PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

MAESTRÍA EN REDES DE COMUNICACIÓN

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DE MASTER EN REDES DE TELECOMUNICACIONES

"SELECCIÓN DE RUTAS EN UNA RED DE SENSOR INALAMBRICA, EN BASE AL NIVEL DE BATERIA Y DISTANCIA ENTRE NODOS SENSORES MEDIANTE LA UTILIZACION DEL ALGORITMO DE KRUSKAL"

ING. MESIAS BASURTO DIANA CAROLINA

Quito, Noviembre 2016

Tabla de contenido

CAPITULO I

ANALISIS DEL PROBLEMA

1.1. Introducción

"En los años 90, las redes han revolucionado la forma en la que las personas y las organizaciones intercambian información y coordinan sus actividades". (Sanchis, 2007). Los sistemas de monitoreo inalámbricos de bajo costo han adquirido gran importancia actualmente, esto gracias a los avances tecnológicos en modulación digital y a las necesidades de la industria en la adquisición de datos en línea, por estos motivos se han desarrollado sistemas de sensores cada vez más económicos, de bajo consumo energético, y se pueden instalar, controlar y monitorear de manera remota.

Las redes de sensores inalámbricas pueden estar constituidas por decenas, cientos o miles de pequeños computadores (sensores) que operan con baterías, que son ubicados según las necesidades que se tenga (Cantillo, 2010). Cada nodo recolecta datos de su ambiente, ya sea luz, temperatura, humedad, energía, etc., enviando los datos recolectados a sus vecinos, éstos a su vez a sus propios vecinos y así sucesivamente, hasta que la información alcance un destino específico, donde será procesada dicha información, en tiempo real (Cantillo, 2010). La importancia de este proyecto radica en la utilización de la tecnología para optimizar el consumo de energía de los nodos sensores, y de esta manera optimizar los costos por mantenimiento de la red y propiciar el uso de este tipo de redes en el Internet de las Cosas, con aplicaciones en domótica, monitoreo y control ambiental, entre otras aplicaciones.

Esta tesis, tiene como objetivo, proponer un mecanismo que ayude a la resolución del problema de consumo de energía de los nodos, mediante la aplicación del algoritmo de Kruskal en una topología de una red de sensores inalámbrica.

1.2. Antecedentes

El algoritmo de Kruskal fue descubierto en el año 1956 por Joseph B. Kruskal cuando realizaba un estudio para la resolución de un problema de optimización combinatoria en grafos. Previamente antes del descubrimiento del Algoritmo de Kruskal, Otakar Boruvka ya había realizado un estudio similar (Villalobos, 2003). "Al algoritmo de Kruskal se lo conoce como el algoritmo tacaño ya que siempre busca el árbol con el menor coste" (Alejo, 2013) .

Para tener un mejor entendimiento del algoritmo de Kruskal es necesario conocer que es un árbol de expansión.

Un árbol de expansión pertenece a la teoría de grafos, este consiste en un árbol que contiene varios vértices y algunas aristas que conectan los vértices de manera que exista una arista por cada par de vértices (Figueroa, 2012).

1.3. Funcionamiento del Algoritmo de Kruskal

Los pasos a seguir para la aplicación del Algoritmo de Kruskal son:

1. Ordenar las aristas de menor a mayor peso
2. Unir las aristas con sus vértices siempre y cuando estos no formen ciclos
3. El punto anterior se lo realiza con todas las aristas tomando en cuenta que no estén en la misma componente (Cárdenas, Perez, & Barrera, 2010).

Por lo tanto el algoritmo de Kruskal es un proceso que permite unir todos los nodos de un grafo formando un árbol, tomando en cuenta el peso de las aristas y cuyo coste total es el

mínimo posible. Este principio puede ser aplicado en el tema de enrutamiento, cuando se tiene varios nodos comunicados entre sí, este enrutamiento estático, se basa en la creación de una topología tipo árbol para que los nodos de la red puedan enviar su información al nodo raíz que en este caso será el Gateway que permite la conexión a internet.

1.4. Objetivos

1.4.1. Objetivo General

Seleccionar la ruta óptima para optimizar el consumo de energía de los nodos que forman parte de una red de sensores inalámbricos (WSN) mediante la obtención del mejor camino de transmisión considerando como datos de entrada el nivel de batería, posición de los nodos y la utilización del algoritmo de Kruskal.

1.4.2. Objetivos Específicos

- Encontrar el mejor camino de transmisión de datos en una topología de red WSN utilizando el algoritmo de Kruskal.
- Relacionar el nivel de batería y distancia como peso para el algoritmo.
- Proponer un procedimiento para calcular el tiempo en el cual el Gateway debe solicitar nuevamente la actualización de datos. El algoritmo tomará como información inicial la potencia consumida por el nodo sensor, y un lapso de tiempo en el cual el nodo sensor está transmitiendo los datos por una ruta.
- Simular la propuesta, para validar los resultados obtenidos e implementar el prototipo

1.5. Justificación

Es común ver día a día el incremento del uso de redes inalámbricas tanto en hogares como en empresas. El costo que implica el cambio de una tecnología cableada a una inalámbrica puede

llegar a ser muy elevada, pero al usar redes de sensores inalámbricas supone un costo menor y se utiliza de mejor manera los recursos que se tienen. Las redes de sensores inalámbricas presentan varios problemas, siendo uno de ellos el consumo de energía de los nodos, mediante la presente tesis se propone encontrar un mecanismo para que los nodos comuniquen el valor de la energía que poseen a los demás nodos y de esta manera optar por una nueva ruta de comunicación, permitiendo el ahorro de batería de los nodos que presenten valores muy bajos y así permitir un consumo homogéneo de las baterías, para así realizar un solo cambio de fuentes de poder de los sensores lo que implica un solo gasto para la empresa u hogar. El tema de la presente tesis, forma parte de los problemas a resolver encontrados en el desarrollo de aplicaciones de redes de sensores inalámbricas, además de otros temas de tesis de la misma área en el cual están trabajando los graduados.

1.6. Alcance del proyecto

El algoritmo de Kruskal es un proceso que permite unir todos los nodos de un grafo formando un árbol, tomando en cuenta el peso de las aristas y cuyo coste total es el mínimo posible (Villalobos, 2003). Este principio puede ser aplicado en el tema de enrutamiento, cuando se tiene varios nodos comunicados entre sí. Este enrutamiento estático, se basa en la creación de una topología tipo árbol para que los nodos de la red puedan enviar su información al nodo raíz que en este caso será el Gateway que permite la conexión a internet.

El siguiente trabajo consiste en aplicar del algoritmo de Kruskal en una red de sensores inalámbricos (WSN) que utiliza el nivel de batería de los nodos y la posición de los mismos para la selección de rutas con el Gateway de acceso a internet, con el propósito de optimizar el tiempo de duración de las baterías de los nodos, es decir minimizar el consumo de energía que se requiere en los procesos de comunicación de cada nodo con el nodo Gateway.

El software en el cual se implementará el algoritmo de Kruskal dispondrá de una interface gráfica, donde el usuario podrá ver los nodos que forman parte de la red WSN. Inicialmente la topología será tipo malla, es decir la ubicación de nodos y aristas de forma aleatoria, donde todos los nodos están comunicados entre sí, cuyas aristas poseerán un peso que corresponderá a la carga de la batería y la distancia entre los nodos obteniéndose este valor a partir de la posición de cada uno de ellos. La resolución del problema, toma como punto de partida, el hecho de que el Gateway de la red WSN no tiene limitaciones de procesamiento y de energía, además es este nodo el que determinará a que nodos sensores debe enviar la información. Es por esta razón que la solución determina que el algoritmo de Kruskal se ejecutará en el nodo Gateway con el propósito de definir las rutas por las cuales los nodos sensores deberán enviar la información a dicho Gateway y así permitir optimizar el consumo de energía de los nodos sensores.

Mediante este trabajo se busca desarrollar una alternativa que permita optimizar el consumo de energía de los nodos sensores, y que todos ellos tengan la posibilidad de tener el mismo tiempo de vida, y que los cambios de batería sean realizados de manera eficiente en un solo momento para todos los nodos.

El aporte principal del trabajo consistirá en proponer un nuevo proceso, que calculará el tiempo en el cual el Gateway debe solicitar una nueva actualización de los datos de los nodos sensores, en base a la información proporcionada cuando se ejecute el algoritmo de Kruskal, cuyos datos serán almacenados, de tal forma que puedan ser utilizados por el programa y de esta manera proceder a actualizar las rutas (nuevo árbol) seleccionadas ya sea por disminución de la energía de las baterías de los nodos o por modificación de la topología de la red sensor inalámbrica sin necesidad de esperar que un nodo deje de formar parte de la topología, además permitirá generar una alarma de cuando una batería tiene que ser reemplazada.

1.7. Resumen de capítulos

La presente tesis consta de 4 capítulos, el primer capítulo posee toda la información que antecede para la realización de este proyecto siendo los temas estudiados justificación, antecedentes y los objetivos a cumplir con esta tesis

El segundo capítulo contiene el algoritmo de Kruskal, sus aplicaciones, estándares utilizados comparación entre ellos, etc.

En el tercer capítulo se indicó como se realizó el diseño, construcción e implementación del algoritmo de Kruskal en un software, así como las pruebas y resultados que se realizaron con el mismo.

Para finalizar se encuentran las conclusiones y recomendaciones de la presente tesis.

CAPITULO II

MARCO TEORICO

2.1. Historia Algoritmo de Kruskal

El algoritmo de Kruskal fue elaborado y publicado en el año 1956 por Joseph B. Kruskal cuando realizaba un estudio para la resolución de un problema de optimización combinatoria en grafos. Previamente antes del descubrimiento del Algoritmo de Kruskal, Otakar Boruvka ya se había realizado un estudio similar (Villalobos, 2003).

Para tener un mejor entendimiento del algoritmo de kruskal es necesario conocer que es un árbol de expansión.

2.1.1. Árbol de expansión

En teoría de grafos, un árbol de expansión de un grafo conexo no dirigido es un árbol que posee todos los vértices y algunas o todas las aristas. Adicional también se puede decir que un árbol de expansión es una selección de aristas que forman un árbol que cubre todos los vértices, es decir que cada vértice está en el árbol, pero no hay ciclos (Fundación Wikimedia).

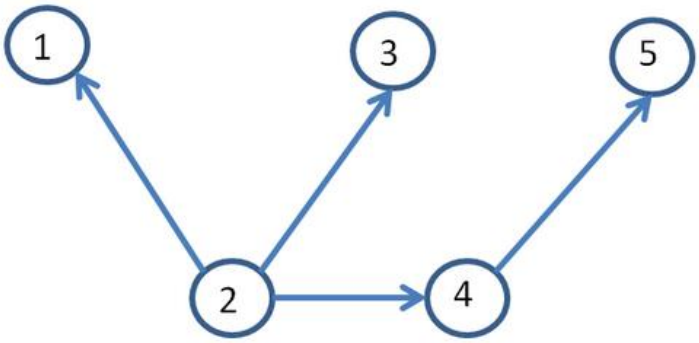


Figura 1. Árbol de Expansión

Un árbol de expansión de un grafo conexo puede ser también definido como el mayor conjunto de aristas que no contiene ciclos, o como el mínimo conjunto de aristas que conecta todos los vértices. (Fundación Wikimedia)

En ciertos campos de la teoría de grafos es útil encontrar el mínimo árbol de expansión de un grafo ponderado.

2.1.2. Algoritmo de Kruskal

El algoritmo de Kruskal consiste en la obtención de un árbol cuyas aristas al ser sumados sus pesos el valor obtenido es el mínimo, tomando el nombre de algoritmo voraz.

El algoritmo está basado en una propiedad de los árboles que permite estar seguros de sí un arco debe pertenecer al árbol o no, y usar esta propiedad para seleccionar cada arco. Se debe tener en cuenta en el algoritmo que siempre que se añade una arista, ésta será siempre la conexión más corta (menor coste) alcanzable desde el nodo que se parte al resto del grafo. Así

que por definición éste deberá ser parte del árbol. Este algoritmo es de tipo greedy, ya que a cada paso, selecciona la arista de menor valor y lo añade al sub-grafo (Villalobos, 2003).

Se debe tomar en cuenta que puede existir más de una solución que tenga el mismo valor mínimo en especial cuando los valores de las aristas se repiten (Alonso Revenga, 2008).

2.1.3. Resolución del Algoritmo de Kruskal

Para generar el árbol de mínima expansión se va a partir desde el siguiente Grafo.

