Universidad Nacional de San Agustín

Facultad de Ingeniería de Producción y Servicios

Escuela Profesional de Ciencia de la Computación

TRABAJO FINAL:

EL VIAJERO

Curso: Algoritmos y Estructuras de Datos

Estudiantes:

- HUAYLLASCO CARLOS, Edward Luis
- ROMERO CHACÓN, Solange Aracely

Arequipa - Perú

2020

EL VIAJANTE

INTRODUCCIÓN

El problema del vendedor viajero, problema del vendedor ambulante, problema del agente viajero o problema del viajante (TSP por sus siglas en inglés (Travelling Salesman Problem)), responde a la siguiente pregunta: dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen? Este es un problema NP-Hard dentro en la optimización combinatoria, muy importante en investigación operativa y en ciencias de la computación.

El problema del viajante, que es el que tratamos de resolver esta vez, tiene una pequeña variación y es que en este caso el viajante puede pasar por todos los nodos más de una vez, puesto que no todas las ciudades están conectadas entre sí.

MÉTODO Y ALGORITMOS PARA LA RESOLUCIÓN

Algoritmo de Christofides

El algoritmo de Christofides es un algoritmo en donde los pesos de las aristas del grafo satisfacen la desigualdad triangular. Fue desarrollado en 1976 por Nicos Christofides, profesor del Imperial College London.

ALGORITMO DE CHRISTOFIDES

PASO 1

Encontrar un árbol de expansión mínima de T a G (MST)

PASO 2

Sea O el conjunto de vértices con grado impar en T. Por el lema del apretón de manos, O tiene un número par de vértices.

PASO 3

Encuentra un peso mínimo que coincida perfectamente con M en el subgrafo inducido dado por los vértices de O.

PASO 4

Combina los bordes de M y T para formar un multigráfico conectado H en el que cada vértice tiene un grado uniforme.

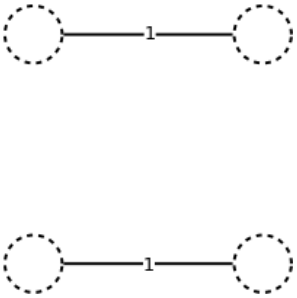
PASO 5

Formar un circuito euleriano en H.

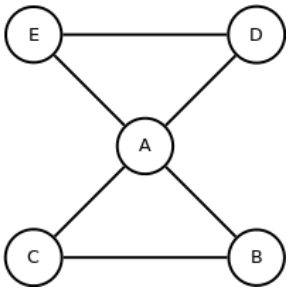
PASO 6

Haga el circuito encontrado en el paso anterior en un circuito Hamiltoniano saltándose los vértices repetidos (shortcutting).

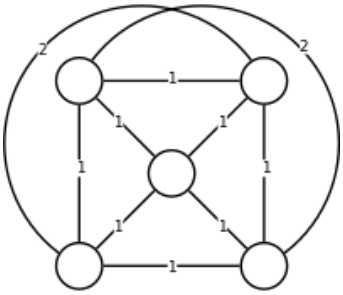
Ejemplo



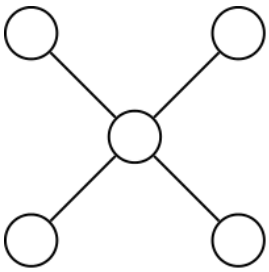
Dado una gráfica completa cuyos pesos de aristas obedecen a la desigualdad del triángulo.



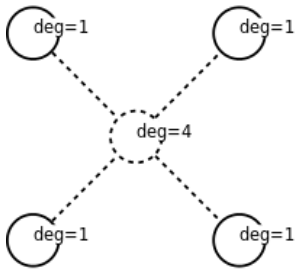
Calcular el árbol de expansión mínimo T.



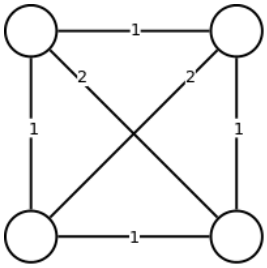
Calcular el conjunto de vértices O con grado impar en T.



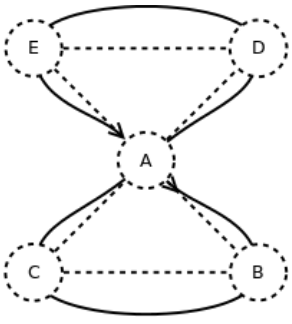
Formar el subgrafo de G usando solo los vértices de O.



