GRUPO 5
INTEGRANTES:

CRISTIAN FERNANDEZ
JORGE MONDRAGON
LEONARDO HERRERA

TEMA: Algoritmo Dijkstra
TALLER 5

1. Encontrar un problema que se solucione mediante algoritmos avaros.
2. Publicar en la wiki: El enunciado, la justificación de ¿Porqué es conveniente el uso de los algoritmos avaros? Y análisis del Algoritmo.
3. Implementar en java el Algoritmo.

Algoritmos Avaros

Algoritmo de Dijkstra

El método de Dijkstra, también llamado el método del camino más cortos; se utiliza Precisamente para eso, para encontrar la longitud del camino más corto de un grafo, conexo, simple y con pesos positivos. El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo dirigido y con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959. El algoritmo de Dijkstra, también llamado *algoritmo de caminos mínimos* fue descubierto por Edsger Dijkstra, Edsger fue un científico de la computación de origen neerlandés y diseño el algoritmo en 1959. Según la definición de wikipedia *"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, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de costo 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)."*

Descripción conceptual

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, al resto de vértices que componen el

grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de costo 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).

Descripcion del problema

El algoritmo de Dijkstra también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista, el proyecto consiste en implementar dicho algoritmo aplicándolo a una matriz de incidencia. Los datos serán proporcionados mediante dos archivos de entrada, el primero contendrá las filas y columnas de la matriz y el segundo contendrá, los valores para completar la matriz de incidencia (pesos), los vértices serán presentados haciendo uso del alfabeto ingles. Los archivos podrán ser cargados buscando su localización en cualquier medio de almacenamiento secundario, además, también se debe de ser capaz de proveer la opción de cargar archivos mediante conexión remota (es decir, se le proporcionara un URL).
Una vez cargados los archivos de entrada y armada la matriz de incidencia, se produce a mostrar gráficamente las diferentes rutas que conectan todos los vértices, la aplicación debe proveer una opción en la cual, se ingresen dos vértices, estos corresponde al punto origen (partido) y el punto destino (finalización) con los cuales debe encontrarse el camino más corto que debe calcularse mediante la implementación del algoritmo de Dijkstra, una vez calculado dicho camino debe mostrarse gráficamente su recorrido, debido a que pueden existir varios recorridos que cumplan con lo establecido, la aplicación debe identificar mediante un número entero los diferentes recorridos e ingresando en la aplicación el número correspondiente a un recorrido esta deberá redibujar todos los vértices y sus conexiones y mostrar el recorrido que se ha solicitado.

PROBLEMA DEL ALGORITMO DIJKSTRA

Algoritmo de Dijkstra

· Sea G=(V,A) un grafo dirigido y etiquetado.
· Sean los vértices $a \in V$ y $z \in V$; a es el vértice de origen y z el vértice de destino.
· Sea un conjunto $C \subset V$, que contiene los vértices de V cuyo camino más corto desde a
todavía no se conoce.
· Sea un vector D, con tantas dimensiones como elementos tiene V, y que "guarda" las distancias entre a y cada uno de los vértices de V.
· Sea, finalmente, otro vector, P, con las mismas dimensiones que D, y que conserva la información sobre qué vértice precede a cada uno de los vértices en el camino.
El algoritmo para determinar el camino de longitud mínima entre los vértices a y z es:

1. C ← V
2. Para todo vértice $i \in C$, $i \neq a$, se establece D$i \leftarrow \infty$; D$a \leftarrow 0$
3. Para todo vértice $i \in C$ se establece P$i = a$
4. Se obtiene el vértice $s \in C$ tal que no existe otro vértice $w \in C$ tal que D$w <$ Ds
o Si $s = z$ entonces se ha terminado el algoritmo.
5. Se elimina de C el vértice s: C ← C−{s}
6. Para cada arista $e \in A$ de longitud l, que une el vértice s con algún otro vértice $t \in C$,
o Si l+D$s <$ Dt, entonces:
1. Se establece D$t \leftarrow l$+Ds
2. Se establece P$t \leftarrow s$
7. Se regresa al paso 4

¿ Por qué es conveniente el uso de Algoritmos Avaros?

