Polynomial Hash Function

$$h(\text{"}c_{k-1}c_{k-2}\cdots c_1 c_0\text{"}) = \left((\text{int})c_{k-1}x^{k-1} + (\text{int})c_{k-2}x^{k-2} + \dots + (\text{int})c_1 x^1 + (\text{int})c_0\right) \bmod M$$

$$= ((\cdots(((\text{int}\, c_{k-1})x + (\text{int})c_{k-2})x + \dots + (\text{int})c_1)x + (\text{int})c_0) \bmod M$$

M must be prime

Algorithm hashFunction(s)

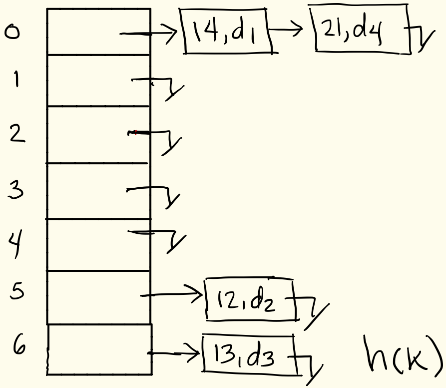
In: string $s = \text{"}c_{k-1}c_{k-2}\cdots c_1 c_0\text{"}$

Out: position for s in hash table

$$c\begin{cases} \text{val} \leftarrow (\text{int})c_{k-1} \\ \textbf{for } i = k-2 \textbf{ downto } 0 \textbf{ do} \\ \quad \text{val} \leftarrow (\text{val} * x + (\text{int})c_i) \bmod M \quad \Big\} c_1 \Big\} c_1(k-1) \\ \textbf{return } \text{val} \bmod M \end{cases}$$

$f(n) = \cancel{c} + \cancel{c_1}(k-\cancel{1})$ is $O(k)$ is $O(1)$ if k is constant

Separate Chaining



T

M = 7

14, d_1
12, d_2
13, d_3
21, d_4

$h(k) = K \underline{\bmod} \ 7$

	Worst	Average
get(k)	$O(n)$	$O(1)$
put(k,n)	$O(n)$	$O(1)$
remove(k):	$O(n)$	$O(1)$

Main drawback: uses too much memory

<u>Algorithm</u> get (key)
In: key
Out: Data for key or null if key not in table

$O(1)$ { $P \leftarrow T[h(\text{key})]$ (c_3)

<u>while</u> ($p \neq$ null) <u>and</u> (p.getKey() $\neq$ key) <u>do</u>) # iter = length of list
  $p \leftarrow p$.getNext() } c_2

c_1 { <u>if</u> p = null <u>then</u> <u>return</u> null
  <u>else</u> <u>return</u> p.getData()

$f(n) = \cancel{c_3} + \cancel{c_1} + c_2 \times \text{length list}$ is O(length of list)

Worst case (bad hash function)	Average Case
$O(n)$	$O(1)$

Open Addressing

Initialize

$h(k) = k \ \underline{\text{mod}}\ 7$

	T
0	DELETED
1	21, d_4
2	19, d_5
3	2, d_6
4	5, d_7
5	12, d_2
6	13, d_3

Re-hashing

lazy evaluation

14, d_1
12, d_2
13, d_3
21, d_4
19, d_5
2, d_6

$h(14) = 0$
$h(12) = 5$
$h(13) = 6$

$h(21) = 0$
$h(19) = 5$
$h(2) = 2$
$h(1) = 1$

put(5)

remove(14)

Initially every entry of T is null. $\neq$ DELETED $\neq$ keys

Algorithm get(key)
In: key
Out: Data for key, or null if key not in table

$c_1 \begin{cases} pos \leftarrow h(key) \\ c \leftarrow 0 \end{cases}$

$n \begin{cases} \underline{\text{while}}\ (T[pos] \neq null)\ \underline{\text{and}}\ (T[pos].getKey() \neq key)\ \underline{\text{do}}\ \{ \\ \quad c_2 \begin{cases} pos \leftarrow (pos+1)\ \underline{\text{mod}}\ M \\ c \leftarrow c+1 \\ \underline{\text{if}}\ c = M\ \underline{\text{then}}\ \underline{\text{return}}\ null \end{cases} \\ \} \end{cases}$

$\begin{cases} \underline{\text{if}}\ T[pos] = null\ \underline{\text{then}}\ \underline{\text{return}}\ null \\ \underline{\text{else}}\ \underline{\text{return}}\ T[pos].getData() \end{cases}$

$f(n) = c_1 + c_2 n$ is $\underline{O(n)}$

Linear Probing:

$h(k),\ (h(k)+1) \text{ mod } M,\ (h(k)+2) \text{ mod } M,\ (h(k)+3) \ \underline{\text{mod}}\ M, \ldots$

Double Hashing

$h(k),\ (h(k) + h'(k)) \text{ mod } M,\ (h(k) + 2h'(k)) \text{ mod } M,\ (h(k) + 3h'(k)) \text{ mod } M, \ldots$

Linear probing and doble hashing

$h(k) = k \ \underline{\text{mod}} \ 11$

$h(3)=3$
$h(14)=3$
$h(25)=3$
$h(5)=5$
$h(28)=6$
$h(91)=3$

Keys to store in the hash table:

3
14
25
5
28
91

	T
0	
1	
2	
3	3
4	14
5	25
6	5
7	28
8	91
9	
10	

cluster

Linear Probing

$h(k) = k \text{ mod } 11$

Second hash function

$h'(k) = q - (k \text{ mod } q)$

q prime

$h'(k) = 7 - (k \text{ mod } 7)$

$h'(14) = 7 - (14 \text{ mod } 7)$
$= 7 - 0 = 7$

$h'(25) = 7 - (25 \text{ mod } 7)$
$= 7 - 4 = 3$

$h'(28) = 7 - (28 \text{ mod } 7)$
$= 7 - 0 = 7$

$h'(91) = 7 - (91 \text{ mod } 7)$
$= 7 - 0 = 7$

	T
0	
1	
2	28
3	3
4	
5	5
6	25
7	
8	
9	91
10	14

Double Hashing

Double hashing does not create clusters

Double hashing: Why the size of the table must be prime

$h(k) = k \bmod 8$

$h'(k) = 7 - (k \bmod 7)$

2

6

10

0	
1	
2	2
3	
4	
5	
6	6
7	

$N = 8$

$h(2) = 2$

$h(6) = 6$

$h(10) = 2$

$h'(10) = 7 - (10 \bmod 7)$

$= 7 - 3 = 4$

The size of the hash table must be a prime number otherwise we will not be able to store the key 10 in the hash table.