

Lyft 60 min round

1. Key-Value Database:

Input is a file

Command line

2. **KeyVersionStore**

3. **Autocomplete**

4. **Image LRU**

5. **Excel**

6. **map digit to letter**

Lyft Onsite

In-Memory Key-Value Database

You are to build a data structure for storing integers. You will not persist the database to disk, you will store the data in memory.

For simplicity's sake, instead of dealing with multiple clients and communicating over the network, your program will

Your database should accept the following commands.

SET <name> <value>

Set the variable name to the value value. Neither variable names nor values will contain spaces.

GET <name>

Print out the value of the variable name, or `NULL` if that variable is not set.

UNSET <name>

Unset the variable name, making it just like that variable was never set.

NUMWITHVALUE <value>

Print out the number of variables that are currently set to value. If no variables equal that value, print 0.

END

Exit the program. Your program will always receive this as its last command.

PART 2:

Once your database accepts the above commands and is tested and works, implement commands below

Open a new transaction block. Transaction blocks can be nested (`BEGIN` can be issued inside of an existing

block) but you should get non-nested transactions working first before starting on nested. **A `GET` within a transaction returns the latest value by any command. Any data command that is run outside of a transaction block should commit immediately.**

- ROLLBACK

Undo all of the commands issued in the most recent transaction block, and close the block. Print nothing if successful, or print `NO TRANSACTION` if no transaction is in progress.

- COMMIT

Close all open transaction blocks, permanently applying the changes made in them. Print nothing if successful, or print `NO TRANSACTION` if no transaction is in progress.

Your output should contain the output of the GET and NUMWITHVALUE commands. GET will print out the value of the specified key, or NULL. NUMWITHVALUE will return the number of keys which have the specified value.

申请的是2019软件工程实习 找地里的大哥内推的 我分到的hr特别好 基本秒回邮件
一面是lc tag题 小哥就问了一道秒了聊了会儿天30分. more info on 1point3acres
终面是30min behavioral 90 min programming 60min 正常lc 也是问的tag题 但让我写了三个function细微改条件
签了nda我就不说lc是哪道题了 怕被认出来 好好刷面经就行 但是我把programming的原题贡献出来了
我个人认为我写的代码readability和correctness都是很好的 但是。。因为时间不够 某个function在list是空的时候
忘记加return以至于
接下来会有idx out of range exception 也没来得及写comment 这里要骂一下我的面试官 30分钟我连题都没读懂
问他一些example cases.
他说他也不知道 迟到了5分钟打来说 还用了10+分钟问我简历 因为他找不到google docs的链接了 我最后快交
的时候都急哭了 可定是negative signal
其实这道题地里我见过 但当时好多帖子就说的是什么kv store 而且只有一个帖子是提到了rollback,begin block功
能我根本没看懂 还是希望大家贡献面经的时候说清楚点吧.
准备lyft准备了一周都没有去上课 最后就因为这个少加一句return跪了 哪怕再多给我2分钟我就查到了 心情堵的慌.

Command line

```
1 """
2 1. Start program: python myprogram.py
3 2. Type commands: "BEGIN", "ROLLBACK","COMMIT", "SET A 3", "UNSET A","GET A"
4 * Output will be printed to console.
5 * Add return statements for unit testing,not necessary but save code.
6
7 """
8 class Database(object):
9     def __init__(self):
10         """
11         :param self.store: database
12         :param self.transaction_stack: a stack storing transaction blocks
13         :param self.has_open_block: True if at least one transaction is open
14         """
15         self.store = {}
16         self.transaction_stack = []
17         self.has_open_block = False
18
```

```

19 def operate(self):
20     """
21     Process user commands
22     """
23     while True:
24         commands = input()
25         commands = commands.strip()
26         if not commands:
27             print("Please enter a valid command.")
28         else:
29             commands = commands.split()
30             if commands[0] == "BEGIN":
31                 self.begin_block()
32             elif commands[0] == "ROLLBACK":
33                 self.rollback()
34             elif commands[0] == "COMMIT":
35                 self.commit()
36             elif commands[0] == "SET" and len(commands) == 3:
37                 name, value = commands[1], commands[2]
38                 self.set(name, value)
39
40             elif commands[0] == "GET" and len(commands) == 2:
41                 name = commands[1]
42                 self.get(name)
43
44             elif commands[0] == "UNSET" and len(commands) == 2:
45                 name = commands[1]
46                 self.unset(name)
47
48             elif commands[0] == "NUMWITHVALUE" and len(commands) == 2:
49                 value = commands[1]
50                 self.num_with_value(value)
51
52             elif commands[0] == "END":
53                 return
54             else:
55                 return "Please enter a valid command."
56
57 def get(self, name):
58     if name not in self.store:
59         print(None)
60         return None
61     else:
62         print(self.store[name])
63         return self.store[name]
64
65 def set(self, name, value, undo = True):
66     """
67     If name in db and value != previous value, set to new value and record "set, previous
68     value" for future rollback
69     If name not in db, set value and record "unset"

