Data Structures and Algorithms: Lecture 12

Barbara Morawska

October 16, 2018

Open addressing

- Elements of a set are stored in the hash table.
- This method avoids pointers (and linked lists).
- But the hash table can fill up and no new elements can then be stored in it.
- Problem: find free slots in the table so that you can insert a new element.

Inserting into the open addressing hash table

- Successively probe the hash table to find a free slot
- The sequence of probes depends on the key (value) of the element.
- The hash function takes as input the key and a probe number.

The hash function extended with a probe number

$$h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$$

A probe sequence:

$$< h(k,0), h(k,1), \dots, h(k, m-1) >$$

is a permutation of $< 0, 1, \dots, m-1 >$

- For now we assume that the elements of our dynamic set do not contain any satellite data.
- And each slot of the table contains either a key of an element of NIL

Pseudocode for insertion

Notice: we insert a key (value), not a pointer.

```
Procedure HASH-INSERT(T, k)
i = 0
repeat
    j = h(k, i)
    if T[j] == NIL then
        T[j] = k
        return j
    else
        i = i + 1
until i == m
error "hash table overflow"
```

Pseudocode for search

Searching follows the same path.

Search fails if an empty slot is found.

Procedure HASH-SEARCH(T, k)

```
i = 0
repeat
    j = h(k, i)
    if T[j] == k then
        return j
    else
        i = i + 1
until T[j] == NIL or i == m
error "hash table overflow"
```

Deletion is difficult

- If we put NIL in the slot for the key, a subsequent search can fail incorrectly.
- Solution: mark such slot with DELETED, not just NIL.
- Then no need to modify the procedure SEARCH
- Notice: now SEARCH time does not depend on load factor, hence chaining is often preferred.

Time analysis

- Assume uniform hashing: any probe sequence is equally likely.
- There are $m!$ possible probe sequences.
- The following techniques do not satisfy this assumption: they generate only up to m^2 different probe sequences

The following are 3 kinds of defining different probe sequences:

1. linear probing
2. quadratic probing
3. double hashing

Linear probing

- We need an auxiliary hash function:

$$h' : U \to \{0, 1, \ldots, m-1\}$$

- Then the definition of hashing function is:

$$h(k, i) = (h'(k) + i) \bmod m$$

for $i = 0, 1, \ldots, m-1$

- Obviously, $h(k, 0)$ determines the probe sequence. Hence there are only m different probe sequences.
- Problem: primary clustering $(h(k_1, 0) = h(k_2, 0))$,
- long runs of occupied slots and average searching time increases.

Quadratic probing

- We need an auxiliary hash function, h'
- We need two positive auxiliary constants: c_1, c_2
- Then the hashing function is:

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

- Initial probing position is $T[h'(k)]$
- This is better than linear probing, but
- If all the table should be used, m, c_1, c_2 have to be constrained.
- If $h(k_1, 0) = h(k_2, 0)$ the sequences are the same (clastering)
- Hence only m distinct probe sequences!

Double hashing

The best method for open addressing.

- We need two auxiliary hashing functions: h_1, h_2
- Then the definition of double hashing function is:

$$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

- The initial probe is $T[h_1(k)]$
- The successive probes offset this value for: $i \cdot h_2(k) \bmod m$

Example

- Let T be a table of length $m = 13$
- $h_1(k) = k \bmod 13$, $h_2(k) = 1 + (k \bmod 11)$
- Where we will insert $79, 69, 98, 72, 50, 14$ in this order?
- What should be the value of $h_2(k)$ so that all table is searched/ probed?

How to choose m and h_2 in the case of double-hashing?

Value of h_2 should be relatively prime to m

- Choose $m = 2^p$ and h_2 to return always odd number
- Choose m to be a prime and h_2 to return an integer smaller than m
  Example:
  - $h_1(k) = k \bmod m$
  - $h_2(k) = 1 + (k \bmod m')$, where $m' = m - 1$

Advantage of double hashing

- When m and h_2 are chosen in the way above, all possible pairs $(h_1(k), h_2(k))$ yield different probe sequences.
- We have then $\Theta(n^2)$ different probe sequences.

Analysis of open-address hashing

- Assume uniform hashing functions (each probing sequence is equally likely)
- We use the notion of load factor for the table: $\alpha = n/m$
- Notice: $\alpha \leq 1$.
- Problem: What is the expected number of probes for an unsuccessful search?

