We want to customize insertion and deletion in extendible hashing in the following manner.

While inserting a key, the bucket in which the key is supposed to be placed is called the target bucket.

- Hash table cannot store duplicates. If key is already present then just reject the insertion command.
- If the target bucket has enough space then just insert the key.
- If the target bucket does not have enough space, then split the corresponding bucket. There are two possible cases in this scenario
    - Case 1: Splitting of the target bucket requires doubling of the directory. In this case, double the directory as required and split the target bucket. However, also split all other buckets that have occupancy greater than 70%.
    - Case 2: Splitting of the target bucket does not require doubling of the directory. In this case, just split the target bucket.

While handling deletion of a key, if the key is absent then just ignore the deletion command.

While deleting a key, the bucket from which we delete the key is called the victim bucket. There are two possible cases while deleting a key.

- Case 1: Occupancy of victim bucket is above 25% after deleting the key. In this case, just delete the key.
- Case 2: Occupancy of victim bucket is less than or equal to 25% after deleting the key. In this case, check the occupancy of its mirror bucket. If the occupancy of mirror bucket is also less than or equal to 25% then merge both buckets into a single bucket.

Although we are reducing the number of buckets, we will never compact the directory.

Insertions and deletions can come in any order.

Whenever number of bucket change in the hash table, print the status of hash table in terms of the global depth and total number of buckets.

Example input:

Codes for various operations

1: Create new hash table (Parameters: global depth, size of each bucket)

2: Insert a new key (Parameters: Key). Do not allow duplicates.

3: Delete a key (Parameters: Key). Delete the key if present.

Example input and output

Line 1: Create new hash table

Line 2: Initial global depth of new hash table. For this input it is 2. We have to create an empty hash table with four slots in the directory and four corresponding buckets.

Line 3: Size of each bucket. For this input it is 5. Each bucket can contain 5 keys.

As we just created a new hash table, print its status. (Global depth and number of buckets)

The output should be "2 4"

Lines 4 to 29: Insert 13 keys in to the hash table. After processing these lines your hash table should look like as follows

Bucket 00: 4, 8, 12, 16, 20

Bucket 01: 1, 5, 9, 13

Bucket 10: 2, 6

Bucket 11: 3, 7

However, there is no change in the number of buckets. So we will not output anything.

Lines 30 and 31: Insert key 24. The key maps to bucket 00. This bucket is full. Local depth of the target bucket is same as the global depth. Hence will require the doubling of directory. As we are doubling the directory, we will also split the buckets that have occupancy more than 70%. Bucket 01 is one such bucket. It has four keys while its capacity is five. That gives us the occupancy value of 80%. After doubling the directory we will have following six buckets

000, 001, 10, 11, 100, 101.

The hash table would look like as follows

Bucket 000: 8, 16, 24

Bucket 001: 1, 9

Bucket 10: 2, 6

Bucket 11: 3, 7

Bucket 100: 4, 12, 20

Bucket 101: 5, 13

After inserting 24, the stautus of hahs table has changed to 3,6 (global depth, number of buckets).

Hence the output will be “3 6”.

Lines 32 and 33: Delete 24.

Lines 34 and 35: Delete 16. Occupancy of the bucket is not more than 25%. Check the occupancy of the mirror bucket (100). It is more than 25%. Therefore we will no merge buckets.

Lines 36 and 37: Delete 20

Lines 38 and 39: Delete 12. Occupancy of the bucket is not more than 25%. Its mirror also does not have occupancy more than 25%. We will merge the two buckets. You hash table should look as follows.

Bucket 00: 4, 8

Bucket 001: 1, 9

Bucket 10: 2, 6

Bucket 11: 3, 7

Bucket 101: 5, 13

Status of the hash table changed to “3 5”

Lines 40 and 41: Insert 32

How to submit your code:

<roll number>_ehash.cpp

How your code should compile:

```
g++ <roll number>_ehash.cpp
```

How your code should run:

```
a.out <input.txt  >output.txt
```