Правительство Российской Федерации

Государственное образовательное бюджетное учреждение

высшего профессионального образования

«Научно-исследовательский университет –

Высшая школа экономики»

Факультет: МИЭМ

Направление: ***Компьютерная безопасность***

Отчет по лабораторной работе №2 (алгоритмы сортировки)

по дисциплине

«Методы программирования»

Выполнил

Студент группы СКБ151

Михалицын Пётр

Москва, 2017.

Описание проекта

Проект заключался в сравнении двух видов сортировки

1) Вставками
2) Быстрая

Проект содержит в себе несколько модулей, которые можно разделить на 2 типа

1) Основные
    a. sort_db.py
    b. schedule_maker.py
    c. main.py
2) Вспомогательные
    a. sqlwrep.py
    b. train.py
    c. train_schedule.py

Все вспомогательные модули находятся в директории source

Вся информация сохраняется в базе данных sqlite

Спецификация модулей

Для начала перечислим спецификацию всех функций, классов и их методов, предоставляемых каждым модулем

Модуль train.py содержит класс Train, который обладает следующей спецификацией

```
class Train:
    """
    Consist values necessary for identification one train
    instance of Train more then other if depart time is later
    and if depart time is equal, train with more travel time is more
    Also have method form for getting tuple, identified train (necessary for
sqlite)
    """
```

Конструктор Train

```
def __init__(self, number, type, d_time, t_time):
    """
    Make instance of Train
    :param number: train number
    :param type: Express or Passenger
    :param d_time: department time
    :param t_time: travel time
    """
```

Класс предоставляет перегрузки всех операторов, необходимые для сравнения двух объектов этого типа, по критерию указного в спецификации

Кроме того, класс предоставляет метод form, который возвращает словарь значения, необходимый для заполнения записи в записи в базе денных.

```
def form(self):
    """
    :return: dictionary with keys train_number, type, d_time, t_time
    """
```

Модуль train_schedule.py – своего рода контейнер предоставляющий интерфей для работы с базой данных. Модуль содержит реализацию класса Schedule со следующей спецификацией.

```
class Schedule:
    """
    Attribute database consist wrapping database with type DatabaseWrap
    :param database_name: is str, consist path to database
    Class consist
    Attribute db: contain all data, before using call method load_database
    Attribute database: contain connection with database
    Special Method __init__,
    Method load_database for establish connection with database and load all
records in list
```

```
    Method unload_database for sever connection with database and load all
records in db to sqlite database
    Method insert_sort for insert sorting,
    Method quick_sort for quick sorting,
    Method-generator get_schedule for getting format string of records
    """
```

Конструктором

```
def __init__(self, database_name):
    """
    :param database_name: is str, consist path to database
    """
```

Методом load_database необходимым для загрузки базы данных в атрибут массив db, находящегося в классе

```
def load_database(self):
    """
    Establish connection with database and load all records in attribute db
    :return: None
    """
```

Методом unload_database выгружающий атрибут массив db в базу данных, находящуюся на диске

```
def unload_database(self, name):
    """
    Sever connection with database and load all records from db to sqlite
database
    :param name: name of database
    :return:  None
    :raise raise RunrimeError if attribute db doesn't exist

    """
```

Методом insert_sort, сортирующий атрибут массив db с помощью сортировки вставками

```
def insert_sort(self, autounload_into=None):
    """
    Implamentation of insert sort
    sort attribute db via insert sor
    :param autounload_into: if consist path to file name, automatically load
db in database
    :return None
    :raise raise RunrimeError if attribute db doesn't exist
    """
```

Методом quick_sort, сортирующий атрибут массив db с помощью быстрой сортировки

```
def quick_sort(self, db=None, first=True, autounload_into=None):
    """
    Implementation og quick sort,
    sort db via quick sort algorithm
    :param db: for recursion call
    :param first: for mark first call
    :param autounload_into: if consist path to file name, automatically load
db in database
    :return: None
```

```
    :raise raise RunrimeError if attribute sb doesn't exist
    """
```

Методом-генератором get_schedule, возвращающим форматированные записи в базе данных db

```
def get_schedule(self):
    """
    Method-generator for getting format string of records
    :return: format string of records
    :raise raise RunrimeError if attribute sb doesn't exist
    """
```

