

# Practical 7: Implementation of sorting based two pass algorithm

GAHAN SARAIYA, 18MCEC10

[18mcec10@nirmauni.ac.in](mailto:18mcec10@nirmauni.ac.in)

## I. INTRODUCTION

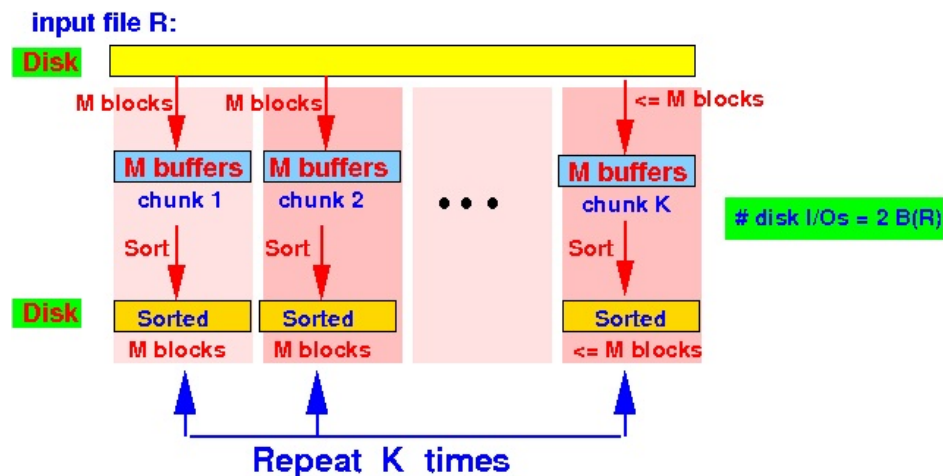
Aim of this practical is to implement sort based two pass algorithm to find distinct values.

## 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 chunks(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) chunks 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

---

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Author: Gahan Saraiya
5  GiT: https://github.com/gahan9
6  StackOverflow: https://stackoverflow.com/users/story/7664524
7
8  Implementation of sorting based two pass algorithm
9  """
10 import os
11 import math
12
13 from itertools import islice
14 from faker import Faker
15
16 fak = Faker()
17
18
19 class Iterator(object):
20     """
21     Iterator class to add tuple in form of table
22     attributes -> <attrib1, attrib2, attrib3,...>
23     Adding values
24     values -> <val1, val2, val3, ....>
25     """
26
27 def __init__(self, attribute_tuple, file_path, *args, **kwargs):
28     """
29         :param attribute_tuple: attribute tuple in form of string containing
30         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: {}".format(self.records_per_block,
43           ← self.total_records)
44     )
45
46     @staticmethod
47     def read_in_chunks(file_object, chunk_size=1024):
48         """Lazy function (generator) to read a file piece by piece.
49         Default chunk size: 1k."""
50         while True:
51             data = file_object.read(chunk_size)
52             if not data:
53                 break
54             yield data
55
56     @property
57     def free_memory(self):
58         # calculate how many blocks can be accommodated in memory buffer
59         num_lines = sum(1 for line in open(self.file_path))
60         no_of_records = num_lines - 2 # remove header line and last new line
61         return 101 # for now return available memory statically for basic
62         ← implementation
63
64     @property
65     def total_blocks(self):
66         # calculate total number of blocks by record size
67         return math.ceil(self.total_records / self.records_per_block)
68
69     @property
70     def total_records(self):
71         # 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))
82         ↪ else False
83
84     def initialize_file(self):
85         # check if file exists or not
86         if os.path.exists(self.file_path):
87             pass
88         else:
89             # create file with header if file not exist
90             with open(self.file_path, "w") as f:
91                 f.write(self.separator.join(self.attributes))
92                 f.write("\n")
93             return True
94
95     def add_dummy_data(self, number_of_record=100):
96         """
97         :param number_of_record: number of records to be inserted in given file
98         ↪ path
99         :return:
100         """
101         with open(self.file_path, "a+") as file: # open file in append mode
102             for _ in range(number_of_record):
103                 f = fak.profile()
104                 data_tuple = (
105                     f['name'], f['ssn'], f['sex'], f['job'].replace("\n", ""),
106                     ↪ f['company'].replace("\n", ""), f['address'].replace("\n",
107                     ↪ "")
108                 )
109                 data_string = self.separator.join(data_tuple) + "\n"
110                 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,
133     ↳ output_write=False):
134     output_obj = self.create_file_obj(attribute) if output_write else None
135     sort_key = attribute if attribute else "ssn"
136     print("{0}\n DISTINCT ON {1}\n{0}".format('#'*50, sort_key))
137     _result_set = []
138     if self.can_be_one_pass:
139         print("Processing One Pass Algorithm")
140         with open(self.file_path, "r") as f:
141             content = f.read().split("\n")
142             for record in content:
143                 if record not in _result_set:
144                     _result_set.append(record)
145     elif self.can_be_two_pass:
146         # apply 2 pass algorithm to sort and use operation on database
147         print("Processing Two Pass Algorithm")
148         f = open(self.file_path, "r")
149         writer = open(self.write_back_path, "w")
```

```
149 header = f.readline()
150 writer.write(header)
151 _idx = header.split(self.separator).index(sort_key)
152 while True:
153     # read blocks one by one
154     block_records = list(islice(f, self.free_memory - 1))
155     if not block_records:
156         break
157     else:
158         # sort sublist by "ssn" or any other attribute
159         sorted_sublist = sorted(block_records, key=lambda x:
160                                 ↪ x.split(self.separator)[_idx])
161         writer.writelines(sorted_sublist)
162         # write sorted block/sublist data back to disk(secondary memory)
163 f.close()
164 writer.close()
165 # read sublist from each block and output desire result
166 last_read = ""
167 total_results = 0
168 # for line in open(self.write_back_path, "r"):
169 file = open(self.write_back_path, "r")
170 header = file.readline()
171 sorted_blocks = self.split_file_in_blocks(file, self.free_memory - 1)
172 while sorted_blocks:
173     temp_lis = [i[0].split(self.separator)[_idx] for i in
174                 ↪ sorted_blocks if i]
175     if not temp_lis:
176         break
177     current_record = min(temp_lis)
178     chunk_no = temp_lis.index(current_record)
179     try:
180         del sorted_blocks[chunk_no][0]
181     except IndexError:
182         del sorted_blocks[chunk_no]
183     if current_record and current_record != last_read:
184         if not only_summary:
185             print(current_record)
186         if output_write:
187             output_obj.write(current_record + "\n")
188         last_read = current_record
189         total_results += 1
190 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",
197                                     "address"),
198                      file_path="iterator.dbf")
199     table.get_distinct("name", only_summary=True)
200     table.get_distinct("job", only_summary=True, output_write=True)
201     table.get_distinct("ssn", only_summary=True)
202     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

One pass algorithm can only work if whole block of relation can fit in to main memory otherwise we require more than one pass to determine correct result such as **Two-Pass Multiway Merge Sort (TPMMS) Algorithm** as implemented here for unary operator - distinct ( $\delta$ ).

For Two pass algorithm we need to fetch all blocks and then write the individually sorted blocks and again we need to read all blocks to perform query operation hence this algorithm can only work if below requirements are satisfied

##### I. Requirements of Two Pass

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

##### II. File Size Constraint

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