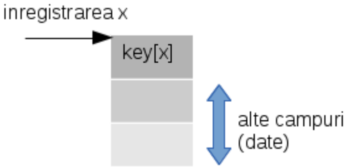
Hash tables

DS 2018/2019

Content

Symbol table

- A symbol table S with n records;
- Each record has associated a (unique) key;
- Operations: $search(S, k)$, $insert(S, x)$, $delete(S, x)$;
- How to organize the data structure S?



The direct-address table

- $U = \{0, 1, ..., m-1\}$ the universe set of keys;
- An array $T[0..m-1]$:

$$T[k] = \begin{cases} x & \text{if } x \in S \text{ and } x.key = k \\ NULL & \text{otherwise.} \end{cases}$$

- Each position (slot) in the array corresponds to a key in the universe U.
- If $|S| = n$, then $n \leq m$.

The direct-address table - Operations

- Operations

  Function *search*(T, k)
  begin
  return $T[k]$
  end

  Procedure *insert*(T, x)
  begin
  $T[x.key] = x$
  end

  Procedure *delete*(T, x)
  begin
  $T[x.key] = NULL$
  end

- The time complexity of operations: $\Theta(1)$

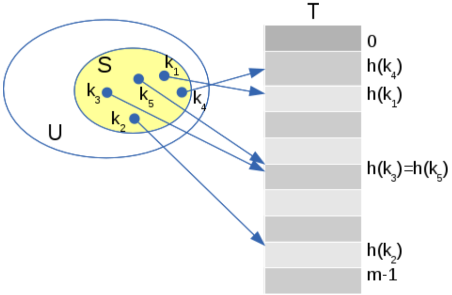
The direct-address table

- The memory space: $\Theta(|U|)$.
- Problems:
  - the keys can be non integers;
  - the domain of keys is very large:
    - numbers on 64 bits (18.446.744.073.709.551.616 of different keys)
    - strings;
  - the set of stored keys is very small relative to U.
- Solution: hash tables
  - a generalization of the concept of direct-address table;
  - an efficient data structure for implementing dictionaries.

Content

Hash tables

- It uses a hash function h to associate to the keys of universe U a value from the set $\{0, 1, \cdots, m-1\}$.

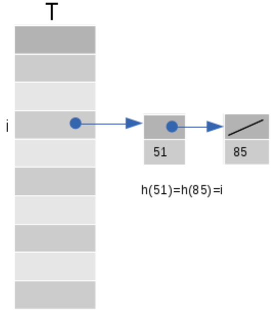


- An element with the key k has associated the position $h(k)$ in the table T.
- The hash function reduces the domains of indices and implicitly the size of the stored array.
- Collision: $\exists\ x_1, x_2 \in S$ such that $h(x_1.key) = h(x_2.key)$

Content

Collision resolution by chaining

- The records that have associated the same slot will be stored in a linked list. T becomes an array of pointers.



- A simple solution, but it requires additional memory space.

- Worst case scenario: all keys have associated the same slot
  - the access time: $\Theta(n)$.

Chaining – Operations

Function *search*(T, k)
begin
    search for the element with the key k in the list $T[h(k)]$
end

Procedure *insert*(T, x)
begin
    insert x at the beginning of the list $T[h(x.cheie)]$
end

Procedure *delete*(T, x)
begin
    delete x from the list $T[h(x.cheie)]$
end

Chaining – Complexity analysis

- *Search*:
  The worst case complexity depends on the length of the list.

- *Insertion*:
  The worst case complexity: $O(1)$.

- *Deletion*:
  $O(1)$ for doubly linked lists; for simple linked lists, first search x and store his predecessor in order to restore the link.

Chaining – The average case complexity analysis

- The assumption of simple uniform hashing: each key $k \in U$ has an equal probability to be stored in any location in the table T and independently of the locations of other keys.

- The load factor of the table T is

$$\alpha = n/m,$$

  where n is the number of keys ($|S|$), and m is the number of locations (the size of the array T).

- The time to compute the hash function is $\Theta(1)$.

Chaining – The average case complexity analysis

Theorem:

In a hash table in which collisions are resolved by chaining, an **unsuccessful** *search takes* **average case** *time* $\Theta(1+\alpha)$*, under the assumption of simple uniform hashing.*

Theorem:

In a hash table in which collisions are resolved by chaining, a **successful** *search takes* **average case** *time* $\Theta(1+\alpha)$*, under the assumption of simple uniform hashing.*

Corollary:

If the number of slots is at least proportional to the number of elements $(n = O(m)$ *or, equivalently,* $\alpha = O(1))$*, then the search operation has the complexity, in* **average**, $O(1)$.

Content

The hash function

- *Deterministic*: for a key k, the function must provide always the same value $h(k)$.
- *Random*: aims to minimize collisions.
- A good hash function distributes the keys uniformly in the locations of the table.
- The assumption of simple uniform hashing is difficult to guarantee, but there are heuristic techniques that work well in practice (as long as their shortcomings can be avoided).

