Universidad Autónoma del Estado de México

Centro Universitario UAEM Atlacomulco

“Dijkstra”

Presenta:

Elias Edgardo Segundo Antonio

Revisa:

Ingeniero José Luis García Morales

Unidad de Aprendizaje:

Programación avanzada

Fecha de entrega:

26 de abril de 2017

En este informe se desarrollará el proceso y pruebas del programa que implementa el algoritmo de Dijkstra para encontrar las distancias más cortas, primero se mostrará el grafo de prueba utilizado para realizar el proyecto, después el proceso de solución a mano y finalmente la prueba del programa.

Dicho programa según las especificaciones del proyecto debía de permitir:

1.- Meter la matriz de costos

2.- Ver el grafo original (este lo dibujan en HTML, en canvas)

3.- Resolver el problema

4.- Ver el grafo con las distancias más cortas

5.- Salir

Sin embargo, debido al diseño del programa realizado, se tuvieron las siguientes excepciones:

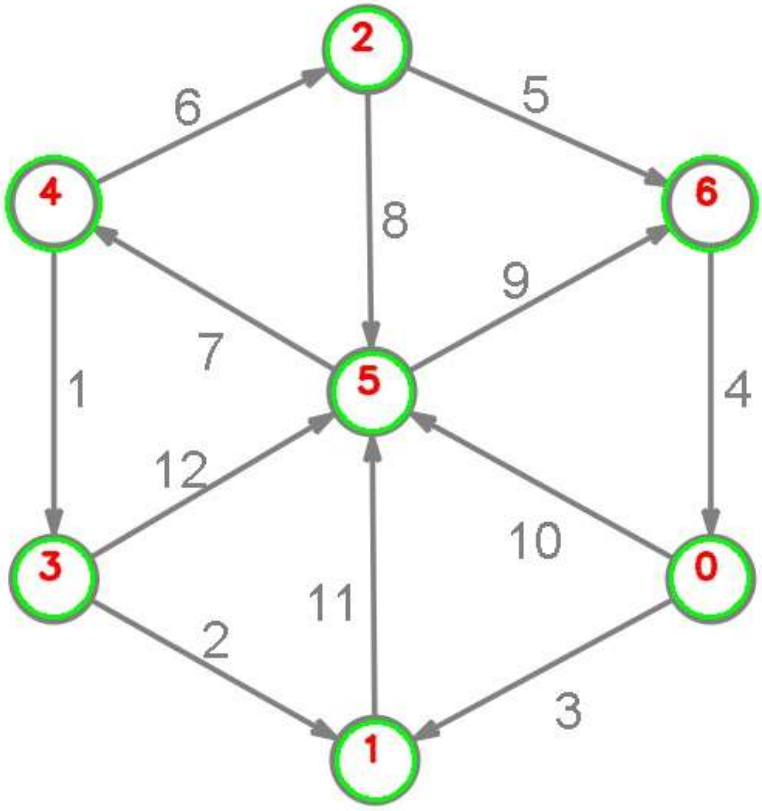
- El grafo no será ingresado mediante la consola, se utilizó la librería de visión por computadora Open CV y el lenguaje de programación python, lo que elimina las siguientes opciones del menú:
    - Ya no es necesario un módulo para ingresar la matriz de costos, pues en este caso se utiliza como entrada una imagen con el grafo a procesar, en cambio se ocuparon varios módulos para tratar la imagen y detectar tanto nodos como conexiones automáticamente, complicando el desarrollo, pero simplificando la interfaz para posibles usuarios.
    - Por otra parte, tampoco es necesario un módulo para mostrar el grafo usando Canvas, dado que la entrada nativa será una imagen y esta misma servirá como consulta, sin embargo, si se genera una imagen con los nodos detectados que se guarda en la carpeta del programa para consultarla cuando se requiera.
    - Finalmente, tampoco se necesita una función para mostrar el grafo con las distancias más cortas, ya que después de terminar el proceso se guarda una imagen con los caminos dibujados sobre la misma, con la ventaja de ser totalmente dinámico y funcional para todas las imágenes con ruido moderado.

De esta manera, el programa se simplifica a lo siguiente:

1. Petición del nombre de la imagen con extensión (ubicada junto al programa)
2. Petición de las distancias.
3. Guardado de imagen con los resultados.

