Array-based Hashtables

- For simplicity, we will assume that we only insert numeric keys into the hashtable
- hash(x) = x % B; where B is the number of buckets

Implementation

```
class Hashtable {
  int buckets[B];
  bool occupied[B];
}
```

Insert(35)

h=hash(35) = 35 % 9 = 8

	Occupied	buckets
0		
1		
2		
3		
4		
5		
6		
7		
8		

Insert

```
class Hashtable {
  int buckets[B];
  bool occupied[B];
}
```

Insert(35)

h=hash(35) = 35 % 9 = 8

Insert(13)

h=hash(13) = 13 % 9 = 4

	Occupied	buckets
0		
1		
2		
3		
4		
5		
6		
7		
8	×	35

Insert

```
class Hashtable {
  int buckets[B];
  bool occupied[B];
}
```

Insert(35)

h=hash(35) = 35 % 9 = 8

Insert(13)

h=hash(13) = 13 % 9 = 4

	Occupied	buckets
0		
1		
2		
3		
4	×	13
5		
6		
7		
8	×	35

Insert

```
class Hashtable {
  int buckets[B];
  bool occupied[B];
}
```

```
insert(x) {
  h = hash(x);
  buckets[h] = x;
  occupied[h] = true;
}
```

	Occupied	buckets
0		
1		
2		
3		
4	×	13
5		
6		
7		
8	×	35

Insert

```
class Hashtable {
  int buckets[B];
  bool occupied[B];
}
```

Insert(31)

h=hash(31) = 31 % 9 = 4

Collision

	Occupied	buckets
0		
1		
2		
3		
4	×	13
5		
6		
7		
8	×	35

Collision Resolution

- If h(x) is occupied, we try other buckets
- We try $h_{0(x)}, h_{1(x)}, h_2(x)$, ...
  until an empty bucket is found
- $h_i(x) = hash(x) + f(i)$
  where f is called the collision resolution function

Linear Probing

- $f(i) = i$

Insert(31)
h=hash(31) = 31 % 9 = 4
h_0=4 + 0 = 4 (Collision)
h_1=4 + 1 = 5

	Occupied	buckets
0		
1		
2		
3		
4	×	13
5		
6		
7		
8	×	35

Linear Probing

- $f(i) = i$

Insert(31)
h=hash(31) = 31 % 9 = 4
h_0=4 + 0 = 4 (Collision)
h_1=4 + 1 = 5

	Occupied	buckets
0		
1		
2		
3		
4	×	13
5	×	31
6		
7		
8	×	35

Linear Probing

- $f(i) = i$

Insert(20)
h=hash(20) = 20 % 9 = 2
h_0=2 + 0 = 2

	Occupied	buckets
0		
1		
2		
3		
4	×	13
5	×	31
6		
7		
8	×	35

Linear Probing

- $f(i) = i$

Insert(20)
h=hash(20) = 20 % 9 = 2
h_0=2 + 0 = 2

	Occupied	buckets
0		
1		
2	×	20
3		
4	×	13
5	×	31
6		
7		
8	×	35

Linear Probing

- $f(i) = i$

Insert(26)

h=hash(26) = 26 % 9 = 8

h_0=8 + 0 = 8 (Collision)

h_1=8 + 1 = 0

	Occupied	buckets
0		
1		
2	×	20
3		
4	×	13
5	×	31
6		
7		
8	×	35

Linear Probing

- $f(i) = i$

Insert(26)
h=hash(26) = 26 % 9 = 8
h_0=8 + 0 = 8 (Collision)
h_1=8 + 1 = 0

	Occupied	buckets
0	×	26
1		
2	×	20
3		
4	×	13
5	×	31
6		
7		
8	×	35

Linear Probing

- $f(i) = i$

Insert(40)
h=hash(40) = 40 % 9 = 4
h_0=4 + 0 = 4 (Collision)
h_1=4 + 1 = 5 (Collision)
h_2=4 + 2 = 6

	Occupied	buckets
0	×	26
1		
2	×	20
3		
4	×	13
5	×	31
6		
7		
8	×	35

Linear Probing

- $f(i) = i$

Insert(40)
h=hash(40) = 40 % 9 = 4
h_0=4 + 0 = 4 (Collision)
h_1=4 + 1 = 5 (Collision)
h_2=4 + 2 = 6

	Occupied	buckets
0	×	26
1		
2	×	20
3		
4	×	13
5	×	31
6	×	40
7		
8	×	35

Find

Find(40)
h=hash(40) = 40 % 9 = 4
h_0=4 + 0 = 4

		Occupied	buckets
	0	×	26
	1		
	2	×	20
	3		
? →	4	×	13
	5	×	31
	6	×	40
	7		
	8	×	35

