Department of Computer Science - Τμήμα Πληροφορικής
University of Cyprus - Πανεπιστήμιο Κύπρου

Εργαστήρια ΕΠΛ 231
Χειμερινό 2018

Εργασία Αρ. 1
Ημερομηνία έκδοσης: 20/9/2018
Ημερομηνία Παράδοσης: 4/10/2018

Ονοματεπώνυμο: Σωκράτης Γιαννακού
Αριθμός Ταυτότητας: 913240

Υπεύθυνος Εργαστηρίου
Πέτρος Παναγή

ΣΥΝΤΟΜΗ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΛΥΣΗΣ ΜΕ ΑΝΑΦΟΡΕΣ ΣΤΟΝ ΠΗΓΑΙΟ ΚΩΔΙΚΑ
(10%)

Υλοποιώ 3 κλάσεις: 1)WebCrawler.java (main)
2)Url.java
3)BackQueue.java

```java
public class Url {
    private String url;
    private int priority;
    private String domain;

    public Url(String u) {
        url=u;
        domain=url.substring(0,url.indexOf('/'));
    }
```

```java
public class BackQueue extends LinkedList implements Queue{

    private String domain;
    private int timer;

    public BackQueue(String d) {
        domain=d;
        // timer between 3 and 10
        timer=(int)((Math.random()*7)+3);
    }
```

Αρχικά δημιουργείται Set με τα Top 20 Domains τα οποία διαβάζονται από αρχείο txt.

```java
HashSet<String> topTwenty = new HashSet<String>();
//Reads top 20 domains from text
readTop20(args[0], topTwenty);
```

Δημιουργούνται 4 ουρές προτεραιότητας(Front Queues) και ένα ArrayList από Back Queues.

```java
Queue<Url> frontOne = new LinkedList<>();
Queue<Url> frontTwo = new LinkedList<>();
Queue<Url> frontThree = new LinkedList<>();
Queue<Url> frontFour = new LinkedList<>();

//List of all the back queues, one unique back queue for each domain
ArrayList<BackQueue> qlist=new ArrayList<>();
```

Σε κάθε κύκλο του προσομοιωτή διαβάζονται 5 Urls από αρχείο txt, τα οποία τοποθετούνται στα Front Queues.

```java
// New URL from txt
Url item = new Url(line);

//If URL is from top 20 domain, gets priority 1 else random priority from 2-4
if (topTwenty.contains(item.getDomain())) {
    item.setPriority(1);
} else {
    item.setPriority(0);
}

// URL is placed in front queue based on priority
switch (item.getPriority()) {
case 1:
    frontOne.add(item);
    break;
case 2:
    frontTwo.add(item);
    break;
case 3:
    frontThree.add(item);
    break;
case 4:
    frontFour.add(item);
}
```

Μετά δημιουργείται ένα Back Queue για κάθε Url αν δεν υπάρχει ήδη.

```
BackQueue bq=new BackQueue(item.getDomain());
boolean flag=true;
//Checking if back queue already exists
for (BackQueue b:qlist) {
    if (b.getDomain().equals(bq.getDomain())) {
        flag=false;
    }

}
//Adding new Back Queue
if (flag) {
    qlist.add(bq);
}
```

Έπειτα με τη χρήση του "D Queue", ψάχνει για κάποιο Back Queue που είναι έτοιμο για επεξεργασία, βάση της πολιτικής καλής συμπεριφοράς.
Αν το Back Queue είναι άδειο τότε εφαρμόζεται αλγόριθμος επαναγέμισης όλων των Back Queues.

```
while (!frontOne.isEmpty() && num!=0) {
    Url y=frontOne.poll();

    //Find the Back Queue that matches the domain
    for (BackQueue bb:qlist) {
        if (bb.getDomain().equals(y.getDomain())) {
            bb.push(y);
            break;
        }
    }
    num--;
}
```

Όταν το διαθέσιμο Back Queue έχει πλέον στοιχεία, βγάζουμε το πρώτο στοιχείο (Dequeue) για επεξεργασία και μετά επανατοποθετείται στο Front Queue που ανήκει βάση της προτεραιότητας του. Ο κύκλος τερματίζει και παρουσιάζονται αποτελέσματα.

```
//Extraction of URL if Back Queue is not empty, URL to be placed in the correct Front Queue
if (!b.isEmpty()) {

    Url x=(Url) b.poll();

    System.out.println("\nWorking on URL: "+x.getURL());


    if (x.getPriority()==1)
        frontOne.add(x);
    else if (x.getPriority()==2)
        frontTwo.add(x);
    else if (x.getPriority()==3)
        frontThree.add(x);
    else
        frontFour.add(x);
}
```