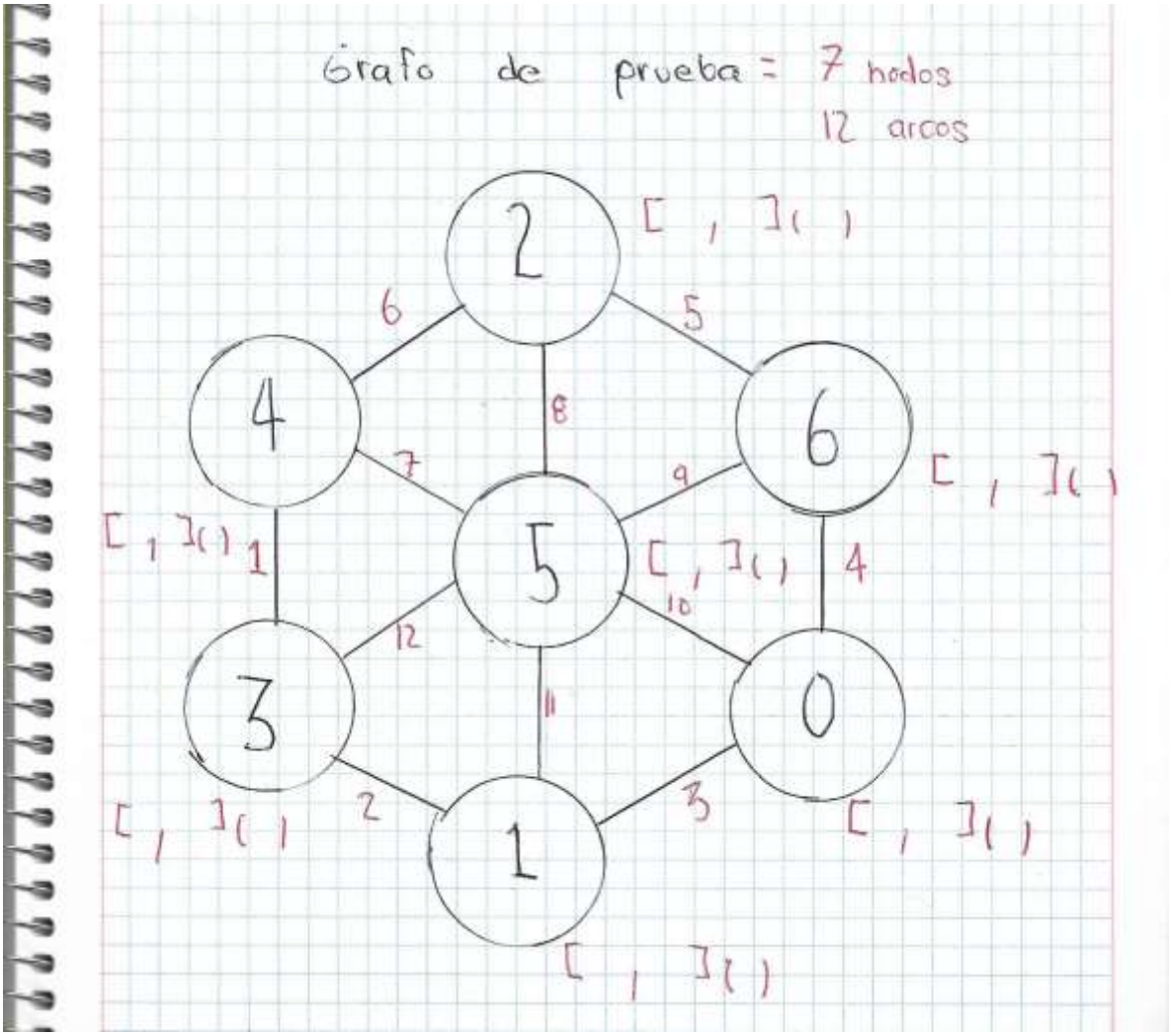
Proceso a mano

Para el proceso a mano y posterior prueba en el programa se planteó el siguiente grafo con 7 nodos y 12 arcos como requiere el proyecto:



1.-Grafo del proyecto

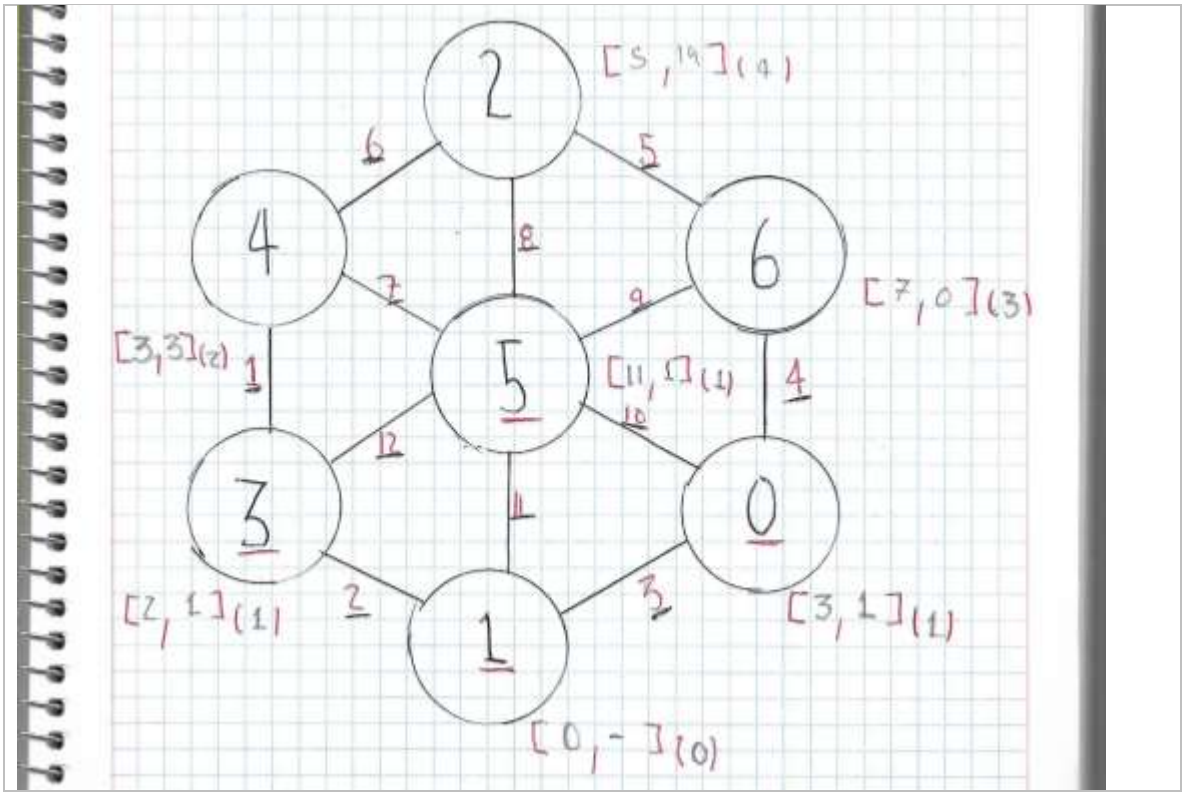
Para el proceso a mano se utilizará el siguiente esquema:



Dicho esquema sigue el siguiente etiquetado para realizar el algoritmo de Dijkstra:

[Distancia Recorrida, Nodo predecesor] (iteración)

A continuación, se muestran todas las capturas del proceso de etiquetado para rutas más cortas:



Ahora se mostrará el código fuente:

grafos.py

```
import cv2
import numpy as np
from math import *

INF=1e9

#Encontrar las conexiones
def getConections(lines,circles,ms):
                        #Se encuentran los adyacentes
                        adj=[]
                        for i in range(len(circles)):
                            row=[]
                            for j in range(len(circles)):
                                    row.append(-1)
                            adj.append(row)
                            #Todas las lineas son testeadas
                        for l in range(len(lines)):
                            d1=-1
```

```
                d2=-1
                x1,y1,x2,y2=lines[l][0][0],lines[l][0][1],lines[l][0][2],lines[l][0][3]
                size=sqrt( (x2-x1)**2  + (y2-y1)**2)
                #Encuentro una conexion para el primer punto de cualquier recta
                while d1==-1 and x1>0 and x1<2000 and (y1>0 and y1<2000):
                        for c in range(len(circles)):
                                cx,cy,cr=circles[c][0],circles[c][1],circles[c][2]
                                #cr+=70
                                if x1>=(cx-cr) and x1<=(cx+cr) and y1>=(cy-cr) and
y1<=(cy+cr):
                                        if d1==-1:
                                                d1=c
                        if ms[l]!=-1:
                                x1-=1
                                y1-=ms[l]
                        else:
                                y1-=1
                #Y despues para el segundo punto  extendiendo las lineas
                while d2==-1 and x2<2000 and x2<2000 and (y2>0 and y2<2000):
                        for c in range(len(circles)):
                                cx,cy,cr=circles[c][0],circles[c][1],circles[c][2]
                                #cr+=70
                                if x2>=(cx-cr) and x2<=(cx+cr) and y2>=(cy-cr) and
y2<=(cy+cr):
                                        if d2==-1:
                                                d2=c
                        if ms[l]!=-1:
                                x2+=1
                                y2+=ms[l]
                        else:
                                y2+=1

                #cv2.line(img2,(int(x1),int(y1)),(int(x2),int(y2)),(0,0,255),7)
                if(d1!=-1 and d2!=-1 and d1!=d2):
                        adj[d1][d2]=1
                        adj[d2][d1]=1
                elif d1==d2:
                        adj[d1][d2]=0
                        adj[d2][d1]=0
        for i in range(len(adj)):
                for j in range(len(adj[i])):
                        if adj[i][j]==-1:
                                adj[i][j]=0
        return adj
```

```
#Generadores
def inclination(lines):
                        for line in lines:
                            for x1,y1,x2,y2 in line:
                                    if x2-x1!=0 and x2!=x1:
                                            yield ( float (y2-y1)/(x2-x1)  )
                                    elif x2==x1:
                                            yield(-1)
                                    else: yield(0)


def adjacents(row):
                        for i in range(len(row)):
                            if row[i]!=0 and row[i]!=INF:
                                    yield(i)


#Se carga la imagen y se inicializan los arreglos
name=raw_input("Ingrese el nombre de la imagen con extension (debe estar junto a este
programa): ")
img=cv2.imread(name,0)
img2=cv2.imread(name)
img3=cv2.imread(name)
img = cv2.medianBlur(img,5)
C=[]

circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,20,
param1=50,param2=30,minRadius=20,maxRadius=50)
font = cv2.FONT_HERSHEY_SIMPLEX
circles = np.uint16(np.around(circles))
count=0
for i in circles[0,:]:
   cv2.circle(img2,(i[0],i[1]),i[2],(0,255,0),2)
   cv2.putText(img2,str(count),(i[0]-10,i[1]), font, 0.75,(0,0,255),2,cv2.LINE_AA)
   cv2.circle(img3,(i[0],i[1]),i[2],(0,255,0),2)
   cv2.putText(img3,str(count),(i[0]-10,i[1]), font, 0.75,(0,0,255),2,cv2.LINE_AA)
   count+=1



#Deteccion de las lineas
gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,80,120,apertureSize = 5)
minLineLength = 40
```

```
maxLineGap = 10
lines = cv2.HoughLinesP(edges,1,np.pi/270,100,minLineLength,maxLineGap)

#for line in lines:
                        #for x1,y1,x2,y2 in line:
                            #cv2.line(img2,(x1,y1),(x2,y2),(0,255,0),2)
m=list(inclination(lines))

c=circles[0]
adj=getConections(lines,c,m)

for i in range(len(adj)):
                        print adj[i]
                        #j=0
                        #for n in adj[i]:
                            #if i!=j and n:
                                    #cv2.line(img2, (c[i][0],c[i][1]) ,  (c[j][0],c[j][1])  , (255,0,0), 6  )
                            #j+=1



# Se guarda la imagen con nodos detectados
if cv2.imwrite("nodos_detectados.jpg",img2):
                        print("Se ha guardado la imagen 'nodos_detectados' en la ubicacion del
programa: ")
                        print("Revisar la imagen para ingresar la matriz de adyacencia")
                        print("")

# Se genera la matriz de distancias
for i in range(len(adj)):
                        row=[]
                        for j in range(len(adj[i])):
                            if i==j:
                                    dist=0
                            elif j < len(C):
                                    dist=C[j][i]
                            elif int(adj[i][j]):
                                    dist=int(raw_input("Cual es la distancia del nodo "+str(i)+" al
nodo "+ str(j)+"?: "))
                            else:
                                    dist=INF
                            row.append(dist)
                        C.append(row)
for a in C:
                        print a
```

```
# Inicializacion de las estructuras necesarias para Dijkstra
S=[]
NS=[]
pred=[]
d=[]
for i in range(len(C)):
                        NS.append(i)
                        pred.append(-1)
                        d.append(INF)

init=int(raw_input("Cual es el nodo inicial?: "))
d[init]=0

while len(S)!=len(C):
                        best=0
                        bestDistance=INF

                        for i in range(len(d)):
                            if d[i]<bestDistance and NS[i]!=-1:
                                    best=i

                        NS[best]=-1
                        S.append(best)
                        ady=list(adjacents(C[best]))

                        for y in ady:
                            if d[y] > d[best]+C[best][y]:
                                    d[y]=d[best]+C[best][y]
                                    pred[y]=best
print(S)
print(d)
print(pred)
c=circles[0]


#Se trazan las lineas en la imagen con ayuda del arreglo pred y se guarda la imagen
for i in range(len(pred)):
                        if pred[i]!=-1:
                            cv2.line(img3, (c[i][0],c[i][1]), (c[pred[i]][0],c[pred[i]][1]),(0,255,0),8 )
if cv2.imwrite("caminos_cortos.jpg",img3):
                        print("Se ha guardado una imagen 'caminos_cortos' con las rutas optimas")
print(S)
print(pred)
```

