Hashing

Alexandra Stefan

Hash tables

- Tables
  - **Direct access** table (or key-index table): key => index
  - **Hash** table: key => hash value => index
- Main components
  - Hash function
  - Collision resolution
    - Different keys mapped to the same index
- Dynamic hashing – reallocate the table as needed
  - If an Insert operation brings the load factor to ½, double the table.
  - If a Delete operation brings the load factor to 1/8 , half the table.
- Properties:
  - Good time-space trade-off
  - Good for:
    - Search, insert, delete – O(1)
  - Not good for:
    - Select, sort – not supported, must use a different method
- Reading: chapter 11, CLRS (chapter 14, Sedgewick – has more complexity analysis)

Example

- Let M = 10, h(k) = k%10

Insert keys:
46 -> 6
15 -> 5
20 -> 0
37 -> 7
23 -> 3
25 -> 5 collision
35 ->
9 ->

Collision resolution:
- Separate chaining
- Open addressing
  - Linear probing
  - Quadratic probing
  - Double hashing

index	k
0	20
1	
2	
3	23
4	
5	**15**
6	46
7	37
8	
9	

Hash functions

- M – table size.
- h – hash function that maps a key to an index
  - We want random-like behavior:
    - any key can be mapped to any index with equal probability.

  - Typical functions:
    - h(k,M) = floor(((k-mn)/(mx-s))* M)
      - Here $mn \le k < mx$.
      - Simple, good if keys are random, not so good otherwise.
    - h(k,M) = k % M
      - Best M is a prime number. (Avoid M that has a power of 2 factor, will generate more collisions).
      - Choose M a prime number that is closest to the desired table size.
      - If $M = 2^p$, it uses only the lower order p bits => bad, ideally use all bits.
    - h(k,M) = floor(M*(k*A mod 1)) , 0<A<1 (in CLRS)
      - Good A = 0.618033 (the golden ratio)
      - Useful when M is not prime (can pick M to be a power of 2)
      - Alternative: h(k,M) = (16161 * (unsigned)k) % M (from Sedgewick)

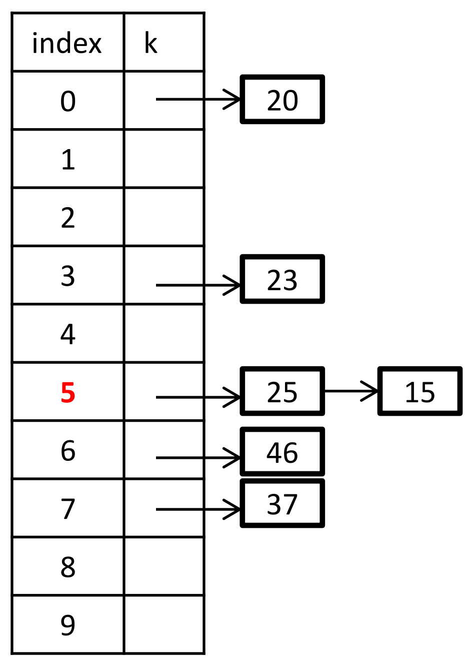
Collision Resolution: Separate Chaining

- $\alpha = N/M$ (N – items in the table, M – table size)
  - load factor
- Separate chaining
  - Each table entry points to a list of all items whose keys were mapped to that index.
  - Requires extra space for links in lists
  - Lists will be short. On average, size α.
  - Preferred when the table must support deletions.
- Operations:
  - Chain_Insert(T,x) - O(1)
    - insert x in list T[h(x.key)] at beginning. No search for duplicates
  - Chain_Delete(T,x) – O(1)
    - delete x from list T[h(x.key)]
  - Chain_Search(T, k) – $\Theta(1+\alpha)$ (both successful and unsuccessful)
    - search in list T[h(k)] for an item x with x.key == k.

