ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Εργασία Μαθήματος

ΣΥΣΤΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Αριθμός εργασίας – Τίτλος εργασίας	ΣΥΣΤΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ
Όνομα φοιτητή	Δημήτρης Αμουργιανός - Λορέντζος
Αρ. Μητρώου	Π16003
Ημερομηνία παράδοσης	27/02/2021

Task 1.3 -Hash Indexes

Το task που έπρεπε να υλοποιήσω ήταν αυτό το ευρετηρίου τύπου hash.

Δημιουργήθηκε το αρχείο hashIndex.py το οποίο περιέχει τις λειτουργίες του ευρετηρίου και έχουν γίνει κάποιες προσθήκες στο αρχείο database.py.

Πιο συγκεκριμένα:

Για το αρχείο hashIndex.py

HashNode

Αρχικά υπάρχει το αντικείμενο HashNode το οποίο περιέχει 4 μεταβλητές.

Η μεταβλητή values δέχεται σαν όρισμα ή την τιμή από την hashfunction ή τις τιμές του πεδίου του πίνακα στον οποίο φτοιάχνεται το ευρετήριο.

Η μεταβλητή child αποθηκεύει την θέση μνήμης ενός άλλου node.

Η μεταβλητή next αποθηκεύει την θέση μνήμης ενός άλλου node.

Η μεταβλητή parent αποθηκεύει την θέση μνήμης του προηγούμενου node.

```
class HashNode:
    '''
    Node abstraction. Represents a single bucket
    '''

    def __init__(self, values, child=None, next=None, parent=None):
        self.values = values
        self.child = child
        self.next = next
        self.parent = parent

    '''
        Insert the value and its ptr/s to the appropriate place (node wise).
        User can input two ptrs to insert to a non leaf node.

        value: the value that we are inserting to the node
        ptr: the ptr of the inserted value (its index for example)
        ptr1: the 2nd ptr (in case the user wants to insert into a nonleaf node for ex)
    '''
```

HashIndex

Το αντικείμενο HashIndex καθορίζει τις λειγουργίες του ευρετηρίου. Περιέχει 3 κύριες μεταβλητές και τις functions που το αφορούν.

Η μεταβλητή b αποθηκεύει το μέγεθος κάθε bucket στο ευρετήριο.

Η μεταβλητή hashed_column αποθηκεύει την στίλη του πίνακα που θα ο παράγοντας ομαδοποίησης του πίνακα για γρηγορότερη ικανοποίηση των ερωτημάτων.

Η μεταβλητή head αποθηκεύει την θέση μνήμης του πρώτου node.

```
class HashIndex:
    def __init__(self, b, column_id):
        '''
        The Hash Index abstraction
        '''
        self.b = b # branching factor
        self.hashed_column = column_id
        self.nodes = None # list of nodes. Every new node is appended here
        self.head = None # the index of the root node
```

Insert

Η συνάρτηση insert κατασκευάζει το ευρετήριο. Δέχεται σαν ορίσματα τις τιμές της στίλης του πίνακα που θα χρησιμοποιηθεί για γρηγορότερη αναζήτηση και τα υπόλοιπα στοιχεία του πίνακα για κάθε τιμή της προηγούμενης στίλλης αντίστοιχα.

Αρχικάελέγχει εάν έχει δοθεί τιμή στην μεταβλητή column και μετά ελέγχει εάν η μεταβλητή head είναι άδια ή όχι. Εάν είναι άδεια τότε σημαίνει ότι είναι το πρώτο στοιχείο και μετά την δημιουργία του το περνάει στην μεταβλητή head.

