Implementing MapReduce Programming Framework using ZeroMQ sockets

CS403/534 - Distributed Systems

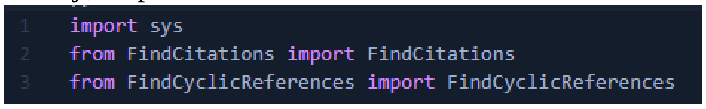
25 November 2023

PA2 Solution Report

Ataollah Hosseinzadeh Fard
ID: 28610

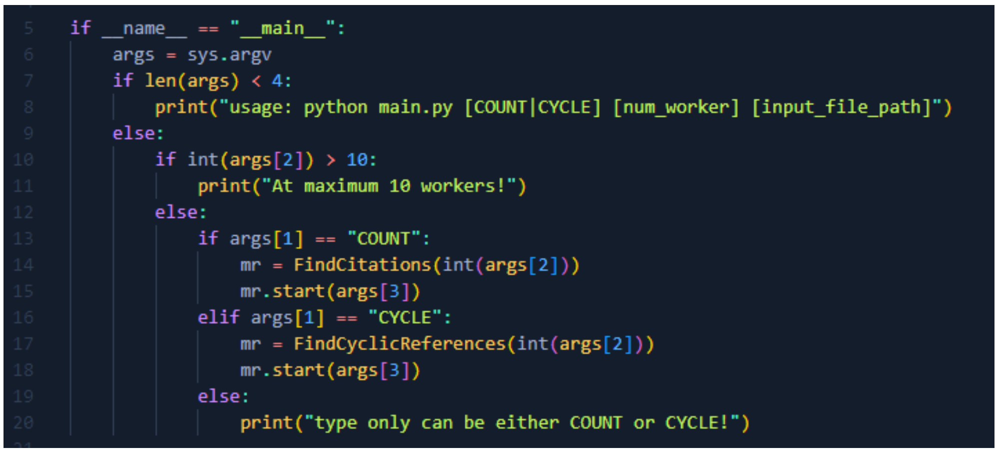
main.py

1) Firstly, imported the needed modules.



```python
import sys
from FindCitations import FindCitations
from FindCyclicReferences import FindCyclicReferences
```

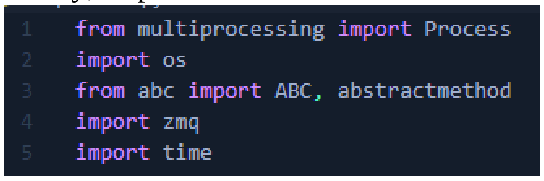
2) Then, the program looks for given arguments, and if argument count is less than 4, it gives an error and displays usage guide to user. If argument count is greater or equal to 4 then runs the program and parses arguments into number of workers to run, type of MapReduce to run and path to txt file that holds initial data to begin with. But before checking the ‘args[1]’ is COUNT or CYCLE, it checks if number of workers is valid, which means if it is less than or equal to 10.



```python
if __name__ == "__main__":
    args = sys.argv
    if len(args) < 4:
        print("usage: python main.py [COUNT|CYCLE] [num_worker] [input_file_path]")
    else:
        if int(args[2]) > 10:
            print("At maximum 10 workers!")
        else:
            if args[1] == "COUNT":
                mr = FindCitations(int(args[2]))
                mr.start(args[3])
            elif args[1] == "CYCLE":
                mr = FindCyclicReferences(int(args[2]))
                mr.start(args[3])
            else:
                print("type only can be either COUNT or CYCLE!")
```

