**Algorithms – CS 5329.011**

**Fall 2018**

**PROJECT REPORT**

**Group – 1**

**Quick Sort**

**And**

**Web Crawler**

Submitted by

Nikitha Reddy Pullalarevu-A04862493

Apoorva Bapat-A04828475

**Median Finding, Order Statistics and Quick Sort**

**Introduction:**

**Order Statistics:** The *i*th order statistic of a set of n elements is the *i*th smallest element.

**Median Finding:** A median, is the “halfway point” of the set(array).

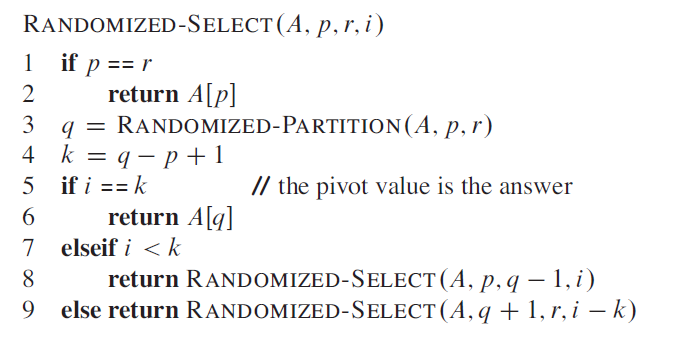
**QuickSort:** It uses divide and conquer algorithm to sort the given set of elements from A[1……..n].

**Project Overview**

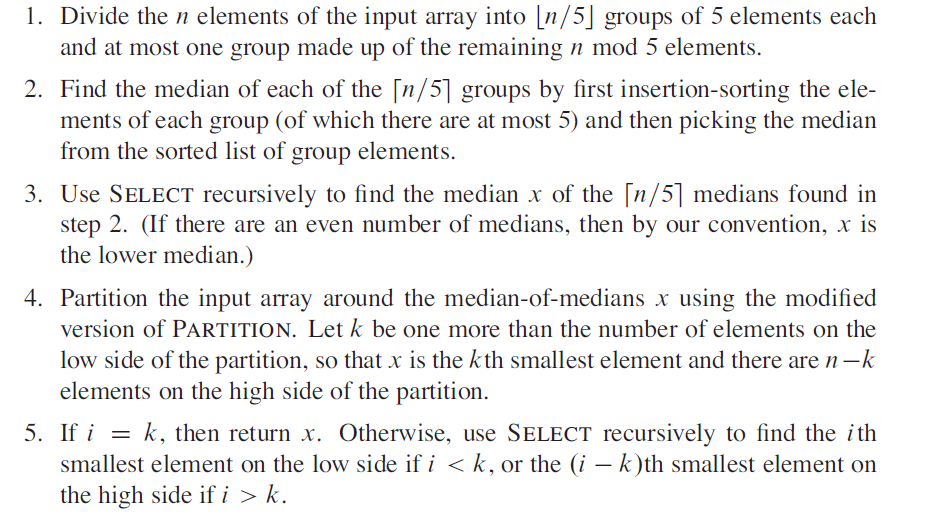
* Implementation of Median of Median with groups of 3,5, and 7.
* Implementation of Randomized Median finding algorithm.
* Comparing Median of Medians and Randomized Median Finding algorithm.
* Implementation of Quick Sort, where pivot is chosen from previous algorithms.
* Implementation Randomized Quicksort.
* Implementation of simplified Quicksort.
* Compare the performance of all implementations.

**Pseudocode**

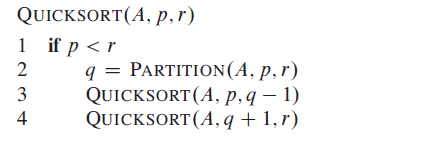
**Randomized Median:**

****

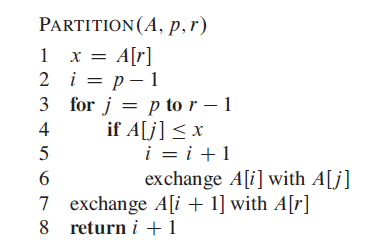
**Median Of Medians:**



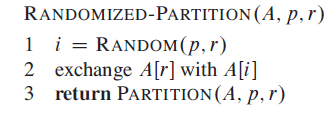
**QuickSort:**



**Partition:**

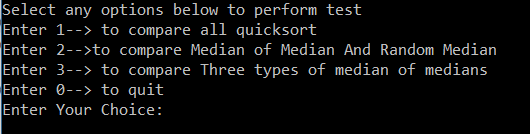


**Randomised Partition:**



**Working Of Code:**

* Compile main.cpp file and run the program to get the below ouput screen.



* Select any of the three options or zero to quit.
* Ouput Files will be produced for each test input the values in excel to get graph analysis.