Find

Find(40)

h=hash(40) = 40 % 9 = 4

h_0=4 + 0 = 4 (No match)

h_1=4 + 1 = 5 (No match)

	Occupied	buckets
0	×	26
1		
2	×	20
3		
4	×	13
? → 5	×	31
6	×	40
7		
8	×	35

Find

Find(40)

h=hash(40) = 40 % 9 = 4

h_0=4 + 0 = 4 (No match)

h_1=4 + 1 = 5 (No match)

h_2=4 + 2 = 6 (Match)

	Occupied	buckets
0	×	26
1		
2	×	20
3		
4	×	13
5	×	31
✓ → 6	×	40
7		
8	×	35

Find

Find(22)

$h = hash(22) = 22 \% 9 = 4$

$h_0 = 4 + 0 = 4$

		Occupied	buckets
	0	×	26
	1		
	2	×	20
	3		
? →	4	×	13
	5	×	31
	6	×	40
	7		
	8	×	35

Find

Find(22)

h=hash(22) = 22 % 9 = 4

h_0=4 + 0 = 4 (No match)

h_1=4 + 1 = 5 (No match)

		Occupied	buckets
	0	×	26
	1		
	2	×	20
	3		
	4	×	13
? →	5	×	31
	6	×	40
	7		
	8	×	35

Find

Find(22)

h=hash(22) = 22 % 9 = 4

h_0=4 + 0 = 4 (No match)

h_1=4 + 1 = 5 (No match)

h_2=4 + 2 = 6 (No match)

		Occupied	buckets
	0	×	26
	1		
	2	×	20
	3		
	4	×	13
	5	×	31
? →	6	×	40
	7		
	8	×	35

Find

Find(22)

h=hash(22) = 22 % 9 = 4

h_0=4 + 0 = 4 (No match)

h_1=4 + 1 = 5 (No match)

h_2=4 + 2 = 6 (No match)

h_3=4 + 3 = 7 (Empty)

		Occupied	buckets
	0	×	26
	1		
	2	×	20
	3		
	4	×	13
	5	×	31
	6	×	40
? →	7		
	8	×	35

Find

Find(22)

h=hash(22) = 22 % 9 = 4

h_0=4 + 0 = 4 (No match)

h_1=4 + 1 = 5 (No match)

h_2=4 + 2 = 6 (No match)

h_3=4 + 3 = 7 (Empty)

Return false (not found)

		Occupied	buckets
	0	×	26
	1		
	2	×	20
	3		
	4	×	13
	5	×	31
	6	×	40
? →	7		
	8	×	35

Insert & Find

```
void insert(x) {
  h = hash(x);
  while (occupied[h]) {
    h++;
  }
  buckets[h] = x;
  occupied[h] = true;
}
```

```
bool find(x) {
  h = hash(x);
  while (occupied[h]) {
    if (buckets[h] == x)
      return true;
    h++;
  }
  return false;
}
```

Do you spot an error?

Insert & Find

```
void insert(x) {
  h = hash(x);
  while (occupied[h]) {
    h = (h+1) % B;
  }
  buckets[h] = x;
  occupied[h] = true;
}
```

```
bool find(x) {
  h = hash(x);
  while (occupied[h]) {
    if (buckets[h] == x)
      return true;
    h = (h+1) % B;
  }
  return false;
}
```

Deletion

```
bool delete(x) {
  h = hash(x);
  while (occupied[h]) {
    if (buckets[h] == x) {
      occupied[h] = false;
      return true;
    }
    h = (h+1) % B;
  }
  return false;
}
```

Delete

Delete(13)
h=hash(13) = 13 % 9 = 4
h_0=4 + 0 = 4 (Match)

Is this correct?

? → 4

	Occupied	buckets
0	×	26
1		
2	×	20
3		
4	×	13
5	×	31
6	×	40
7		
8	×	35

Delete

Find(40)
h=hash(40) = 40 % 9 = 4
h_0=4 + 0 = 4 (No match)
return false!!!