ΠΡΟΓΡΑΜΜΑ JAVA: (60%) (Courier New 12) (Όχι βασικές Πράξεις)

Java CODE

```
public class WebCrawler {

public static void main(String[] args) throws IOException {

     //Set of the top 20 domains
HashSet<String> topTwenty = new HashSet<String>();
//Reads top 20 domains from text
readTop20(args[0], topTwenty);

     int n=1;

     // 4 priority front queues
     Queue<Url> frontOne = new LinkedList<>();
     Queue<Url> frontTwo = new LinkedList<>();
     Queue<Url> frontThree = new LinkedList<>();
     Queue<Url> frontFour = new LinkedList<>();

     //List of all the back queues, one unique back queue for each
domain
     ArrayList<BackQueue> qlist=new ArrayList<>();

     //max number of cycles
     int simulationTime = 30;
     //number of current cycle
     int time = 1;
//Number of domains read per cycle from text file
     int urlCount = 5;
     int j=0;

     while (time <=simulationTime) {
          int i=0;

          while (((line = br.readLine()) != null) && (i<urlCount)){

               j++;
               // New URL from txt
               Url item = new Url(line);

               //If URL is from top 20 domain, gets priority 1 else
random priority from 2-4
               if (topTwenty.contains(item.getDomain())) {
                    item.setPriority(1);
               } else {
                    item.setPriority(0);
               }
```

```
        // URL is placed in front queue based on priority
        switch (item.getPriority()) {
         case 1:
                frontOne.add(item);
                break;
         case 2:
                frontTwo.add(item);
                break;
         case 3:
                frontThree.add(item);
                break;
         case 4:
                frontFour.add(item);
        }

        BackQueue bq=new BackQueue(item.getDomain());
        boolean flag=true;

        //Checking if back queue already exists
        for (BackQueue b:qlist) {

             if (b.getDomain().equals(bq.getDomain())) {
                          flag=false;
             }
        }

        //Adding new Back Queue
        if (flag) {
             qlist.add(bq);
        }
        i++;
        if (i==urlCount) {
             break;
        }
   }

   //Reducing the timer for every Back Queue
   for (BackQueue b:qlist) {
        b.reduceTimer();
   }

   // "D Queue selects a Back Queue to be serviced"
   for (BackQueue b:qlist) {
        if (b.isReady()) {
             boolean   condition=true;
             while(b.isEmpty()) {
                  int num=10;
                  //
                  //Filling Back queues from front Queue 1
               while (!frontOne.isEmpty() && num!=0) {
                  Url y=frontOne.poll();
```

```
                    //Find the Back Queue that matches the domain
                    for (BackQueue bb:qlist) {
                         if (bb.getDomain().equals(y.getDomain())){
                              bb.push(y);
                              break;
                         }
                    }
                    num--;
               }

                num=8;
               //Filling Back queues from front Queue 2
               while (!frontTwo.isEmpty() && num!=0) {
                    Url y=frontTwo.poll();

                    //Find the Back Queue that matches the domain
                    for (BackQueue bb:qlist) {
                         if (bb.getDomain().equals(y.getDomain())){
                              bb.push(y);
                              break;
                         }
                    }
                    num--;
               }

                num=6;
               //Filling Back queues from front Queue 3
               while (!frontThree.isEmpty() && num!=0) {
                    Url y=frontThree.poll();

                    //Find the Back Queue that matches the domain
for (BackQueue bb:qlist) {
                         if (bb.getDomain().equals(y.getDomain()){
                              bb.push(y);
                              break;
                         }
                    }
                    num--;
               }

               num=4;
               //Filling Back queues from front Queue 4
               while (!frontFour.isEmpty() && num!=0) {
                    Url y=frontFour.poll();

                    //Find the Back Queue that matches the domain
                    for (BackQueue bb:qlist) {
                         if (bb.getDomain().equals(y.getDomain()){
                              bb.push(y);
                              break;
```

```
}
                              }
                              num--;
                         }
                    }
                    //Extraction of URL if Back Queue is not empty, URL to be
placed in the correct Front Queue
                    if (!b.isEmpty()) {

                         Url x=(Url) b.poll();

                         System.out.println("\nWorking on URL: "+x.getURL());


                         if (x.getPriority()==1)
                              frontOne.add(x);
                         else if (x.getPriority()==2)
                              frontTwo.add(x);
                         else if (x.getPriority()==3)
                              frontThree.add(x);
                         else
                              frontFour.add(x);
                         }

                    else {
                         System.out.println("\nRefilling Front queues ...");
                         }
                    break;
}
}
      time++;
      }
   }
 }
}
```

