Practical 7: Implementation of sorting based two pass algorithm

GAHAN SARAIYA, 18MCEC10

18mcec10@nirmauni.ac.in

I. INTRODUCTION

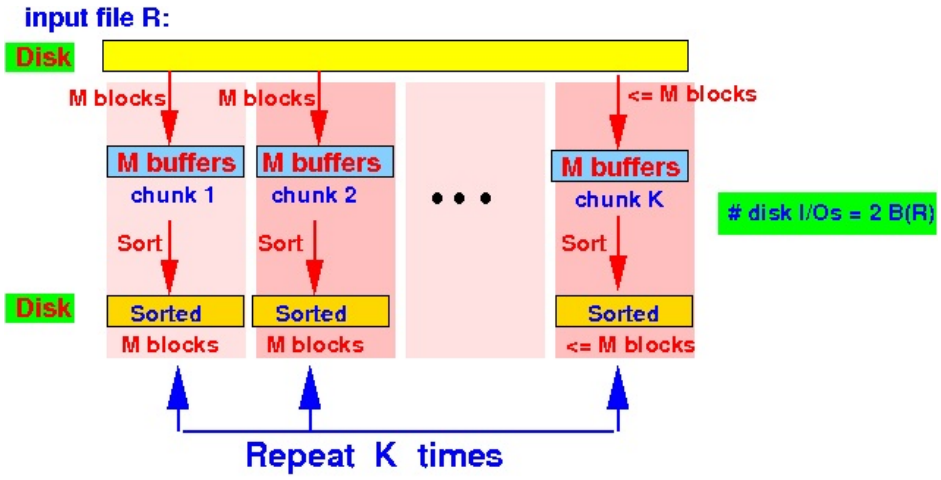
Aim of this practical is to implement sort based two pass algorithm to find distinct values.

II. LOGIC

Prerequisite setup: Created a data file and added dummy in it

I. Pass 1

1. Get $Available\ Memory\ Buffer(M)$
2. Determine file size and decide whether file can be read in one pass or two pass or more
3. For this experiment we'll focus on implementing two pass algorithm to evaluate distinct values
4. Divide file in to chunks of block
5. sort this chucks(sublist) individually with any in memory sorting algorithm
6. Write these sorted chunks(sublist) to disk



II. Pass 2

1. We (re)-use the M buffers to merge the first $(Available\ Memory\ Buffer - 1)$ chucks into a chunk of size $(Available\ Memory\ Buffer) \times (Available\ Memory\ Buffer - 1)$ blocks
2. Iterate over every first element of chunks and pick least value
3. Output if it is not same as previously picked element
4. Repeat from Step 2 until all the elements in all chunks are evaluated

III. Implementation

The code is implemented in Python as below

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Author: Gahan Saraiya
GiT: https://github.com/gahan9
StackOverflow: https://stackoverflow.com/users/story/7664524

Implementation of sorting based two pass algorithm
"""
import os
import math

from itertools import islice
from faker import Faker

fak = Faker()


class Iterator(object):
    """
    Iterator class to add tuple in form of table
    attributes -> <attrib1, attrib2, attrib3,...>
    Adding values
    values -> <val1, val2, val3, ....>
    """

    def __init__(self, attribute_tuple, file_path, *args, **kwargs):
        """
        :param attribute_tuple: attribute tuple in form of string containing
 ↪  attributes of file (if to be created)
```

```
30          :param file_path: location of data file
31          :param args:
32          :param kwargs:
33          """
34          self.attributes = attribute_tuple
35          self.file_path = file_path
36          self.write_back_path = kwargs.get("write_back_path", "temp.write")
37          self.separator = "\t"
38          self.records_per_block = kwargs.get("records_per_block", 30)
39          self.initialize_file()
40          print("Consideration:\n"
41                "Records per block: {}\n"
42                "Total Records per block: {}\n".format(self.records_per_block,
                  ↪  self.total_records)
43                )
44
45      @staticmethod
46      def read_in_chunks(file_object, chunk_size=1024):
47          """Lazy function (generator) to read a file piece by piece.
48          Default chunk size: 1k."""
49          while True:
50              data = file_object.read(chunk_size)
51              if not data:
52                  break
53              yield data
54
55      @property
56      def free_memory(self):
57          # calculate how many blocks can be accommodated in memory buffer
58          num_lines = sum(1 for line in open(self.file_path))
59          no_of_records = num_lines - 2  # remove header line and last new line
60          return 101  # for now return available memory statically for basic
            ↪  implementation
61
62      @property
63      def total_blocks(self):
64          # calculate total number of blocks by record size
65          return math.ceil(self.total_records / self.records_per_block)
66
67      @property
68      def total_records(self):
69          # calculate total number of blocks by record size
```