```

```

69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
"""
if name in self.store:
    prev_val = self.store[name]
    self.store[name] = value
    if prev_val != value and undo:
        self.prepare_rollback("set", (name, prev_val))
    else:
        self.store[name] = value
        self.prepare_rollback("unset", (name))

def unset(self, name, undo = True):
    """
    If name in db, remove it and record "(set, previous value)" for future rollback
    """
    if name in self.store:
        prev_val = self.store[name]
        if undo:
            self.prepare_rollback("set", (name, prev_val))
        del self.store[name]

def prepare_rollback(self, operation, info):
    """
    Append undo commands to the latest transaction block on the stack
    :param operation: "set"/"unset"
    :param info: (name,previous value) for "set"; (name) for "unset"
    """
    if self.has_open_block and self.transaction_stack:
        last_block = self.transaction_stack[-1]
        last_block.append((operation, info))

def num_with_value(self, val):
    """
    Print out the number of variables that are currently set to value
    """
    count = 0
    for k, v in self.store.items():
        if v == val:
            count += 1
    print(count)
    return count

def rollback(self):
    """
    Undo all commands issued in the latest transaction block, and close the block
    Print nothing if successful, or print `NO TRANSACTION` if no transaction is in
progress
    """
    if not self.has_open_block:
        print("NO TRANSACTION")
        return "NO TRANSACTION"

```

```

119
120     else:
121         last_block = self.transaction_stack.pop()
122         for (operation, info) in last_block:
123             if operation == "set":
124                 # Set undo to False, we don't want to undo rollback
125                 # info: (name,previous value)
126                 self.set(*info, undo = False)
127             elif operation == "unset":
128                 # info: name
129                 self.unset(info[0], undo = False)
130
131     def begin_block(self):
132         self.transaction_stack.append([])
133         self.has_open_block = True
134
135     def commit(self):
136         """
137         Close all open transaction blocks, permanently applying the changes made in them.
138         Print nothing if successful, or print `NO TRANSACTION` if no transaction is in
139         progress.
140         """
141         if not self.transaction_stack:
142             print("NO TRANSACTION")
143             return "NO TRANSACTION"
144         else:
145             self.transaction_stack = []
146             self.has_open_block = False
147
148
149 import unittest
150
151 class Test(unittest.TestCase):
152     def test_set(self):
153         db = Database()
154         db.set("A",10)
155         self.assertEqual(db.get("A"), 10)
156         db.set("A",20)
157         self.assertEqual(db.get("A"), 20)
158
159     def test_unset(self):
160         db = Database()
161         db.set("A",10)
162         db.unset("A")
163         self.assertEqual(db.get("A"), None)
164
165     def test_num_with_value(self):
166         db = Database()
167         db.set("A",10)
168         db.set("B",10)

