Journal on Big Data JBD, vol.1, no.2, pp.89-106, 2019

On Multi-Thread Crawler Optimization for Scalable Text Searching

Guang Sun[1], Huanxin Xiang[2] and Shuanghu Li[1, *]

Abstract: Web crawlers are an important part of modern search engines. With the development of the times, data has exploded and humans have entered a "big data era". For example, Wikipedia carries the knowledge from all over the world, records the real-time news that occurs every day, and provides users with a good database of data, but because of the large amount of data, it puts a lot of pressure on users to search. At present, single-threaded crawling data can no longer meet the requirements of text crawling. In order to improve the performance and program versatility of single-threaded crawlers, a high-speed multi-threaded web crawler is designed to crawl the network hyper-scale text database. Multi-threaded crawling uses multiple threads to process web pages in parallel, combining breadth-first and depth-first algorithms to control web crawling. The practice project is based on the Python language to achieve multi-threaded optimization network hyper-large-scale text database-Wikipedia book crawling method, the project is inspired by the article on the Wikipedia article in the Big Data Digest public number.

Keywords: Multi-threading, text database, optimization, breadth-first search, depth-first search.

1 Introduction

Wikipedia-a treasure trove of human knowledge

As the world's largest and most popular online encyclopedia, Wikipedia brings together the wisdom of the world. Wikipedia is not just an encyclopedia textbook, it is also a multi-person collaborative writing system. Wiki is a hypertext system that is open on the web and can be collaboratively created by multiple people. As an encyclopedia with free content and free editing, Wikipedia's goal is to provide a freely created encyclopedia to all human beings, which can be maintained by many people (or even any visitors), and everyone can express their opinions, or Extend or explore common themes. Wikipedia is not only the crystallization of knowledge in the academic world, but also the real-time news and major events that happen every day. It is comparable to the world's largest text database. A large amount of information provides a valuable warehouse for data analysis and data mining, but it is precisely because of the large amount of data, the search method is single, many analysts have no way to start. In the way of searching and downloading Wikipedia, the traditional single-threaded data crawling has been unable to

[1] Hunan University of Finance and Economics, Changsha, 410205, China.

[2] The University of Alabama, Tuscaloosa, 35401, USA.

[*] Corresponding Author: Shuanghu Li. Email: lishuanghu13787031773@163.com.

meet the current data volume requirement. Even if the single-threaded method can climb large-scale network data, the time cost required is ours. unacceptable. For example, I want to search and download all the Wikipedia articles and papers published in 2018. If I use the traditional single-threaded crawling method, it will take 15 hours, and the manual download is even more exaggerated. However, if the multi-threaded download parsing is used, the whole process can be downloaded for 10 minutes, and the ultra-large-scale text data crawling download can be realized (the project is derived from an experiment in the public number of the big data article, cited here by way of example). The multi-threaded network hyper-scale text crawling introduced in this paper is to improve the defects of single-threaded existence, and provide a convenient, flexible, efficient and fast data collection method for data science projects.

2 Related technology and program introduction

The following is an introduction to the related technologies and overall implementation of this article.

2.1 Multi-threaded-breadth-first multi-threaded crawler

Usually we talk about the thread is actually an execution flow in the program, a thread in the computer is a program, so the thread also needs to occupy memory, occupy resources. Multithreading refers to the fact that a program contains multiple execution streams. In other words, a program can run multiple different threads to perform different tasks at the same time. In other words, a single program is allowed to create multiple threads of parallel execution to complete their respective tasks. Multithreaded thoughts like a distributed parallel processing, we are talking about big data when a machine when there is no way to load a large amount of data processing, with a distributed cluster the data batch stored in different machines, with the method of parallel processing data, so as to reduce the single load, improve the process speed, better, faster to achieve the batch processing of data. Multithreading is often associated with multiple processes, so what is a process? The definition of the computer to explain is: the program in the computer every execution process is a process, it is the system for resource allocation and scheduling of the basic units, is the basis of the operating system structure. In other words, a process is a program with certain independent functions about a running activity on a certain data set, and a process is an independent unit of the system for resource allocation and scheduling. A thread is an entity of a process, the basic unit of CPU scheduling and dispatching, and it is a smaller unit that can run independently than a process.

The relationship between process and thread is as follows:

(1) A thread can only belong to one process, and a process can have multiple threads, but at least one thread.

(2) Resources are allocated to the process, and all threads of the same process share all resources of the process.

(3) The CPU is allocated to the thread, that is, the thread actually runs on the CPU.

Introducing multiple processes we have to introduce two concepts: parallel processing and concurrent processing. Parallel Processing is a computing method in which two or more processes can be performed simultaneously in a computer system. Parallel processing can work on different aspects of the same program at the same time. Concurrency Processing: we are talking about a time period when we have several programs running from startup to completion, all of which are running on the same processor (CPU) but all of which are running on one processor (CPU) at any given point in time.

The core of multithreading is parallel processing + batch processing, which has the following advantages:

(1) It can improve the utilization rate of CPU. Different from single-thread, multi-thread can run other threads in parallel while waiting for a process to improve the utilization efficiency of CPU

(2) It can improve the utilization of network bandwidth. When multiple threads are processing at the same time, the network resources will be maximized

(3) Save the time cost of resource search

(4) High efficiency, multi-thread crawling data can be immediately used to download the data analysis process.

