# 全站/大规模网页的爬取与信息采集

前文叙述了单一网页的爬取与内容解析方法，如果现在你需要采集整个网站上的数据，会遇到一些新的问题。例如：

1. 如何遍历网站内所有的内部URL(简称内链），且免于陷入死循环？
2. 如何提高网页获取与解析的速度，以应对大量出现的URL和页面内容？
3. 如何将解析得到的数据进行结构化的存储，以方便后续查阅和处理？
4. ...

那还需要掌握以下技术：

- 网站地图技术
  - 遍历策略
  - URL判重策略
- 多线程技术
- 动态内的获取
- 数据库技术
- 文件操作技术
- 数据清洗技术

## 网站地图技术

网站地图也就是sitemap，是一个网站所有链接的容器。很多网站的链接层次比较深，网站地图可以方便搜索引擎/爬虫抓取网站页面，通过抓取网站地图页面，可以清晰的了解网站的架构。网站地图一般存放在根目录下并命名为sitemap，为搜索引擎引路，增加网站重要内容页面的收录。

通常网站地图不会罗列所有的页面链接，一般会只列出网站最主要的链接，如一级分类，二级分类；或者将网站地图分成几个文件，主网站地图列出通往那次级网站的链接，次级网站地图在列出子层页面链接。

网站地图既可以是html版本的网页，也可以是XML文件。XML版本的网站地图是由goole首先提出的，HTML版本中的sitemap首字母s是小字写的，XML版本中的S则是大写的。XML版本的网站地图由XML标签组成的，文件为UTF-8编码。

例如：

- http://www.pythonclub.org/start?do=index
- http://www.okpython.com/data/sitemap.html

### 发现页面内所有内部链接

内部链接是指向当前域名网站下某个路径的链接，我们可以设计程序进行收集

In [3]:
```
"""最简单的页面链接获取方法"""

from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('https://en.wikipedia.org/wiki/Kevin_Bacon')
```

```
bs = BeautifulSoup(html, 'html.parser')
breakcount = 0
for link in bs.find_all('a'):
    if 'href' in link.attrs:
        print(link.attrs['href'])
        breakcount += 1
    if breakcount > 10:
        print('----------暂停---------')
        break
```

```
/wiki/Wikipedia:Protection_policy#semi
#mw-head
#p-search
/wiki/Kevin_Bacon_(disambiguation)
/wiki/File:Kevin_Bacon_SDCC_2014.jpg
/wiki/Philadelphia
/wiki/Pennsylvania
/wiki/Kyra_Sedgwick
/wiki/Sosie_Bacon
/wiki/Edmund_Bacon_(architect)
/wiki/Michael_Bacon_(musician)
----------暂停---------
```

In [ ]:

```
"""Radom Walk 算法遍历"""

from urllib.request import urlopen
from bs4 import BeautifulSoup
import datetime
import random
import re

links = set()
random.seed(datetime.datetime.now())
def getLinks(articleUrl):
    html = urlopen('http://en.wikipedia.org{}'.format(articleUrl))
    bs = BeautifulSoup(html, 'html.parser')
    return bs.find('div', {'id':'bodyContent'}).find_all('a', href=re.c
ompile('^(/wiki/)((?!:).)*$'))

links.update(getLinks('/wiki/Kevin_Bacon'))

while len(links) > 0:
    newArticle = links[random.randint(0, len(links)-1)].attrs['href']
    print(newArticle)
    links = getLinks(newArticle)
```

In [ ]:

```
"""Recursively crawling an entire site"""

from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

pages = set()
def getLinks(pageUrl):
    global pages
    html = urlopen('http://en.wikipedia.org{}'.format(pageUrl))
    bs = BeautifulSoup(html, 'html.parser')
    for link in bs.find_all('a', href=re.compile('^(/wiki/)')):
        if 'href' in link.attrs:
            if link.attrs['href'] not in pages:
                #We have encountered a new page
                newPage = link.attrs['href']
                print(newPage)
                pages.add(newPage)
                getLinks(newPage)
getLinks('')
```

