Dig Data Management Assignment 6

信科四 郑元嘉 1800920541

In this report, we've got to run 4 tasks individually implemented by 3 databases – Redis, MongoDB and Neo4j.
The 4 tasks are given below :

1. Given si，find all its P and O，<si, P, O>. In particular, we run this task 5 times to have a more reliable consumed time where si are 'Shamsuzzaman_Khan', 'Jake_Gyllenhaal', 'Rubn_Arocha', 'frBgnet', 'Cody_Kessler'.
2. Given oi, find all S and P，<S, P,oi>. In particular, we run this task 5 times to have a more reliable consumed time where oi are 'France', 'Belgium', 'Canada', 'Scotland', 'Spain'.
3. Given p1,p2, find all S which has both p1 and p2，<S, p1, *>, <S, p2, *>. In particular, the (p1, p2) pair is ('isLeaderOf', 'isCitizenOf').
4. Given oi, find the S which has the most oi. If there's more than one result, we only return one of them. In particular, we run this task 5 times to have a more reliable consumed time where oi are 'DurhamSud', 'Norway', 'Germany', 'France', 'China'

Plus, for every single task in some database, both insertion method and search method would be demonstrated below to show how we design the storage strategies and how we search based on it.

Database 1 - Redis

- **Task 1**
  Insertion Method
  (Partial Source Code)

```python
if task_index == 0:
    with open(file_path) as f:
        for line in f.readlines():
            data = line.strip().split(' ')
            if len(data) != 3:
                print('Datum error:', data)
            client.rpush(data[0], ' '.join([data[1], data[2]]))
```

Intuitively, we simply apply list key-value to store data with S as the key and (P, O) pair as the value element of the list.

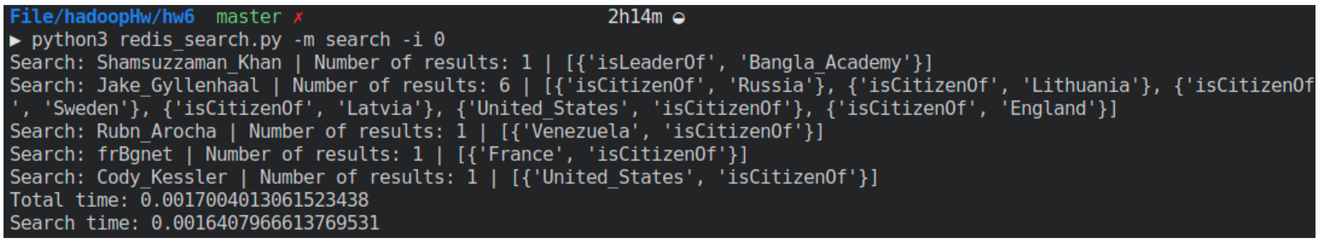
Search Method

(Partial Source Code)

```python
def search_0(redis_host, redis_port, redis_db_index):
    global timestamp
    client = redis.Redis(host=redis_host, port=redis_port, db=redis_db_inde

    timestamp.stamp('search')

    si_list = ['Shamsuzzaman_Khan', 'Jake_Gyllenhaal', 'Rubn_Arocha', 'frBg

    for si in si_list:
        res = client.lrange(si, 0, -1)
        pair_list = [set(r.split(' ')) for r in res]
        print('Search: {} | Number of results: {} | {}'.format(si, len(res)

    timestamp.stamp('search')
```

We search by the key equal to the si, retrieve all elements in its value as a list, and decompose the form of (P, O) pair.

Consumed Time :



```
File/hadoopHw/hw6  master ✗                          2h14m ◒
▶ python3 redis_search.py -m search -i 0
Search: Shamsuzzaman_Khan | Number of results: 1 | [{'isLeaderOf', 'Bangla_Academy'}]
Search: Jake_Gyllenhaal | Number of results: 6 | [{'isCitizenOf', 'Russia'}, {'isCitizenOf', 'Lithuania'}, {'isCitizenOf
', 'Sweden'}, {'isCitizenOf', 'Latvia'}, {'United_States', 'isCitizenOf'}, {'isCitizenOf', 'England'}]
Search: Rubn_Arocha | Number of results: 1 | [{'Venezuela', 'isCitizenOf'}]
Search: frBgnet | Number of results: 1 | [{'France', 'isCitizenOf'}]
Search: Cody_Kessler | Number of results: 1 | [{'United_States', 'isCitizenOf'}]
Total time: 0.0017004013061523438
Search time: 0.0016407966613769531
```

