WovenPlanet Assignment

Enviroment

1) Prefer Linux OS/Mac OS

2) Prefer Python3.8+

3) requirement.txt: heapq/collections/wraps;

QuickStart

Please prepare your target file first. The format is like

<unique record identifier><white_space><numeric value>

e.g.

1426828011 9

1426828028 350

1426828037 25

1426828056 231

1426828058 109

1426828066 111

You can also generate the target file by script(**./tools/mock_data_generator.py**)

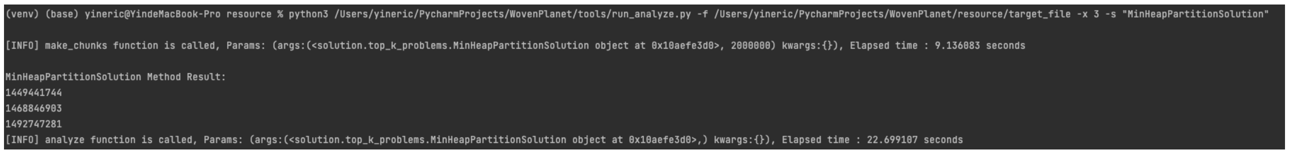
```
python3 /Users/yineric/PycharmProjects/WovenPlanet/tools/mock_data_generator.py
```

This script is responsible for analyze the target file. Please follow the reference help doc below. (***./tools /run_analyze.py***)

> Usage: run_analyze.py [options]
>
> Options:
> –h, ––help show this help message and exit
> –f TARGET_FILE, ––file=TARGET_FILE
> Input the target file path
> –x X_LARGEST, ––x_largest=X_LARGEST
> Give the X value for calculating X–largest item from

> the target file
> –s SOLUTION, ––solution=SOLUTION
> Assign the solutions:
> [”EntireSortSolution”,”MinHeapSolution”, ”MinHeapPartitionSolution”;], the default Solution is
> ”MinHeapPartitionSolution”

```
python3 /Users/yineric/PycharmProjects/WovenPlanet/tools/run_analyze.py –f /Users/yineric/PycharmProjects
/WovenPlanet/resource/target_file –x 3 –s ”MinHeapPartitionSolution”
```



```
(venv) (base) yineric@YindeMacBook-Pro resource % python3 /Users/yineric/PycharmProjects/WovenPlanet/tools/run_analyze.py -f /Users/yineric/PycharmProjects/WovenPlanet/resource/target_file -x 3 -s "MinHeapPartitionSolution"

[INFO] make_chunks function is called, Params: (args:(<solution.top_k_problems.MinHeapPartitionSolution object at 0x10aefe3d0>, 2000000) kwargs:{}), Elapsed time : 9.136083 seconds

MinHeapPartitionSolution Method Result:
1449441744
1468846903
1492747281
[INFO] analyze function is called, Params: (args:(<solution.top_k_problems.MinHeapPartitionSolution object at 0x10aefe3d0>,) kwargs:{}), Elapsed time : 22.699107 seconds
```

Problem Description

```
Imagine a file in the following fixed format:
<unique record identifier><white_space><numeric value>
e.g.
1426828011 9
1426828028 350
1426828037 25
1426828056 231
1426828058 109
1426828066 111
.
.
.
Write a program that reads from ’stdin’ the contents of a file, and optionally accepts
the absolute path of a file from the command line. The file/stdin stream is expected
to be in the above format. The output should be a list of the unique ids associated
with the X–largest values in the rightmost column, where X is specified by an input
parameter. For example, given the input data above and X=3, the following would be
valid output:
1426828028
1426828066
1426828056
Note that the output does not need to be in any particular order. Multiple instances
of the same numeric value count as distinct records of the total X. So if we have 4
records with values: 200, 200, 115, 110 and X=2 then the result must consist of the two
IDs that point to 200 and 200 and no more.
Your solution should take into account extremely large files.

What to return back to us
1. Your code in the language of your preference. If it’s in Python, Java or C++ it’s
preferable for us.
2. Include in your code comments about your solution’s algorithmic complexity
```

```
for both time and memory.
3. Include instructions on how to build and run your code in a README file.
Please include operating system information if necessary.
4. Provide tests for your code to illustrate it works and it’s robust.
5. Please zip everything in a directory named your first.lastname/ and
return via email.
6. In your email response please let us know roughly how many hours you spent
for this exercise.
Notes
● For your reference, successful candidates reported to have spent
approximately 10 hours on average on this challenge.
● Write your project as if it will be reviewed by your peers and ultimately
deployed to production.
```

;;

Code Structure

/resource: target file and target_file partitions will be generated in this folder

/solution: conclude the main solutions in this ”top_*k_problems*”

/test: conclude test case

/tools: conclude scripts.

”mock_data_generate.py” is used to generate the mock data of target file.

”run analyze” is used to start to analyze the target file

/utils: put the basic common class here. such as basic data structure, basic decorators, other basic common utils.

Solution

;According to the requirement, the number of the total records is extremely huge, and it looks like the X of the ”X–largest” number is not such huge since the sample X equals to 2. Otherwise, it does not require the particular order of the output. Here we figure out those solutions below.

