CAN201 CW2 Report

Chengrui Zhang

1823142

1. Abstract

This coursework's aim is to implement the Bellman-Ford algorithm to every node to exchange distance data. UDP socket are used to exchange information. Nodes use same code to simulate routers. This application successfully meet all requirements in this project.

Key words: Markdown, routing algorithm, Bellman-Ford, multi-threading.

2. Introduction

2.1 Project requirement

The coursework requirements are as follows:

1. Implement Bellman-Ford algorithm to get minimum distance.
2. For each node, input is `json` file which is distance between its neighbors.
3. Application on an unfixed number of nodes should be adaptive to face.
4. UDP socket is used to exchange distance information among neighbors.
5. There should be an output file to document the results as the algorithm converges.

2.2 Background

The Bellman-Ford algorithm is an algorithm which is used extensively in network programming to compute the shortest path between nodes [1] [2]. It can accommodate negative distances and the outcome would approximate the globe's shortest path after several iterations [3].

2.3 Contribution

In this project, the author has achieved a successful implementation of the Bellman-ford distance vector routing algorithm. The application reads a `json` description file and for each node, a process is created to collaboratively find the respective shortest local DV table. Finally, after the application is completed, a set of `json` files is created to save the corresponding local DV tables.

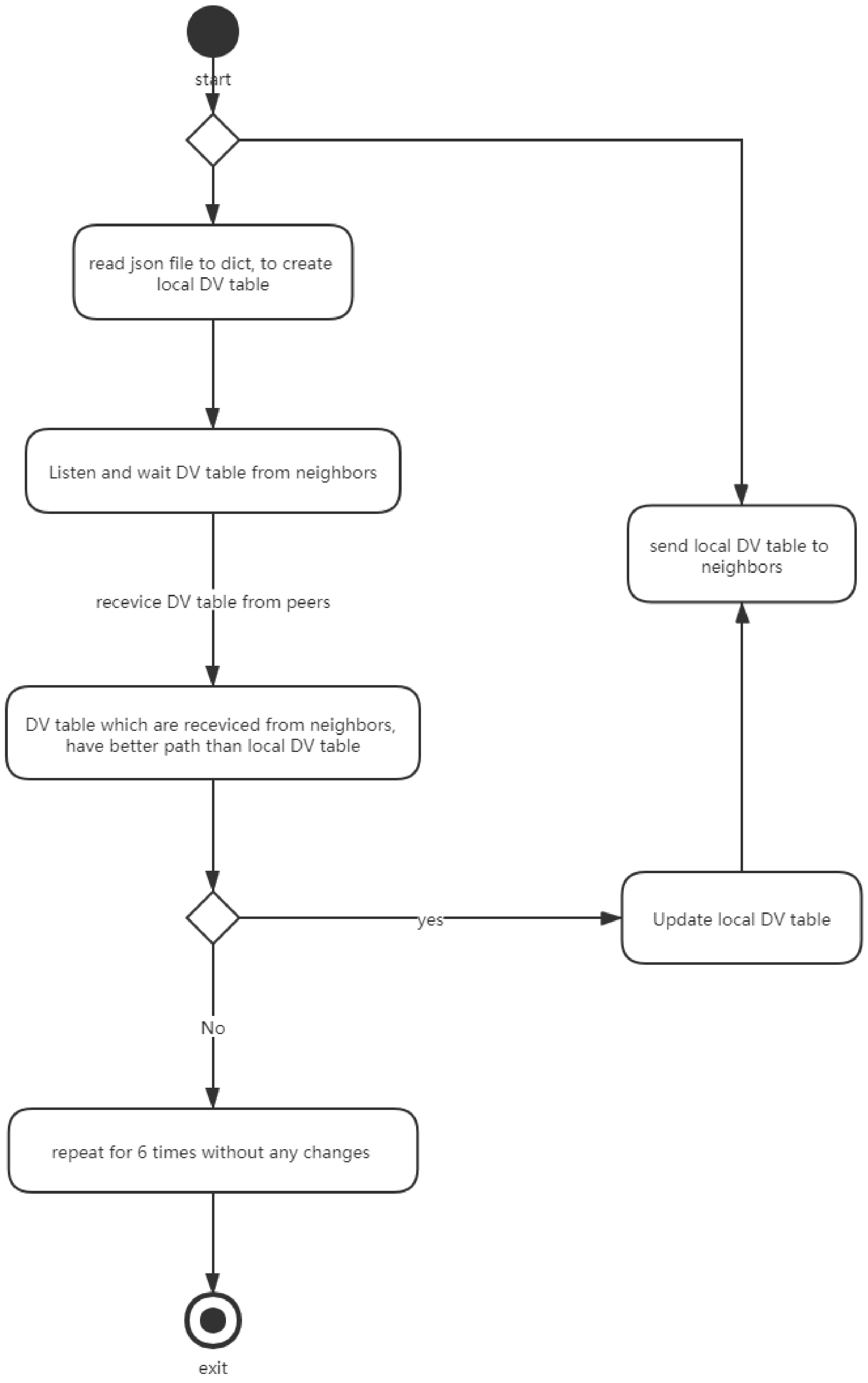
3. Methodology

3.1 Proposed protocol

This application consists primarily of two parts. The aim of the information sharing information is to connect with neighbors, to receive and submit status information. The algorithm portion seeks to get the best route (best answer for routing). The following are the protocols for this application:

1. Network part:
    1. Use UDP socket to communicate with neighbors.
    2. Bind `ip` and `port` in the configuration file of nodes.
    3. Send local DV table in `json` format. `Json` is simple and easy format used in python.
2. Algorithm part:
    1. Compare the distance between the local DV table and the DV table retrieved.

2. Using the smaller collection and place it at the local DV table.
3. If the local DV table has been updated then it send it to neighbors.
4. Repeat until there are never going to be any adjustments.
5. The convergence and output final outcome is assumed to be 6 rounds without any modifications.


start
read json file to dict, to create local DV table
Listen and wait DV table from neighbors
recevice DV table from peers
DV table which are receviced from neighbors, have better path than local DV table
yes
Update local DV table
send local DV table to neighbors
No
repeat for 6 times without any changes
exit

4. Implementation

4.1 Steps of implementation

My steps to implement this coursework are following steps:

1. Classify the specifications for the coursework and recognize input and output.
2. Start to categorize the relevant functions and modules. This application has 2 primary components: network part and algorithm part which are mentioned above.
3. The following are general functions:
    1. Receive and send of socket,
    2. Bellman-Ford algorithm,
    3. convert between `dict` and `json`,
    4. Read from and write to `json`,
4. Testing and debugging.

4.2 Difficulties met

The last one deals with the "next hop". The software would record the next hop in the first version as the node which sent the message. This isn't the case, though. For example, the shortest path from u to v may be path u - w - v. The next hop from u may be w instead of v as node v sends a message to v. The final solution is that if a path is modified, the path length in the local DV table is first tested against the origin length, and if the previous hop is shorter, the next hop from the local DV table node to the corresponding neighbor is checked against the origin length. The node that should be modified should be the next hop on the local stack.

5. Testing and results

5.1 Testing environment

`tinycore linux` in `Virtual box` in `Windows 10`

Linux system version

```
tc@box:~$ cat /proc/version
Linux version 5.4.3-tinycore (tc@box) (gcc version 9.2.0 (GCC)) #2020 SMP Tue Dec 17 17:00:50 UTC 2019
```