- **Task 2**

Insertion Method

(Partial Source Code)

```python
elif task_index == 1:
    with open(file_path) as f:
        for line in f.readlines():
            data = line.strip().split(' ')
            if len(data) != 3:
                print('Datum error:', data)
            client.rpush(data[2], ' '.join([data[0], data[1]]))
```

It's highly logically similar to task 1. We use O as the key and (S, P) pair as the element in the list value this time.

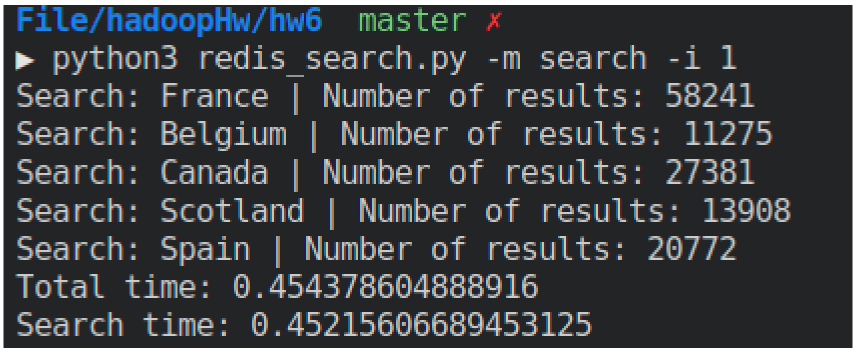
Search Method

(Partial Source Code)

```python
def search_1(redis_host, redis_port, redis_db_index):
    global timestamp
    client = redis.Redis(host=redis_host, port=redis_port, db=redis_db_inde

    timestamp.stamp('search')

    oi_list = ['France', 'Belgium', 'Canada', 'Scotland', 'Spain']

    for oi in oi_list:
        res = client.lrange(oi, 0, -1)
        pair_list = [set(r.split(' ')) for r in res]
        print('Search: {} | Number of results: {}'.format(oi, len(res)))

    timestamp.stamp('search')
```

It's almost the same as task 1 as well. Use oi as the key to retrieve the list value, and decompose the elements in it.

Consumed Time



```
File/hadoopHw/hw6  master ✗
▶ python3 redis_search.py -m search -i 1
Search: France | Number of results: 58241
Search: Belgium | Number of results: 11275
Search: Canada | Number of results: 27381
Search: Scotland | Number of results: 13908
Search: Spain | Number of results: 20772
Total time: 0.454378604888916
Search time: 0.45215606689453125
```

- **Task 3**

  Insertion Method

  (Partial Source Code)

```python
elif task_index == 2:
    with open(file_path) as f:
        for line in f.readlines():
            data = line.strip().split(' ')
            if len(data) != 3:
                print('Datum error:', data)
            client.rpush(data[1], data[0])
```

P is selected as the key this time with a list value carrying S as elements.

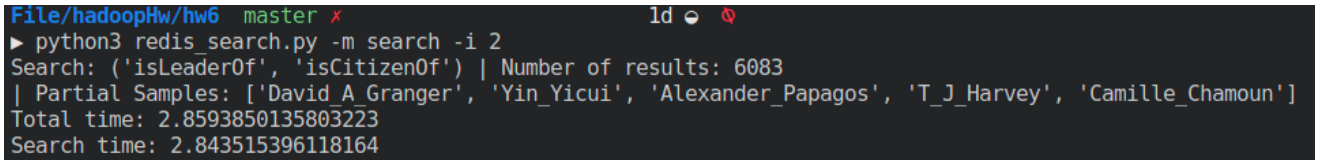
Search Method

(Partial Source Code)

```python
def search_2(redis_host, redis_port, redis_db_index):
    global timestamp
    client = redis.Redis(host=redis_host, port=redis_port, db=redis_db_inc

    timestamp.stamp('search')

    p_pair = ('isLeaderOf', 'isCitizenOf')

    res_0 = client.lrange(p_pair[0], 0, -1)
    res_1 = client.lrange(p_pair[1], 0, -1)

    res_cross = set(res_0) & set(res_1)

    print('Search: {} | Number of results: {} \n| Partial Samples: {}'.for

    timestamp.stamp('search')
```

We retrieve two values as lists by the p1 and p2 as the keys individually. Then we turn the two lists into instances of `set` to see all elements in the cross set which is our goal.