In [ ]:

```
"""页面内链获取示例"""

from urllib.request import urlopen
from urllib.parse import urlparse
from bs4 import BeautifulSoup
import re
import datetime
import random

pages = set()


# Retrieves a list of all Internal links found on a page
def getInternalLinks(bs, includeUrl):
    includeUrl = '{}://{}'.format(urlparse(includeUrl).scheme, urlparse
(includeUrl).netloc)
    internalLinks = []
    #Finds all links that begin with a "/"
    for link in bs.find_all('a', href=re.compile('^(/|.*'+includeUrl+')
')):
        if link.attrs['href'] is not None:
            if link.attrs['href'] not in internalLinks:
                if(link.attrs['href'].startswith('/')):
                    internalLinks.append(includeUrl+link.attrs['href'])
                else:
                    internalLinks.append(link.attrs['href'])
    return internalLinks
```

In [ ]:

```
"""Retrieves a list of all external links found on a page"""
"""页面外链随机获取示例"""

from urllib.request import urlopen
from urllib.parse import urlparse
from bs4 import BeautifulSoup
import re
import datetime
import random

pages = set()

def getExternalLinks(bs, excludeUrl):
    externalLinks = []
    #Finds all links that start with "http" that do
    #not contain the current URL
    for link in bs.find_all('a', href=re.compile('^(http|www)((?!'+excl
udeUrl+').)*$')):
        if link.attrs['href'] is not None:
            if link.attrs['href'] not in externalLinks:
                externalLinks.append(link.attrs['href'])
    return externalLinks

def getRandomExternalLink(startingPage):
    html = urlopen(startingPage)
    bs = BeautifulSoup(html, 'html.parser')
    externalLinks = getExternalLinks(bs, urlparse(startingPage).netloc)
    if len(externalLinks) == 0:
        print('No external links, looking around the site for one')
        domain = '{}://{}'.format(urlparse(startingPage).scheme, urlpars
e(startingPage).netloc)
        internalLinks = getInternalLinks(bs, domain)
        return getRandomExternalLink(internalLinks[random.randint(0,
                                    len(internalLinks)-1)])
    else:
        return externalLinks[random.randint(0, len(externalLinks)-1)]

def followExternalOnly(startingSite):
    externalLink = getRandomExternalLink(startingSite)
    print('Random external link is: {}'.format(externalLink))
    followExternalOnly(externalLink)

followExternalOnly('http://oreilly.com')
```

In [ ]:

```
"""简单的URL去重程序示例

"""
import requests
from bs4 import BeautifulSoup
import re
import urllib.parse

# 应用数据结构集合（set）的特性来去除重复。
pages = set()
def getLinks(baseUrl,pageUrl):
    global pages
    try:
        headers = {
                    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) A
ppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36
',
                    'Accept-Language': 'zh-CN,zh;q=0.8'
        }
        #baseurl = 'https://en.wikipedia.org/wiki/' + pageUrl
        url = urllib.parse.urljoin(baseUrl,pageUrl)
        print('---'*10)
        print('Begin to fetch url: %s ' % url)
        r = requests.get(url,headers = headers)
        r.raise_for_status()
        bsobj = BeautifulSoup(r.text,'html.parser')

        print('  Title of this page: '+ bsobj.title.get_text())

        # 查找内链
        for link in bsobj.findAll('a',href = re.compile('^(/wiki/)')):
            if 'href' in link.attrs:
                if link.attrs['href'] not in pages:
                    # new url
                    newPage = link.attrs['href']
                    print('  Find new url : %s' % newPage)
                    pages.add(newPage)
                    getLinks(baseUrl,newPage) #递归层次，默认有限 约1000层
    except requests.RequestException as e:
        print(e)
    except :
        print('Some exception had raised.')

getLinks(baseUrl= 'https://en.wikipedia.org/wiki/',pageUrl='Web_crawler'
)
```

