CS 467 Graphical Web Crawler

Kerensa Crump - crumpke@oregonstate.edu
Brent Freeman - freeembre@oregonstate.edu
Long Le - lelon@oregonstate.edu

Capstone Project to create a web application that graphically demonstrates two methods of traversing the internet.

Description

The Gudja online web crawler was Created by OSU's team Gudja for the CS 467 Capstone class. In less than 10 weeks, our team successfully built a single page application for a web crawler that allows the user to traverse the internet using two different algorithms: breadth first search (BFS) and depth first search (DFS).

BFS examines all links at each level starting from the user supplied URL. The algorithm expands the search at every level, like an expanding bubble. DFS searches one branch, or unbroken sequence of links, at a time. It begins with the first link and continues until it reaches the desired depth. As an option, the user can tell the crawler to end the search if it encounters a keyword on a web page.

The graphical result is a high level map of all the external pages that were visited by the web crawler. You'll find that BFS is much slower than DFS and each algorithm will give you different information about the connectivity of the internet.

Our graphical web crawler allows users to visually see a web crawling program in action while exploring the internet. The simulation will provide basic insight on how a real life web crawler, such as Googlebot, might visit pages on the web.

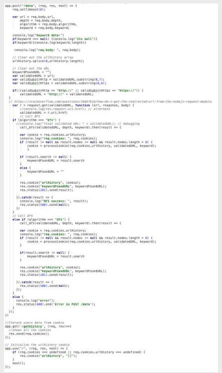
Kerensa Crump
https://github.com/cadelx

Brent Freeman
https://github.com/freeman-bw

Long Le
https://github.com/lelon32

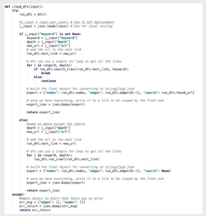
Oregon State University

Nodejs server



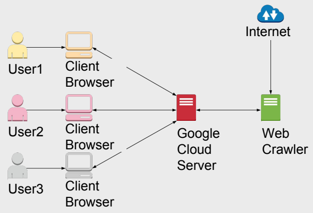
```
app.post("/data", (req, res, next) => {
    req.setTimeout(0);

    var url = req.body.url,
        depth = req.body.depth,
        algorithm = req.body.algorithm,
        keyword = req.body.keyword;

    console.log("keyword data:")
    if(keyword === null) {console.log("its null")}
    if(keyword){console.log(keyword.length)}

     console.log("req.body: ", req.body);

    // Clear out the urlHistory array
    urlHistory.splice(0,urlHistory.length);

    // Clear out the URL
    keywordFoundURL = "";
    var validatedURL = url;
    var validSubstrHttp = validatedURL.substring(0,7);
    var validSubstrHttps = validatedURL.substring(0,8);

    if(!(validSubstrHttp == "http://" || validSubstrHttps == "https://")) {
        validatedURL = "http://" + validatedURL;
  }
  // https://stackoverflow.com/questions/16687618/how-do-i-get-the-redirected-url-from-the-nodejs-request-module
    var r = request.get(validatedURL, function (err, response, body) {
        //console.log(res.request.uri.href); // alternate
        validatedURL = r.uri.href;
        // Call BFS
    if (algorithm === "bfs") {
      //console.log("final validated URL: " + validatedURL); // debugging
        call_bfs(validatedURL, depth, keyword).then(result => {

        var cookie = req.cookies.urlHistory;
        console.log("req.cookies: ", req.cookies);
        if (result != null && result.nodes != null && result.nodes.length > 0) {
            cookie = processCookie(req.cookies.urlHistory, validatedURL, keyword);
            }

        if (result.search != null) {
            keywordFoundURL = result.search
        }
        else {
            keywordFoundURL = ""
        }

        res.cookie("urlHistory", cookie);
        res.cookie("keywordFoundURL", keywordFoundURL);
        res.status(201).send(result);

      }).catch(result => {
        console.log("BFS success: ", result);
        res.status(500).send(null);
      })
    }
    // Call DFS
    else if (algorithm === "dfs") {
        call_dfs(validatedURL, depth, keyword).then(result => {

        var cookie = req.cookies.urlHistory;
        console.log("req.cookies: ", req.cookies);
        if (result != null && result.nodes != null && result.nodes.length > 0) {
            cookie = processCookie(req.cookies.urlHistory, validatedURL, keyword);
        }

        if(result.search != null) {
            keywordFoundURL = result.search
        }

        res.cookie("urlHistory", cookie);
        res.cookie("keywordFoundURL", keywordFoundURL);
        res.status(201).send(result);

      }).catch(result => {
        res.status(500).send(null);
      });
    }
    else {
      console.log("error");
      res.status(400).end('Error in POST /data');
    }
  });
})

//Iterate users data from cookie
app.get('/gethistory', (req, res)=>{
  //shows all the cookies
  res.send(req.cookies);
});

// Initialize the urlHistory cookie
app.use("/", (req, res, next) => {
    if (req.cookies === undefined || req.cookies.urlHistory === undefined) {
        res.cookie("urlHistory", "[]");
    }
    next();
});
```

