EX.1 universal hashing

- $m \to$ size of hash table, $n \to$ # of elements

  $\Rightarrow$ c-universal hash function:

  $$P(h(x)=h(y)) \le \frac{c}{m}$$

  $\Rightarrow$ find / delete is in $O(1+\frac{cn}{m})$

- 1-universal class example:
  - $p \to$ a prime number. e.g. $p=17$
  - $w = \lfloor \log_2 p \rfloor$, e.g. $w = \lfloor \log_2 17 \rfloor = 4$
  - keys are bitstrings of length $k \cdot w$

    e.g. keys are 32-bit $\Rightarrow$ $k=8$
  - Interpret each substring of length w as a natural number, get $x=(x_1, x_2, \ldots, x_k)$

    $$\text{key} = \overbrace{\underbrace{0110}_{x_1=6}\ \underbrace{1101}_{x_2=13}\ \cdots\cdots\ \underbrace{0011}_{x_8=3}}^{32\text{ bits}}$$
  - For $a=(a_1, a_2, \ldots, a_k)$, $(0 \le a_i \le p-1)$

    define $h_a(x) = \left(\sum_{i=1}^{k} a_i x_i\right) \bmod p$

    e.g., $a = (16, 2, \ldots, 7)$

    $h_a(x) = (16 \times 6 + 2 \times 13 + \ldots + 7 \times 3) \bmod 17$
  - To implement, you can keep $a=(a_1, \ldots, a_k)$ as a property of your class and update it when rehash is called

EX.2 Linear Hashing variant

- l: maximum length of the list stored in an entry
- M: size of the hash table (# of entries)
- m: fixed number (constant), $m \le M$
- h: a hash function that takes values in $\{0, 1, \ldots, m-1\}$
- $h_0 = h$, $h_{j+1}(e) = h_j(e) + m \cdot 2^j$

$$\Rightarrow h_j(e) = m \cdot (2^{j-1} + 2^{j-2} + \ldots + 2^0) + h_0(e)$$

$$= h(e) + m \cdot (2^j - 1)$$

- The hash value of an element e is:

  $h_j(e)$, s.t. j is maximal with $h_j(e) \le M-1$

(i) Whenever an element e is given:

$$h_j(e) \le M-1 \Leftrightarrow h(e) + m \cdot (2^j - 1) \le M-1$$

- compute $h(e)$ to get j
- compute $h_j(e)$ to the hash value

(ii) One entry is full (length $= l$)

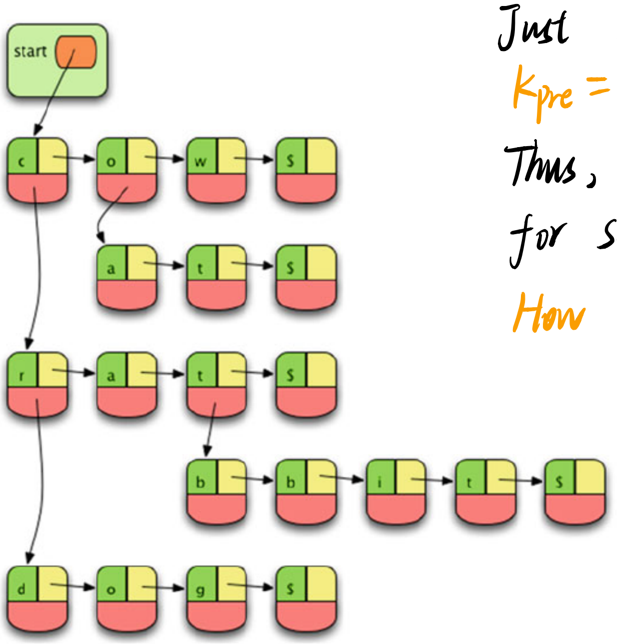
↳ rehash, split it into two sublist and distribute one of them to location h_{j+1}

↳ Increase M accordingly

↳ split other lists if neccessary since M changed

(iii) If using h_{j+1} is permitted, even though M has changed, we can keep other lists unchanged (but this increases the complexity of search. think about why)

EX.3 TRIE



(i) Unique prefixes of min length

e.g., to store:

$k = \{cow, cat, rat, rabbit, dog\}$

Just store:

$K_{pre} = \{co, ca, rat, rab, d\}$

Thus, reduce the time required for search.

How to implement?

One solution:

Instead of end with "#", end with a new kind of "tail nodes" that contain the left letters after prefixes.

(ii)

To implement a key-value storage, just modify the class definition of node, i.e., add key and value properties in node class

Other ways may also work, implementation details are up to you.