Εάν η μεταβλητή head δεν είναι άδεια τότε πέρνει το hash της μεταβλητής “column” και ελέγχει εάν υπάρχει. Εά δεν υπάρχει δημιουργεί ένα καινούργιο node και το περνάει στην τελευταία θέση και μετά στην μεταβλητή node περνάει την πρώτη γραμμή του πίνακα της οποίας το hash της τιμής της μεταβλητής column είναι ίδιο με το hash που υπάρχει ήδη στο υποκατασκευή ευρετήριο.

```
def insert(self,column, values):
    # print (values)
    if column is not None:
        NewNode = HashNode(custom_hash_function(column))
        if self.head is None:
            self.head = NewNode
            tmp = HashNode(values)
            self.head.child = tmp
            tmp.parent = self.head
            return

        hash_prefix = self.head
        while hash_prefix:

            sum = 0
            if str(hash_prefix.values) == str(NewNode.values):
                # print("hash: ", hash_prefix.values)
                first_child = hash_prefix.child
                next_child = hash_prefix.child
                while next_child:

                    if next_child.child:
                        next_child = next_child.child
                    else:
                        break
                tmp = HashNode(values)
                next_child.child = tmp
                tmp.parent = next_child
                # print("bucket: ", next_child.child.values)
                return
            if hash_prefix.next:
                hash_prefix = hash_prefix.next
            else:
                break
        hash_prefix.next=NewNode
        tmp = HashNode(values)
        hash_prefix.next.child = tmp
        tmp.parent = hash_prefix.next
```

split_to_buckets

Η συνάρτηση αυτή δέχεται το ευρετήριο και χωρίζει τους κόμβους “παιδιά” της κεντρικής λίστας (hash prefix) σε buckets το μέγεθος των οποίων καθορίζεται από την τιμή της μεταβλητής b.

```
def split_to_buckets(self):
    b = self.b
    c_id = self.hashed_column

    hash_prefix = self.head

    sum = 0
    while hash_prefix is not None:
        sum = 0

        first_child = hash_prefix.child
        next_child = hash_prefix.child
        while next_child:
            sum += 1
            if sum>b:

                parent = next_child.parent
                first_child.next = next_child
                first_child = next_child
                parent.child = None
                sum = 0

            if next_child.child:
                next_child = next_child.child
            else:
                sum = 0
                break

        hash_prefix = hash_prefix.next
```

show

Η συνάρτηση show είναι το visualization για το ευρετήριο. Εμφανίζει κάθε κόμβο της κεντρικής λίστας (hash prefix) και ενδιάμεσα τους κόμβους παιδιά.

```
def show(self):
    print ("\nPrint from show function")

    tmp = self.head
    while tmp is not None:

        print ('\nkey :', tmp.values)
        bucket = tmp.child
        while bucket is not None:
            tmp_child = bucket.child
            print ('  bucket: ')
            print ("  ->", bucket.values)
            while tmp_child is not None:
                print ("  ->", tmp_child.values)
                if tmp_child.parent:
                    tmp_child = tmp_child.child
            bucket = bucket.next
        tmp = tmp.next
```

hash_search

Λειτουργεί με παρόμοιο τρόπο με την συνάρτηση show αλλα δέχεται σαν όρισμα μία τιμη που εάν υπάρχει στην κεντρική λίστα εμφανίζει όλα τα πεδία παιδιά. Δηλαδή εκτελεί την λειτουργία της αναζήτησης.

```
def hash_search(self, key):

    tmp = self.head
    c_id = self.hashed_column
    key_hash = custom_hash_function(key)
    while tmp:
        if str(tmp.values) == str(key_hash):

            bucket = tmp.child
            while bucket is not None:
                if bucket.values[c_id] == key:
                    print ("->", bucket.values)

                tmp_child = bucket.child
                while tmp_child is not None:
                    if tmp_child.values[c_id] == key:
                        print ("->", tmp_child.values)
                    tmp_child = tmp_child.child

                bucket = bucket.next
        tmp = tmp.next
```

