Copyright: Group16

1. Instructions

This file is an instruction file to help test our computing assignment and show the results of our operations.

Since it is a big project, I divide the whole task into seven files, which are corresponding to four exercises.

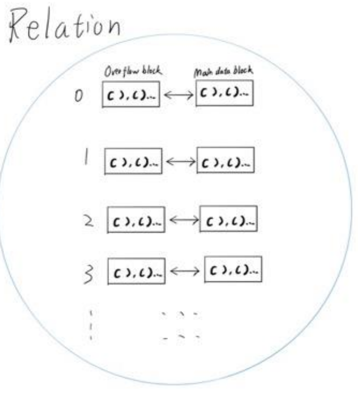
I will show the functions and demo results of different jupyter notebook files at first, then talk about the efficiency of our query part and B/B+ tree implement part. We also have a reference list to strengthen the correctness of our implementation and the credibility of the proof.

2. Details for test ipynb file.

More demo test examples are given in the file.

a. Exercise1.ipynb

This file is related to Exercise1. The basic implementation form of an important data structure is given, Relation. In addition, I provide some demo examples for testing insert, delete, merge and split, etc.



```
#test example
print('before merge')
print(Movie.findpointer((20,'Mad Max',1979,'Australia',88,'action')))
Movie.deletion((21,'Strictly Ballroom',1992,'Australia',94,'comedy'))
print('after merge')
print(Movie.findpointer((20,'Mad Max',1979,'Australia',88,'action')))
#Output
before merge
(1, 0, 0)
merge call
after merge
(2, 0, 0)
```

b. **Exercise2_full.ipynb**

This file is related to exercise2. We implement (i) and (iii) as queries. An important step in query implementation is database conversion, and a sample example of our Database is shown here:

(1,'Role',{1.1:(1.1,'id,"00000343'),1.2:(1.2,'m','Strictly Ballroom'),1.3:(1.3,'y',1992),1.4:(1.4,'d','Scott Hastings')})

The type of identifiers may be different, for value, it is in float type, since we store lots of tuples, which may cause collision if we only set identifiers as integer.

Here is one example for our query:

q1=(('Movie actor',),'satisfy',('not in same scene',))

and the result to eval it is:

[('Amy Irving', 'Catherine Zeta-Jones'), ('Amy Irving', 'Luis Guzman'), ('Stephen Bauer', 'Benicio Del Toro'), ('Erika Christensen', 'Catherine Zeta-Jones'), ('Erika Christensen', 'Benicio Del Toro'), ('Erika Christensen', 'Luis Guzman'), ('Clifton Collins', 'Catherine Zeta-Jones'), ('Clifton Collins', 'Benicio Del Toro'), ('Clifton Collins', 'Luis Guzman'), ('Catherine Zeta-Jones', 'Benicio Del Toro'), ('Catherine Zeta-Jones', 'Jacob Vargas'), ('Catherine Zeta-Jones', 'Michael Douglas'), ('Luis Guzman', 'Michael Douglas')]

c. **Exercise3_bbptreepart.ipynb**

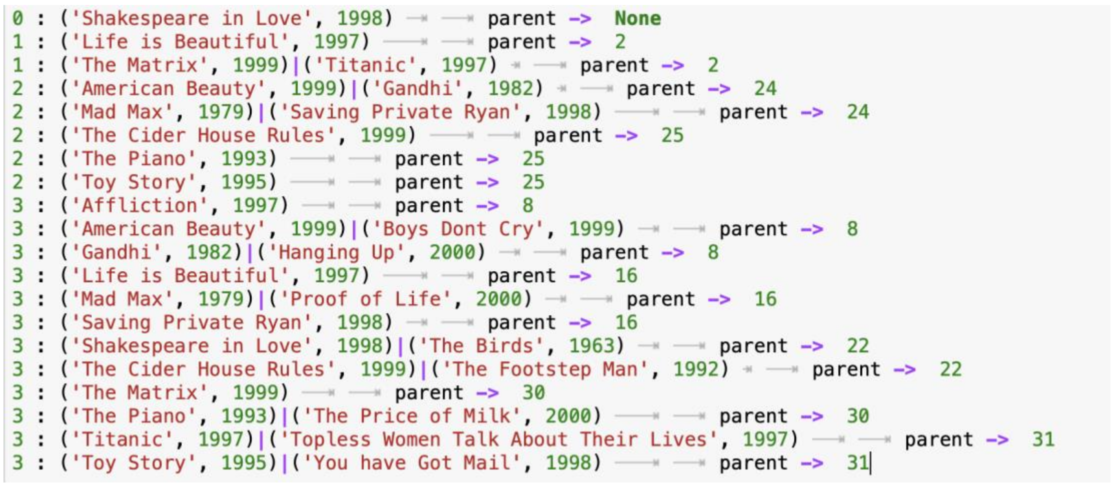
This file is related to (i), (ii), (iii) of Exercise3. In this file, we present the realization of B tree and B+ tree, and combine them with the structure Relation to achieve the function required by this exercise.