## 网站地图生成实例

下面是一个用于爬取全站URLs，生成网站地图的实例。

程序来源: https://github.com/c4software/python-sitemap.git

In [ ]:
```
import logging
from urllib.parse import urljoin, urlunparse
import re
from urllib.parse import urlparse
from urllib.request import urlopen, Request
from urllib.robotparser import RobotFileParser
from datetime import datetime

import mimetypes
import os

class Crawler():
    """
    网站地图生成（爬虫）类，定义了用于生成网站地图的基本方法

    基本使用方法:
        crawl = Crawler(domain = 'some website's domain name', output =
'some filename')
        crawl.run()


    """
    # Sitemap.xml文件头尾信息
    xml_header = """<?xml version="1.0" encoding="UTF-8"?>
                  <urlset xmlns="http://www.sitemaps.org/schemas/si
temap/0.9"
                        xmlns:image="http://www.google.com/schemas/s
itemap-image/1.1"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"
                        xsi:schemaLocation="http://www.sitemaps.org/
schemas/sitemap/0.9
                              http://www.sitemaps.org/schemas/sitema
p/0.9/sitemap.xsd">
               """
    xml_footer = "</urlset>"
    crawler_user_agent = 'Sitemap crawler'

    # Variables
    parserobots = False
    output = None
    report = False

    config = None
    domain = ""

    exclude = []
    skipext = []
    drop    = []

    debug = False

    # 设置待爬取、已爬取、不爬取的集合
    tocrawl = set([])
    crawled = set([])
    excluded = set([])

    marked = {}
    # 程序不解析处理的内容类型
    not_parseable_ressources = (".epub", ".mobi", ".docx", ".doc", ".opf
", ".7z",
                                ".ibooks", ".cbr", ".avi", ".mkv", ".mp4"
, ".jpg",
                                ".jpeg", ".png", ".gif" ,".pdf", ".iso",
".rar",
                                ".tar", ".tgz", ".zip", ".dmg", ".exe",
                               )
    # 设立URL链接正则表达式和图片链接正则表达式
    # TODO also search for window.location={.*?}
    linkregex = re.compile(b'<a [^>]*href=[\'|"](.*?)[\'"][^>]*?>')
    imageregex = re.compile (b'<img [^>]*src=[\'|"](.*?)[\'"].*?>')

    rp = None
    response_code={}
    nb_url=1 # Number of url.
    nb_rp=0 # Number of url blocked by the robots.txt
    nb_exclude=0 # Number of url excluded by extension or word

    output_file = None

    target_domain = ""
    scheme = ""

    def __init__(self, parserobots=False, output=None, report=False ,do
main="",
                exclude=[], skipext=[], drop=[], debug=False, verbose=F
alse, images=False):
        """
        初始化方法

        参数:
            output: 用于指定输出文件名
            domain: 用于指定待爬取的目标域名
        """
        print('The python sitemap creator is initializing...')
        self.parserobots = parserobots
        self.output = output
        self.report = report
        self.domain = domain
        self.exclude = exclude
        self.skipext = skipext
        self.drop = drop
        self.debug = debug
        self.verbose = verbose
        self.images = images

        if self.debug:
            log_level = logging.DEBUG
        elif self.verbose:
            log_level = logging.INFO
        else:
            log_level = logging.ERROR

        logging.basicConfig(level=log_level)
        # 设置待爬取的初始URL,即网站域名
        self.tocrawl = set([self.clean_link(domain)])

        try:
            url_parsed = urlparse(domain)
            self.target_domain = url_parsed.netloc
            self.scheme = url_parsed.scheme
        except:
            logging.error("Invalide domain")
            raise ("Invalid domain")

        if self.output:
            try:
                self.output_file = open(self.output, 'w')
            except:
                logging.error ("Output file not available.")
                exit(255)

        print('  Initial process is done.')

    def run(self):
        """
        爬虫启动函数,用于启动爬取域名全部URL的程序

        """
        print('Begin to run the crawler...')
        print('  the sitemap of %s will be written into %s' % (self.doma
in,self.output_file))
        print(self.xml_header, file=self.output_file)

        if self.parserobots:
            self.check_robots()
        logging.info("Start the crawling process")

        while len(self.tocrawl) != 0:
            self.__crawling()
        logging.info("Crawling has reached end of all found links")

        print(self.xml_footer, file=self.output_file)
        print("Crawling has reached end of all found links")

    def __crawling(self):
        """
        爬虫实现函数
        """
        crawling = self.tocrawl.pop()

        url = urlparse(crawling)
        self.crawled.add(crawling)
        logging.info("Crawling #{}: {}".format(len(self.crawled), url.g
eturl()))
        request = Request(crawling, headers={"User-Agent":self.crawler_u
ser_agent})