- Seleccionar el arco o arista de menor valor

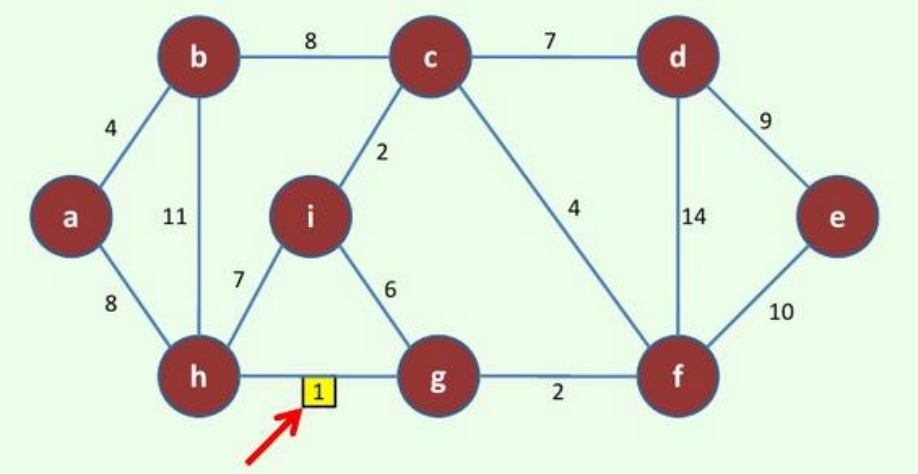


Figura 2. Selección de arco menor valor

- Seleccionar la arista con menor valor de toda la red incluyendo los nodos que la conforman

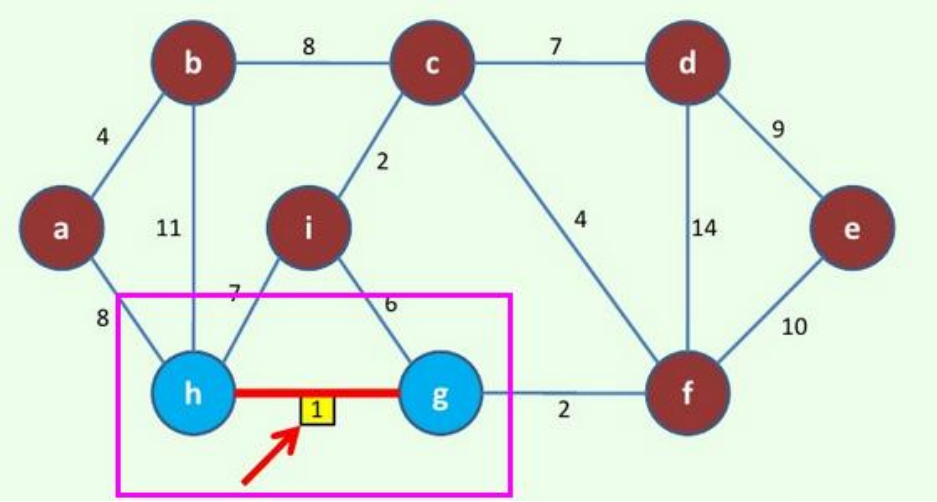


Figura 3. Selección de nodos unidos por la arista

- Los dos pasos anteriores se lo debe hacer con todo el grafo y cuando se empiezan a formar ciclos, estos se deben eliminar

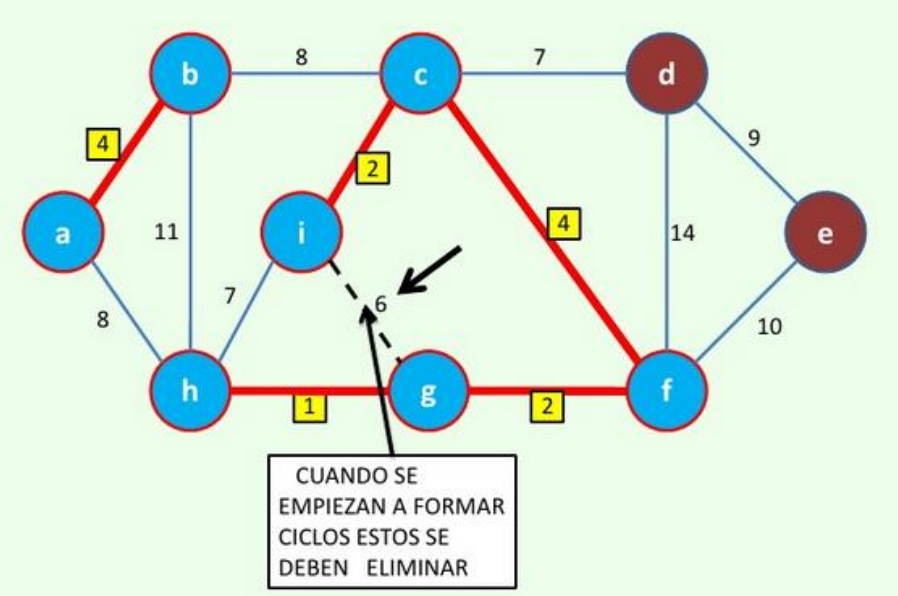


Figura 4. Eliminar ciclos

En este caso se han generado dos soluciones como se observa a continuación

Solución 1

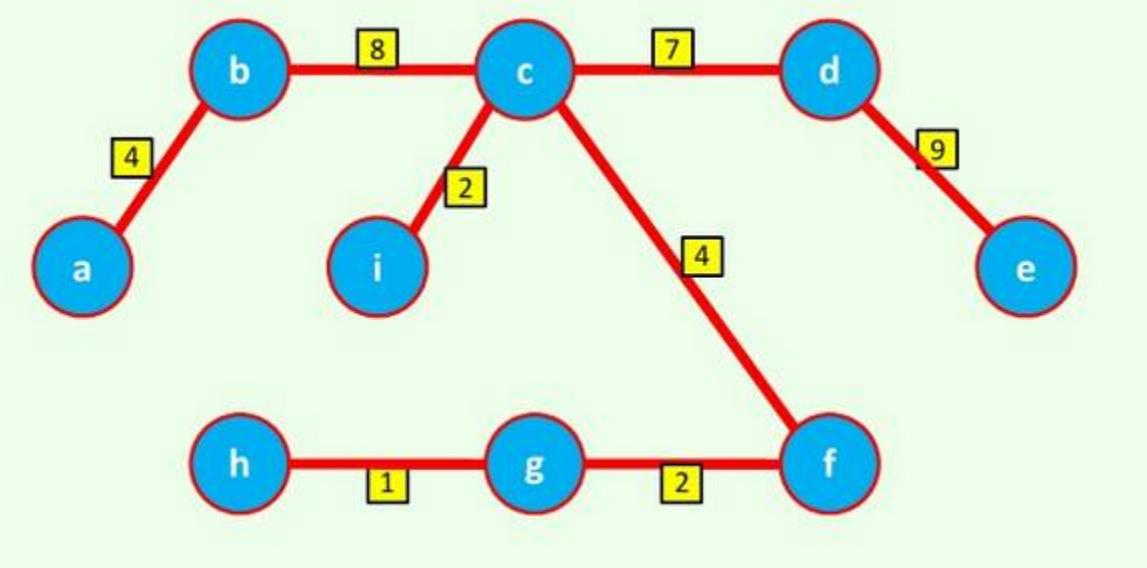


Figura 5. Solución 1 Algoritmo Kruskal

Solución 2

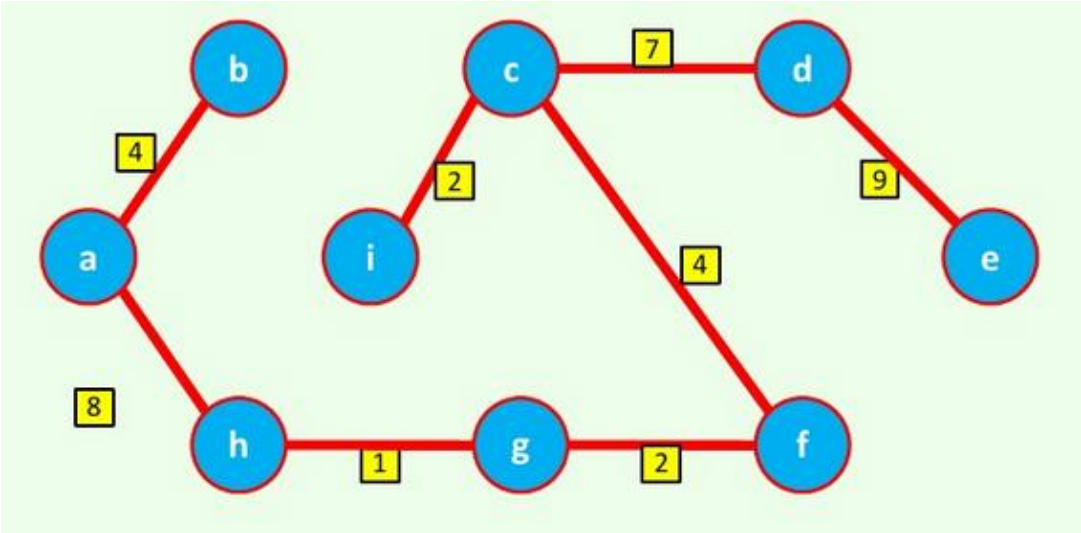


Figura 6. Solución 2 Algoritmo Kruskal

- Se deben sumar el valor de las aristas de las dos soluciones encontradas

Solución 1

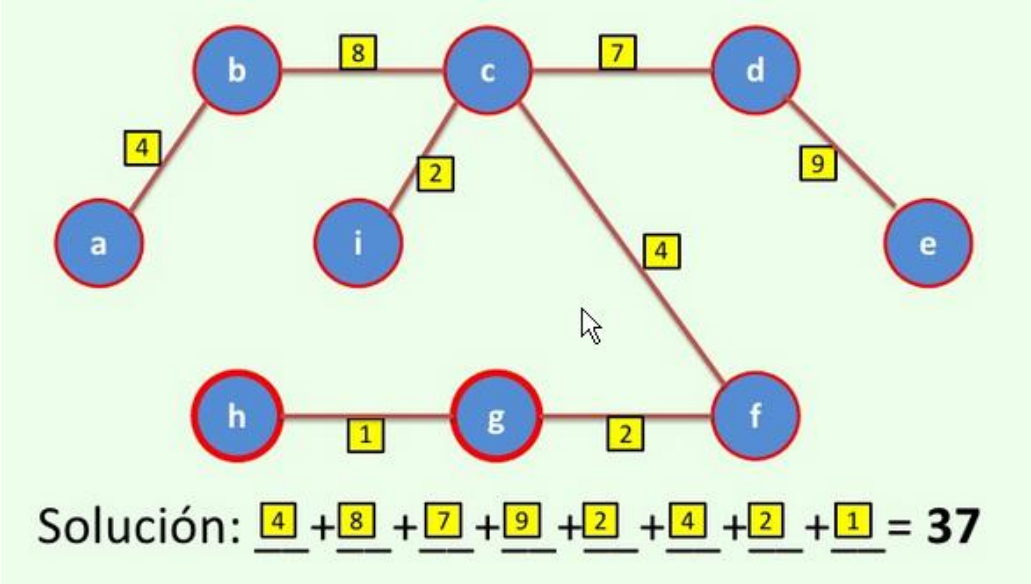


Figura 7. Suma de aristas

Solución 2

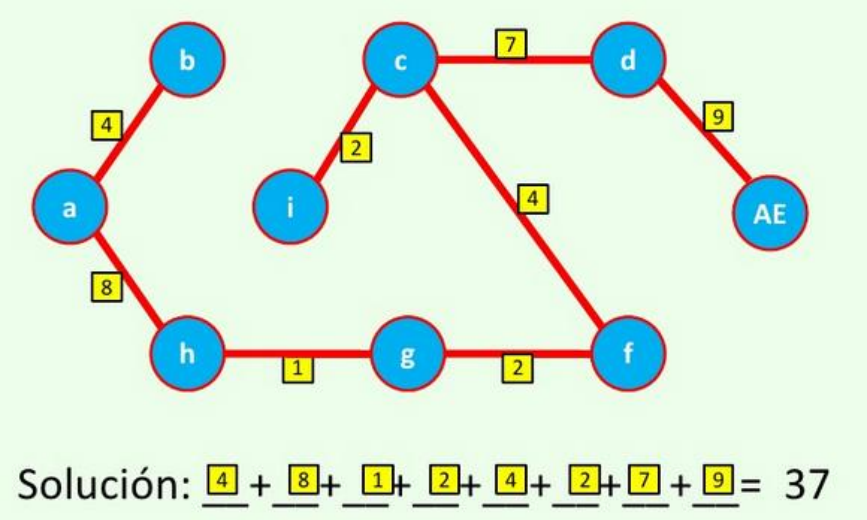


Figura 8. Suma de aristas

En este caso se encontraron 2 soluciones óptimas del algoritmo de Kruskal.

