class HNode {
key
value

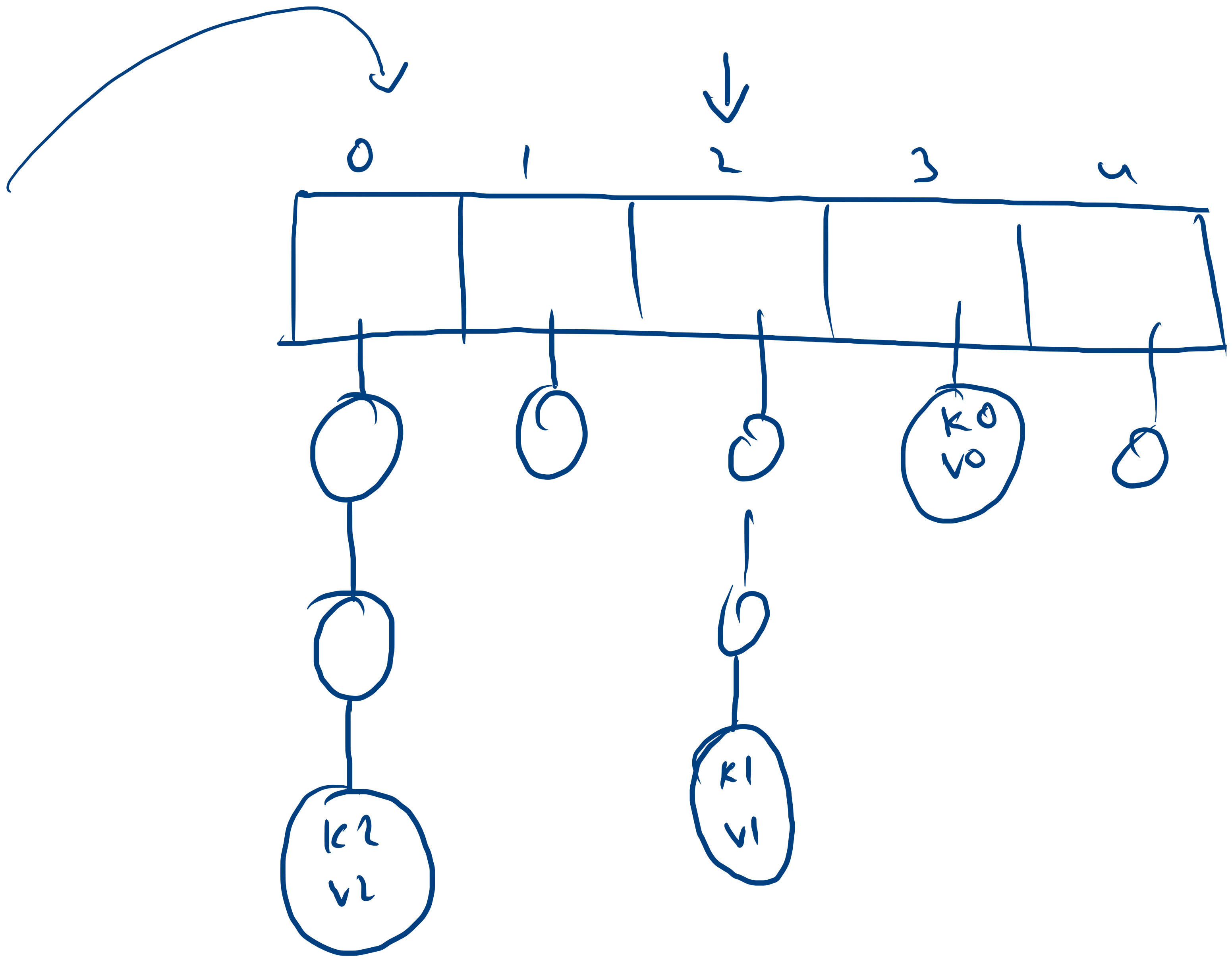
HashFunction(key)

put(key, value)
k1 v1

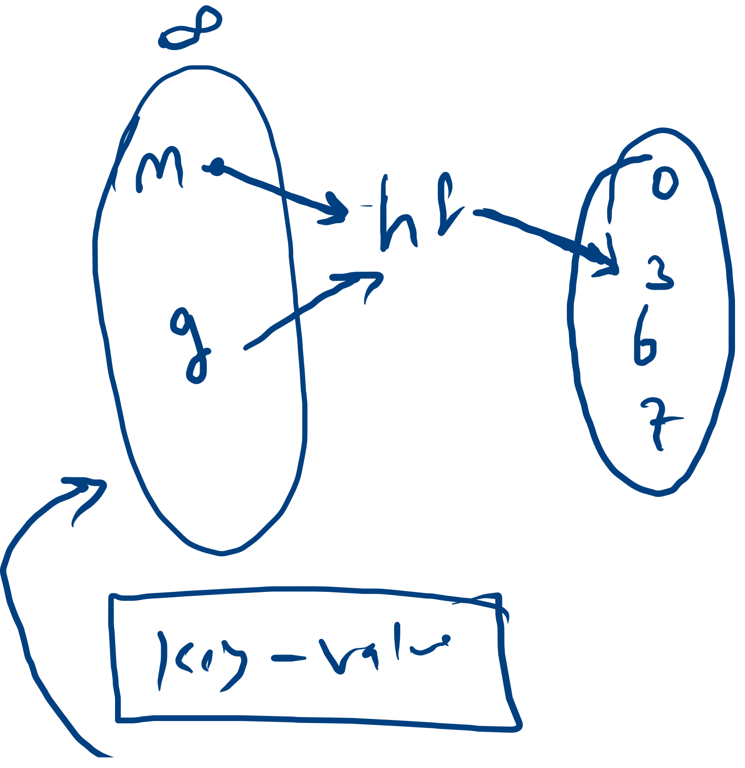
put(k2, v2)

contains(k3) → id → 3

remove



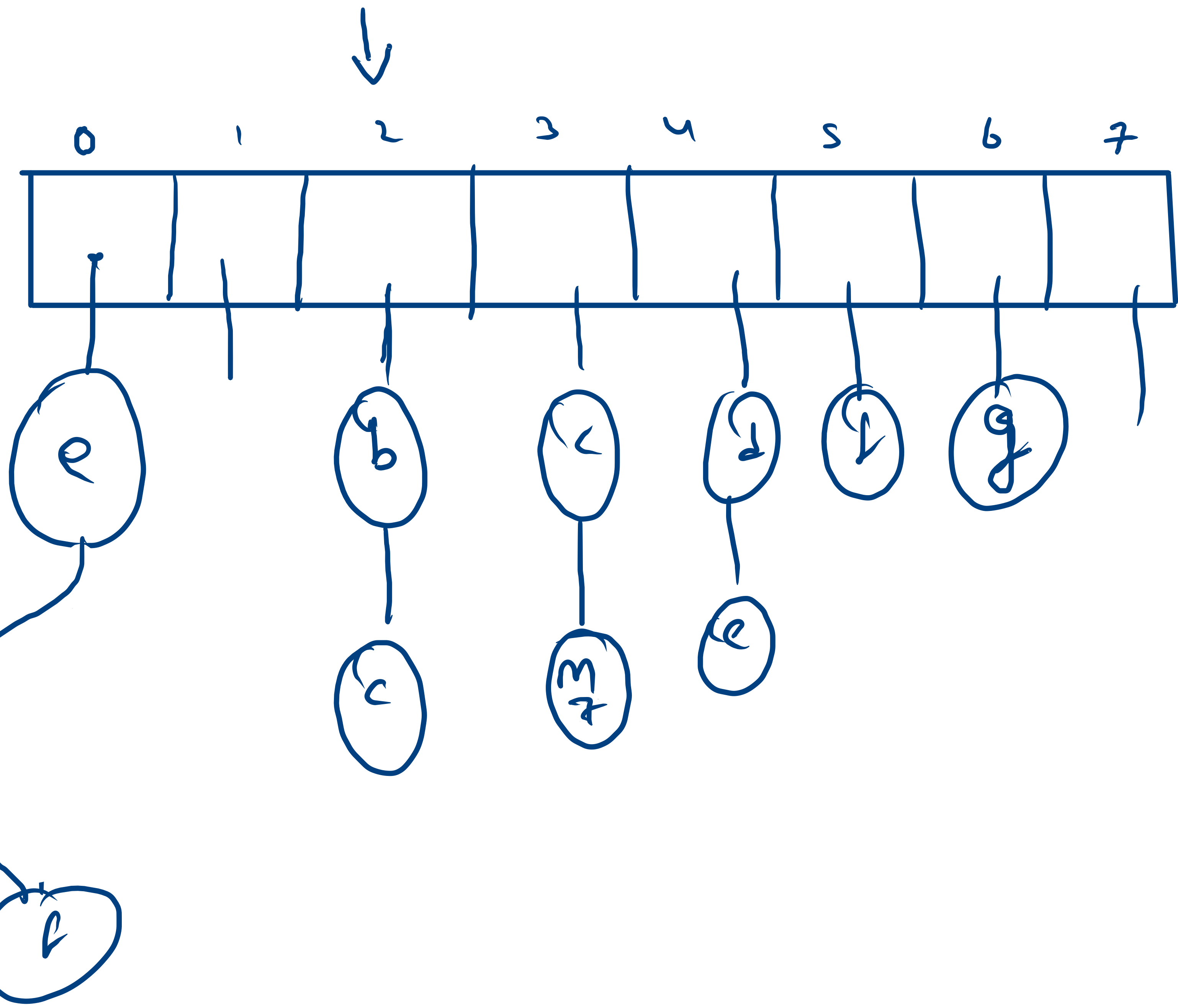
put(k, v)
contains(key)
remove(key)



add (a, -) a–0

remove (a)

remove (2).