```

```
        # Ignore ressources listed in the not_parseable_ressources
        # Its avoid dowloading file like pdf… etc
        if not url.path.endswith(self.not_parseable_ressources):
            try:
                response = urlopen(request)
            except Exception as e:
                if hasattr(e,'code'):
                    if e.code in self.response_code:
                        self.response_code[e.code]+=1
                    else:
                        self.response_code[e.code]=1

                    #  管理 urls marked 并 reporting
                    if self.report:
                        if e.code in self.marked:
                            self.marked[e.code].append(crawling)
                        else:
                            self.marked[e.code] = [crawling]

                logging.debug ("{1} ==> {0}".format(e, crawling))
                return self.__continue_crawling()
        else:
            logging.debug("Ignore {0} content might be not parseable.".f
ormat(crawling))
            response = None

        # Read the response
        if response is not None:
            try:
                msg = response.read()
                if response.getcode() in self.response_code:
                    self.response_code[response.getcode()]+=1
                else:
                    self.response_code[response.getcode()]=1

                response.close()

                # Get the last modify date
                if 'last-modified' in response.headers:
                    date = response.headers['Last-Modified']
                else:
                    date = response.headers['Date']

                date = datetime.strptime(date, '%a, %d %b %Y %H:%M:%S %Z
')

            except Exception as e:
                logging.debug ("{1} ===> {0}".format(e, crawling))
                return None
        else:
            # Response is None, content not downloaded, just continu and
add
            # the link to the sitemap
            msg = "".encode( )
            date = None

        # Image sitemap enabled ?
        image_list = "";
        if self.images:
            # Search for images in the current page.
            images = self.imageregex.findall(msg)
```

```
                for image_link in list(set(images)):
                    image_link = image_link.decode("utf-8", errors="ignore")

                    # Ignore link starting with data:
                    if image_link.startswith("data:"):
                        continue

                    # If path start with // get the current url scheme
                    if image_link.startswith("//"):
                        image_link = url.scheme + ":" + image_link
                    # Append domain if not present
                    elif not image_link.startswith(("http", "https")):
                        if not image_link.startswith("/"):
                            image_link = "/{0}".format(image_link)
                        image_link = "{0}{1}".format(self.domain.strip("/"),
image_link.replace("./", "/"))

                    # Ignore image if path is in the exclude_url list
                    if not self.exclude_url(image_link):
                        continue

                    # Ignore other domain images
                    image_link_parsed = urlparse(image_link)
                    if image_link_parsed.netloc != self.target_domain:
                        continue


                    # Test if images as been already seen and not present in
the
                    # robot file
                    if self.can_fetch(image_link):
                        logging.debug("Found image : {0}".format(image_link)
)
                        image_list = "{0}<image:image><image:loc>{1}</image:l
oc></image:image>".format(image_list, self.htmlspecialchars(image_link)
)

        # Last mod fetched ?
        lastmod = ""
        if date:
            lastmod = "<lastmod>"+date.strftime('%Y-%m-%dT%H:%M:%S+00:00
')+"</lastmod>"

        print ("<url><loc>"+self.htmlspecialchars(url.geturl())+"</loc>"
+ lastmod + image_list + "</url>", file=self.output_file)
        if self.output_file:
            self.output_file.flush()

        # Found links
        links = self.linkregex.findall(msg)
        for link in links:
            link = link.decode("utf-8", errors="ignore")
            link = self.clean_link(link)
            logging.debug("Found : {0}".format(link))

            if link.startswith('/'):
                link = url.scheme + '://' + url[1] + link
            elif link.startswith('#'):
                link = url.scheme + '://' + url[1] + url[2] + link
            elif link.startswith(("mailto", "tel")):