2.1.4. Aplicación del algoritmo de Kruskal

Una de las aplicaciones más comunes del algoritmo de Kruskal está en el diseño de redes telefónicas, ya que al querer interconectar unas líneas telefónicas con otras se utiliza el algoritmo para realizar la interconexión al menor costo.

El algoritmo de Kruskal también ha sido aplicado para hallar soluciones en diversas áreas como es el diseño de redes de transporte, telecomunicaciones, TV por cable, sistemas distribuidos, interpretación de datos climatológicos, visión artificial, entre otros (Villalobos, 2003).

Otra aplicación menos conocida del algoritmo de Kruskal es que el árbol de coste total mínimo que se obtiene puede ser usado como solución aproximada al problema del viajante de comercio, ya que la manera formal de definir este problema es encontrar la trayectoria más corta para visitar cada punto al menos una vez. Nótese que si se visitan todos los puntos exactamente una vez, lo que se tiene es un tipo especial de árbol (Villalobos, 2003).

2.2. Red de sensores inalámbricos

En la actualidad las redes de sensores inalámbricos tienen gran acogida debido a sus múltiples usos ya que cubren todas las necesidades que tiene el cliente para una red de comunicación, es decir precios bajos, espacio físico, autonomía, etc.

2.2.1. Origen e Historia de red de sensores inalámbricos

Como muchas tecnologías avanzadas, el origen de las WSNs o RIS (Red Sensores Inalámbricos), pueden verse en aplicaciones militares e industriales del pasado, estas lejos de ser retiradas son frecuentes en la actualidad. La primera red inalámbrica que tiene cualquier parecido

a una WSN actual, es el Sistema de Vigilancia de sonido, desarrollado por el ejército de Estados Unidos en la década de 1950 para detectar y rastrear submarinos soviéticos. Esta red utiliza sensores acústicos sumergidos (hidrófonos) distribuidos en el Atlántico y el Pacífico, esta tecnología de detección todavía se encuentra en servicio hoy en día, aunque sirviendo funciones más pacíficas como es el monitoreo de la fauna submarina y la actividad volcánica.

Haciéndose eco de las inversiones realizadas en los años 1960 y 1970 para desarrollar el hardware para la Internet de hoy, también se dio inicio al programa de Redes de sensores distribuidos (DSN) en 1980 para explorar formalmente los desafíos en la implementación de redes inalámbricas de sensores que luego encontró un hogar en el mundo académico y la investigación científica. Con el tiempo los gobiernos y las universidades comenzaron a usar WSNs en aplicaciones tales como monitoreo de la calidad del aire, la detección de incendios forestales, la prevención de desastres naturales, estaciones meteorológicas y monitoreo estructural, etc. La ingeniería al abrirse paso en el mundo corporativo de los gigantes de la tecnología de la época, tales como IBM y Bell Labs comenzaron a promover el uso de redes inalámbricas de sensores en aplicaciones industriales, como la distribución de energía, tratamiento de aguas residuales y la automatización especializada en fábricas.

Mientras que la demanda del mercado de WSNs crecía día tras día, ir más allá de estas aplicaciones limitadas demostraba ser un desafío puesto que las aplicaciones industriales de esa época se basaban en sensores caros, voluminosos y protocolos de redes propietarias. Estas redes se colocaron en una posición privilegiada por su funcionalidad y rendimiento, mientras que por otros factores como el hardware, costos, estándares de redes, el consumo de energía y la escalabilidad cayeron al borde del camino. La combinación del alto costo y bajo volumen de

producción impidió la adopción generalizada y el despliegue de redes inalámbricas de sensores en una gama más amplia de aplicaciones.

Aunque la tecnología para aplicaciones industriales y de consumo de gran volumen no existía en el siglo 20 se reconoció el gran potencial que tienen las redes de sensores inalámbricos dando origen a grandes esfuerzos para resolver estos desafíos de ingeniería. La principal meta de este desafío era generar nuevos desarrollos y sensores cuyos costos fueran reducidos, accesibles y de poco mantenimiento. La reducción de los costos y tamaños de los sensores originó un incremento en la funcionalidad de varias áreas de la tecnología como por ejemplo en dispositivos semiconductores, protocolos de red, tecnologías de generación y almacenamiento de energía, dando origen a lo que ahora se conoce como red de sensores inalámbricos.

2.2.2. Características

Una red inalámbrica de sensores presenta las siguientes características

- Integración con otras tecnologías y ciencias como lo es la agricultura, biología, medicina, meteorología, etc.
- Permite aplicaciones que antes no se podían realizar permitiendo la interacción con los seres humanos y su entorno.
- Se genera un ahorro significativo de recursos
- Gran densidad, es decir se puede utilizar un número infinito de sensores, esto dependerá de la superficie en la que se va a trabajar
- Cooperación y coordinación, al poseer recursos limitados y ser propenso a fallos los nodos de una RIS deben trabajar conjuntamente para que la red trabaje sin inconvenientes

- Una red inalámbrica de sensores utiliza como mecanismo de comunicación de broadcast mientras que otros tipos de redes utiliza la comunicación punto a punto.
- Posee limitada su capacidad de almacenamiento y almacenamiento de energía
- Los nodos en una red inalámbrica de sensores no poseen un identificador global como lo es una dirección ip debido a la sobrecarga de envío de datos y el número elevado de sensores

2.2.3. Elementos y arquitectura de una red inalámbrica de sensores

2.2.3.1. Elementos

Como se mencionó con anterioridad una red inalámbrica de sensores posee dispositivos de bajo coste que se los conoce como SENSORES, los cuales obtienen la información del medio que los rodea, procesa los datos y la envía a manera de señales eléctricas. Las WSN también están compuestas de NODOS que son los encargados de tomar los datos de los sensores y enviar esta información a la estación base, una esta ESTACION BASE es encargada en recolectar los datos en un Pc normal cuyos datos se reflejaran en un aplicación ya sea de escritorio o web, también posee un GATEWAY, el cual posee los elementos para realizar la interconexión entre la red inalámbrica de sensores y una red de datos.

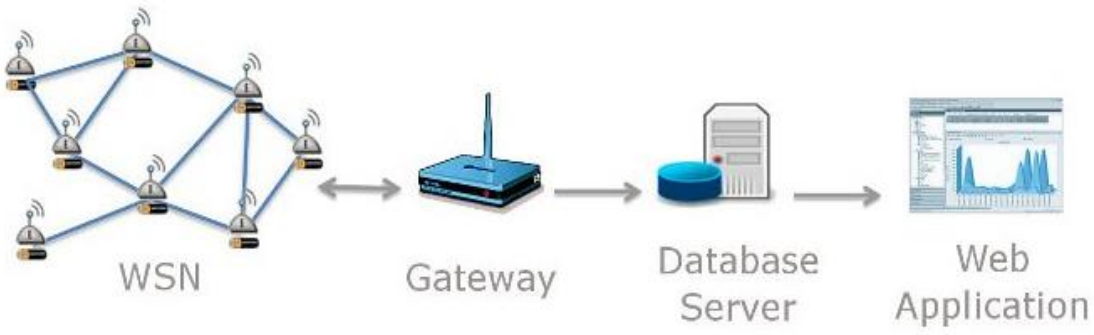


Figura 9. Elementos de una red de sensores inalámbrica

2.2.3.2. Arquitectura

Existen varias arquitecturas que puede presentar una red inalámbrica de sensores como se detalla a continuación:

- **Arquitectura Centralizada**

En este tipo de arquitectura los nodos envían sus datos directamente al Gateway más cercano, pero se debe tener en consideración que al utilizar este tipo de arquitectura se estará generando un cuello de botella, el consumo de energía será mucho mayor que al utilizar otro tipo de arquitectura, por lo tanto generando un tiempo de vida muchos más corto en este tipo de red.

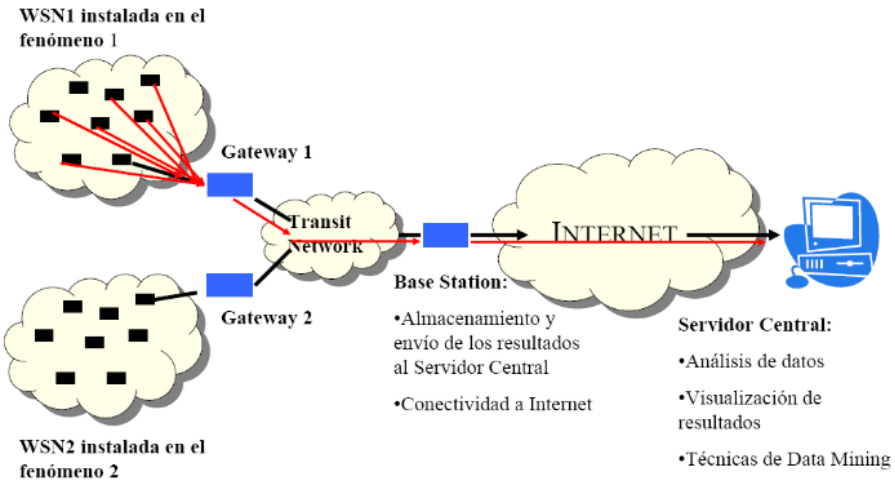


Figura 10. Arquitectura Centralizada WSN

- **Arquitectura Distribuida**

Una red inalámbrica de sensores al tener una cooperación entre sus nodos es más factible la utilización de una arquitectura distribuida ya que luego de interactuar entre nodos los datos

irán al Gateway más conveniente, de esta forma evitando los problemas que genera una red centralizada

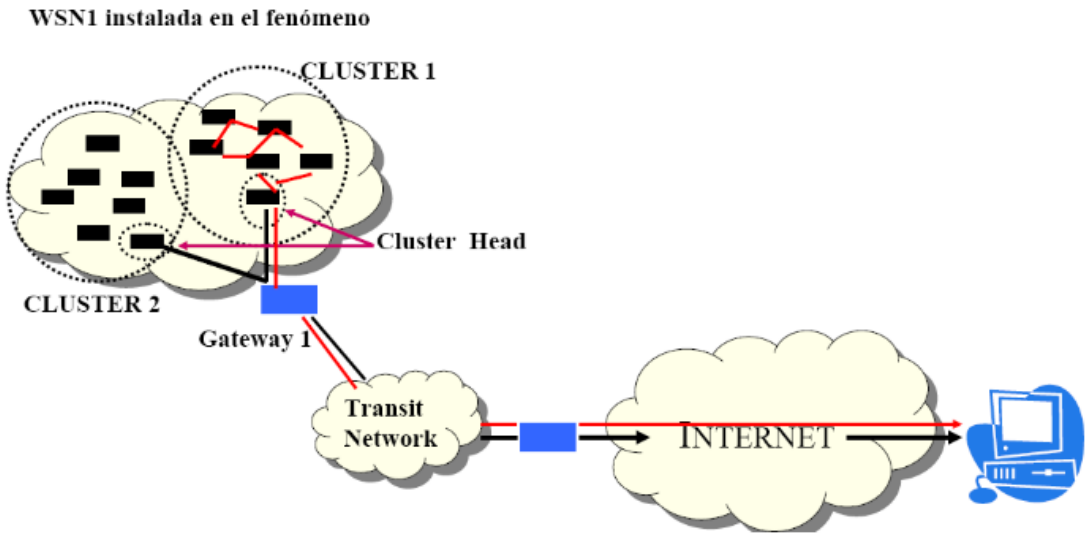


Figura 11. Arquitectura distribuida WSN

2.2.4. Aplicaciones y usos

Las redes inalámbricas de sensores pueden estar compuestas por una gran variedad de sensores que pueden ser sísmicos, ambientales, térmicos, etc. Uno de los usos que tiene este tipo de redes es en la monitorización de entornos, ya que los nodos se encuentran sincronizados y comunicados de manera periódica en entornos poco accesibles, en este tipo de redes no se requiere datos en tiempo real por lo que su topología es estable y la información recopilada se la utiliza para análisis futuros un ejemplo frecuente de este tipo de redes se lo encuentra en el monitoreo de la contaminación de una ciudad

Figura 12. Aplicaciones de WSN en la ciudad

Otra aplicación de una red de sensores inalámbrica la encontramos en la detección de inundaciones, estas redes poseen sensores que detectan lluvia, niveles de agua, cambios meteorológicos los cuales envían la información a la estación base en cuanto se producen cambios en el clima

La domótica es otro campo donde se utilizan las redes de sensores inalámbricos, ya que los sensores pueden ser colocados en la mayoría de los electrodomésticos puesto que sus tamaños son muy pequeños, estos sensores se comunican dentro de la habitación unos con otros y son manejados de manera remota por el usuario final.