$\lambda \to$ constant $\to 2$

$n \to$ size ~~8~~ ~~9~~ ~~10~~ ~~14~~ 15

$N \to 7$

$$\boxed{\frac{n}{N} \leq \lambda}$$

Rehashing

$$\boxed{\frac{15}{7} \leq 2} \quad \alpha$$

add(h, v)

add (-

get (g)

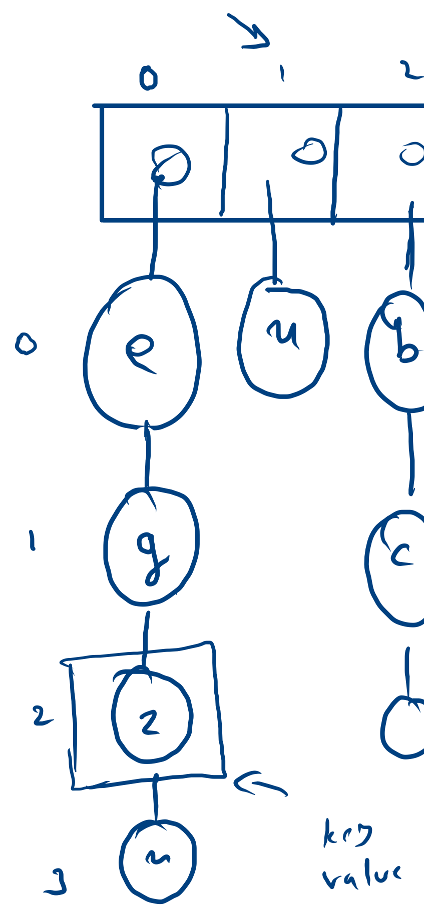
add (x,)

$g \to 1$

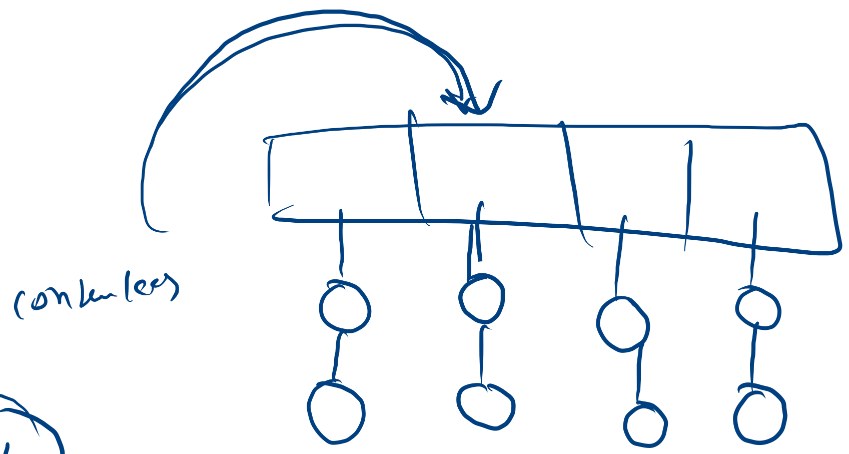
$e \to 0$

$y \to -1$

get (z)



