Merin Mundt

311 PA1 Report

Pseudocode:

```
crawl() {
    My graph webgraph;
    Queue is linked list of type UrlAndDepth
    List discovered
    int pagecounter; webhitcounter;

    while (Q is not empty & < maxPages
        - get first element in Q
        - check for maxDepth
        - check for 50 searches
            ↳ webcounter == 50
            ↳ if 50 → wait 3 seconds
        - search first element if valid
            - ++webhitcounter
            - for all links (less than numPages)

                - if (not in discovered)
                - update Queue
                - update discovered
                - ++pagecounter
                - add edge to webgraph

            - remove element from Q
    - return webgraph
```

makeIndex()
- webhits = 0
- for (every entry URL in URL map)
  - new hashmap wordmap
  - check for 50 webhits
    ↳ if so wait 3 seconds
  - Jsoup on body
  - scanner on body
  - webhits ++
  - while (scanner.hasnext)
    check for punctuation & stop word (scanner.next())
    ↳ then put word in wordmap
  - close scanner
  - ~~for~~ put wordmap into index map hashmap (hashmap of hashmaps)

---

search()
- Array list list of type URL and Ranking
- for (every entry in index map)
  - if (word w is in entry)
    - ranking = indegree + #occurrences
    - if (ranking > 0)
      - add to list;
- sort list

search cont.

→ Arraylist of type Tagged vertex

- create finalist of url & rankings from list.
- return finallist.

Search With And()

- Arraylist list of type Url And ranking
- for(every entry in _indexmap)
  - if (both words, (w_1 & w_2) are in entry)
    - ranking = indegree * (# of w_1 occurrences + # of w_2 occurrences)
    - if (ranking > 0)
      - add to list
- sort list
- create finallist Array list of type Tagged vertex
- include entry urls & rankings from list in finallist
- return finallist

Search with Or()

- Arraylist of type url and ranking called list
- for(every entry in _indexmap)
  - if (either (w_1 or w_2) is in entry) at least one
    - ranking = indegree * (w_1 occurrence & w_2 occurrences)
    - if (ranking > 0)
      - add to list

adjust for 0 value

- sort list
- Arraylist finallist of type Tagged vertex
- include entry url & corresponding ranking
- return finallist

searchWithNot()
- Arraylist list of type UrlAndRanking
- for (each entry in -indexmap)
  - if (w1 is in entry && w2 is not in entry)
    - ranking = inDegree + # of w1 occurrences
    - if (ranking > 0)
      - add to list
- sort list
- Arraylist finallist of type ~~tag~~ ranked retrieved
- include entry Urls & rankings from list in finallist
- return finallist

Choice of data structures:

for my Graph:

I decided to use a hashmap so I could store the URL's & their corresponding links. The URL's are the keys. So crawler implemented this hashmap & added to it.

for the Indexing, I chose to use 2 hashmaps 1 hashmap holds the wordmap, which is a hashmap of words. & the next hashmap holds the URL's & their corresponding word hashmaps.

Runtimes of algorithms

crawl(): O(maxPages * maxdepth)
// dependend on Jsoup
makeIndex(): O(maxPages * maxDepth * sizes of url)
// depends on Jsoup
search(s): O(2^numUrl's)) = O(numUrls)
searchWithAnd(): O(numUrls)
searchWithOr(): O(numUrls)
searchWithNot(): O(numURLs)
↑
#ofUrls in entry list

addVertex(): O(1)
addEdge(): O(1)
vertexData(): O(1)
VertexDataWithIncomingEdges(): O(VertexDataSize)
getNeighbors(): O(Url neighbors)
↑
how many neighbors it has
getIncoming(): O(size of map)