Data Structures and Algorithms: Lecture 11

Barbara Morawska

October 11, 2018

Insert, Search, Delete

We focus on dictionary operations only. How to implement them most efficiently?

Hash table

- searching in worst case $\Theta(n)$, but
- in average case $O(1)$.

Effective data structure for dictionaries!

Generalization of a concept of an array

In an array an index allows access element of the array in $O(1)$.
Idea: to make an index connected closely to the element stored at this index.

Recall:

- COUNTING SORT, where the array C has indexes equal to values of elements in A
- BUCKET SORT, where in the array of buckets B an element was stored at the index computed from its value.

Direct-address tables

- Assume that our set of elements has values (keys) only from some interval from 0 to $n-1$.
- Assume that out set contains only elements with different values (keys).
- Let U be a universe of keys $U = [0, .., n-1]$
- We can implement the set as a direct-address table $T[0..n-1]$.
- Each slot in T corresponds to one key in U.
- Slot k contains a pointer to an element in S with the key k or
- if there is no element in S with key k, $T[k] = NIL$.

Direct-addressing procedures

Procedure DIRECT-ADDRESS-SEARCH(T, k)

return $T[k]$

Procedure DIRECT-ADDRESS-INSERT(T, x)

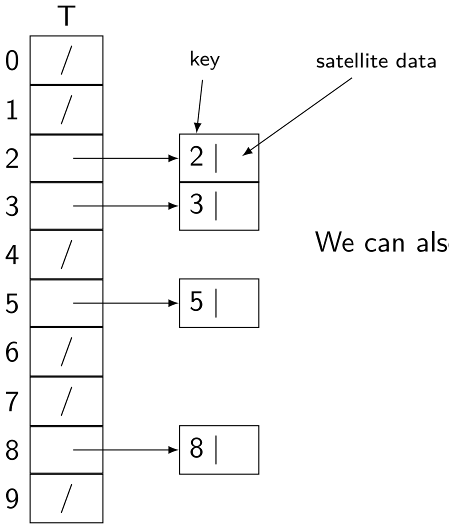
$T[x.key] = x$

Procedure DIRECT-ADDRESS-DELETE(T, x)

$T[x.key] = NIL$

Running time for each is $O(1)$.

Example



We can also store elements of the set in slots of T.

Hash tables

- Direct addressing is bad when U (the universe of keys) is large
- T is a table of size $|U|$
- Actual keys may use a small portion of T: waste of space

hash function

- In the direct addressing an element with key k is stored in the slot k
- In a hash table it is stored in the slot $h(k)$, where h is a hash function.

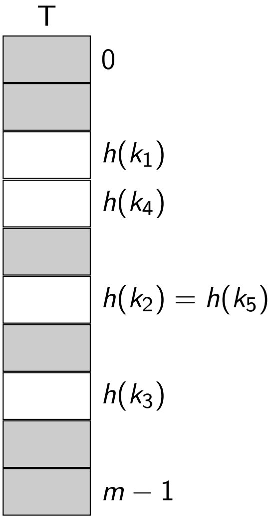
$$h : U \to \{0, 1, \ldots, m - 1\}$$

  where $m < |U|$.

We say:

- an element with key k hashes to the slot $h(k)$
- $h(k)$ is the hash value of k

Example



Problem: k_2 and k_5 collide!

Collisions

Collision:

two keys hash to the same slot (value)

How to avoid collisions?

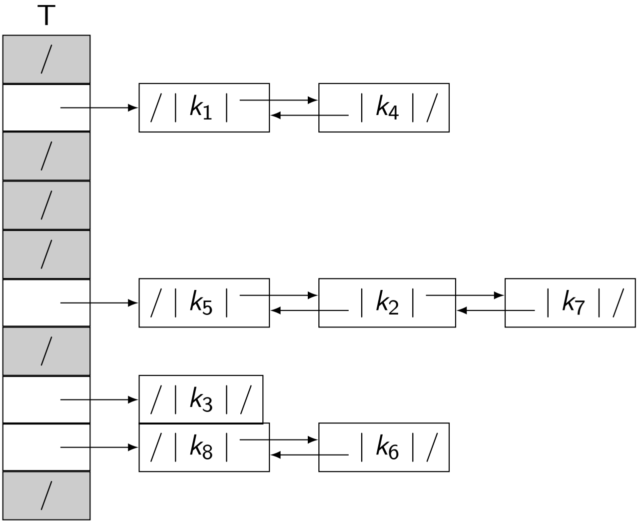
- choose hash function h is a smart way:
  - let it appear random
    (for a given k, $h(k)$ will be most probably unique...)

But since $|U| > m$, avoiding collisions all the time is impossible.

Collision resolution by chaining

Chaining

Put the elements that collide into a linked list.



Dictionary operations

Procedure CHAINED-HASH-INSERT(T, x)

insert x at the head of list $T[h(x.key)]$

Procedure CHAINED-HASH-SEARCH(T, k)

search for element with key k in the list $T[h(k)]$

Procedure CHAINED-HASH-DELETE(T, x)

delete x from the list $T[h(x.key)]$

Running time

- Assume $O(1)$ time for computing $h(k)$
- Worst case for CHAINED-HASH-INSERT: $O(1)$
- Worst case for CHAINED-HASH-SEARCH: proportionate to the length of a list
- Worst case for CHAINED-HASH-DELETE:
  - if lists are doubly-linked, then $O(1)$.
  - if the lists are singly-linked, then proportionate to the length of a list

  Notice: deleting and searching have the same asymptotic time in the last case.