Separate Chaining
Example: insert 25

- Let M = 10, h(k) = k%10

Insert keys:
46 -> 6
15 -> 5
20 -> 0
37 -> 7
23 -> 3
25 -> 5 collision
35
9 ->



Inserting at the beginning of the list is advantageous in cases where the more recent data is more likely to be accessed again (e.g. new students will have to go see several offices at UTA and so they will be looked-up more frequently than continuing students.

α =

Collision Resolution: Open Addressing

- $\alpha = N/M$ (N – items in the table, M – table size)
  - load factor
- Open addressing:
  - Use empty cells in the table to store colliding elements.
  - M > N
  - α – ratio of used cells from the table (<1).
  - Probing – examining slots in the table. Number of probes = number of slots examined.
  - *h(k,i,M)* where i gives the number of probes/attempts: i=0,1,2,… until successful hash.

  - Linear probing: $h(k,i,M) = (h_1(k) + i)\% M,$
    - If the slot where the key is hashed is taken, use the next available slot and wrap around the table.
    - Bad: Long chains
  - Quadratic probing: $h(k,i,M) = (h_1(k) + c_1 i + c_2 i^2)\% M,$
    - Bad: If two keys hash to the same value, they follow the same set of probes. But better than linear.
  - Double hashing: $h(k,i,M) = (h_1(k) + i * h_2(k)) \% M,$
    - $h_2(k)$ should not evaluate to 0. (E.g. use: $h_2(k) = 1 + k\%(M-1)$)
    - Use a second hash value as the jump size (as opposed to size 1 in linear probing).
    - Want: $h_2(k)$ relatively prime with M. (relatively prime: they have no common divisor)
      - M prime and $h_2(k) = 1 + k\%(M-1)$
      - $M = 2^p$ *and* $h_2(k) = odd$ (M and $h_2(k)$ will be relatively prime since all the divisors of M are powers of 2, thus even).
  - See figure 14.10, page 596 (Sedgewick) for clustering produced by linear probing and double hashing.

Open Addressing: quadratic Worksheet

$M = 10$, $h_1(k) = k\%10$.

Table already contains keys: 46, 15, 20, 37, 23

Next want to insert 25:

$h_1(25) = 5$ (collision: 25 with 15)

Index	Linear	Quadratic	Double hashing $h_2(k) = 1+(k\%7)$	Double hashing $h_2(k) = 1+(k\%9)$
0	20	20	20	20
1				
2				
3	23	23	23	23
4				
5	15	15	15	15
6	46	46	46	46
7	37	37	37	37
8				
9				

Linear probing

- $h(k,i,M) = (h_1(k) + i)\% M$
  (try slots: 5,6,7,8)

Quadratic probing example:

- $h(k,i,M) = (h_1(k) + 2i+i^2)\% M$
  (try slots: 5, 8)
- Inserting 35(not shown in table):
  (try slots: 5, 8, 3,0)

i (probe)	$h_1(k) + 2i+i^2$	%10
0		
1		
2		
3		
4		

Where will 9 be inserted now (after 35)?

Open Addressing: quadratic Answers

M = 10, $h_1(k) = k\%10$.

Table already contains keys: 46, 15, 20, 37, 23

Next want to insert 25:

$h_1(25) = 5$ (collision: 25 with 15)

Index	Linear	Quadratic	Double hashing $h_2(k) = 1+(k\%7)$	Double hashing $h_2(k) = 1+(k\%9)$
0	20	20	20	20
1				**25**
2				
3	23	23	23	23
4				
5	15	15	15	15
6	46	46	46	46
7	37	37	37	37
8	**25**	**25**		
9				

Linear probing

- $h(k,i,M) = (h_1(k) + i)\% M$
  (try slots: 5,6,7,8)

Quadratic probing example:

- $h(k,i,M) = (h_1(k) + 2i+i^2)\% M$
  (try slots: 5, 8)
- Inserting 35(not shown in table):
  (try slots: 5, 8, 3,0)

i (probe)	$h_1(k) + 2i+i^2$	%10
0	5+0=5	5
1	5+3=8	8
2	5+8=13	3
3	$5+2*3+3^2 = 5+15=20$	0
4		

Where will 9 be inserted now (after 35)?