Consumed Time



```
File/hadoopHw/hw6  master ✗                          1d
▶ python3 redis_search.py -m search -i 2
Search: ('isLeaderOf', 'isCitizenOf') | Number of results: 6083
| Partial Samples: ['David_A_Granger', 'Yin_Yicui', 'Alexander_Papagos', 'T_J_Harvey', 'Camille_Chamoun']
Total time: 2.8593850135803223
Search time: 2.843515396118164
```

- **Task 4**

  Insertion Method

  (Partial Source Code)

```python
elif task_index == 3:
    with open(file_path) as f:
        for line in f.readlines():
            data = line.strip().split(' ')
            if len(data) != 3:
                print('Datum error:', data)
            client.zincrby(data[2], 1, data[0])
```

In this task, we apply `zset` type to store the data. O is the key while S is the element in `zset` value. And the score in `zset` type is taken advantage of by us to do the ranking task by the number of occurrences.

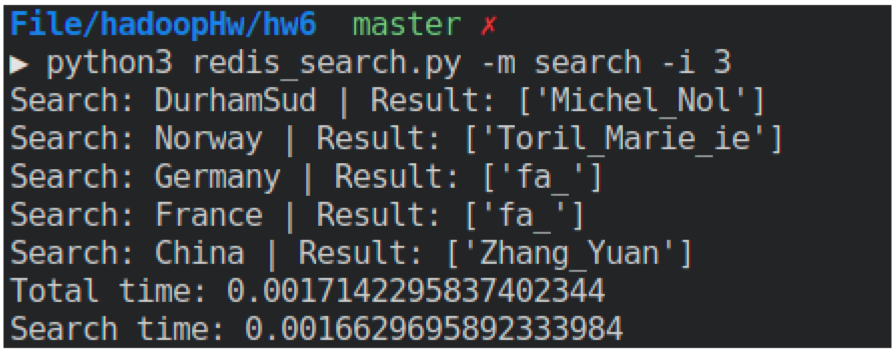
Search Method

(Partial Source Code)

```python
def search_3(redis_host, redis_port, redis_db_index):
    global timestamp
    client = redis.Redis(host=redis_host, port=redis_port, db=redis_db_inde

    timestamp.stamp('search')

    oi_list = ['DurhamSud', 'Norway', 'Germany', 'France', 'China']

    for oi in oi_list:
        res = client.zrevrange(oi, 0, 0)

        print('Search: {} | Result: {}'.format(oi, res))

    timestamp.stamp('search')
```

Thanks to the scoring property in `zset` type, we can obtain the element with highest score (number of occurrences) via mere function `zrevrange()` , and then the S which has the most oi is returned.

Consumed Time



```
File/hadoopHw/hw6  master ✗
▶ python3 redis_search.py -m search -i 3
Search: DurhamSud | Result: ['Michel_Nol']
Search: Norway | Result: ['Toril_Marie_ie']
Search: Germany | Result: ['fa_']
Search: France | Result: ['fa_']
Search: China | Result: ['Zhang_Yuan']
Total time: 0.0017142295837402344
Search time: 0.0016629695892333984
```

Database 2 - MongoDB

- **Task 1**

  Insertion Method

  (Partial Source Code)

```python
if task_index == 0:
    collection.create_index('S')
    bulk = []

    with open(file_path) as f:
        for line in f.readlines():
            doc = line.strip().split(' ')
            doc = dict([(k, v) for (k, v) in zip(['S', 'P', 'O'], doc)])
            bulk.append(doc)

    collection.insert_many(bulk)
```

We purely save a datum as a document in MongoDB with keys `S` , `P` and `O` . To speed up the search task, we create index of `S` field. Here, we can use function `insert_many()` to insert all data at once so as to reduce the transmission.