2.2 BFS-Breadth first search algorithm

Breadth-first algorithm is one of the traversal algorithms of graphs, and the other is depth-first algorithm. Breadth-first search algorithm inherits the core idea of breadth-first traversal, which is similar to the hierarchical traversal structure of trees. The breadth-first algorithm is different from the depth-first algorithm we have come into contact with. Its core is traversing, traversing layer by layer, looking for unvisited neighbor nodes for access, and traversing one by one without going deep. Breadth-first search is also called breadth-first multi-threaded search. Python multi-threading is pseudo-multi-threading. In fact, Python itself is a single thread, which will automatically allocate resources when the program is running. When the crawl is made, a call to the threading library is made to the web text data crawl using the breadth-first search pseudo-multithreading approach.

The core idea of breadth-first search algorithm:

1. Starting from some vertex v in the graph, first visit the fixed point v0

2. Visit the unvisited adjacency points of v0 successively after visiting v1;

3. Then, starting from these adjacency points, visit their adjacency points in turn, and make the adjacency points of the first accessed vertex are accessed before the adjacency points of the later accessed vertex;

4. Until the adjacency points of all vertices that have been accessed in the graph are accessed;

5. If there are still unaccessed vertices in the graph at this time, another unaccessed vertex shall be selected as the new starting point, and the above process shall be repeated until all the vertices in the graph are accessed.

We can set v1 as the first layer, v2, v3, and v4 as the second layer, and v5, v6, v7, v8, v9, and v10 as the third layer. Breadth-first traversal search is realized by means of queue

data access and readout process. Starting from v1 in the figure, and then traversing each vertex of each layer one by one, breadth first is mainly traversing layer by layer, which emphasizes a wide range but not in-depth.

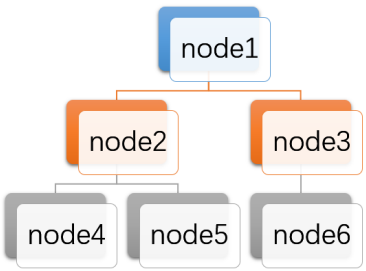


Figure 1: Structure diagram of Breadth-first

2.3 DFS-Depth first search algorithm

Depth-first search is a classical algorithm in graph theory. By using depth-first search algorithm, the corresponding topological sorting table of the target graph can be generated, and many related graph theory problems, such as maximum path problem, can be conveniently solved by using topological sorting table. The depth-first traversal of a graph is similar to the preorder traversal of a tree. The characteristic of the search method adopted is to search the depth direction first as far as possible. This Search method is called depth-first Search. Accordingly, traversing a graph in this way is naturally called depth-first traversal of the graph. The depth traversal algorithm of the figure can be roughly divided into the following three steps:

(1) First select an unaccessed vertex V as the starting vertex (or access the specified starting vertex V), and mark it as accessed;

(2) Then search all the vertices adjacent to vertex V, and judge whether these vertices have been accessed. If there are any vertices that have not been accessed, select any vertex W for access. Then select any vertex adjacent to vertex W that has not been accessed and visit it, and repeat in turn. When all adjacent vertices of a vertex are accessed, the most recently accessed vertex is retraced in turn. If the vertex has other adjacent vertices that are not accessed, take one of these unaccessed vertices and repeat the process until all the vertices connected to the starting vertex V are accessed.

(3) If there are still some unaccessed vertices in the graph at this time, then select one of them as the starting vertex and access it, turn (2); otherwise, the traversal ends.

Traversal rule of depth-first algorithm: traversal continuously along the depth direction of vertices. Depth direction of vertex refers to its adjacency point direction, and the principle of depth-first algorithm is “dig deep but not dig wide”. In a crawling program, depth-first search starts from node 1 and then goes down to node 2 and node 3 to find the first unaccessed adjacency point of node 1 and access the vertex. Take the vertex as a new vertex, and repeat this step until the vertex just visited has no unvisited adjacency points. Return the previous visited vertex that still has unvisited adjacency point, continue to visit

the next unvisited collar point of this vertex, repeat Steps 2, 3, until all the vertices are accessed, and the search ends. During the holiday so is accessed points stored in a list array.

Search depth first search algorithm process as shown in the figure below: All nodes are not being accessed, when the initial depth-first traversal search from node 1 trip, then income down to node 2, and 3, in order to search deep down, has been looking for is not access vertices, layer upon layer thorough, digging to the bottom, the depth first search through the stack to implement the data in the readout.

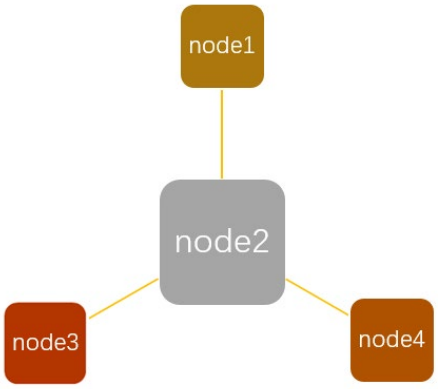


Figure 2: Structure diagram of depth-first algorithm

2.4 Scheme introduction

This article mainly uses the contrast reference method, through realizes the single thread crawls the data and the multithread crawls the data to carry on the comparison, obtains the conclusion.