Open Addressing : double hashing - Worksheet

M = 10, $h_1(k)$ = k%10.

Table already contains keys: 46, 15, 20, 37, 23

Try to insert 25:

$h_1(25)$ = 5 (collision: 25 with 15)

Double hashing example

- $h(k,i,M) = (h_1(k) + i^* h_2(k))$ % M

Choice of h_2 matters:

- $h_{2a}(k)$ = 1+(k%7): try slots: 5, 9,
  - $h_2(25)$ = 1+ (25%7) = 1+ 4 = 5 =>

    h(k,i,M) = (5 + i*5)%M => slots: 5,0,5,0,…

    Cannot insert 25.
- $h_{2b}(k)$ = 1+(k%9):
  - $h_2(25)$ = 1 + (25%9) = 1 + 7 = 8 =>

    h(k,i,M) = (5 + i*8)%M => slots: 5,3,1,9,7,5,…

Quadratic probing

i (probe)	$h_1(k) + 2i+i^2$	%10
0	5+0=5	5
1	5+3=8	8
2	5+8=13	3
3	$5+2*3+3^2$ = 5+15=20	0
4		

Index	Linear	Quadratic	Double hashing $h_2(k)$ = 1+(k%7)	Double hashing $h_2(k)$ = 1+(k%9)
0	20	20	20	20
1				25
2				
3	23	23	23	23
4				
5	15	15	15	15
6	46	46	46	46
7	37	37	37	37
8	25	25		
9				

Double hashing

i (probe)	Index	$(h_1(k) + i^*h_{2a}(k))$%M (5+i*5)%10
0		
1		
2		
3		

i (probe)	Index	$(h_1(k) + i^*h_{2b}(k))$%M (5+i*8)%10
0		
1		
2		
3		
4		
5		

Where will 9 be inserted now?

Open Addressing : double hashing - Answers

M = 10, $h_1(k)$ = k%10.
Table already contains keys: 46, 15, 20, 37, 23

Try to insert 25:

$h_1(25)$ = 5 (collision: 25 with 15)

Double hashing example

- h(k,i,M) = ($h_1(k)$ + i* $h_2(k)$) % M

Choice of h_2 matters:

- $h_{2a}(k)$ = 1+(k%7): try slots: 5, 9,
  - $h_2(25)$ = 1+ (25%7) = 1+ 4 = 5 =>
    h(k,i,M) = (5 + i*5)%M => slots: 5,0,5,0,…
    Cannot insert 25.
- $h_{2b}(k)$ = 1+(k%9):
  - $h_2(25)$ = 1 + (25%9) = 1 + 7 = 8 =>
    h(k,i,M) = (5 + i*8)%M => slots: 5,3,1,9,7,5,…

Quadratic probing

i (probe)	$h_1(k)$ + 2i+i^2	%10
0	5+0=5	5
1	5+3=8	8
2	5+8=13	3
3	5+2*3+3^2 = 5+15=20	0
4		

Double hashing

i (probe)	Index	($h_1(k)$ + i*$h_{2a}(k)$)%M (5+i*5)%10
0	5	(5+0)%10=5
1	0	(5+5)%10=0
2	5	(5+2*5)%10 =15%10= 5 Cycles back to 5 => Cannot insert 25
3	0	(5+3*5)%10=20%10= 0

i (probe)	Index	($h_1(k)$ + i*$h_{2b}(k)$)%M (5+i*8)%10
0	5	(5+0)%10=5
1	3	(5+8)%10=3
2	1	(5+2*8)%10 =21%10= 1
3	9	(5+3*8)%10=29%10= 9
4	7	(5+4*8)%10=37%10= 7
5	5	(5+5*8)%10=45%10= 5 Cycles back to 5

Index	Linear	Quadratic	Double hashing $h_2(k)$ = 1+(k%7)	Double hashing $h_2(k)$ = 1+(k%9)
0	20	20	20	20
1				25
2				
3	23	23	23	23
4				
5	15	15	15	15
6	46	46	46	46
7	37	37	37	37
8	25	25		
9				

Where will 9 be inserted now?