                continue
            elif not link.startswith(('http', "https")):
                link = url.scheme + '://' + url[1] + '/' + link

            # Remove the anchor part if needed
            if "#" in link:
                link = link[:link.index('#')]

            # Drop attributes if needed
            for toDrop in self.drop:
                link=re.sub(toDrop,'',link)

            # Parse the url to get domain and file extension
            parsed_link = urlparse(link)
            domain_link = parsed_link.netloc
            target_extension = os.path.splitext(parsed_link.path)[1][1:]

            if link in self.crawled:
                continue
            if link in self.tocrawl:
                continue
            if link in self.excluded:
                continue
            if domain_link != self.target_domain:
                continue
            if parsed_link.path in ["", "/"]:
                continue
            if "javascript" in link:
                continue
            if self.is_image(parsed_link.path):
                continue
            if parsed_link.path.startswith("data:"):
                continue

            # Count one more URL
            self.nb_url+=1

            # Check if the navigation is allowed by the robots.txt
            if not self.can_fetch(link):
                self.exclude_link(link)
                self.nb_rp+=1
                continue

            # Check if the current file extension is allowed or not.
            if (target_extension in self.skipext):
                self.exclude_link(link)
                self.nb_exclude+=1
                continue

            # Check if the current url doesn't contain an excluded word
            if (not self.exclude_url(link)):
                self.exclude_link(link)
                self.nb_exclude+=1
                continue

            self.tocrawl.add(link)

        return None

    def clean_link(self, link):
        l = urlparse(link)
        l_res = list(l)
```

```
            l_res[2] = l_res[2].replace("./", "/")
            l_res[2] = l_res[2].replace("//", "/")
            return urlunparse(l_res)

    def is_image(self, path):
            mt,me = mimetypes.guess_type(path)
            return mt is not None and mt.startswith("image/")

    def __continue_crawling(self):
            if self.tocrawl:
                self.__crawling()

    def exclude_link(self,link):
            if link not in self.excluded:
                self.excluded.add(link)

    def check_robots(self):
            robots_url = urljoin(self.domain, "robots.txt")
            self.rp = RobotFileParser()
            self.rp.set_url(robots_url)
            self.rp.read()

    def can_fetch(self, link):
            try:
                if self.parserobots:
                    if self.rp.can_fetch("*", link):
                        return True
                    else:
                        logging.debug ("Crawling of {0} disabled by robots.tx
t".format(link))
                        return False

                if not self.parserobots:
                    return True

                return True
            except:
                # On error continue!
                logging.debug ("Error during parsing robots.txt")
                return True

    def exclude_url(self, link):
            for ex in self.exclude:
                if ex in link:
                    return False
            return True

    def htmlspecialchars(self, text):
            return text.replace("&", "&amp;").replace('"', "&quot;").replace
("<", "&lt;").replace(">", "&gt;")

    def make_report(self):
            print ("Number of found URL : {0}".format(self.nb_url))
            print ("Number of link crawled : {0}".format(len(self.crawled)))
            if self.parserobots:
                print ("Number of link block by robots.txt : {0}".format(sel
f.nb_rp))
            if self.skipext or self.exclude:
                print ("Number of link exclude : {0}".format(self.nb_exclude
))
```

```
        for code in self.response_code:
            print ("Nb Code HTTP {0} : {1}".format(code, self.response_c
ode[code]))

        for code in self.marked:
            print ("Link with status {0}:".format(code))
            for uri in self.marked[code]:
                print ("\t- {0}".format(uri))

crawler = Crawler(domain = 'http://10.10.10.135',output = 'Sitemap.xml')

crawler.run()
crawler.make_report()
```

```
The python sitemap creator is initializing...
  Initial process is done.
Begin to run the crawler...
  the sitemap of http://10.10.10.135 will be written into <_io.TextIOWra
pper name='Sitemap.xml' mode='w' encoding='cp936'>
```

