Database Project (Fall 2023)

Homework #2 (Due date: Oct 20)

Student ID: 2020315798

Name: Choi Jin Woo

Compress 1) your codes and 2) the document as follows:

- ‘DBP_HW2_STUDENTID.zip’
  - ✓ Code
    - brute_force.py
    - mapper.py, reducer.py, combiner.py
  - ✓ Document: DBP_HW2_STUDENTID.pdf

NOTE: You need to install matplotlib library.

1. [60pts] Consider you’re searching for restaurants to book in Suwon. You must cho ose optimal restaurants in **‘Suwon’** while considering conflicting features. Use pare to-optimal set (skyline query) to filter the results from the database to keep only t hose objects that are not worse than others.

(1) [30pts] Write the function named ‘pareto_optimal’ in the code file ‘brute_force.py’ to obtain the set of Pareto optimal restaurants in **‘Suwon’** from the ‘restaurant.txt’ dataset and plot the result.

Instructions:

- Refer to the definitions below and the baseline code which uses a brute-force algorithm to find the Pareto-optimal set. When plotting, please use the provided code. All data features should be considered. In general, higher quality and service are preferred, and lower prices are preferred.
- You can check the implementation of pareto_optimal function with the compare_result function in the main.py file.

[Definition 1: Dominant relationship]

For tuple $t_i \in R^n = \{t_{i1}, t_{i2}, \dots, t_{ik}, \dots, t_{in}\}$, $t_i \prec t_j$ (Dominant relationship) satisfies when t wo conditions are true.

1) $\forall t_{ik} \in t_i, \forall t_{jk} \in t_j : t_{ik} \leq t_{jk}$

   For each k, t_{ik} is less than or equal to t_{jk}.

2) $\exists t_{ik} \in t_i, \exists t_{jk} \in t_j : t_{ik} < t_{jk}$

   There is at least one k that satisfies $t_{ik} < t_{jk}$.

[Definition 2: Dominant tuple]

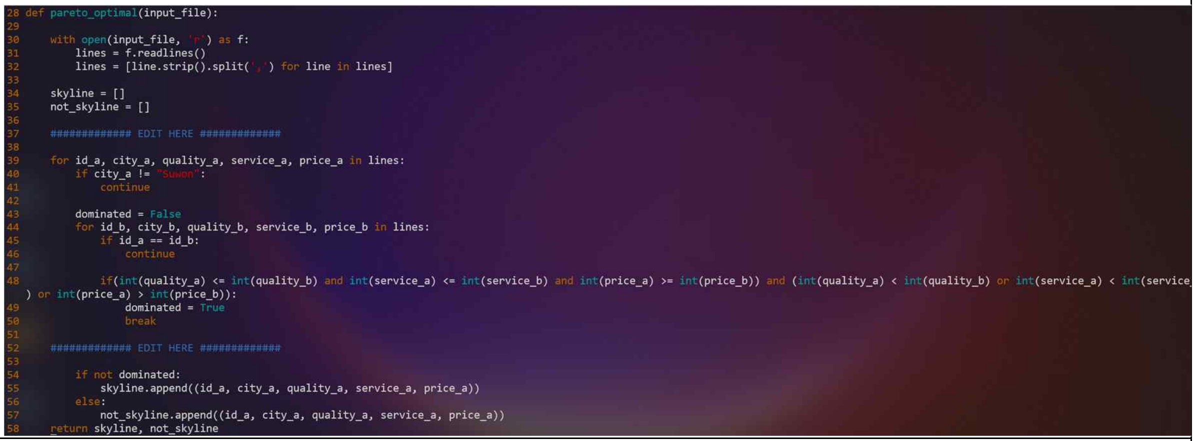
Given a set of tuples T, a set of dominant tuples is defined as:

$$Dom(T) = \{t \in T \mid \nexists p \in T\text{¥}t : t \prec p\}$$

$Dom(T) = \{t \in T|$ a set of tuples T except for t has no element p that satisfies $t \prec p\}$

