**Федеральное агентство связи**

**Ордена трудового Красного Знамени федеральное государственное бюджетное**

**образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математическая кибернетика и информационные технологии

Лабораторная работа № 8

Выполнил: студент группы БФИ2002

Ковачев В.Е.

Проверила:

Москва, 2021

**Задание:**

В данной лабораторной работе вы расширите и измените разработанную ранее программу:

1. Реализуйте класс с именем URLPool, который будет хранить список всех URL-адресов для поиска, а также относительный "уровень" каждого из этих URL-адресов (также известный как "глубина поиска"). Первый URL-адрес, который нужно будет найти, будет иметь глубину поиска равную 0, URLадреса, найденные на этой странице, будут иметь глубину поиска равную 1 и т.д. Необходимо сохранить URL-адреса и их глубину поиска вместе, как экземпляры класса с именем URLDepthPair, как это было сделано в прошлой лабораторной работе. LinkedList рекомендуется использовать для хранения элементов, так как это поможет эффективно выполнить необходимые операции.

У пользователя класса URLPool должен быть способ получения пары URL-глубина из пула и удаления этой пары из списка одновременно. Должен также быть способ добавления пары URL-глубина к пулу. Обе эти операции должны быть поточно-ориентированы, так как несколько потоков будут взаимодействовать с URLPool одновременно.

У пула URL не должно быть максимального размера. Для этого нужен список необработанных URL-адресов, список уже отсканированных URLадресов и еще одно поле, о котором будет написано ниже.

2.Чтобы выполнить веб-сканирование в нескольких потоках, необходимо создать класс CrawlerTask, который реализует интерфейс Runnable. Каждый экземпляр CrawlerTask должна иметь ссылку на один экземпляр класса URLPool, который был описан выше. (Обратите внимание на то, что все экземпляры класса CrawlerTask используют единственный пул!) Принцип работы веб-сканера заключается в следующем:

1). Получение пары URL-Depth из пула, ожидая в случае, если пара не будет сразу доступна.

2). Получение веб-страницы по URL-адресу.

3). Поиск на странице других URL-адресов. Для каждого найденного URL-адреса, необходимо добавить новую пару URL-Depth к пулу URL-адресов. Новая пара должна иметь глубину на единицу больше, чем глубина текущего URL-адреса, по которому происходит сканирование.

4). Переход к шагу 1. Данный цикл должен продолжаться до тех пор, пока в пуле не останется пар URL-Depth.

3. Так как веб-сканер будет порождать некоторое количество потоков, измените программу так, чтобы она принимала третий параметр через командную строку, который будет определять количество порождаемых потоков веб-сканера. Функция main должна выполнять следующие задачи:

1). Обработать аргументы командной строки. Сообщить пользователю о любых ошибках ввода.

2). Создать экземпляр пула URL-адресов и поместить указанный пользователем URL-адрес в пул с глубиной 0.

3). Создать указанное пользователем количество задач (и потоков для их выполнения) для веб-сканера. Каждой задаче поискового робота нужно дать ссылку на созданный пул URL-адресов.

4). Ожидать завершения веб-сканирования.

5) Вывести получившийся список URL-адресов, которые были найдены. Синхронизируйте объект пула URL-адресов во всех критических точках, так как теперь код должен быть ориентирован на многопоточность. . Веб-сканер не должен постоянно опрашивать пул URL-адресов в случае, если он пуст. Вместо этого пусть они ожидают в случае, когда нет доступных URL-адресов. Реализуйте это, используя метод wait() внутри «get URL» в случае, если ни один URL-адрес в настоящее время недоступен. Соответственно, метод "add URL" пула URL-адресов должен использовать функцию notify() в случае, когда новый URL-адрес добавлен к пулу.

Обратите внимание на то, что потоки веб-сканера не сами будут выполнять какие-либо из этих операций синхронизации/ожидания/уведомления. По той же причине, что и пул URLадресов скрывает детали того, как URL-адреса хранятся и извлекаются: инкапсуляция! Точно так же, как и в вашей реализации пользователи пула URL-адресов не должны знать о деталях реализации, также они не должны знать о деталях организации потоков.