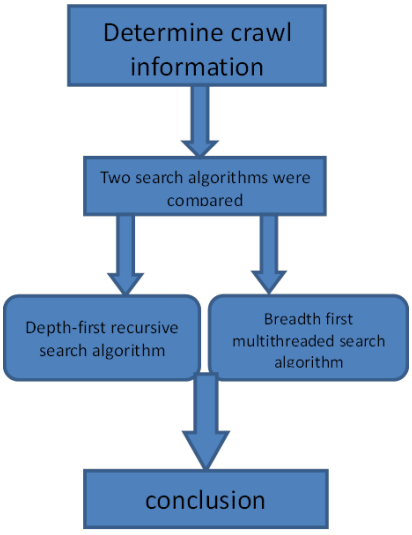


Figure 3: Project process diagram

3 Experiment and result

The designer of the website adopts the hierarchical structure design, the top layer is the top-level domain name, followed by the sub-domain name, the sub-domain name and so on, at the same time, each sub-domain may have multiple domain names of the same level, and the URL may be linked to each other, a variety of shapes, thus forming a complex network. In order to highlight the breadth-first multithreaded search advantage, we use the method of reference effect contrast, crawl the same Wikipedia database content, with a depth first search algorithm, a breadth-first search algorithm is used to implement, in both a timer is added to a python program, climbing the time required to take large-scale data are calculated respectively, then comparing the timing results, two realization process is as follows（The programming language for this experiment is based on python. The experimental environment is as follows: Windows 64-bit operating system; anaconda3 manager; Visual Studio Code compiler; python3.x）

3.1 The depth-first search implementation process

The depth-first process is actually implemented in a recursive way. First, define a function to implement the depth-first process, and then pass in the node parameters. If the node is not empty, print it out. You can analogize the top point A in the binary tree. After the node is printed, check to see if it has a left node (link B) and a right node (link C). If the left node is not empty, return it and call the depth-first function itself for recursion to get a new one. Left node (link D) and right node (link E), and so on, until all nodes are traversed or reach the established conditions to stop.

(1) Initially, so the nodes are not accessed, the list array is empty.

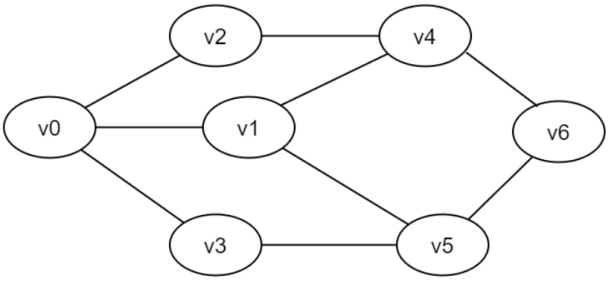
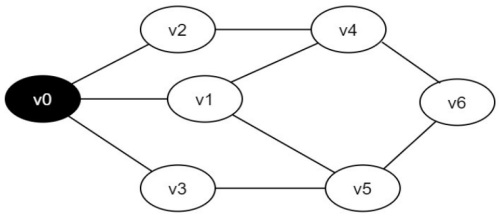


Figure 4: The initial conditions

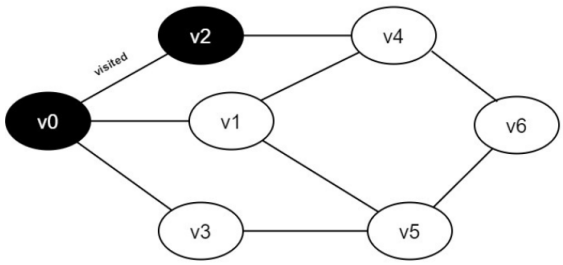
The first step is to access the v0 node, and the v0 is marked and accessed (the number 1 indicates that it has been accessed).



List	1	0	0	0	0	0	.....	o

Figure 5: Access the v0 node

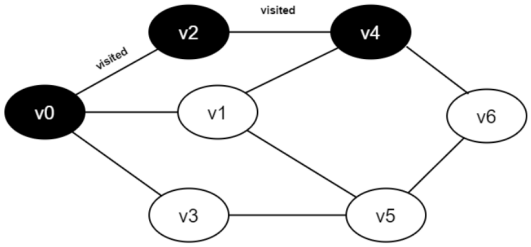
(2) Accessing the neighboring point v2 of v0. Determine list [2], if the value of list [2] is 0, access v2 and set list [2] to 1.



List	1	0	1	0	0	0	.....	o

Figure 6: Access the v2 node

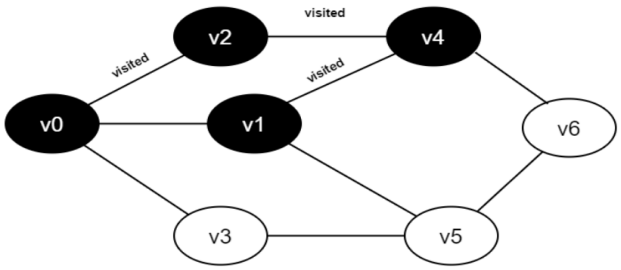
(3) Access v2's neighboring point v0, and judge that list has a value of 1, not accessing. Continue to access v2's neighbor v4, determine that list has a value of 0, access v4, and set list4 to 1.



List	1	0	1	0	1	0	.....	o

Figure 7: Access the v4 node

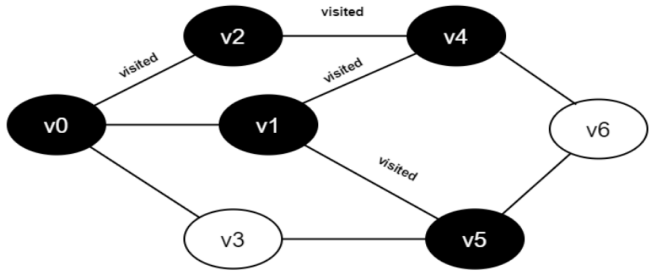
(4) Access the v4 neighboring point v1, determine that the list has a value of 0, access v1, and list is 1.



List	1	1	1	0	1	0	.....	o

Figure 8: Access the v1 node

(5) The neighboring point v0 of v1 determines that the list1 value is 1, and does not access. Continue to access v1's neighboring point v4, and the segment list value is 1, not accessed. Continue to access v1's neighboring point v5, and the list is judged to have a value of 0 and access v5. And set list [5] to 1.



