# 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)
            bucket2 = Bucket(self.bucket_size)
            for k, v in bucket.data.items():
                # print_log("key", k, "value", v)
                if ((k & ((1 << self.total_global_data) - 1)) >>
                 ↪  bucket.total_data) & 1 == 1:
                    bucket2.add(k, v)
                else:
                    bucket1.add(k, v)
            for idx, value in enumerate(self.buckets):
                # print_log("idx", idx)
                if value == bucket:
                    if (idx >> bucket.total_data) & 1 == 1:
                        self.buckets[idx] = bucket2
                    else:
                        self.buckets[idx] = bucket1
            bucket2.total_data = bucket1.total_data = bucket.total_data + 1
        else:
            bucket.add(key, val)
        # print("bucket after adding", val, "operation: ", bucket.data)

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

    def __repr__(self):
        return ", ".join("{}".format(b) for b in self.buckets if b.__repr__())


def test(input_nums=10):
    """
    test function to test the implementation
    :param input_nums: number of inputs to be added
    :return:
    """
    BUCKET_SIZE = 5  # defining single bucket size
    g = GlobalBucket(BUCKET_SIZE)
    inputs = [random.randint(1, 1000) for i in range(input_nums)]
    print("Bucket Size: ", BUCKET_SIZE)
    print("Total Inputs : ", input_nums)
    print("Input Sequence: ", inputs)
    for i in inputs:
        print_log("*>Adding {} in to bucket".format(i))
        g.add(i, i)
    print("-"*40)
    print("global bucket > ", g)
    print("global depth > ", g.total_global_data)
    for _bucket in g.buckets:
        if _bucket.__repr__():
            print("-----Exploring bucket: {} with depth : {}".format(_bucket,
             ↪  _bucket.total_data))
            _bucket.show()


if __name__ == "__main__":
    TEST_NUM = 25
    import sys
    if len(sys.argv) > 1:
        try:
            TEST_NUM = int(sys.argv[1])
        except:
            pass
    test(TEST_NUM)
```

## Output 1

```
Bucket Size:  5
Total Inputs :  25
Input Sequence:  [942, 135, 98, 596, 18, 99, 680, 896, 106, 646, 661, 374, 751,
 ↪  533, 578, 446, 195, 532, 531, 620, 759, 566, 764, 34, 71]
----------------------------------------
global bucket >  {680: 680, 896: 896}, {661: 661, 533: 533}, {98: 98, 18: 18, 106:
 ↪  106, 578: 578, 34: 34}, {99: 99, 195: 195, 531: 531}, {596: 596, 532: 532,
 ↪  620: 620, 764: 764}, {661: 661, 533: 533}, {942: 942, 646: 646, 374: 374, 446:
 ↪  446, 566: 566}, {135: 135, 751: 751, 759: 759, 71: 71}
global depth >  3
-----Exploring bucket: {680: 680, 896: 896} with depth : 3
1010101000 680
1110000000 896
-----Exploring bucket: {661: 661, 533: 533} with depth : 2
1010010101 661
1000010101 533
-----Exploring bucket: {98: 98, 18: 18, 106: 106, 578: 578, 34: 34} with depth :
 ↪  3
0001100010 98
0000010010 18
0001101010 106
1001000010 578
0000100010 34
-----Exploring bucket: {99: 99, 195: 195, 531: 531} with depth : 3
0001100011 99
0011000011 195
1000010011 531
-----Exploring bucket: {596: 596, 532: 532, 620: 620, 764: 764} with depth : 3
1001010100 596
1000010100 532
```

```
1001101100 620
1011111100 764
-----Exploring bucket: {661: 661, 533: 533} with depth : 2
1010010101 661
1000010101 533
-----Exploring bucket: {942: 942, 646: 646, 374: 374, 446: 446, 566: 566} with
 ↪  depth : 3
1110101110 942
1010000110 646
0101110110 374
0110111110 446
1000110110 566
-----Exploring bucket: {135: 135, 751: 751, 759: 759, 71: 71} with depth : 3
0010000111 135
1011101111 751
1011110111 759
0001000111 71
```

## Output 2

