ComS 311
Recitation 3, 2:00 Monday
Project 1

Sean Gordon

November 14, 2019

Algorithm 1 Pseudocode for crawl().

```
#Run the proposed BFS with modifications:
#I used a null sentinel to mark the end of a level/depth
Add seed to queue
while Queue isn't empty && we haven't reached max depth do
    Grab the next link
    If we have reached the end of this depth/level, continue

    if the link is not in the graph then
        If we have reached max # of unique pages, skip it

        Grab all links contained in the current link's webpage
        for Every link l returned do
            Add l to the queue
        end for
    else Grab the link from the graph
    end if
    Update the incoming and outgoing edges of each link
end while
Return the completed graph
```

Runtime of algorithm:

Assuming there are n total vertices and m total edges in our 'internet'...
Assuming our depth and max pages = ∞ ...

Loop through all links in the queue = O(n)
Grab all links within a link = O(m)
Add link to queue if new = O(1)

Runtime = O(n*m)

Algorithm 2 Pseudocode for makeIndex().

```
for For every vertex in our graph do
    Grab the vertex data (url and indegree)
    Grab all words from the page and throw them in a hashmap with their frequency
    for Every word we have recieved do
        Calculate the weight (freq * indegree)

        If the word isn't already in the index, add an empty SortedList
        #SortedList is an extension of ArrayList
        #It adds new items in decreasing order using binary search

        Add the data to the index with the word as the key
    end for
end for
```

Runtime of algorithm:
Assuming there are n total vertices in our graph...
Assuming there are m total words in our graph (duplicates included)...

Loop through all vertices = O(n)
Get all words in a webpage = O(m)
Loop through all words = O(m)
Add a word to SortedList = O(log(m))

Runtime = O(n*2m*log(m)) = O(n*m*log(m))

Algorithm 3 Pseudocode for search().

```
Grab list of TaggedVertices for a word

#The SortedLists used in the crawler ensure the pages are ordered
#in decreasing order by weight(freq*indegree) already
```

Runtime of algorithm:
Grab list = O(1)
Runtime = O(1)

Algorithm 4 Pseudocode for searchWithAnd().

```
Grab list of all TaggedVertices for w1, name it pageList1
Grab list of all TaggedVertices for w2, name it pageList2

#Allows finding weight for url in pageList2 take O(1) time
for Each TaggedVertex v in pageList2 do
    Add v to hashmap mapList2 with the url as the key
end for
Make a SortedList searchResults
#SortedList is an extension of ArrayList
#It adds new items in decreasing order using binary search

for All urls in pageList1 do
    If mapList2 doesn't contain the url, continue

    Add both url weights together (weight = freq*indegree)

    Make TaggedVertex v with the url and the new conbined weight
    Add it to the search results
end for
return searchResults
```

Runtime of algorithm:
Assuming there are n total words in our index...

Loop through pageList2 = O(n)
Loop through pageList1 = O(n)
Runtime = O(2n) = O(n)

Algorithm 5 Pseudocode for searchWithOr().

```
Grab list of all TaggedVertices for w1, name it pageList1
Grab list of all TaggedVertices for w2, name it pageList2

#Allows finding weight for url in pageList2 take O(1) time
for Each TaggedVertex v in pageList2 do
    Add v to hashmap mapList2 with the url as the key
end for
Make a SortedList searchResults
#SortedList is an extension of ArrayList
#It adds new items in decreasing order using binary search

for All urls in pageList1 do
    If mapList2 doesn't contain the url, continue

    *Remove this url from mapList2

    Add both url weights together (weight = freq*indegree)

    Make TaggedVertex v with the url and the new conbined weight
    Add it to the search results
end for
for Everything left in mapList2 do
    Make TaggedVertex v with the url and weight
    Add v to searchResults
end for
return searchResults
```

Runtime of algorithm:
Assuming there are n total words in our index...

Loop through pageList2 = O(n)
Loop through pageList1 = O(n)
Loop through what remains of pageLis2 = O(n)

Runtime = O(3n) = O(n)

Algorithm 6 Pseudocode for searchWithNot().

```
#Near identical to searchWithAnd, but skips url if it is in the hashmap

Grab list of all TaggedVertices for w1, name it pageList1
Grab list of all TaggedVertices for w2, name it pageList2

#Allows finding weight for url in pageList2 take O(1) time
for Each TaggedVertex v in pageList2 do
    Add v to hashmap mapList2 with the url as the key
end for
Make a SortedList searchResults
#SortedList is an extension of ArrayList
#It adds new items in decreasing order using binary search

for All urls in pageList1 do
    If mapList2 does contain the url, continue

    Add both url weights together (weight = freq*indegree)

    Make TaggedVertex v with the url and the new conbined weight
    Add it to the search results
end for
return searchResults
```

Runtime of algorithm:
Assuming there are n total words in our index...

Loop through pageList2 = O(n)
Loop through pageList1 = O(n)
Runtime = O(2n) = O(n)