Hashing - resolving collisions

Open addressing : linear probing

- hash function: $(h(k)+i) \bmod m$ for $i=0, 1, \ldots, m-1$
- **Insert** : start with the location where the key hashed and do a sequential search for an empty slot.
- **Search** : start with the location where the key hashed and do a sequential search until you either find the key (success) or find an empty slot (failure).
- **Delete** : (lazy deletion) follow same route but mark slot as DELETED rather than EMPTY, otherwise subsequent searches will fail (why?).

Hashing - resolving collisions

Open addressing : linear probing

- Disadvantage: primary clustering
  - long sequences of used slots build up with gaps between them.
  - result : performance near sequential search

Hashing - resolving collisions

Open addressing : quadratic probing

- Probe the table at slots $(h(k)+i^2) \bmod m$ for i=0, 1,2, 3, ..., m-1
- **Careful** : for some values of *m* (hash table size), very few slots end up being probed.
  - Try m=16
  - It <u>is</u> possible to probe all slots for certain *m*s.
- For prime *m*, we get pretty good results.
  - if the table is at least half-empty, an element can always be inserted

Hashing - resolving collisions

Open addressing : quadratic probing

- Better than linear probing but may result in secondary clustering: if $h(k_1)==h(k_2)$ the probing sequences for k_1 and k_2 are exactly the same
- general hash function: $(h(k)+c_1 i+c_2 i^2) \bmod m$

Hashing - resolving collisions

Open addressing : double hashing

- The hash function is $(h(k)+i\, h_2(k)) \bmod m$
- In English: use a second hash function to obtain the next slot.
- The probing sequence is:
  $h(k),\ h(k)+h_2(k),\ h(k)+2h_2(k),\ h(k)+3h_3(k), \ldots$
- Performance :
  - Much better than linear or quadratic probing.
  - Does not suffer from clustering
  - BUT requires computation of a second function

Hashing - resolving collisions

Open addressing : double hashing

- The choice of $h_2(k)$ is important
  - It must never evaluate to zero
    - consider $h_2(k)=k \bmod 9$ for $k=81$
- The choice of m is important
  - If it is not prime, we may run out of alternate locations very fast.
- If m and $h_2(k)$ are relatively prime, we'll end up probing the entire table.
- A good choice for h_2 is $h_2(k)=p-(k \bmod p)$ where p is a prime less than m.

Hashing - resolving collisions

Open addressing : random probing

- Use a pseudorandom number generator to obtain the sequence of slots that are probed.

Hashing - resolving collisions

Open addressing : expected # of probes

- Assuming uniform hashing...
  - Insert/Unsuccessful search : $1/(1-\lambda)$
  - Successful search : $(1+\ln(1/(1-\lambda)))/\lambda$

Hashing - resolving collisions

Example

- $m = 13$
- sequence of keys: 18-26-35-9-64-47-96-36-70
- $h_1(k) = k \bmod 13$
- Insert the sequence into a hash table using
  - linear probing
  - quadratic probing
  - double hashing with $h_2(k) = k \bmod 7 + 6$

Hashing - rehashing

- If the table becomes too full, its performance falls.
  - The O(1) property is lost
- Solution:
  - build a bigger table (e.g. approximately twice as big) and rehash the keys of the old table.
  - When should we rehash?
    - when the table is half full?
    - when an insertion fails?
    - when a certain load factor has been reached?