## URL的遍历

如果我们把每一个网页抽象化，看作“图”的一个结点，而页面中存在指向另一页面的URL（链接），则将其抽象为一条有向边。这样一个网站中的网页与URL可以抽象化为一个“有向图”，而整个Internet可以抽象为有向图构成的森林。

通过抽象，可以将URL的遍历问题转化为有向图的遍历问题。经典的图遍历算法有深度优先（DFS）和广度优先（BFS）两类。下面的例子给出了图数据结构的定义与基本操作。

In [16]:
```
"""python 图的遍历算法示例"""

#!/usr/bin/python
# -*- coding: utf-8 -*-

class Graph(object):

    def __init__(self,*args,**kwargs):
        self.node_neighbors = {}
        self.visited = {}

    def add_nodes(self,nodelist):

        for node in nodelist:
            self.add_node(node)

    def add_node(self,node):
        if not node in self.nodes():
            self.node_neighbors[node] = []

    def add_edge(self,edge):
        u,v = edge
        if(v not in self.node_neighbors[u]) and ( u not in self.node_nei
ghbors[v]):
            self.node_neighbors[u].append(v)

            if(u!=v):
                self.node_neighbors[v].append(u)

    def nodes(self):
        return self.node_neighbors.keys()

    def depth_first_search(self,root=None):
        order = []
        def dfs(node):
            self.visited[node] = True
            order.append(node)
            for n in self.node_neighbors[node]:
                if not n in self.visited:
                    dfs(n)


        if root:
            dfs(root)

        for node in self.nodes():
            if not node in self.visited:
                dfs(node)

        print(order)
        return order

    def breadth_first_search(self,root=None):
        queue = []
        order = []
        def bfs():
            while len(queue)> 0:
                node  = queue.pop(0)

                self.visited[node] = True
                for n in self.node_neighbors[node]:
                    if (not n in self.visited) and (not n in queue):
                        queue.append(n)
                        order.append(n)

        if root:
            queue.append(root)
            order.append(root)
            bfs()

        for node in self.nodes():
            if not node in self.visited:
                queue.append(node)
                order.append(node)
                bfs()
        print(order)

        return order


if __name__ == '__main__':
    g = Graph()
g.add_nodes([i+1 for i in range(8)])
g.add_edge((1, 2))
g.add_edge((1, 3))
g.add_edge((2, 4))
g.add_edge((2, 5))
g.add_edge((4, 8))
g.add_edge((5, 8))
g.add_edge((3, 6))
g.add_edge((3, 7))
```

```
g.add_edge((6, 7))
print("nodes:", g.nodes())

order = g.breadth_first_search(1)
order = g.depth_first_search(1)
```

```
nodes: dict_keys([1, 2, 3, 4, 5, 6, 7, 8])
[1, 2, 3, 4, 5, 6, 7, 8]
[1]
```

## URL判重策略

当遍历多个网页中的URLs时，可能会出现许多重复的URLs，这些URLs可能会形成有向环，这对于未考虑环的遍历算法而言将进入“死循环”状态，这显然是应当避免出现的。

- 爬取资源量较小时，可以采用数据结构“集合”
- 爬取资源量中等时，可以采用经典的Bitmap算法或更强大的Bloom Filter方法;
- 如果爬取量很大，则需要采用redis分布式策略，把URL转为md5作为key来进行分布式Bloom Filter判重。