hash_join_search

Λειτουργεί με τον ίδιο τρόπο με την hash_search με την μόνη διαφορά ότι δέχεται σαν όρισμα και μια ολόκληρη γραμμή ενός πίνακα και την εμφανίζει μαζί με τις τιμές του ευρετηρίου που ικανοποιούν τις συνθήκες.

```
def hash_join_search(self, key, values):

    tmp = self.head
    c_id = self.hashed_column
    key_hash = custom_hash_function(key)
    while tmp:
        if str(tmp.values) == str(key_hash):

            bucket = tmp.child
            while bucket is not None:
                if bucket.values[c_id] == key:
                    print ("\n->", bucket.values)
                    print (',', values)

                tmp_child = bucket.child
                while tmp_child is not None:
                    if tmp_child.values[c_id] == key:
                        print ("\n->", tmp_child.values)
                        print (',', values)
                    tmp_child = tmp_child.child

                bucket = bucket.next
        tmp = tmp.next
```

hash_function

Δέχεται σαν όρισμα ένα string και αφου το μετατρέψει σε αριθμό εκτελεί τις εξείς πράξεις: hashmod2**(το μήκος του string).

custom_hash_function

Μία μικρή παραλλαγη της από πάνω hash συνάρτησης που αντί για mod2 εκτελεί mod με το μήκος του string.

```
def hash_function(column):
    sum = 0
    column_len = len(column)

    for val in [ord(c) for c in column]:
        sum = sum + val

    hash = sum % 2
    hash = hash ** column_len

    return hash

def custom_hash_function(column):
    sum = 0
    column_len = len(column)

    for val in [ord(c) for c in column]:
        sum = sum + val

    hash = sum % column_len
    hash = hash ** column_len

    return hash
```

Για το αρχείο database.py

Στο αρχείο αυτό έχουν γίνει κάποιες προσθήκες.

Αρχικά στην κορυφή του αρχείου έχει προστεθεί η εξής γραμμή:

from hashIndex import HashIndex

hash_index

Δέχεται σαν ορίσματα το όνομα που θέλουμε να έχει το ευρετήριο, το όνομα και το πεδίο του πίνακα στον οποίο θέλουμε να φτοιάξουμε το ευρετήριο και το μέγεθος των buckets.

Αφού ελέγχει ότι υπάρχουν ο πίνακας και το πεδίο, δημιουργεί ένα HashIndex αντικείμενο και καλει επαναληπτικά την συνάρτηση insert για να δημιουργίσει το ευρετήριο. Στην συνέχεια καλεί την split_to_buckets για να δημιουργηθούν τα buckets, το αποθηκεύει και τέλος καλεί την show για να το εμφανήσει.

```
def hash_index(self, index_name, table_name, column_name, branching):

    ''' ---------------------------------------------------- '''
    '''  Create an index on a specified table with a given name  '''
    '''  index_name -> name of the created index                 '''
    '''  table_name -> table's name (needs to exist in database) '''
    '''  column_name -> name of the column for indexing          '''
    '''  branching -> the size of every bucket                   '''
    ''' ---------------------------------------------------- '''

    db_table = self.tables[table_name]
    column_id = db_table.column_names.index(column_name)
    test_table = table_name

    hash_index = HashIndex(branching, column_id)
    for key in range(len(db_table.data)):
        hash_index.insert(db_table.data[key][column_id], db_table.data[key])

    hash_index.split_to_buckets()
    self._save_index(index_name, hash_index)
    hash_index.show()
```

```
>>> db.hash_index('test_student', 'student', 'dept_name', 3)

Print from show function

key : 0
  bucket:
  -> ['00128', 'Zhang', 'Comp. Sci.', 102]
  -> ['12345', 'Shankar', 'Comp. Sci.', 32]
  -> ['54321', 'Williams', 'Comp. Sci.', 54]
  bucket:
  -> ['76543', 'Brown', 'Comp. Sci.', 58]

key : 78125
  bucket:
  -> ['19991', 'Brandt', 'History', 80]

key : 279936
  bucket:
  -> ['23121', 'Chavez', 'Finance', 110]

key : 16384
  bucket:
  -> ['44553', 'Peltier', 'Physics', 56]
  -> ['45678', 'Levy', 'Physics', 46]
  -> ['70557', 'Snow', 'Physics', 0]
  bucket:
  -> ['98988', 'Tanaka', 'Biology', 120]

key : 243
  bucket:
  -> ['55739', 'Sanchez', 'Music', 38]

key : 59049
  bucket:
  -> ['76653', 'Aoi', 'Elec. Eng.', 60]
  -> ['98765', 'Bourikas', 'Elec. Eng.', 98]
>>>
```