Test example:

print('show b+ tree data')

bplustree.show_all_data()

result show:



```
0 : ('Shakespeare in Love', 1998)       parent ->  None
1 : ('Life is Beautiful', 1997)         parent ->  2
1 : ('The Matrix', 1999)|('Titanic', 1997)      parent ->  2
2 : ('American Beauty', 1999)|('Gandhi', 1982)      parent ->  24
2 : ('Mad Max', 1979)|('Saving Private Ryan', 1998)         parent ->  24
2 : ('The Cider House Rules', 1999)         parent ->  25
2 : ('The Piano', 1993)         parent ->  25
2 : ('Toy Story', 1995)         parent ->  25
3 : ('Affliction', 1997)        parent ->  8
3 : ('American Beauty', 1999)|('Boys Dont Cry', 1999)       parent ->  8
3 : ('Gandhi', 1982)|('Hanging Up', 2000)       parent ->  8
3 : ('Life is Beautiful', 1997)         parent ->  16
3 : ('Mad Max', 1979)|('Proof of Life', 2000)       parent ->  16
3 : ('Saving Private Ryan', 1998)       parent ->  16
3 : ('Shakespeare in Love', 1998)|('The Birds', 1963)       parent ->  22
3 : ('The Cider House Rules', 1999)|('The Footstep Man', 1992)      parent ->  22
3 : ('The Matrix', 1999)        parent ->  30
3 : ('The Piano', 1993)|('The Price of Milk', 2000)         parent ->  30
3 : ('Titanic', 1997)|('Topless Women Talk About Their Lives', 1997)        parent ->  31
3 : ('Toy Story', 1995)|('You have Got Mail', 1998)         parent ->  31
```

d. **Exercise3_hashtable.ipynb**

This file is related to (i),(iii),(iv) in exercise3. We use the combination of hash table and relation to achieve the target provided in this exercise.

Test examples:

```
print('original')
print(Scenetable_pr.items)
print(Scenetable_se.items)
print('after delete')
magicdelete(1,Scene,Scenetable_pr,Scenetable_se)
print(Scenetable_pr.items)
print(Scenetable_se.items)
```

show result:

```
original
[None, None, None, ('Rear Window', 1954, 2), None, None, ('Rear Window', 1954, 5), ('The Birds', 1963, 1), ('Rear Window', 1954, 7), None, ('Rear Window', 1954, 1), None, ('Rear Window', 1954, 3), ('Rear Window', 1954, 8), ('The Birds', 1963, 2), None, None, ('Rear Window', 1954, 4), None, ('Rear Window', 1954, 6)]
[None, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, None, None, None, None, None, None, None, None, None]
after delete
[None, None, None, ('Rear Window', 1954, 2), None, None, ('Rear Window', 1954, 5), ('The Birds', 1963, 1), None, None, None, None, ('Rear Window', 1954, 3), ('Rear Window', 1954, 8), ('The Birds', 1963, 2), None, None, ('Rear Window', 1954, 4), None, ('Rear Window', 1954, 6)]
[None, <__main__.HashSet.__Placeholder object at 0x10fe8bf60>, 2, 3, 4, 5, 6, <__main__.HashSet.__Placeholder object at 0x10fe8bf98>, 8, 9, 10, None, None, None, None, None, None, None, None, None]
```

e. **Exercise3_q5.ipynb**

This file is related to (v) in exercise3. We choose to implement this query in exercise2 on the structure b tree plus b+ tree: List all actors who never played in a movie suitable for small children.

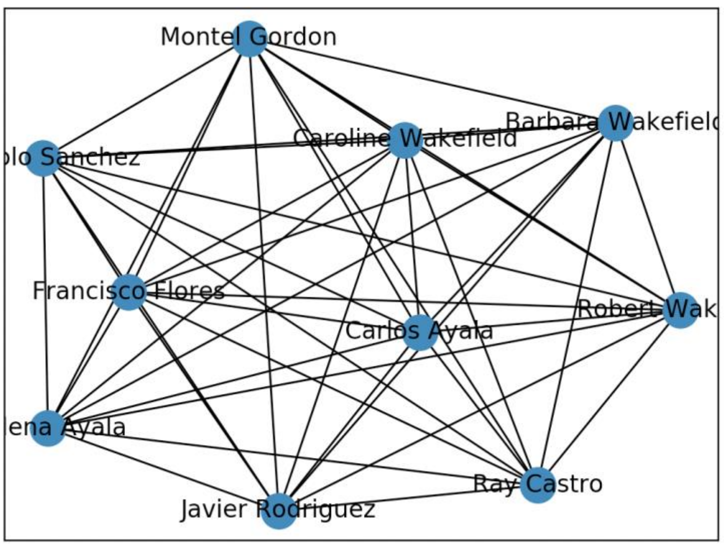
The output is same as second query of b.