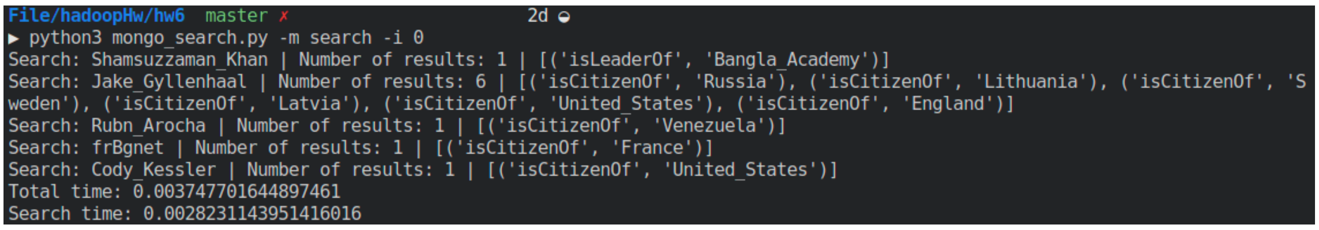
Search Method

(Partial Source Code)

```python
def search_0(db_host, db_port, db_name, db_col_name, task_index):
    global timestamp
    client = MongoClient(db_host, db_port)
    db = client[db_name]
    collection = db[db_col_name]

    si_list = ['Shamsuzzaman_Khan', 'Jake_Gyllenhaal', 'Rubn_Arocha', 'frBg

    timestamp.stamp('search')

    for si in si_list:
        cursor = collection.find({'S': si})
        res = [(_doc['P'], _doc['O']) for _doc in cursor]
        print('Search: {} | Number of results: {} | {}'.format(si, len(res)

    timestamp.stamp('search')
```

We use a simple function `find()` to search for our goal.

Consumed Time



```
File/hadoopHw/hw6  master ✗                    2d ◒
▶ python3 mongo_search.py -m search -i 0
Search: Shamsuzzaman_Khan | Number of results: 1 | [('isLeaderOf', 'Bangla_Academy')]
Search: Jake_Gyllenhaal | Number of results: 6 | [('isCitizenOf', 'Russia'), ('isCitizenOf', 'Lithuania'), ('isCitizenOf', 'S
weden'), ('isCitizenOf', 'Latvia'), ('isCitizenOf', 'United_States'), ('isCitizenOf', 'England')]
Search: Rubn_Arocha | Number of results: 1 | [('isCitizenOf', 'Venezuela')]
Search: frBgnet | Number of results: 1 | [('isCitizenOf', 'France')]
Search: Cody_Kessler | Number of results: 1 | [('isCitizenOf', 'United_States')]
Total time: 0.003747701644897461
Search time: 0.0028231143951416016
```

- **Task 2**

  Insertion Method

  (Partial Source Code)

```python
elif task_index == 1:
    collection.create_index('O')
    bulk = []

    with open(file_path) as f:
        for line in f.readlines():
            doc = line.strip().split(' ')
            doc = dict([(k, v) for (k, v) in zip(['S', 'P', 'O'], doc)])
            bulk.append(doc)

    collection.insert_many(bulk)
```

It's almost similar to Task 1 while we set field `O` as index here.

