**Database Project(SWE3033) (Fall 2024)**

**Project #2 (100pts, Due date: Oct 22)**

**Student ID**: \_\_\_\_\_2018310773\_\_\_\_

**Name**: \_\_\_\_임승재\_\_\_\_

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

* ‘DBP\_P2\_STUDENTID\_NAME.zip’
  + Code
    - brute\_force.py
    - mapper.py, reducer.py, combiner.py
  + Document: DBP\_ P2\_STUDENTID\_NAME.pdf

**NOTE:** You need to install matplotlib library. If you cannot install the matplotlib library in your docker container, you can run it in your environment for problem 1-(1).

1. **[60pts]** Consider you’re searching for restaurants to book in Suwon. You must choose optimal restaurants in **‘Suwon’** while considering conflicting features. Use pareto-optimal set (skyline query) to filter the results from the database to keep only those 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 , (Dominant relationship) satisfies when two conditions are true.

For each , is less than or equal to .

There is at least one that satisfies .

**[Definition 2: Dominant tuple]**

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

a set of tuples except for has no element that satisfies }

Answer: Enter your code and result here.

|  |
| --- |
| **pareto\_optimal function code**  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:  dominated = False  if city\_a != "Suwon":  dominated = True  for id\_b, city\_b, quality\_b, service\_b, price\_b in lines:  if city\_a != "Suwon" or city\_b != "Suwon" or 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 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** |

* 1. **[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**  #!/usr/bin/env python3  import sys  for line in sys.stdin:  ########## EDIT HERE ##########  line = line.strip()  fields = line.split(',')    # Check if the city is Suwon  if fields[1] == "Suwon":  print("{0} {1} {2} {3} {4}".format(fields[0], fields[1], fields[2], fields[3], fields[4]))  ########## EDIT HERE ########## |
| **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 line\_a in lines:  # print(line\_a[0].split(','))  fields\_a = line\_a  id\_a, city\_a, quality\_a, service\_a, price\_a = fields\_a[0], fields\_a[1], fields\_a[2], fields\_a[3], fields\_a[4]  dominated = False  for line\_b in lines:  fields\_b = line\_b  id\_b, city\_b, quality\_b, service\_b, price\_b = fields\_b[0], fields\_b[1], fields\_b[2], fields\_b[3], fields\_b[4]  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 int(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])) |

1. **[40pts]** Write additionalcode **‘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.**
* 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**  #!/usr/bin/env python3  import sys  lines = []  for line in sys.stdin:  line = line.strip().split()  lines.append(line)  ########## EDIT HERE ##########  skyline = []  for line\_a in lines:  fields\_a = line\_a  id\_a, city\_a, quality\_a, service\_a, price\_a = fields\_a[0], fields\_a[1], fields\_a[2], fields\_a[3], fields\_a[4]  dominated = False  for line\_b in lines:  fields\_b = line\_b  id\_b, city\_b, quality\_b, service\_b, price\_b = fields\_b[0], fields\_b[1], fields\_b[2], fields\_b[3], fields\_b[4]    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 int(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 combiner:    With combiner:    Experiments with combiner improves map-reduce performance. The combiner aggregates data before sending it to the reducer. This reduces the amount of data transferred through network. By minimizing the volume of sending data, processing speed improves. Also, With the combiner reducing the amount of intermediate data, less data needs to be written to local storage. This lowers the disk I/O, which is another factor that can slow down processing. |