Collision resolution by chaining

- Collision resolution by chaining is simple.
- Use hashing function on the key to get the index to a table of pointers.
- Each pointer points to a linked list whose nodes store the actual key.
- When two keys hash to same table index, a collision occurs.
- But this collision is handled quite easily by inserting the new key into the linked list pointed to by the pointer provide by the hash value.
- This means by chaining or linking the nodes storing the keys that map to same hash value the collisions are handled.
- Since hash function is assumed to be designed with care, it likely that only collisions may occur.

Search and delete

- When a membership search is required, we first obtain the hash value of the key being searched.
- After that traverse the appropriate linked list following the pointer provided by the hash value.
- Clearly each linked list will contain only O(1) nodes if the hash function is carefully designed as explained earlier and the number of keys n is $O(M)$.
- The deletion can also be done by proceeding like search and then deleting the node corresponding to the key from the appropriate linked list.

Example

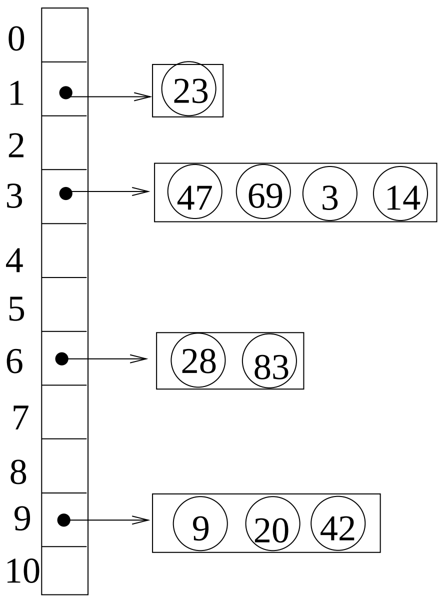

Table size = 11, using h(x) = x mod 11

Open addressing

- Chaining is fine if space is not a problem.
- Furthermore, there may a particular sequence of input keys which may generate a long chain of linked keys.
- To remedy this we can store the key directly in the table.
- But it requires complicated handling of collisions.
- There are several methods for the same. These methods collectively called as *open addressing* schemes.
- We must ensure that all the methods should peg the load factor to at most 1, that is $n \leq M$.

Linear probing

- We when we try to insert a key x that results in a collision we try out the sequentially next available slot in the table.

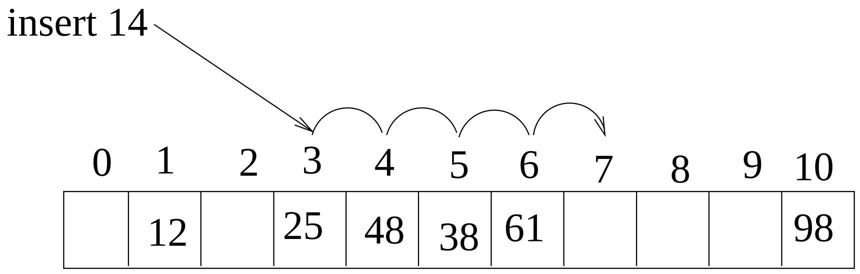

Table size = 11, using h(x) = x mod 11

Deletion with linear probe

- Deletion is slightly complicated. Suppose x to be deleted.
- Then the table should be restored to the position which was before the key x was inserted.
- The restoration would mean shifting the all the keys in the slots after x's home which belong to the same cluster group as x.
- However, all the keys of the same cluster group which appear in the slots before x should not be disturbed.
- For example, the figure above if 48 were to be deleted, then restoring the table would mean shifting up 38 and 61 by one position to left.

Deletion with linear probe

- Clearly this restoration is possible, but a simple technique avoid the shifting is to mark the object as deleted.
- This would mean both search and insertion algorithms should be modified to take care of these special object slots differently from empty slots.
- Namely, search for a key x should skip the occupied slots which do not contain x as well as the slots marked with deleted object until it either find x or reaches an empty slot.
- Obviously reaching an empty slot means the cluster is over.
- Clusturing definitely slows down the search in linear probing.

Quadratic probe

- Another open addressing scheme called quadratic probing helps to remove the clustering of the type formed by linear probe.
- The space between two colliding keys increase in a quadratic sequence.
- More precisely, the indices of the slots are examined in order $(i + f(j)) \bmod M$, where $f(j) = j^2$ for $j = 0, 1, \ldots$ until an empty slot is found.
- So slot sequence examined is $\{i, i+1, i+4, i+9, i+16, \ldots\}$.

Quadratic probe

- As clear from the pattern of indices in the preceding sequence, it form another kind of clustering known as secondary clustering.
- In this type of clustering the a cyclic pattern of grouping is noticed in the set of filled slots for the keys that collide.
- So, if the table size is not a prime, it is possible that the pattern of occupied slots repeated in cyclic pattern and that another key that collide with same set of keys cannot be a new insertion can not go through even when an empty slot may exist.

Quadratic probe

Suppose a hash table is of size M where M is prime. If quadratic probe is used then slots examined during first $M/2$ probes are distinct.

- Proof is by contradiction.
- Let $i < j \leq M/2$ be two probes within first $M/2$ probes such that

$$(h(x) + i^2) \bmod M = (h(x) + j^2) \bmod M$$

- Then there exists an integer $q > 0$ such that

$$\begin{aligned} h(x) + j^2 &= h(x) + i^2 + qM \\ j^2 &= i^2 + qM \\ j^2 - i^2 &= qM \\ (j-i)(i+j) &= qM \end{aligned}$$

- M is a prime, therefore, $j - i$ or $j + i$ must be divisible by M.
- Since $i < j \leq M/2$, $(j + i) < M$ and $(j - i) < M$.
- Therefore assumption that i and j probes visit same slot is a contradiction.

Quadratic probe

- Smallest i and j that probe to the same slot.
- This happens when $q = 1$. We also remove restriction on the range of i and j, and define instead $0 < i < j <= M$.
- Let $q = 1$. Then from earlier discussion if i and j probe the same slot then $(j + i)(j - i) = M$.
- Since M is prime we must have $j + i = M$ as we know $i < j \leq M/2$.
- This also means $j - i = 1$.
- So we find $2i + 1 = M$ or $i = (M - 1)/2$.
- Then $j = (M + 1)/2$.
- The implication of this result is using quadratic probe method *an empty slot can be always be found provided the table is half empty*
- In fact, we can use *rehashing* to double the table size every time it becomes half full.

Double hashing

- Designed to avoid all clusterings.
- Another hash function is used for resolution of collision.
- Suppose $h(x)$ maps the key x to index i, if a collision is detected then examine the slot indexed as $i + f(j)$, where $f(j) = j.h'(x)$.
- $h'(.)$ is a secondary hash function chosen carefully to minimize the clustering.
- Obviously the secondary hash should not evaluate to 0.
- A common choice is $h'(x) = M' - (x \text{ mod } M')$, where $M' < M$ is a another prime number.