Модуль sqlwrap.py необходим для работой с базой данных обращение к которой инкапсулировано с помощью методов и перегруженных операторов класса DatabaseWrap, находящегося в этом модуле

```
class DatabaseWrap:
    """
    :argument database_name:
    Attribute database_name
    Attribute connection contain connect with database
    Attribute cursor conatin cursor with database
    2 Special-methods for index operator
    Methods open, clods and getsize
    """
```

Конструктор

```
def __init__(self, database_name):
    """
    :param database_name: must contain database name if database doesn't
exist make new
    """
```

Перегрузка операций обращения по индексу для чтения обращения и обращения по индексу для записи

```
def __getitem__(self, item):
    """
    Provide access to database
    :param item: is id-1 of record, which return
    :return: tuple contain record with id == item+1 (without id)
    """


def __setitem__(self, key, value):
    """
    Replace or make new record with id == key + 1 with values == value in
database
    :param key: is id-1
    :param value: tuple (train_number, type_of_train, departure_time,
travel_time)
    :return: None
    """
```

Метод get_size, предоставляющий возможность эффективно узнать кол-во записей в базе данных

```
def get_size(self):
    """
    :return: count of records in database
    """
```

Методы open и close, для начала и прекращения работы с базой данных

```
def open(self):
    """
    Open connection with database and commit changes
    and put in attribute connection connection with database
    and in attribute cursor -- cursor
    :return: None


def close(self):
    """
    Close connection with database and commit changes
    :return: None
    """
```

Модуль schedule_maker.py предоставляет функцию, необходимую для генерации случайной базы данных. Также в случае запуска модуля, как самостоятельной программы, имеется возможность создать базу данных интерактивно. Спецификация функции make_database, находящейся в этом модуле

```
def make_database(name, db=None):
    """
        Make database on storage
        Record is id, train number, type, depart time, travel time

        :param:name : int if want to get *number* random records
                      or str, that consist path to database file
        :param:db: None, if set name with type int. Further get value of name
                      db, if want to get database store records in db
        :return: -> name of created db

        If name is number and db is None create number of random records and
return "./data/random_schedule{name}.db
        If name is string and db is number, create number of random records
and return name
        If name is string and db is list of records, return database with
with records (with keeping order) and return name
    """
```

Модуль main.py является ключевым для расчета времени необходимого для сортировки и занесения результатов в лог. файл и при указании соответствующего флага вывода результатов в консоль (при запуске, как самостоятельной программы)

```
def main(log='log.txt', trace=False):
    """
    Make measure of two algorithms of sort and make log file according this
measure
    Before making measure ask user about confidence in action
    Measure timing of computing 5, 10, 50, 100 and so on to 10**5 of records
for insert sort and 10**7 for quick sort
    For each number generate random schedule in sqlite

    :param log: path to logfile
    :param trace: flag to showing measure in standard output stream
    :return: 0
    """
```

Модуль sort_db.py а основном может использоваться как самостоятельная программа, которая предоставляет функцию sort_db

```
def sort_db():
    """
    Sort database, which path input from stdin
    :return: 0
    """
```

Взаимодействие модулей

Модуль main.py сначала создает логфайл в который будут записываться результаты времени сортировки. Перед каждой итерацией создается случайная база данны размерностью 5, 10, 50, 100, …, 10^6, 5*10^6 x с помощью функции make_database находящейся в модуле schedule_maker.py. на каждой итерации создается путь к базе данных передается в конструктор класса Schedule. Каждый объект этого класса сначала загружает базу данных с диска, потом сортирует его с помощью insert_sort, потом другая сущность того же типа инициализированной той же базой данных сортируется с помощью quick_sort. Результаты заносятся в лог файл и при указании флага выводятся в поток вывода. На quick_sort проверяются все базы данных. На Insert sort проверяются только базы данных размер которых не больше 10^5 записей

Для модуля sort_db.py реализован похожим образом, за исключением того, что ничего не выводится в лог файл, а выводится только в поток вывода и итерация у нас всего лишь одна и способ сортировки мы выбираем сами.

Объект класса Schedule проживает следующую жизнь (в основном. В нашем случае только такую). Сначала мы его инициализируем, потом загружаем в него базу данных (точнее в его атрибут db) с помощью соответствующего метода, потом db сортируется. И выгружается в базу данных.

db хранит в себе объекты класса Train которые можно сравнивать между собой. В самой же базе данных sqlite хранятся картежи необходимые для инициализации объектов класса Train. Класс DatabaseWrap в модуле sqlwrap.py предоставляет все необходимые методы для извлечения этих картежей (путем перегрузки операции индексирования).