Figura 13. WSN en Robótica

2.2.5. Ventajas y desventajas

Las redes de sensores inalámbricos presentas las siguientes ventajas y desventajas:

Ventajas

- Las redes de sensores inalámbricos presentan un tiempo de vida muy alto
- Sus coberturas pueden alcanzar distancias muy extensas
- Las RIS tienen costos bajos y son de fácil instalación
- Los tiempos de respuesta son rápidos
- Poseen precisión en sus mediciones

Desventajas

- Una de las desventajas de las redes inalámbricas es poco capacidad de almacenamiento de datos
- La energía de una baterías es de poca duración

- Cuando las redes no tienen intervención del ser humano presentan una alta probabilidad falla.

2.3. Protocolos y estándares de comunicación

2.3.1. Protocolos MAC

Los protocolos MAC sirven de base para los protocolos de más alto nivel que se utilizan en las redes inalámbricas de sensores, por ejemplo los protocolos de enrutamiento, ya que estos utilizarán las funciones que se establecen en la MAC para realizar el envío y recepción de paquetes, sincronización de los sensores etc. (Serna Sanchis, 2007)

Los protocolos MAC presentan varias características, una de ellas es la flexibilidad, esto debido a que una WSN está en un entorno que varía por razones de cambios en el clima, interferencia de ondas, ingreso de nuevos nodos, etc., por lo que se deben adaptar según el cambio que se presente. Otra de las características de los protocolos MAC es la eficiencia ya que en una red inalámbrica de sensores los datos que son transmitidos en su mayoría son en tiempo real para lo cual debe ser fiable y robusta. Estos protocolos afectan directamente a la disipación de la energía, ya que son la capa más próxima al nivel físico, también determinará, en parte, el coste del sistema. Así como serán clave a la hora de especificar la latencia y el nivel de seguridad del sistema. En las redes de sensores estos protocolos determinan los canales de radio a utilizar, implementan las transmisiones y recepciones a bajo nivel, además de controlar los errores. (Serna Sanchis, 2007, pág. 11).

Como su nombre lo indica un protocolo MAC se encarga de controlar el acceso al medio, este debe evitar las interferencias, minimizando colisiones, etc. Hay protocolos MAC que son comunes para las comunicaciones inalámbricas, una de las más populares es IEEE 802.11 que se

usa en redes inalámbricas locales WLAN por su velocidad de intercambio de paquetes, sus anchos de banda que van desde los 54 hasta los 300 Mbits por segundo, etc., se debe tener en cuenta la latencia, esta debe ser reducida lo que más se pueda para así disminuir los tiempos de espera. Los dispositivos que forman una WLAN posen baterías de larga duración y son fáciles de recargar, lo contrario a un nodo sensor cuya energía es limitada. En una WLAN los dispositivos se encuentran en modo listen todo el tiempo incluso cuando se encuentran en modo reposo. Al momento se recibir una trama de comunicación, estos despiertan y se comunican con su base lo cual puede tomar mucho tiempo produciendo perdidas de paquetes, este inconveniente no se puede generar en una red de sensores inalámbricos, por lo tanto la idea básica consiste en deshabilitar el nodo y habilitarse únicamente cuando necesite transmitir datos, generando ahorros de energía en los nodos sensores.

Una red de sensores inalámbrica generalmente transmite valores de medida, esto significa que se tiene una comunicación de varios a uno, por lo que estos datos deben ser constantemente modificados, su tamaño es pequeño, es por esta razón que los intervalos para modificar los datos son altos en comparación al tiempo que se necesita en una red inalámbrica normal WLAN.

2.3.1.1. Clasificación

Los protocolos de acceso al medio se han estudiado durante muchos años y estos pueden variar según la aplicación que se esté utilizando, estos protocolos pueden ser centralizados y o distribuidos. Existe una gran variedad de protocolos que son utilizados en las redes de sensores y varios de ellos son:

- **S-MAC**

  Es el primer protocolo creado para redes de sensores inalámbricos. Este protocolo está basado en el concepto de escucha adaptable, es decir los nodos se encuentran en modo reposo

y escuchan de forma periódica si se está anunciando la generación de transmisión de datos (ahorro de energía), de esa manera la detección de portadora virtual es utilizada mientras que la radio base permanece apagada hasta que la comunicación con el nodo vecino se realiza, la sincronización de paquetes se la realiza vía broadcast.

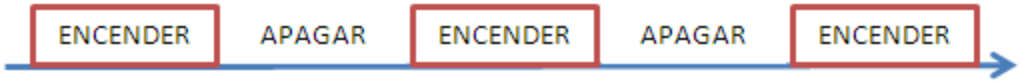


Figura 14. S - MAC

- **T-MAC**

Es un protocolo mejorado ya que el consumo de energía es más eficiente en comparación con el protocolo S-MAC, aunque los requisitos de latencia y el espacio de transmisión son generalmente fijos, la tasa de mensajes suele variar. Siempre que la carga es inferior a la que se espera, el tiempo no es óptimamente usado y la energía se desperdicia en las escuchas inactivas. Para resolver esta ineficiencia, la implementación del protocolo T-MAC reduce las escuchas ociosas y transmite todos los mensajes en ráfagas de longitud variable, y durmiendo entre ráfagas.

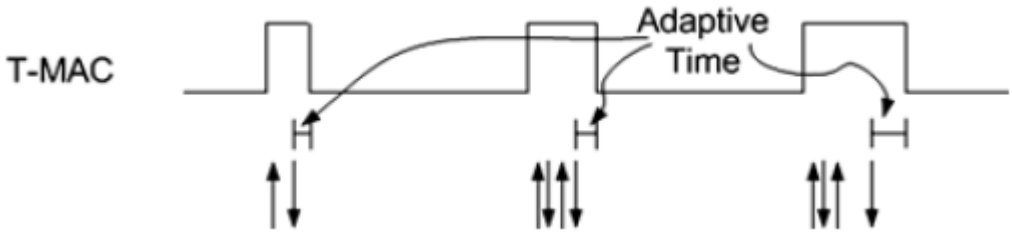


Figura 15. T - MAC

- **B-MAC**

B-MAC, es un protocolo nuevo que se encuentra en la capa MAC de una red de sensores inalámbrica, este protocolo es capaz de reducir considerablemente la escucha ociosa que se produce en una red, mediante la utilización de este protocolo cada nodo se activa periódicamente para comprobar la actividad que hay en el canal, si existe transmisión de datos permanece activo, caso contrario vuelve a su estado de reposo. (Universidad de Castilla , 2006)

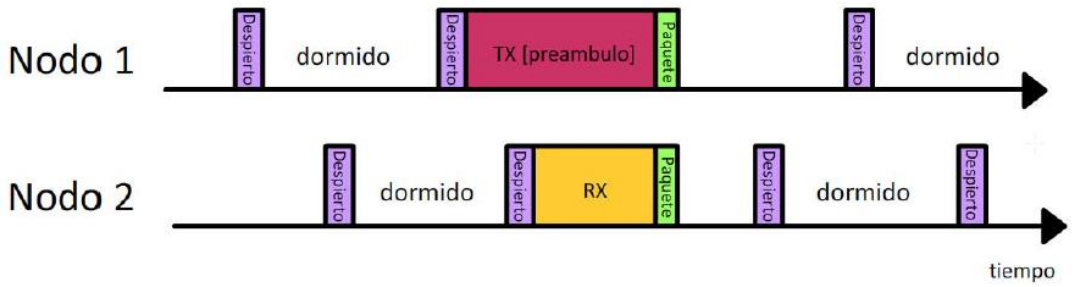


Figura 16. B - MAC

2.3.2. Protocolos de Enrutamiento

En una red inalámbrica de sensores los nodos no saben cuál es la topología actual de la red, la idea es que cuando un nuevo nodo se integra a la red y escucha un broadcast de transmisión de sus vecinos, este informa sus características, alcance y se cómo puede enrutarse con los nodos vecinos. Existen varias reglas que deben cumplir los protocolos de enrutamiento y estos son:

- Deben poseer una tabla de enrutamiento pequeña y actualizada con los cambios que se producen en la red
- Tienen que ser capaces a escoger el mejor camino de para transmitir los datos al nodo vecino o a su destino final.

Los protocolos de enrutamiento de una red de sensores inalámbricos se clasifican en tres grandes grupos que son aquellos que están basados en la estructura de la red y en criterios de encaminamiento.

2.3.2.1. Protocolos de enrutamiento basados en la estructura de la red

Existen tres tipos de protocolos que se basan netamente en la estructura de la red y estos son:

- **Enrutamiento plano**

Al enrutamiento plano también se lo conoce como encaminamiento centrado de datos, donde en una red todos los sensores se comunican entre sí ejecutando la misma función para realizar el monitoreo de su entorno y el envío de datos , este tipo de redes poseen una gran cantidad de sensores lo que dificulta la asignación de un identificador único a cada uno de ellos, motivo por el cual se realiza la centralización de los datos, es decir que la estación base envía consultas a ciertas partes de la red y de esta manera obtiene los datos que necesita.

- **Enrutamiento jerárquico**

El enrutamiento jerárquico presenta muchas ventajas relacionadas con la escalabilidad y una comunicación más eficiente. Este tipo de protocolos es utilizado para realizar un encaminamiento eficaz en términos de energía de los nodos, es decir que los nodos que poseen más energía se utilizan para comunicación y procesamiento de información, mientras que aquellos nodos que poseen poca energía serán utilizados para monitorizar el entorno. En

este tipo de enrutamiento se crean clúster o grupos los cuales tienen tareas específicas con el objetivo de mejorar la escalabilidad, tiempo de vida y eficiencia de la energía en la red.

2.3.2.2. Protocolos de enrutamiento basado en criterios de encaminamiento

- Protocolos de enrutamiento multicamino
- Encaminamiento basado en petición
- Enrutamiento basado en negociación
- Encaminamiento basado en calidad de servicio

2.3.3. Estándar IEEE 802.11

El protocolo IEEE 802.11 o WI-FI es un estándar de protocolo de comunicaciones del IEEE que define el uso de los dos niveles más bajos de la arquitectura OSI (capas física y de enlace de datos), especificando sus normas de funcionamiento en una WLAN (Wikipedia, Wikipedia). En general, los protocolos de la rama 802.x definen la tecnología de redes de área local. La familia 802.11 actualmente incluye seis técnicas de transmisión por modulación que utilizan todos los mismos protocolos. El estándar original de este protocolo data de 1997, tenía velocidades de 1 hasta 2 Mbps y trabajaba en la banda de frecuencia de 2,4 GHz. En la actualidad no se fabrican productos sobre este estándar (ITSSMT, 2011).

