Topic3 Probing Collisions Example

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 this Example the hash table size is 11.

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

1. Linear Probing Collisions Example

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.

Linear Probing Example

Let consider the linear probing example with table size=11 (and for double hashing double factor R=7)

The first index is calculated as hash(x) % table size. For this Example hash(x) = x;

Former hash function for Lab3 in Spring 2021 is (hash(key) = `(key>>8)|((key&0xff)<<16))`

Keys are 89 18 51 62 73 75, 40 let value = key*2 for each key, just for convenience of calculation, in real word value is definitely does not as easy and does not necessary depend from it’s key.

89%11 is 1, it means that startIndex = 1, 18%11 is 7, it means that startIndex for key 18 is 7

0	1	2	3	4	5	6	7	8	9	10
	89						18			

Revision 20210913

hash(89)=89, 89%11 is 1, so we insert it in 1st position

hash(18)=18, 18%11 is 7, empty, we insert 18 in 7th position

51%11 is 7, occupied, one **collision** during inserting (put() function), using linear probing we check (7+1)%11 = 8 position, it is empty, occupy it. When try to get value 102 (we use get() function and get one more collision)

0	1	2	3	4	5	6	7	8	9	10
	89						18	51		

Next number is 62, 62%11 is 7, occupied, **collision(2)**, we take next spot (7+1)%11, it is 8– occupied, **collision(3)**, try next, (8+1)%11 is 9, empty, take this spot

0	1	2	3	4	5	6	7	8	9	10
	89						18	51	62	

Next number is 73; 73%11 is 7, busy, **collision(4)**, try (7+1)%11 is 8, busy, **collision(5)** (8+1)%11 is 9, busy, **collision(6)**, (9+1)%11 is 10, free, take it.

0	1	2	3	4	5	6	7	8	9	10
	89						18	51	62	73

Next number is 75. 75%11 is 9, busy, **collision(7)**, (9+1) %11 is 10, busy, **collision(8)**, (10+1)%11 is 0, free, take it.

0	1	2	3	4	5	6	7	8	9	10
75	89						18	51	62	73

Next number is 40. 40%11 is 7(=startIndex) , busy, **collision(9),** (7+1) %11 is 8, busy, **collision(10)**,(8+1) %11 is 9, busy, **collision(11)**,(9+1) %11 is 10, busy, **collision(12)**, (10+1)%11 is 0, busy, **collision(13)**, (11+1)%11 is 1, busy, **collision(14)**, (1+1)%11 is 2, free, take it.

0	1	2	3	4	5	6	7	8	9	10
75	89	40					18	51	62	73

If the hash table stores values, you will have a pair record in each table element. For example, if key/values, the pairs are

0	1	2	3	4	5	6	7	8	9	10
75/ 150	89/ 178	40/80					18/ 36	51/ 102	62/ 124	73/ 146

where key/value in each cell.

0	1	2	3	4	5	6	7	8	9	10
75/ 150	89/ 178	40/ 80					18/ 36	51/ 102	62/ 124	73/ 146

to put these numbers to the table we will get 14 collisions.

To retrieve these numbers from the table we will get the same 14 collisions. So we can have common function loopUp() that find empty spot to occupy. We call lookUp() function in put() function and we call lookUp() in get() function.

So to put these numbers to the table and retrieve these numbers from the table we will get 28 collisions in Linear Probing.

```
print table.size()=11

index=0 key=75 value=150

index=1 key=89 value=178

index=2 key=40 value=80

index=7 key=18 value=36

index=8 key=51 value=102

index=9 key=62 value=124

index=10 key=73 value=146

*** Linear probing Random Order End ***
```

2. Quadratic Probing Collisions Example

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.

Quadratic Probing Example.

Let consider the linear probing example with table size=11 (and for double hashing double factor R=7)

The first index is calculated as hash(x) % table size. For this Example hash(x) = x;

Former hash function for Lab3 in Spring 2021 is (hash(key) = `(key>>8)|((key&0xff)<<16))`

Keys are 89 18 51 62 73 75, 40 let value = key*2 for each key, just for convenience of calculation, in real word value is definitely does not as easy and does not necessary depend from it's key.

89%11 is 1, it means that startIndex = 1, 18%11 is 7, it means that startIndex for key 18 is 7

0	1	2	3	4	5	6	7	8	9	10
	89						18			

hash(89)=89, 89%11 is 1, so we insert it in 1st position

hash(18)=18, 18%11 is 7, empty, we insert 18 in 7th position

51%11 is 7, occupied, one **collision** during inserting (put() function), using quadratic probing we check (7+1*1)%11 = 8 position, it is empty, occupy it. When try to get value 102 (we use get() function and get one more collision)

0	1	2	3	4	5	6	7	8	9	10
	89						18	51		

Next number is 62, 62%11 is 7(startIndex=7), occupied, **collision(2)**, we take next spot (7+1*1)%11=8%11=8, occupied, **(collision (3))**, (7+2*2)=11%11 = 0, free, take this spot

0	1	2	3	4	5	6	7	8	9	10
62	89						18	51		

Next number is 73; 73%11 is 7, busy, **collision(4)**, try (7+1*1)%11=8%11 is 8, is busy, **collision(5)**, (7+2*2)%11=11%11=0 is busy, **collision(6)**, (7+3*3)%11=16%11=5, free, take it.

0	1	2	3	4	5	6	7	8	9	10
62	89				73		18	51		

Next number is 75. 75%11 is 9, free, take it.

0	1	2	3	4	5	6	7	8	9	10
62	89				73		18	51	75	

Next number is 40. 40%11 is 7, busy, **collision(7),** (7+1*1) %11=8%11 is 8, busy, **collision(8)**,(7+2*2) %11 is 11%11=0, busy, **collision(9)**,(7+3*3) %11=16%11 is 5 busy, **collision(10)**, (7+4*4)%11=23%11 is 1, busy, **collision(11)**, (7+5*5)%11=32%11 is 10, free, take it.