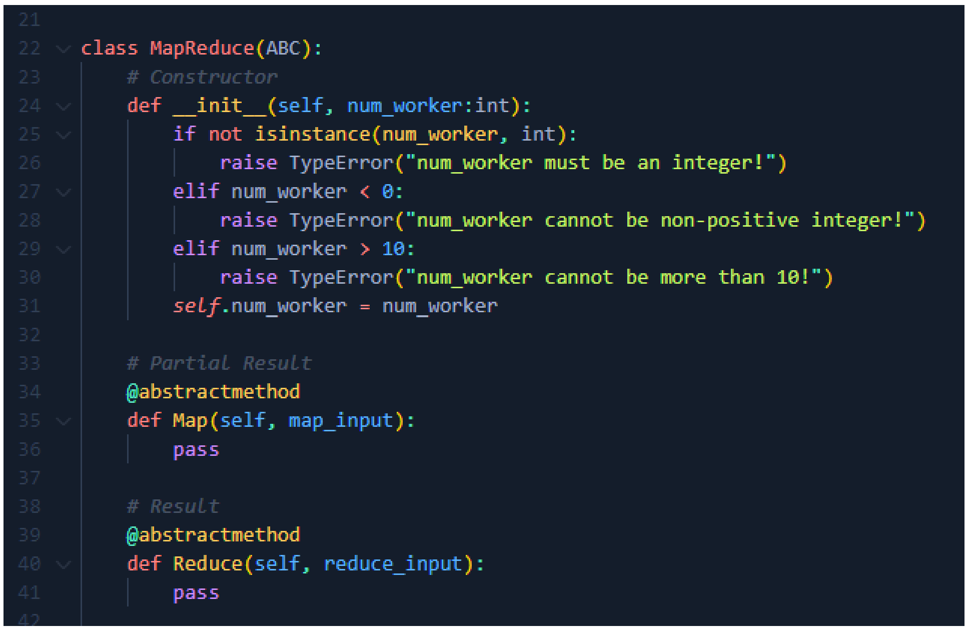
MapReduce.py

1) Firstly, imported the needed modules.



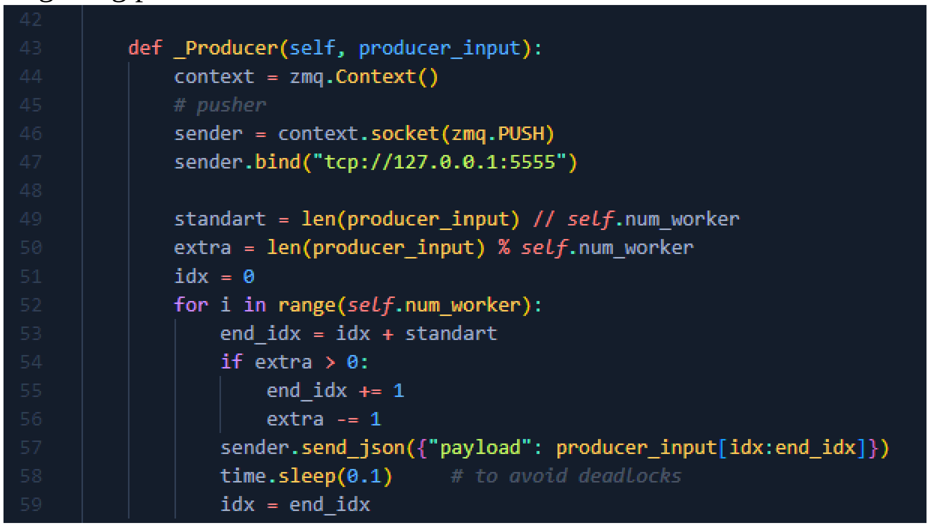
```python
from multiprocessing import Process
import os
from abc import ABC, abstractmethod
import zmq
import time
```

2) Defined the MapReduce as abstract class using ‘ABC’ from ‘abc’ module. Then, there is a constructor that takes an integer parameter and assigns it to ‘num_worker’ variable. Before assigning, it checks for the parameter to be integer and it should be positive and less than or equal to 10; otherwise, it will throw an error. (these errors will never happen since we always enter correct parameter but to be safe these lines of codes are added.) Secondly, I have created 2 abstract method that are ‘Map()’ and ‘Reduce()’, which they will be implemented in subclasses.



