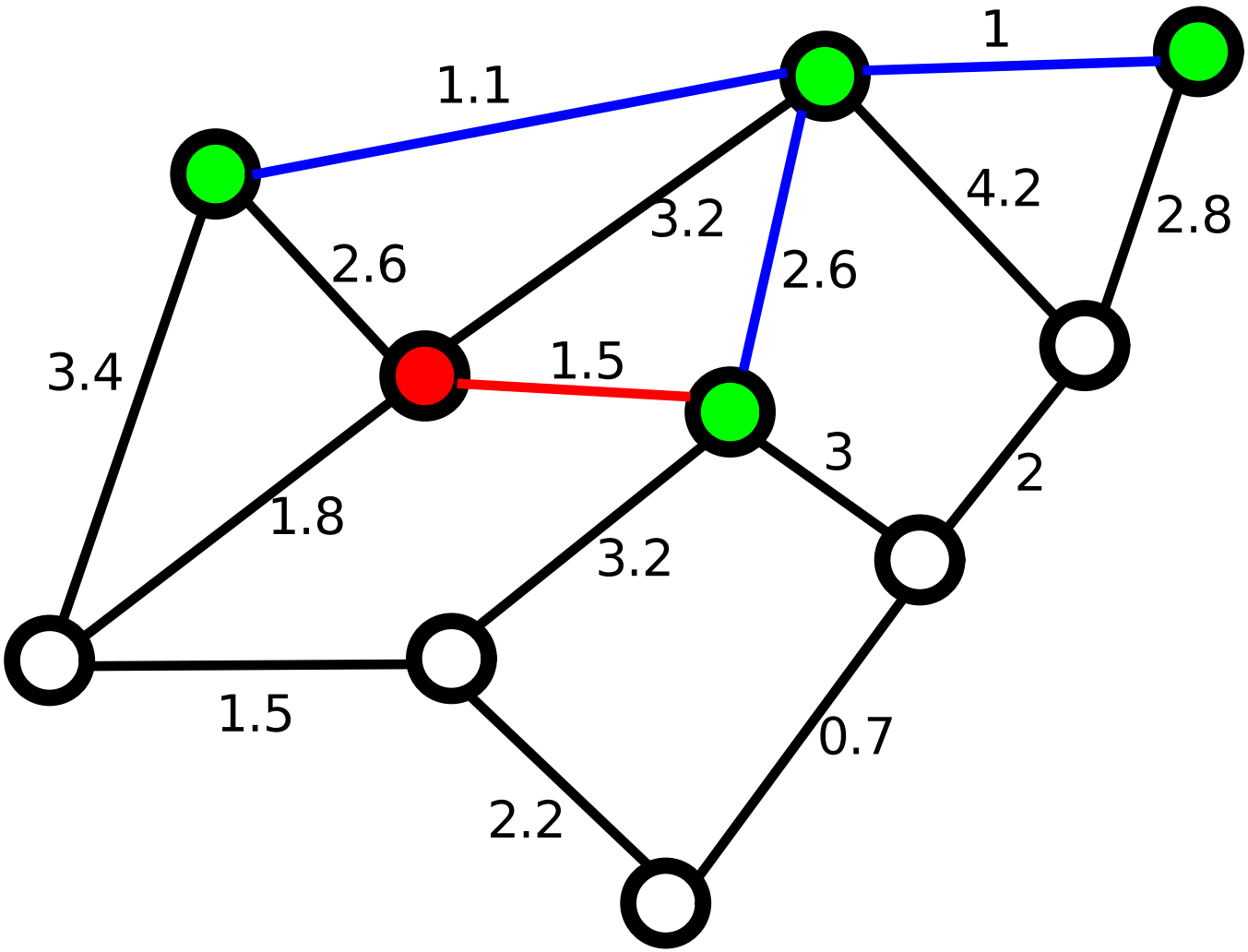
Algoritmo de Prim



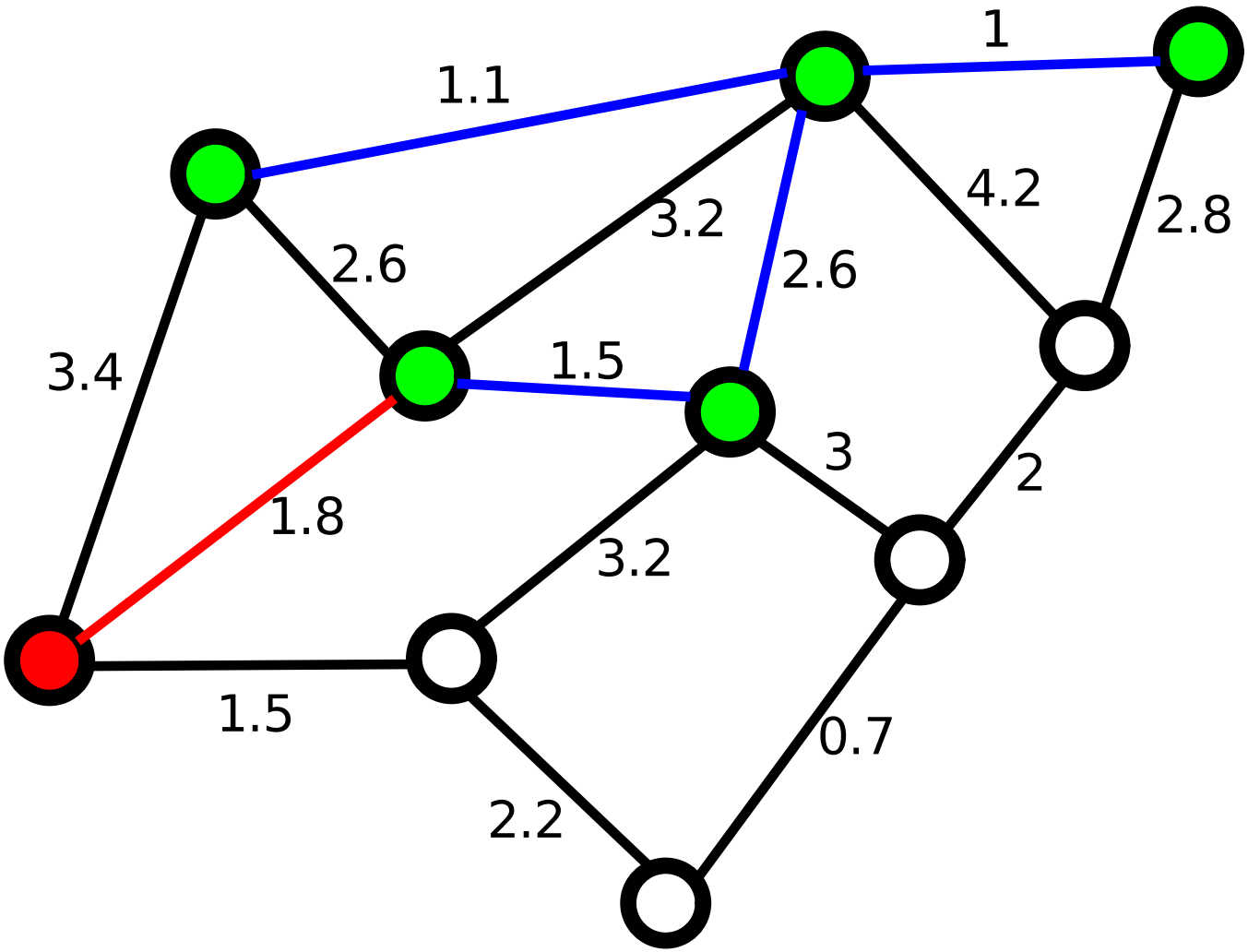
Algoritmo de Prim



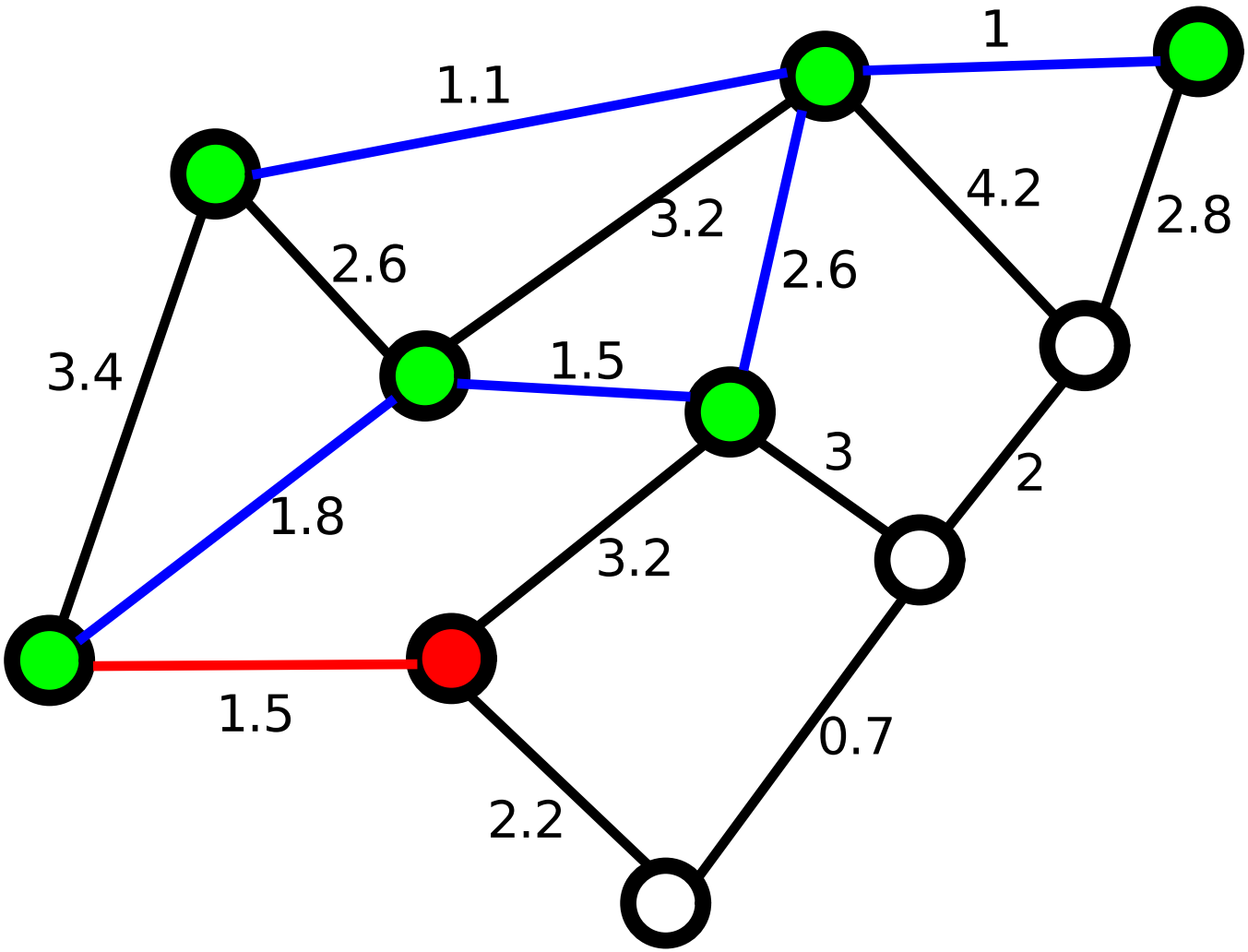