Σαν key είναι οι τιμές της hash prefix λίστας, μετά τα βελάκια είναι τα οιγραμμές του πίνακα και κάθε φορα που ξεκινάει κάποιο καινούργιο bucket εμφανίζεται η αντίστηχη λέξη.

show_hash_index

Δέχεται σαν όρισμα το όνομα το ευρετηρίου που θέλουμε να εμφανήσουμε

```
def show_hash_index(self, index_name):

    ''' ------------------------------------------------------- '''
    '''  Load and Print the specified Hash Index                '''
    '''  index_name -> name of the created index                '''
    ''' ------------------------------------------------------- '''

    load_index = self._load_idx(index_name)
    if load_index:
        load_index.show()
    else:
        print("Index not Found!")
```

```
>>> db.show_hash_index('test_student')

Print from show function

key : 0
  bucket:
  -> ['00128', 'Zhang', 'Comp. Sci.', 102]
  -> ['12345', 'Shankar', 'Comp. Sci.', 32]
  -> ['54321', 'Williams', 'Comp. Sci.', 54]
  bucket:
  -> ['76543', 'Brown', 'Comp. Sci.', 58]

key : 78125
  bucket:
  -> ['19991', 'Brandt', 'History', 80]

key : 279936
  bucket:
  -> ['23121', 'Chavez', 'Finance', 110]

key : 16384
  bucket:
  -> ['44553', 'Peltier', 'Physics', 56]
  -> ['45678', 'Levy', 'Physics', 46]
  -> ['70557', 'Snow', 'Physics', 0]
  bucket:
  -> ['98988', 'Tanaka', 'Biology', 120]

key : 243
  bucket:
  -> ['55739', 'Sanchez', 'Music', 38]

key : 59049
  bucket:
  -> ['76653', 'Aoi', 'Elec. Eng.', 60]
  -> ['98765', 'Bourikas', 'Elec. Eng.', 98]
>>>
```

search_hash_index

Δέχεται σαν όρισμα το όνομα του ευρετηρίου και μία τιμή και εάν υπάρχει στην hash prefix τότε εμφανίζει και όλα τα πεδία παιδιά. Με την βοήθεια της hash_search εκτελεί αναζήτηση στο ευρετήριο.

```
def search_hash_index(self, index_name, val):

    ''' ------------------------------------------------------ '''
    '''  Load specified Hash Index                             '''
    '''  Print values tha matches the given value              '''
    '''  index_name -> name of the created index               '''
    '''  val -> the value to be searched                       '''
    ''' ------------------------------------------------------ '''

    load_index = self._load_idx(index_name)
    if load_index:
        load_index.hash_search(val)
        print("End of Searching!")
    else:
        print("Index not Found!")
```

```
>>> db.search_hash_index('test_student', 'Comp. Sci.')
-> ['00128', 'Zhang', 'Comp. Sci.', 102]
-> ['12345', 'Shankar', 'Comp. Sci.', 32]
-> ['54321', 'Williams', 'Comp. Sci.', 54]
-> ['76543', 'Brown', 'Comp. Sci.', 58]
End of Searching!
>>>
```

