```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.firefox_profile import FirefoxProfile
from selenium.webdriver import FirefoxOptions

import warnings
warnings.filterwarnings("ignore")
from time import sleep
from collections import defaultdict, deque

import os, requests

# create a profile that disables caching
profile = FirefoxProfile()
profile.set_preference('browser.cache.disk.enable', False)
profile.set_preference('browser.cache.memory.enable', False)
profile.set_preference('browser.cache.offline.enable', False)
profile.set_preference('network.cookie.cookieBehavior', 2)

# For kali linux
# opts = FirefoxOptions()
# opts.add_argument("--headless true")
# opts.add_argument("--no-sandbox")
# opts.add_argument("--disable-gpu")
# opts.add_argument("--window-size=1920,1080")

# open geckodriver with that profile and get our class webpage
driver = webdriver.Firefox(firefox_profile = profile)#, options=opts)
driver2 = webdriver.Firefox(firefox_profile = profile)#, options=opts)

# Or read it from the step3 json file, pasting it here for now
dic = defaultdict(int,
            {1: 182859,
             2: 253880,
             3: 154018,
             4: 2258,
             5: 230588,
             6: 48865,
             7: 109593,
             8: 341472,
             9: 3367,
             10: 58278,
             11: 248544,
             12: 143584,
             13: 169507,
             14: 229726,
             15: 82225,
             16: 255593,
             17: 328785,
             18: 181659,
             19: 189007,
             20: 92155})

graph = defaultdict(list)
link_visited = []
threshold = 6000

def visit_page_get_image_total_size(link):
    driver2.get(link)
```

```
    total_size = 0

    # get list of images on that page
    lst_images = driver2.find_elements(By.TAG_NAME, "img")
    for img in lst_images:
        link_to_img = img.get_attribute("src")
        response = requests.get(link_to_img)

        with open('./temp.jpg', 'wb') as file:
            file.write(response.content)

        size = os.path.getsize("./temp.jpg")
        file.close()

        total_size += size

    # This will help us get closer to the actual size value
    # Otherwise some outputs will be ZERO, but that works since threshold is used
    # total_size += len(driver2.page_source)
    print(f"Total size for {link} is {total_size}")

    return total_size

def which_page_was_visited(curr_page_number, sizes):
    # if ghost_size is less than threshold (6000), it is only html page
    # therefore, there is a possibility for all pages that are html based
    # do a bfs for those pages only

    # the page ghost visited had this size
    ghost_size = dic[curr_page_number]
    print(type(ghost_size), ghost_size, sizes)

    if ghost_size < threshold:
        print("yes ghost size is small")
        possibilities = []
        for iter, val in enumerate(sizes):
            if val < threshold:
                possibilities.append(iter)

        return possibilities

    # otherwise there are images and it would be good to simply see that
    min_size = float('inf')

    # our size will be smaller than ghost_size always, just looking for the closest
one
    for iter, val in enumerate(sizes):
        print("---- WE GOT IMAGES ----")
        if (val < ghost_size) and (ghost_size - val < min_size):
            print(f'size for ghost was {ghost_size} and this page size had {val}')
            min_size = ghost_size - val
            page_num = iter

    return [page_num]

def which_page_was_visited_html(possible_page_numbers, next_to_next_page_size,
all_links):

    print("checking which html page was visited")
```

```
    print(f"Expecting a next to next page size of {next_to_next_page_size}")
    # possible page numbers say are [1, 5, 6, 7]
    dic_ = defaultdict(list)

    driver3 = webdriver.Firefox(firefox_profile = profile)#, options=opts)

    for pg_num in possible_page_numbers:
        print(f"Looing at page number {pg_num} in the set of page numbers
{possible_page_numbers}")
        linkX = all_links[pg_num].get_attribute("href")

        # visit that page
        driver3.get(linkX)

        # look at sizes of pages connected to that page
        sizes = []
        all_links_ = driver3.find_elements(By.XPATH, "//a[@href]")
        for link in all_links_:
            weblink = link.get_attribute("href")
            print(f"visiting link {weblink} connected to {linkX}")
            sizes.append (visit_page_get_image_total_size(weblink))
            sleep(1)

        dic_[pg_num] = sizes

    driver3.quit()

    # print(dic_)
    min_size = float('inf')
    most_likely_page_number = None

    for i, n in enumerate(dic_):
        # i is the index
        # n is the key
        for val in dic_[n]:
            if (val < next_to_next_page_size) and (next_to_next_page_size - val <
min_size):
                most_likely_page_number = n
                min_size = next_to_next_page_size - val

    return most_likely_page_number

def iterative_crawl_bfs(visited):
    next_page = 'https://computersecurityclass.com/4645316182537493008.html'

    link_visited.append(next_page)

    # q = deque([[0, page_1]]) # depth, pagelink
    # while q:

    i = 2
    while i <= len(dic) and next_page:
        driver.get(next_page)
        sleep(.1)

        # # get the first element in the queue
        # depth, k = q.popleft()

        # no need to go any further
```

```
        # if depth >= 22:
        #     return

        # visit that page
        # print(f"Visiting page: {k} at depth {depth}")
        # driver.get(k)
        # link_visited.append((depth, k))

        # wait for 6 seconds
        # sleep(6)

        # look for all links on that page
        all_links = driver.find_elements(By.XPATH, "//a[@href]")

        print("JUST EXTRACTED ALL LINKS")

        # store all those nodes in the dictionary and append them to q if not
visited before
        sizes = []
        for link in all_links:
            weblink = link.get_attribute("href")

            # if weblink not in visited:
            sizes.append (visit_page_get_image_total_size(weblink))
            # visited.add(weblink)
            sleep(.1)


        pg_number_selected = which_page_was_visited(i, sizes)
        # print("------------------------")
        # print(pg_number_selected)
        # print("------------------------")

        # case of images ~ straighforward
        if len(pg_number_selected) == 1:
            next_page = all_links[pg_number_selected[0]].get_attribute("href")

        # html only page ~ find using the next page size
        else:
            page_number = which_page_was_visited_html(pg_number_selected, dic[i+1],
all_links)
            # print(f"received a page_number value of {page_number}")
            next_page = all_links[page_number].get_attribute("href")

        # print(all_links)
        link_visited.append(next_page)
        print(next_page)

        i+=1
                # q.append([depth+1, weblink])
                # graph[(depth, k)].append(weblink)

    driver.quit()
    driver2.quit()



iterative_crawl_bfs(set())
```