DATA: 18/01/2022 EXP No: 04	Title of the Lab BFS implementation in Web crawling DFS implementation in 6 * 6 Sudoku	Name:Avinash reddy Vasipalli Registration Number: RA1911027010007 Section: N1 Lab Batch: 1 Day Order: 2

4 a - BFS implementation in Web crawling

AIM:

Implementation of Web crawling using Breath first Search

Description of problem:

BFS implementation in web crawling works such that in a given link we have to extract all the links available of that page at that level. These links are sorted as important and unimportant to use case requirement.

Manual Solution

Breadth first search or BFS follows shallow node approach. This will search all the available links at a given level. We start a one point, known as root and searching all neighbour nodes and then move to next level and searching their neighbour nodes again. This is done by a queue and used to carry out the search which saves the nodes and adds them to neighbour nodes into queue and pop the initial node.

Source code

```
from urllib.request import urljoin
from bs4 import BeautifulSoup
import requests
from urllib.request import urlparse


links_intern = set()
input_url = "https://www.linkedin.com/feed/"
depth = 1


links_extern = set()
```

```
def level_crawler(input_url):
    temp_urls = set()
    current_url_domain = urlparse(input_url).netloc


    beautiful_soup_object = BeautifulSoup(
        requests.get(input_url).content, "lxml")

    for anchor in beautiful_soup_object.findAll("a"):
        href = anchor.attrs.get("href")
        if(href != "" or href != None):
            href = urljoin(input_url, href)
            href_parsed = urlparse(href)
            href = href_parsed.scheme
            href += "://"
            href += href_parsed.netloc
            href += href_parsed.path
            final_parsed_href = urlparse(href)
            is_valid = bool(final_parsed_href.scheme) and bool(
                final_parsed_href.netloc)
            if is_valid:
                if current_url_domain not in href and href not in links_extern:
                    print("Extern - {}".format(href))
                    links_extern.add(href)
                if current_url_domain in href and href not in links_intern:
                    print("Intern - {}".format(href))
                    links_intern.add(href)
                    temp_urls.add(href)
    return temp_urls


if(depth == 0):
    print("Intern - {}".format(input_url))

elif(depth == 1):
    level_crawler(input_url)

else:

    queue = []
    queue.append(input_url)
    for j in range(depth):
        for count in range(len(queue)):
            url = queue.pop(0)
            urls = level_crawler(url)
            for i in urls:
                queue.append(i)
```

Output

Intern - https://www.linkedin.com/legal/user-agreement
Intern - https://www.linkedin.com/legal/privacy-policy
Intern - https://www.linkedin.com/legal/cookie-policy
Intern - https://www.linkedin.com/uas/login
Intern - https://www.linkedin.com/help/linkedin/answer/710
Extern - https://about.linkedin.com
Intern - https://www.linkedin.com/accessibility
Intern - https://www.linkedin.com/legal/copyright-policy
Extern - https://brand.linkedin.com/policies
Intern - https://www.linkedin.com/psettings/guest-controls
Intern - https://www.linkedin.com/legal/professional-community-policies

Result:

Successfully implemented Breath first search and done web crawling

4 b - DFS implementation in 6 * 6 Sudoku

AIM:

Implementation of Depth first search in a 6x6 Sudoku

Description of problem:

Sudoku puzzle is grid of n rows and columns filled partially with numbers. The main goal is to fill the grid in the black spaces such that each column and rows have distinct integers. Here we considered a 6x6 grid.

Manual Solution:

Let's assume a number in an empty cell but check if it is safe to assign by checking both roe and column and also the sub grid of 2x3. If it is safe assign the number and check recursively whether the assigned number leads to a solution. If failed trace back to initial stage and re assign and try again with other number.

Source code

```
def print_sudoku(arr):
        for i in range(len(arr)):
                for j in range(len(arr[0])):
                        print(str(arr[i][j])+" ",end="")
                print ('')


def empty_locs(arr, l):
        for row in range(6):
                for col in range(6):
                        if(arr[row][col]== 0):
```

```
                                        l[0]= row
                                        l[1]= col
                                        return True
        return False

def used_in_row(arr, row, num):
        for i in range(6):
                if(arr[row][i] == num):
                        return True
        return False

def used_in_col(arr, col, num):
        for i in range(6):
                if(arr[i][col] == num):
                        return True
        return False

def used_in_box(arr, row, col, num):
        for i in range(2):
                for j in range(3):
                        if(arr[i + row][j + col] == num):
                                return True
        return False

def check_location_is_safe(arr, row, col, num):


        return not used_in_row(arr, row, num) and not used_in_col(arr, col, num) and not used_in_box(arr,
row - row % 2,
                                                        col - col % 3, num)


def solver(arr):


        l =[0, 0]

        if(not empty_locs(arr, l)):
                return True

        row = l[0]
        col = l[1]

        for num in range(1,7):

                if(check_location_is_safe(arr,
                                                        row, col, num)):
```

```
                        arr[row][col]= num

                        if(solver(arr)):
                                return True

                        arr[row][col] = 0

        return False

if _name=="main_":

        grid =[[0 for x in range(6)]for y in range(6)]

        grid = [[0,0,0,1,0,6],
       [6,0,4,0,0,0],
       [1,0,2,0,0,0],
       [0,0,0,5,0,1],
       [0,0,0,6,0,6],
       [5,0,6,0,0,0]]

        if(solver(grid)):
                print_sudoku(grid)
        else:
                print("No solution exists")
```

Output:

```
Sudoku.py ×
E: > AI > Lab 4 > Sudoku.py > ...
70      grid =[[0 for x in range(6)]for y in range(6)]
71
72      grid = [[0,0,0,1,0,6],
73              [6,0,3,0,0,0],
74              [1,0,2,0,4,0],
75              [0,0,0,5,0,1],
76              [0,1,0,6,0,6],
77              [5,0,6,0,0,4]]
78
79      if(solver(grid)):
80          print_sudoku(grid)
81      else:
82          print("No solution exists")

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\AI\Lab 4>  & 'C:\Users\Avinash\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\Avinash\.vscode\extensions\ms-python.python-2022.0.1814523869
\pythonFiles\lib\python\debugpy\launcher' '62610' '--' 'e:\AI\Lab 4\Sudoku.py'
2 3 5 1 4 6
6 1 4 2 3 5
1 5 2 3 6 4
4 6 3 5 2 1
3 4 1 6 5 6
5 2 6 4 1 3
PS E:\AI\Lab 4>  e:; cd 'e:\AI\Lab 4'; & 'C:\Users\Avinash\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\Avinash\.vscode\extensions\ms-python.py
thon-2022.0.1814523869\pythonFiles\lib\python\debugpy\launcher' '62702' '--' 'e:\AI\Lab 4\Sudoku.py'
No solution exists
PS E:\AI\Lab 4>
```

Result:

Successfully Implemented Depth first search in a 6x6 Sudoku