Algoritmo de Prim



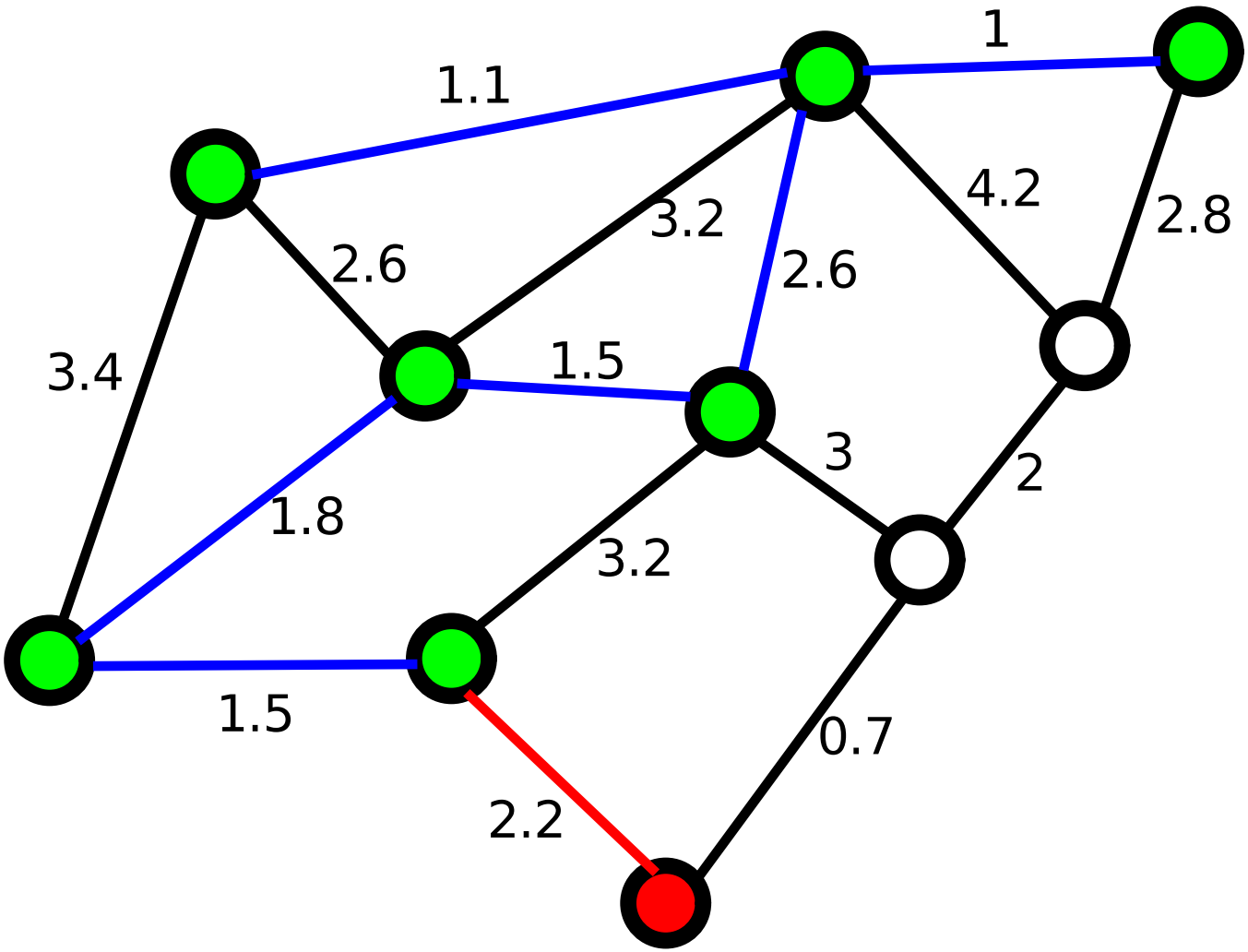
Algoritmo de Prim



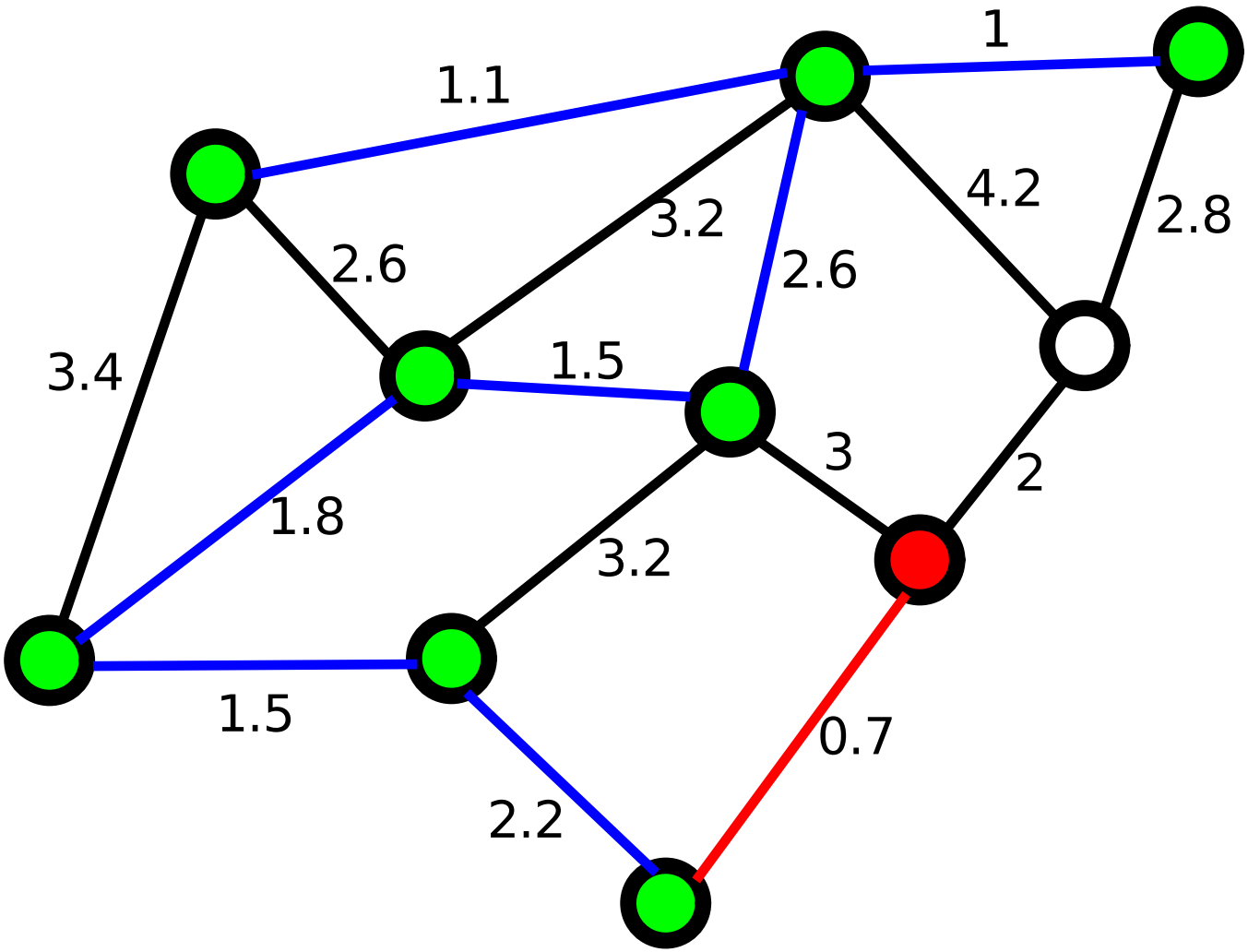
Algoritmo de Prim



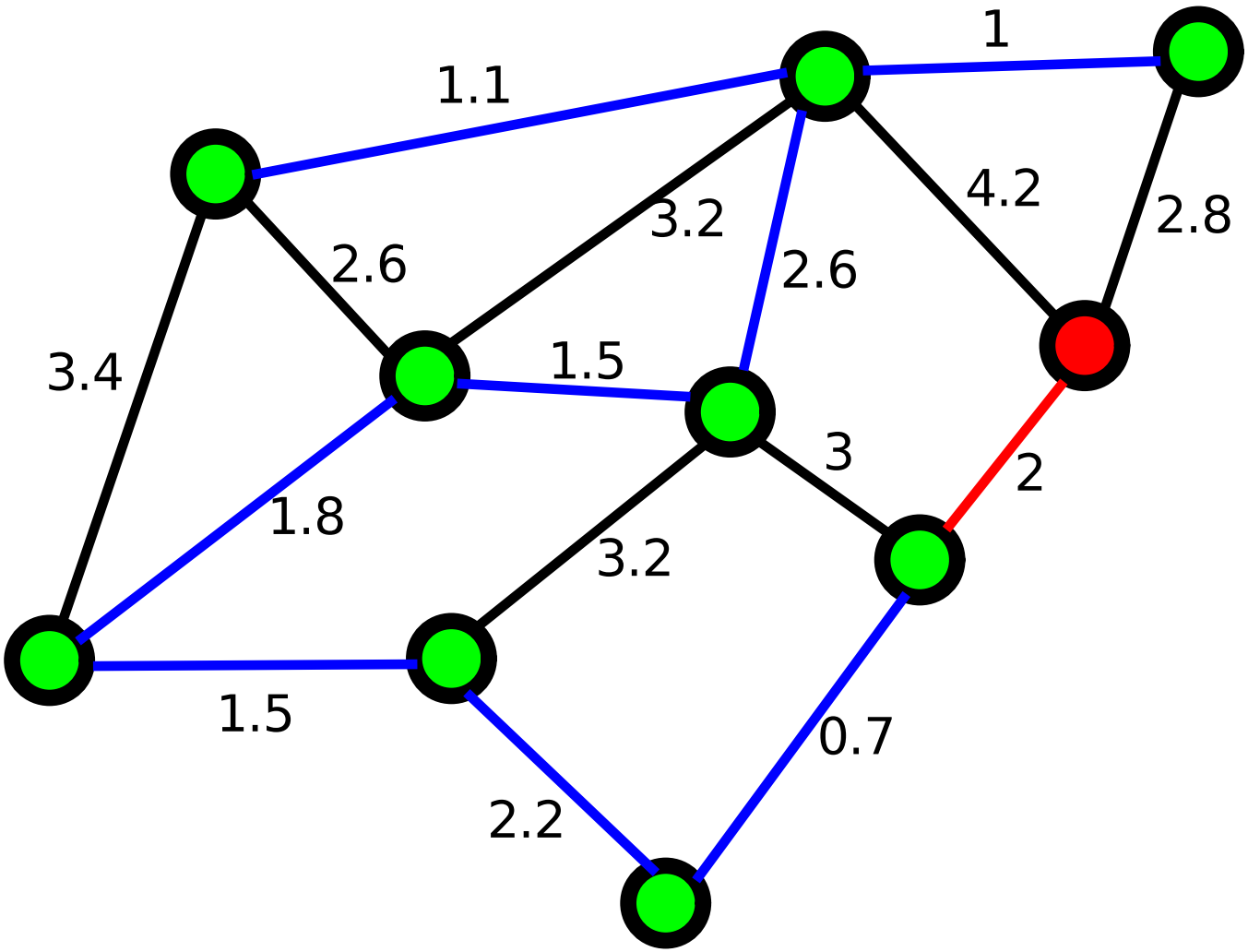