```
70          num_lines = sum(1 for line in open(self.file_path))
71          no_of_records = num_lines - 2  # remove header line and last empty line
72          return no_of_records
73
74      @property
75      def can_be_one_pass(self):
76          # return False  # for testing
77          return True if self.total_blocks < self.free_memory else False
78
79      @property
80      def can_be_two_pass(self):
81          return True if self.free_memory > math.ceil(math.sqrt(self.total_blocks))
            ↪  else False
82
83      def initialize_file(self):
84          # check if file exits or not
85          if os.path.exists(self.file_path):
86              pass
87          else:
88              # create file with header if file not exist
89              with open(self.file_path, "w") as f:
90                  f.write(self.separator.join(self.attributes))
91                  f.write("\n")
92          return True
93
94      def add_dummy_data(self, number_of_record=100):
95          """
96
97          :param number_of_record: number of records to be inserted in given file
    ↪  path
98          :return:
99          """
100         with open(self.file_path, "a+") as file:  # open file in append mode
101             for _ in range(number_of_record):
102                 f = fak.profile()
103                 data_tuple = (
104                     f['name'], f['ssn'], f['sex'], f['job'].replace("\n", ""),
                        ↪  f['company'].replace("\n", ""), f['address'].replace("\n",
                        ↪  "")
105                 )
106                 data_string = self.separator.join(data_tuple) + "\n"
107                 file.write(data_string)
```

```
108
109     @staticmethod
110     def summary(total_results, total_records):
111         print("-"*30)
112         print("Total Results: {}".format(total_results))
113         print("Total Records: {}".format(total_records))
114         return True
115
116     @staticmethod
117     def split_file_in_blocks(file_obj, split_size):
118         blocks = []
119         while True:
120             block_records = list(islice(file_obj, split_size))
121             if not block_records:
122                 break
123             else:
124                 blocks.append(block_records)
125         return blocks
126
127     @staticmethod
128     def create_file_obj(attribute):
129         file_name = "output_distinct_on_{}.tsv".format(attribute)
130         return open(file_name, "w")
131
132     def get_distinct(self, attribute=None, only_summary=True,
     ↪   output_write=False):
133         output_obj = self.create_file_obj(attribute) if output_write else None
134         sort_key = attribute if attribute else "ssn"
135         print("{0}\n DISTINCT ON {1}\n{0}".format('#'*50, sort_key))
136         _result_set = []
137         if self.can_be_one_pass:
138             print("Processing One Pass Algorithm")
139             with open(self.file_path, "r") as f:
140                 content = f.read().split("\n")
141             for record in content:
142                 if record not in _result_set:
143                     _result_set.append(record)
144         elif self.can_be_two_pass:
145             # apply 2 pass algorithm to sort and use operation on database
146             print("Processing Two Pass Algorithm")
147             f = open(self.file_path, "r")
148             writer = open(self.write_back_path, "w")
```

```
            header = f.readline()
            writer.write(header)
            _idx = header.split(self.separator).index(sort_key)
            while True:
                # read blocks one by one
                block_records = list(islice(f, self.free_memory - 1))
                if not block_records:
                    break
                else:
                    # sort sublist by "ssn" or any other attribute
                    sorted_sublist = sorted(block_records, key=lambda x:
                    ↪  x.split(self.separator)[_idx])
                    writer.writelines(sorted_sublist)
                # write sorted block/sublist data back to disk(secondary memory)
            f.close()
            writer.close()
            # read sublist from each block and output desire result
            last_read = ""
            total_results = 0
            # for line in open(self.write_back_path, "r"):
            file = open(self.write_back_path, "r")
            header = file.readline()
            sorted_blocks = self.split_file_in_blocks(file, self.free_memory - 1)
            while sorted_blocks:
                temp_lis = [i[0].split(self.separator)[_idx] for i in
                ↪  sorted_blocks if i]
                if not temp_lis:
                    break
                current_record = min(temp_lis)
                chunk_no = temp_lis.index(current_record)
                try:
                    del sorted_blocks[chunk_no][0]
                except IndexError:
                    del sorted_blocks[chunk_no]
                if current_record and current_record != last_read:
                    if not only_summary:
                        print(current_record)
                    if output_write:
                        output_obj.write(current_record + "\n")
                    last_read = current_record
                    total_results += 1
            self.summary(total_results, self.total_records)
```

```
189         else:
190             # can not proceed all given blocks with memory constraint
191             print("Require more than two pass to handle this large data")
192         return _result_set
193 
194 
195 if __name__ == "__main__":
196     table = Iterator(attribute_tuple=("name", "ssn", "gender", "job", "company",
        ↪  "address"),
197                      file_path="iterator.dbf")
198     table.get_distinct("name", only_summary=True)
199     table.get_distinct("job", only_summary=True, output_write=True)
200     table.get_distinct("ssn", only_summary=True)
201     table.get_distinct("gender", only_summary=False)
```

I. Output

```
Consideration:
Records per block: 30
Total Records per block: 5000

##################################################
 DISTINCT ON name
##################################################
Processing Two Pass Algorithm
------------------------------
Total Results: 4747
Total Records: 5000
##################################################
 DISTINCT ON job
##################################################
Processing Two Pass Algorithm
------------------------------
Total Results: 632
Total Records: 5000
##################################################
 DISTINCT ON ssn
##################################################
Processing Two Pass Algorithm
------------------------------
Total Results: 4913
```

```
Total Records: 5000
################################################
 DISTINCT ON gender
################################################
Processing Two Pass Algorithm
F
M
------------------------------
Total Results: 2
Total Records: 5000
```

IV. Summary

I. Requirements of Two Pass

- $number\ of\ chunks \leq Available\ Memory\ Buffer - 1$

II. File Size Constraint

- $Max\ File\ Size \leq (Available\ Memory\ Buffer) \times (Available\ Memory\ Buffer - 1)$