- Hay muchos algoritmos que pueden ser clasificados bajo la categoría de Algoritmos Avaros, Oportunistas (Greedy Algorithms). La idea es optar por la mejor opción de corto plazo. Sacar mayor provecho inmediato.
- Éstos se caracterizan por hacer la elección que parece mejor en el momento.
- Éstos no conducen siempre a una solución óptima, pero para muchos casos sí.
- Éstos producen una solución óptima cuando se puede llegar a ésta a través de elecciones localmente óptimas.
- Hay un parecido con programación dinámica (investigación de operaciones). La diferencia es que en programación dinámica, la elección en cada paso depende de la solución a un sub-problema. En los algoritmos avaros hacemos la elección que parece mejor en el momento y luego se resuelve el sub-problema que queda.
- Se puede pensar que algoritmos avaros resuelven el problema en forma top-down (de arriba hacia abajo); mientras que la programación dinámica lo hace bottom-up (de abajo hacia arriba).
- Éste método es muy poderoso y trabaja bien en muchos algoritmos que veremos en las próximas semanas; por ejemplo: Minimum-spanning-tree (árbol de mínima extensión), el algoritmo de Dijkstra para el camino más corto en un grafo y desde una única fuente.

¿ Por qué es conveniente Dijkstra?

Es usado en muchas cosas, como por ejemplo Planificación y Explotación del Transporte, Caminos Canales y Puertos, obtener el camino mínimo, ruta más corta entre las ciudades, Una red de comunicaciones involucra un conjunto de nodos conectadas mediante arcos, que transfiere vehículos desde determinados nodos origen a otros nodos destino. La forma más común para seleccionar la trayectoria (o ruta) de dichos vehículos, se basa en la formulación de la ruta más corta. En

DOC.MX

particular a cada arco se le asigna un escalar positivo el cual se puede ver como su longitud.

Un algoritmo de trayectoria más corta, cada vehículo a lo largo de la trayectoria de longitud mínima (ruta más corta) entre los nodos origen y destino. Hay varias formas posibles de seleccionar la longitud de los enlaces. La forma más simple es que cada enlace tenga una longitud unitaria, en cuyo caso, la trayectoria más corta es simplemente una trayectoria con el menor número de enlaces. De una manera más general, la longitud de un enlace puede depender de su capacidad de transmisión y su carga de tráfico.

La solución es encontrar la trayectoria más corta. Esperando que dicha trayectoria contenga pocos enlaces no congestionados; de esta forma los enlaces menos congestionados son candidatos a pertenecer a la ruta

Metas del programa en Java

· Calcular el camino más corto de un grafo conexo
· Mostrar el (o los) camino más corto en forma gráfica y en forma descriptiva.

Funciones

Referencia	Función
1.0	Cargar los archivos de entrada por cualquier medio secundario de almacenamiento o mediante conexión remota.
1.1	Parsear los archivos de entrada
1.2	Nombrar los vértices y lados, colocando también el peso correspondiente a cada lado.
1.3	Dibujar el grafo con sus vértices y lados correspondientes
1.4	Calcular el camino más corto a través de la implementación del algoritmo de Dijkstra.
1.5	Mostrar (si es que existe más de uno) todos los caminos míniales existentes en el grafo.

Descripción: El usuario solicita al sistema calcular el camino más corto del grafo cargado
previamente (matriz de incidencia y archivo de pesos).La aplicación pide el vértice de partida y el vértice de llegada. El usuario ingresa lo solicitado y la aplicación calcula el camino más corto, por medio del algoritmo de Dijkstra.

Flujo Base:

1. El usuario solicita a la aplicación calcular el camino más corto del grafo antes cargado.
2. La aplicación solicita el vértice de llegada y el vértice de partida.
3. El usuario ingresa los datos
4. Excepción: el sistema verifica que los vértices no sean iguales y que exista en el grafo, sino

existen: Mensaje de Error y acaba el caso de uso, de lo contrario: *paso 5*.
5. La aplicación calcula el camino más corto, si hay más de un camino corto, solicita al
usuario ingresar el número de camino ha mostrar.
6. El usuario ingresa el número del camino.
Excepción: valida que el camino exista y que sea un número ingresado.

Pseudocódigo: Algoritmo Dijkstra
[inicio] => buscar el nodo que contiene el vértice de destino del grafo y cambiar la etiqueta a cero
del vértice de partida.
[condición]=> ¿la etiqueta del vértice de destino es "false"?. Si es falsa(marca permanente) salir
del algoritmo y retornar la longitud del camino más corto, de lo contrario si es true (marca temporal)
continuar con el algoritmo.
[marca]=> buscar el vértice entre los vértices con marca "true" y cambia la marca, por la marca
"false".
[Vertices adyacentes]=> recorre el grafo en busca de los vértices adyacentes.
[Etiqueta minima]=> Se calcula la nueva etiqueta y se compara para ver si es minima o permanece
la que contiene.
[regreso]=> ir al estado "condición".