**Ход работы:**

**Для выполнения лабораторной работы был изменён уже существующий метод Crawler:**

import java.net.MalformedURLException;

import java.util.LinkedList;

public class Crawler

{

public static void showResult(LinkedList<URLDepthPair> resultLink)

{

for (URLDepthPair c : resultLink)

{

System.out.println("Глубина: " + c.getDepth()+"\tСсылка: "+c.getURL());

}

}

public static boolean checkDigit(String line)

{

boolean isDigit = true;

for (int i = 0; i < line.length() && isDigit; i++)

{

isDigit = Character.isDigit(line.charAt(i));

}

return isDigit;

}

public static void main(String[] args) throws MalformedURLException

{

args = new String[]{"http://ito.edu.ru/", "4", "100"};

if (args.length == 3&&checkDigit(args[1])&&checkDigit(args[2]))

{

String lineUrl = args[0];

int numThreads = Integer.parseInt(args[2]);

URLPool pool = new URLPool(Integer.parseInt(args[1]));

pool.addPair(new URLDepthPair(lineUrl, 0));

for (int i = 0; i < numThreads; i++)

{

**//передаём пул адресов в каждый созданный поток**

CrawlerTask c = new CrawlerTask(pool);

Thread t = new Thread(c);

t.start();

}

**//Ожидаем завершения работы всех потоков**

while (pool.getWait() != numThreads)

{

try

{

Thread.sleep(500);

}

catch (InterruptedException e)

{

System.out.println("Игнорирование InterruptedException");

}

}

try

{

showResult(pool.getResult());;

}

catch (NullPointerException e)

{

System.out.println("Не ссылка");

}

System.exit(0);

}

else

{

System.out.println("usage: java Crawler <URL> <maximum\_depth> <num\_threads> or second/third not digit");

}

}

}

**Также был добавлен класс CrawlerTask:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.MalformedURLException;

import java.net.Socket;

import java.net.SocketTimeoutException;

public class CrawlerTask implements Runnable

{

URLPool urlPool;

public static final String URL\_PREFIX = "http://";

public CrawlerTask(URLPool pool)

{

this.urlPool = pool;

}

**//ќтправка http запроса**

public static void request(PrintWriter out,URLDepthPair pair)

{

out.println("GET " + pair.getPath() + " HTTP/1.1");

out.println("Host: " + pair.getHost());

out.println("Connection: close");

out.println();

out.flush();

}

**//метод для посимвольного считывания строки и записи, найденного URL**

public static void buildNewUrl(String str,int depth,URLPool pool) throws MalformedURLException

{

try

{

int end\_of\_link = str.indexOf("\"", str.indexOf(URL\_PREFIX));

if (end\_of\_link == -1 || (str.indexOf("'", str.indexOf(URL\_PREFIX)) != -1 && str.indexOf("'", str.indexOf(URL\_PREFIX)) < end\_of\_link))

{

end\_of\_link = str.indexOf("'", str.indexOf(URL\_PREFIX));

}

if (end\_of\_link == -1 || (str.indexOf("<", str.indexOf(URL\_PREFIX)) - 1 != -1 && str.indexOf("<", str.indexOf(URL\_PREFIX)) - 1 < end\_of\_link))

{

end\_of\_link = str.indexOf("<", str.indexOf(URL\_PREFIX)) - 1;

}

String currentLink = str.substring(str.indexOf(URL\_PREFIX), end\_of\_link);

pool.addPair(new URLDepthPair(currentLink, depth + 1));

}

catch (StringIndexOutOfBoundsException e)

{

}

}

@Override

public void run()

{

while (true)

{

URLDepthPair currentPair = urlPool.getPair();

try

{

Socket my\_socket = new Socket(currentPair.getHost(), 80);

my\_socket.setSoTimeout(1000);

try

{

PrintWriter out = new PrintWriter(my\_socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(my\_socket.getInputStream()));

request(out,currentPair);

String line;

while ((line = in.readLine()) != null)