”MinHeapPartitionSolution” which is default, should be best solution to handle with huge file.

MinHeapSolution;

a concrete subclass of BaseSolution that implements the Top–K computation using a min heap data structure. It adds an instance variable ”min_heap” that holds the min heap, and the ”analyze” function reads the target file, adds each row of data to the min heap, and computes the final Top–K results.

The time complexity of this solution is O(N * log(K)) where N is the number of rows in the target file and K is the number of top records to be computed. The memory complexity of this solution is O(K) since the min heap data structure will store the top K smallest elements at any given time.

1. Construct a ”min heap” with K size (K–largest)
2. Read the target file row by row.
3. Read one row record, try to add the record into the Min Heap with K size.
4. If the element value is higher than the lowest element of the heap, then pop the lowest element and add the current element into ”heap”.

Finally, the largest–K elements will be stored into this ”min heap”.

MinHeapPartitionSolution;

Considering the size of the file is extremely huge. Maybe hundreds MB/GB/TB.

Spliting the data should be considered and it will accelerate the speed for the CPU insentive calculation by multi–process .

Assume that we have a single standalone server to calculate the result with multiple process.

Time complexity: O(N*logK / P) for each process, where N is the number of records, K is the value of X, and P is the number of processes since we are splitting the records into P chunks, processing each chunk in parallel, and each chunk has a time complexity of O(N/P * logK). The final total time complexity is O(N*logK).

Memory complexity: O(K + P), because we are storing the K largest values in the heap for each process, and P is the number of processes.

This solution will accelerate the process by multiprocess compared with the MinHeapSolution.

1. Make chunks to split the target file based on the size of the file.
2. One process handles with one chunk file and read the file row by row.
3. For each chunk file, each process will construct a ”min–heap” similar with the **minHeapSolution** by read the target file row by row
4. When the process finish reading the file & construct its own ”min–heap”, each process will return its ”min–heap”
5. The main process get each process’s ”min–heap” and merge all the k–largest ”min–heap” into the main ”min–heap”.

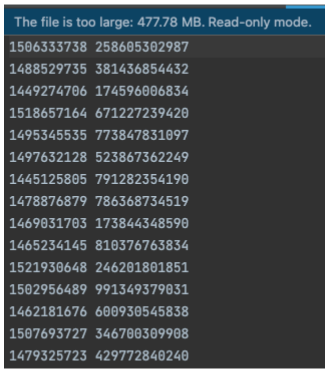
EntireSortSolution;

This solution is the comparing method to verify the result. If the file is extremely huge, it will bring the OOM problem. Also, the performance is also not good comparing with MinHeapSolution.

The memory complexity of this code is O(n) and The time complexity of this code is O(nlogn) where n is the number of records in the target file

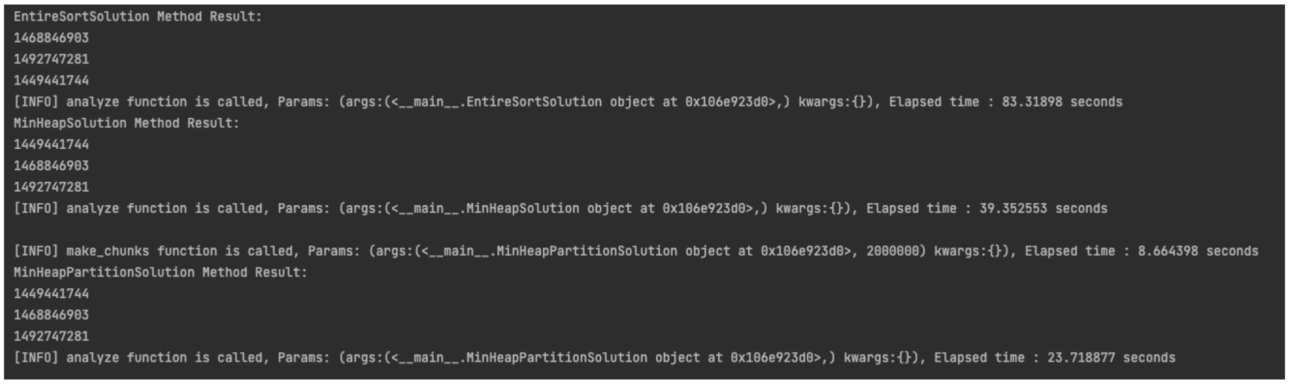
1. read the row into list
2. sort them and pop up the K-largest elements.