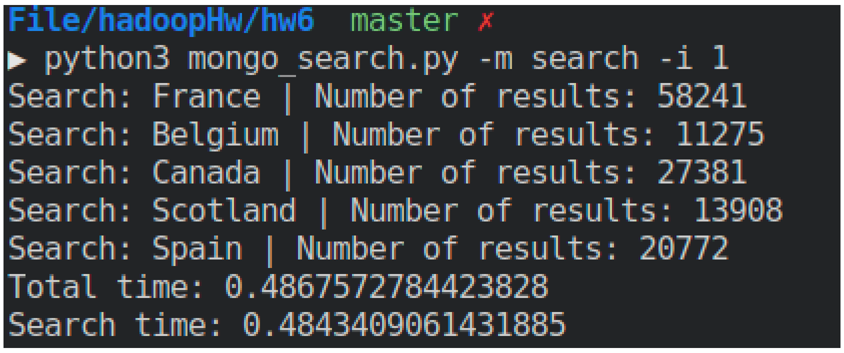
Search Method

(Partial Source Code)

```python
def search_1(db_host, db_port, db_name, db_col_name, task_index):
    global timestamp
    client = MongoClient(db_host, db_port)
    db = client[db_name]
    collection = db[db_col_name]

    oi_list = ['France', 'Belgium', 'Canada', 'Scotland', 'Spain']

    timestamp.stamp('search')

    for oi in oi_list:
        cursor = collection.find({'O': oi})
        res = [(_doc['S'], _doc['P']) for _doc in cursor]
        print('Search: {} | Number of results: {}'.format(oi, len(res)))

    timestamp.stamp('search')
```

Doing the same thing as the one in Task 1, we use function `find()` and use `O` as our condition,

Consumed Time



```
File/hadoopHw/hw6  master ✗
▶ python3 mongo_search.py -m search -i 1
Search: France | Number of results: 58241
Search: Belgium | Number of results: 11275
Search: Canada | Number of results: 27381
Search: Scotland | Number of results: 13908
Search: Spain | Number of results: 20772
Total time: 0.4867572784423828
Search time: 0.4843409061431885
```

- **Task 3**

  Insertion Method

  (Partial Source Code)

```python
elif task_index == 2:
    collection.create_index('S', unique=True)
    collection.create_index('P_list')
    bulk = []

    with open(file_path) as f:
        for line in f.readlines():
            doc = line.strip().split(' ')
            bulk.append(UpdateMany({'S': doc[0]}, {'$addToSet': {'P_list':
```

```

    collection.bulk_write(bulk)
```

Here’s something different. We save data as a document with fields `S` and `P_list` where all Ps would be saved into a list in the same document by operation `$addToSet` and unique index `S` . To speed up update work, we have class `UpdateMany` and function `bulk_write()` to give us a favor, which allows us to send mere one request to MongoDB and run multiple operations.

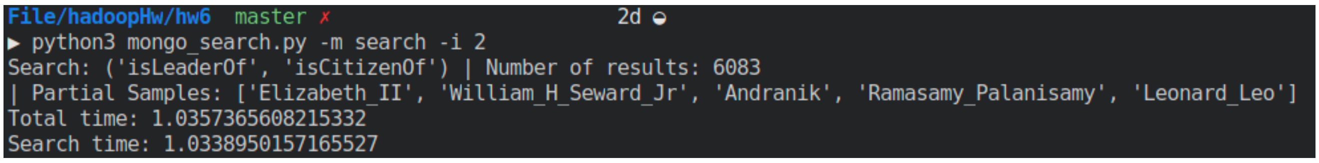
Search Method

(Partial Source Code)

```
def search_2(db_host, db_port, db_name, db_col_name, task_index):
    global timestamp
    client = MongoClient(db_host, db_port)
    db = client[db_name]
    collection = db[db_col_name]

    p_pair = ('isLeaderOf', 'isCitizenOf')

    timestamp.stamp('search')

    cursor = collection.find({'P_list': {'$all': list(p_pair)}})
    res = [_doc['S'] for _doc in cursor]
    print('Search: {} | Number of results: {} \n| Partial Samples: {}'.form

    timestamp.stamp('search')
```

We use MongoDB list operation `$all$` to check whether the elements in `P_list` can match all of ones in our query list. By so, we can retrieve all matched documents with ease.

