2. Quadratic Probing

$f(i) = i^2$; /* a quadratic function */

【Theorem】 If quadratic probing is used, and the table size is prime, then a new element can always be inserted if the table is at least half empty.

Proof: Just prove that the first $\lfloor TableSize/2 \rfloor$ alternative locations are all distinct. That is, for any $0 < i \neq j \leq \lfloor TableSize/2 \rfloor$, we have

$$(h(x) + i^2) \% TableSize \neq (h(x) + j^2) \% TableSize$$

Suppose: $h(x) + i^2 = h(x) + j^2 \quad (\text{mod } TableSize)$

then: $i^2 = j^2 \quad (\text{mod } TableSize)$

$$(i + j) (i - j) = 0 \quad (\text{mod } TableSize)$$

TableSize is prime ➡ either $(i + j)$ or $(i - j)$ is divisible by TableSize

Contradiction !

For any x, it has $\lceil TableSize/2 \rceil$ distinct locations into which it can go. If at most $\lfloor TableSize/2 \rfloor$ positions are taken, then an empty spot can always be found. ■

Note: If the table size is a prime of the form $4k + 3$, then the quadratic probing $f(i) = \pm i^2$ can probe the entire table.

Read Figures 7.15 - 7.16 for detailed representations and implementations of initialization.

```
Position  Find ( ElementType Key, HashTable H )
{   Position  CurrentPos;
    int  CollisionNum;
    CollisionNum = 0;
    CurrentPos = Hash( Key, H->TableSize );
    while( H->TheCells[ CurrentPos ].Info != Empty &&
              H->TheCells[ CurrentPos ].Element != Key ) {
              CurrentPos += 2 * ++CollisionNum - 1;
              if ( CurrentPos >= H->TableSize )  CurrentPos - = H->TableSize;
    }
    return CurrentPos;
}
```

What is returned?

```
void  Insert ( ElementType Key, HashTable H )
{
   Position  Pos;
   Pos = Find( Key, H );
   if ( H->TheCells[ Pos ].Info != Legitimate ) { /* OK to insert here */
          H->TheCells[ Pos ].Info = Legitimate;
          H->TheCells[ Pos ].Element = Key; /* Probably need strcpy */
   }
}
```

Question: How to delete a key?

Note: ① Insertion will be seriously slowed down if there are too many deletions intermixed with insertions.

② Although primary clustering is solved, ***secondary clustering*** occurs – that is, keys that hash to the same position will probe the same alternative cells.

3. Double Hashing

$$f(i) = i * \text{hash}_2(x);$$ /* $\text{hash}_2(x)$ is the 2nd hash function */

□ $\text{hash}_2(x) \not\equiv 0$; □ make sure that all cells can be probed.

Tip: $\text{hash}_2(x) = R - (x \% R)$ with R a prime smaller than TableSize, will work well.

Note: ① If double hashing is correctly implemented, simulations imply that the **expected** number of probes is almost the same as for a **random** collision resolution strategy.
② Quadratic probing does not require the use of a second hash function and is thus likely to be **simpler and faster** in practice.

§5 Rehashing

Then what can we do?

- **Build another table that is about twice as big;**
- **Scan down the entire original hash table for non-deleted elements;**
- **Use a new function to hash those elements into the new table.**

If there are N keys in the table, then $T(N) = O(N)$

Question: When to rehash?

Answer:

① As soon as the table is half full
② When an insertion fails
③ When the table reaches a certain load factor

Note: Usually there should have been $N/2$ insertions before rehash, so $O(N)$ rehash only adds a constant cost to each insertion.

However, in an interactive system, the unfortunate user whose insertion caused a rehash could see a slowdown.

Read Figures 7.23
for detailed implementation of rehashing.