Experiment



```
The file is too large: 477.78 MB. Read-only mode.
1506333738 258605302987
1488529735 381436854432
1449274706 174596006834
1518657164 671227239420
1495345535 773847831097
1497632128 523867362249
1445125805 791282354190
1478876879 786368734519
1469031703 173844348590
1465234145 810376763834
1521930648 246201801851
1502956489 991349379031
1462181676 600930545838
1507693727 346700309908
1479325723 429772840240
```

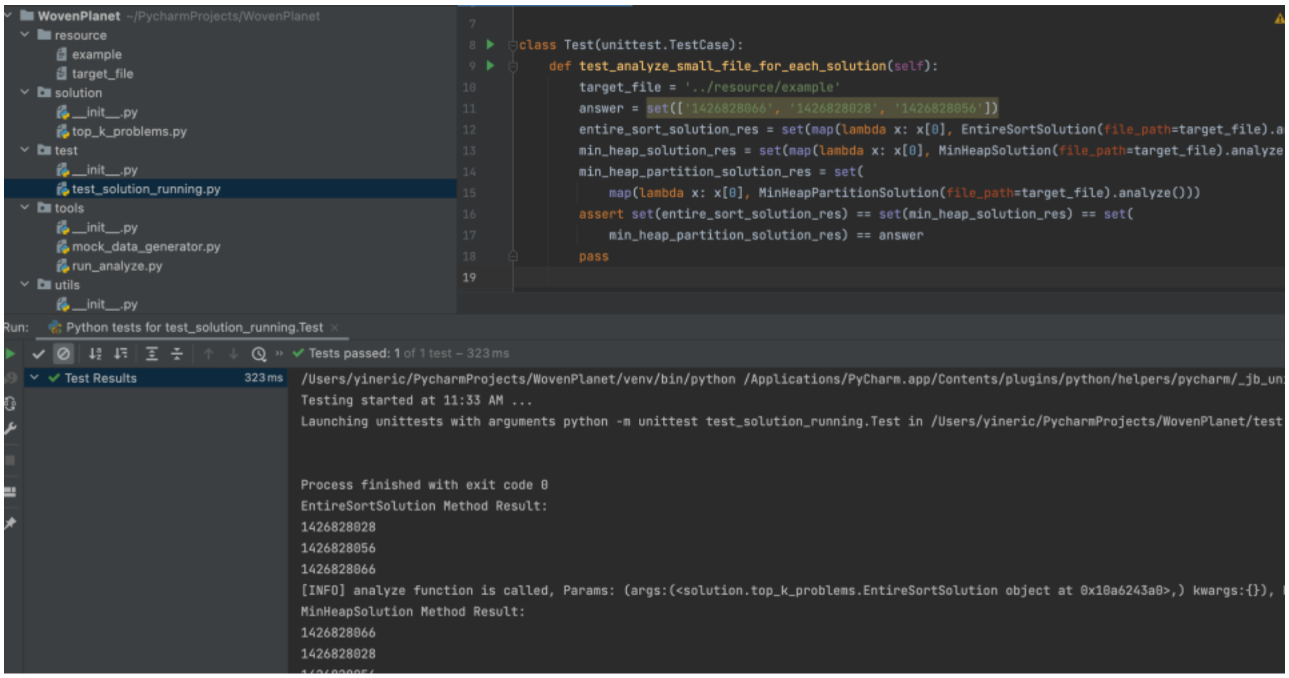
Analyze the 477 MB file and we can find that MinHeapPartitionSolution speed is fastest and EntireSortSolution is the slowest.



```
EntireSortSolution Method Result:
1468846903
1492747281
1449441744
[INFO] analyze function is called, Params: (args:(<__main__.EntireSortSolution object at 0x106e923d0>,) kwargs:{}), Elapsed time : 83.31898 seconds
MinHeapSolution Method Result:
1449441744
1468846903
1492747281
[INFO] analyze function is called, Params: (args:(<__main__.MinHeapSolution object at 0x106e923d0>,) kwargs:{}), Elapsed time : 39.352553 seconds

[INFO] make_chunks function is called, Params: (args:(<__main__.MinHeapPartitionSolution object at 0x106e923d0>, 2000000) kwargs:{}), Elapsed time : 8.664398 seconds
MinHeapPartitionSolution Method Result:
1449441744
1468846903
1492747281
[INFO] analyze function is called, Params: (args:(<__main__.MinHeapPartitionSolution object at 0x106e923d0>,) kwargs:{}), Elapsed time : 23.718877 seconds
```

Test Case:



```python

class Test(unittest.TestCase):
    def test_analyze_small_file_for_each_solution(self):
        target_file = '../resource/example'
        answer = set(['1426828066', '1426828028', '1426828056'])
        entire_sort_solution_res = set(map(lambda x: x[0], EntireSortSolution(file_path=target_file).a
        min_heap_solution_res = set(map(lambda x: x[0], MinHeapSolution(file_path=target_file).analyze
        min_heap_partition_solution_res = set(
            map(lambda x: x[0], MinHeapPartitionSolution(file_path=target_file).analyze()))
        assert set(entire_sort_solution_res) == set(min_heap_solution_res) == set(
            min_heap_partition_solution_res) == answer
        pass

```

```
/Users/yineric/PycharmProjects/WovenPlanet/venv/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pycharm/_jb_un
Testing started at 11:33 AM ...
Launching unittests with arguments python -m unittest test_solution_running.Test in /Users/yineric/PycharmProjects/WovenPlanet/test


Process finished with exit code 0
EntireSortSolution Method Result:
1426828028
1426828056
1426828066
[INFO] analyze function is called, Params: (args:(<solution.top_k_problems.EntireSortSolution object at 0x10a6243a0>,) kwargs:{}),
MinHeapSolution Method Result:
1426828066
1426828028
```