Closed Hashing

• We have presented open hashing, where all items that hash to the same position are stored in a bucket that is represented by a linked list.

• With closed hashing, whenever an item is hashed to a position that is already occupied, search for some other place in the table to put it.

• Items (or pointers to them) are stored in the table (all table positions have ≤ 1 element); "buckets" consist of a collection of table positions.

• PROBE(i) = A function that returns a value in the range 0 to m–1.

  Note: Generally, starting at any position, when repeatedly applying PROBE, we want to cycle through the entire hash table before a value is repeated.

Examples:

  Linear probing: Scan from that position forward (wrapping around if you get to the end of the table) until an empty position is found.

  Quadratic probing: Scan forward by quadratic amounts (in the hope of avoiding bunching up of entries). For example, when m is a prime of the form 4k+3, all positions get visited by searching these positions:

  \[ i + j^2 \mod m \] and \[ i - j^2 \mod m \], \( 1 \leq j \leq (m-1)/2 \)
Basic operations:

INITIALIZE $A$:

Set each value of $A$ to a special $nil$ value that is not an integer in the range 0 to $m-1$ and not equal to any hashed item.

SEARCH($d$):

$$i = h(d)$$

while ($A[i] \neq nil$) and ($A[i] \neq d$) do $i = \text{PROBE}(i)$

return $i$

MEMBER($d$):

return $A[\text{SEARCH}(d)]$ /*$nil$ means $d$ was not found*/

INSERT($d$):

$$i = \text{SEARCH}(d)$$


Note: Although it is possible to implement DELETE, closed hashing is usually used when only INSERT and MEMBER operations are needed. In any case, table size is important; performance goes way down when the table is close to full and an INSERT fails when the table becomes full.