Capturas de pantalla

```
Ingrese el nombre de la imagen con extension (debe estar junto a este programa): tarea.png
[0, 1, 0, 0, 0, 1, 1]
[1, 0, 0, 1, 0, 1, 0]
[0, 0, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 0]
[0, 0, 1, 1, 0, 1, 0]
[1, 1, 1, 1, 1, 0, 1]
[1, 0, 1, 0, 0, 1, 0]
Se ha guardado la imagen 'nodos_detectados' en la ubicacion del programa:
Revisar la imagen para ingresar la matriz de adyacencia
```

```
Cual es la distancia del nodo 0 al nodo 1?: 3
Cual es la distancia del nodo 0 al nodo 5?: 10
Cual es la distancia del nodo 0 al nodo 6?: 4
Cual es la distancia del nodo 1 al nodo 3?: 2
Cual es la distancia del nodo 1 al nodo 5?: 11
Cual es la distancia del nodo 2 al nodo 4?: 6
Cual es la distancia del nodo 2 al nodo 5?: 8
Cual es la distancia del nodo 2 al nodo 6?: 5
Cual es la distancia del nodo 3 al nodo 4?: 1
Cual es la distancia del nodo 3 al nodo 5?: 12
Cual es la distancia del nodo 4 al nodo 5?: 7
Cual es la distancia del nodo 5 al nodo 6?: 9
[0, 3, 1000000000.0, 1000000000.0, 1000000000.0, 10, 4]
[3, 0, 1000000000.0, 2, 1000000000.0, 11, 1000000000.0]
[1000000000.0, 1000000000.0, 0, 1000000000.0, 6, 8, 5]
[1000000000.0, 2, 1000000000.0, 0, 1, 12, 1000000000.0]
[1000000000.0, 1000000000.0, 6, 1, 0, 7, 1000000000.0]
[10, 11, 8, 12, 7, 0, 9]
[4, 1000000000.0, 5, 1000000000.0, 1000000000.0, 9, 0]
Cual es el nodo inicial?: 1
[1, 5, 6, 4, 3, 2, 0]
[3, 0, 19, 2, 3, 11, 7]
[1, -1, 5, 1, 3, 1, 0]
Se ha guardado una imagen 'caminos_cortos' con las rutas optimas
[1, 5, 6, 4, 3, 2, 0]
[1, -1, 5, 1, 3, 1, 0]
```

Imagen generada por el programa