hashcode $\to$ % N $\to$ id



LL

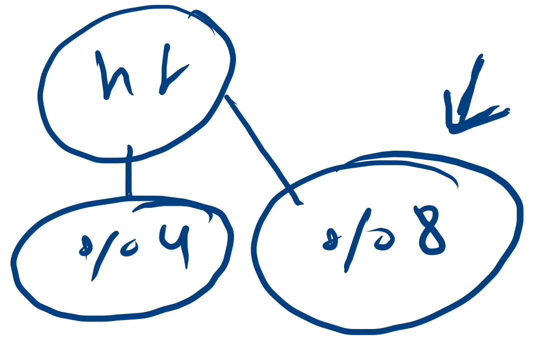
$O(n)$

λ^{c}

$\Theta(\lambda) \to \Theta(1)$

average

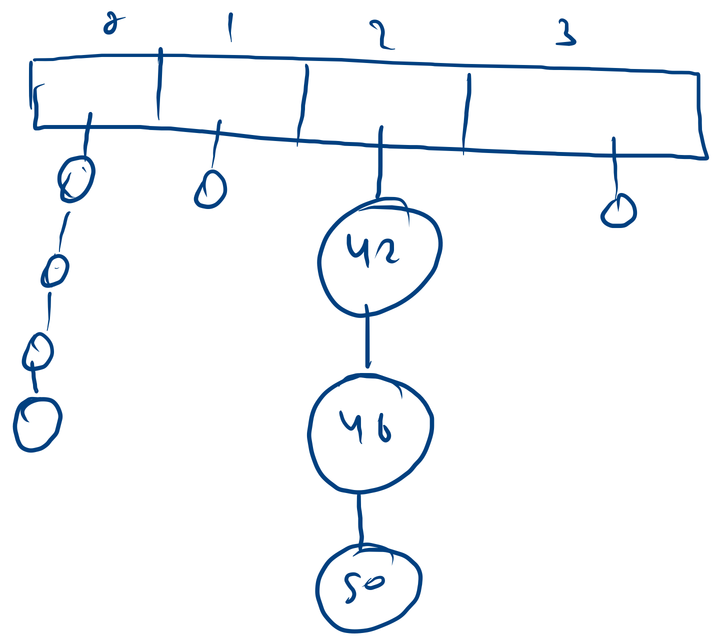
key



$42 — 2$

$46 — 2$

50



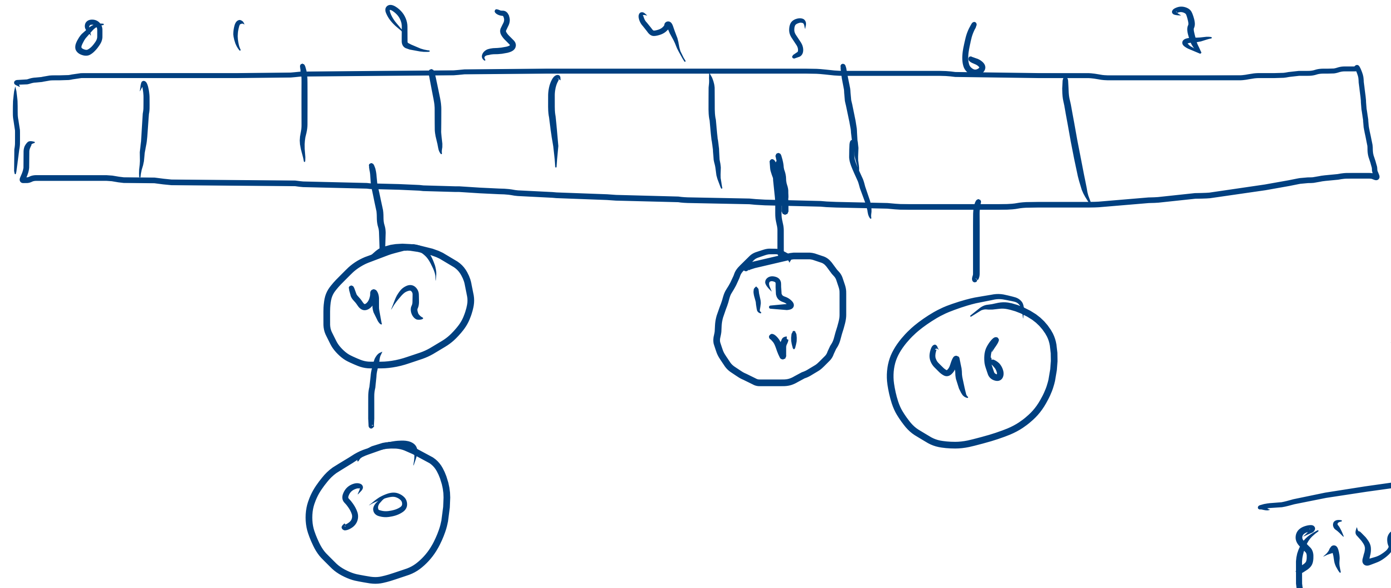
$\boxed{\lambda \to 2}$

$n \to 3$

$N \to 4$

✓ $\boxed{\frac{n}{N} \leq \lambda}$

$\frac{9}{4} \leq 2$



$42 \% 8 = 2$

$46 \% 8 = \boxed{6}$

$50 \% 8 = 2$

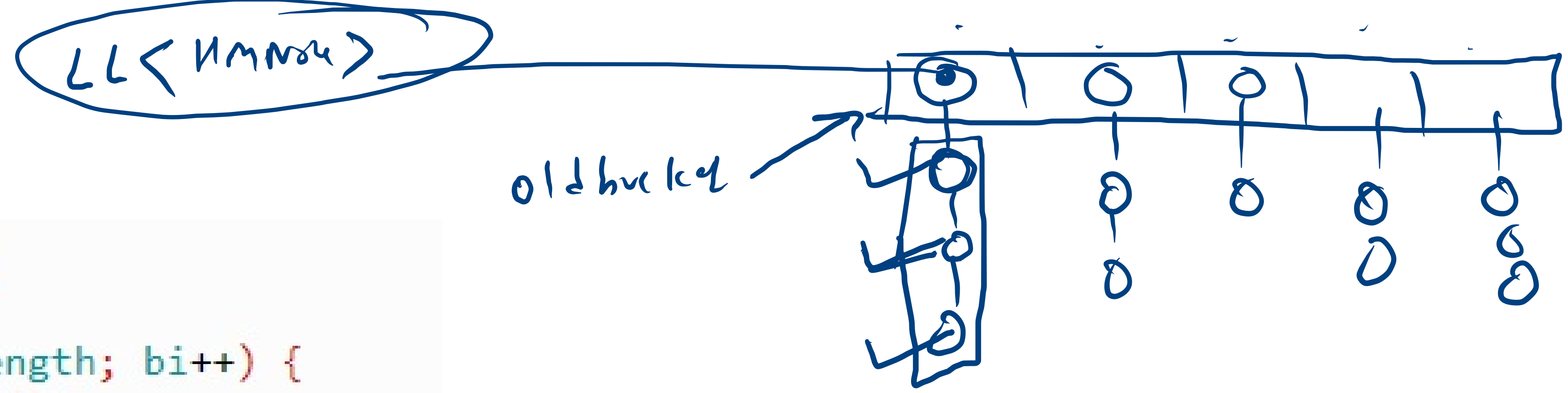
$13 \% 8 \to 5$

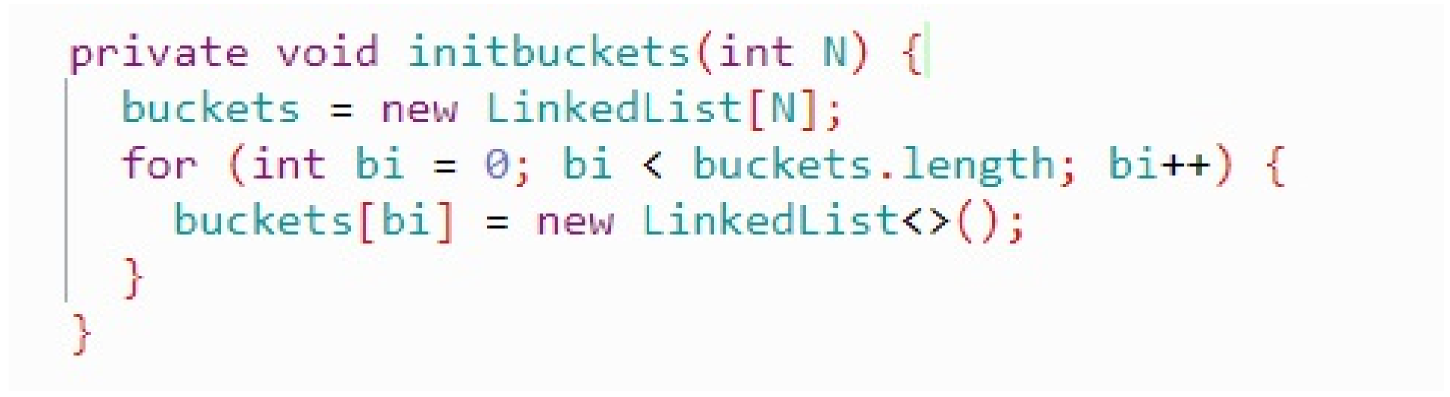
$\frac{\text{size}}{\text{bucket}} > 2$

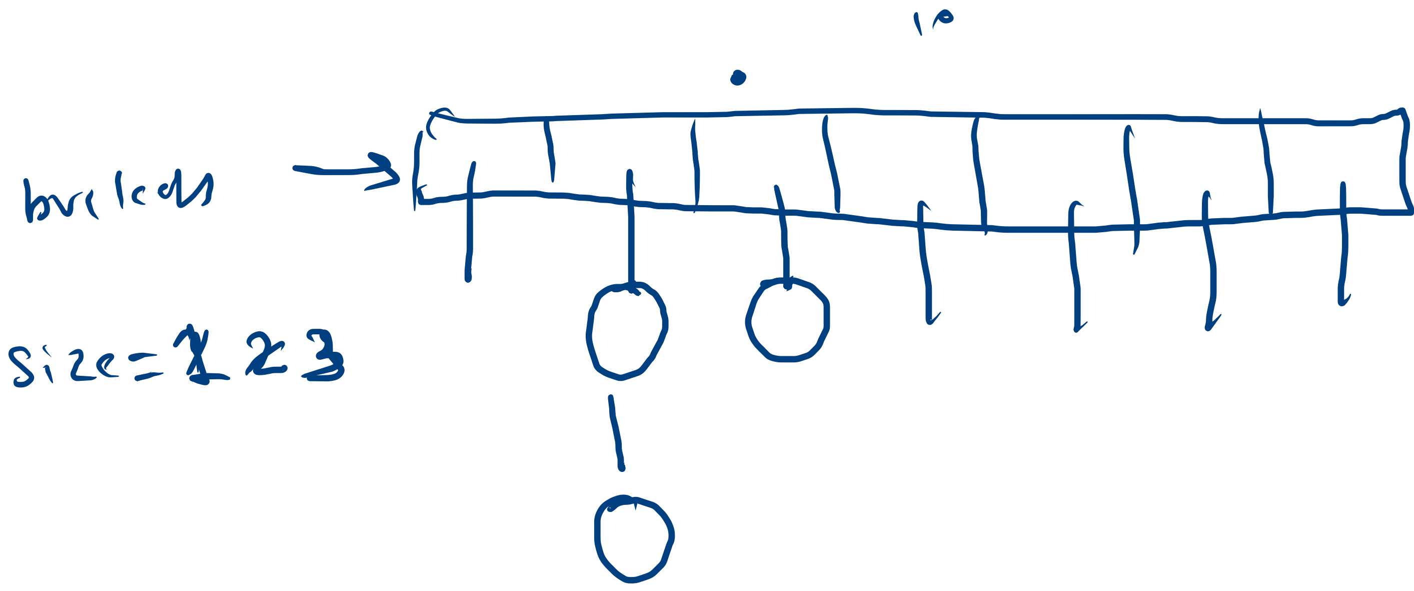
$\boxed{\text{size} > \text{bucket} * 2}$