f. **Exercise4.ipynb**

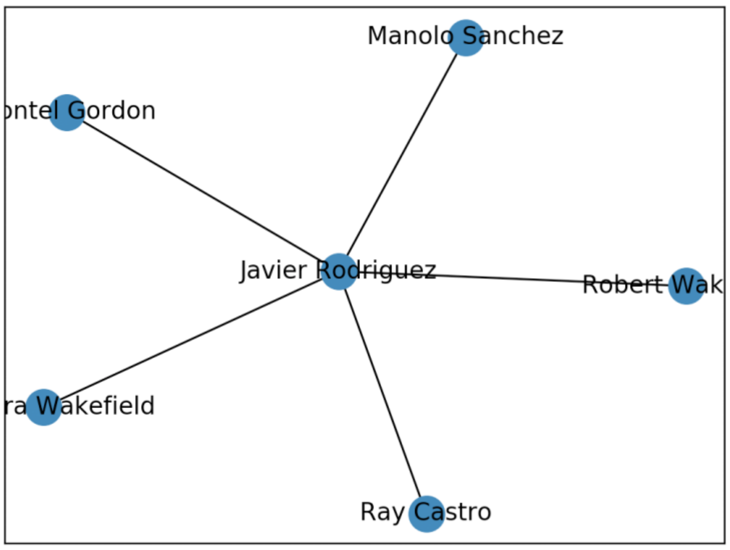
This file is related to Exercise4, as types of graph query. Since in Exercise2 we prove we can change the type of tuple into our database, we skip this step in Exercise4.

Sample query:
q1=(('Traffic',),'contains',('associate',))
q2=(('Traffic',),'contains',('admirer',))
Result (after visualization):

An overview for actor associate relation



An overview for actor admirer relation



g. **bptreetest.ipynb**
   It is a separate file to test the correctness of our b plus tree implement, as the basic version and most important part in CA2.

3. Efficiency

Based on the pursuit of efficiency, our team studied how to optimize the relevant data structure and execute the query algorithm to improve efficiency. We will list running time with our test data in above files. Now the conclusions are as follows:

a. In find operation belongs to Relation structure, we adopt linear search + binary search to decrease the complexity of part of find function from $O(n)$ (using basic find algorithm) into $O(logn)$.

b. For the structure and algorithm where need to sort, we choose to use quick sort algorithm. Quicksort algorithm is stable and of low time complexity, which is $O(nlogn)$.

c. For finding the primary key, and for some of the operations that occur in the algorithm that require the removal of duplicates, we chose to use python's built-in collections implementation. The set class is based on hash table. Therefore, its insert operation and find operation have time complexity as $O(1)$. In addition, its intersection operation s&t has time complexity as $O(\min(len(s), len(t))$. The use of collections greatly reduces the time complexity of our operations. Taking remove duplicates as example, normally the quickest removing operation spends time complexity $O(nlogn)$, but using set it only costs $O(n)$.

d. Because the database created by this group is implemented based on a dictionary, some operations can be done by accessing dictionary elements directly. Dictionary structure in python is also based on hash table. The time complexity of accessing the dictionary element is $O(1)$. This advantage also decreases our running time.

e. For the advantages of using B plus tree rather than B tree, we think they can be concluded as: 1. A single node stores more elements, resulting in fewer IO times of queries; 2. Leaf nodes should be found for all queries, so the query performance is more stable; 3. All leaf nodes form an ordered linked list to facilitate range query.

f. For the exercise4, we initially think it is a task to find spanning forest. However, the space complexity of this algorithm is too large, we usually expect the space complexity can be $O(n)$, since we already create a big graph class. Finally, we choose to finish this task using definition.

4. Reference (In no particular order)

1.Lecture slide 6&6 cont
2.https://blog.csdn.net/hao65103940/article/details/89032538
3.《Fundamentals of Database Systems》
4.https://zhuanlan.zhihu.com/p/24455663
5.https://github.com/WckdAwe/Python-BPlusTree
6.Subieta, Kazimierz & Beeri, Catriel & Matthes, Florian & Schmidt, Joachim. (1997). A Stack-Based Approach to Query Languages. 10.1007/978-1-4471-3577-7_12.
7. https://blog.csdn.net/baoli1008/article/details/48059623