List	1	1	1	0	1	1	.....	o

Figure 9: Access the v5 node

(6) Accessing v5's neighboring point v1, determining that list [1] is 1 is not accessed, continuing to access v5's neighboring point v3, determining list [3], its value is 0, accessing v3, and setting v3 to 1.

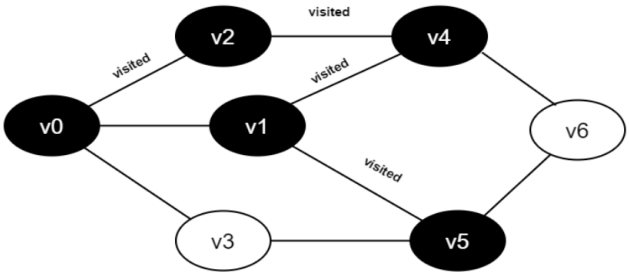


Figure 10: Access the v3 node

(7) Access v3's neighboring point v0, judge list [0], its value is 1, no access. Continue to access v3's neighboring point v5, and judge list [5], its value is 1, not accessed. All neighbors of v3 have been accessed, going back to their previous vertex v5, traversing all the neighbors of v5. Access v5's neighbor v6, judge list [6], its value is 0, access v6, set list [6] to 1.

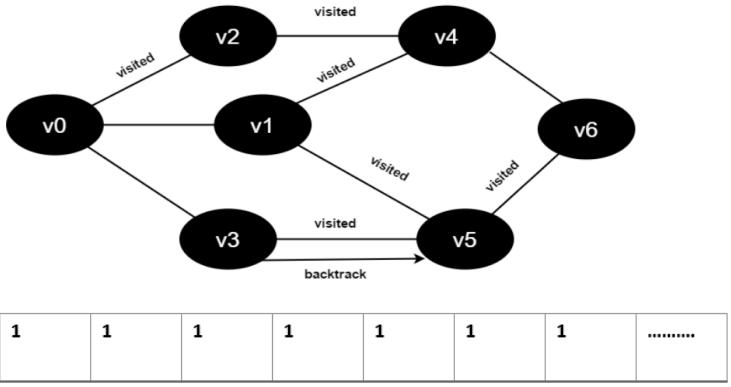


Figure 11: Access the v6 node

(8) Access v6's neighboring point v4, judge list [4], its value is 1, no access. Access v6's neighboring point v5, and judge list [5], its value is 1, not accessed. All neighbors of v6 have been accessed, going back to their previous vertex v5, traversing the remaining neighbors of v5, all neighbors of v5 have been accessed, going back to their previous vertex v1. All neighbors of v1 have been accessed, going back to their last vertex v4, traversing v4 remaining neighbors v6. All neighbors of v4 have been accessed, going back to their previous vertex v2. All neighbors of v2 have been accessed, going back to their previous vertex v1, traversing the remaining neighbor v3 of v1. All neighbors of v1 have been accessed and the search ends.

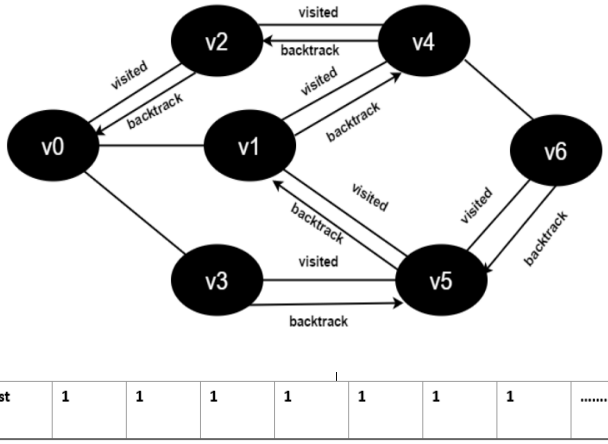


Figure 12: Complete all traversals

(9) The following is a program to achieve Wikipedia title text crawling, using anaconda to run the program results are as follows:

No.206113 Depth:2 IP_addresses > Scalability

No.206114 Depth:2 IP_addresses -> virtual_IP_address

No.206115 Depth:2IP_addresses -> Network_address_translation

No.206116 Depth:2IP_adresses -> Blacklist_(computing)

totally cost 1000.2916378974915

Result 1: The result diagram of depth-first algorithm implementation

3.2 The Breadth-first search implementation process

The whole breadth-first crawling process starts with a series of seed nodes, extracts the "child nodes" (that is, hyperlinks) in these pages, and puts them into the queue to fetch them in turn. The processed links need to be placed in a table. Before each new link is processed, you need to see if the link already exists in the Visited table. If it exists, the proof link has been processed, skipped, and not processed, otherwise the next step is processed.

The initial URL address is the seed URL provided in the crawler system (generally specified in the system's configuration file). When parsing the web page represented by these seed URLs, a new URL will be generated (for example, http://www.admin.com is extracted from <a href="http://www.admin.com" in the page. link). Then, do the following:

Compare the parsed link with the link in the Viseded table. If the link does not exist in the Visited table, it indicates that it has not been accessed. Put the link in the TODO table. After processing, get a link from the TODO table again and put it directly into the Visited table. Continue with the above process for the web page represented by this link. This

cycle. Breadth-first traversal is one of the most widely used crawling strategies in crawlers. There are three main reasons for using breadth-first search strategies:

(1) Important web pages are often close to the seeds. For example, when we open a news website, it is often the most popular news. As we surf deeper, the importance of the web pages we see is getting lower and lower.