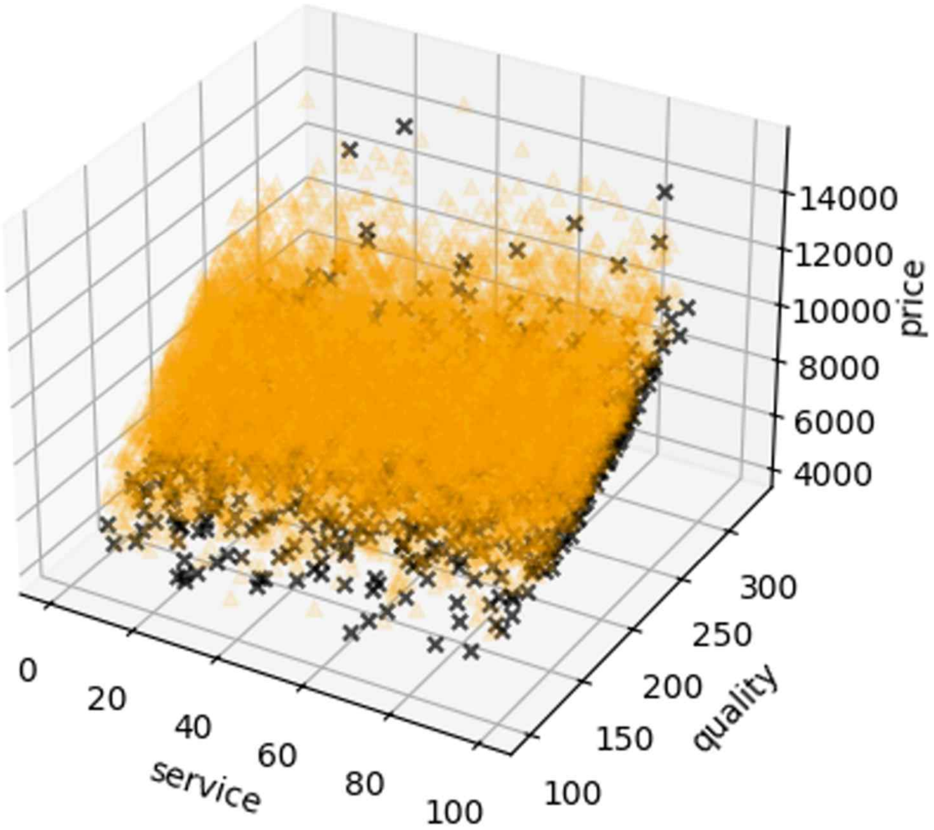
Answer: Enter your code and result here.

pareto_optimal function code



```python
def pareto_optimal(input_file):

    with open(input_file, 'r') as f:
        lines = f.readlines()
        lines = [line.strip().split(',') for line in lines]

    skyline = []
    not_skyline = []

    ############# EDIT HERE #############

    for id_a, city_a, quality_a, service_a, price_a in lines:
        if city_a != "Suwon":
            continue

        dominated = False
        for id_b, city_b, quality_b, service_b, price_b in lines:
            if id_a == id_b:
                continue

            if(int(quality_a) <= int(quality_b) and int(service_a) <= int(service_b) and int(price_a) >= int(price_b)) and (int(quality_a) < int(quality_b) or int(service_a) < int(service_
) or int(price_a) > int(price_b)):
                dominated = True
                break

    ############# EDIT HERE #############

        if not dominated:
            skyline.append((id_a, city_a, quality_a, service_a, price_a))
        else:
            not_skyline.append((id_a, city_a, quality_a, service_a, price_a))
    return skyline, not_skyline
```