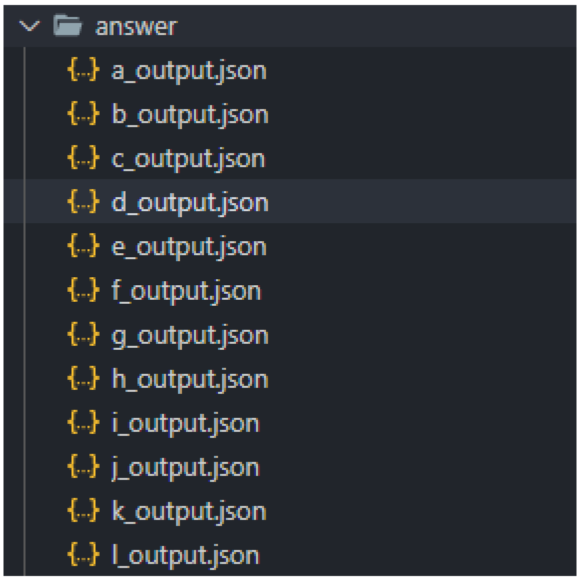
This is professor provide standard linux version for coursework.

5.2 Testing plan

5.2.1 Pre test

I and my friends create a test script to test this application. Now it open on github. It is coded in `shell` and `python`. This test could support at most 11 nodes to simulate routes actions.

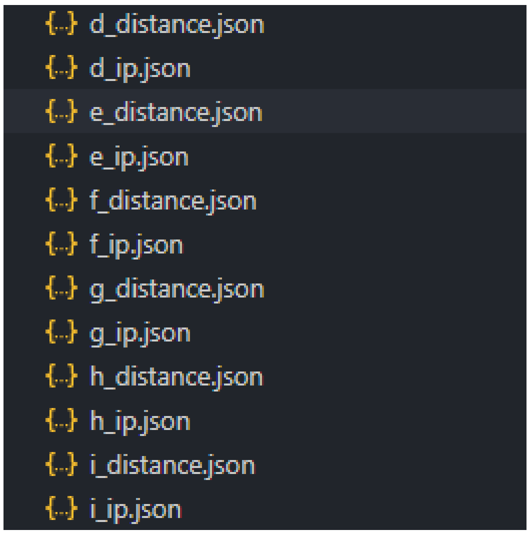
1. Create a routers map and calculate all answers for every node.



```
answer
  a_output.json
  b_output.json
  c_output.json
  d_output.json
  e_output.json
  f_output.json
  g_output.json
  h_output.json
  i_output.json
  j_output.json
  k_output.json
  l_output.json
```

Image: output answers

2. Create distance files for every node.



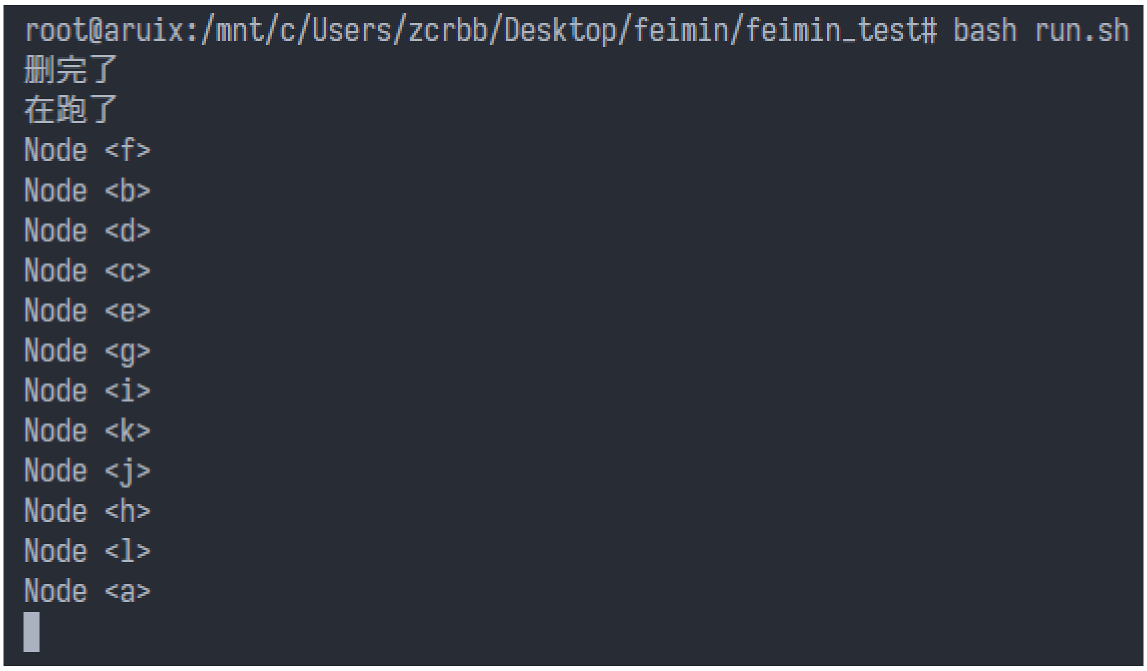
```
d_distance.json
d_ip.json
e_distance.json
e_ip.json
f_distance.json
f_ip.json
g_distance.json
g_ip.json
h_distance.json
h_ip.json
i_distance.json
i_ip.json
```