(2) The actual depth of the World Wide Web can reach up to 17 layers, but there is always a short path to a certain web page. The breadth-first traversal will reach this page as quickly as possible.

(3) Breadth priority is conducive to the cooperation of multi-reptiles. Multi-reptile cooperation usually first grabs the link in the station, and the sealing is very strong.

The crawling steps are as follows:

(1) Initially all vertices are not accessed, the list array is initialized to 0, and there are no elements in the queue.

(2) Start to access v0, access vertex v0, and set the value of list [0] to 1, and enter v0 into the queue.

(3) Dequeue v0 and access the v2 neighbor v2. Determine list [2], because the value of list [2] is 0, access v2, set list [2] to 1, and join v2

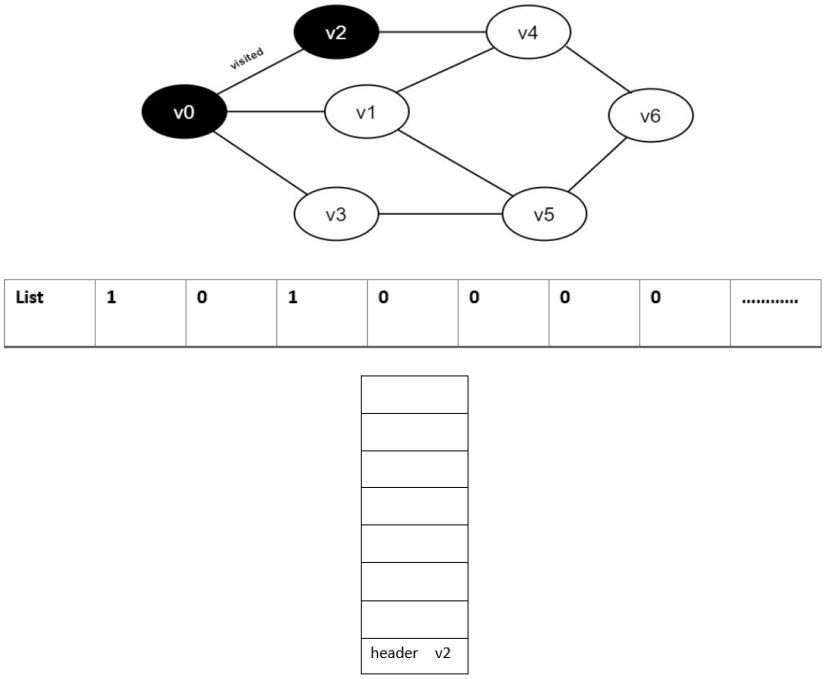


Figure 15: Access the v2 node

(4) Access v0 neighbor point v1. Determine list [1], because list [1] has a value of 0, access v1, set list [1] to 0, and queue v1.

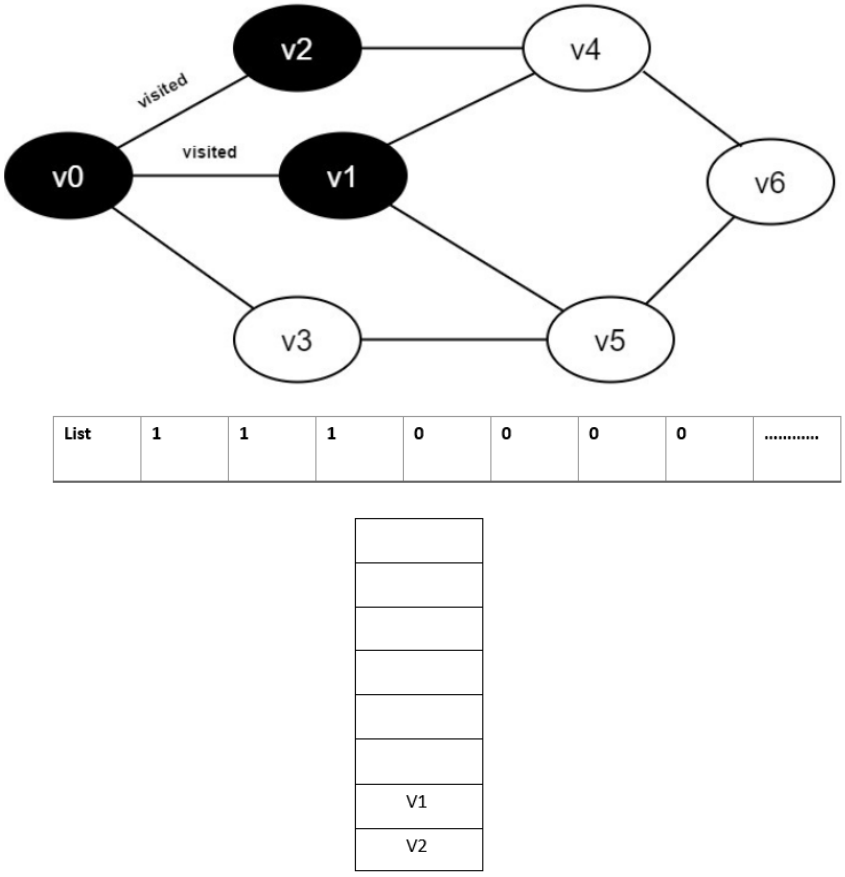
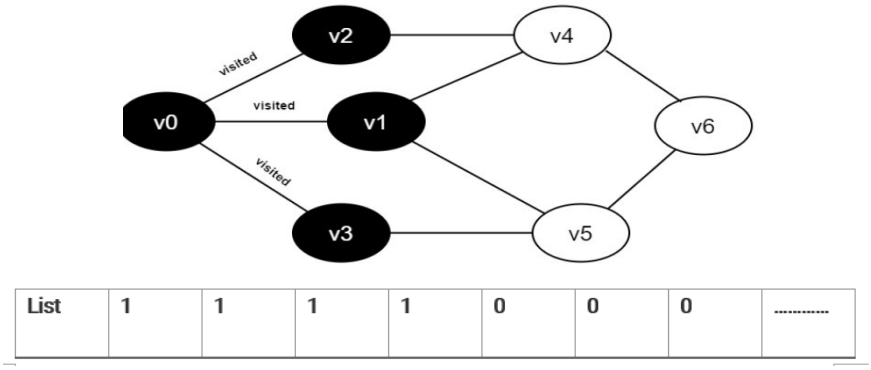