Algoritmo de Prim



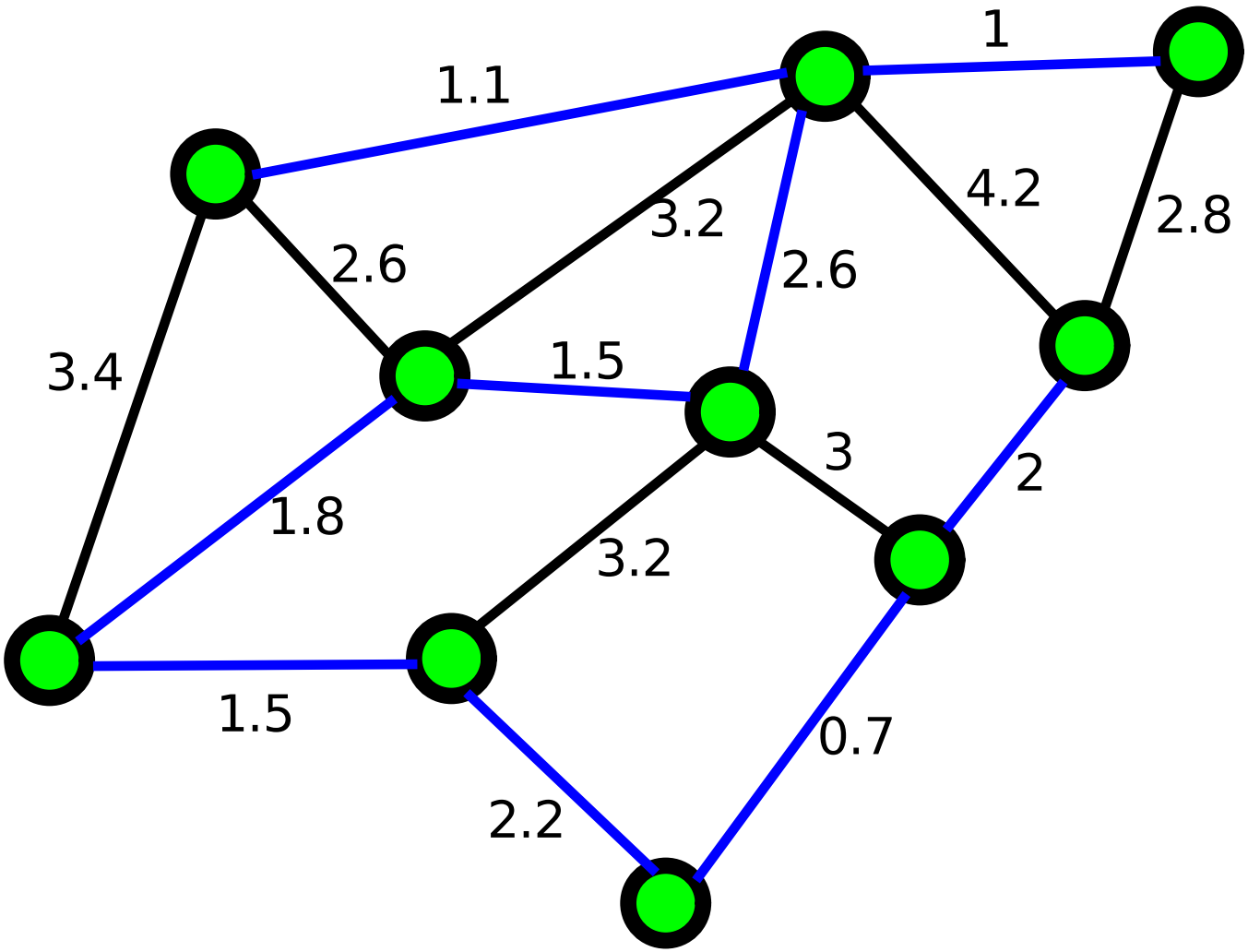
Algoritmo de Prim



Algoritmo de Prim



Algoritmo de Prim



¿Por qué funciona?