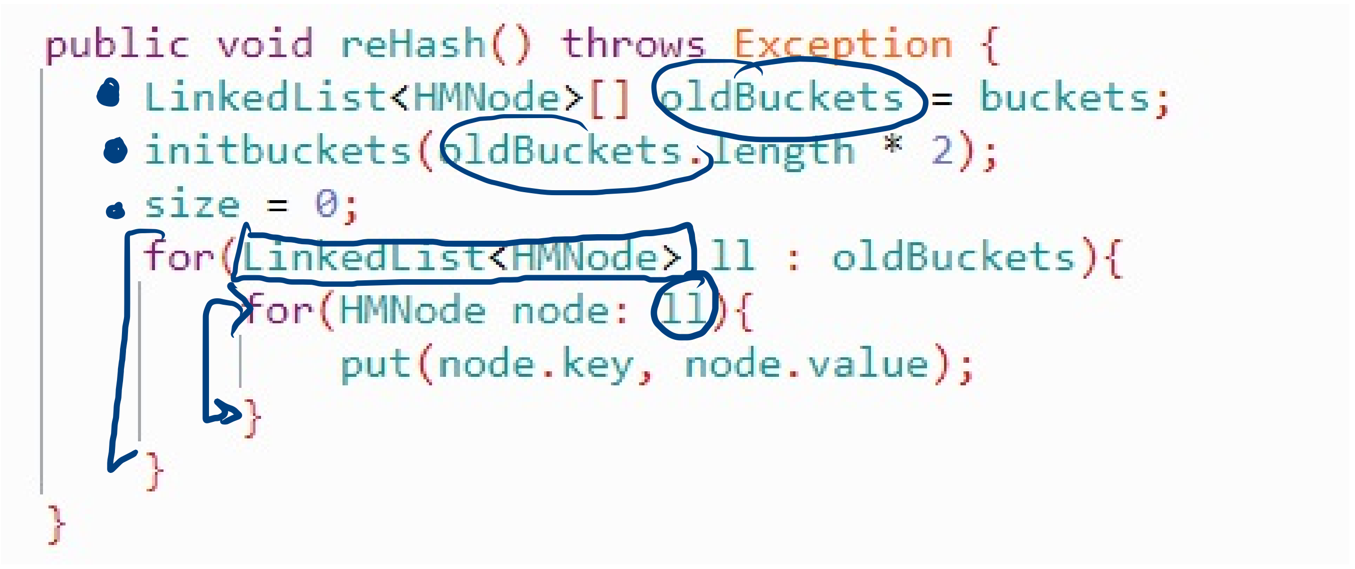
$\boxed{\frac{9}{4} > 2}$

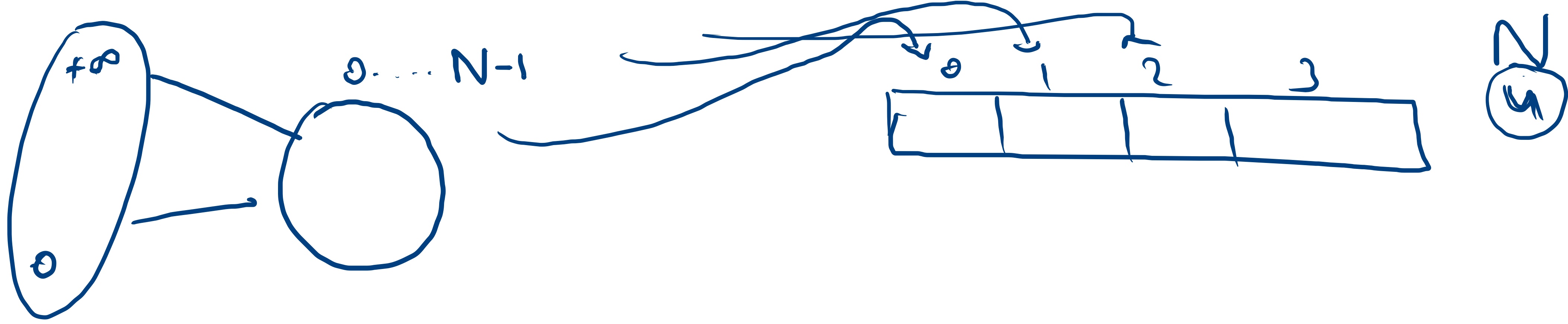
rehashing

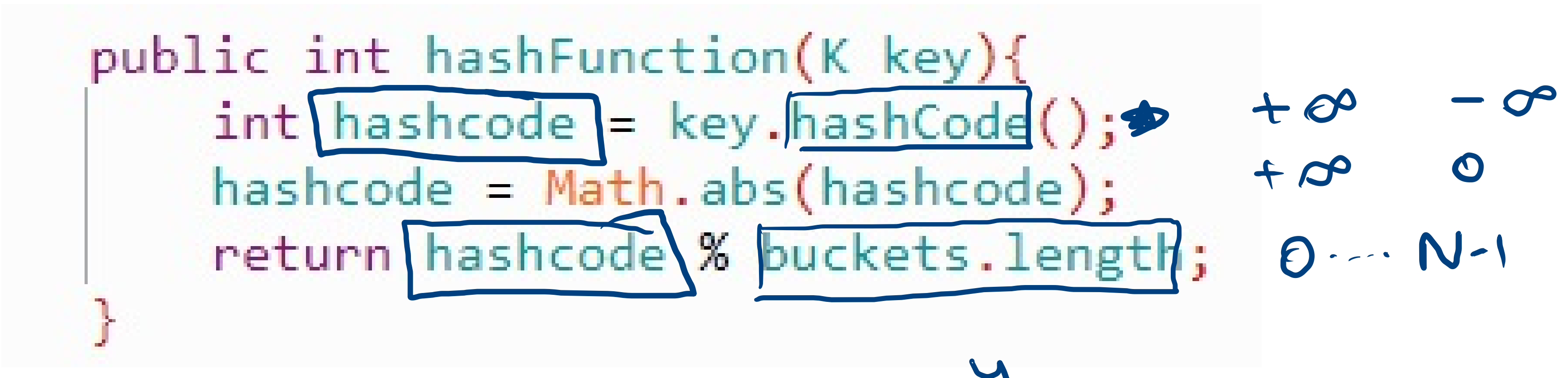



```
private void initbuckets(int N) {
  buckets = new LinkedList[N];
  for (int bi = 0; bi < buckets.length; bi++) {
    buckets[bi] = new LinkedList<>();
  }
}
```




```
public void reHash() throws Exception {
  LinkedList<HMNode>[] oldBuckets = buckets;
  initbuckets(oldBuckets.length * 2);
  size = 0;
  for(LinkedList<HMNode> ll : oldBuckets){
    for(HMNode node: ll){
      put(node.key, node.value);
    }
  }
}
```



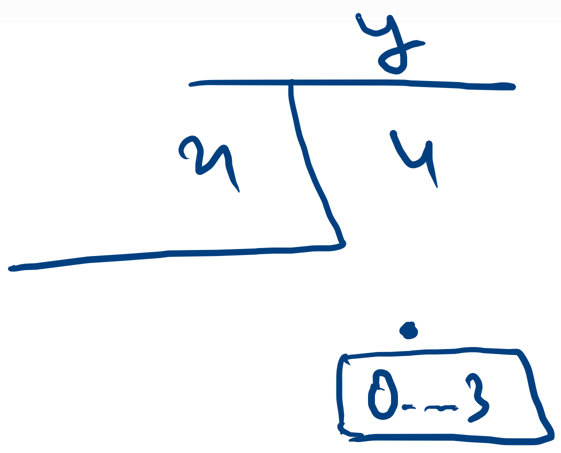


```
public int hashFunction(K key){
    int hashcode = key.hashCode();
    hashcode = Math.abs(hashcode);
    return hashcode % buckets.length;
}
```