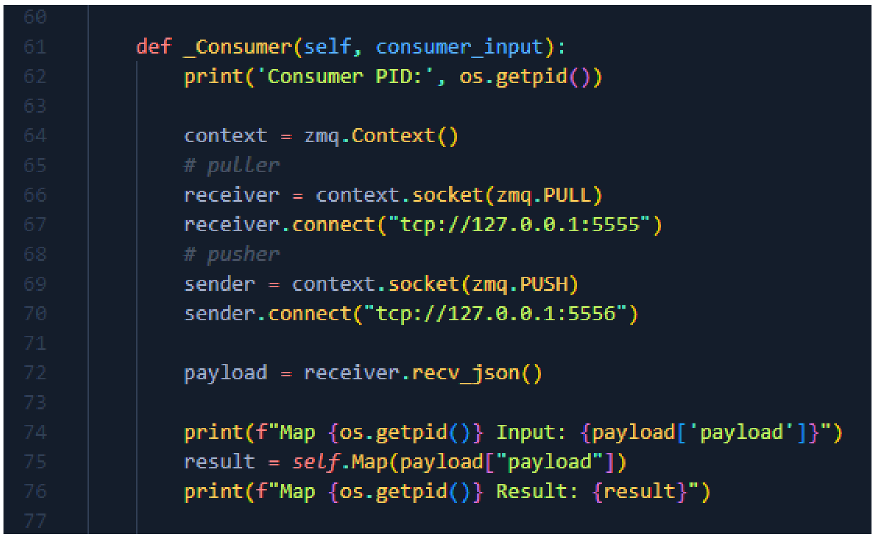
```python
class MapReduce(ABC):
    # Constructor
    def __init__(self, num_worker:int):
        if not isinstance(num_worker, int):
            raise TypeError("num_worker must be an integer!")
        elif num_worker < 0:
            raise TypeError("num_worker cannot be non-positive integer!")
        elif num_worker > 10:
            raise TypeError("num_worker cannot be more than 10!")
        self.num_worker = num_worker

    # Partial Result
    @abstractmethod
    def Map(self, map_input):
        pass

    # Result
    @abstractmethod
    def Reduce(self, reduce_input):
        pass
```

3) The producer method has an underscore (‘_’) which indicates that this method is protected and can be accessed from inherited classes. If it was ‘__’ (double underscores) then this method would have become a private method and even inherited subclasses couldn’t access this method. In producer method, it takes a list. First it opens a ‘PUSH’ socket, and this socket will be used to send data to consumers. Since here we have one-to-many connection type and producer is the one in this connection, socket in producer must bind to Ip address and port. Then the producer tries to split the list according to number of workers, so that they will have equal number of elements or difference of elements among them will be at most 1. The important part is that after sending data producer sleeps for 0.1 seconds so that consumers can catch and read the message without getting stuck in deadlock. *here I tried every combination of bind-connect but since in one-to-many connections always one side uses bind and I was using bind correct way, could not find a solution to deadlocks in my code and as the result kept using ‘time.sleep(0.1)’. I could use different port for each connection between 2 nodes, but sometimes due to errors occurred earlier while developing code, my ports were busy and multiple ports were giving ‘already in use’ error and I thought that during tests of student code, if any problem occur this error will be seen since if I use less number of ports, the probability of getting port in use error is smaller.



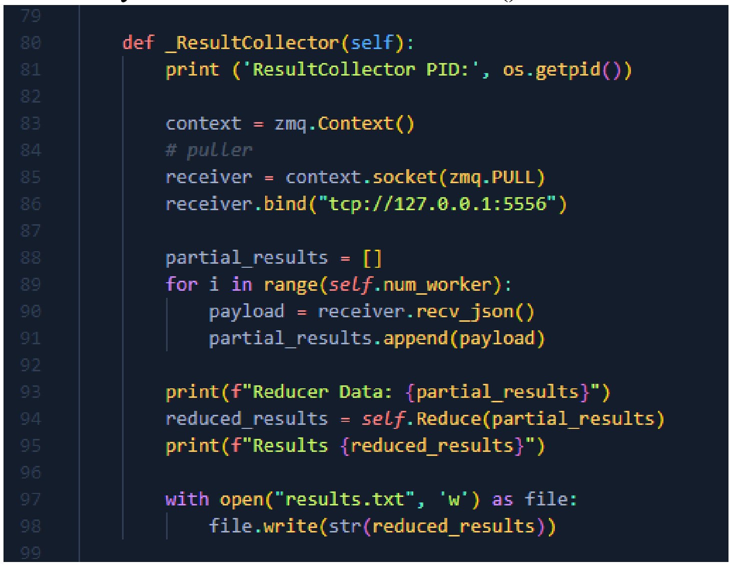
```python

    def _Producer(self, producer_input):
        context = zmq.Context()
        # pusher
        sender = context.socket(zmq.PUSH)
        sender.bind("tcp://127.0.0.1:5555")

        standart = len(producer_input) // self.num_worker
        extra = len(producer_input) % self.num_worker
        idx = 0
        for i in range(self.num_worker):
            end_idx = idx + standart
            if extra > 0:
                end_idx += 1
                extra -= 1
            sender.send_json({"payload": producer_input[idx:end_idx]})
            time.sleep(0.1)      # to avoid deadlocks
            idx = end_idx
```