Image: part of distance of nodes

3. Create a `run.sh` to manage all nodes processes which start applications up at same time.
4. Create `check.py` to check all output distance as same as answers. If output distance different with answers, it will give exceptions.
5. Create `init.py` to delete all output files to init state.

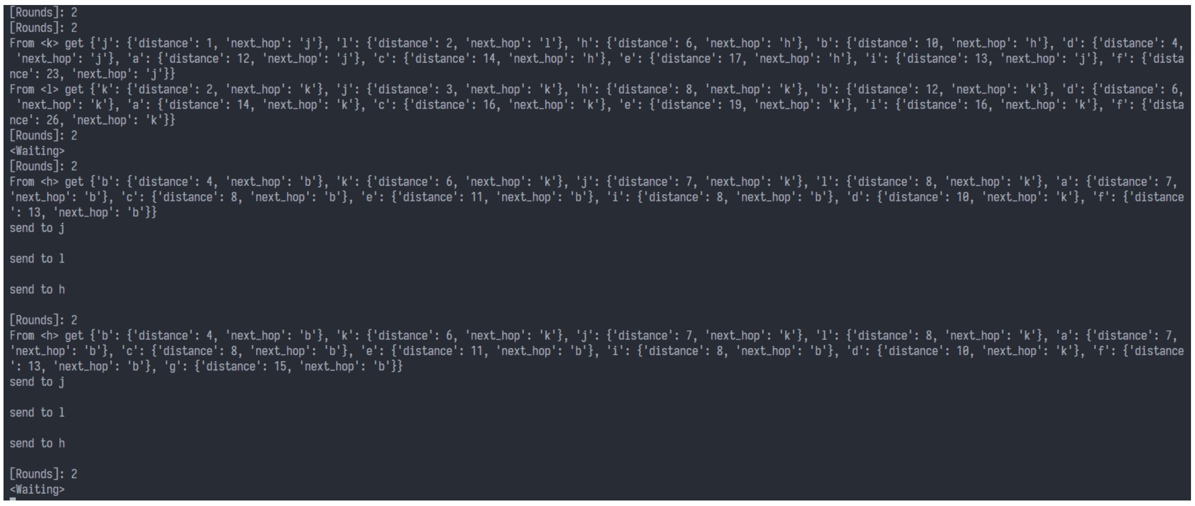
5.2.2 Testing

When nodes start to exchange distance table.