Open Addressing: Quadratic vs double hashing

$M = 10$, $h_1(k) = k\%10$.

Table already contains keys: 46, 15, 20, 37, 23

Try to insert 25:

$h_1(25) = 5$ (collision: 25 with 15)

Double hashing:
$h(k)=(h_1(k)+i*h_2(k))\%M$ where:
$h_1(k) = k\%M$
$h_2(k) = 1+(k\%(M-1)) = 1+(k\%9)$
$h_2(25) = 1+(25\%9) = 1+7 = 8$

Quadratic hashing with: $2i+i^2$
$h(k)= (h_1(k) + 2i+i^2)\%M$ where
$h_1(k) = k\%M$

i (probe)	Index	$h(k)=(h_1(k) + 2i+i^2)\%M$ $=(5 + 2i+i^2)\%10$ (k=25)
0		
1		
2		
3		
4		

i (probe)	Index	$h(k)=(h_1(k)+i*h_2(k))\%M$ $= (5+i*8)\%10$ (for k=25)
0		
1		
2		
3		
4		
5		

Index	Linear	Quadratic	Double hashing $h_2(k) = 1+(k\%7)$	Double hashing $h_2(k) = 1+(k\%9)$
0	20	20	20	20
1				**25**
2				
3	23	23	23	23
4				
5	15	15	15	15
6	46	46	46	46
7	37	37	37	37
8	**25**	**25**		
9				

Choice of h_2 matters. See $h_2(k) = 1+(k\%7)$
$h_2(25) = 1+4 = 5$ => h(25) cycles: 5,0,5,0
=> Could not insert 25.

Where will 9 be inserted now?

Open Addressing: Quadratic vs double hashing

$M = 10$, $h_1(k) = k\%10$.

Table already contains keys: 46, 15, 20, 37, 23

Try to insert 25:

$h_1(25) = 5$ (collision: 25 with 15)

Quadratic hashing with: $2i+i^2$
$h(k) = (h_1(k) + 2i+i^2)\%M$ where
$h_1(k) = k\%M$

i (probe)	Index	$h(k)=(h_1(k) + 2i+i^2)\%M$ $=(5 + 2i+i^2)\%10$ (k=25)
0	5	5+0=5
1	8	5+3=8
2	3	5+8=13
3	0	$5+2*3+3^2 = 5+15=20$
4		

Double hashing:
$h(k)=(h_1(k)+i*h_2(k))\%M$ where:
$h_1(k) = k\%M$
$h_2(k) = 1+(k\%(M-1)) = 1+(k\%9)$
$h_2(25) = 1+(25\%9) = 1+7 = 8$

i (probe)	Index	$h(k)=(h_1(k)+i*h_2(k))\%M$ $= (5+i*8)\%10$ (for k=25)
0	**5**	(5+0)%10=5
1	3	(5+8)%10=3
2	1	(5+2*8)%10 =31%10= 1
3	9	(5+3*8)%10=29%10= 9
4	7	(5+4*8)%10=37%10= 7
5	**5**	(5+5*8)%10=45%10= 5 Cycles back to 5

Choice of h_2 matters. See $h_2(k) = 1+(k\%7)$
$h_2(25) = 1+4 = 5$ => h(25) cycles: 5,0,5,0
=> Could not insert 25.

Index	Linear	Quadratic	Double hashing $h_2(k) = 1+(k\%7)$	Double hashing $h_2(k) = 1+(k\%9)$
0	20	20	20 ●	20
1				**25** ●
2				
3	23	23	23	23 ●
4				
5	15 ●	15 ●	15 ●	15 ●
6	46 ●	46	46	46
7	37 ●	37	37	37
8	**25** ●	**25** ●		
9				

Where will 9 be inserted now?

Search and Deletion in Open Addressing

- Searching:
  - Report as not found when land on an EMPTY cell
- Deletion:
  - Mark the cell as ‘DELETED’, not as an EMPTY cell
    - Otherwise you will break the chain and not be able to find elements following in that chain.
    - E.g., with linear probing, and hash function $h(k,i,10) = (k + i) \%10$, insert 15,25,35,5, search for 5, then delete 25 and search for 5 or 35.