In [24]:
```
"""集合操作"""
a = set('abracadabra')
b = set('alacazam')
print(a)
print(b)
print('---'*10)
# 集合差运算
print(a-b)
# 集合并运算
print(a | b)
# 集合交运算
print(a & b)
# 不同时包含于a和b的元素
print(a ^ b)

s = set()
# 添加元素
#  将元素 x 添加到集合 s 中，如果元素已存在，则不进行任何操作
s.add( 'x' )
#  添加元素，且参数可以是列表，元组，字典等
x = 'abcd'
s.update(x)

# 移除元素
#    将元素 x 从集合 s 中移除，如果元素不存在，则会发生错误。
thisset = set(("Google", "Runoob", "Taobao"))
thisset.remove("Taobao")
print(thisset)
#{'Google', 'Runoob'}
#thisset.remove("Facebook")

#   移除集合中的元素，且如果元素不存在，不会发生错误。
thisset = set(("Google", "Runoob", "Taobao"))
thisset.discard("Facebook")  # 不存在不会发生错误
print(thisset)
{'Taobao', 'Google', 'Runoob'}
#   随机删除集合中的一个元素
thisset = set(("Google", "Runoob", "Taobao", "Facebook"))
```

```
x = thisset.pop()
print(x)

# 清空集合
thisset = set(("Google", "Runoob", "Taobao"))
thisset.clear()
print(thisset)

#判断元素是否在集合中存在
#  语法格式如下：
#   x in s
#   判断元素 s 是否在集合 x 中，存在返回 True，不存在返回 False。

thisset = set(("Google", "Runoob", "Taobao"))
print("Runoob" in thisset)

print("Facebook" in thisset)
```

```
{'a', 'd', 'r', 'b', 'c'}
{'a', 'm', 'l', 'z', 'c'}
--------------------------------
{'b', 'd', 'r'}
{'a', 'd', 'r', 'm', 'l', 'b', 'z', 'c'}
{'c', 'a'}
{'b', 'l', 'z', 'd', 'm', 'r'}
{'Google', 'Runoob'}
{'Google', 'Taobao', 'Runoob'}
Google
set()
True
False
```

布隆过滤器

布隆过滤器 (Bloom Filter)是由Burton Howard Bloom于1970年提出，它是一种space efficient的概率型数据结构，用于判断一个元素是否在集合中。在垃圾邮件过滤的黑白名单方法、爬虫(Crawler)的网址判重模块中等等经常被用到。哈希表也能用于判断元素是否在集合中，但是布隆过滤器只需要哈希表的1/8或1/4的空间复杂度就能完成同样的问题。布隆过滤器可以插入元素，但不可以删除已有元素。其中的元素越多，false positive rate(误报率)越大，但是false negative (漏报)是不可能的。

基本概念

如果想判断一个元素是不是在一个集合里，一般想到的是将所有元素保存起来，然后通过比较确定。链表，树等等数据结构都是这种思路. 但是随着集合中元素的增加，我们需要的存储空间越来越大，检索速度也越来越慢。不过世界上还有一种叫作散列表（又叫哈希表，Hash table）的数据结构。它可以通过一个Hash函数将一个元素映射成一个位阵列（Bit Array）中的一个点。这样一来，我们只要看看这个点是不是 1 就知道可以集合中有没有它了。这就是布隆过滤器的基本思想。

Hash面临的问题就是冲突。假设 Hash 函数是良好的，如果我们的位阵列长度为 m 个点，那么如果我们想将冲突率降低到例如 1%, 这个散列表就只能容纳 m/100 个元素。显然这就不叫空间有效了（Space-efficient）。解决方法也简单，就是使用多个 Hash，如果它们有一个说元素不在集合中，那肯定就不在。如果它们都说在，虽然也有一定可能性它们在说谎，不过直觉上判断这种事情的概率是比较低的。

优点

相比于其它的数据结构，布隆过滤器在空间和时间方面都有巨大的优势。布隆过滤器存储空间和插

入/查询时间都是常数。另外, Hash 函数相互之间没有关系，方便由硬件并行实现。布隆过滤器不需要存储元素本身，在某些对保密要求非常严格的场合有优势。

布隆过滤器可以表示全集，其它任何数据结构都不能；

k 和 m 相同，使用同一组 Hash 函数的两个布隆过滤器的交并差运算可以使用位操作进行。

缺点

但是布隆过滤器的缺点和优点一样明显。误算率（False Positive）是其中之一。随着存入的元素数量增加，误算率随之增加。但是如果元素数量太少，则使用散列表足矣。

另外，一般情况下不能从布隆过滤器中删除元素. 我们很容易想到把位列阵变成整数数组，每插入一个元素相应的计数器加1, 这样删除元素时将计数器减掉就可以了。然而要保证安全的删除元素并非如此简单。首先我们必须保证删除的元素的确在布隆过滤器里面. 这一点单凭这个过滤器是无法保证的。另外计数器回绕也会造成问题。

分布式策略

分布式策略在后面的课程中介绍，例如有python 基于redis实现的bloomfilter(布隆过滤器)。