SearchEngine

---algoritmi---

Aleksandar Stevanović RA176/2017

Jovan Bosnić RA173/2017

Search

```
def search(search_string):
    words = validate(search_string)
    result_sets = {}
    ret_words = []
    for word in words:
        if not is_operator(word):
            result_sets[word] = trie.find(word)
            ret_words.append(word)
    result = Set()
    operator = None
    for word in words:
        if is_operator(word):
            operator = word
        else:
            if len(result_sets[word]) > 0:
                if operator == 'and':
                    result = result.__and__(result_sets[word])
                elif operator == 'not':
                    result = result.__not__(result_sets[word])
                elif operator == 'or':
                    result = result.__or__(result_sets[word])
                elif operator is None:
                    result = result_sets[word]
    return result, ret_words
```

Prosleđujemo string sa konzole i prvo ga prosledimo funkciji validate da proveri da li je unos validan. Iteriramo kroz reči liste koju je validate vratio i ako reč nije operator pozivamo trie.find, ako jeste operator onda odradimo logičku operaciju koja je prosleđena. Search vraća "result" odnosno set, spisak html stranica na kojma je pronađeno ono što je traženo i "ret_words" odnosno reči po kojima kasnije od trie.getCounters dobijamo broj pojavljivanja tih reči.

Validate

```
def validate(user_input):
    input_list = user_input.split()
    check = -1
    valid = False
    len1 = len(input_list)
    string_list = []
    if len1 > 0:
        valid = True
        i = 0
        for word in input_list:
            i = i + 1
            if is_operator(word):
                if i == len1:
                    valid = False
                    break
                else:
                    if check == -1:
                        valid = False
                        break
                    elif check == 0:
                        string_list.append(word)
                        check = 1
                    else:
                        valid = False
                        break
            else:
                if check == 1 or check == -1:
                    string_list.append(word)
                    check = 0
                else:
                    string_list.append('or')
                    string_list.append(word)
                    check = 0
    if valid:
        return string_list
```

Prosleđujemo string koji je unos za pretragu sa konzole, koji delimo po razmacima i dobijamo listu strignova. "check" služi da proverimo da li je prethodno unesena reč ili operator, da ne bi unosili dva operatora jedan za drugim, postavimo ga na –1 na pocetku, 0 za rec, 1 za operator. "valid" stavljamo na false za slučaj da ne uđe u prvi if, jer unos nije validan ako je prazan string prosleđen, ako je bilo šta drugo prosleđeno u pretragu "valid" se stavlja na true i vršimo provere. Proveravamo da li je prva ili poslednja reč operator ako jeste znači da unos nije validan, sve ostalo je validno. "string_list" punimo rečima i na svaku reč pretrage po defaultu dodajemo or da bi pretraga bila kompletna. Ukoliko je unos validan vraćamo "string_list" na osnovu koga vršimo pretragu.

Rang

```python
def rang(graph, trie, result, words):
    rang_result = {}

    for html in result:
        dolazeci = graph.odlazeci[html]
        number_links = len(dolazeci)
        number_words_in_links = 0
        for link in dolazeci:
            if link in result:
                for word in words:
                    try:
                        number_words_in_links += trie.get_counters(word)[link]
                    except KeyError:
                        continue

        number_words = 0
        for word in words:

            try:
                number_words += trie.get_counters(word)[html]
            except KeyError:
                continue

        rang_result[html] = int(number_words + number_links * 0.7 + number_words_in_links * 0.5)

    return rang_result
```

Na rangiranje rezultujućih stranica pretrage utiče: broj pojavljivanja traženih reči na njoj, broj linkova iz drugih stranica na pronađenu stranicu i broj traženih reči u stranicama koje sadrže link na traženu stranicu. "rang_result" je rečnik koji za ključ ima html stranicu a vrednost mu je njen rang. "result" je rečnik koji za ključ ima html stranicu a vrednost mu je broj pojavljivanja reči u toj html stranici."dolazeci" sadrži sve linkove koji pokazuju na traženu html stranicu. "number_links" je broj linkova koji se nalaze u listi "dolazeci"."number_words_in_links" je broj pojavljivanja traženih reči u stranicama koje sadrže link na traženu stranicu. Prolazimo kroz sve stranice koje pokazuju na traženu i brojimo koliko puta se tražene reči pojavljuju u njima i čuvamo tu vrednost u "number_words_in_links". "number_words" inicijalizujemo na 0 i tu sačuvamo broj koliko puta se tražene reči pojavljuju u traženoj stranici. Na kraju primenimo formulu za rang: (broj reči na stranici) + (broj linkova koji pokazuju na stranicu)*0.7+(broj reči u stranicama koje pokazuju na traženu)*0.5.

Sort

```
def partition(result, low, high):
    i = low-1

    pivot = result[high]

    for j in range(low, high):
        element = result[j]

        if element[1] > pivot[1]:
            i = i + 1
            result[i], result[j] = result[j], result[i]

    result[i+1], result[high] = result[high], result[i+1]

    return i+1


def sort(result, low, high):
    if low < high:
        pi = partition(result,low, high)
        sort(result, low, pi-1)
        sort(result, pi+1, high)
```

Sortiranje rezultata, odrađen je Quick sort. Sledi opis rada algoritma koji elemente sortira u rastućem poretku. Osnovni princip rada algoritma se deli u tri sledeće celine:

1. Izabiranje pivot-elementa na datom intervalu
2. Raspored svih elemenata manjih ili jednakih ovom pivot-elementu levo od njega, a svih većih desno od njega u nizu
3. Rekurzivno ponavljanje ovog postupka na novonastale intervale levo i desno od ovog pivot-elementa.