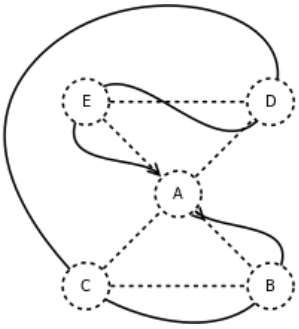
Calcular el conjunto de vértices O con grado impar en T.



Encontrar un peso mínimo que coincida perfectamente con M en el subgrafo.



Calcular el recorrido de Euler.



Eliminar los vértices repetidos, dando la salida del algoritmo

a. Árbol de Expansión Mínima (PRIM)

Al igual que el algoritmo de Kruskal, el algoritmo de Prim también es un algoritmo codicioso. Comienza con un árbol de expansión vacío. La idea es mantener dos conjuntos de vértices. El primer conjunto contiene los vértices ya incluidos en el MST, el otro conjunto contiene los vértices aún no incluidos. En cada paso, considera todos los bordes que conectan los dos conjuntos y selecciona el borde de peso mínimo de estos bordes. Después de seleccionar el borde, mueve el otro extremo del borde al conjunto que contiene MST.

ALGORITMO DE PRIM

PASO 1

Crear un conjunto mstSet que lleve la cuenta de los vértices ya incluidos en MST.

PASO 2

Asignar un valor clave a todos los vértices del gráfico de entrada. Inicializar todos los valores clave como INFINITO. Asignar el valor de la clave como 0 para el primer vértice, de modo que sea elegido primero.

PASO 3

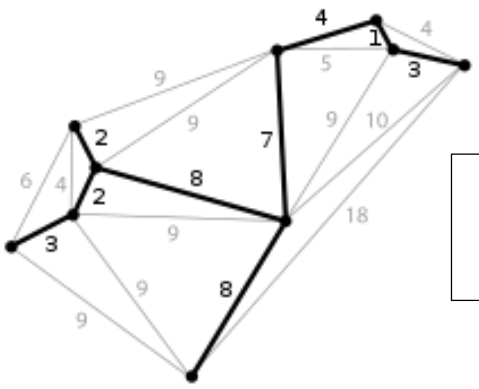
Aunque mstSet no incluye todos los vértices

a) Escoger un vértice u que no esté en mstSet y que tenga un valor clave mínimo.

b) Incluir la u en mstSet.

c) Actualiza el valor clave de todos los vértices adyacentes de u. Para actualizar los valores clave, itera a través de todos los vértices adyacentes. Para cada vértice adyacente v, si el peso del borde u-v es menor que el valor clave anterior de v, actualizar el valor clave como peso de u-v

Ejemplo



El algoritmo de Prim permite encontrar el árbol de expansión mínima de un grafo.

b. Circuito Euleriano

Un ciclo euleriano es aquel camino que recorre todas las aristas de un grafo pasando una y sólo una vez por cada arista del grafo, siendo condición necesaria que regrese al vértice inicial de salida. Una definición más formal lo define como: "*aquel ciclo que contiene todas las aristas de un grafo solamente una vez*".

ALGORITMO PARA UN CIRCUITO EULERIANO

PASO 1

Verificar que es conexo con todos los vértices pares.

PASO 2

Seleccionar un vértice arbitrario.

PASO 3

Seleccionar una arista a partir del vértice actual que no sea puente(es decir que no desconecte el grafo), a menos que no haya otra alternativa.

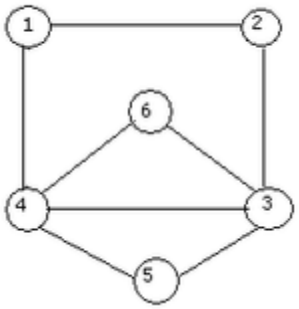
PASO 4

Desconectar los vértices que están unidos por la arista seleccionada.

PASO 5

Si todos los vértices ya están desconectados, ya se tiene el circuito de Euler. De otra forma continuar con el paso 3.

Ejemplo



C = {1,2,3,4,6,3,5,4,1} es un ciclo euleriano, por ende es un grafo euleriano.

Un grafo es una representación, un modelo, compuesto por un número determinado de vértices (nodos) y un número de arcos (aristas) que los relacionan, cada arista o arco tiene la capacidad de relacionar dos nodos. La palabra ciclo se emplea en teoría de grafos para indicar un camino cerrado en un grafo, es decir, en que el nodo de inicio y el nodo final son el mismo.