Figure 16: Access the v1 node

(5) Then judge list [3] because its value is 0 and access v3. Set list [3] to 0 and queue v3.



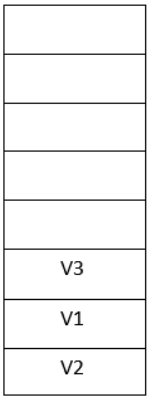


Figure 17: Access the v3 node

(6) All neighbors of v0 have been accessed. Dequeue the team head element v2, start to access all the neighboring points of v2, start to access the v2 neighboring point v0, and judge list [0], because its value is 1, no access is made. Continue to access v2 neighbor v4, judge list [4], because its value is 0, access v4, set list [4] to 1, and enter v4.

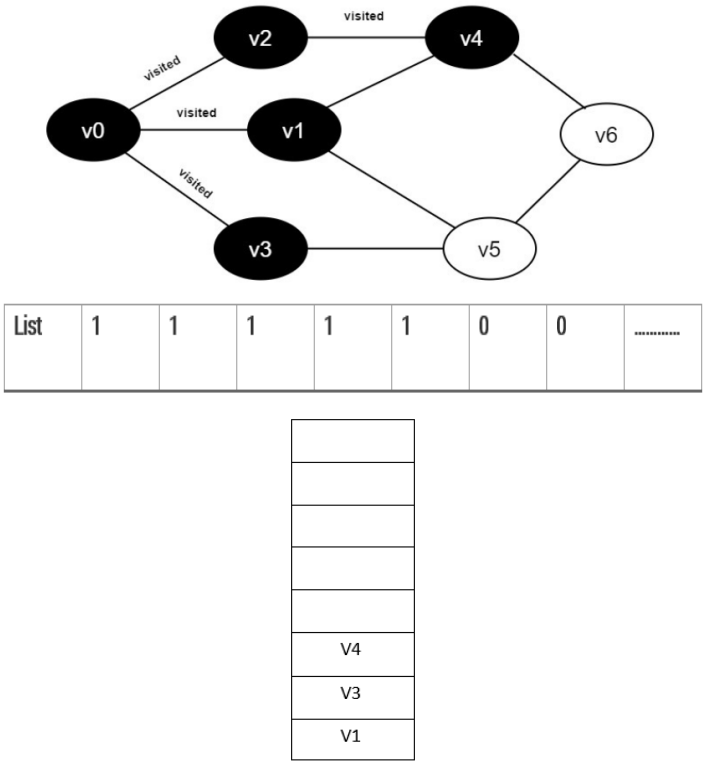


Figure 18: Access the v4 node

(7) All the neighbors of v2 have been accessed. Dequeue the team head element v1 and start accessing all the neighbors of v1. Start accessing the v1 neighbor point v0 because the list [0] value is 1 and no access is made. Continue to access v1 neighbor v4, because the value of list [4] is 1, no access is made. Continue to access v1 neighbor v5, because list [5] has a value of 0, access v5, set list [5] to 1, and queue v5.