Plot

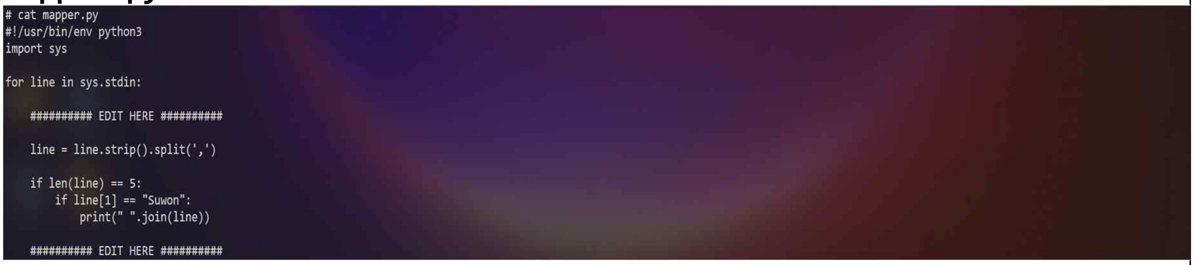


(2) **[30pts]** Write the **'mapper.py'** and **'reducer.py'** code to produce the same results using MapReduce as the pareto_optimal code you wrote in Problem 1 – (1).

Instructions:

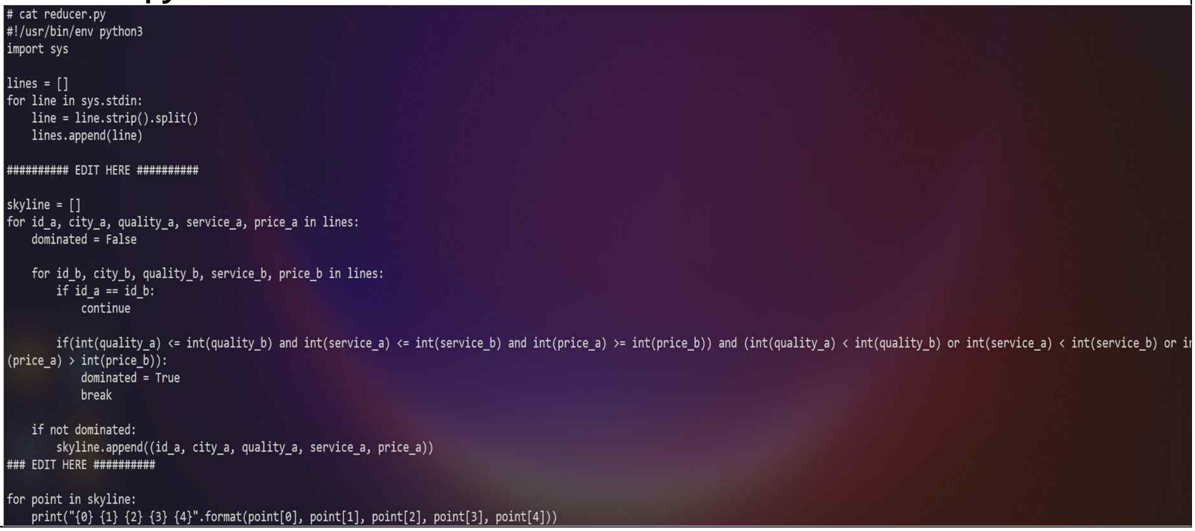
- You can check the result output with the compare result function in the main.py file.

mapper.py



```
# cat mapper.py
#!/usr/bin/env python3
import sys

for line in sys.stdin:

    ########## EDIT HERE ##########

    line = line.strip().split(',')

    if len(line) == 5:
        if line[1] == "Suwon":
            print(" ".join(line))

    ########## EDIT HERE ##########
```