- Sea G una gráfica conexa con pesos y sea P el árbol generado por Prim.

¿Por qué funciona?

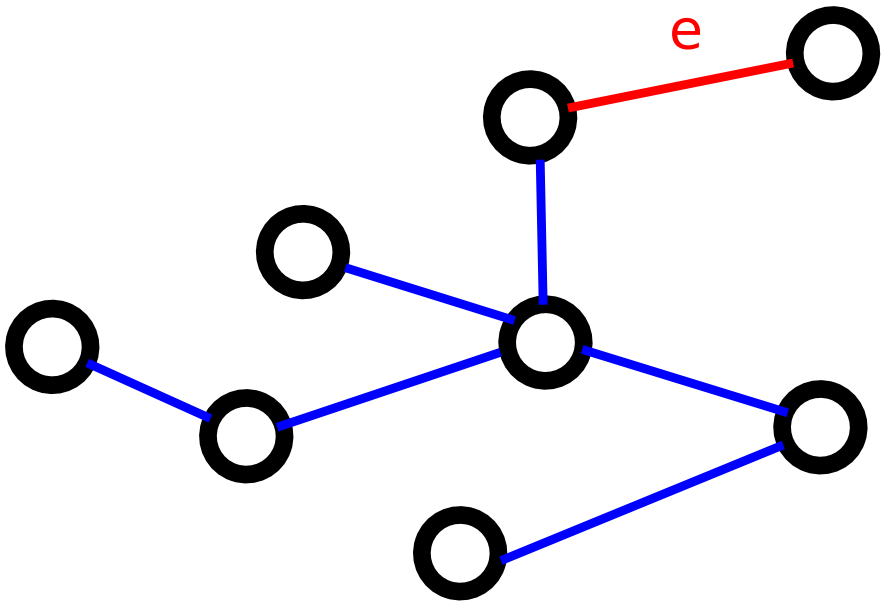
- Sea G una gráfica conexa con pesos y sea P el árbol generado por Prim.
- Sea A un árbol mínimo (luego le ponemos más condiciones).

¿Por qué funciona?

- Sea G una gráfica conexa con pesos y sea P el árbol generado por Prim.
- Sea A un árbol mínimo (luego le ponemos más condiciones).
- Si P fue generado por Prim, veamos el orden en que fuimos agregando aristas.