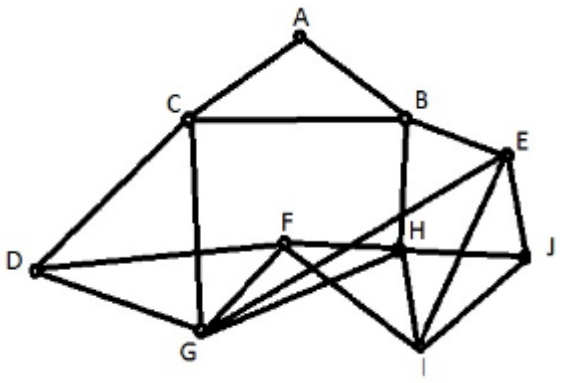
c. Circuito Hamiltoniano

Un camino hamiltoniano, en el campo matemático de la teoría de grafos, es un camino de un grafo, una sucesión de aristas adyacentes, que visita todos los vértices del grafo una sola vez. Si además el último vértice visitado es adyacente al primero, el camino es un ciclo hamiltoniano. El problema de encontrar un ciclo (o camino) hamiltoniano en un grafo arbitrario se sabe que es NP-completo.

ALGORITMO PARA UN CIRCUITO HAMILTONIANO

```
mientras que hay conflagraciones no probadas{
  do
        if ( hay bordes entre dos vértices consecutivos de este  y hay un borde desde
        el último vértice hasta el primero).{
              imprimir esta configuración;
              break;
  }
}
```

Ejemplo



C = {D,G, F, H, I, J, E, B, A, C, D} es un ciclo hamiltoniano.

Algoritmo de Dijkstra

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en

un grafo que tiene pesos en cada arista. Su nombre alude a Edsger Dijkstra, científico de la computación de los Países Bajos que lo concibió en 1956 y lo publicó por primera vez en 1959. La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene. Se trata de una especialización de la búsqueda de costo uniforme y, como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

ALGORITMO DE DIJKSTRA

PASO 1

Marca el nodo inicial que elegiste con una distancia actual de 0 y el resto con infinito.

PASO 2

Establece el nodo no visitado con la menor distancia actual como el nodo actual A.

PASO 3

Para cada vecino V de tu nodo actual A: suma la distancia actual de A con el peso de la arista que conecta a A con V. Si el resultado es menor que la distancia actual de V, establécelo como la nueva distancia actual de V.

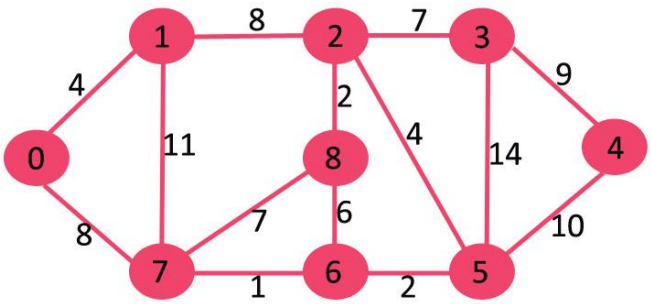
PASO 4

Marca el nodo actual A como visitado.

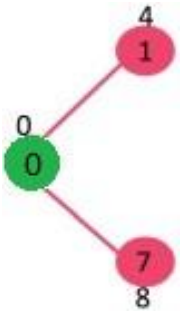
PASO 5

Si hay nodos no visitados, ve al paso 2.

Ejemplo

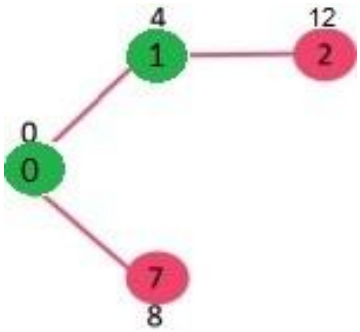


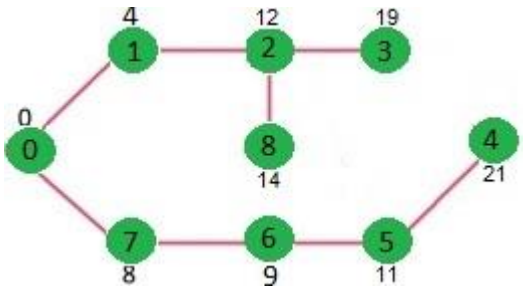
Dado el siguiente grafo.