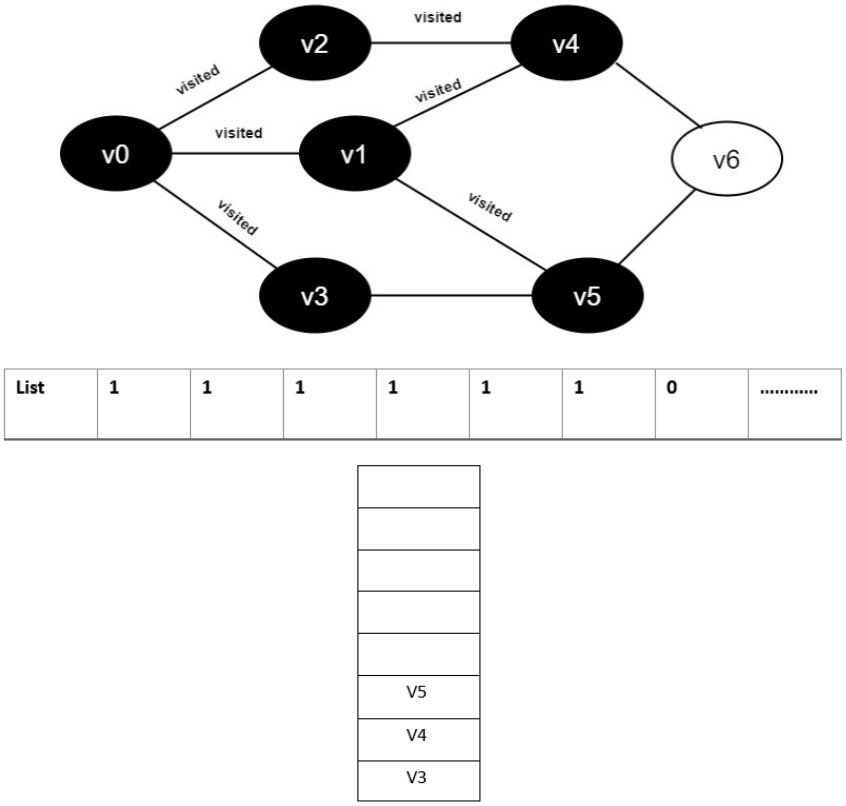


Figure 19: Access the v5 node

(8) All the neighboring points of v1 have been accessed, the team head element v3 is dequeued, all neighbors of v3 are started to access, and the v3 neighboring point v0 is started to access, because the list[0] value is 1, and no access is made. Continue to access v3 neighbor v5, because list [5] has a value of 1, no access. All the neighboring points of v3 have been accessed, the team head element v4 is dequeued, and all neighboring points of v4 are started to be accessed. Start accessing v4's neighbor v2, because the value of list [2] is 1, no access is made. Continue to access v4's neighbor v6, because list [6] has a value of 0, access v6, list [6] has a value of 1, and v6 is enqueued. All the neighboring points of v4 have been accessed, the team head element v5 is dequeued, and all neighboring points of v5 are started to be accessed. Start accessing the v5 neighbor v3, because the value of list [3] is 1, no access is made. Continue to access v5 neighbor v6, because the value of list [6] is 1, no access is made. All the neighbors of v5 have been accessed, the team head element v6 is dequeued,

and all neighbors of v6 are started to access. Start accessing the v6 neighbor v4, because the value of visited [4] is 1, no access is made. Continue to access the v6 neighbor v5, because the value of visited [5] is 1, and no access is made. The queue is empty, exits the loop, and all vertices are accessed.

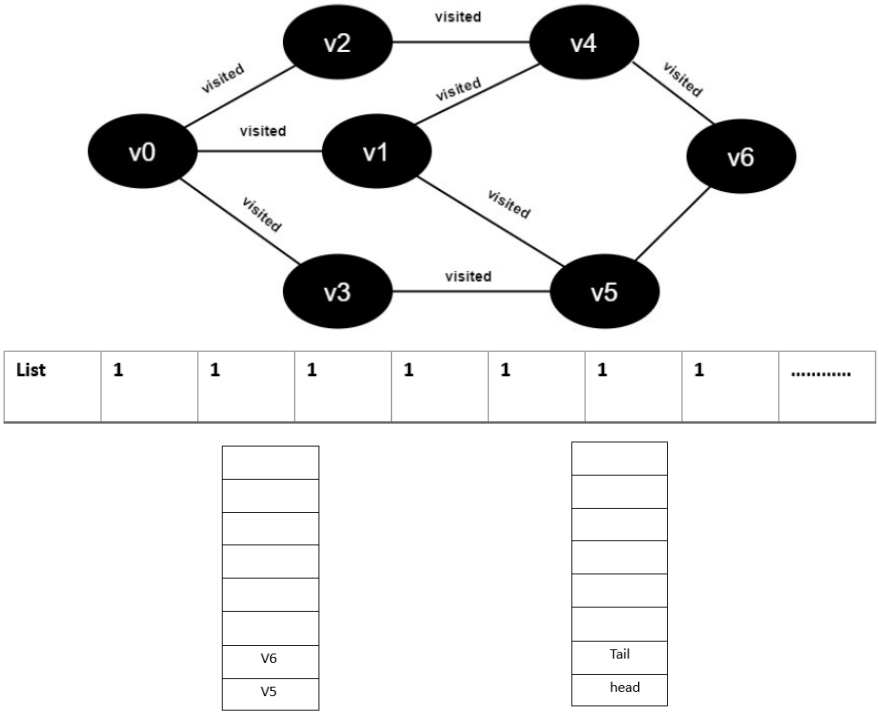


Figure 20: Access the node, Complete all traversals

(9) Using breadth-first to implement Wikipedia site title text crawling, In this program, I added a python timer to record the time it takes for the entire process. The timing is in seconds and the program is run with anaconda, the code implementation results are as follows:

1 crawl Digital_object_identifier

2 crawl Rignt_to_privacy

3 crawl Venda_Wikipedia

4 crawl Freedom_of_panorana

0 crawl Chuvash_Wikipedia

1 cranl David_Weinberger

2 cranl Peter_Stone_(professor)

3 crawl Litigation_involving_the_Wikimedia_Foundation

4 crawl Sicilian_Wikipedia

totally cost 273.05155420303345

Result 2: The result Breadth-first search algorithm implementation

4 Conclusion

Through the comparison of the results of the two crawling algorithms, we can clearly find that the breadth-first multi-threaded search algorithm is far superior to the depth-first recursive crawling method in crawling large-scale text files regardless of time or efficiency. In the single-threaded crawling mode, the python timer records a total of 1000.29163 seconds from the start of the process to the end of the process, which is about 16 minutes and a half, and we look at the result of multi-threaded crawling, python The timer's timing is 273.05155 seconds, which is about 4 minutes and a half. In contrast, the single-thread time is 4 times that of multi-threading. It can be seen that multi-threading has a large-scale crawl. The crucial impact is that it not only saves crawl time, but also provides a convenient, simple, efficient and flexible crawling tool for many people who do data analysis, data mining, and data science projects. However, multithreading also has its own limitations. Not all projects are suitable for multi-threading, or it depends on individual project needs and internship process. The original intention of writing this article is to solve the problem of Wikipedia data crawling, and to crawl Wikipedia data information in the least amount of time, but with in-depth research, multi-threading and multi-process are often accompanied by more There are many extension applications in the process, so as long as we are willing to explore and are willing to delve into it, every knowledge point can be surprisingly discovered.