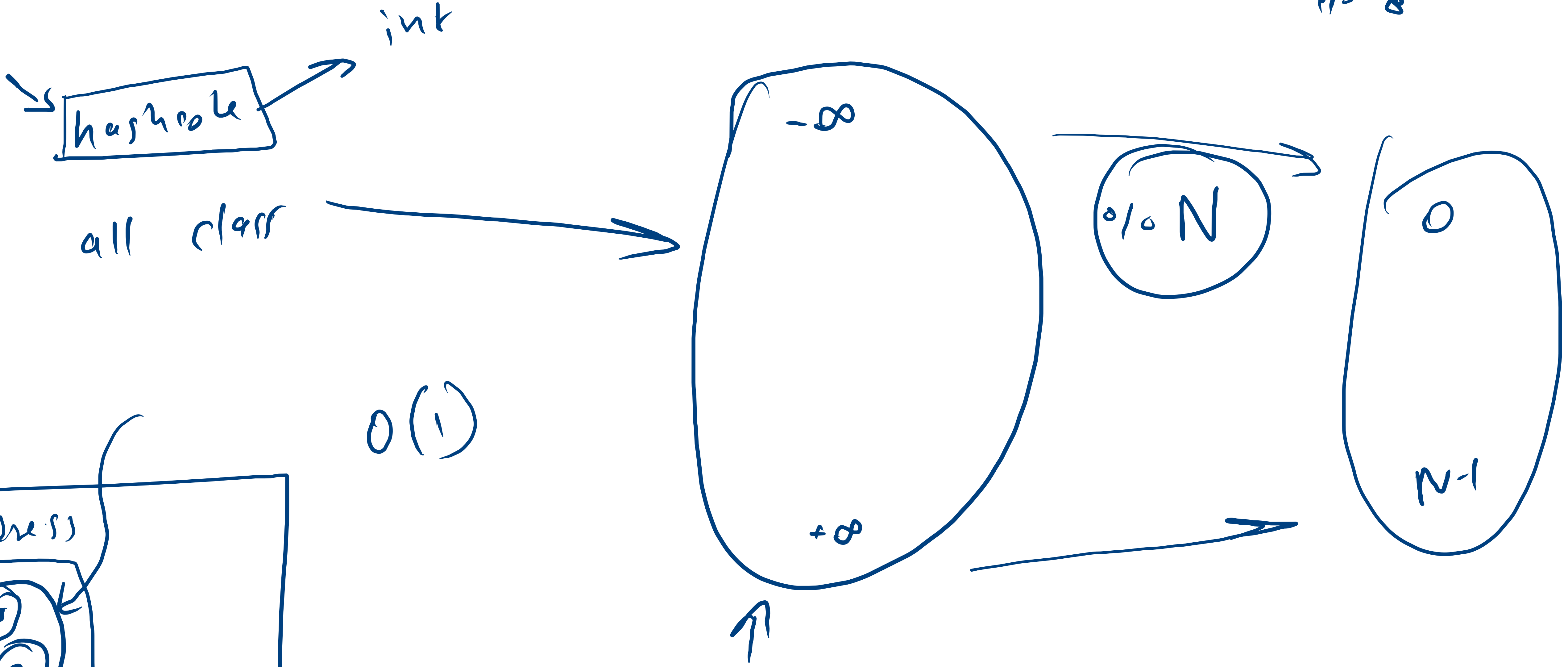
+∞ −∞

+∞ 0

0 .... N-1

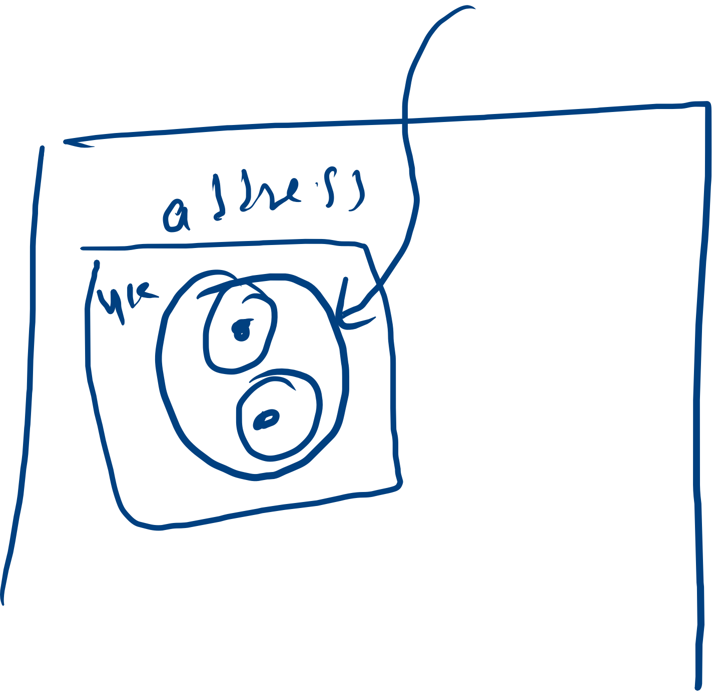


%10 → 0 1 2
%8 → 0 7

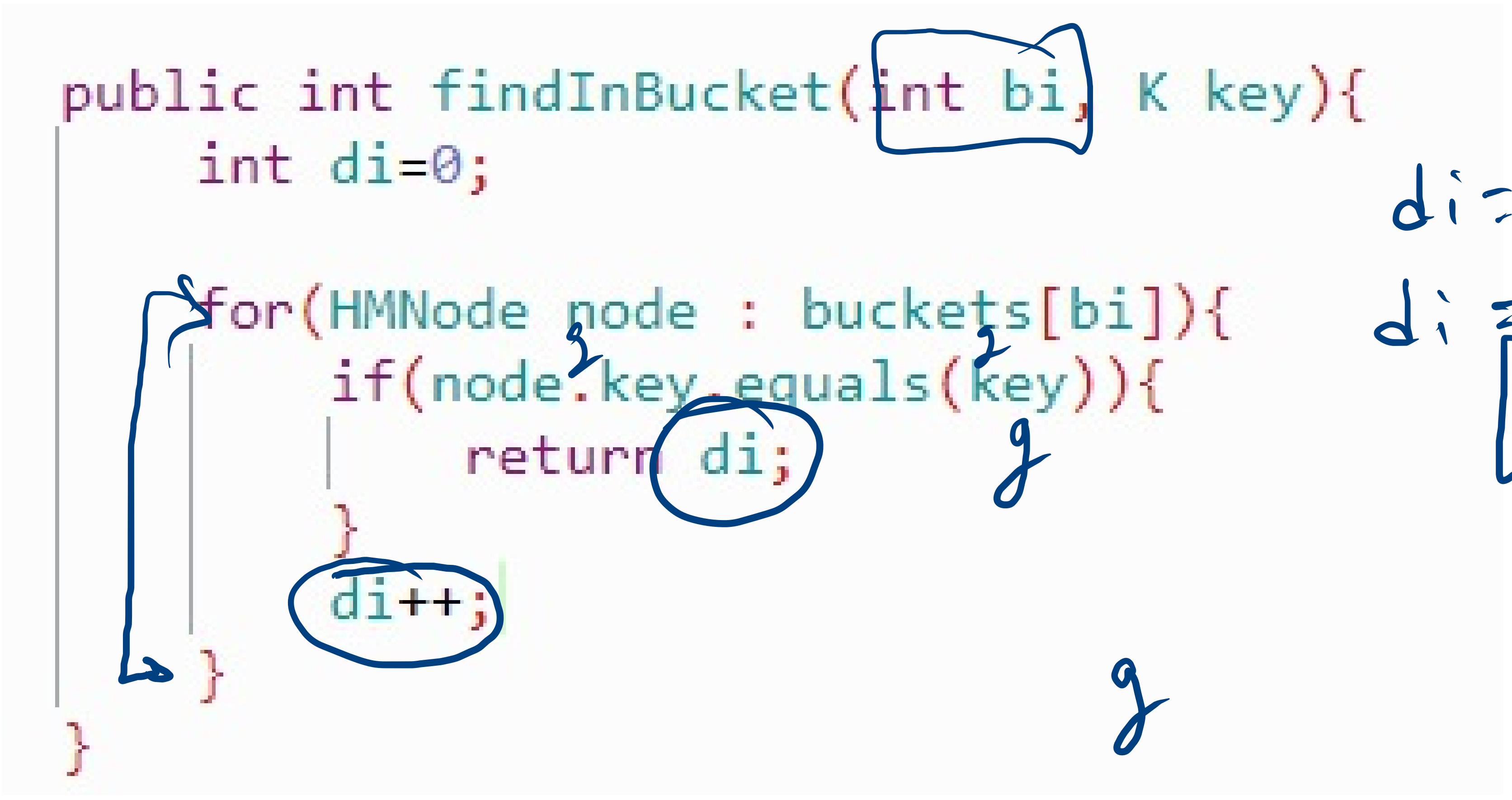


O

O(1)



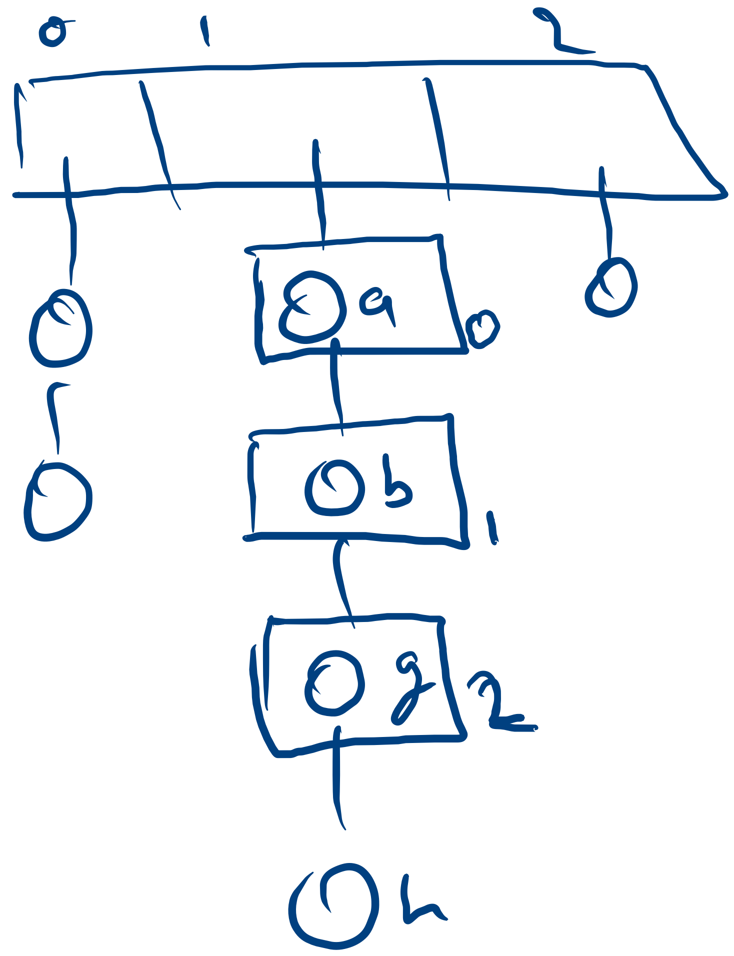
bi -> 1



```java
public int findInBucket(int bi, K key){
    int di=0;

    for(HMNode node : buckets[bi]){
        if(node.key.equals(key)){
            return di;
        }
        di++;
    }
}
```