Во время выгрузки же с помощью метода form класса Train получаются обратно картеж значений необходимых для инициализации и с помощью перегрузки операции присвоения по индексу заносятся в базу данных эти картежи.

Результаты, полученные после оценки алгоритмов сортировки

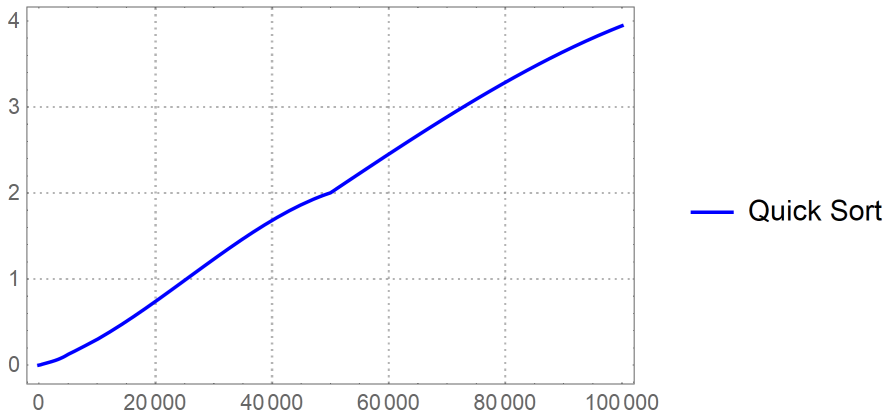
Как ожидалось алгоритм вставками имеет алгоритмическую сложность **O(n^2)**, но в то же время является стойким (из-за реализации) алгоритмом и при обладающим естественностью, т.к. при запуске на случайной базе данных размером 10^5 pfgbctq сортировал ее 3560 секунд. При сортировке же уже отсортированной таблицы сортировал ее 0.13 секунд. К тому же он расходовал **O(1)** дополнительной памяти, и поэтому является эффективны по памяти алгоритмом.

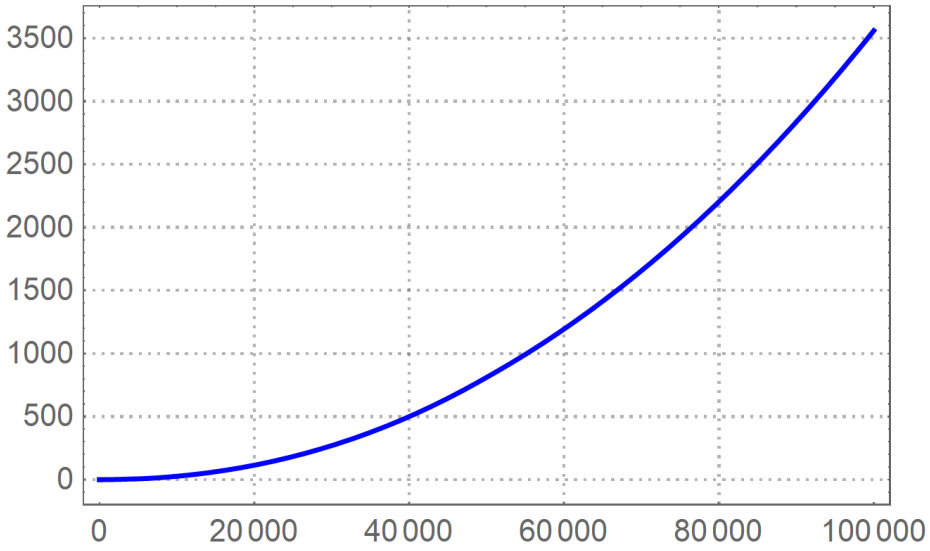
Алгоритм же быстрой сортировки имеет сложность **O(n*log(n))** по времени. Но зато не обладает стойкостью, о чем свидетельствует данная запись в уже отсортированной базе данных (в неотсортированной базе данных, все номера поездов идут по порядку)

499992	325470	Express	00:00	06:04
499993	348301	Express	00:00	06:04
499994	198281	Express	00:00	06:04

