from imutils.video import VideoStream

from pyzbar import pyzbar

import argparse

import datetime

import imutils

import time

import cv2

from collections import defaultdict

def scanqr():

barcodeData=''

# construct the argument parser and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-o", "--output", type=str, default="barcodes.csv",

help="path to output CSV file containing barcodes")

args = vars(ap.parse\_args())

# initialize the video stream and allow the camera sensor to warm up

vs = VideoStream(src=0).start()

time.sleep(2.0)

# open the output CSV file for writing and initialize the set of

# barcodes found thus far

csv = open(args["output"], "w")

found = set()

# loop over the frames from the video stream

while True:

# grab the frame from the threaded video stream and resize it to

# have a maximum width of 400 pixels

frame = vs.read()

frame = imutils.resize(frame, width=400)

# find the barcodes in the frame and decode each of the barcodes

barcodes = pyzbar.decode(frame)

# loop over the detected barcodes

for barcode in barcodes:

# extract the bounding box location of the barcode and draw

# the bounding box surrounding the barcode on the image

(x, y, w, h) = barcode.rect

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

# the barcode data is a bytes object so if we want to draw it

# on our output image we need to convert it to a string first

barcodeData = barcode.data.decode("utf-8")

barcodeType = barcode.type

# draw the barcode data and barcode type on the image

text = "{} ({})".format(barcodeData, barcodeType)

cv2.putText(frame, text, (x, y - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

# if the barcode text is currently not in our CSV file, write

# the timestamp + barcode to disk and update the set

if barcodeData not in found:

csv.write("{},{}\n".format(datetime.datetime.now(),

barcodeData))

csv.flush()

found.add(barcodeData)

# show the output frame

cv2.imshow("Barcode Scanner", frame)

key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop

if barcodeData!='':

break

# close the output CSV file do a bit of cleanup

#print("[INFO] cleaning up...")

return(barcodeData)

csv.close()

cv2.destroyAllWindows()

vs.stop()

#Class to represent a graph

class Graph:

# A utility function to find the

# vertex with minimum dist value, from

# the set of vertices still in queue

def minDistance(self,dist,queue):

# Initialize min value and min\_index as -1

minimum = float("Inf")

min\_index = -1

# from the dist array,pick one which

# has min value and is till in queue

for i in range(len(dist)):

if dist[i] < minimum and i in queue:

minimum = dist[i]

min\_index = i

return min\_index

# Function to print shortest path

# from source to j

# using parent array

def printPath(self, parent, j):

#Base Case : If j is source

if parent[j] == -1 :

print (j,end='')

return

self.printPath(parent , parent[j])

print ('->',j,end='')

# A utility function to print

# the constructed distance

# array

def printSolution(self, dist, parent,src,des):

i=des

print("%d --> %d \ndistance:%d \t\t\t\t\t" % (src, i, dist[i]))

print('PATH: ',end='')

self.printPath(parent,i)

'''Function that implements Dijkstra's single source shortest path

algorithm for a graph represented using adjacency matrix

representation'''

def dijkstra(self, graph, src,des):

row = len(graph)

col = len(graph[0])

# The output array. dist[i] will hold

# the shortest distance from src to i

# Initialize all distances as INFINITE

dist = [float("Inf")] \* row

#Parent array to store

# shortest path tree

parent = [-1] \* row

# Distance of source vertex

# from itself is always 0

dist[src] = 0

# Add all vertices in queue

queue = []

for i in range(row):

queue.append(i)

#Find shortest path for all vertices

while queue:

# Pick the minimum dist vertex

# from the set of vertices

# still in queue

u = self.minDistance(dist,queue)

# remove min element

queue.remove(u)

# Update dist value and parent

# index of the adjacent vertices of

# the picked vertex. Consider only

# those vertices which are still in

# queue

for i in range(col):

'''Update dist[i] only if it is in queue, there is

an edge from u to i, and total weight of path from

src to i through u is smaller than current value of

dist[i]'''

if graph[u][i] and i in queue:

if dist[u] + graph[u][i] < dist[i]:

dist[i] = dist[u] + graph[u][i]

parent[i] = u

# print the constructed distance array

self.printSolution(dist,parent,src,des)

g= Graph()

graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],

[4, 0, 8, 0, 0, 0, 0, 11, 0],

[0, 8, 0, 7, 0, 4, 0, 0, 2],

[0, 0, 7, 0, 9, 14, 0, 0, 0],

[0, 0, 0, 9, 0, 10, 0, 0, 0],

[0, 0, 4, 14, 10, 0, 2, 0, 0],

[0, 0, 0, 0, 0, 2, 0, 1, 6],

[8, 11, 0, 0, 0, 0, 1, 0, 7],

[0, 0, 2, 0, 0, 0, 6, 7, 0]

]

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Scan the source QR CODE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s=int(scanqr())

print('Source node pinned successfully to node number : ',s)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Scan the destination QR CODE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

des=int(scanqr())

print('Destination node pinned successfully to node number : ',des)

# Print the solution

g.dijkstra(graph,s,des)