4) In consumer it again has an underscore in beginning to indicate that it is a protected method. Since consumer is the many side in one-to-many of producer to consumers and many side in many-to-one in consumers to resultCollector, in both connections I used ‘connect’. Code will run normally until line 72, where ZMQ blocks code until a message is received, and then the message is given to ‘Map()’ and its result which is partial result in big picture is sent to resultCollector.



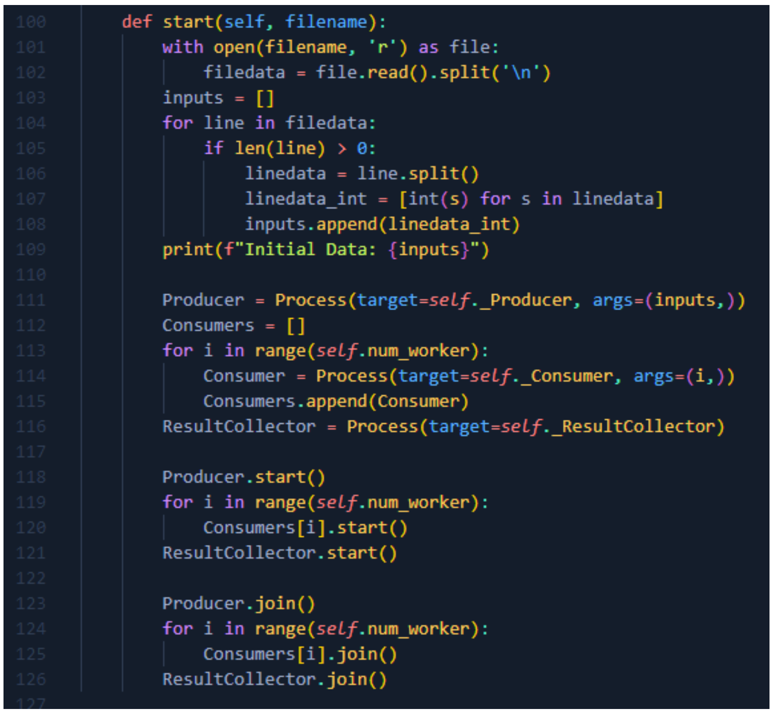
```
60
61      def _Consumer(self, consumer_input):
62          print('Consumer PID:', os.getpid())
63
64          context = zmq.Context()
65          # puller
66          receiver = context.socket(zmq.PULL)
67          receiver.connect("tcp://127.0.0.1:5555")
68          # pusher
69          sender = context.socket(zmq.PUSH)
70          sender.connect("tcp://127.0.0.1:5556")
71
72          payload = receiver.recv_json()
73
74          print(f"Map {os.getpid()} Input: {payload['payload']}")
75          result = self.Map(payload["payload"])
76          print(f"Map {os.getpid()} Result: {result}")
77
```

5) In result collector since it is one side of many-to-one in consumers to resultCollector, I used ‘bind’. First it binds to Ip and port and waits for consumers to send their partial results. resultCollector collects all their partial results in single list and then gives this list to ‘Reduce()’. And lastly writes the result of ‘Reduce()’ into ‘results.txt’.



```
    def _ResultCollector(self):
        print ('ResultCollector PID:', os.getpid())

        context = zmq.Context()
        # puller
        receiver = context.socket(zmq.PULL)
        receiver.bind("tcp://127.0.0.1:5556")

        partial_results = []
        for i in range(self.num_worker):
            payload = receiver.recv_json()
            partial_results.append(payload)

        print(f"Reducer Data: {partial_results}")
        reduced_results = self.Reduce(partial_results)
        print(f"Results {reduced_results}")

        with open("results.txt", 'w') as file:
            file.write(str(reduced_results))
```