Pseudocódigo, para encontrar los vértices del o los caminos más cortos.
Consideración anterior: Se considera que el grafo ya ha sido cargado, con sus vértices, lados y
pesos correspondientes y ya se sabe el camino mínimo.

INICIO

CaminoMínimo _ Se toma de la implementación del algoritmo de Dijkstra
Vpartida _ Tomar el vértice de partida
Vdestino _ Tomar el vértice destino
Encontrar todos vértices adyacentes a Vpartida que poseen la etiqueta de permanentes (esto
según el algoritmo de Dijkstra).
Para (cada vértice adyacente al Vpartida)
Inicio
Si (el peso del lado [incidente a Vpartida y al vértice actual] es menor que CaminoMínimo)
g _ Guardar el lado en un nuevo grafo
Ir a procedimiento CaminoMinimal(Vactual, Vdestino, Grafo g, pesoActual)
Si (el peso del lado [incidente a Vpartida y al vértice actual] es igual a CaminoMínimo y
además el vértice ayacente es el Vdestino)

g _ Guarda el lado en un nuevo grafo
Agrega el nuevo grafo a la ListaGrafo
Si (el peso del lado[incidente a Vpartida y al vértice actual] es mayo a CaminiMínimo)
El lado se descarta
Pasar al siguiente vertice
Fin

FIN

Procedimiento recursivo **CaminoMinimal(Vactual, Vdestino, Grafo g, pesoActual)**
Inicio
Vactual _ el vértice al que le aplicaremos el procedimiento
Vdestino _ el vértice destino del camino mínimo
Grafo g _ contiene los vértices ya recorridos para llegar a este procedimiento
Encontrar todos los vértices adyacentes a Vactual que poseen la etiqueta de permanentes (según
el algoritmo de Dijkstra)
Para (cada vértice adyacente a Vactual)

Inicio

Si (pesoActual + el peso del lado [incidente entre Vactual y su ayacente actual] es menor
que caminoMinimo)
g1 _ guarda el lado en un nuevo grafo
agrega los lados de g a los lados de g1
Vactual _ el vértice adyacente al Vactua
ir al procedimiento CaminiMinimal(Vactual, Vdestino, g1, pesoActual+pesoDelado)
Si (pesoActual + el peso del lado es igual a CaminoMínimo y además el vértice adyacente
es el vértice Vdestino)
G1 _ guarda el lado en un nuevo grafo
Agrega los lados de g a los lados de g1
Agrega el nuevo grafo (g1) a ListaGrafo
Si (pesoActual + el peso del lado es mayor que CaminoMínimo)
El lado incidente se descarta
Siguiente vértice adyacente.
Fin

Fin

XDOC.MX

CODIGO JAVA

```
package algoritmos.dijkstra.unipiloto;



import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Canvas;
import java.awt.Choice;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Event;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.Insets;
import java.awt.Label;
import java.awt.Panel;
import java.awt.Point;
import java.awt.TextArea;

public class AlgoritmoD extends java.applet.Applet {
  GraphCanvas grafocanvas = new GraphCanvas(this);
  Options options = new Options(this);
  Documentacion documentacion = new Documentacion();
  public void init() {
        setLayout(new BorderLayout(10, 10));
        add("Center", grafocanvas);
        add("North", documentacion);
        add("East", options);
  }
  public Insets insets() {
        return new Insets(10, 10, 10, 10);
  }
  public void lock() {
        grafocanvas.lock();
        options.lock();
  }
  public void unlock() {
        grafocanvas.unlock();
        options.unlock();
```