reducer.py



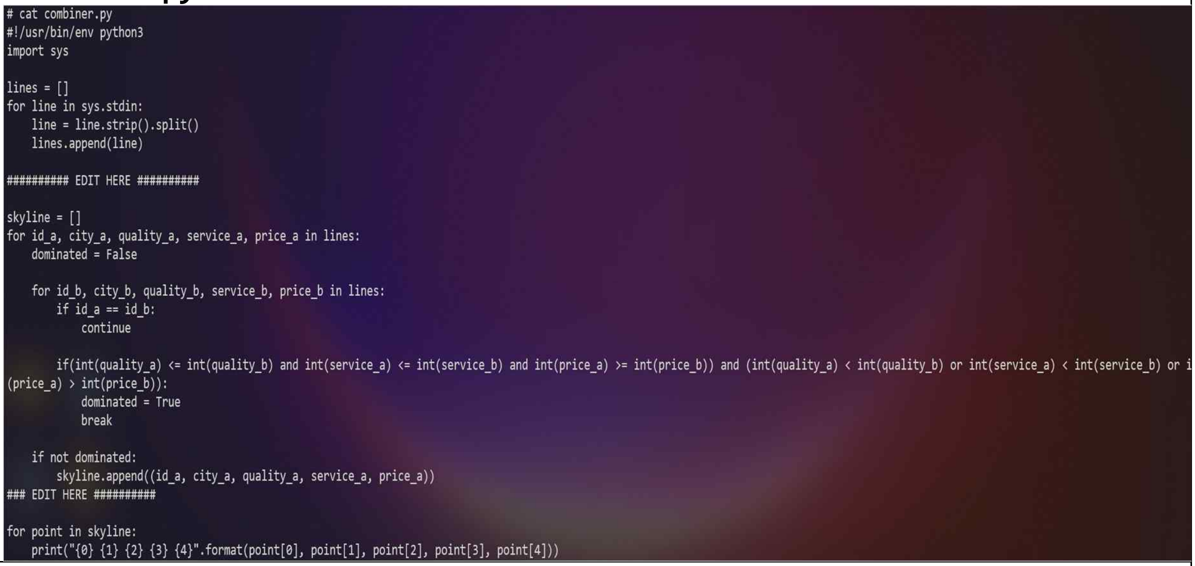
```
# cat reducer.py
#!/usr/bin/env python3
import sys

lines = []
for line in sys.stdin:
    line = line.strip().split()
    lines.append(line)

########## EDIT HERE ##########

skyline = []
for id_a, city_a, quality_a, service_a, price_a in lines:
    dominated = False

    for id_b, city_b, quality_b, service_b, price_b in lines:
        if id_a == id_b:
            continue

        if(int(quality_a) <= int(quality_b) and int(service_a) <= int(service_b) and int(price_a) >= int(price_b)) and (int(quality_a) < int(quality_b) or int(service_a) < int(service_b) or ir
(price_a) > int(price_b)):
            dominated = True
            break

    if not dominated:
        skyline.append((id_a, city_a, quality_a, service_a, price_a))
### EDIT HERE ##########

for point in skyline:
    print("{0} {1} {2} {3} {4}".format(point[0], point[1], point[2], point[3], point[4]))
```

2. **[40pts]** Write additional code **'combiner.py'**. Compare the time difference when using the combiner versus not using it (from 1-(2)) and provide an explanation for the observed variance.

Instruction:

- Run the command *$ bash base.sh* and *$ bash base_combiner.sh* to run MapReduce and compare the time consumed. **Capture and report the images from each case, and briefly explain the result.**
- **If you are using M1 mac and wxw-matt:docker-hadoop repository, you must modify the version of Hadoop streaming in shell files.**
- To ensure correct functionality of the shell files, the output file in HDFS must be in the **/hw/output** path. If you are going to use your own path, you must modify the shell files.