Open Addressing: clustering

- Linear probing
  - primary clustering: the longer the chain, the higher the probability that it will increase.
  - Given a chain of size T in a table of size M, what is the probability that this chain will increase after a new insertion?
- Quadratic probing
  - Secondary clustering

Expected Time Complexity for Hash Operations
(under ‘perfect’ conditions)

Operation \Methods	Separate chaining	Open Addressing
Successful search	$\Theta(1+\alpha)$	$(1/\alpha)\ln(1/(1-\alpha))$
Unsuccessful search	$\Theta(1+\alpha)$	$1/(1-\alpha)$
Insert	$\Theta(1)$ When: insert at beginning and no search for duplicates	$1/(1-\alpha)$
Delete	$\Theta(1)$ Assumes: doubly-linked list and node with item to be deleted is given.	The time complexity does not depend only on α, (but also on the deleted cells). In such cases separate chaining may be preferred to open addressing as its behavior has better guarantees.
Perfect conditions:	simple uniform hashing	uniform hashing
Reference	Theorem 11.1 and 11.2	Theorem 11.6 and 11.8 and corollary 11.7 in CLRS

$\alpha = N/M$ is the load factor

Perfect Hashing

Similar to separate chaining, but **use another hash table instead of a linked list**.

- Can be done for ***static*** keys (once the keys are stored in the table, the set of keys never changes).
- Corollary 11.11:Suppose that we store N keys in a hash table using perfect hashing. Then the **expected storage used for the secondary hash tables is less than 2N**.

Hashing Strings
(Implementation - avoid number overflow)

- For long strings, the previous method will result in number overflow. E.g.:
  - 64-bit integers
  - 11 character long word: $c*128^{10} = c*(2^7)^{10} = c*2^{70}$
    - View the word as a number in base 128 (each letter is a 'digit').
- Solution: partial calculations:
  - Replace: $c_{10}*128^{10} + c_9*128^9 + \ldots c_1*128^1 + c_0$ with:
  - $((c_{10} * 128 + c_9) * 128 + c_8)*128+\ldots.+ c_1)*128 + c_0)$
  - Compute it iteratively and apply mod M at each iteration as shown in code below.

  - Are these two computations identical?
    - $(c_{10}*128^{10} \% M + c_9*128^9 \% M + \ldots c_1*128^1 \% M + c_0 \% M) \% M$
    - $((c_{10} * 128 + c_9) \% M * 128 + c_8) \% M *128+\ldots.+ c_1) \% M *128 + c_0) \% M$ (this one is the same as the given code)

```
// (Sedgewick)
int hash(char *v, int M)
  { int h = 0, a = 128;
    for (; *v != '\0'; v++)
      h = (a*h + *v) % M;
    return h;
  }
```

```
// improvement: use 127, not 128: (Sedgewick)
int hash(char *v, int M)
  { int h = 0, a = 127;
    for (; *v != '\0'; v++)
      h = (a*h + *v) % M;
    return h;
  }
```

Hashing Strings
(Mathematical discussion)

Given strings with 7-bit encoded characters

- View the string as a number in base 128 where each letter is a 'digit'.
- Convert it (the word) to base 10
- Apply mod M

Example:

- "now": 'n' = 110, 'o' = 111, 'w' = 119:
- h("now",M) = $(110*128^2 + 111*128^1 + 119)$ % M
- See Sedgewick page 576 (figure 14.3) for the importance of M for collisions
  - M = 31 is better than M = 64
  - When M = 64, it is a divisor of 128 =>

  $(110*128^2 + 111*128^1 + 119)$ % M = 119 % M => only the last letter is used. => if lowercase English letters: a,…,z=> 97,…,122 => (%64) =>

  26 indexes, [33,...,58], used out of [0,63] available.
    - Additional collisions will happen due to last letter distribution (how many words end in 't'?).