{

if (line.indexOf(currentPair.URL\_PREFIX)!=-1)

{

buildNewUrl(line,currentPair.getDepth(),urlPool);

}

}

my\_socket.close();

}

catch (SocketTimeoutException e)

{

my\_socket.close();

}

}

catch (IOException e)

{

}

}

}

}

**Был расширен метод URLDepthPair:**

import java.util.LinkedList;

import java.util.regex.Pattern;

import java.net.MalformedURLException;

import java.net.URL;

public class URLDepthPair

{

public static final String URL\_PREFIX = "http://";

public String URL;

private int depth;

URL host\_path;

**//Проверяет ссылку на формат заданный при помощи регулярного выражения URL\_PREFIX**

public static boolean isUrlValid(String url)

{

if (url == null)

return false;

Pattern urlValidationPattern = Pattern.compile(URL\_PREFIX);

return urlValidationPattern.matcher(url).find();

}

**//Принимает на вход пару для хранения**

public URLDepthPair (String URL, int depth) throws MalformedURLException

{

if (!isUrlValid(URL))

{

throw new MalformedURLException();

}

this.URL=URL;

this.depth=depth;

try

{

this.host\_path= new URL(URL);

}

catch (MalformedURLException e)

{

e.printStackTrace();

}

}

public String getHost()

{

return host\_path.getHost();

}

**//Возвращает имя ресурса у хоста**

public String getPath()

{

return host\_path.getPath();

}

public int getDepth()

{

return depth;

}

public String getURL()

{

return URL;

}

**//Возвращает результат проверки ссылки**

public static boolean check(LinkedList<URLDepthPair> resultLink, URLDepthPair pair)

{

boolean isAlready = true;

for (URLDepthPair c : resultLink)

{

if (c.getURL().equals(pair.getURL()))

{

isAlready=false;

}

}

return isAlready;

}

}

**Был создан класс для организации работы потоков URLPool:**

import java.util.LinkedList;

public class URLPool

{

LinkedList<URLDepthPair> findLink;

LinkedList<URLDepthPair> viewedLink;

int maxDepth;

int cWait;

public URLPool(int maxDepth)

{

this.maxDepth = maxDepth;

findLink = new LinkedList<URLDepthPair>();

viewedLink = new LinkedList<URLDepthPair>();

cWait = 0;

}

**//Метод для получения пары для обработки из потока**

public synchronized URLDepthPair getPair()

{

while (findLink.size() == 0)

{

cWait++;

try

{

wait();

}

catch (InterruptedException e)

{

System.out.println("Ignoring InterruptedException");

}

cWait--;

}

URLDepthPair nextPair = findLink.removeFirst();

return nextPair;

}

**//Метод для добавления пары из потока в список просмотренных или найденных ссылок**

public synchronized void addPair(URLDepthPair pair)

{

if (URLDepthPair.check(viewedLink,pair))

{

viewedLink.add(pair);

if (pair.getDepth() < maxDepth)

{

findLink.add(pair);

notify();

}

}

}

public synchronized int getWait()

{

return cWait;

}

public LinkedList<URLDepthPair> getResult()

{

return viewedLink;

}

}

**Результат работы программы**

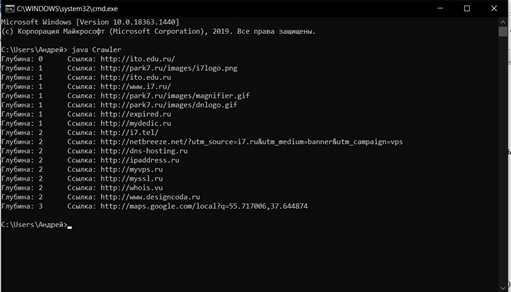


Рисунок №1- Результат работы программы

**Вывод**

В данной лабораторной работе был расширен веб-сканер, для использования поточной обработки Java, что позволило сканировать несколько страниц одновременно.

**Список использованной литературы**

1. Камаев В.А., Костерин В.В. Технологии программирования. М.: Высшая школа,

2006.

2. Жоголев Е.А.Технология программирования. – М.: Научный мир, 2004.