**Results:**

**Graph:** y – axis : time in micro-seconds x-axis: input size.

**Comparison of Median Finding Using different Groups:**

**Seven Groups performed well.**

**Comparison of Median Finding and Random Median Finding:**

**Comparison of All Quicksort Functions:**

**Analysis:**

* Simplified Quick Sort Performance was highest.
* Quick Sort using median of medians Performance was below Simplified Quick Sort.
* Randomized Quicksort Performance was below Quick Sort using median.
* Normal Quicksort Performance was below all other implementations.

**References:**

1. Pseudocode: Introduction to algorithms by cormen.
2. Random Numbers generation: <https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution>
3. Execution Time of a function:

<https://www.geeksforgeeks.org/measure-execution-time-function-cpp/>

**Web Crawler**

1. **Web crawler overview**

A web crawler is a program which searches through the World Wide Web in a systematic way. It is a program that acts as an automated script. The WC looks at the keywords in the web pages, the kind of content each page has and the links, before returning the information gathered. Basically, web crawler gathers multiple pages from the web or a specific domain and then, indexes them in a systematic and automated manner. Another use of web crawlers is that they help in validating HTML codes and checking links.

Google is the most famous product making use of web crawlers, here the crawlers would go through each of the pages indexed in their database and fetch those pages to Google’s servers. The web crawler follows all the hyperlinks in the websites and visits other websites as well.

1. **Breadth First Search Algorithm**

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'[1]), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.



1. **Project**

In this project, we have chosen to parse or run the web crawler on the **cs.txstate.edu** domain only. In this domain, all the internal links (within cs domain) and external links (outside cs.domain) are searched and catalogued.

The main web crawler function is written using the **Breadth First Search BFS algorithm.**

**Steps:**

- The process is that, csu.txstate.edu will be the root node

- and then all the links in that page, will form the next level nodes, and for each link, its own links will form the third layer and so on.

- We are using a queue to implement this graph.

- This is used in the process to list all the external and internal links of the csu domain and we use adjacency matrix and **networkx** library to find the diameter of the underlying graph.

1. **Project’s main functions code.**

To crawl through the source code of every webpage, we are using the **beautifulsoup** library.

Using this library, we extract the source HTML page, then we look and search every valid html link and put it into the queue. Then to process each new node, we pop out the node(web page link), find its own links and then put it back into the queue. Thereby breaking down each page link into its sub links and processing each one of them.

***# this function extracts all the links in BFS manner***

def csu\_links(self, url):

*# set to keep track of visited links*

self.visitedLinks.add(url)

try:

source\_code = requests.get(url)

soup = BeautifulSoup(source\_code.text, features="html.parser")

except requests.exceptions.RequestException as e:

print("Exception occured while crawling website. Few links might not be formatted properly.",e)

sys.exit(1)

if len(self.queue) > 99:

return

for link in soup.findAll('a'):

myLink = link.get('href')

base\_url = 'https://cs.txstate.edu'

url\_found = ''

*#This is to find only valid links*

if myLink is not None and myLink != '/' and myLink != '#' and 'javascript:' not in myLink.strip() and 'mailto:' not in myLink.strip() and \

'tel:' not in myLink.strip():

if myLink.startswith('/'):

*# create complete url eg. /about\_us => cs.txstate.edu/about\_us*

url\_found = base\_url + myLink

self.intExt(url\_found)

else:

*# skip urls from other domain*

if "cs.txstate.edu" in myLink:

url\_found = myLink

self.intExt(url\_found)

*# check if it exists in queue*

if not self.url\_in\_queue(url\_found):

if len(self.queue) > 99:

return

if url\_found not in self.visitedLinks:

self.queue.append(url\_found)

if self.queue:

current = self.queue.popleft()

if current is None or current == '':

