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:  [786, 154, 678, 873, 786]
----------------------------------------
global bucket >  {786: 786, 154: 154, 678: 678, 873: 873}
global depth >  0
-----Exploring bucket: {786: 786, 154: 154, 678: 678, 873: 873} with depth : 0
1100010010 786
0010011010 154
1010100110 678
1101101001 873
```

Output 2

```
Bucket Size:  5
Total Inputs :  10
Input Sequence:  [956, 469, 265, 417, 494, 117, 633, 1000, 753, 508]
----------------------------------------
global bucket >  {956: 956, 494: 494, 1000: 1000, 508: 508}, {469: 469, 265: 265,
 ↪  417: 417, 117: 117, 633: 633, 753: 753}, {956: 956, 494: 494, 1000: 1000, 508:
 ↪  508}
global depth >  2
-----Exploring bucket: {956: 956, 494: 494, 1000: 1000, 508: 508} with depth : 1
1110111100 956
0111101110 494
1111101000 1000
0111111100 508
```

```
-----Exploring bucket: {469: 469, 265: 265, 417: 417, 117: 117, 633: 633, 753:
 ↪  753} with depth : 2
0111010101 469
0100001001 265
0110100001 417
0001110101 117
1001111001 633
1011110001 753
-----Exploring bucket: {956: 956, 494: 494, 1000: 1000, 508: 508} with depth : 1
1110111100 956
0111101110 494
1111101000 1000
0111111100 508
```

OUTPUT 3

```
Bucket Size:  5
Total Inputs :  25
Input Sequence:  [818, 413, 1000, 348, 752, 429, 549, 783, 886, 580, 515, 883, 24,
 ↪  252, 309, 459, 352, 493, 382, 915, 222, 80, 711, 190, 18]
---------------------------------------
global bucket >  {1000: 1000, 752: 752, 24: 24, 352: 352, 80: 80}, {413: 413, 429:
 ↪  429, 549: 549, 309: 309, 493: 493}, {818: 818, 18: 18}, {515: 515, 883: 883,
 ↪  459: 459, 915: 915}, {348: 348, 580: 580, 252: 252}, {413: 413, 429: 429, 549:
 ↪  549, 309: 309, 493: 493}, {886: 886, 382: 382, 222: 222, 190: 190}, {783: 783,
 ↪  711: 711}
global depth >  3
-----Exploring bucket: {1000: 1000, 752: 752, 24: 24, 352: 352, 80: 80} with
 ↪  depth : 3
1111101000 1000
1011110000 752
0000011000 24
0101100000 352
0001010000 80
-----Exploring bucket: {413: 413, 429: 429, 549: 549, 309: 309, 493: 493} with
 ↪  depth : 2
0110011101 413
0110101101 429
1000100101 549
0100110101 309
0111101101 493
-----Exploring bucket: {818: 818, 18: 18} with depth : 3
1100110010 818
0000010010 18
-----Exploring bucket: {515: 515, 883: 883, 459: 459, 915: 915} with depth : 3
1000000011 515
1101110011 883
```

```
0111001011 459
1110010011 915
-----Exploring bucket: {348: 348, 580: 580, 252: 252} with depth : 3
0101011100 348
1001000100 580
0011111100 252
-----Exploring bucket: {413: 413, 429: 429, 549: 549, 309: 309, 493: 493} with
 ↪  depth : 2
0110011101 413
0110101101 429
1000100101 549
0100110101 309
0111101101 493
-----Exploring bucket: {886: 886, 382: 382, 222: 222, 190: 190} with depth : 3
1101110110 886
0101111110 382
0011011110 222
0010111110 190
-----Exploring bucket: {783: 783, 711: 711} with depth : 3
1100001111 783
1011000111 711
```

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 as other spaces are empty we can conclude that it may cause wastage of space in worst case.

Table 1: 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 $\frac{R}{B*N}$

Average Utilization $\ln 2 \approx 0.69$