La capa física define la modulación de las ondas de radio y las características de señalización para la transmisión de datos mientras que la capa de enlace de datos define la interfaz entre el bus del equipo y la capa física, en particular un método de acceso parecido al utilizado en el estándar Ethernet, y las reglas para la comunicación entre las estaciones de la red. (Kioskea, 2014). En realidad, el estándar 802.11 tiene tres capas físicas que establecen modos de transmisión alternativos:

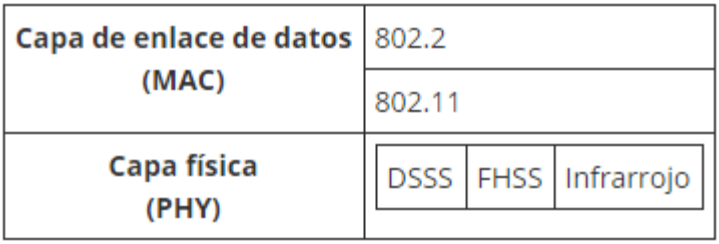


Figura 17. WiFi 802.11 (Kioskea, 2014)

El estándar 802.11 en realidad es el primer estándar y permite un ancho de banda de 1 a 2 Mbps. El estándar original se ha modificado para optimizar el ancho de banda (incluidos los estándares 802.11a, 802.11b y 802.11g, denominados estándares físicos 802.11) o para especificar componentes de mejor manera con el fin de garantizar mayor seguridad o compatibilidad (Kioskea, 2014). La tabla a continuación muestra las distintas modificaciones del estándar 802.11 y sus significados:

Nombre del estándar	Nombre	Descripción
802.11a	Wifi5	El estándar 802.11 (llamado WiFi 5) admite un ancho de banda superior (el rendimiento total máximo es de 54 Mbps aunque en la práctica es de 30 Mbps). El estándar 802.11a provee ocho canales de radio en la banda de frecuencia de 5 GHz.
802.11b	Wifi	El estándar 802.11 es el más utilizado actualmente. Ofrece un rendimiento total máximo de 11 Mbps (6 Mbps en la práctica) y tiene un alcance de hasta 300 metros en un espacio abierto. Utiliza el rango de frecuencia de 2,4 GHz con tres canales de radio disponibles.
802.11c	Combinación del 802.11 y el 802.1d	El estándar combinado 802.11c no ofrece ningún interés para el público general. Es solamente una versión modificada del estándar 802.1d que permite combinar el 802.1d con dispositivos compatibles 802.11 (en el nivel de enlace de datos).
802.11d	Internacionalización	El estándar 802.11d es un complemento del estándar 802.11 que está pensado para permitir el uso internacional de las redes 802.11 locales. Permite que distintos dispositivos intercambien información en rangos de frecuencia según lo que se permite en el país de origen del dispositivo.
802.11e	Mejora de la calidad del servicio	El estándar 802.11e está destinado a mejorar la calidad del servicio en el nivel de la *capa de enlace de datos*. El objetivo del estándar es definir los requisitos de diferentes paquetes en cuanto al ancho de banda y al retardo de transmisión para permitir mejores transmisiones de audio y video.
802.11f	Itinerancia	El 802.11f es una recomendación para proveedores de puntos de acceso que permite que los productos sean más compatibles. Utiliza el *protocolo IAPP* que le permite a un usuario itinerante cambiarse claramente de un punto de acceso a otro mientras está en movimiento sin importar qué marcas de puntos de acceso se usan en la infraestructura de la red. También se conoce a esta propiedad simplemente como *itinerancia*.
802.11g		El estándar 802.11g ofrece un ancho de banda elevado (con un rendimiento total máximo de 54 Mbps pero de 30 Mbps en la práctica) en el rango de frecuencia de 2,4 GHz. El estándar 802.11g es compatible con el estándar anterior, el 802.11b, lo que significa que los dispositivos que admiten el estándar 802.11g también pueden funcionar con el 802.11b.
802.11h		El estándar *802.11h* tiene por objeto unir el estándar 802.11 con el estándar europeo (HiperLAN 2, de ahí la *h* de 802.11h) y cumplir con las regulaciones europeas relacionadas con el uso de las frecuencias y el rendimiento energético.
802.11i		El estándar *802.11i* está destinado a mejorar la seguridad en la transferencia de datos (al administrar y distribuir claves, y al implementar el cifrado y la autenticación). Este estándar se basa en el *AES* (estándar de cifrado avanzado) y puede cifrar transmisiones que se ejecutan en las tecnologías 802.11a, 802.11b y 802.11g.

Figura 18. Estandares WiFi (Kioskea, 2014).

2.3.4. Estándar IEEE 802.15

IEEE 802 se dio origen en el año 1980, para luego dar lugar a la creación del estándar 802.15 en el año 1999, cuyos fundamentos se centran en redes PAN y HAN. El estándar 802.15 es un conjunto de protocolos especializados en redes inalámbricas como las WPAN, este protocolo se divide en 7 grandes grupos y estos son:

- WPAN / BLUETOOTH (**IEEE 802.15.1)**
- Coexistencia (**IEEE 802.15.2)**
- WPAN alta velocidad (**IEEE 802.15.3)**
- WPAN baja velocidad (**IEEE 802.15.4)**
- Redes en malla (**IEEE 802.15.5)**
- BAN (**IEEE 802.15.6)**
- WNG (**IEEE 802.15.7)**

2.3.5. Zigbee

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (Wireless personal area network, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías (SecurityMex).

2.3.6. ZigBee vs Bluetooth

ZigBee es muy similar al Bluetooth pero con algunas diferencias y ventajas para domótica:

- Una red ZigBee puede constar de un máximo de 65535 nodos distribuidos en subredes de 255 nodos, frente a los ocho máximos de una subred (Piconet) Bluetooth (Cantillo, 2010).
- Menor consumo eléctrico que el de Bluetooth. En términos exactos, ZigBee tiene un consumo de 30 mA transmitiendo y de 3 μA en reposo, frente a los 40 mA transmitiendo y 0,2 mA en reposo que tiene el Bluetooth. Este menor consumo se debe a que el sistema ZigBee se queda la mayor parte del tiempo dormido, mientras que en una comunicación Bluetooth esto no se puede dar, y siempre se está transmitiendo y/o recibiendo (Cantillo, 2010).
- Debido a las velocidades de cada uno, uno es más apropiado que el otro para ciertas cosas. Por ejemplo, mientras que el Bluetooth se usa para aplicaciones como los celulares y la informática casera, la velocidad del ZigBee se hace insuficiente para estas tareas, desviándolo a usos tales como la Domótica, los productos dependientes de la batería, los sensores médicos, y en artículos de juguetería, en los cuales la transferencia de datos es menor (Cantillo, 2010).

CAPITULO III

DISEÑO Y APLICACIÓN DEL ALGORITMO DE KRUSKAL

3.1. Implementación del algoritmo de kruskal

Para la aplicación del algoritmo se debe tomar en cuenta dos aspectos en la creación de una red de sensores inalámbricos, siendo estos la distancia entre los nodos sensores y sus niveles de energía en sus baterías. Para relacionar los niveles de batería y distancia de los nodos, se va a considerar que los pesos de los enlaces se los calcula tomando en cuenta dos variables (carga de batería y distancia), mientras más grande sea el valor de la batería (es decir este mas cargada), implica que este enlace será mejor que un nodo que tiene poca batería, además si la distancia entre los nodos es pequeña, el enlace será superior comparado con un enlace cuya distancia es más grande. Los datos del nivel de la batería así como la longitud de los enlaces son datos de entrada, con ellos se calcula el peso de cada enlace y de esta manera se pone a ejecutar el algoritmo. Esta es una herramienta que nos servirá para la creación de una red de sensores previa su instalación, para así poder realizar las pruebas necesarias a sus componentes en diferentes ambientes con el fin de ofrecer un trabajo óptimo de bajo costo de instalación.

3.2. Diseño del aplicativo

El aplicativo es un software para ser instalado en las estaciones de trabajo para ser utilizado a nivel de escritorio, que puede ser instalado en varios sistemas operativos sean estos Windows o Linux.

Esta diseñado de tal forma que puede ser utilizado por todos, ya que posee una interfaz intuitiva que facilita el correcto funcionamiento del aplicativo.

3.2.1. Seudocódigo del algoritmo de Kruskal a ser implementado

Para realizar la implementación del algoritmo se realizó el siguiente seudocódigo

1. Método Kruskal(Grafo):
2. Inicializamos como vacío
3. Inicializamos estructura unión-find
4. Ordenamos las aristas del grafo por peso de menor a mayor.
5. Para cada arista e que une los vértices u y v
6. Si u y v no están en la misma componente
7. Agregamos la arista e
8. Realizamos la unión de las componentes de u y v

3.2.2. Diagrama de Flujo del Algoritmo de Kruskal

A continuación se visualiza el diagrama de flujo a ser desarrollador para la implementación del Algoritmo de Kruskal

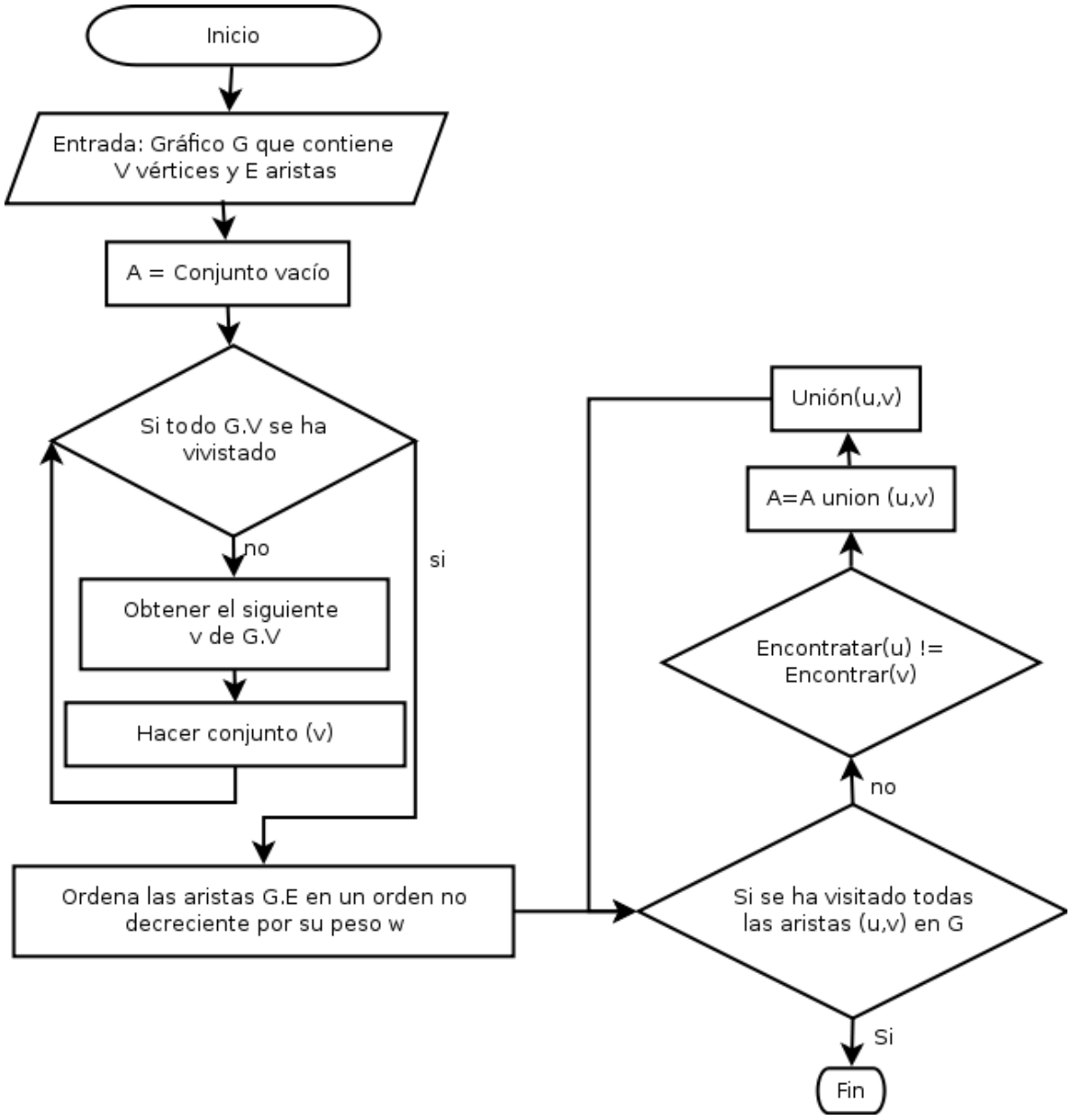


Figura 17. Diagrama de flujo Algoritmo de Kruskal

3.2.3. Requerimientos para el desarrollo

Nombre	Descripción
Visual Studio 2010	Es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C# y Visual C++ utilizan todos el mismo entorno de desarrollo integrado (IDE), que habilita el uso compartido de herramientas y hace más sencilla la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes utilizan las funciones de .NET Framework, las cuales ofrecen acceso a tecnologías clave para simplificar el desarrollo de aplicaciones web ASP y Servicios Web XML.
Service Pack 1	Es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C# y Visual C++ utilizan todos el mismo entorno de desarrollo integrado (IDE), que habilita el uso compartido de herramientas y hace más sencilla la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes utilizan las funciones de .NET Framework, las cuales ofrecen acceso a tecnologías clave para simplificar el desarrollo de aplicaciones web ASP y Servicios Web XML.
RAM	Al menos 1GB de RAM para el desarrollo

3.2.4. Ambiente de desarrollo

El aplicativo se lo construirá con un lenguaje de programación de alto nivel, el cual formara parte de un sistema de software que además está compuesto de una interfaz gráfica donde se podrá ver la generación de la red de sensores así como sus respectivos pesos.