Elegir el vértice con el valor de distancia mínimo. Se selecciona el vértice 0, inclúyalo en sptSet . Entonces sptSet se convierte en {0}. Después de incluir 0 en sptSet , actualice los valores de distancia de sus vértices adyacentes. Los vértices adyacentes de 0 son 1 y 7. Los valores de distancia de 1 y 7 se actualizan como 4 y 8. El siguiente subgráfico muestra los vértices y sus valores de distancia, solo se muestran los vértices con valores de distancia finitos. Los vértices incluidos en SPT se muestran en color verde.

Elegir el vértice con un valor de distancia mínimo y que no esté incluido en SPT (no en sptSET). El vértice 1 se selecciona y se agrega a sptSet. Entonces sptSet ahora se convierte en {0, 1}. Actualice los valores de distancia de los vértices adyacentes de 1. El valor de distancia del vértice 2 se convierte en 12.





Repetimos los pasos anteriores hasta que sptSet incluya todos los vértices del gráfico dado. Finalmente, obtenemos el siguiente árbol de ruta más corta (SPT).

Algoritmo de 2-opt

El algoritmo de 2 opciones se considera un algoritmo de mejora local para el TSP basado en modificaciones de recorrido simples. Dado un recorrido factible, el algoritmo luego realiza repetidamente una secuencia de operaciones, siempre que cada una reduzca la duración del recorrido actual, hasta que se alcance un recorrido para el cual ninguna operación produce una mejora (un recorrido localmente óptimo).

El algoritmo 2-Opt fue propuesto por primera vez por Croes. El algoritmo 2-Opt elimina dos bordes, dividiendo así el recorrido en dos caminos, y luego vuelve a conectar esos caminos de la otra manera posible (solo si la suma de la longitud de los bordes recién agregados es menor que la suma de la longitud del bordes eliminados).

ALGORITMO DE 2-OPT

PASO 1

Deja S ser la solución inicial, F S su valor de función objetivo. Establecer S* = S , i = 1 , j = i + 1 = 2 .

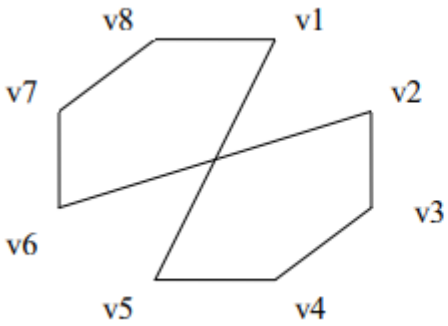
PASO 2

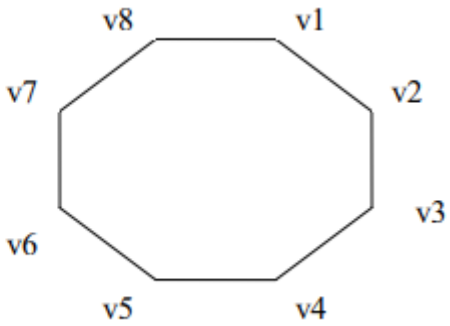
Considere el resultado del intercambio S ' como f(S ') < f S*. Establecer S* = S '. Si j < n repita el paso 2. De lo contrario, establezca i = i + 1 y j = i + 1. Si i < n, repita el paso 2 ; de lo contrario, vaya al paso 3 .

PASO 3

Si S ≠ S ∗ establecer S = S ∗ , i = 1 , j = i + 1 y vaya al paso 2 . De lo contrario, la salida será la mejor solución S y finalizar el proceso.

Ejemplo





El recorrido inicial se muestra a la izquierda. Esto se ha mejorado eliminando los bordes "v1v5" y "v2v6" y agregando los bordes "v1v2" y "v5v6".

BIBLIOGRAFIA

- Problema del viajante, extraído de https://es.wikipedia.org/wiki/Problema_del_viajante
- Algoritmo de Christofides, extraído de https://en.wikipedia.org/wiki/Christofides_algorithm
- Algoritmo de Prim, extraído de https://es.wikipedia.org/wiki/Algoritmo_de_Prim#Pseudoc%C3%B3digo_del_algoritmo
- Ciclo Euler, extraído de http://caminoseuler.blogspot.com/p/algoritmo-leury.html
- Algoritmo de Dijkstra, extraído de https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- Algoritmo de 2-opt, extraído de https://www.intechopen.com/books/novel-trends-in-the-traveling-salesman-problem/cuda-accelerated-2-opt-local-search-for-the-traveling-salesman-problem