```

```

169         self.assertEqual(db.num_with_value(10), 2)
170         db.set("B",5)
171         self.assertEqual(db.num_with_value(10), 1)
172
173
174     def test_rollback(self):
175         db = Database()
176         db.set("A",10)
177         db.begin_block()
178         db.set("A",20)
179         db.begin_block()
180         db.set("A",30)
181         self.assertEqual(db.get("A"), 30)
182         db.rollback()
183         self.assertEqual(db.get("A"), 20)
184         db.rollback()
185         self.assertEqual(db.get("A"), 10)
186
187     def test_commit(self):
188         db = Database()
189         db.set("A",10)
190         db.begin_block()
191         db.set("A",20)
192         db.commit()
193         self.assertEqual(db.get("A"), 20)
194
195     def test_commit_rollback(self):
196         db = Database()
197         db.set("A",10)
198         db.begin_block()
199         db.set("A",20)
200         db.begin_block()
201         db.set("A",30)
202         db.commit()
203         self.assertEqual(db.get("A"), 30)
204         self.assertEqual(db.rollback(), "NO TRANSACTION")
205         self.assertEqual(db.commit(), "NO TRANSACTION")
206
207     def test_rollback_commit(self):
208         db = Database()
209         db.set("A",10)
210         db.begin_block()
211         db.set("A",20)
212         db.begin_block()
213         db.set("A",30)
214         db.rollback()
215         self.assertEqual(db.get("A"), 20)
216         db.rollback()
217         db.commit()
218         self.assertEqual(db.get("A"), 10)
219         self.assertEqual(db.commit(), "NO TRANSACTION")

```

```

220
221
222 if __name__ == '__main__':
223     db = Database()
224     unittest.main()

```

Input is a file

```

1 class Database(object):
2     def __init__(self):
3         """
4         :param self.store: database
5         :param self.transaction_stack: a stack storing transaction blocks
6         :param self.has_open_block: True if at least one transaction is open
7         """
8         self.store = {}
9         self.transaction_stack = []
10        self.has_open_block = False
11
12    def operate(self, commands):
13        """
14        Process user commands
15        """
16        commands = commands.strip()
17        if not commands:
18            return "Please enter a valid command."
19
20        else:
21            commands = commands.split()
22            if commands[0] == "BEGIN":
23                return self.begin_block()
24            elif commands[0] == "ROLLBACK":
25                return self.rollback()
26
27            elif commands[0] == "COMMIT":
28                return self.commit()
29            elif commands[0] == "SET" and len(commands) == 3:
30                name, value = commands[1], commands[2]
31                self.set(name, value)
32
33            elif commands[0] == "GET" and len(commands) == 2:
34                name = commands[1]
35                return self.get(name)
36
37            elif commands[0] == "UNSET" and len(commands) == 2:
38                name = commands[1]
39                self.unset(name)
40
41            elif commands[0] == "NUMWITHVALUE" and len(commands) == 2:
42                value = commands[1]

```

```

43         return self.num_with_value(value)
44
45     elif commands[0] == "END":
46         return
47     else:
48         return "Please enter a valid command."
49
50     def get(self, name):
51         if name not in self.store:
52             return "NULL"
53         else:
54             return self.store[name]
55
56     def set(self, name, value, undo = True):
57         """
58         If name in db and value != previous value, set to new value and record "set, previous
value" for future rollback
59         If name not in db, set value and record "unset"
60         """
61         if name in self.store:
62             prev_val = self.store[name]
63             self.store[name] = value
64             if prev_val != value and undo:
65                 self.prepare_rollback("set", (name, prev_val))
66         else:
67             self.store[name] = value
68             self.prepare_rollback("unset", (name))
69
70     def unset(self, name, undo = True):
71         """
72         If name in db, remove it and record "(set, previous value)" for future rollback
73         """
74         if name in self.store:
75             prev_val = self.store[name]
76             if undo:
77                 self.prepare_rollback("set", (name, prev_val))
78             del self.store[name]
79
80     def prepare_rollback(self, operation, info):
81         """
82         Append undo commands to the latest transaction block on the stack
83         :param operation: "set"/"unset"
84         :param info: (name, previous value) for "set"; (name) for "unset"
85         """
86         if self.has_open_block and self.transaction_stack:
87             last_block = self.transaction_stack[-1]
88             last_block.append((operation, info))
89
90
91     def num_with_value(self, val):
92         """

