Lab5 Probing Collisions Commentary

The hash table is implemented as a dynamic array (vector). Each element of the array is a reference (pointer) to a record. The record is a class (struct) with two data fields - a key and a value.

See section "Algorithms" in the assignment for a general description of hash probing. The lookup function calls hash function to calculate first location (array index) and, if the location is occupied, calculates next location, until unoccupied location is found. The algorithm stops when the next location is the same as the first location.

Hash Table Size

In the textbook, unfortunately, the hash table size is 10. It is why it is 10 in the examples below.

In the assignment and real-world programs, you shall use a prime number size.

Hash Function and Index

In the textbook and in the examples below, to make them simple, the hash function is trivial.

hash(x) = x

The assignment uses a different hash function that scrambles number. In real-world programs use default hash function if appropriate.

To calculate an index, hash value must be adjusted via modulo table size (like index = hash % table size).

Linear Probing

If the location is occupied, next index = index+1. If the second location is occupied, third index = first index+2. In general, next index = first index+i, where i is increased by 1 on each attempt.

Quadratic Probing

If the first location is occupied, second index = first index+1. If the second location is occupied, third index = first index+4. In general, next index = first index+i*i, where i is increased by 1 on each attempt.

Double Hashing Probing

In addition to the main hash function, you define a secondary hash function, for example

hash2(x) = R-(x mod R), where x is a key, R is the maximum prime number that is less than the table size.

If the first location is occupied, second index = first index+hash2(x), then third index = first index+2*hash2(x). In general, next index = first index+i*hash2(x), where i is increased by 1 on each attempt.

Note that in the real program, you shall calculate hash2(x) only once.

Linear Probing Example

Let consider the linear probing example from the textbook.

The first index is calculated as hash(x) % table size.

Numbers to insert: 89 18 49 58 69 60,

0	1	2	3	4	5	6	7	8	9
								18	89

hash(18)=18, 18%10 is 8, empty, we insert 18 in 8th position

hash(89)=89, 89%10 is 9, so we insert it in 9th position

49%10 is 9, occupied, collision, using linear probing we check (9+1)%10 =0 position, it is empty, occupy it.

0	1	2	3	4	5	6	7	8	9
49								18	89

Next number is 58, 58%10 is 8, occupied, collision, we take next spot (8+1)%10, it is 9 – occupied, collision, try next, (9+1)%10 is 0, occupied, collision, try next (0+1)%10 is 1. Empty, take it.

0	1	2	3	4	5	6	7	8	9
49	58							18	89

Next number is 69; 69%10 is 9, busy, (9+1)%10 is 0, busy, (0+1)%10 is 1, busy, (1+1)%10 is 2, free, take it.

0	1	2	3	4	5	6	7	8	9
49	58	69						18	89

Next number is 60. 60%10 is 0, busy, (0+1) is 1, busy, (1+1)%10 is 2, busy, (2+1)%10 is 3, free, take it.

0	1	2	3	4	5	6	7	8	9
49	58	69	60					18	89

If the hash table stores values, you will have a pair record in each table element. For example, if key/values, the pairs are 89/a 18/b 49/c 58/d 69/e 60/f

0	1	2	3	4	5	6	7	8	9
49/c	58/d	69/e	60/f					18/b	89/a

Prime Numbers

Why is it not good to have non-prime table size?

For this example, let us assume that

- the table size is 10
- we use the double hashing probing
- for some x, hash(x)=3 and hash2(x) = 2
- elements with indexes 1, 3, 5, 7, 9 are occupied

So, the first index is 3. The hashing algorithm tries the following locations

3 ,5, 7, 9, (9+2)%10 is 1, STOP because (1+2)%10 is 3

The table can be half empty, but it will not find the empty locations.

Now, assume that the table size is 11. The hashing algorithm tries the following locations

3, 5, 7, 9, (9+2)%11 is 0, (0+2)%11 is 2, 4, 6, 8, 10, (10+2)%11 is 1, STOP because 3

The algorithm traverses the whole table without leaving even a single element unchecked. If there is an empty location, it will be found.

If the algorithm stops at the first location, we can officially announce, in case of put(), that table is full and we do not have a place to insert the record. Or in case of get(), that there is no record with this key in the table.