El algoritmo de Kruskal ha sido implementado en este caso en lenguaje de programación C#. Net que permitirá su implementación a futuro en los Gateway para redes sensores inalámbricos. El algoritmo será implementado de la siguiente manera:

- Kruskal.cs (interfaz gráfica)

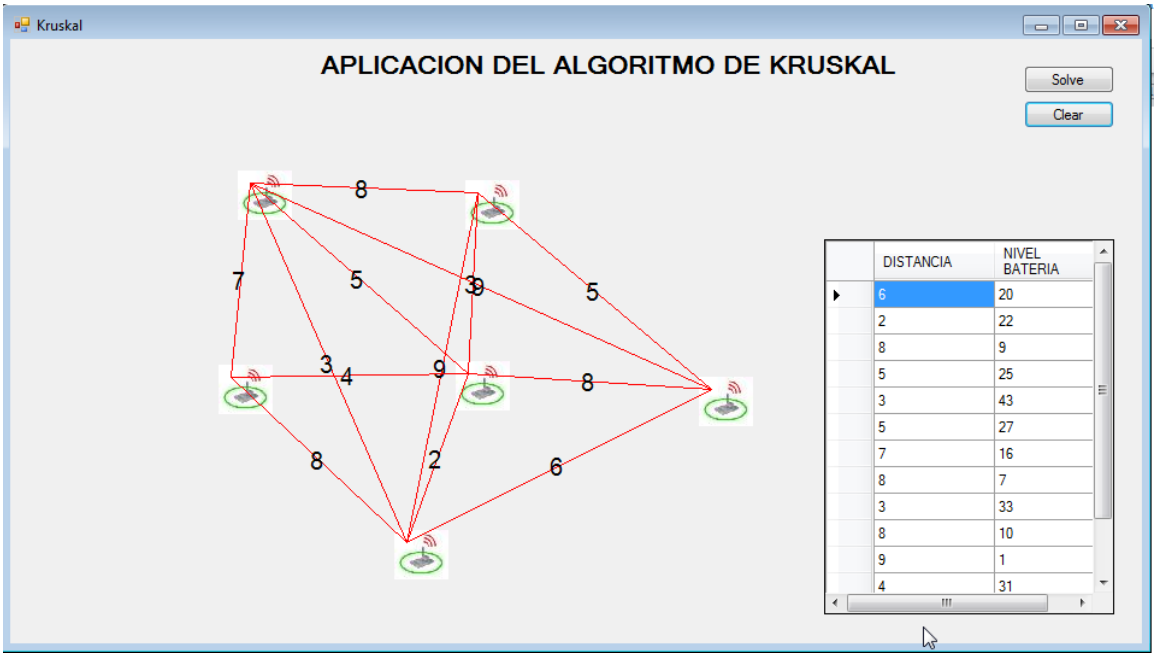


Figura 18. Interfaz Gráfica Algoritmo de Kruskal

La interfaz gráfica nos sirve para visualizar todos los nodos sensores que voy a utilizar en mi red así como todos sus enlaces con sus respectivos pesos que están tomando en cuenta los valores de las baterías de los sensores. El aplicativo está constituido por tres secciones, la primera de ellas es donde colocaremos nuestros nodos de sensores según la distribución que se

desee y en la siguiente los botones con los cuales aplicaremos el algoritmo de Kruskal y otro para limpiar nuestra área de trabajo y una última sección donde observaremos los datos ingresados inicialmente para la aplicación del algoritmo.

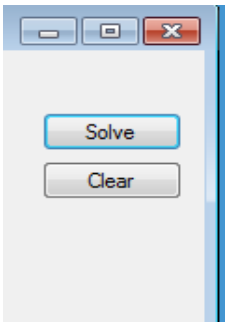


Figura 20. Botones de aplicación del algoritmo

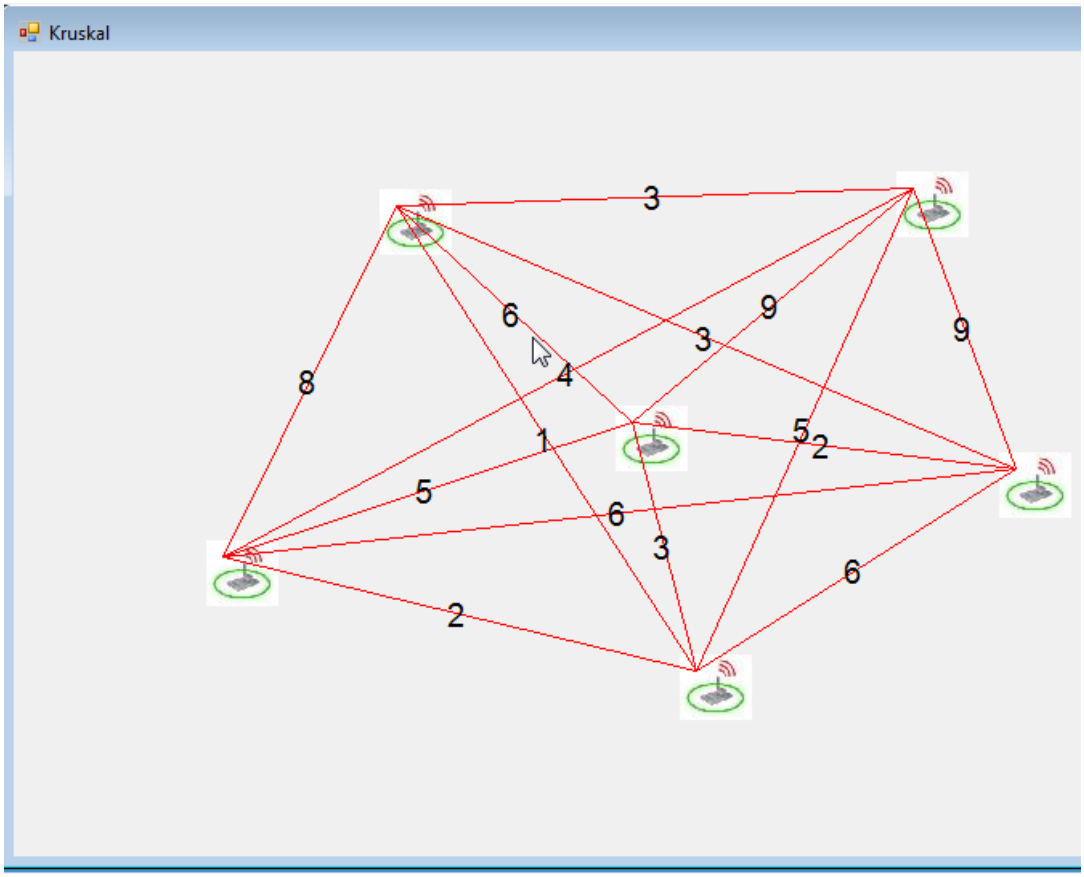


Figura 21. Zona de colocación de nodos y aristas

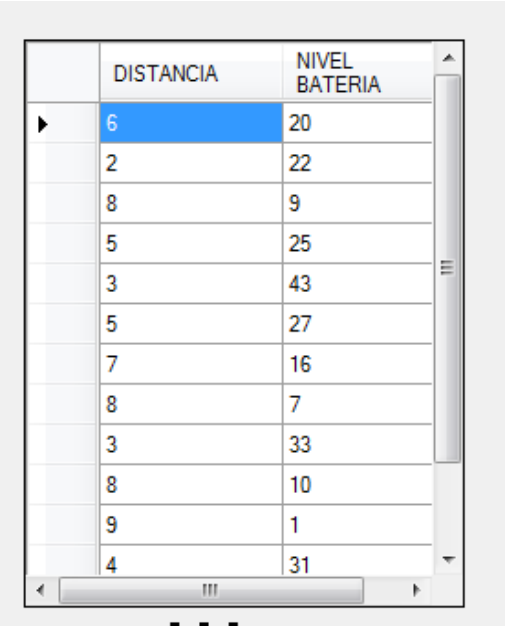


Figura 22. Datos iniciales del algoritmo

El aplicativo posee las siguientes clases:

- Link.cs

En esta clase se realiza la programación para la colocación de los pesos de las aristas que unen todos los nodos, los valores que se generan son aleatorios y el nivel de la batería se las obtiene a partir del nivel de carga del Gateway o pc en el que se encuentra instalado el aplicativo.

- Node.cs

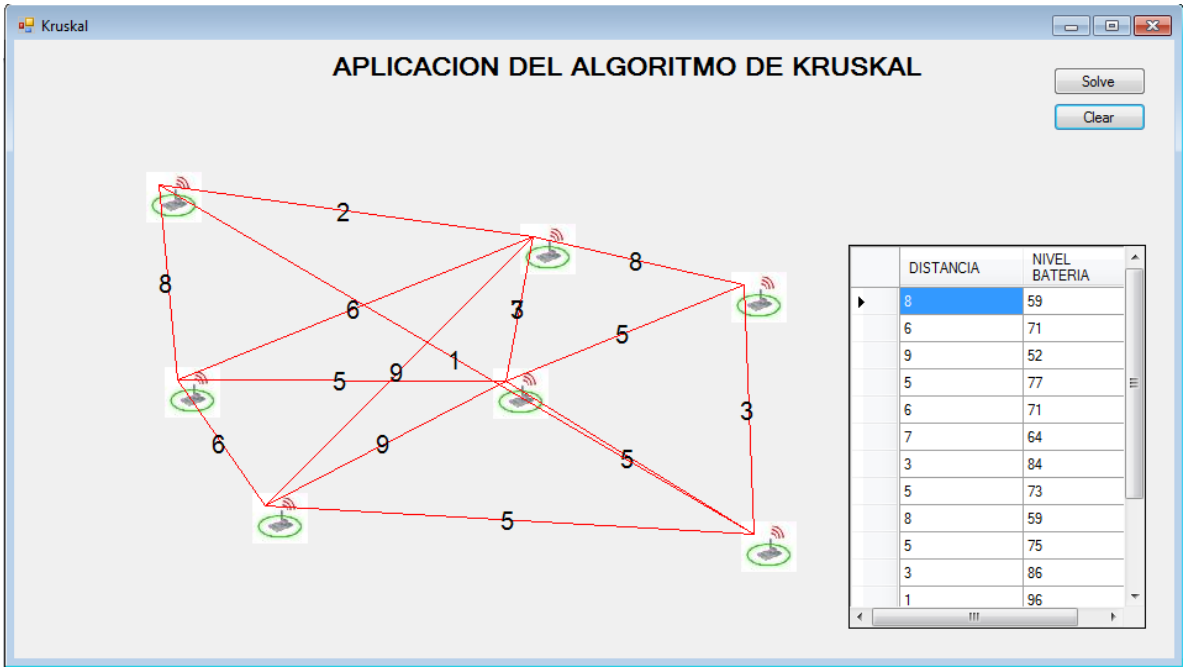
La clase node es donde se generan las coordenadas donde son colocados los nodos sensores, estos valores de posición se almacenan para que al momento de aplicar el algoritmo de Kruskal solo se visualicen los nodos que se cumplen las condiciones según el algoritmo

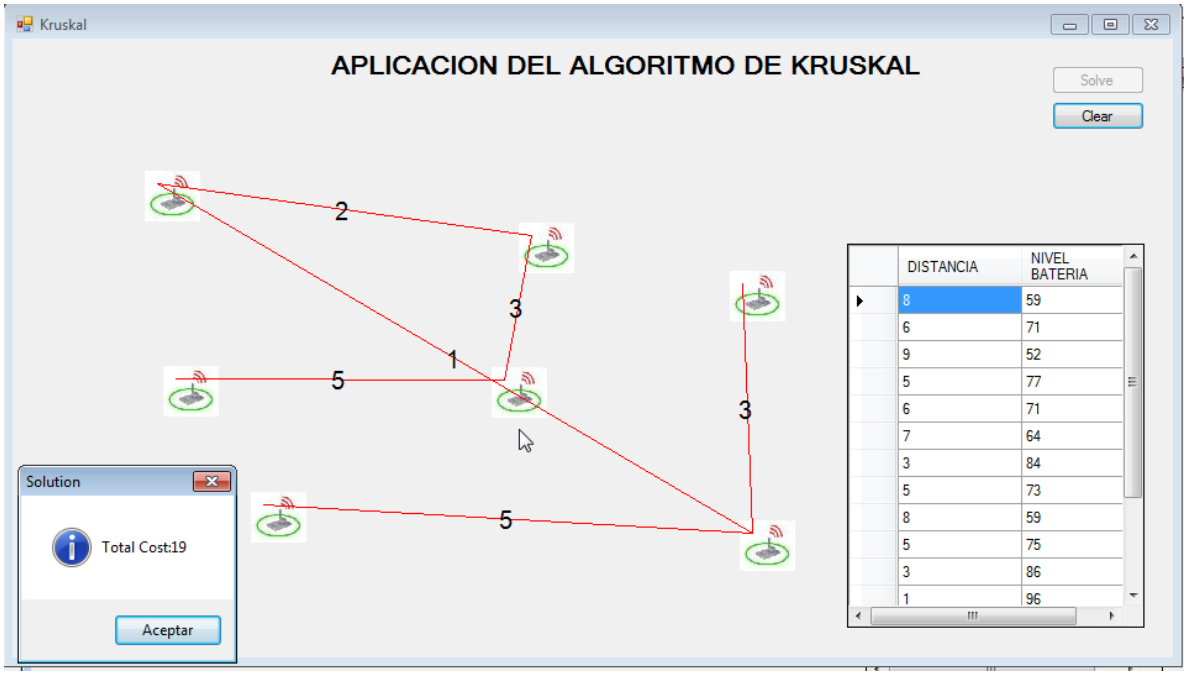
- Program.cs

En esta clase se centra la programación del algoritmo de Kruskal para ser aplicado a la red de sensores diseñada en el aplicativo