```



```

93     Print out the number of variables that are currently set to value
94     """
95     count = 0
96     for k, v in self.store.items():
97         if v == val:
98             count += 1
99     return count
100
101     def rollback(self):
102         """
103         Undo all commands issued in the latest transaction block, and close the block
104         Print nothing if successful, or print `NO TRANSACTION` if no transaction is in
progress
105         """
106         if not self.has_open_block:
107             return "NO TRANSACTION"
108
109         else:
110             last_block = self.transaction_stack.pop()
111             for (operation, info) in last_block:
112                 if operation == "set":
113                     # Set undo to False, we don't want to undo rollback
114                     # info: (name,previous value)
115                     self.set(*info, undo = False)
116                 elif operation == "unset":
117                     # info: name
118                     self.unset(info[0], undo = False)
119
120     def begin_block(self):
121         self.transaction_stack.append([])
122         self.has_open_block = True
123
124     def commit(self):
125         """
126         Close all open transaction blocks, permanently applying the changes made in them.
127         Print nothing if successful, or print `NO TRANSACTION` if no transaction is in
progress.
128         """
129         if not self.transaction_stack:
130             return "NO TRANSACTION"
131         else:
132             self.transaction_stack = []
133             self.has_open_block = False
134
135
136
137 import unittest
138
139 class Test(unittest.TestCase):
140     def test_set(self):
141         db = Database()

```

```

142     db.set("A",10)
143     self.assertEqual(db.get("A"), 10)
144     db.set("A",20)
145     self.assertEqual(db.get("A"), 20)
146
147     def test_unset(self):
148         db = Database()
149         db.set("A",10)
150         db.unset("A")
151         self.assertEqual(db.get("A"), "NULL")
152
153     def test_num_with_value(self):
154         db = Database()
155         db.set("A",10)
156         db.set("B",10)
157         self.assertEqual(db.num_with_value(10), 2)
158         db.set("B",5)
159         self.assertEqual(db.num_with_value(10), 1)
160
161
162     def test_rollback(self):
163         db = Database()
164         db.set("A",10)
165         db.begin_block()
166         db.set("A",20)
167         db.begin_block()
168         db.set("A",30)
169         self.assertEqual(db.get("A"), 30)
170         db.rollback()
171         self.assertEqual(db.get("A"), 20)
172         db.rollback()
173         self.assertEqual(db.get("A"), 10)
174
175     def test_commit(self):
176         db = Database()
177         db.set("A",10)
178         db.begin_block()
179         db.set("A",20)
180         db.commit()
181         self.assertEqual(db.get("A"), 20)
182
183     def test_commit_rollback(self):
184         db = Database()
185         db.set("A",10)
186         db.begin_block()
187         db.set("A",20)
188         db.begin_block()
189         db.set("A",30)
190         db.commit()
191         self.assertEqual(db.get("A"), 30)
192         self.assertEqual(db.rollback(), "NO TRANSACTION")

```

```

193         self.assertEqual(db.commit(), "NO TRANSACTION")
194
195     def test_rollback_commit(self):
196         db = Database()
197         db.set("A",10)
198         db.begin_block()
199         db.set("A",20)
200         db.begin_block()
201         db.set("A",30)
202         db.rollback()
203         self.assertEqual(db.get("A"), 20)
204         db.rollback()
205         db.commit()
206         self.assertEqual(db.get("A"), 10)
207         self.assertEqual(db.commit(), "NO TRANSACTION")
208
209 import sys
210 if __name__ == '__main__':
211     unittest.main(argv=['first-arg-is-ignored'],exit=False)
212     if len(sys.argv) != 2:
213         print("Please inform the file name")
214         exit(1)
215     f1 = sys.argv[1]
216     with open(f1) as f1:
217         db = Database()
218         with open("output.txt", "w") as f2:
219             for line in f1:
220                 result = db.operate(line)
221                 if result:
222                     print(result, file= f2)

```

KeyVersionStore

Everytime we add a new key or change the value of that key, we increment the version. Return the value of a specific key in a specific version.