Cloud_dfs

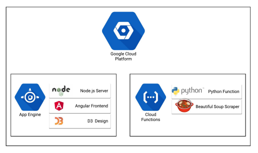


```
def cloud_dfs(input):
    try:
        run_dfs = dfs()

        #j_input = input.get_json() # Use fo GCF deployement
        j_input = json.loads(input) # Use for local testing

        if j_input["keyword"] is not None:
            keyword = j_input["keyword"]
            depth = j_input["depth"]
            new_url = j_input["url"]
            # add the url to the next link
            run_dfs.next_link = new_url

            # dfs can use a simple for loop to get all the links
            for i in range(0, depth):
                if run_dfs.search_crawl(run_dfs.next_link, keyword):
                    break
                else:
                    continue

            # build the final object for converting to stringified json
            export = {"nodes": run_dfs.nodes, "edges": run_dfs.edges[0:-1], "search": run_dfs.found_url}

            # once we have everything, write it to a file to be viewed by the front end
            export_json = json.dumps(export)

            return export_json

        else:
            #same as above except the search
            depth = j_input["depth"]
            new_url = j_input["url"]

            # add the url to the next link
            run_dfs.next_link = new_url

            # dfs can use a simple for loop to get all the links
            for i in range(0, depth):
                run_dfs.run_crawl(run_dfs.next_link)

            # build the final object for converting to stringified json
            export = {"nodes": run_dfs.nodes, "edges": run_dfs.edges[0:-1], "search": None}

            # once we have everything, write it to a file to be viewed by the front end
            export_json = json.dumps(export)

            return export_json
    except:
        #empty object to alert that there was an error
        err_msg = {"edges": [], "nodes": []}
        err_return = json.dumps(err_msg)
        return err_return
```

Client-Server Architecture



Hosted App Architecture



D3 Graphical Features

- Web page title - tooltip
- Web page URL - tooltip
- Active link - node onClick
- Domain name - node color
- Links between web pages
- Directionality of the link --->
- Node connectivity - diameter
- Keyword found - highlight

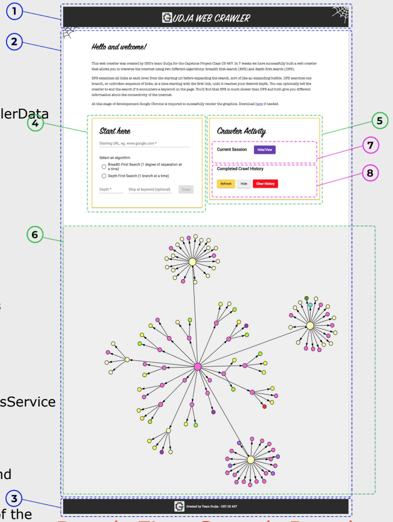
Front-End Architecture

Angular Data Models
Post, CookieData, CrawlerData

Angular Components
1. Header
2. Home
3. Footer
4. Post-Create
5. Activity
6. Crawler
7. Post-Display
8. Cookie

Angular Service Calls
1. None
2. None
3. None
4. PostsService
5. None
6. CrawlerService, PostsService
7. PostsService
8. CookieDataService

GUI & Breadth First Search


1
2
3
4
5
6
7
8
GUDJA WEB CRAWLER
Start here
Starting URL, eg. www.google.com *
Select an algorithm
Breadth First Search (1 degree of separation at a time)
Depth First Search (1 branch at a time)
Depth *
Stop at keyword (optional)
Crawl
Crawler Activity
Current Session
Hide/View
Completed Crawl History
Refresh
Hide
Clear History
Created by Team Gudja - OSU CS 467

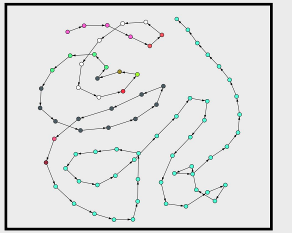
Hello and welcome!

This web crawler was created by OSU's team Gudja for the Capstone Project Class CS 467. In 7 weeks we have successfully built a web crawler that allows you to traverse the internet using two different algorithms: breadth first search (BFS) and depth first search (DFS).

BFS examines all links at each level from the starting url before expanding the search, sort of like an expanding bubble. DFS searches one branch, or unbroken sequence of links, at a time starting with the first link, until it reaches your desired depth. You can optionally tell the crawler to end the search if it encounters a keyword on the page. You'll find that BFS is much slower than DFS and both give you different information about the connectivity of the internet.

At this stage of developement Google Chrome is required to sucessfully render the graphics. Download here if needed.

Depth First Search Result



Technologies Used

Angular - Angular provided the framework to create a responsive single page application. Angular's MVC structure utilized models, components, and services along with observables to implement two way data binding and reducing coupling. **See Front-End Architecture, center right**

D3- D3 is a JavaScript library that uses native HTML, CSS, and Javascript with data binding to turn create a wide array of visualizations. The core of the network graph visualization used the forceSimulation module. Features like the tooltip, dynamic svg sizing animation, and pulsing highlight were managed separately using custom functions tied to the events like mouseover and simulation tick. **See D3 Graphical Features, center**

Node.js - The backbone of or app, Node is both familiar to the whole team and also simple to use. Node Packages: Express, Cors, Request-Promise, Body-Parser, Cookie-Parser. **See Node.js server, left**

Python 3.x - Python was chosen to develop the web crawler programs because it was an opportunity to learn something new. There are many positives to using Python including: clean code, higher levels of abstraction, fast development speed, great documentation, and a large number of useful libraries. Python Libraries: Beautiful Soup 4, lxml, NumPy.

Google Cloud Functions - Google's answer to serverless technology. We created our individual functions written out as Python methods, Google hosts these functions, and we can access them with a http request. **See Cloud_dfs, left.**

Google App Engine - Google App Engine serves as the host for our application. This platform allows for a relatively easy method to share our application with the public with less worry about maintenance. It supports different languages giving us freedom to develop in any languages.

Slack - The go-to communication platform. Slack was instrumental in allowing us to share our progress, ideas and frustrations and fixes.

Git and GitHub - GitHub was used for version control during the development of the Gudja Web Crawler. It was particularly useful for managing four different compiled front ends needed for our different testing/deployment environments