<u>Ενδεικτικά</u> αποτελέσματα εκτέλεσης του προγράμματος σας (Περιεκτικά) για να καταδείξετε την ορθότητα του προγράμματος σας. (20%)

Κύκλοι 3, 4 και 5:

```
Cycle 3:

        >Filling the front queues from txt file if possible . . .

        URL11:  adobe.com/info.html
        URL12:  facebook.com/index.html
        URL13:  craigslist.org/info.html
        URL14:  littlethings.com/info.html
        URL15:  wayfair.com/info.html

Front Queue 1 contains: ebay.com | facebook.com |
Front Queue 2 contains: go.com | merriam-webster.com | twentytwowords.com | usmagazine.com | littlethings.com |
Front Queue 3 contains: rollingstone.com | politico.com | lifehacker.com | craigslist.org |
Front Queue 4 contains: goodreads.com | giphy.com | adobe.com | wayfair.com |

Back Queue with domain rollingstone.com is ready to be serviced . . .

Filling Back Queues with URLs . . .

Working on URL: rollingstone.com/content.html

Back Queue 1 contains: go.com |
Back Queue 2 is empty
Back Queue 3 contains: politico.com |
Back Queue 4 contains: merriam-webster.com |
Back Queue 5 contains: goodreads.com |
Back Queue 6 contains: twentytwowords.com |
Back Queue 7 contains: usmagazine.com |
Back Queue 8 contains: ebay.com |
Back Queue 9 contains: lifehacker.com |
Back Queue 10 contains: giphy.com |
Back Queue 11 contains: adobe.com |
Back Queue 12 contains: facebook.com |
Back Queue 13 contains: craigslist.org |
Back Queue 14 contains: littlethings.com |
Back Queue 15 contains: wayfair.com |

Cycle 4:

        >Filling the front queues from txt file if possible . . .

        URL16:  stackexchange.com/en/index.html
        URL17:  comcast.net/el/index.html
        URL18:  pinterest.com/content.html
        URL19:  ups.com/en/index.html
        URL20:  imdb.com/el/index.html

Front Queue 1 contains: pinterest.com |
Front Queue 2 contains: ups.com |
Front Queue 3 contains: rollingstone.com | comcast.net | imdb.com |
Front Queue 4 contains: stackexchange.com |

Back Queue with domain rollingstone.com is ready to be serviced . . .

Filling Back Queues with URLs . . .
```

```
Cycle 5:

       >Filling the front queues from txt file if possible . . .

       URL21:  netflix.com/contact.html
       URL22:  officedepot.com/el/index.html
       URL23:  usmagazine.com/index.html
       URL24:  about.com/en/index.html
       URL25:  yelp.com/info.html

Front Queue 1 contains: netflix.com | yelp.com |
Front Queue 2 contains: officedepot.com |
Front Queue 3 contains: rollingstone.com | usmagazine.com | about.com |
Front Queue 4 is empty

Back Queue with domain rollingstone.com is ready to be serviced . . .

Filling Back Queues with URLs . . .

Working on URL: rollingstone.com/content.html

Back Queue 1 contains: go.com |
Back Queue 2 is empty
Back Queue 3 contains: politico.com |
Back Queue 4 contains: merriam-webster.com |
Back Queue 5 contains: goodreads.com |
Back Queue 6 contains: twentytwowords.com |
Back Queue 7 contains: usmagazine.com | usmagazine.com |
Back Queue 8 contains: ebay.com |
Back Queue 9 contains: lifehacker.com |
Back Queue 10 contains: giphy.com |
Back Queue 11 contains: adobe.com |
Back Queue 12 contains: facebook.com |
Back Queue 13 contains: craigslist.org |
Back Queue 14 contains: littlethings.com |
Back Queue 15 contains: wayfair.com |
Back Queue 16 contains: stackexchange.com |
Back Queue 17 contains: comcast.net |
```

```
Back Queue 18 contains: pinterest.com |
Back Queue 19 contains: ups.com |
Back Queue 20 contains: imdb.com |
Back Queue 21 contains: netflix.com |
Back Queue 22 contains: officedepot.com |
Back Queue 23 contains: about.com |
Back Queue 24 contains: yelp.com |
```

<u>Ενδεικτικά</u> αποτελέσματα εκτέλεσης του προγράμματος σας (Συνοπτικά) για να καταδείξετε την ορθότητα του προγράμματος σας. Για τα ίδια Cycles όπως πιο πάνω.(10%)

Cycle:	Number of new URLs:	Processed URL:	Waiting Time:
3	5	-	-
4	5	politico.com/index.html	3
5	5	go.com/en/index.html	2