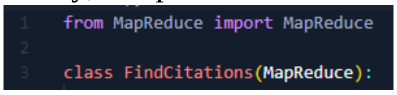
6) The start method takes filename (file path possible) and reads data inside. It creates a list of lines, and then converts each line into list of strings again, as a result I have created a 2D list, that each row represents each line in file and each column represent each word or number in line. Then I define and initialize 1 Producer, 1 resultCollector, and ‘num_worker’ Consumer processes. Then I start them and lastly wait for them to finish by using ‘join()’.



```python
    def start(self, filename):
        with open(filename, 'r') as file:
            filedata = file.read().split('\n')
        inputs = []
        for line in filedata:
            if len(line) > 0:
                linedata = line.split()
                linedata_int = [int(s) for s in linedata]
                inputs.append(linedata_int)
        print(f"Initial Data: {inputs}")

        Producer = Process(target=self._Producer, args=(inputs,))
        Consumers = []
        for i in range(self.num_worker):
            Consumer = Process(target=self._Consumer, args=(i,))
            Consumers.append(Consumer)
        ResultCollector = Process(target=self._ResultCollector)

        Producer.start()
        for i in range(self.num_worker):
            Consumers[i].start()
        ResultCollector.start()

        Producer.join()
        for i in range(self.num_worker):
            Consumers[i].join()
        ResultCollector.join()
```

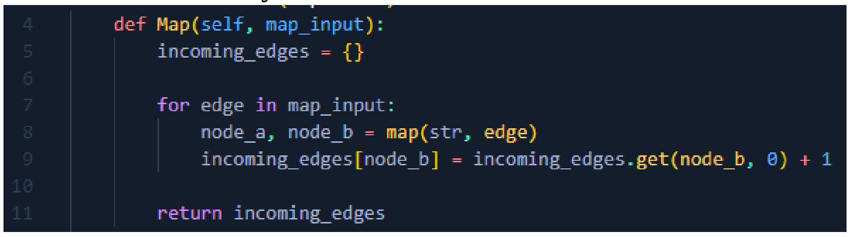
FindCitations.py

1) Firstly, I import the abstract class and then define the subclass.



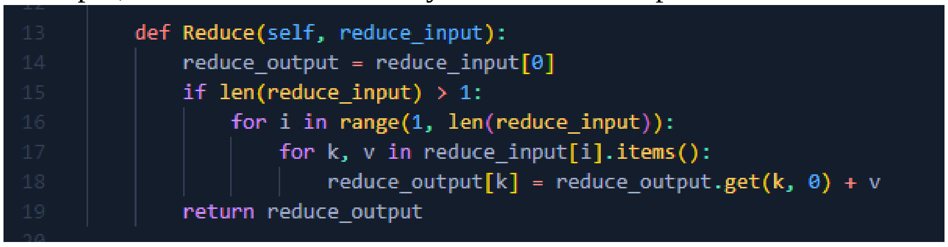
```python
from MapReduce import MapReduce

class FindCitations(MapReduce):
```

2) In Map method, it gets list of edges as ‘map input’. Each edge is list of 2 number, the first number represents A and second represents B in an edge of AB. Since we only have to find count of incoming edges and we are sure that there is no duplicate edge in ‘map_input’ we can easily iterate over the edges, if I see vertex B for first time then I set count of that B to 1, otherwise I increment the count of that B’s count. I keep counts in a dictionary called ‘incoming_edges’ that its keys are B vertices, and their values are their counts. And at the last returns the dictionary.



```python
    def Map(self, map_input):
        incoming_edges = {}

        for edge in map_input:
            node_a, node_b = map(str, edge)
            incoming_edges[node_b] = incoming_edges.get(node_b, 0) + 1

        return incoming_edges
```

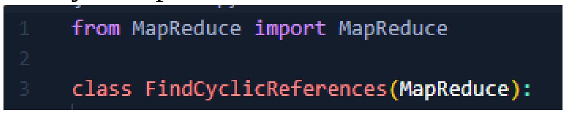
3) The Reduce takes list of dictionaries as ‘reduce_input’. To be efficient in terms of space and time, to avoid iterating over first dictionary, I directly set it as return object of method. Then I iterate over other dictionaries, for every key in other that exist in output, I add value of that key to output, otherwise I add that key and value to output.



```python
    def Reduce(self, reduce_input):
        reduce_output = reduce_input[0]
        if len(reduce_input) > 1:
            for i in range(1, len(reduce_input)):
                for k, v in reduce_input[i].items():
                    reduce_output[k] = reduce_output.get(k, 0) + v
        return reduce_output
```