1. `PUT <key> <value>`

Set the key name to the value. Key strings will not contain spaces. Print out the version number, the key and the value as `PUT(#<version number> <key> = <value>)`. The first write in the file should be version number `1`, the second should be version number `2`, etc.

2. `GET <key>`

Print out the key and the last value of the key, or `<NULL>` if that key has never been set as in: `GET <key> = <value>`

3. `GET <key> <version number>`

Print out the key, the version number and the value of key as it was at the time of the version number,

or `<NULL>` if that key was not set at that time, as in `GET <key>(#version) = <value>`. If the version number has not yet been recorded, return the most recent value for the key. See below for examples of formatted output.

1

Image LRU

<https://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=195179&highlight=lyft>

```
1 import io
2 import os
3 import sys
4 import unittest
5 import wget
6
7 class LRUCache:
8     def __init__(self, capacity):
9         """
10         :param self.LRUCache: a dictionary for storing url-image pairs in the LRU Cache
11         :param self.queue: a double-ended queue with the most recently used item appended to
12         the end,
13         an item in the queue denotes a usage of the item, not the item
14         itself
15         :param self.totalSize: the total size of images in LRUCache
16         :param self.counter: a frequency counter for the urls in LRU Cache
17         :param self.cacheSize: maximum capacity of the LRU Cache
18         """
19         from collections import deque, Counter
20         self.LRUCache = {}
21         self.queue = deque()
22         self.totalSize = 0
23         self.counter = Counter()
24         self.cacheSize = capacity
25
26     def getImage(self, url):
27         """
28         If url in cache, print IN CACHE
29         Else call downloadImage(url)
30         """
31         url = url.strip()
32         if not url:
33             print("Empty url")
34             return
35         if url in self.LRUCache:
36             # Append the most recently used item to the end of the self.queue
37             self.queue.append(url)
38             self.counter[url] += 1
39             print(url + " IN CACHE",)
```

```

40         self.downloadImage(url)
41
42
43     def downloadImage(self, url):
44         """
45         Download the image from the url
46         If the cache cannot hold the image, it will use LRU eviction policy to evict the
existing images.
47         If the image size is greater than the maximum cache size then output an error
message
48         Else print DOWNLOADED + size of image
49         :type url: string
50         :type image: image object
51         :return: void
52         """
53
54         image = self.downloadHelper(url)
55         size = self.getImgSize(image)
56         if size > self.cacheSize:
57             print("Image size exceeds cache size")
58             return
59         else:
60             print(url + " DOWNLOADED " + str(size))
61
62         self.LRUCache[url] = (image, size)
63         self.queue.append(url)
64         self.totalSize += size
65
66
67         while self.totalSize > self.cacheSize:
68             URL = self.queue.popleft()
69             IMG, SIZE = self.LRUCache[URL]
70             self.counter[URL] -= 1
71             # If an item becomes the least recently used,remove it from self.LRUCache.
72             if self.counter[URL] == 0:
73                 self.totalSize -= SIZE
74                 del self.LRUCache[URL]
75                 del self.counter[URL]
76
77
78     def downloadHelper(self, url):
79         import wget
80         try:
81             image = wget.download(url)
82         except Exception as e:
83             print("Exception: ",e," at ",url)
84         return image
85     def getImgSize(self, image):
86         return os.stat(image).st_size
87
88