Consumed Time



```
File/hadoopHw/hw6  master ✗                          2d ⊖
▶ python3 mongo_search.py -m search -i 2
Search: ('isLeaderOf', 'isCitizenOf') | Number of results: 6083
| Partial Samples: ['Elizabeth_II', 'William_H_Seward_Jr', 'Andranik', 'Ramasamy_Palanisamy', 'Leonard_Leo']
Total time: 1.0357365608215332
Search time: 1.0338950157165527
```

- **Task 4**

  Insertion Method

  (Partial Source Code)

```
elif task_index == 3:
    collection.create_index('O')
    # create compound index
    collection.create_index([('O', pymongo.ASCENDING), ('S', pymongo.ASCEND
    bulk = []

    with open(file_path) as f:
        for line in f.readlines():
            doc = line.strip().split(' ')
            bulk.append(UpdateOne({'O': doc[2], 'S': doc[0]}, {'$inc': {'co

    collection.bulk_write(bulk)

timestamp.stamp('insert')
```

This time, we save a datum as a document with fields `O` , `S` and `count` . Fields `O` and `S` work as the index for searching with field `count` denotes the number of occurrence of the (`O` , `S`) pair. We create a unique compound index consisting of `O` and `S` to ensure the uniqueness and to enable us to use class `UpdateOne` rather than `UpdateMany` furthermore. Also, we use function `bulk_write()` to speed up.

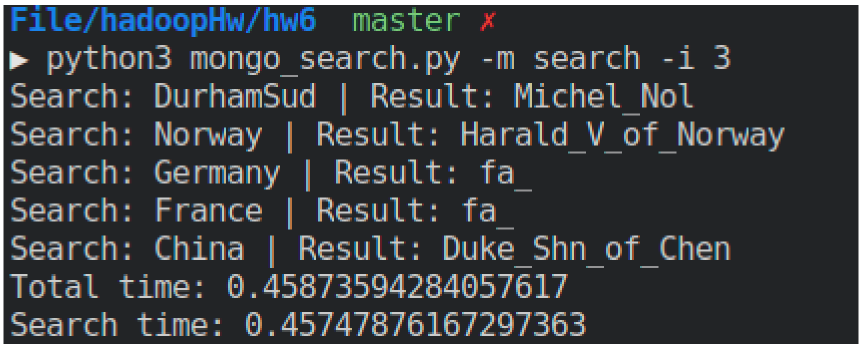
Search Method

(Partial Source Code)

```
def search_3(db_host, db_port, db_name, db_col_name, task_index):
  global timestamp
  client = MongoClient(db_host, db_port)
  db = client[db_name]
  collection = db[db_col_name]

  oi_list = ['DurhamSud', 'Norway', 'Germany', 'France', 'China']

  timestamp.stamp('search')

  for oi in oi_list:
      cursor = collection.aggregate([
          {'$match': {'O': oi}},
          {'$sort': {'count':-1}},
          {'$limit': 1},
      ])

      res = list(cursor)[0]['S']
      print('Search: {} | Result: {}'.format(oi, res))

  timestamp.stamp('search')
```

To find the document with maximal value of `count` , we adapt an aggregation pipeline. The first step `$match` is simply to filter and get the candidates, and the second step `$sort` as well as third step `$limit` help us to obtain the first element in a list in descendent order by `count` .

Consumed Time



```
File/hadoopHw/hw6  master ✗
▶ python3 mongo_search.py -m search -i 3
Search: DurhamSud | Result: Michel_Nol
Search: Norway | Result: Harald_V_of_Norway
Search: Germany | Result: fa_
Search: France | Result: fa_
Search: China | Result: Duke_Shn_of_Chen
Total time: 0.45873594284057617
Search time: 0.45747876167297363
```

Database 3 - Neo4j

- **Task 1**

  Insertion Method

  (Partial Source Code)

```python
if task_index == 0:
    statement = 'CREATE (:set1{s: {s}, p: {p}, o: {o}})'
    with driver.session() as session:
        with open(file_path) as f:
            for i, line in enumerate(f.readlines()):
                if i % 100 == 0: print('{} rounds'.format(i))
                doc = line.strip().split(' ')
                session.run(statement, s=doc[0], p=doc[1], o=doc[2])
```

  Because the speed of creating nodes in Neo4j is dramatically slow, we apply this simplest way to all 4 tasks. We create one node with 3 fields `S` , `P` and `O` . Therefore, there's no distinct insertion method going to be demonstrated below.