Analysis of search time

We use the notion of load factor α for hash table T with m slots and n elements stored.

$$\alpha = n/m$$

Intuitively, α is the average number of elements stored in a chain in one slot.

- Worst case for searching is when n keys are hashed into the same slot. Then search is $\Theta(n)$
- We want to know the average performance.
- How the hash function h distributes objects in T
- Assume that every element is equally likely to be hashed into any of m slots of T
- This means we assume simple uniform hashing

Analysis of search time

- Let the length of a list in $T[j]$ be denoted by n_j $(j = 0, 1, \ldots, m-1)$
- We can treat n_j as a random variable, i.e. a function from an event of hashing n elements to m slots of T, to the number of elements in the slot $T[j]$.
- What is the expected value of n_j?
- In order to find this, we define an indicator variable for the event that the ith element out of n elements hashes into $T[j]$
- $X_i = I\{i\text{th element hashes into slot } T[j]\}$
- Then $n_j = X_1 + X_2 + \cdots + X_n$
- Then expectation of n_j:

$$E[n_j] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$$

- We know that $E[X_i] = Pr\{i\text{th element is hashed to the slot } T[j]\} = 1/m$
- Hence $E[n_j] = \sum^{n} E[X_i] = \sum^{n} 1/m = n/m = \alpha$

Analysis of search time

Assume simple uniform hashing.

Theorem

In a hash table where collisions are resolved by chaining, an unsuccessful search takes $\Theta(1+\alpha)$ *time.*

Proof.

- We search for an element with the key k.
- The expected time to search unsuccessfully for this element is equal to the expected time to search till the end of list in $T[h(k)]$
- This is equal to $E[n_{h(k)}] = \alpha$
- 1 in $\Theta(1+\alpha)$ is time needed to compute the hash value of k.

$\square$

Analysis of search time

Theorem

In a hash table where collisions are resolved by chaining , a successful search takes average time $\Theta(1 + \alpha)$

Proof is in the book...

Conclusion of this analysis

- If $n = O(m)$, then the load factor $\alpha = n/m = O(m)/m = O(1)$
- Hence then search takes constant time on average.
- Then all dictionary operations take constant time on average.

Hash functions

- Hashing by division
- Hashing by multiplication
- Universal hashing: choosing hashing function randomly.

Hash functions – conditions

What is a good hashing function?

- Every key should be equally likely to hash to any slot (m slots, n elements),
- independently from other elements.
- Hash function should be independent of any patterns in the keys
- For example: key "pt" should not be in the same slot as "pts"
- Close values should possibly be hashed far apart.

Example

Let keys be random numbers independently and uniformly distributed in [0..1]. Then the following function satisfies the conditions:

$$h(k) = \lfloor km \rfloor$$

Interpreting keys as natural numbers

- We will always assume that the keys are natural numbers.
- What is keys are not such numbers?
- We will always assume that there is encoding into natural numbers.
- For example a key is "pt"
- We can interpret it as a pair of ASCII codes for letters: $p = 112, t = 116$
- This pair of numbers $(112, 116)$ can be seen as the number based 128,
- and hence is equal to $112 \cdot 128 + 116 = 14452$ decimal number.
- Usually natural numbers are used as binary numbers

The division method

Map k to the remainder of k/m

$$h(k) = k \bmod m$$

Example

- Let $m = 12$ and $k = 100$.
- Then $h(k) = 100 \bmod 12 = 4$

How to choose m?

- Do not choose $m = 2^p$
- Recall dividing by 2 shifts a binary number to the right.
- Dividing by 2 p times, shifts the number p places to the right.
- Hence the reminder will be p last digits of the key.
- Hence use only if all p low-order bits of keys are equally likely.

Example

For example:

- We want to define a hash table with chaining to store $n = 2000$ character strings.
- One character has 8 bits.
- We allow to examine 3 elements on a successful search.

Then what should be m?

- $2000/3 = 666.6...$ slots?
- Choose $m = 701$, because it is prime close to $2000/3$ but not near any power of 2
- The hash function is then $h(k) = k \bmod 701$

Multiplication method

Hash function:

- Choose a constant A, $0 < A < 1$
- For a key k, multiply k by A, kA
- Take only fractional part of the result , $kA \bmod 1$
- Multiply m by this value.

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

Multiplication method

How to choose m for a hash table?

- Choose $m = 2^p$, because it makes computations easy to implement.
- Suppose word size of a computer is ω bits and k fits in this size.
- Choose s an integer, $0 < s < 2^\omega$
- Set $A = s/2^\omega$
- Multiply k by $A \cdot 2^\omega$ (Just not to bother with fractional parts)
- We obtain 2ω bit number. Take ω lower bits (this was to be the fractional part)
- The p most significant bits is the value for the hash function for this key.

Example

- Suppose $k = 123456$ and $p = 14$, $m = 2^{14} = 16384$ and $\omega = 32$
- We choose $A = 2654435769/2^{32}$
- $k \cdot s = (76300 \cdot 2^{32}) + 17612864$
- The value of the hash function is then 14 most significant bits of 17612864
- Hence $h(k) = 67$

Example

- Let $k = 1101011$, $\omega = 7$ bits and $m = 2^3$
- Choose $A = .1011001$. If we multiply it by 2^7 we get 1011001 (shift to left 7 places).
- Multiply k by this number to get: 1001010 0110011
  Notice we get $2 \cdot 7$ bits number.
- Take the second part (this is the fractional part in the original definition of the hash function): 0110011
- Now the value of the function is: 3 first most significant bits.
  $h(k) = 011$