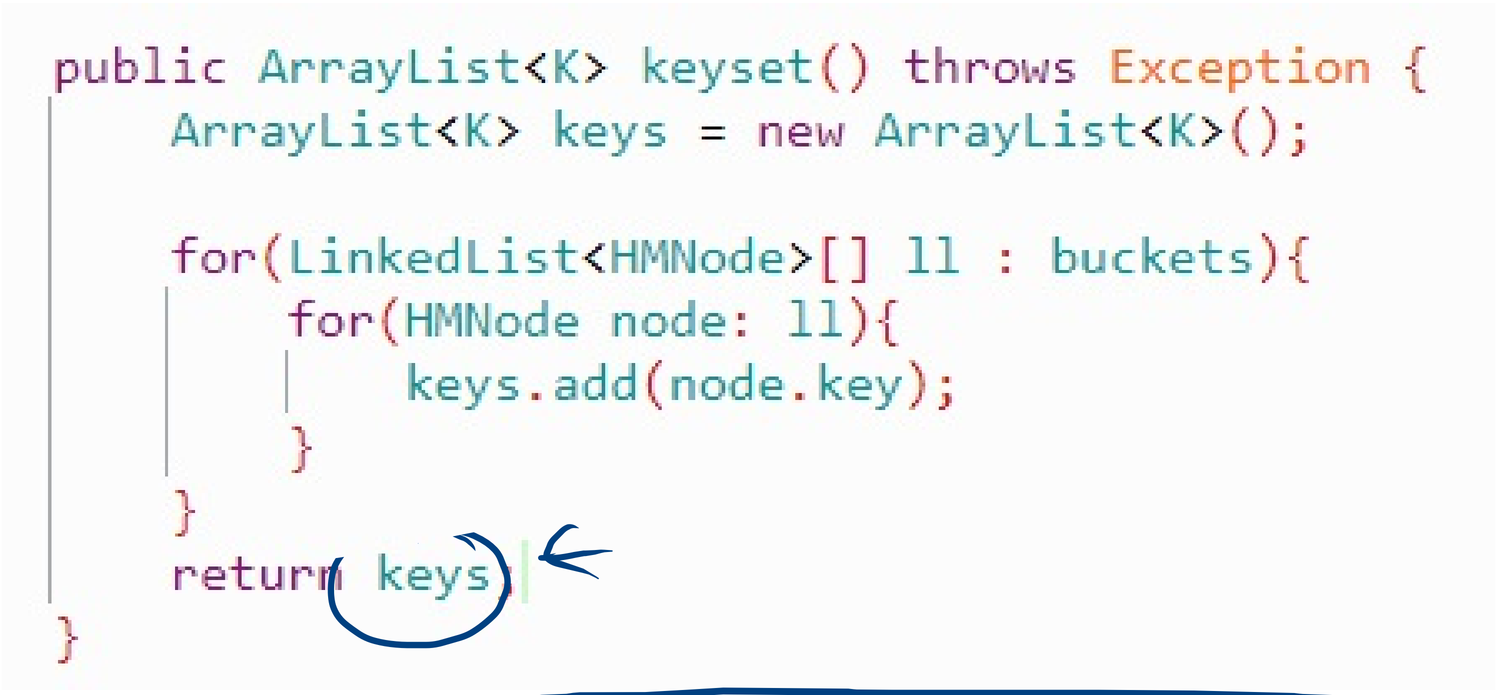
di=0

di = ~~1~~ $\boxed{2}$



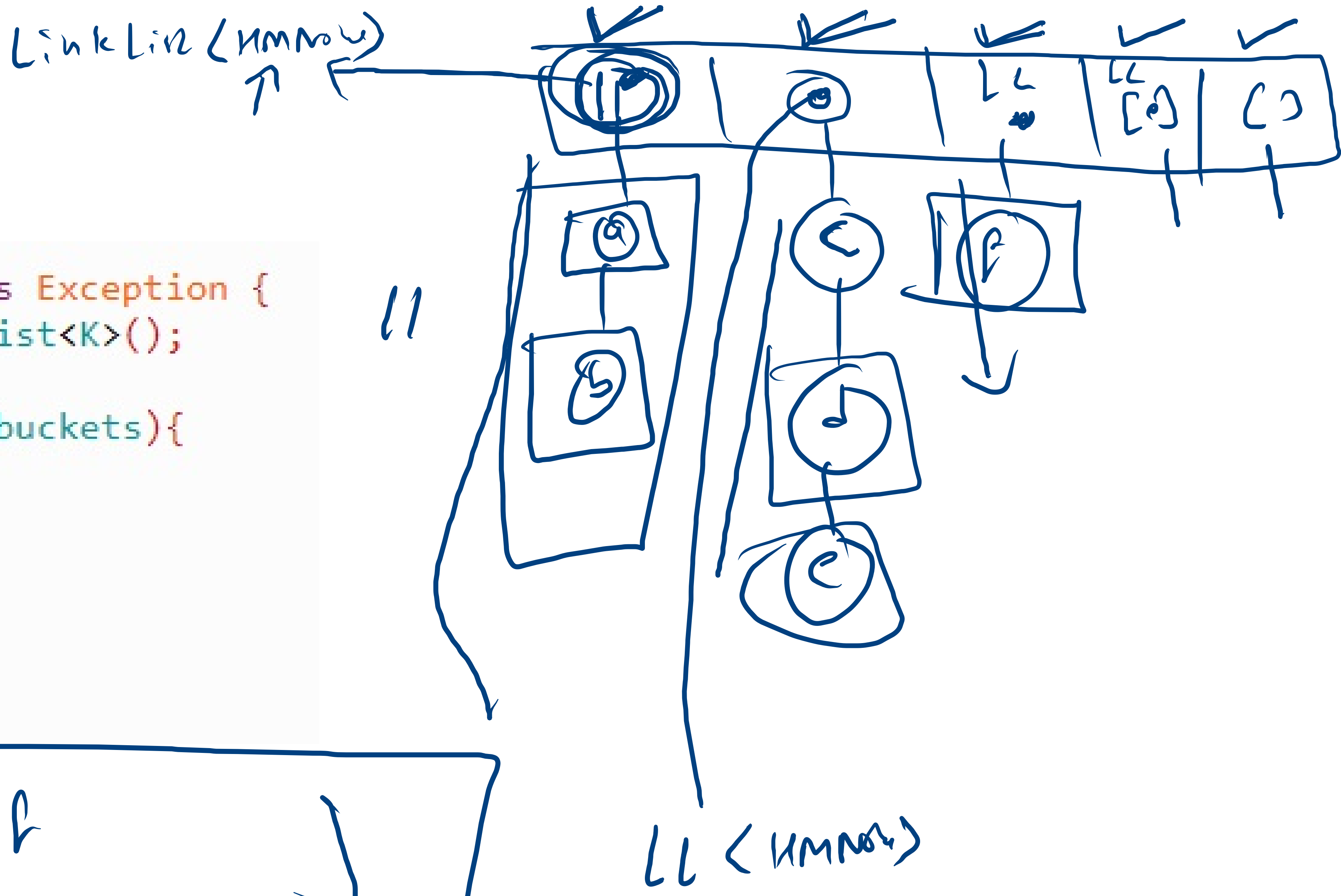
arr int [1 2 3 4]

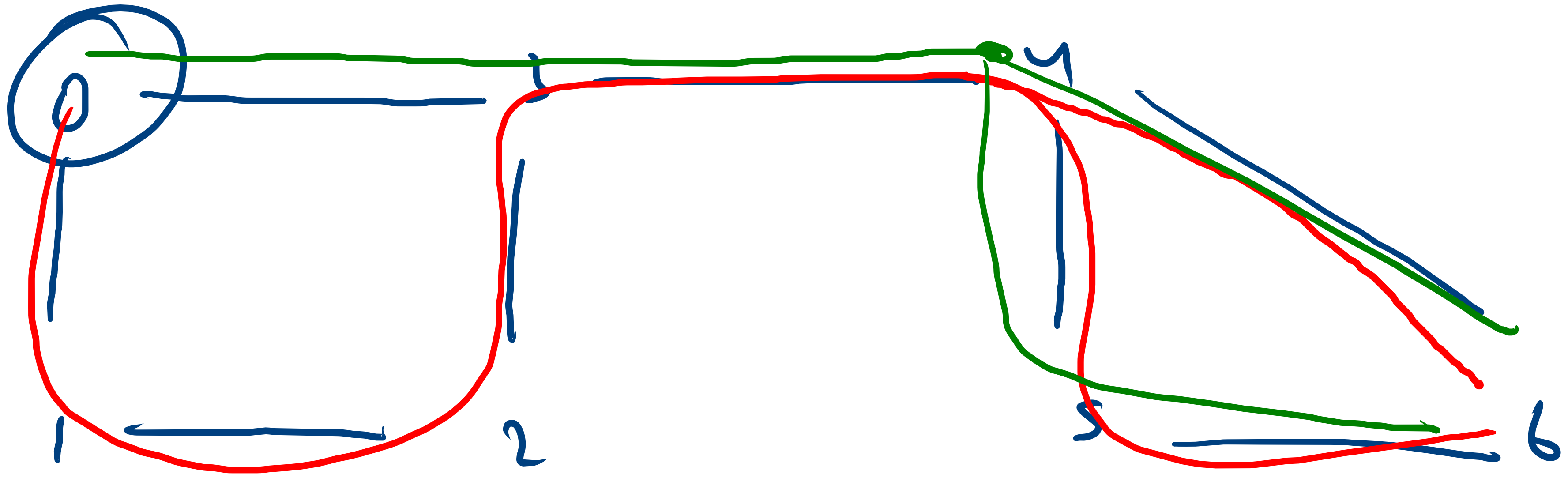
fo(int b : arr){
string



```
public ArrayList<K> keyset() throws Exception {
    ArrayList<K> keys = new ArrayList<K>();

    for(LinkedList<HMNode>[] ll : buckets){
        for(HMNode node: ll){
            keys.add(node.key);
        }
    }
    return keys;
}
```

[a b c d e f]





