**EX 1**

**EX 2 Data Structures used:** Arrays

1. **Proposed Algorithm**
2. First we import the installed libraries urllib, urlopen, BeautifulSoup, re, requests, numpy.
3. Take user input for URL of the webpage and word to be searched along with number of web pages.
4. Initialize variables reqs and soups for retrieving an arbitrary page and hence produce a list of links on that page.
5. Initialize urls as array to store the links.
6. Use a for loop to find all the valid links.
7. Next we use another for loop to find whether the keyword is present in those links or not else print “failure”.

**EX 3 Data Structures used:** Trees (BFS & DFS), Graphs

1. **Proposed Algorithm**
2. First we import the installed libraries.
3. Then, we create two empty sets called internal\_links and external\_links which will store internal and external links separately and ensure that they do not contain duplicates.
4. We then create a method called level\_crawler which takes an input URL and crawls it and displays all the internal and external links using the following steps –

* Define a set called url to temporarily store the URLs.
* Extract the domain name of the url using urlparse library.
* Create a beautifulsoup object using HTML parser.
* Extract all the anchor tags from the beautifulsoup object.
* Get the href tags from the anchor tags and if they are empty, don’t include them.
* Using urljoin method, create the absolute URL.
* Check for the validity of the URL.
* If the url is valid and the domain of the url is not in the href tag and is not in external links set, include it into external links set.
* Else, add it into internal links set if it is not there and print and put it in temporary url set.
* Return the temporary url set which includes the visited internal links. This set will be used later on.

1. If the depth is 0, we print the url as it is. If the depth is 1, we call the level\_crawler method defined above.
2. Else, we perform a breadth first search (BFS) traversal considered the formation of a URL page as tree structure. At the first level we have the input URL. At the next level, we have all the URLs inside the input URL and so on.
3. We create a queue and append the input url into it. We then pop an url and insert all the urls inside it into the queue. We do this until all the urls at a particular level is not parsed. We repeat the process for the number of times same as the input depth.

**(ii) Proposed Algorithm**

1. For each link on the current page, recursively explore it before visiting the remaining links on the page.
2. Use a visited set to keep track of which pages have already been crawled to avoid getting caught in cycles.
3. A "depth" cap allows us to explore all of the links max\_depth pages away from the current one. Hence we use an ‘if’ loop to compare the depth and max\_depth and hence execute the following try and except statements.