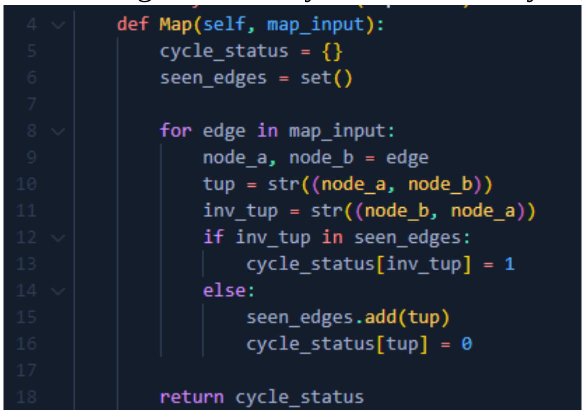
FindCyclicReferences.py

1) Firstly, I import the abstract class and then define the subclass.



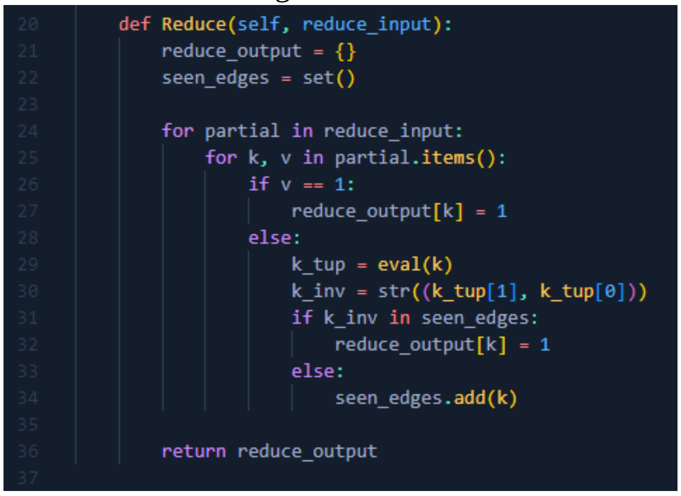
```python
from MapReduce import MapReduce

class FindCyclicReferences(MapReduce):
```

2) In Map method, to I had to iterate over all edges to find if there is a cycle, also had to keep iterations in low count, so I kept all the edges in a set as seen edges, but the added edges to set are behaved like an edge that does not have a direction so that it can be both AB and BA. So, while iterating over edges, if the edge is inside the set, which means we have seen it at least 1 time before, and now we see it for second time, it means that this is a cycle and I add it to output dictionary with value of 1 and key of first seen edge. What I mean about first seen edge is that if AB is in set and then we see BA and decide it is a cycle, I add AB to output. However, since this method give partial results, I have to keep the edges that do not have cycle, so that in Reduce it may get decided as cycle if their other direction is in another partial result. To indicate that the edge is currently does not have cycle, I give 0 as value to that edge.



```python
    def Map(self, map_input):
        cycle_status = {}
        seen_edges = set()

        for edge in map_input:
            node_a, node_b = edge
            tup = str((node_a, node_b))
            inv_tup = str((node_b, node_a))
            if inv_tup in seen_edges:
                cycle_status[inv_tup] = 1
            else:
                seen_edges.add(tup)
                cycle_status[tup] = 0

        return cycle_status
```

3) In Reduce method, I had same approach of using set to keep track of seen edges. Here the input is list of dictionaries. I iterate over every key-value pair in each dictionary and see if value is 1 or 0. If it is 1 then this edge (the key) is directly added to output. Otherwise, it will add the key to seen edges, but since our key is string of tuple format, I used ‘eval()’ built-in function to extract the tuple from string, and after extracting the tuple, the notion is same as Map method, I check for if the other direction of this edge is seen, if yes then add its seen direction to output, otherwise add this edge to set.



```python
    def Reduce(self, reduce_input):
        reduce_output = {}
        seen_edges = set()

        for partial in reduce_input:
            for k, v in partial.items():
                if v == 1:
                    reduce_output[k] = 1
                else:
                    k_tup = eval(k)
                    k_inv = str((k_tup[1], k_tup[0]))
                    if k_inv in seen_edges:
                        reduce_output[k] = 1
                    else:
                        seen_edges.add(k)

        return reduce_output
```