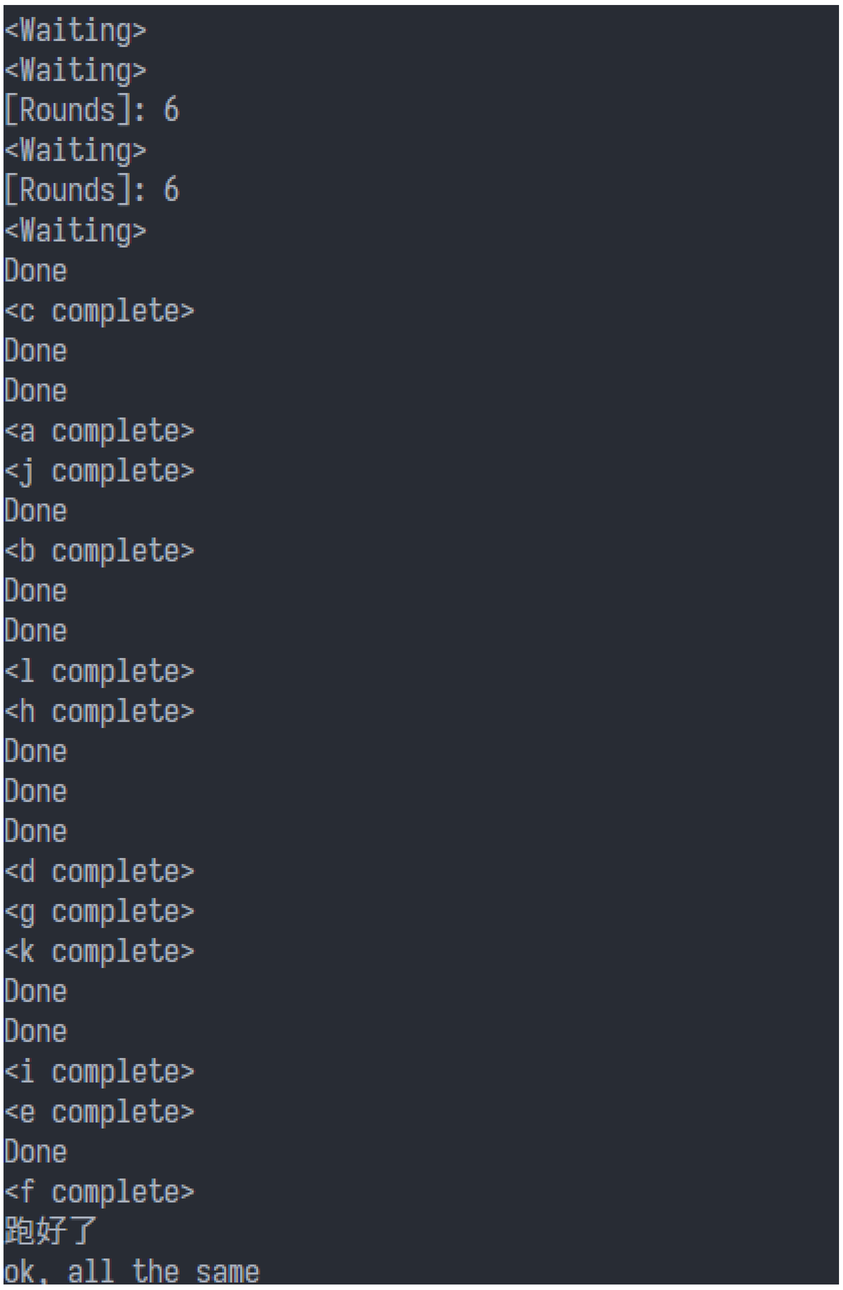
```
root@aruix:/mnt/c/Users/zcrbb/Desktop/feimin/feimin_test# bash run.sh
删完了
在跑了
Node <f>
Node <b>
Node <d>
Node <c>
Node <e>
Node <g>
Node <i>
Node <k>
Node <j>
Node <h>
Node <l>
Node <a>
```