DOC.MX

```
  }
}

class Options extends Panel {
//opciones a un lado de la pantalla
  Button b1 = new Button("limpiar");
  Button b2 = new Button("ejecutar");
  Button b3 = new Button("paso");
  Button b4 = new Button("inicializar");
  Button b5 = new Button("ejemplo");
  Button b6 = new Button("salir");
  AlgoritmoD parent;
  boolean Locked=false;

  Options(AlgoritmoD myparent) {
        parent = myparent;
        setLayout(new GridLayout(6, 1, 0, 10));
        add(b1);
        add(b2);
        add(b3);
        add(b4);
        add(b5);
        add(b6);
  }
  public boolean action(Event evt, Object arg) {
        if (evt.target instanceof Button) {
          if (((String)arg).equals("paso")) {
            if (!Locked) {
              b3.setLabel("siguiente paso");
              parent.grafocanvas.stepalg();
            }
            else parent.documentacion.doctext.showline("cerrado");
          }
          if (((String)arg).equals("siguiente paso"))
            parent.grafocanvas.nextstep();
          if (((String)arg).equals("inicializar")) {
            parent.grafocanvas.inicializar();
            b3.setLabel("paso");
            parent.documentacion.doctext.showline("referencia");
          }
          if (((String)arg).equals("limpiar")) {
            parent.grafocanvas.limpiar();
            b3.setLabel("paso");
            parent.documentacion.doctext.showline("referencia");
          }
          if (((String)arg).equals("ejecutar")) {
            if (!Locked)
```

DOC.MX

```
            parent.grafocanvas.runalg();
          else parent.documentacion.doctext.showline("cerrado");
        }
        if (((String)arg).equals("ejemplo")) {
          if (!Locked)
            parent.grafocanvas.showejemplo();
          else parent.documentacion.doctext.showline("cerrado");
        }
        if (((String)arg).equals("salir")) {
          System.exit(0);
        }
      }
      return true;
  }

  public void lock() {
      Locked=true;
  }
  public void unlock() {
      Locked=false;
      b3.setLabel("paso");
  }
}

class Documentacion extends Panel {
//Documentacion arriba de la pantalla
  DocOptions docopt = new DocOptions(this);
  DocText doctext = new DocText();
  Documentacion() {
      setLayout(new BorderLayout(10, 10));
      add("West", docopt);
      add("Center", doctext);
  }
}

class DocOptions extends Panel {
  Choice doc = new Choice();
  Documentacion parent;

  DocOptions(Documentacion myparent) {
      setLayout(new GridLayout(2, 1, 5, 0));
      parent = myparent;
      add(new Label("DOCUMENTACION:"));
      doc.addItem("dibujar nodos");
      doc.addItem("remover nodos");
      doc.addItem("mover nodos");
      doc.addItem("el nodo_inicial");
```

DOC.MX

```
        doc.addItem("dibujar aristas");
        doc.addItem("cambiar pesos");
        doc.addItem("remover aristas");
        doc.addItem("limpiar / inicializar");
        doc.addItem("ejecutar algoritmo");
        doc.addItem("pasar");
        doc.addItem("ejemplo");
        doc.addItem("salir");
        doc.addItem("referencia");
        add(doc);
    }

    public boolean action(Event evt, Object arg) {
        if (evt.target instanceof Choice) {
            String str=new String(doc.getSelectedItem());
            parent.doctext.showline(str);
        }
        return true;
    }
}

class DocText extends TextArea {
    final String drawnodos = new String("DIBUJAR NODOS:\n"+
        "Dibuje un nodo haciendo click en el mouse.\n\n");
    final String rmvnodos = new String("REMOVER NODOS:\n"+
        "Para remover un nodo presione <ctrl> y haga click en el nodo.\n"+
        "No se puede remover el nodo_inicial.\n"+
        "Seleccione otro nodo_inicial, asi podra remover el nodo.\n\n");
    final String mvnodos = new String("MOVER NODOS\n"+
        "Para mover un nodo presione <Shift>, haga click en el nodo,\ny
arrastrelo a"+
        " su nueva posicion.\n\n");
    final String nodo_inicial = new String("NODO INICIAL:\n"+
        "El nodo_inicial es azul, los otros nodos son grises.\n"+
        "El primer nodo que usted dibuja en la pantalla sera el
nodo_inicial.\n"+
        "Para seleccionar otro nodo_inicial presione <ctrl>, haga click en el
nodo_inicial,\n"+
        "y arrastre el mouse a otro nodo.\n"+
        "Para borrar el nodo_inicial, primero seleccione otro nodo_inicial, y
despues"+
        "\nremueva el nodo normalmente.\n\n");
    final String drawaristas = new String("DIBUJAR ARISTAS:\n"+
        "Para dibujar una arista haga click al mouse en un nodo,"+
        "y arrastrelo a otro nodo.\n\n");
    final String peso = new String("CAMBIAR PESOS:\n"+
        "Para cambiar el peso de una arista, haga click en la flecha y \n"+
```

XDOC.MX