Hash functions – The division method

$$\boxed{h(k) = k \mod m}$$

- Assume that all keys are natural numbers.
    - if the keys are not natural numbers, then we must find a way to interpret them as natural numbers;
    - *Example:* suppose an identifier of the form $(112, 116)$; in the base 128, it becomes $(112 \times 128) + 116 = 14452$.
- Do not choose for m a value with a small divisor d. The predominance of congruent modulo d keys can affect negatively the uniformity.
- If $m = 2^r$, then the value of the function depends only on the lasts r bits of k.
    - *Example:* $k = 1011000111011010$ and $r = 6 \mapsto h(k) = 011010$.
- Choose m a prime number that is not close to a power of 2 or 10.

Hash functions – The multiplication method

$$\boxed{h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor}$$

- $A \in (0, 1)$ is a constant.
- The value of m is not critical (usually a power of 2).

$$\boxed{h(k) = (kA \mod 2^w) rsh(w - r)}$$

- $m = 2^r$, (machine with words of w–bits).
- A is an odd integer in the range $(2^{w-1}, 2^w)$.
- *rsh* is the bitwise right shift operator.

Hash functions – The multiplication method

- *Example:* $m = 2^3$ and words on $w = 7$ bits.

$$\begin{array}{lr} & 1011001 = A \\ \times & 1101011 = k \\ \hline & 1001010\underbrace{011}_{h(k)}0011 \end{array}$$

- Do not choose A too close to 2^{w-1} or 2^w.
- Knuth: $A = (\sqrt{5} - 1)/2$.
- The multiplication modulo 2^w is faster compared to the division; the operator *rsh* is fast.

Hash functions – Universal hashing

$$\boxed{h(k) = [(ak + b) \mod p] \mod m}$$

- p a prime number with $p > |U|$;
- a, b random numbers in $\{0, ..., p-1\}$.

- $k_1 \neq k_2$, $Pr_{a,b}\{h(k_1) = h(k_2)\} = 1/m$.

Content

Solving collisions by open addressing

- All items are stored inside the table T; no additional memory space is used, except for the hash table.
- The insert function examines the table until an empty location is found.
- The hash function depends on the key as well as on the number of examination:

$$\boxed{h : U \times \{0, 1, ..., m-1\} \mapsto \{0, 1, ..., m-1\}}$$

- The sequence of examinations (**probe sequence**) $< h(k, 0), h(k, 1), \cdots, h(k, m-1) >$ must be a permutation of $\{0, 1, .., m-1\}$.
- Disadvantages: the table can be filled; the deletion may become difficult.

Open addressing – Operations

Function *search*(T, k)
begin
    $i \leftarrow 0$
    repeat
        $j \leftarrow h(k, i)$
        if $T[j] == k$ **then**
            return j
        else
            $i \leftarrow i + 1$
    until $T[j] == NULL\ OR\ i == m$;
    return $NULL$
end

Open addressing – Operations

Function *insert(T, k)*
begin
    $i \leftarrow 0$
    repeat
        $j \leftarrow h(k, i)$
        if $T[j] == NULL$ **then**
            $T[j] \leftarrow k$
            return j
        else
            $i \leftarrow i + 1$
    until $i == m$;
    return -1
end

Open addressing – Strategies for probing

Linear probing:

$$\boxed{h(k,i) = (h'(k) + i) \mod m}$$

- $h'(k)$ an ordinary hash function.
- For a key k, the probe sequence is

$$h'(k),\ h'(k)+1,\ h'(k)+2, \ldots,\ m-1,\ 0,\ 1, \ldots,\ h'(k)-1.$$

- Advantage: a simple method.
- Disadvantage: *primary clustering* – long strings of occupied slots build up, increasing the average search time.

Open addressing – Strategies for probing

Quadratic probing:

$$\boxed{h(k,i) = (h'(k) + c_1 i + c_2 i^2) \mod m}$$

- $h'(k)$ an ordinary hash function.
- For a key k, the first location probed is $h'(k)$, and the next positions probed are offset by amounts that depend in a quadratic manner on the previously probed position.
- Disadvantage: *secondary clustering* – if two keys have the same initial probe position, then their probe sequences are the same.
- It works better than linear probing.

Open addressing – Strategies for probing

Double hashing:

$$\boxed{h(k,i) = (h_1(k) + ih_2(k)) \mod m}$$

- $h_1(k)$ si $h_2(k)$ two ordinary hash functions.
- For a key k, the first location probed is $h_1(k)$, and the next positions probed are offset by $h_2(k) \mod m$ towards the previous position.
- This method has in general good results, assuming that $h_2(k)$ is relatively prime to m. One way to accomplish this is to consider m a power of 2 and to choose $h_2(k)$ such that to result only odd numbers.

Open addressing – Complexity analysis

The uniform hashing assumption: each key is equally likely to have any of the $m!$ permutations as probe sequence.

Theorem:

Given an open-address hash table with load factor $\alpha < 1$, assuming uniform hashing, the average number of probes is at most

- $\frac{1}{1-\alpha}$ *in an unsuccessful search, and*
- $\frac{1}{\alpha} ln \frac{1}{1-\alpha}$ *in a successful search.*

Corollary:

If α is constant, then accessing an open-address hash table requires in average a constant time, $\Theta(1)$.

Applications

- Hash tables are used for: database indexing, compilers - symbol tables, caches, etc.
- Applications of hash functions: CRC, Cryptographic hash functions, etc.