3.3. Pruebas y resultados del algoritmo

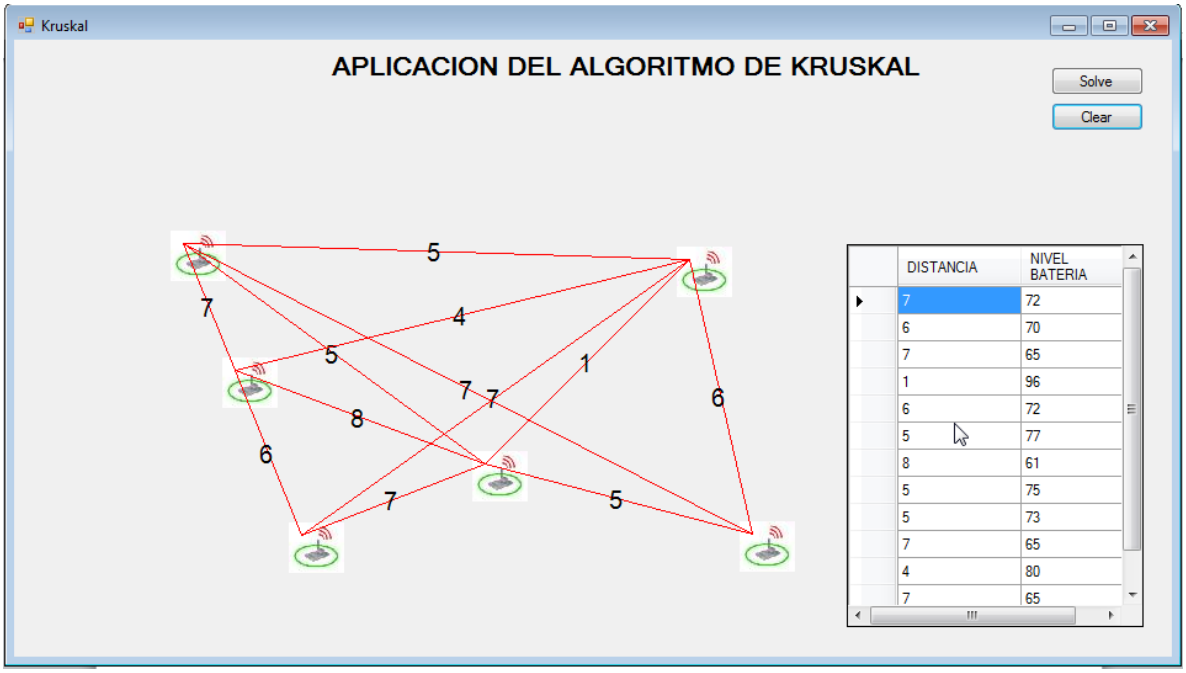
3.3.1. Prueba sin tomar en cuenta distancias y nivel de batería



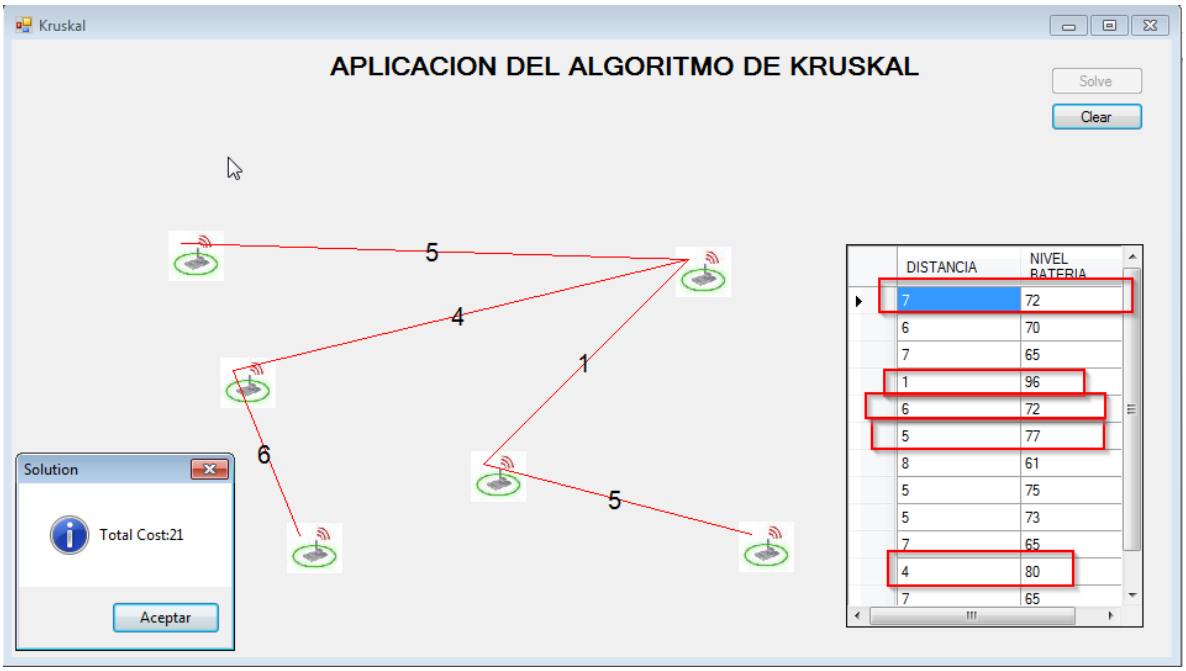


Luego de realizar la primera prueba con el sistema se observa que el algoritmo de kruskal toma todos los valores de la red y luego de realizar la comparación entre nodos y sus pesos, para de esta manera escoger el mejor camino y así obtener el costo total del camino más corto por donde los datos serán transmitidos. En este caso el valor de las baterías no es tomado en cuenta por lo que se correrá el riesgo de perder uno de los nodos hasta que la batería pueda ser reemplazada.

3.3.2. Pruebas tomando en consideración el valor de carga de las baterías de los nodos



Inicialmente la red es construida en el sistema, como se observa en los datos iniciales el primer nodo presenta un nivel de batería de 72%, el algoritmo se encuentra en esta ocasión programado para que tome los nodos con mayor nivel de carga en comparación a la del primer nodo y de esta manera a más de aplicar el algoritmo obtenemos el camino más eficiente y reduciendo el gasto innecesario de batería en nodos cuyos niveles de carga son menores al nodo inicial



Se puede observar que se toma los nodos cuyos niveles de batería son iguales o superiores al nodo inicial.

3.4. Procedimiento para el cálculo de tiempo de actualización de datos en el Gateway

El procedimiento que se propone para el cálculo de tiempo para la actualización de datos en el Gateway se tomara como punto inicial si en toda la topología existe al menos un nodo con un porcentaje menor al 20% entonces el Gateway hará una nueva petición de sus valores para obtener un nuevo árbol para una nueva transmisión de datos.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

Como resultado del presente proyecto se obtienen las siguientes conclusiones

- Mediante la aplicación del programa se facilita la obtención de la mejor ruta con la utilización del algoritmo de kruskal
- Al aplicar el criterio de usabilidad de carga en la batería junto con el algoritmo y al observar los resultados obtenidos se reducirán considerablemente los costos por reemplazos de las baterías
- Con la utilización de este programa se mejoran los tiempos de creación e implementación de redes de sensores inalámbricas
- El sistema podrá ser utilizado en redes SDN por su fácil utilización por ejemplo en redes compuestos por nodos GPS

RECOMENDACIONES

- Se recomienda la utilización de esta tesis para posteriores investigaciones sobre el tema

Bibliografía

Alejo, J. F. (2013). *Algoritmo de Kruskal.* Monografia, Universidad Nacional del Altiplano .

Alonso Revenga, J. M. (2008). *Flujo en Redes y Gestión de Proyectos. Teoría y Ejercicios Resueltos.*

Cantillo, S. R. (Septiembre de 2010). *DESARROLLO DE APLICACIONES BASADAS EN WSN*. Obtenido de https://riunet.upv.es/bitstream/handle/10251/8592/PFC%20-%20DESARROLLO%20DE%20APLICACIONES%20BASADAS%20EN%20WSN.pdf

Capella Hernandez, J. V. (2010). *Redes inalámbricas de sensores:una nueva arquitectura eficiente y robusta basada en jerarquía dinámica de grupos.* Valencia: Editorial Universitat Politècnica de València.

Cárdenas, J. R., Perez, F., & Barrera, E. (25 de Agosto de 2010). *slideshare*. Obtenido de http://es.slideshare.net/fher969/algoritmos-de-kruskal-y-prim

Figueroa, J. G. (2012). *Algorithms and More*. Obtenido de https://jariasf.wordpress.com/2012/04/19/arbol-de-expansion-minima-algoritmo-de-kruskal/

Fundación Wikimedia, I. (s.f.). *Wikimedia.* Obtenido de https://es.wikipedia.org/wiki/%C3%81rbol_de_expansi%C3%B3n

Garbarino, J. (2011). *Protocolos para redes inalambricas de sensores.* Buenos Aires: Universidad de Buenos Aires.

ITSSMT. (22 de 05 de 2011). *Familia de Protocolos IEEE*. Obtenido de https://drago1214.files.wordpress.com/2011/03/familia-de-protocolos-ieee-8021.pptx

Kioskea. (Junio de 2014). *http://es.ccm.net/contents/789-introduccion-a-wi-fi-802-11-o-wifi*.

REGOLI, C. S. (10 de Junio de 2013). DISEÑO DE UN METODO DE ENRUTAMIENTO ACTIVO PARA REDES INALAMBRICAS DE SENSORES. SEVILLA, ESPAÑA.

Sanchis, J. S. (2007). REDES DE SENSORES INALAMBRICOS. 3. Obtenido de http://www.uv.es/~montanan/ampliacion/trabajos/Redes%20de%20Sensores.pdf

SecurityMex. (s.f.). *Brochure 2 Aplicaciones*. Obtenido de http://www.securitymex.com/brochure/aplicacion/aplicacion.pdf

Serna Sanchis, J. (10 de 01 de 2007). *REDES DE SENSORES INALAMBRICAS*. Obtenido de http://www.uv.es/~montanan/ampliacion/trabajos/Redes%20de%20Sensores.pdf

Universidad de Castilla . (27 de 09 de 2006). B-MAC Y OTROS PROTOCOLOS. *B-MAC Y OTROS PROTOCOLOS*. Obtenido de https://www.dsi.uclm.es/descargas/technicalreports/DIAB-06-09-2/TRBMAC.pdf

Villalobos, A. R. (2003). *Grafos - software para la construcción, edición y análisis de grafos*. Obtenido de http://arodrigu.webs.upv.es/grafos/doku.php?id=inicio

Wikipedia. (s.f.). *Arbol de Expansion*. Obtenido de https://es.wikipedia.org/wiki/%C3%81rbol_de_expansi%C3%B3n

Wikipedia. (s.f.). *Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/IEEE_802.11

ANEXO 1

CODIGO FUENTE PROGRAMADO

Link.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Kruskal
{
    class Link : IComparable
    {
        #region Members
        Node v1, v2;
        int nCost;
        System.Drawing.Point pStringPosition;
        private int p;
        #endregion

        #region Properties
        public Node V1
        {
            get
            {
                return v1;
            }
        }
        public Node V2
        {
            get
            {
                return v2;
            }
        }
        public int Cost
        {
            get
            {
                return nCost;
            }
        }
        public System.Drawing.Point StringPosition
        {
            get
            {
                return pStringPosition;
            }
        }
        #endregion

        public Link(Node v1, Node v2, int nCost, System.Drawing.Point pStringPosition)
        {
            this.v1 = v1;
```