src → 0

dest → 6

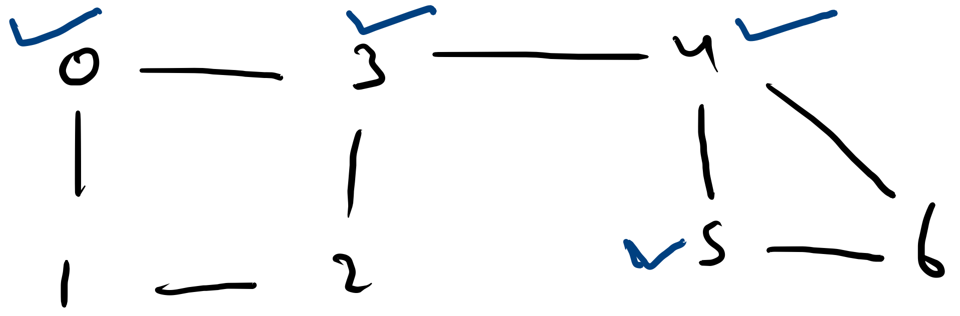
0 1 2 3 4 5 6

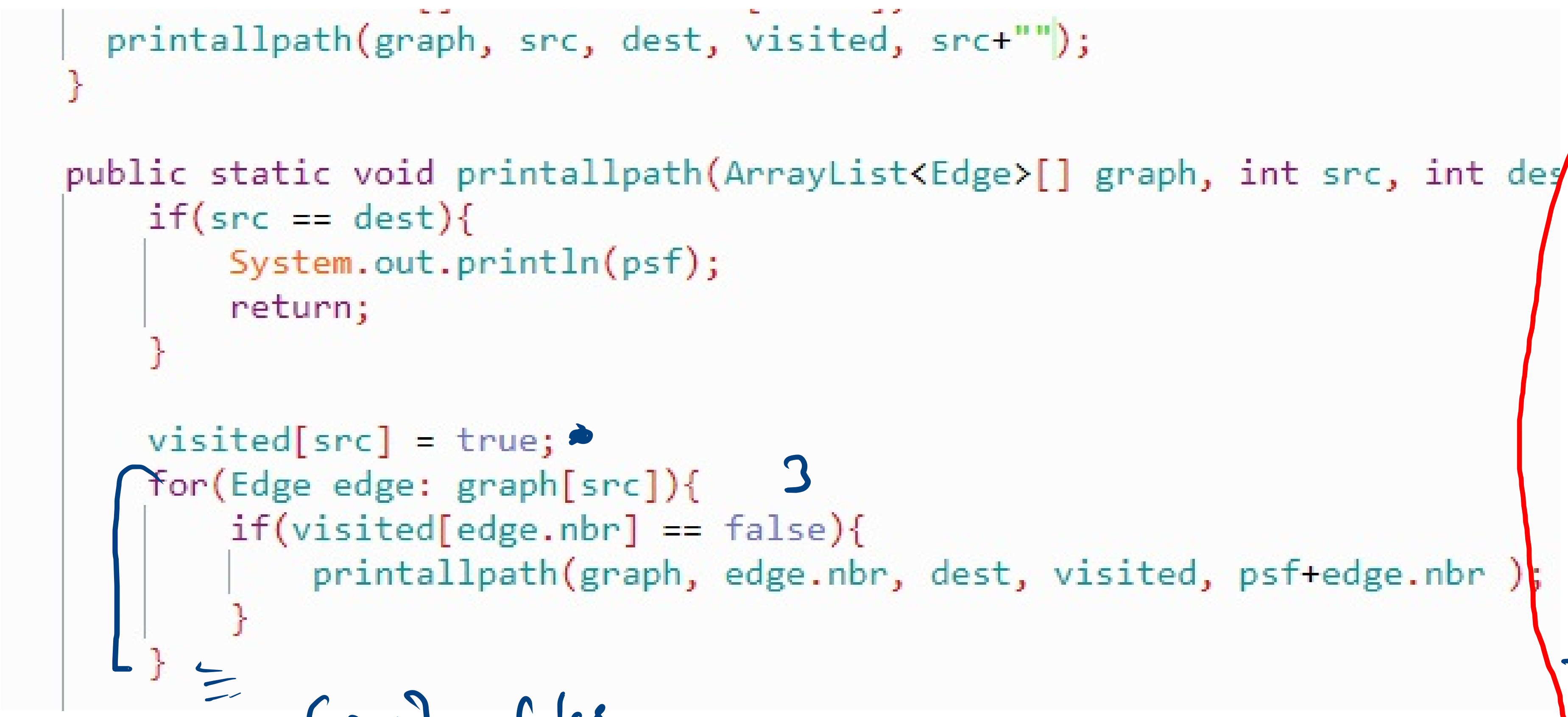
0 1 2 3 4 6

0 3 4 5 6

0 3 4 6

0123456
012346
03456

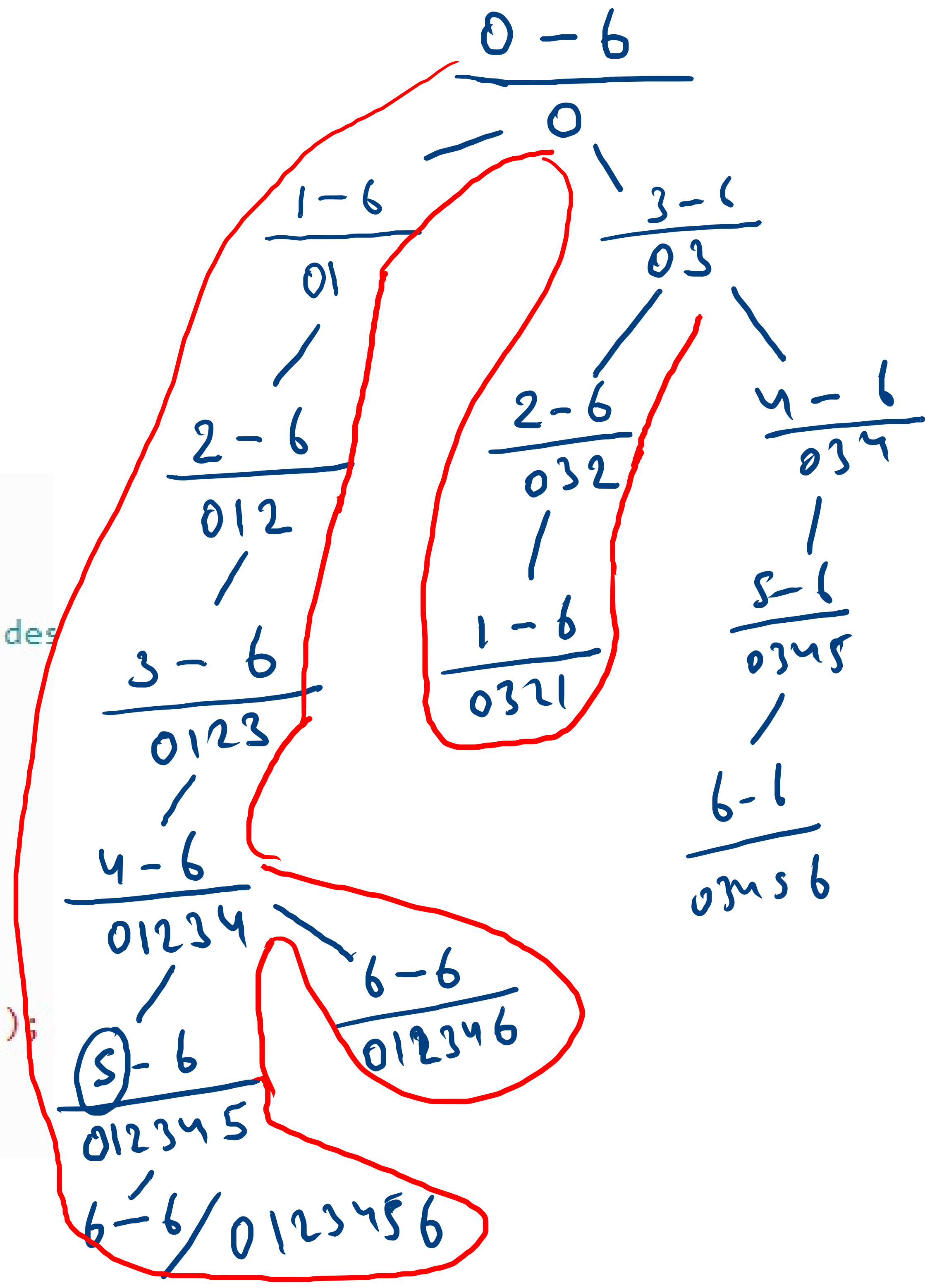


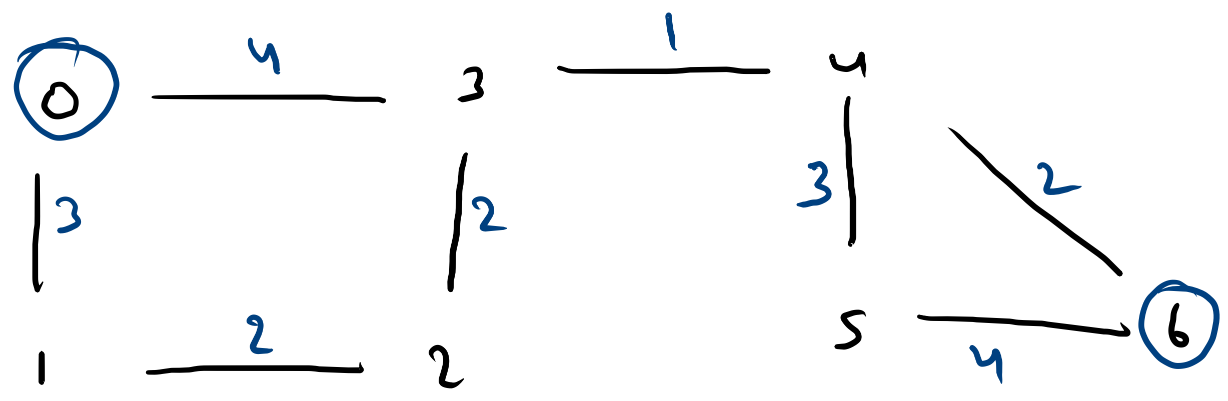


```
    printallpath(graph, src, dest, visited, src+"");
}

public static void printallpath(ArrayList<Edge>[] graph, int src, int des
    if(src == dest){
        System.out.println(psf);
        return;
    }

    visited[src] = true;
    for(Edge edge: graph[src]){
        if(visited[edge.nbr] == false){
            printallpath(graph, edge.nbr, dest, visited, psf+edge.nbr );
        }
    }
```

