PROGRAMA 1 – Algoritmo de KRUSKAL

Asignatura: INTELIGENCIA ARTIFICIAL
Profesor: D. Sc. Gerardo García Gil
2022-B
López Arellano Ricardo David
Universidad de Guadalajara (CUCEI)

Presentación

En este documento hablaremos sobre el algoritmo de Kruskal el cual es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. El objetivo del algoritmo de Kruskal es construir un árbol (subgrafo sin ciclos) formado por aristas sucesivamente seleccionados de mínimo peso a partir de un grafo ponderado. Un árbol (spanning tree) de un grafo es un subgrafo que contiene todos sus vértices o nodos. Un grafo puede tener múltiples árboles. Por ejemplo, un grafo completo de cuatro nodos (todos relacionados con todos) tendría 16 árboles.

Introducción

El algoritmo de Kruskal es conocido como el algoritmo tacaño, puesto que siempre busca el menor coste posible. En 1956 el matemático americano norteamericano Joseph Kruskal descubrió un algoritmo muy simple cuya aplicación nos garantiza encontrar un árbol generador mínimo en cualquier grafica ponderada. Este algoritmo busca las distancias más cortas (de menor costo) de un árbol expandido que conectan todos los vértices. Siendo así, una variación del algoritmo de mínima conexión.

La idea principal de Kruskal consiste en ser ambicioso, escogiendo siempre la arista menos costosa disponible y cuidando que en cada paso del proceso no se forme ningún circuito. La aplicación típica de este problema es el diseño de redes telefónicas. Una empresa con diferentes oficinas, trata de trazar líneas de teléfono para conectarlas unas con otras. La compañía telefónica le ofrece está interconexión, pero ofrece tarifas diferentes o costos por conectar cada par de oficinas. ¿Cómo conectar entonces las oficinas al mínimo costo total?

Antecedentes

Kruskal creó un algoritmo para encontrar un árbol encubridor mínimo en un grafo ponderado y convexo. Este algoritmo de la teoría de grafos busca un subconjunto de aristas que incluyen todos los vértices formando un árbol y donde todos los valores de las aristas de este son mínimas. Si el grafo no es convexo, busca un bosque de expandido mínimo.

Desarrollo

1) Elige la arista de menor peso (en caso de empate elige una arbitrariamente)
2) Elige la siguiente arista disponible de menor peso. Si hay más de una, elige una arbitrariamente.

3) Elige la siguiente arista disponible de menor peso. Que no cierra un circuito con las aristas ya elegidas. Si hay más de una, elige arbitrariamente.
4) Una gráfica de N vértices, repite la regla 3 hasta que se hayan elegido n-1 aristas de la gráfica. Los vértices de la grafica y las n-1 aristas así elegidas constituyen el árbol generador mínimo.

Implementación

```
#include <stdio.h>
#include <algorithm>
#include <cstring>
#define MAX 100  //VERTICES MAXIMOS

int padre[ MAX ]; //Este arreglo contiene el padre del ultimo nodo

//Método de inicialización
void hacer_raiz( int n ){
   for( int i = 1 ; i <= n ; ++i ) padre[ i ] = i;
}

//Método para encontrar la raiz del vértice actual
int buscar_raiz( int x ){
   return ( x == padre[ x ] ) ? x : padre[ x ] = buscar_raiz( padre[ x ] );
}

//Método para unir 2 vertices
void Union( int x , int y ){
   padre[ buscar_raiz( x ) ] = buscar_raiz( y );
}

//Método que me determina si 2 vértices estan o no en la misma componente conexa
bool vertices_conexos( int x , int y ){
   if( buscar_raiz( x ) == buscar_raiz( y ) ) return true;
   return false;
}

int V , A;//numero de vertices y aristas

struct Estructura_Arista{
   int origen;     //Vértice origen
   int destino;    //Vértice destino
   int peso;       //Peso entre el vértice origen y destino
   Estructura_Arista(){}
   bool operator<( const Estructura_Arista &e ) const {
      return peso < e.peso;
   }
}arista[ MAX ];//Arreglo de aristas
Estructura_Arista MST[ MAX ];     //Arreglo de aristas del MST encontrado

void Kruskal(){
   int origen , destino , peso;
   int total = 0;         //Peso total
   int numAristas = 0;    //Numero de Aristas

   hacer_raiz( V );          //Inicializamos cada componente
```

```
   std::sort( arista , arista + A );    //Ordenamos las aristas por su comparador

   for( int i = 0 ; i < A ; ++i ){     //Recorremos las aristas ya ordenadas por peso
      origen = arista[ i ].origen;    //Vértice origen de la arista actual
      destino = arista[ i ].destino;  //Vértice destino de la arista actual
      peso = arista[ i ].peso;        //Peso de la arista actual

      //Verificamos si estan o no en la misma componente conexa
      if( !vertices_conexos( origen , destino ) ){  //Evitar ciclos
         total += peso;             //Incrementa el peso total del MST
         MST[ numAristas++ ] = arista[ i ];  //Agregar al MST la arista actual
         Union( origen , destino );  //Union de ambas componentes en una sola
      }
   }

   if( V - 1 != numAristas ){     //Si el MST encontrado no posee todos los vértices mostramos mensaje de error
      puts("No existe MST (ARBOL DE EXPANSION MINIMO) valido para el grafo ingresado,\nEl grafo debe ser
conexo...");
      return;
   }
   puts( "\n\t< El MST (ARBOL DE EXPANSION MINIMO) \n\tencontrado contiene las siguientes aristas >");
   printf("\nVertice origen|Vertice destino|Peso arista: \n");
   for( int i = 0 ; i < numAristas ; ++i )
      printf("( \t%d \t-->\t%d )   =\t %d\n" , MST[ i ].origen , MST[ i ].destino , MST[ i ].peso ); //( vertice u ,
vertice v ) : peso
   printf( "\nEl costo minimo de todas las aristas del MST es : %d\n" , total );
}

int main(){
   int mst;
   printf("\n\t\t < ALGORITMO DE KRUSKAL >\n\n");
            printf("Ingresa el numero de vertices: ");
   scanf("%d", &V);
            printf("Ingresa el numero de aristas: ");
   scanf("%d",&A );
   printf("\nIngresa el vertice origen, despues el vertice destino y\nal final el peso de la arista: \n");
            for( int i = 0 ; i < A ; ++i )
      scanf("%d %d %d" , &arista[ i ].origen , &arista[ i ].destino , &arista[ i ].peso );

   Kruskal();
   return 0;
}
```

Experimentos

Para lograr programar el algoritmo de kruskal tuve que intentar hacerlo desde en Python, java y termine haciéndolo en c++ ya que siento que ahí soy más fuerte y podía realizarlo de una mejor manera.

Resultados

Conclusión

El algoritmo de Kruskal siempre tiene una solución optima a este tipo de problemas donde se busca el árbol de expansión mínimo eso quiere decir que este algoritmo pertenece a P porque se puede resolver de forma eficiente por una maquina determinista en tiempo polinomial.

Tuve algo de complicaciones para programarlo para a fin de cuentas se pudo sacar el programa funcionando correctamente.

Referencias

1. https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal
2. https://sites.google.com/site/complejidadalgoritmicaes/kruskal
3. http://arodrigu.webs.upv.es/grafos/doku.php?id=algoritmo_kruskal