		Occupied	buckets
	0	×	26
	1		
	2	×	20
	3		
? →	4		
	5	×	31
	6	×	40
	7		
	8	×	35

Revisit the Design

```
class Hashtable {
  int buckets[B];
  enum {E, O, D} status[B];
}
```

The status of each bucket is either
Empty, Occupied, or Deleted

Initially all buckets are empty

Delete

Delete(13)
h=hash(13) = 13 % 9 = 4
h_0=4 + 0 = 4 (Match)

		Status	buckets
	0	O	26
	1	E	
	2	O	20
	3	E	
? →	4	O	13
	5	O	31
	6	O	40
	7	E	
	8	O	35

Delete

Delete(13)
h=hash(13) = 13 % 9 = 4
h_0=4 + 0 = 4 (Match)

Find(40)
h=hash(40) = 40 % 9 = 4
h_0=4 + 0 = 4 (No match)
h_1=4 + 1 = 5 (No match)
h_2=4 + 2 = 6 (Match)
return true

		Status	buckets
	0	O	26
	1	E	
	2	O	20
	3	E	
? →	4	D	
	5	O	31
	6	O	40
	7	E	
	8	O	35

Insert, Find & Delete

```
void insert(x) {
  h = hash(x);
  while (status[h]==‘O’) {
    h = (h+1) % B;
  }
  buckets[h] = x;
  status[h] = ‘O’;
}
```

```
bool delete(x) {
  h = hash(x);
  while (status[h] != ‘E’) {
    if (buckets[h] == x) {
      status[h] = ‘D’;
      return true;
    }
    h = (h+1) % B;
  }
  return false;
}
```

```
bool find(x) {
  h = hash(x);
  while (status[h] != ‘E’) {
    if (status[h] == ‘O’ &&
          buckets[h] == x)
      return true;
    h = (h+1) % B;
  }
  return false;
}
```

Primary Clustering

- A cluster (a block) of buckets are all occupied
- Any key that hits the cluster will have to linearly probe until the end of the cluster

Primary clustering (buckets 4–6)

	Occupied	buckets
0	O	26
1	E	
2	O	20
3	E	
4	O	13
5	O	31
6	O	40
7	E	
8	O	35

Quadratic Probing

- $f(i) = i^2$

Insert(15)

	Status	buckets
0	E	
1	E	
2	E	
3	E	
4	E	
5	E	
6	E	
7	E	
8	E	

Quadratic Probing

- $f(i) = i^2$

Insert(15)

Insert(23)

	Status	buckets
0	E	
1	E	
2	E	
3	E	
4	E	
5	E	
6	O	15
7	E	
8	E	

Quadratic Probing

- $f(i) = i^2$

Insert(15)

Insert(23)

Insert(14)

	Status	buckets
0	E	
1	E	
2	E	
3	E	
4	E	
5	O	23
6	O	15
7	E	
8	E	

Quadratic Probing

- $f(i) = i^2$

Insert(15)

Insert(23)

Insert(14)
h_0=5+0 (Occupied)
h_1=5+1 (Occupied)
h_2=5+4=0 (Empty)

	Status	buckets
0	E	
1	E	
2	E	
3	E	
4	E	
5	O	23
6	O	15
7	E	
8	E	

Quadratic Probing

- $f(i) = i^2$

Insert(15)

Insert(23)

Insert(14)

h_0=5+0 (Occupied)

h_1=5+1 (Occupied)

h_2=5+4=0 (Empty)

Insert(33)

	Status	buckets
0	O	14
1	E	
2	E	
3	E	
4	E	
5	O	23
→ 6	O	15
7	E	
8	E	

Quadratic Probing

- $f(i) = i^2$

Insert(15)

Insert(23)

Insert(14)
h_0=5+0 (Occupied)
h_1=5+1 (Occupied)
h_2=5+4=0 (Empty)

Insert(33)
h_0=6+0 (Occupied)
h_1=6+1 (Empty)

	Status	buckets
0	O	14
1	E	
2	E	
3	E	
4	E	
5	O	23
6	O	15
7	E	
8	E	

Quadratic Probing

- $f(i) = i^2$

Insert(41)
h_0=5+0 (Occupied)
h_1=5+1 (Occupied)
h_2=5+4=0 (Occupied)
h_3=5+9=5 (Occupied)
h_4=5+16=3 (Empty)

	Status	buckets
0	O	14
1	E	
2	E	
→ 3	O	41
4	E	
5	O	23
6	O	15
7	O	33
8	E	

Secondary Clustering

- For two distinct keys x_1 and x_2, if $h(x_1) = h(x_2)$, then
- $h_i(x_1) = h_i(x_2), \forall i$
- In other words, if two keys start at the same position, they will probe the same sequence of buckets

	Status	buckets
0	O	14
1	E	
2	E	
3	O	41
4	E	
5	O	23
6	O	15
7	O	33
8	E	

Double Hashing

- $f(i) = i \cdot hash_2(x)$
- Where $hash_2$ is an alternative hash function
- Double hashing can eliminate both primary and secondary clustering