```
Bucket Size:  5
Total Inputs :  25
Input Sequence:  [95, 167, 280, 598, 303, 859, 14, 542, 3, 897, 174, 662, 379,
 ↪  504, 915, 148, 369, 824, 766, 627, 656, 445, 370, 202, 808]
---------------------------------------
global bucket >  {280: 280, 504: 504, 824: 824, 656: 656, 808: 808}, {897: 897,
 ↪  369: 369, 445: 445}, {370: 370, 202: 202}, {859: 859, 3: 3, 379: 379, 915:
 ↪  915, 627: 627}, {148: 148}, {897: 897, 369: 369, 445: 445}, {598: 598, 14: 14,
 ↪  542: 542, 174: 174, 662: 662, 766: 766}, {95: 95, 167: 167, 303: 303}
global depth >  3
-----Exploring bucket: {280: 280, 504: 504, 824: 824, 656: 656, 808: 808} with
 ↪  depth : 3
0100011000 280
0111111000 504
1100111000 824
1010010000 656
1100101000 808
-----Exploring bucket: {897: 897, 369: 369, 445: 445} with depth : 2
1110000001 897
0101110001 369
0110111101 445
-----Exploring bucket: {370: 370, 202: 202} with depth : 3
101110010 370
011001010 202
-----Exploring bucket: {859: 859, 3: 3, 379: 379, 915: 915, 627: 627} with depth :
 ↪  3
1101011011 859
```

```
0000000011 3
0101111011 379
1110010011 915
1001110011 627
-----Exploring bucket: {148: 148} with depth : 3
10010100 148
-----Exploring bucket: {897: 897, 369: 369, 445: 445} with depth : 2
1110000001 897
0101110001 369
0110111101 445
-----Exploring bucket: {598: 598, 14: 14, 542: 542, 174: 174, 662: 662, 766: 766}
 ↪  with depth : 3
1001010110 598
0000001110 14
1000011110 542
0010101110 174
1010010110 662
1011111110 766
-----Exploring bucket: {95: 95, 167: 167, 303: 303} with depth : 3
001011111 95
010100111 167
100101111 303
```

## Output 3

```
Bucket Size:  5
Total Inputs :  25
Input Sequence:  [168, 92, 572, 74, 1000, 756, 337, 468, 37, 772, 110, 736, 475,
 ↪  194, 951, 51, 714, 57, 182, 410, 218, 867, 419, 160, 440]
---------------------------------------
global bucket >  {168: 168, 1000: 1000, 736: 736, 160: 160, 440: 440}, {337: 337,
 ↪  37: 37, 57: 57}, {74: 74, 194: 194, 714: 714, 410: 410, 218: 218}, {475: 475,
 ↪  951: 951, 51: 51, 867: 867, 419: 419}, {92: 92, 572: 572, 756: 756, 468: 468,
 ↪  772: 772}, {337: 337, 37: 37, 57: 57}, {110: 110, 182: 182}, {475: 475, 951:
 ↪  951, 51: 51, 867: 867, 419: 419}
global depth >  3
-----Exploring bucket: {168: 168, 1000: 1000, 736: 736, 160: 160, 440: 440} with
 ↪  depth : 3
0010101000 168
1111101000 1000
1011100000 736
0010100000 160
0110111000 440
-----Exploring bucket: {337: 337, 37: 37, 57: 57} with depth : 2
101010001 337
000100101 37
000111001 57
```

```
-----Exploring bucket: {74: 74, 194: 194, 714: 714, 410: 410, 218: 218} with
 ↪  depth : 3
0001001010 74
0011000010 194
1011001010 714
0110011010 410
0011011010 218
-----Exploring bucket: {475: 475, 951: 951, 51: 51, 867: 867, 419: 419} with
 ↪  depth : 2
0111011011 475
1110110111 951
0000110011 51
1101100011 867
0110100011 419
-----Exploring bucket: {92: 92, 572: 572, 756: 756, 468: 468, 772: 772} with
 ↪  depth : 3
0001011100 92
1000111100 572
1011110100 756
0111010100 468
1100000100 772
-----Exploring bucket: {337: 337, 37: 37, 57: 57} with depth : 2
101010001 337
000100101 37
000111001 57
-----Exploring bucket: {110: 110, 182: 182} with depth : 3
01101110 110
10110110 182
-----Exploring bucket: {475: 475, 951: 951, 51: 51, 867: 867, 419: 419} with
 ↪  depth : 2
0111011011 475
1110110111 951
0000110011 51
1101100011 867
0110100011 419
```

## 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
- ✓ Hash Performance doesn't degrade with growth of file
- ✗ need to maintain local and global depth
- ✗ Extra level of indirection to find desired record

✗ 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** $\ln 2 \approx 0.69$