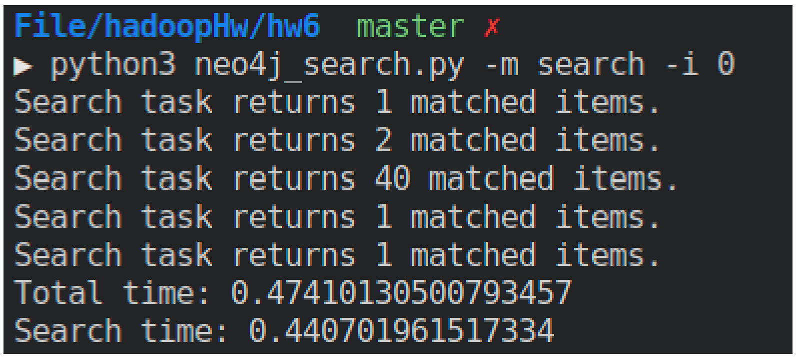
  Search Method

  (Partial Source Code)

```python
def search_0(file_path, uri, acc, pwd, task_index):
    driver = GraphDatabase.driver(uri, auth=(acc, pwd))

    global timestamp
    timestamp.stamp('search')

    si_list = ['Shamsuzzaman_Khan', 'Spencer_Chandra_Herbert', 'Minnesota_P
    statement = 'MATCH (set:set1{s: {s}}) RETURN set'

    for si in si_list:
        with driver.session() as session:
            res = session.run(statement, s=si)
            data = list(res.records())
            print('Search task returns {} matched items.'.format(len(data))

    timestamp.stamp('search')
```

It's a simple search task. We only apply the condition to retrieve our wanted data.

Consumed Time



```
File/hadoopHw/hw6  master ✗
▶ python3 neo4j_search.py -m search -i 0
Search task returns 1 matched items.
Search task returns 2 matched items.
Search task returns 40 matched items.
Search task returns 1 matched items.
Search task returns 1 matched items.
Total time: 0.47410130500793457
Search time: 0.440701961517334
```

- **Task 2**

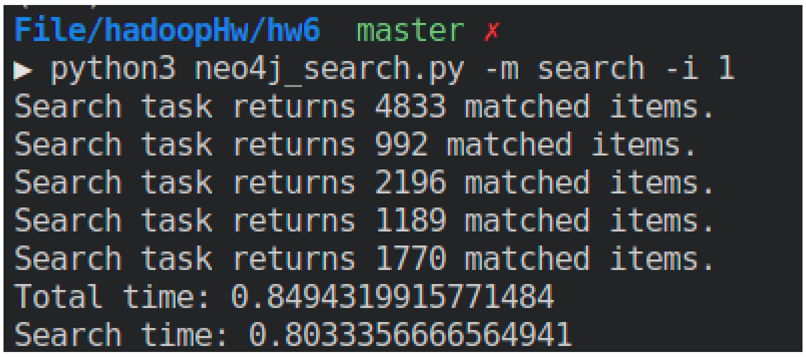
  Search Method

  (Partial Source Code)

```python
def search_1(file_path, uri, acc, pwd, task_index):
    driver = GraphDatabase.driver(uri, auth=(acc, pwd))

    global timestamp
    timestamp.stamp('search')

    oi_list = oi_list = ['France', 'Belgium', 'Canada', 'Scotland', 'Spain'
    statement = 'MATCH (set:set1{o: {o}}) RETURN set'

    for oi in oi_list:
        with driver.session() as session:
            res = session.run(statement, o=oi)
            data = list(res.records())
            print('Search task returns {} matched items.'.format(len(data))

    timestamp.stamp('search')
```

It's the same as Task 1.

Consumed Time



```
File/hadoopHw/hw6  master ✗
▶ python3 neo4j_search.py -m search -i 1
Search task returns 4833 matched items.
Search task returns 992 matched items.
Search task returns 2196 matched items.
Search task returns 1189 matched items.
Search task returns 1770 matched items.
Total time: 0.8494319915771484
Search time: 0.8033356666564941
```

