Exercise on Hash Tables - 44251017 HUANG,Jiahui

(1) Consider building a hashtable, such that *N*=13 is the size of the table. Show insertion of keys 23, 42, 14, 32, 13, 26, 18 to the hashtable in this order. Here, use double hashing, with primary hash function h(k) = k mode 13, and secondary hash function *d*(*k*) = 7 - *k* mod 7.

```
// java realization
int N = 13;
Integer[] hashTable = new Integer[N];  // Empty hash table

// Primary hash function: computes base index
int h(int k) {
    return k % 13;
}

// Secondary hash function: computes step size for probing
int d(int k) {
    return 7 - (k % 7);
}

// Insert using double hashing
void insertWithDoubleHashing(int key) {
    int i = 0;
    while (i < N) {
        int index = (h(key) + i * d(key)) % N;
        if (hashTable[index] == null) {
            hashTable[index] = key;
            return;
        }
        i++;
    }
    throw new RuntimeException("Hash table is full");
}

// Insert keys to hashtable
insert(23); // h=10, d=5 → insert to index 10
insert(42); // h=3,  d=7 → insert to index 3
insert(14); // h=1,  d=7 → insert to index 1
insert(32); // h=6,  d=3 → insert to index 6
insert(13); // h=0,  d=1 → insert to index 0
insert(26); // h=0,  d=2 → collision  → double hash → insert to index 2
insert(18); // h=5,  d=3 → insert to idnex 5
```

(2) Consider the hashtable constructed in (1). Now show a key that requires maximum number of probes in search. The key may not be the inserted seven keys.

```
int searchProbeTimes(int k) {
    int i = 0;
    while (i < N) {
        int index = (h(k) + i * d(k)) % N;
        if (table[index] == null) return i + 1;
        if (table[index].equals(k)) return i + 1;    // Found the key
        i++;
    }
    return N; // No empty space
}
```

To experience 7 collisions, a key must probe 7 times and hit 7 occupied indices before landing on an empty slot.

But in the current table, the 7 occupied indices are:

> {0, 1, 2, 3, 5, 6, 10}

They are not a complete arithmetic sequence modulo 13 (for example, step=2, 3, 4 cannot continuously jump out of this set of values), so the detection path of no key can fall exactly on these 7, which leads to the maximum number of detections being 6.

The key 80 requires maximum number of probes in search

> - Probe sequence:
>   - i = 0 → (2 + 0×4) % 13 = 2 → occupied by key 26
>   i = 1 → (2 + 1×4) % 13 = 6 → occupied by key 32
>   i = 2 → (2 + 2×4) % 13 = 10 → occupied by key 23
>   i = 3 → (2 + 3×4) % 13 = 1 → occupied by key 14
>   i = 4 → (2 + 4×4) % 13 = 5 → occupied by key 18
>   i = 5 → (2 + 5×4) % 13 = 9 → empty
>
> Total probe times = 6