Practical 4: Extendsible Hashing

Gahan Saraiya, 18MCEC10

18mcec10@nirmauni.ac.in

I. Introduction

Extendsible Hashing is a dynamic hash system which treats hash as bit string. It has hierarchical nature as it extends buffer as needed. Thus buckets are added dynamically.

II. Logic

- Create Global bucket and initialize with single empty local bucket
- Add item in bucket
- To add item Least Significant bits (LSB) are considered to avoid overhead of recalculating padding MSB (most significant bit)
- until bucket overflows it can accommodate data
- When bucket is full global bucket capacity is to be increased by factor of two and overflowed local buckets are to be spitted

III. Implementation

The code is implemented in Python as below

```python
# coding=utf-8
import random
import math

__author__ = "Gahan Saraiya"
__url__ = "https://github.com/gahan9/DS_lab/blob/master/practical_4/extendsibleHashing.py"

DEBUG = False


def print_log(*args):
    if DEBUG:
        print("DEBUG: ", *args, sep=" <|> ")


class Bucket(object):
    def __init__(self, bucket_size=2):
```

```
        self.bucket_size = bucket_size
        self.data = {}
        self.total_data = 0  # local depth

    def is_full(self):
        return len(self.data) >= self.bucket_size

    def add(self, key, val):
        self.data[key] = val

    def get(self, key):
        return self.data.get(key)

    @staticmethod
    def int_to_bin(data, pad_digits):
        return bin(data).split('b')[-1].zfill(pad_digits)

    def show(self):
        if self.data:
            _max = int(math.log(max(self.data), 2)) + 1
            for k, v in self.data.items():
                print(self.int_to_bin(k, _max), v)

    def __repr__(self):
        return "{}".format(self.data) if self.data else ''


class GlobalBucket(object):
    def __init__(self, bucket_size=2):
        self.bucket_size = bucket_size  # max size of each bucket
        self.total_global_data = 0  # global depth
        bucket = Bucket(self.bucket_size)
        self.buckets = [bucket]  # list of buckets in global buckets

    def get_bucket(self, key):
        return self.buckets[key & ((1 << self.total_global_data) - 1)]

    def add(self, key, val):
        print_log("adding key: {} and value: {}".format(key, val))
        bucket = self.get_bucket(key)
        print_log("bucket status is full?: {}", bucket.is_full())
        if bucket.is_full() and bucket.total_data == self.total_global_data:
            self.total_global_data += 1
            self.buckets *= 2
            print_log("buckets: {}".format(self.buckets))
        if bucket.is_full() and bucket.total_data < self.total_global_data:
            bucket.add(key, val)
            bucket1 = Bucket(self.bucket_size)
```

```
66              bucket2 = Bucket(self.bucket_size)
67              for k, v in bucket.data.items():
68                  # print_log("key", k, "value", v)
69                  if ((k & ((1 << self.total_global_data) - 1)) >>
                    ↪  bucket.total_data) & 1 == 1:
70                      bucket2.add(k, v)
71                  else:
72                      bucket1.add(k, v)
73              for idx, value in enumerate(self.buckets):
74                  # print_log("idx", idx)
75                  if value == bucket:
76                      if (idx >> bucket.total_data) & 1 == 1:
77                          self.buckets[idx] = bucket2
78                      else:
79                          self.buckets[idx] = bucket1
80              bucket2.total_data = bucket1.total_data = bucket.total_data + 1
81          else:
82              bucket.add(key, val)
83          # print("bucket after adding", val, "operation: ", bucket.data)
84
85      def get(self, key):
86          return self.get_bucket(key).get(key)
87
88      def __repr__(self):
89          return ", ".join("{}".format(b) for b in self.buckets if b.__repr__())
90
91
92  def test(input_nums=10):
93      """
94      test function to test the implementation
95      :param input_nums: number of inputs to be added
96      :return:
97      """
98      BUCKET_SIZE = 5  # defining single bucket size
99      g = GlobalBucket(BUCKET_SIZE)
100     inputs = [random.randint(1, 1000) for i in range(input_nums)]
101     print("Bucket Size: ", BUCKET_SIZE)
102     print("Total Inputs : ", input_nums)
103     print("Input Sequence: ", inputs)
104     for i in inputs:
105         print_log("*>Adding {} in to bucket".format(i))
106         g.add(i, i)
107     print("-"*40)
108     print("global bucket > ", g)
109     print("global depth > ", g.total_global_data)
110     for _bucket in g.buckets:
111         if _bucket.__repr__():
```

```
112             print("-----Exploring bucket: {} with depth : {}".format(_bucket,
                 ↪  _bucket.total_data))
113             _bucket.show()
114
115
116 if __name__ == "__main__":
117     TEST_NUM = 25
118     import sys
119     if len(sys.argv) > 1:
120         try:
121             TEST_NUM = int(sys.argv[1])
122         except:
123             pass
124     test(TEST_NUM)
```

Output 1

```
Bucket Size:  5
Total Inputs :  5
Input Sequence:  [653, 917, 981, 736, 882]
----------------------------------------
global bucket >  {653: 653, 917: 917, 981: 981, 736: 736, 882: 882}
global depth >  0
-----Exploring bucket: {653: 653, 917: 917, 981: 981, 736: 736, 882: 882} with
 ↪  depth : 0
1010001101 653
1110010101 917
1111010101 981
1011100000 736
1101110010 882
```