Image 1: start to test



```
[Rounds]: 2
[Rounds]: 2
From <k> get {'j': {'distance': 1, 'next_hop': 'j'}, 'l': {'distance': 2, 'next_hop': 'l'}, 'h': {'distance': 6, 'next_hop': 'h'}, 'b': {'distance': 10, 'next_hop': 'h'}, 'd': {'distance': 4,
 'next_hop': 'j'}, 'a': {'distance': 12, 'next_hop': 'j'}, 'c': {'distance': 14, 'next_hop': 'h'}, 'e': {'distance': 17, 'next_hop': 'h'}, 'i': {'distance': 13, 'next_hop': 'j'}, 'f': {'dista
nce': 23, 'next_hop': 'j'}}
From <l> get {'k': {'distance': 2, 'next_hop': 'k'}, 'j': {'distance': 3, 'next_hop': 'k'}, 'h': {'distance': 8, 'next_hop': 'k'}, 'b': {'distance': 12, 'next_hop': 'k'}, 'd': {'distance': 6,
 'next_hop': 'k'}, 'a': {'distance': 14, 'next_hop': 'k'}, 'c': {'distance': 16, 'next_hop': 'k'}, 'e': {'distance': 19, 'next_hop': 'k'}, 'i': {'distance': 16, 'next_hop': 'k'}, 'f': {'dista
nce': 26, 'next_hop': 'k'}}
[Rounds]: 2
<Waiting>
[Rounds]: 2
From <h> get {'b': {'distance': 4, 'next_hop': 'b'}, 'k': {'distance': 6, 'next_hop': 'k'}, 'j': {'distance': 7, 'next_hop': 'k'}, 'l': {'distance': 8, 'next_hop': 'k'}, 'a': {'distance': 7,
'next_hop': 'b'}, 'c': {'distance': 8, 'next_hop': 'b'}, 'e': {'distance': 11, 'next_hop': 'b'}, 'i': {'distance': 8, 'next_hop': 'b'}, 'd': {'distance': 10, 'next_hop': 'k'}, 'f': {'distance
': 13, 'next_hop': 'b'}}
send to j

send to l

send to h

[Rounds]: 2
From <h> get {'b': {'distance': 4, 'next_hop': 'b'}, 'k': {'distance': 6, 'next_hop': 'k'}, 'j': {'distance': 7, 'next_hop': 'k'}, 'l': {'distance': 8, 'next_hop': 'k'}, 'a': {'distance': 7,
'next_hop': 'b'}, 'c': {'distance': 8, 'next_hop': 'b'}, 'e': {'distance': 11, 'next_hop': 'b'}, 'i': {'distance': 8, 'next_hop': 'b'}, 'd': {'distance': 10, 'next_hop': 'k'}, 'f': {'distance
': 13, 'next_hop': 'b'}, 'g': {'distance': 15, 'next_hop': 'b'}}
send to j

send to l

send to h

[Rounds]: 2
<Waiting>
```

Image 2: exchange distance data



```
<Waiting>
<Waiting>
[Rounds]: 6
<Waiting>
[Rounds]: 6
<Waiting>
Done
<c complete>
Done
Done
<a complete>
<j complete>
Done
<b complete>
Done
Done
<l complete>
<h complete>
Done
Done
Done
<d complete>
<g complete>
<k complete>
Done
Done
<i complete>
<e complete>
Done
<f complete>
跑好了
ok, all the same
```

Image 3: test end and check finish

If all output files are as same as answers, it will print "ok, all the same".

6. Conclusion

This project mainly implement Bellman-Ford algorithm to simulate routers' information exchange to get minimum distance and next hop. Use Bellman-Ford algorithm to get best path which means shortest path for every node. Networking part is used to communicate with its neighbors by UDP socket. This application is completely implement all requirements of this coursework.

6.1 Future plan

Use minimum time to get the shortest distance. On this stage, there must wait for 6 rounds to certify there is no update among nodes. There should have another method to decrease time occupied.

7. Reference

[1] J. F. Kurose and K. W. Ross, Computer networking : a top-down approach. Pearson, 2017. [Online]. Available: http://ez.xjtlu.edu.cn/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=cat01010a&AN=xjtlu.0000960216&site=eds-live&scope=site

[2] J. Bang-Jensen and G. Z. Gutin, Digraphs: theory, algorithms and applications. Springer Science & Business Media, 2008.

[3] R. Kavikondala Praveen Kumar and M. Tamilarasan Senthil, “An efficient routing algorithm for software defined networking using bellman ford algorithm.” International Journal of Online and Biomedical Engineering, no. 14, p. 87, 2019. [Online]. Available: http://ez.xjtlu.edu.cn/login?url=http://search.ebscohost.com/login.aspx?direct=true& db=edsdoj&AN=edsdoj.fe28093adc90425fa05d3506225d3f49&site=eds-live&scope=site