```

```

89
90 class Test(unittest.TestCase):
91     def downloadHelper(self, url):
92         import wget
93         try:
94             image = wget.download(url)
95         except Exception as e:
96             print("Exception: ",e," at ",url)
97         return image
98
99     def getImgSize(self, image):
100         return os.stat(image).st_size
101
102     def test(self):
103         actual = io.StringIO()
104         sys.stdout = actual          # and redirect stdout.
105
106         imageCache = LRUCache(580000)
107         urls = [ "http://i.imgur.com/xGmX4h3.jpg", "http://i.imgur.com/IUfsijF.jpg",
108 "http://i.imgur.com/xGmX4h3.jpg",
109 "http://i.imgur.com/IUfsijF.jpg",
110 "http://i.imgur.com/xGmX4h3.jpg","https://i.imgur.com/HsUw6GN.jpg"]
111
112         imageCache.getImage(urls[0])
113         sys.stdout = sys.__stdout__          # Reset redirect
114         size = self.getImgSize(self.downloadHelper(urls[0]))
115         self.assertEqual(actual.getvalue().strip(), urls[0] + " DOWNLOADED " + str(size))
116
117         actual = io.StringIO()
118         sys.stdout = actual          # and redirect stdout.
119
120         imageCache.getImage(urls[1])
121         sys.stdout = sys.__stdout__          # Reset redirect
122         size = self.getImgSize(self.downloadHelper(urls[1]))
123         self.assertEqual(actual.getvalue().strip(), urls[1] + " DOWNLOADED " + str(size))
124
125         actual = io.StringIO()
126         sys.stdout = actual          # and redirect stdout.
127
128         imageCache.getImage(urls[1])
129         sys.stdout = sys.__stdout__          # Reset redirect
130         self.assertEqual(actual.getvalue().strip(), urls[1] + " IN CACHE")
131
132
133 if __name__ == "__main__":
134     #unittest.main()
135     orig_stdout = sys.stdout
136     outputFile = open("output.txt", 'w')
137     sys.stdout = outputFile

```

```

138 with open('url.txt',encoding="utf-8") as f1:
139     capacity = next(f1).strip()
140     imageCache = LRUCache(int(capacity))
141     for line in f1:
142         imageCache.getImage(line)
143 sys.stdout = orig_stdout
144 outputFile.close()
145

```

Autocomplete words

sentence: <https://leetcode.com/problems/design-search-autocomplete-system/>

```

1 class Autocomplete:
2     def __init__(self, words, ranks):
3         """
4         Build the trie with using prefixes of the input words
5         :param words: List[str]
6         :param ranks: List[int]
7         """
8         from collections import defaultdict
9         self.trie = defaultdict(list)
10        self.rank = defaultdict(int)
11        for i in range(len(words)):
12            self.rank[words[i]] = ranks[i]
13            for j in range(1, len(words[i])+1):
14                self.trie[words[i][:j]].append(words[i])
15
16
17        def input(self, word):
18            """
19            :param word: a test word string
20            :rtype: a list of possible words that have the same prefix as the test word
21            """
22            result = []
23            if word not in self.trie:
24                result.append("This prefix doesn't exist in the autocomplete system.")
25            else:
26                # print candidate words in the order of their ranks
27                self.trie[word].sort(key = lambda x: (self.rank[x], x))
28                for i in range(len(self.trie[word])):
29                    result.append(self.trie[word][i] + " " + str(self.rank[self.trie[word][i]]))
30            return result
31
32
33 if __name__ == '__main__':
34     ranks, words = [], []
35     with open("input.txt") as f1:
36         for line in f1:

```

```

37         line = line.strip()
38         if line:
39             word, rank = line.split(" ")
40             ranks.append(int(rank))
41             words.append(word)
42
43     autocomplete = Autocomplete(words, ranks)
44     with open("test.txt") as test:
45         with open("output.txt", "w") as output:
46             for line in test:
47                 word = line.strip()
48                 if word:
49                     result = autocomplete.input(word)
50                     print(word + ":", file=output)
51                     for res in result:
52                         print(res, file=output)
53                     print(" ", file=output)

```

```

1  #版本2
2  class Autocomplete:
3      def __init__(self, words):
4          """
5          Build the trie with using prefixes of the input words
6          :param words: List[str]
7          :param ranks: List[int]
8          """
9          from collections import defaultdict
10         self.trie = defaultdict(list)
11         for i in range(len(words)):
12             for j in range(1, len(words[i])+1):
13                 self.trie[words[i][:j]].append(words[i])
14
15
16         def input(self, word):
17             """
18             :param word: a test word string
19             :rtype: a list of possible words that have the same prefix as the test word
20             """
21             result = []
22             if word not in self.trie:
23                 result.append("This prefix doesn't exist in the autocomplete system.")
24             else:
25                 # print candidate words in the order of their ranks
26                 for i in range(len(self.trie[word])):
27                     result.append(self.trie[word][i])
28             return result
29
30
31 if __name__ == '__main__':
32     words = []
33     testWords = []