0	1	2	3	4	5	6	7	8	9	10
62	89				73		18	51	75	40

If the hash table stores values, you will have a pair record in each table element. For example, if key/values, the pairs are

0	1	2	3	4	5	6	7	8	9	10
62/ 124	89/ 178				73/ 146		18/ 36	51/ 102	75/ 150	40/80

where key/value in each cell.

0	1	2	3	4	5	6	7	8	9	10
62/ 124	89/ 178				73/ 146		18/ 36	51/ 102	75/ 150	40/ 80

to put these numbers to the table we will get 11 collisions.

To retrieve these numbers from the table we will get the same 11 collisions. So we can have common function loopUp() that find empty spot to occupy. We call lookUp() function in put() function and we call lookUp() in get() function.

So to put these numbers to the table and retrieve these numbers from the table we will get 22 collisions in Quadratic Probing.

```
print table.size()=11

index=0 key=62 value=124

index=1 key=89 value=178

index=5 key=73 value=146

index=7 key=18 value=36

index=8 key=51 value=102

index=9 key=75 value=150

index=10 key=40 value=80

*** Quadratic probing Random Order End ***
```

3. Double Hashing Probing Collisions Example

The hash table size is 11.

Double Factor is 7.

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.

Double Hashing Probing Example

Let consider the linear probing example with table size=11 (and for double hashing double factor R=7)

The first index is calculated as hash(x) % table size. For this Example hash(x) = x;

Former hash function for Lab3 Spring 2021 is (hash(key) = `(key>>8)|((key&0xff)<<16))`

Keys are 89 18 51 62 73 75, 40 let value = key*2 for each key, just for convenience of calculation, in real word value is definitely does not as easy and does not necessary depend from it's key.

89%11 is 1, it means that startIndex = 1, 18%11 is 7, it means that startIndex for key 18 is 7

0	1	2	3	4	5	6	7	8	9	10
	89						18			

hash(89)=89, 89%11 is 1, so we insert it in 1st position

hash(18)=18, 18%11 is 7, empty, we insert 18 in 7th position

51%11 is 7, (startIndex=7), occupied, one **collision(1)** during inserting (put() function), using double hashing probing

```
hash(key)=key;

doubleFactor=7;

table.size() is 11;

hashIndex(key)=hash(key)%table.size();

startIndex = hashIndex(key);

hash2(key)=doubleFactor_-(hash(key) % doubleFactor_);

f(i,key)= i*hash2(key);

index=(f(i, key)+startIndex)%table_.size();
```

we check (f(1,51)+7)%11, where f(1,51) is 1*hash2(51) is 1*(7 – (51%7)) = 7-2 is 5, so index= (5+7)%11 is 1, it is occupied, so **collision(2),** increase i by1, and repeat calculations of next index:

f(2,51) = 2*(7-51%7) =2*(7-2)=10

index = (f(2,51) +7)%11= (10+7)%11 is 6 it is free, take the spot.

0	1	2	3	4	5	6	7	8	9	10
	89					51	18			

Next number is 62, 62%11 is 7(startIndex=7), occupied, **collision(3)**, let's calculate next spot

we check (f(1,62)+7)%11, where f(1,62) is 1*hash2(62) is 1*(7 – (62%7)) = 7-6 is 1, so index= (1+7)%11 is 8, it is free, take the spot.

0	1	2	3	4	5	6	7	8	9	10
	89					51	18	62		

Next number is 73; 73%11 is 7, busy, **collision(4)**,

we check (f(1,73)+7)%11, where f(1,73) is 1*hash2(73) is 1*(7 – (73%7)) = 7-3 is 4, so index= (4+7)%11 is 0, it is free, take the spot.

0	1	2	3	4	5	6	7	8	9	10
73	89					51	18	62		

Next number is 75. 75%11 is 9, free, take it.

0	1	2	3	4	5	6	7	8	9	10
73	89					51	18	62	75	

Next number is 40. 40%11 is 7, busy, **collision(5),**

we check (f(1,40)+7)%11, where f(1,40) is 1*hash2(40) is 1*(7 – (40%7)) = 7-5 is 2, so index= (2+7)%11 is 9 it is busy**, collision(6),** increase i by1, and repeat calculations of next index:

f(2,40) = 2*(7-40%7) =2*(7-5)=4

index = (f(2,40) +7)%11= (4+7)%11 is 0, busy, **collision(7)** it is free, take the spot.

increase i by1, and repeat calculations of next index:

f(3,40) = 3*(7-40%7) =3*(7-5)=6

index = (f(3,40) +7)%11= (6+7)%11 is 2, it is free, take the spot.

0	1	2	3	4	5	6	7	8	9	10
73	89	40				51	18	62	75	

where, key/value in each cell.

0	1	2	3	4	5	6	7	8	9	10
73/ 146	89/ 178	40/ 80				51/ 102	18/ 36	62/ 124	75/ 150	

to put these numbers to the table we will get 7 collisions.

To retrieve these numbers from the table we will get the same 7 collisions. So we can have common function loopUp() that find empty spot to occupy. We call lookUp() function in put() function and we call lookUp() in get() function.

So to put these numbers to the table and retrieve these numbers from the table we will get 14 collisions in Double HashingProbing.

```
 print table_.size()=11

index=0 key=73 value=146

index=1 key=89 value=178

index=2 key=40 value=80

index=6 key=51 value=102

index=7 key=18 value=36

index=8 key=62 value=124

index=9 key=75 value=150


*** Double probing Random Order End ***
```