Prueba

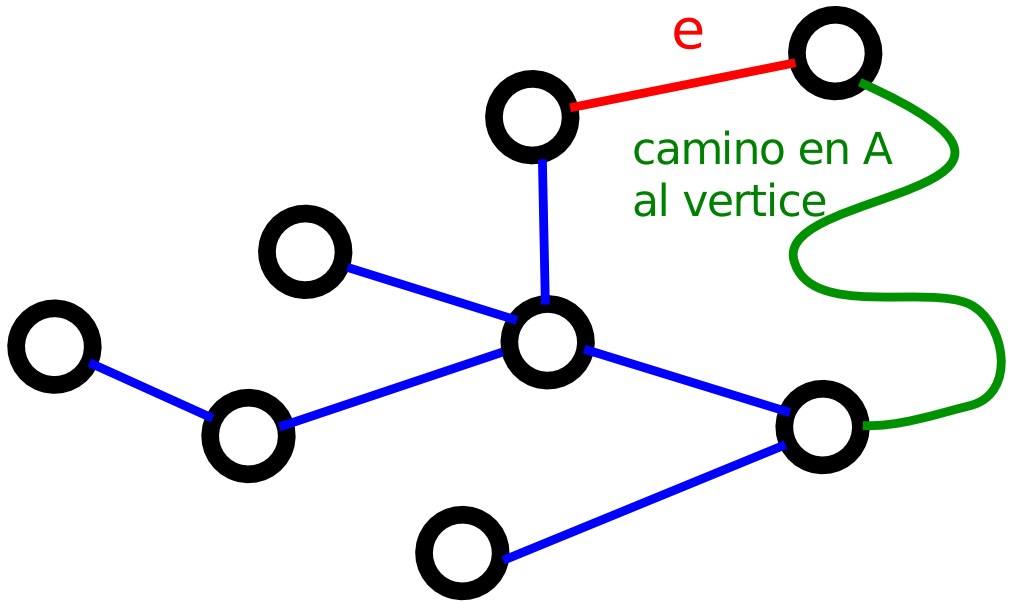
Sea e la primera arista que fue agregada por Prim que NO estaba en A.



e = primer nodo en P que no está en A

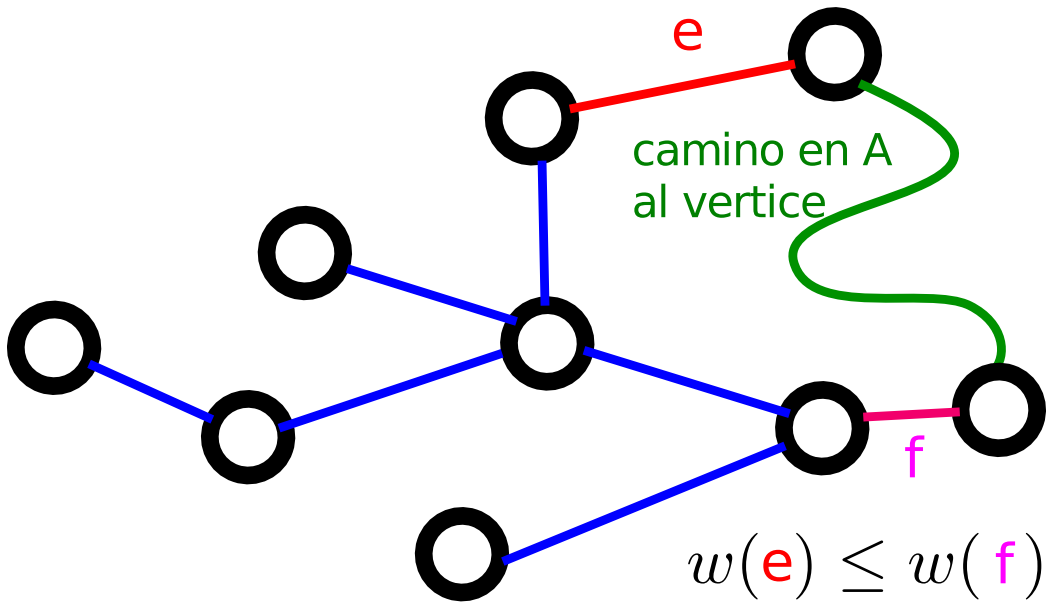
Prueba

Por ser A conexa, existe camino de los vértices que ya estaban al nuevo nodo:



Prueba

Sea f la primera arista del camino verde:



Prueba

Si agregamos e a A y quitamos f, construimos un árbol mínimo con más aristas en común con P que A.

Estructuras de datos

- ¿Qué estructura de datos utilizo para guardar las aristas “por explorar”? Debo:

Estructuras de datos

- ¿Qué estructura de datos utilizo para guardar las aristas "por explorar"? Debo:
  - Seleccionar la de peso más chiquito y quitarla.