hash_join

Η συνάρτηση αυτή αποτελεί ένα συνδιασμό όλων των προηγούμενων. Αρχικά δέχεται σαν όρισμα δύο πίνακες ένα κοινό τους πεδίο και το μέγεθος των buckets. Στην συνέχεια ελέγχει ποιος από τους δύο πίνακες έχει τα λιγότερα στοιχεία. Πάνω στον μικρότερο πίνακα δημιουργεί ένα ευρετήριο και στην συνέχεια εκτελεί μία αναζήτηση και εμφανίζει όλα τα στοιχεία των δύο πινάκων που έχουν τις ίδιες τιμές στο κοινό τους πεδίο.

```
def hash_join(self, table_name_1, table_name_2, column_name, branching):

    ''' ------------------------------------------------------ '''
    '''  Create an index from the smaller table                '''
    '''  then prints the values from both table                '''
    '''  that the values from the common columns matches       '''
    '''  table_name_1 -> the first table's name                '''
    '''  table_name_2 -> the second table's name               '''
    '''  column_name -> name of the column for indexing        '''
    '''  branching -> the size of every bucket                 '''
    ''' ------------------------------------------------------ '''

    db_table_1 = self.tables[table_name_1]
    column_id_1 = db_table_1.column_names.index(column_name)

    db_table_2 = self.tables[table_name_2]
    column_id_2 = db_table_2.column_names.index(column_name)

    if column_id_1 is not None and column_id_2 is not None:
        size_1 = len(db_table_1.data)
        size_2 = len(db_table_2.data)
    else:
        print('Wrong table or column')
        return

    if size_1 < size_2:
        hash_index = HashIndex(branching, column_id_1)
        for key in range(len(db_table_1.data)):
            hash_index.insert(db_table_1.data[key][column_id_1], db_table_1.data[key])
        hash_index.split_to_buckets()

        for key in range(len(db_table_2.data)):
            hash_index.hash_join_search(db_table_2.data[key][column_id_2], db_table_2.data[key])

    else:
        hash_index = HashIndex(branching, column_id_2)
        for key in range(len(db_table_2.data)):
            hash_index.insert(db_table_2.data[key][column_id_2], db_table_2.data[key])
        hash_index.split_to_buckets()

        for key in range(len(db_table_1.data)):
            hash_index.hash_join_search(db_table_1.data[key][column_id_1], db_table_1.data[key])
```

```
>>> db.hash_join('student', 'department', 'dept_name', 3)

-> ['Comp. Sci.', 'Taylor', 100000]
, ['00128', 'Zhang', 'Comp. Sci.', 102]

-> ['Comp. Sci.', 'Taylor', 100000]
, ['12345', 'Shankar', 'Comp. Sci.', 32]

-> ['History', 'Painter', 50000]
, ['19991', 'Brandt', 'History', 80]

-> ['Finance', 'Painter', 120000]
, ['23121', 'Chavez', 'Finance', 110]

-> ['Physics', 'Watson', 70000]
, ['44553', 'Peltier', 'Physics', 56]

-> ['Physics', 'Watson', 70000]
, ['45678', 'Levy', 'Physics', 46]

-> ['Comp. Sci.', 'Taylor', 100000]
, ['54321', 'Williams', 'Comp. Sci.', 54]

-> ['Music', 'Packard', 80000]
, ['55739', 'Sanchez', 'Music', 38]

-> ['Physics', 'Watson', 70000]
, ['70557', 'Snow', 'Physics', 0]

-> ['Comp. Sci.', 'Taylor', 100000]
, ['76543', 'Brown', 'Comp. Sci.', 58]

-> ['Elec. Eng.', 'Taylor', 85000]
, ['76653', 'Aoi', 'Elec. Eng.', 60]

-> ['Elec. Eng.', 'Taylor', 85000]
, ['98765', 'Bourikas', 'Elec. Eng.', 98]

-> ['Biology', 'Watson', 90000]
, ['98988', 'Tanaka', 'Biology', 120]
>>>
```