Time for unsuccessful search

Theorem

The expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$.

Proof

- In an unsuccessful search every slot in the table is occupied, and the last slot to be probed is empty.
- X is a random variable (function on searching event) that returns the number of probes in an unsuccessful search
- Then the event $\{X \geq i\}$ occurs when there are at least i probes required in a search
- Notice: $X = I\{X \geq 1\} + I\{X \geq 2\} + \cdots + I\{X \geq m\}$
- Hence $E[X] = E[I\{X \geq 1\} + I\{X \geq 2\} + \cdots + I\{X \geq m\}]$
- By linearity:
  $E[X] = E[I\{X \geq 1\}] + E[I\{X \geq 2\}] + \cdots + E[I\{X \geq m\}]$

Time for unsuccessful search

- $E[X] = \sum_{i=1}^{m} Pr\{X \geq i\}$
- What is $Pr\{X \geq i\}$
- An event A_i: ith probe occurs and the slot is occupied.
- An event $\{X \geq i\}$ is an intersection:

$$A_1 \cap A_2 \cap \cdots \cap A_{i-1}$$

$$\begin{aligned} Pr\{X \geq i\} &= Pr\{A_1 \cap A_2 \cap \cdots \cap A_{i-1}\} \\ &= Pr\{A_1\} \cdot Pr\{A_2 \mid A_1\} \cdots Pr\{A_{i-1} \mid A_1 \cap \cdots \cap A_{i-2}\} \end{aligned}$$

- $Pr\{A_1\} = n/m$
- $Pr\{A_2 \mid A_1\} = n - 1/m - 1$
- $Pr\{A_j \mid A_1 \cap A_2 \cap \cdots A_{j-1}\} = n - (j-1)/m - (j-1)$
- This means the number of remaining $n - (j-1)$ elements divided by the number of $m - (j-1)$ remaining slots.

Time for unsuccessful search

$$Pr\{X \geq i\} = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \leq (\frac{n}{m})^{i-1} = \alpha^{i-1}$$

$$\begin{aligned} E[X] &= \sum_{i=1}^{\infty} Pr\{X \geq i\} \\ &\leq \sum_{i=1}^{\infty} \alpha^{i-1} \\ &= \sum_{i=0}^{\infty} \alpha^{i} \\ &= \frac{1}{1-\alpha} \end{aligned}$$

Time for unsuccessful search

Theorem

The expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$.

Conclusion

If α is $O(1)$, then an unsuccessful search runs in $O(1)$ time.
For example:

- If the table is half-full, $\alpha = 0.5$
- An average number of probes in an unsuccessful search is:
  $$\frac{1}{1-0.5} = 2$$
- If the table is 90% full:
- An average number of probes in an unsuccessful search is:
  $$\frac{1}{1-0.9} = 10$$

Corollary

Assume uniform hashing.

Corollary (11.7)

Inserting an element into open-address hash table with load factor α requires at most $\dfrac{1}{1-\alpha}$ probes on average.

Proof.

An element is inserted only if there is a free slot. Hence $\alpha < 1$. Inserting a key requires unsuccessful search, followed by placing the key into free slot.

The expected number of probes is $\dfrac{1}{1-\alpha}$. $\square$

Analysis of successful search

Theorem

If the load factor of a table is $\alpha < 1$, the expected number of probes in a successful search is at most $\dfrac{1}{\alpha} \ln \dfrac{1}{1-\alpha}$.

- We search for a key k.
- Assume that k was $i + 1$st element inserted into the table.
- For the insertion, by Corollary 11.7, the expected number of probes was $\dfrac{1}{1 - i/m} = \dfrac{m}{m-i}$.
- X is a random variable returning the number of probes in the successful search for k.
- $E[X] = \sum_{i=1}^{n-1} \dfrac{1}{n} \dfrac{m}{m-i} = \dfrac{1}{n} \sum_{i=1}^{n-1} \dfrac{m}{m-i} = \dfrac{m}{n} \sum_{i=1}^{n-1} \dfrac{1}{m-i} =$
  $\dfrac{1}{\alpha} \sum_{k=m-n+1}^{m} \dfrac{1}{k} \leq \dfrac{1}{\alpha} \ln(\dfrac{1}{1-\alpha})$

Conclusion

- If a hash table is half full, a successful search will require on average less than 1.387 probes.
- If a hash table is 90% full, a successful search will require on average less than 2.559 probes.