- **Task 3**

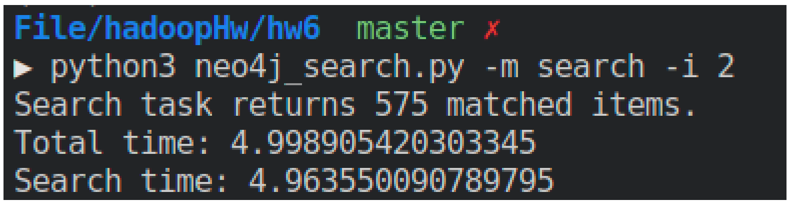
  Search Method

  (Partial Source Code)

```python
def search_2(file_path, uri, acc, pwd, task_index):
    driver = GraphDatabase.driver(uri, auth=(acc, pwd))

    global timestamp
    timestamp.stamp('search')

    p_pair = ('isLeaderOf', 'isCitizenOf')
    statement = 'MATCH (set:set1{p: {p}}) RETURN set'

    with driver.session() as session:
        res_0 = session.run(statement, p=p_pair[0])
        res_1 = session.run(statement, p=p_pair[1])
        data_0 = set([item['set']['s'] for item in list(res_0.records())])
        data_1 = set([item['set']['s'] for item in list(res_1.records())])
        print('Search task returns {} matched items.'.format(len(data_0 & d

    timestamp.stamp('search')
```

  Here, we use the simple way to get two set according to p1 and p2, and then we extract the cross get by manipulating class `set` to get the final result.

  Consumed Time



```
File/hadoopHw/hw6  master ✗
▶ python3 neo4j_search.py -m search -i 2
Search task returns 575 matched items.
Total time: 4.998905420303345
Search time: 4.963550090789795
```

- **Task 4**

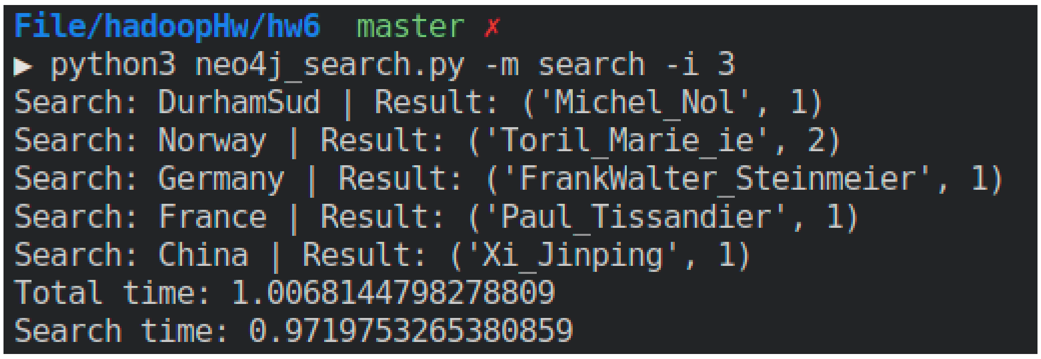
  Search Method

  (Partial Source Code)

```python
def search_3(file_path, uri, acc, pwd, task_index):
    driver = GraphDatabase.driver(uri, auth=(acc, pwd))

    global timestamp
    timestamp.stamp('search')

    oi_list = ['DurhamSud', 'Norway', 'Germany', 'France', 'China']
    statement = 'MATCH (set:set1{o: {o}}) RETURN set'

    for oi in oi_list:
        with driver.session() as session:
            res = session.run(statement, o=oi)
            data = [item['set']['s'] for item in list(res.records())]
            counter = Counter(data)
            max_ele = sorted(counter.items(), key=lambda x: x[1], reverse=
            print('Search: {} | Result: {}'.format(oi, max_ele))

    timestamp.stamp('search')
```

We get the matched data back, and do the sorting task by number of concurrence in aid of class `Counter` and function `sort()` to achieve our goal.

Consumed Time



```
File/hadoopHw/hw6  master ✗
▶ python3 neo4j_search.py -m search -i 3
Search: DurhamSud | Result: ('Michel_Nol', 1)
Search: Norway | Result: ('Toril_Marie_ie', 2)
Search: Germany | Result: ('FrankWalter_Steinmeier', 1)
Search: France | Result: ('Paul_Tissandier', 1)
Search: China | Result: ('Xi_Jinping', 1)
Total time: 1.0068144798278809
Search time: 0.9719753265380859
```