```



```

34     with open("input.txt") as f1:
35         numWords = int(next(f1))
36         numTestWords = int(next(f1))
37         for _ in range(numWords):
38             word = next(f1)
39             word = word.strip()
40             if word:
41                 words.append(word)
42         for _ in range(numTestWords):
43             word = next(f1)
44             word = word.strip()
45             if word:
46                 testWords.append(word)
47     autocomplete = Autocomplete(words)
48     with open("output.txt", "w") as output:
49         for word in testWords:
50             result = autocomplete.input(word)
51             print(result, file = output)

```

6. map digit to letter

第二题是用trie 就是给一个file 里面上半部分是单词下半部分是一些数字的组合

AA
WOT
YOU
ME
968
63
12345

要求output是

968: you, wot

63: me

12345: no result

. more info on 1point3acres

就是把数字对应的字母能组成的单词找出来， 数字对应的是电话号码key pad上的character

```

from collections import defaultdict
class Trie():
    def __init__(self, val):
        self.val = val
        self.children = defaultdict(Trie)
        self.number = ""

    def check(text):
        dic = {"A":2,
"B":2, "C":2, "D":3, "E":3, "F":3, "G":4, "H":4, "I":4, "J":5, "K":5, "L":5, "M":6, "N":6, "O":6, "P":
:7, "Q":7, "R":7, "S":7,
        "T":8, "U":8, "V":8, "W":9, "X":9, "Y":9, "Z":9}
        words = set()
        numbers = set()

```

```

with open(text) as f:
    content = f.readlines()
content = [s.strip() for s in content if s]
print(content)
for w in content:
    if w.isupper():
        words.add(w)
    elif w.isdigit():
        numbers.add(w)
root = Trie(0)
res = defaultdict(list)
print(words, numbers)
for word in words:
    tmp = root
    for l in word:
        if l not in tmp.children:
            node = Trie(l)
            tmp.children[l] = node
            node.number = tmp.number + str(dic[l])
            tmp = tmp.children[l]

    if tmp.number in numbers:
        res[tmp.number].append(word)
return res

```

Column	Value
A1	1
A2	A1 + 4
A3	A1 + A2 + 10
A1	A3 - 5 (throw exception, because of circular reference. Remember: A3 was A1 + A2 + 10)

```

put('A1', '1')
put('A2', 'A1+4')
put('A3', 'A1+A2+10')
put('A3', 'A1 - 5') (throw exception, because of circular reference. Remember: A3 was A1 + A2 + 10)

```

```

Get('A1') => 1
Get('A2') => 5
Get('A3') => 16

```

第一轮90分钟上机coding：白人小哥，题目是设计excel，先讲思路，其实主要是处理dependency的问题，比如 $A1=B1+C1$, $B1=D1+1$ 这种情况如何处理。我的想法跟小白有点不一样，但也是work的，结果这个小白非要和我追根问底，最后我就采用了小白的思路，尼玛这时候已经30分钟过去了。然后小白就走了，留下我一个人在那写code。C++选手这时候就吃亏了，parse传进来的公式什么的就比较麻烦，小白你知道什么库的话可以随便用，但我不知道，只能用string那些硬写，所以代码量还挺大的。到剩15分钟的小白又回来了，跟我讨论，让我解释我的code，到这时候还有一块没写完，更别说测试代码了。最后提一句，我是用的他们的loan laptop写的，上面的gcc

编译器都不支持C++11，非常坑爹。

是 $A=B+C$ 或 $A=B-C$ 的格式，支持+-两种就行了，主要是这里B和C可以是常数或者表达式的，A必须是表达式，然后要自己处理string parse