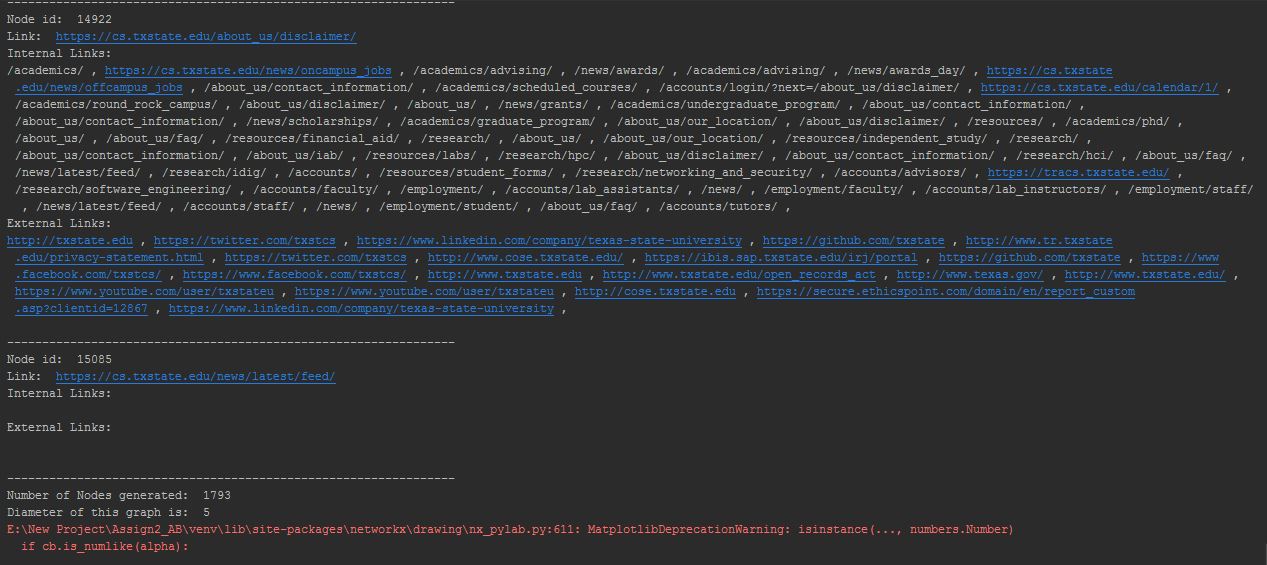
pass

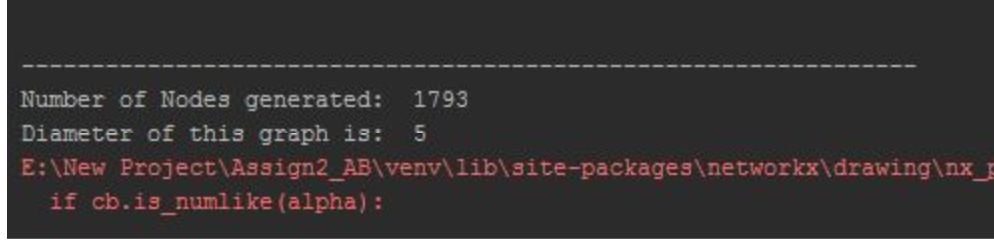
else:

self.csu\_links(current)

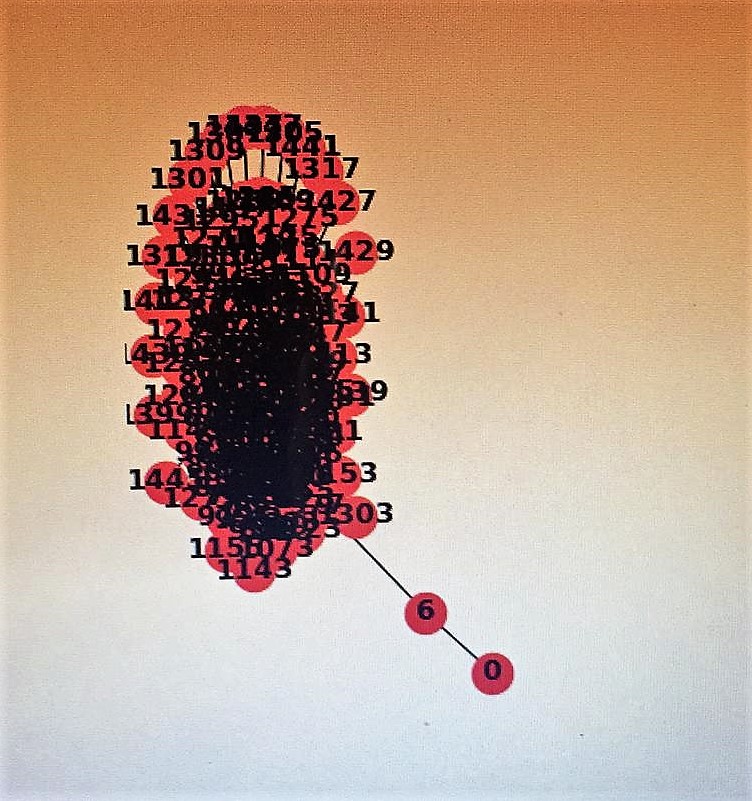
1. **Sample output screenshots.**

**Console output showing diameter and number of nodes and all the links**





**Visual representation of Graph of the cs.txstate.edu domain showing the nodes**



1. **How to run the code**

- Open any Python IDE. Eg. PyCharm

- Load the two files – Graph.py and Node.py (Node.py is the class file)

- Click on terminal and type following three commands:

1. pip install requests

2. pip install bs4

3. pip install networkx

- Run the Graph.py file.

- Make sure Internet connection is active.

7. **References**

a) <https://www.crummy.com/software/BeautifulSoup/>

b) <https://networkx.github.io/>

c) <https://en.wikipedia.org/wiki/Breadth-first_search>

d) <https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9>

e) https://matplotlib.org/api/pyplot\_api.html