Estructuras de datos

- ¿Qué estructura de datos utilizo para guardar las aristas "por explorar"? Debo:
  - Seleccionar la de peso más chiquito y quitarla.
  - Insertar varias nuevas aristas.

Estructuras de datos

- ¿Qué estructura de datos utilizo para guardar las aristas "por explorar"? Debo:
  - Seleccionar la de peso más chiquito y quitarla.
  - Insertar varias nuevas aristas.
- Igual que antes: ¡Priority Queue! (heap usualmente)

Estructuras de datos

- ¿Qué estructura de datos utilizo para guardar las aristas "por explorar"? Debo:
  - Seleccionar la de peso más chiquito y quitarla.
  - Insertar varias nuevas aristas.
- Igual que antes: ¡Priority Queue! (heap usualmente)
- ¿Qué estructura de datos utilizo para guardar los nodos? Las operaciones que haré son:

Estructuras de datos

- ¿Qué estructura de datos utilizo para guardar las aristas "por explorar"? Debo:
  - Seleccionar la de peso más chiquito y quitarla.
  - Insertar varias nuevas aristas.
- Igual que antes: ¡Priority Queue! (heap usualmente)
- ¿Qué estructura de datos utilizo para guardar los nodos? Las operaciones que haré son:
  - Guardar nuevos nodos.

Estructuras de datos

- ¿Qué estructura de datos utilizo para guardar las aristas "por explorar"? Debo:
  - Seleccionar la de peso más chiquito y quitarla.
  - Insertar varias nuevas aristas.
- Igual que antes: ¡Priority Queue! (heap usualmente)
- ¿Qué estructura de datos utilizo para guardar los nodos? Las operaciones que haré son:
  - Guardar nuevos nodos.
  - Revisar si uno ya fue guardado o no.

Estructuras de datos

- ¿Qué estructura de datos utilizo para guardar las aristas "por explorar"? Debo:
  - Seleccionar la de peso más chiquito y quitarla.
  - Insertar varias nuevas aristas.
- Igual que antes: ¡Priority Queue! (heap usualmente)
- ¿Qué estructura de datos utilizo para guardar los nodos? Las operaciones que haré son:
  - Guardar nuevos nodos.
  - Revisar si uno ya fue guardado o no.
- Lo que más conviene es un arreglo L de 0's y 1's, en donde

$$L[n] = 1 \iff n \text{ ya fue explorado.}$$

Consideraciones

- Si quiero el árbol de máximo peso en vez del árbol del mínimo peso, multiplico por -1, o equivalentemente, siempre tomo el “mayor” en vez de el “menor” que salga de lo que llevo.

Consideraciones

- Si quiero el árbol de máximo peso en vez del árbol del mínimo peso, multiplico por -1, o equivalentemente, siempre tomo el "mayor" en vez de el "menor" que salga de lo que llevo.
- El algoritmo, usando heaps como describimos anteriormente, corre en tiempo $O((E+V)\log V) \approx O(E \log V)$.

Consideraciones

- Si quiero el árbol de máximo peso en vez del árbol del mínimo peso, multiplico por -1, o equivalentemente, siempre tomo el "mayor" en vez de el "menor" que salga de lo que llevo.
- El algoritmo, usando heaps como describimos anteriormente, corre en tiempo $O((E+V)\log V) \approx O(E \log V)$.
- Si en vez de heaps se utiliza algo llamado "Fibonacci heap" puede ser disminuido a $O(E + V \log V)$.

Consideraciones

- Si quiero el árbol de máximo peso en vez del árbol del mínimo peso, multiplico por -1, o equivalentemente, siempre tomo el "mayor" en vez de el "menor" que salga de lo que llevo.
- El algoritmo, usando heaps como describimos anteriormente, corre en tiempo $O((E+V)\log V) \approx O(E \log V)$.
- Si en vez de heaps se utiliza algo llamado "Fibonacci heap" puede ser disminuido a $O(E + V \log V)$.
- Hay otros algoritmos que funcionan, unos incluso teóricamente más rápidos. Ustedes probarán en la siguiente tarea que funcionan.

Pseudo-código

NodosExplorados $= \{$algún nodo al azar$\}$
AristasPorExplorar $=$ aristas del nodo escogido antes
Arbol $= \emptyset$
while AristasPorExplorar $\neq \emptyset$:

- Selecciona arista s de menor peso de AristasPorExplorar y quítala.
- Sean n y m los dos nodos de s.
- if NodosExplorados[n] $==$ 0:
  - Añade s a Arbol y n a NodosExplorados
  - Añade todas las demás aristas que inician en n a AristasPorExplorar, si su otro nodo no está en NodosExplorados.
- else if NodosExplorados[m] $==$ 0:
  - Añade s a Arbol y m a NodosExplorados
  - Añade todas las demás aristas que inician en m a AristasPorExplorar, si su otro nodo no está en NodosExplorados.

return Arbol