```
            this.v2 = v2;
            this.nCost = nCost;
            this.pStringPosition = pStringPosition;
        }

        public Link(int p)
        {
            // TODO: Complete member initialization
            this.p = p;
        }

        #region IComparable Members

        public int CompareTo(object obj)
        {
            Link e = (Link)obj;
            return this.nCost.CompareTo(e.nCost);
        }

        #endregion

        internal static void QuickSort(List<Link> m_lstEdgesInitial, int nLeft, int
nRight)
        {
            int i, j, x;
            i = nLeft; j = nRight;
            x = m_lstEdgesInitial[(nLeft + nRight) / 2].Cost;

            do
            {
                while ((m_lstEdgesInitial[i].Cost < x) && (i < nRight)) i++;
                while ((x < m_lstEdgesInitial[j].Cost) && (j > nLeft)) j--;

                if (i <= j)
                {
                    Link y = m_lstEdgesInitial[i];
                    m_lstEdgesInitial[i] = m_lstEdgesInitial[j];
                    m_lstEdgesInitial[j] = y;
                    i++; j--;
                }
            } while (i <= j);

            if (nLeft < j) QuickSort(m_lstEdgesInitial, nLeft, j);
            if (i < nRight) QuickSort(m_lstEdgesInitial, i, nRight);
        }
    }
}
```

Node.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Kruskal
{
```

```
    class Node
    {
        #region Members
        int nName;
        public int nRank;
        public Node vRoot;
        public System.Drawing.Point pPosition;
        #endregion

        #region Properties
        public int Name
        {
            get
            {
                return nName;
            }
        }
        #endregion

        public Node(int nName, System.Drawing.Point pPosition)
        {
            this.nName = nName;
            nRank = 0;
            this.vRoot = this;
            this.pPosition = pPosition;
        }

        #region Methods
        internal Node GetRoot()
        {
            if (this.vRoot != this)// am I my own parent ? (am i the root ?)
            {
                this.vRoot = this.vRoot.GetRoot();// No? then get my parent
            }
            return this.vRoot;
        }

        internal static void Join(Node vRoot1, Node vRoot2)
        {

            if (vRoot2.nRank < vRoot1.nRank)//is the rank of Root2 less than that of
Root1 ?
            {
                vRoot2.vRoot = vRoot1;//yes! then Root1 is the parent of Root2 (since it
has the higher rank)
            }
            else //rank of Root2 is greater than or equal to that of Root1
            {
                vRoot1.vRoot = vRoot2;//make Root2 the parent
                if (vRoot1.nRank == vRoot2.nRank)//both ranks are equal ?
                {
                    vRoot1.nRank++;//increment one of them, we need to reach a single
root for the whole tree
                }
            }
        }
        #endregion
    }
```

```
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Kruskal
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Kruskal());
        }
    }
}
```

Kruskal.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace Kruskal
{
    public partial class Kruskal : Form
    {
        public Kruskal()
        {
            InitializeComponent();
            Reset();
        }

        #region Member Variables
        const int m_nRadius = 20;
        const int m_nHalfRadius = (m_nRadius / 2);
        public int carga;
        public int distancia;

        Color m_colVertex = Color.Aqua;
        Color m_colEdge = Color.Red;
```

```
        List<Node> m_lstVertices;
        List<Link> m_lstEdgesInitial, m_lstEdgesFinal;

        Node m_vFirstVertex, m_vSecondVertex;

        bool m_bDrawEdge, m_bSolved;

        #endregion

        #region Events

        private void panel1_MouseClick(object sender, MouseEventArgs e)
        {
            Point pClicked = new Point(e.X - m_nHalfRadius, e.Y - m_nHalfRadius);
            if (Control.ModifierKeys == Keys.Control)//if Ctrl is pressed
            {
                if (!m_bDrawEdge)
                {
                    m_vFirstVertex = GetSelectedVertex(pClicked);
                    m_bDrawEdge = true;
                }
                else
                {
                    m_vSecondVertex = GetSelectedVertex(pClicked);
                    m_bDrawEdge = false;
                    if (m_vFirstVertex != null && m_vSecondVertex != null &&
m_vFirstVertex.Name != m_vSecondVertex.Name)
                    {
                        Cost formCost = new Cost();
                        formCost.ShowDialog();
                        carga = formCost.charges;
                        distancia=formCost.m_nCost;

                        Point pStringPoint = GetStringPoint(m_vFirstVertex.pPosition,
m_vSecondVertex.pPosition);
                        m_lstEdgesInitial.Add(new Link(m_vFirstVertex, m_vSecondVertex,
formCost.m_nCost, pStringPoint));
                        if (Convert.ToString(carga) != string.Empty &&
Convert.ToString(distancia) != string.Empty)
                        {
                            dataGridView1.Rows.Add(distancia, carga);
                        }
                        panel1.Invalidate();
                    }
                }
            }
            else
            {
                m_lstVertices.Add(new Node(m_lstVertices.Count, pClicked));
                panel1.Invalidate();
            }


        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {
```

```
            Graphics g = e.Graphics;
            DrawVertices(g);
            DrawEdges(g);
            g.Dispose();
        }

        // Find Minimal spanning tree using Kruskal
        private void Solve_Click(object sender, EventArgs e)
        {
            if (m_lstVertices.Count > 2)
            {
                if (m_lstEdgesInitial.Count < m_lstVertices.Count - 1)
                {
                    MessageBox.Show("Missing Edges", "Alert", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
                }
                else
                {
                    Solve.Enabled = false;
                    int nTotalCost = 0;
                    m_lstEdgesFinal = SolveGraph(ref nTotalCost);
                    m_bSolved = true;
                    panel1.Invalidate();
                    MessageBox.Show("Total Cost:" + nTotalCost.ToString(), "Solution",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
        }

        // Clear the drawing area
        private void Clear_Click(object sender, EventArgs e)
        {
            DialogResult dr = MessageBox.Show("Clear form ?", "Alert",
MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);
            if (dr == DialogResult.Yes)
            {
                Solve.Enabled = true;
                Graphics g = panel1.CreateGraphics();
                g.Clear(panel1.BackColor);
                Reset();
                label1.Text = "";
                label2.Text = "";

            }


            this.dataGridView1.Rows.Clear();
        }

        #endregion

        #region Methods

        #region Drawing

        private void DrawEdges(Graphics g)
        {
            Pen P = new Pen(m_colEdge);
```

```
            List<Link> lstEdges = m_bSolved ? m_lstEdgesFinal : m_lstEdgesInitial;
            foreach (Link e in lstEdges)
            {
                Point pV1 = new Point(e.V1.pPosition.X + m_nHalfRadius, e.V1.pPosition.Y
+ m_nHalfRadius);
                Point pV2 = new Point(e.V2.pPosition.X + m_nHalfRadius, e.V2.pPosition.Y
+ m_nHalfRadius);

                //Point arr = new Point[pV1, pV2];

                g.DrawLine(P, pV1, pV2);
                DrawString(e.Cost.ToString(), e.StringPosition, g);
            }
        }

        private void DrawString(string strText, Point pDrawPoint, Graphics g)
        {
            Font drawFont = new Font("Arial", 15);
            SolidBrush drawBrush = new SolidBrush(Color.Black);
            g.DrawString(strText, drawFont, drawBrush, pDrawPoint);
        }

        private void DrawVertices(Graphics g)
        {
            Pen P = new Pen(m_colVertex);
            Brush B = new SolidBrush(m_colVertex);
            foreach (Node v in m_lstVertices)
            {
                //g.DrawEllipse(P, v.pPosition.X, v.pPosition.Y, m_nRadius, m_nRadius);
                //g.FillEllipse(B, v.pPosition.X, v.pPosition.Y, m_nRadius, m_nRadius);
                //DrawString(v.Name.ToString(), v.pPosition, g);

                Image imagen =
Image.FromFile("C:/Diana/tesis/prototipo/kruskal2/Kruskal/Imagenes/nodo.jpg");
                g.DrawImage(imagen, v.pPosition.X, v.pPosition.Y);
            }


        }

        private Node GetSelectedVertex(Point pClicked)
        {
            Node vReturn = null;
            double dDistance;
            foreach (Node v in m_lstVertices)
            {
                dDistance = GetDistance(v.pPosition, pClicked);
                if (dDistance <= m_nRadius)
                {
                    vReturn = v;
                    break;
                }
            }
            return vReturn;
        }

        private double GetDistance(Point pStart, Point pFinish)
        {
```

```
            return Math.Sqrt(Math.Pow(pStart.X - pFinish.X, 2) + Math.Pow(pStart.Y -
pFinish.Y, 2));
        }

        private Point GetStringPoint(Point pStart, Point pFinish)
        {
            int X = (pStart.X + pFinish.X) / 2;
            int Y = (pStart.Y + pFinish.Y) / 2;
            return new Point(X, Y);
        }
        #endregion

        private void Reset()
        {
            m_lstVertices = new List<Node>();
            m_lstEdgesInitial = new List<Link>();
            m_bSolved = false;
            m_vFirstVertex = m_vSecondVertex = null;
        }

        private List<Link> SolveGraph(ref int nTotalCost)
        {
            Link.QuickSort(m_lstEdgesInitial, 0, m_lstEdgesInitial.Count - 1);
            List<Link> lstEdgesRetun = new List<Link>(m_lstEdgesInitial.Count);
            foreach (Link ed in m_lstEdgesInitial)
            {
                Node vRoot1, vRoot2;
                vRoot1 = ed.V1.GetRoot();
                vRoot2 = ed.V2.GetRoot();
                if (vRoot1.Name != vRoot2.Name)
                {
                    nTotalCost += ed.Cost;
                    Node.Join(vRoot1, vRoot2);
                    lstEdgesRetun.Add(new Link(ed.V1, ed.V2, ed.Cost,
ed.StringPosition));
                }
            }
            return lstEdgesRetun;
        }

        #endregion

        private void button1_Click(object sender, EventArgs e)
        {
            Random r = new Random();


            for (int i = 0; i <=10; i++)
            {
                if (i <= 10)
                {

                    int x = r.Next(30, 400);
                    int y = r.Next(40, 400);

                    label1.Text = Convert.ToString(x);
                    label2.Text = Convert.ToString(y);
```

```
                    Point pClicked = new Point(x - m_nHalfRadius, y - m_nHalfRadius + 3);
                    m_lstVertices.Add(new Node(m_lstVertices.Count, pClicked));

                            if (!m_bDrawEdge)
                            {
                                m_vFirstVertex = GetSelectedVertex(pClicked);

                            }
                            else
                            {

                                m_vSecondVertex = GetSelectedVertex(pClicked);

                                if (m_vFirstVertex != null && m_vSecondVertex != null &&
m_vFirstVertex.Name != m_vSecondVertex.Name)
                                {
                                    Cost formCost = new Cost();
                                    formCost.ShowDialog();

                                    Point pStringPoint =
GetStringPoint(m_vFirstVertex.pPosition, m_vSecondVertex.pPosition);
                                    m_lstEdgesInitial.Add(new Link(m_vFirstVertex,
m_vSecondVertex, formCost.m_nCost, pStringPoint));
                                    m_bDrawEdge = false;
                                    panel1.Invalidate();
                                }
                            }
                            m_bDrawEdge = true;

                        }



                    }
                m_bDrawEdge = false;
                    }

        private void btnPrueba2_Click(object sender, EventArgs e)
        {
            pruebas pru = new pruebas();
            pru.ShowDialog();

        }
      }
    }
```

Cost.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace Kruskal
{
    public partial class Cost : Form
    {
        public int m_nCost;
        public int charges;

        public Cost()
        {
            InitializeComponent();
        }

        private void OK_Click(object sender, EventArgs e)
        {
            if (textBox.Text == string.Empty)
                errorProvider1.SetError(textBox, "please enter value");
            else
            {
                Random r1 = new Random();

                m_nCost = r1.Next(1, 10);
                this.Close();
            }

        }

        public void Cost_Load(object sender, EventArgs e)
        {
            Random r1 = new Random();
            m_nCost = r1.Next(1, 10);
            label1.Text = Convert.ToString( m_nCost);

            //bateria
            PowerStatus charge = SystemInformation.PowerStatus;
            int bt = (int)(charge.BatteryLifePercent * 100);

            Random r2 = new Random();
            charges = bt - r2.Next(1,50);
            //if (charges <= 100)
                label2.Text = Convert.ToString(charges);
            //else if (charges == 0)
            //    MessageBox.Show("no tiene bateria ");
            //else
            //    MessageBox.Show("ERROR...!!!!!");
            this.Close();



        }


    }
}
```