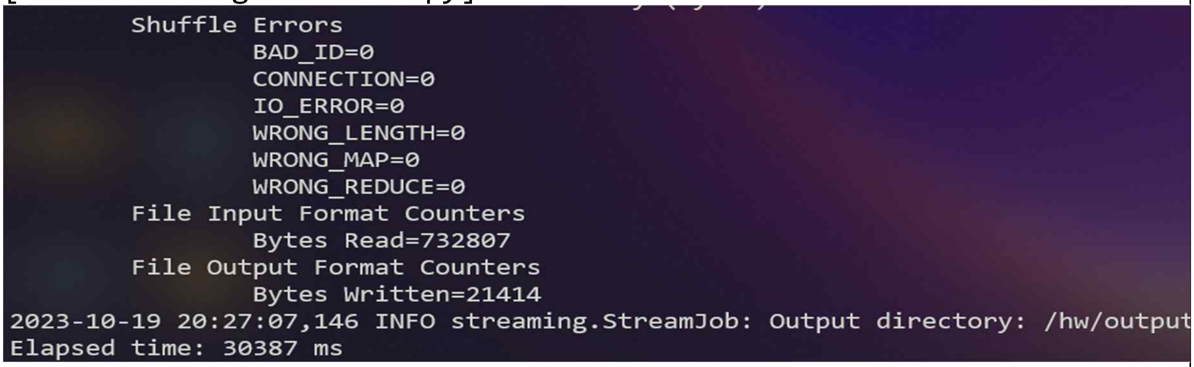
combiner.py



```
# cat combiner.py
#!/usr/bin/env python3
import sys

lines = []
for line in sys.stdin:
    line = line.strip().split()
    lines.append(line)

########## EDIT HERE ##########

skyline = []
for id_a, city_a, quality_a, service_a, price_a in lines:
    dominated = False

    for id_b, city_b, quality_b, service_b, price_b in lines:
        if id_a == id_b:
            continue

        if(int(quality_a) <= int(quality_b) and int(service_a) <= int(service_b) and int(price_a) >= int(price_b)) and (int(quality_a) < int(quality_b) or int(service_a) < int(service_b) or i
(price_a) > int(price_b)):
            dominated = True
            break

    if not dominated:
        skyline.append((id_a, city_a, quality_a, service_a, price_a))
### EDIT HERE ##########

for point in skyline:
    print("{0} {1} {2} {3} {4}".format(point[0], point[1], point[2], point[3], point[4]))
```

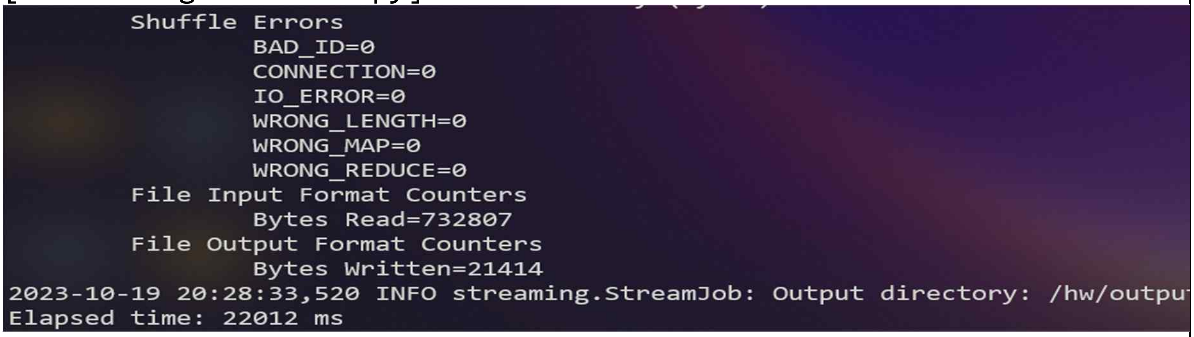
Report

[Without using combiner.py]



```
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=732807
        File Output Format Counters
                Bytes Written=21414
2023-10-19 20:27:07,146 INFO streaming.StreamJob: Output directory: /hw/output
Elapsed time: 30387 ms
```

[With using combiner.py]



```
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=732807
        File Output Format Counters
                Bytes Written=21414
2023-10-19 20:28:33,520 INFO streaming.StreamJob: Output directory: /hw/outpu
Elapsed time: 22012 ms
```

Using combiner.py reduced almost 8 seconds compare to one that is not using it. The purpose of the combiner is to perform local aggregation, and in this scenario, the logic for the local aggregation is the same as the global aggregation logic in the reducer. As a result, employing the combiner can help in minimizing the data transfer and improving the overall performance.