3

visited [src] = false





src = 0
des = 6

→ 0 1 2 3 4 5 6 (7)
w → 15 —

k=1

0 1 2 3 4 6 (6)
10

k=2

0 3 4 5 6
12 5

k=3

0 3 4 6
7 4

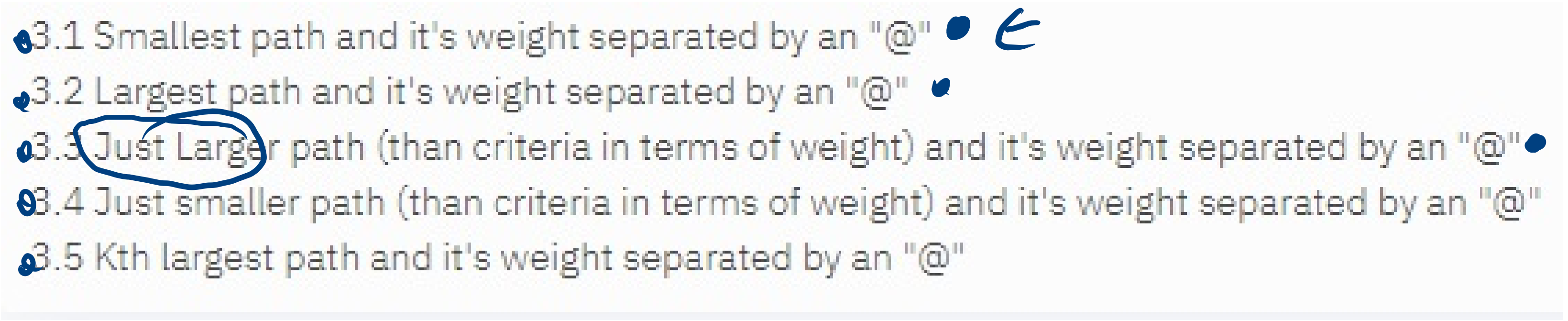
k=4

Criteria 11

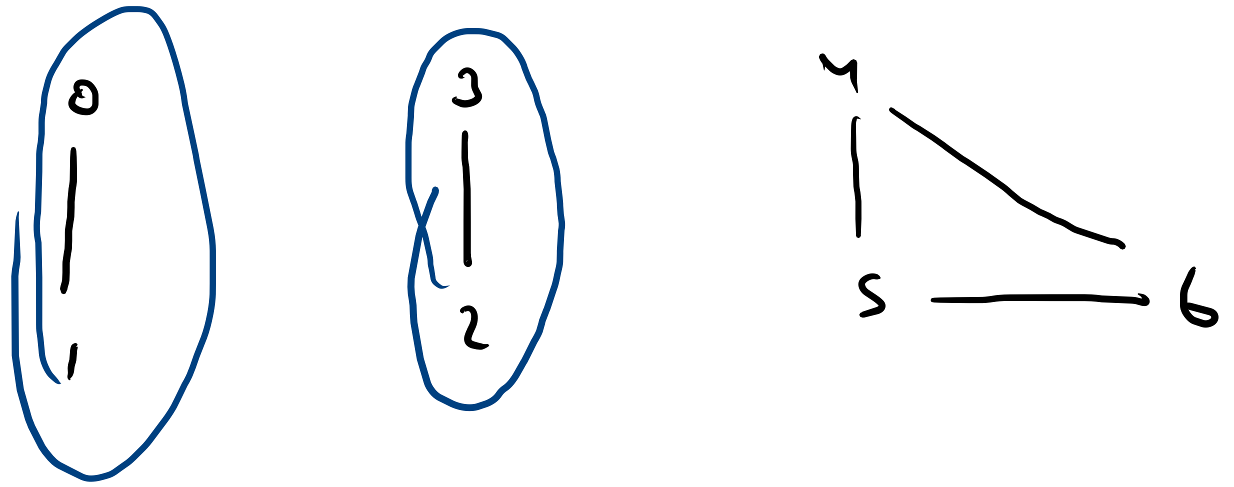
k=1

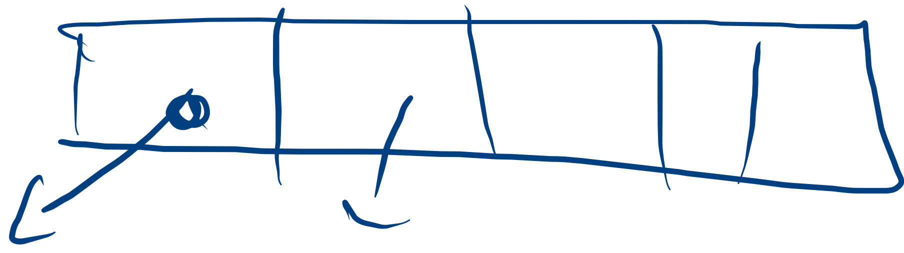
0346@7
0123456@15
03456@12
012346@10

pq
smaller weight



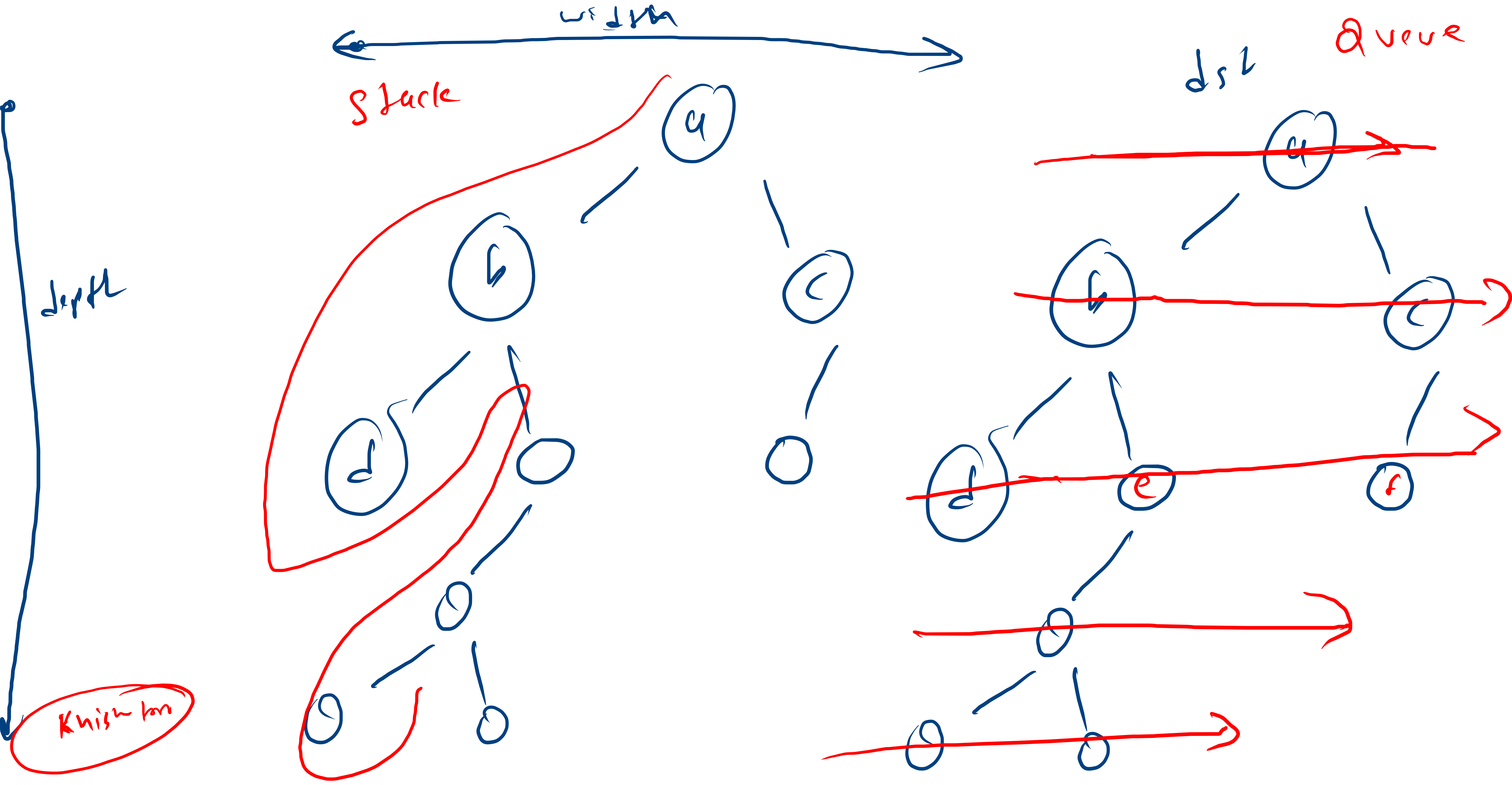
3.1 Smallest path and it's weight separated by an "@"
3.2 Largest path and it's weight separated by an "@"
3.3 Just Larger path (than criteria in terms of weight) and it's weight separated by an "@"
3.4 Just smaller path (than criteria in terms of weight) and it's weight separated by an "@"
3.5 Kth largest path and it's weight separated by an "@"





[[0, 1], [2, 3], [4, 5, 6]]

ArrayList<ArrayList<Integer>> al


width
Stack
depth
Knishton
a
b
c
d
dsl
Queue
a
b
c
d
e
f