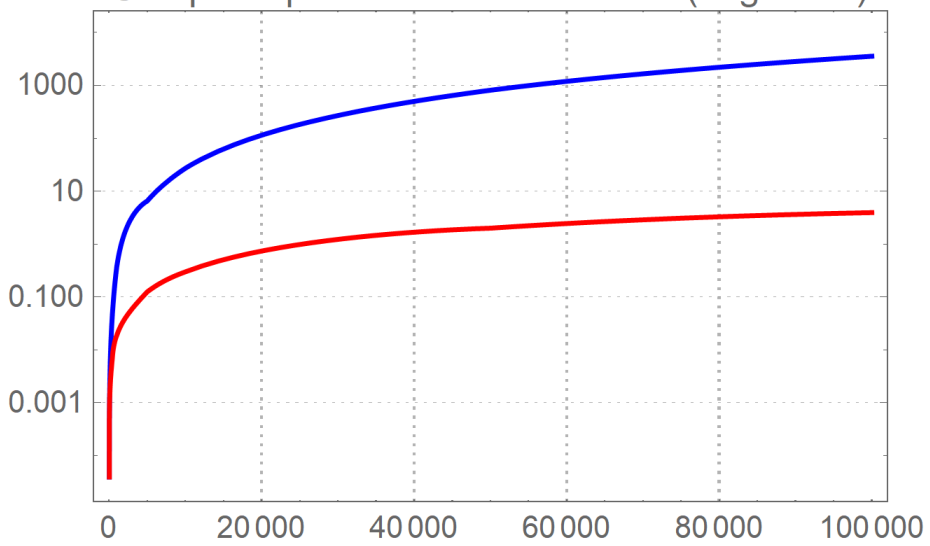
Второе поле – номер поезда. 4ое и 5ое время отправления и время в пути, соответственно. Таким образом три эквивалентных элемента не сохранили порядок номеров, хотя и являются эквивалентными. Так же этот алгоритм расходует дополнительную памяти O(n) памяти необходимой для “держания в голове” подмассивов записей. Данный алгоритм является естественным, но не столь естественным, как сортировка вставками. Естественность алгоритма выводится из того, что в подмассивах не надо производить никаких действий касаемых перемещения из одного подмассива в другой (больших и меньших опорного элемента). Так, например, для 10^5 элементов сортировка произошла в 2 раза быстрее на уже отсортированном массиве (2 секунды вместо 4). Но в любом из случаев, естественно, что гораздо лучше использовать сортировку вставками т.к. для почти отсортированного массива она эффективна она эффективна по времени, памяти, стойкости и более естественнее. В другую сторону быструю сортировку можно было бы модернизировать, введя флаг, который показывал было ли на какой-то глубине вложенности произведено перекидывание элементов из одного подмасива в другой и остановить сортировку в случае лжи (перекидываний не было=> массив отсортировано). Таким образом мы время примерно равное тому же, что и получили бы при сортировке вставками.

Ниже показаны графики и таблица сравнения эффективности этих алгоритмов по времени





Compare quick sort and insert sort (Log scale)



Time	Insert sort	Quick sort
5	0.0000212604	0.0000527845
10	0.0000777106	0.000113633
50	0.000882675	0.000779672
100	0.00279502	0.00153295
500	0.0656244	0.0093535
1000	0.371232	0.0187206
5000	6.53407	0.125347
10 000	27.1463	0.299511
50 000	811.577	2.00307
100 000	3560.33	3.94526
500 000	-	24.1745
1 000 000	-	51.7588
5 000 000	-	323.915

Пример базы данных до/после сортировки

До:

id	train_number	type_of_train	departure_time	travel_time
1	0	Express	06:52	07:57
2	1	Passenger	21:40	08:15
3	2	Passenger	10:20	08:33
4	3	Express	15:00	07:23
5	4	Passenger	08:53	08:09
6	5	Passenger	10:17	08:40
7	6	Express	14:28	06:03
8	7	Express	07:37	06:48
9	8	Express	13:05	06:05
10	9	Express	18:37	08:06

После:

id	train_number	type_of_train	departure_time	travel_time
1	1	Passenger	21:40	08:15
2	9	Express	18:37	08:06
3	3	Express	15:00	07:23
4	6	Express	14:28	06:03
5	8	Express	13:05	06:05
6	2	Passenger	10:20	08:33
7	5	Passenger	10:17	08:40
8	4	Passenger	08:53	08:09
9	7	Express	07:37	06:48
10	0	Express	06:52	07:57