Output 2

```
Bucket Size:  5
Total Inputs :  10
Input Sequence:  [3, 733, 199, 617, 233, 161, 393, 489, 622, 699]
----------------------------------------
global bucket >  {622: 622}, {617: 617, 233: 233, 161: 161, 393: 393, 489: 489},
 ↪  {622: 622}, {3: 3, 199: 199, 699: 699}, {622: 622}, {733: 733}, {622: 622},
 ↪  {3: 3, 199: 199, 699: 699}
global depth >  3
-----Exploring bucket: {622: 622} with depth : 1
1001101110 622
-----Exploring bucket: {617: 617, 233: 233, 161: 161, 393: 393, 489: 489} with
 ↪  depth : 3
```

```
1001101001 617
0011101001 233
0010100001 161
0110001001 393
0111101001 489
-----Exploring bucket: {622: 622} with depth : 1
1001101110 622
-----Exploring bucket: {3: 3, 199: 199, 699: 699} with depth : 2
0000000011 3
0011000111 199
1010111011 699
-----Exploring bucket: {622: 622} with depth : 1
1001101110 622
-----Exploring bucket: {733: 733} with depth : 3
1011011101 733
-----Exploring bucket: {622: 622} with depth : 1
1001101110 622
-----Exploring bucket: {3: 3, 199: 199, 699: 699} with depth : 2
0000000011 3
0011000111 199
1010111011 699
```

Output 3

```
Bucket Size:  5
Total Inputs :  20
Input Sequence:  [583, 654, 642, 789, 107, 953, 719, 126, 943, 270, 141, 575, 586,
 ↪  659, 123, 805, 55, 624, 468, 247]
----------------------------------------
global bucket >  {624: 624, 468: 468}, {789: 789, 953: 953, 141: 141, 805: 805},
 ↪ {654: 654, 642: 642, 126: 126, 270: 270, 586: 586}, {107: 107, 659: 659, 123:
 ↪  123}, {624: 624, 468: 468}, {789: 789, 953: 953, 141: 141, 805: 805}, {654:
 ↪  654, 642: 642, 126: 126, 270: 270, 586: 586}, {583: 583, 55: 55, 247: 247},
 ↪ {624: 624, 468: 468}, {789: 789, 953: 953, 141: 141, 805: 805}, {654: 654,
 ↪  642: 642, 126: 126, 270: 270, 586: 586}, {107: 107, 659: 659, 123: 123}, {624:
 ↪  624, 468: 468}, {789: 789, 953: 953, 141: 141, 805: 805}, {654: 654, 642: 642,
 ↪  126: 126, 270: 270, 586: 586}, {719: 719, 943: 943, 575: 575}
global depth >  4
-----Exploring bucket: {624: 624, 468: 468} with depth : 2
1001110000 624
0111010100 468
-----Exploring bucket: {789: 789, 953: 953, 141: 141, 805: 805} with depth : 2
1100010101 789
1110111001 953
0010001101 141
1100100101 805
```

```
-----Exploring bucket: {654: 654, 642: 642, 126: 126, 270: 270, 586: 586} with
↪  depth : 2
1010001110 654
1010000010 642
0001111110 126
0100001110 270
1001001010 586
-----Exploring bucket: {107: 107, 659: 659, 123: 123} with depth : 3
0001101011 107
1010010011 659
0001111011 123
-----Exploring bucket: {624: 624, 468: 468} with depth : 2
1001110000 624
0111010100 468
-----Exploring bucket: {789: 789, 953: 953, 141: 141, 805: 805} with depth : 2
1100010101 789
1110111001 953
0010001101 141
1100100101 805
-----Exploring bucket: {654: 654, 642: 642, 126: 126, 270: 270, 586: 586} with
↪  depth : 2
1010001110 654
1010000010 642
0001111110 126
0100001110 270
1001001010 586
-----Exploring bucket: {583: 583, 55: 55, 247: 247} with depth : 4
1001000111 583
0000110111 55
0011110111 247
-----Exploring bucket: {624: 624, 468: 468} with depth : 2
1001110000 624
0111010100 468
-----Exploring bucket: {789: 789, 953: 953, 141: 141, 805: 805} with depth : 2
1100010101 789
1110111001 953
0010001101 141
1100100101 805
-----Exploring bucket: {654: 654, 642: 642, 126: 126, 270: 270, 586: 586} with
↪  depth : 2
1010001110 654
1010000010 642
0001111110 126
0100001110 270
1001001010 586
-----Exploring bucket: {107: 107, 659: 659, 123: 123} with depth : 3
0001101011 107
1010010011 659
```

```
0001111011 123
-----Exploring bucket: {624: 624, 468: 468} with depth : 2
1001110000 624
0111010100 468
-----Exploring bucket: {789: 789, 953: 953, 141: 141, 805: 805} with depth : 2
1100010101 789
1110111001 953
0010001101 141
1100100101 805
-----Exploring bucket: {654: 654, 642: 642, 126: 126, 270: 270, 586: 586} with
↪  depth : 2
1010001110 654
1010000010 642
0001111110 126
0100001110 270
1001001010 586
-----Exploring bucket: {719: 719, 943: 943, 575: 575} with depth : 4
1011001111 719
1110101111 943
1000111111 575
```

IV. Summary

✓ If directory fits in memory then point query requires only 1 disk access

✓ Empty buckets can be merge with it's split image when directory becomes half of size

✗ need to maintain local and global depth

✗ Even if directory size doubles only on demand/overflow in worst case it may be possible that directory is getting overflow with similar matching bits in which the data stored in it are utilized about fifty percent only (even if buckets are less multiple pointers are pointing to same bucket).

Time Complexity of Extendible Hashing for single record access

	Condition	
	directory size < memory size	**directory size > memory size**
Access	1	2

Space Complexity

R	Number of records
B	Block Size
N	Number of blocks

Space Utilization

$$\frac{R}{B \times N} \tag{1}$$

Average Utilization $\ln 2 \approx 0.69$