Acknowledgement: This research is funded by the Open Foundation for the University Innovation Platform in the Hunan Province, grant number 16K013; Hunan Provincial Natural Science Foundation of China, grant number 2017JJ2016; 2016 Science Research Project of Hunan Provincial Department of Education, grant number 16C0269. Accurate crawler design and implementation with a data cleaning function, National Students innovation and entrepreneurship of training program, grant number 201811532010. This research work is implemented at the 2011 Collaborative Innovation Center for Development and Utilization of Finance and Economics Big Data Property, Universities of Hunan Province. Open Foundation for the University Innovation Platform in the Hunan Province, grant number 16K013; Hunan Provincial Natural Science Foundation of China, grant number 2017JJ2016; 2016 Science Research Project of Hunan Provincial Department of Education, grant number 16C0269. This research work is implemented at the 2011 Collaborative Innovation Center for Development and Utilization of Finance and Economics Big Data Property, Universities of Hunan Province. Open project, grant number 20181901CRP03, 20181901CRP04, 20181901CRP05.

References

Akbarov, S. D.; Guliyev, H. H.; Sevdimaliyev, Y. M.; Yahnioglu, N. (2018): The discrete-analytical solution method for investigation dynamics of the sphere with inhomogeneous initial stresses. *Computers, Materials & Continua*, vol. 55, no. 2, pp. 359-380.

Bohme, T.; Rahm, E. (2004): Supporting efficient streaming and insertion of XML data in RDBMS. *Third International Workshop on Data Integration over the Web.*

Calcote, J. (1997): Thread pools and server performance. *Dr. Dobb's Journal*, vol. 20, no. 7, pp. 60-64.

Chen, M. (2015): *Big Data Visualization Analysis*. Zhejiang University Press.

Chen, M. (2015): Visualization technology research based on complex data processing. *Computer and Digital Engineering*, vol. 43, no. 11, pp. 194-197.

Cheng, J.; Xu, R. M.; Tang, X. Y.; Sheng, V. S.; Cai, C. T. (2018): An abnormal network flow feature sequence prediction approach for DDoS attacks detection in big data environment. *Computers, Materials & Continua*, vol. 55, no. 1, pp. 95-119.

Farhan, A. M. (2017): Effect of rotation on the propagation of waves in hollow poroelastic circular cylinder with magnetic field. *Computers, Materials & Continua*, vol. 53, no. 2, pp. 129-156.

Geng, P.; Yan, J. (2013): Design and analysis of mutual exclusion semaphores of time-sharing partition operating system. *Computer Technology and Development*.

Han, M. (2012): Comm on problems and solutions in multithreaded programming. *Computer CD Software and Application*.

Jiang, S.; Huang, K.; Lu, Y. (2016): Design and implementation of professional web crawler based on Python. *ICCCS 2018: Cloud Computing and Security*, pp. 427-435.

Kang, D.; Han, S.; Yoo, S.; Park, S. (2008): Prediction-based dynamic threadpool scheme for efficient resource usage. *IEEE 8th International Conference on Computer and Information Technology Workshops*.

Laudon, J.; Spracklen, L. (2007): The coming wave of multithreaded chip multiprocessors. *International Journal of Parallel Programming*, vol. 35, no. 3, pp. 299-330.

Li, J. (2008): *Research and Design of Focused Reptile Algorithm Based on Web Community Recognition*. Hangzhou: Zhejiang University.

Li, L. (2017): Design and implementation of python-based web crawler system. *Journal of Information Communication*.

Ling, Y.; Mullen, T.; Lin, X. (2000): Analysis of optimal thread pool size. *ACM SIGOPS Operating System Review*, vol. 34, no. 2, pp. 42-55.

Liu, J.; Lu, Y. (2007): Summary of research on topic web reptiles. *Computer Applied Research*.

Liu, J.; Liu, G.; Zhang, Y.; Lv, D. (2015): Design and implementation of network crawler bird audio data acquisition system based on multi-threading and translation. *Modern Computer*.

Liu, S.; Yue, S. (2017): Python-based network information crawling technology in the era of big data.

Lu, S. (2019): Design and Implementation of web crawler based on Python. *Computer Programming Skills & Maintenance*.

Mao, G. (2006): Development and application of multi-threaded download tools. *Computer Applications and Software*.

Richard Stevens, W. (2004): *UNIX Network Programming*. Prentice Hall.

Richard, L. (2016): *Writing Web Crawlers in Python*. Beijing: People's Posts and Telecommunications Press.

Schmidt, D. G.; Kuhns, F. (2005): An overview of the real-time CORBA specification. *Computer*, vol. 33, no. 6, pp. 56-63.

Shao, X. (2019): Design and implementation of multi-threaded concurrent web crawler. *Modern Computer*, vol. 1.

Tanenbaum, A. S. (2009): *Modern Operating System*. Beijing: Mechanical Industry Press.

Yang, K.; Liu, Q.; Xu, T. (2010): Research on multithreading concurrency control technology of thread pool. *Computer Applications and Software*.

Yang, W. J.; Dong, P. P.; Tang, W. S.; Lou, X. P.; Zhou, H. J. et al. (2018): A MPTCP scheduler for web transfer. *Computers, Materials & Continua*, vol. 57, no. 2, pp. 20.

Zhao, M. (2013): *Research and Implementation of Web Crawler System*. University of Electronic Science and Technology of China.

Zhou, L.; Lin, L. (2005): Summary of research on focusing reptile technology. *Computer Applications*.

Zhu, W. (2009): Design and implementation of multi-thread based super node crawling algorithm.