بسم الله الرحمن الرحیم

ساختمان‌های داده

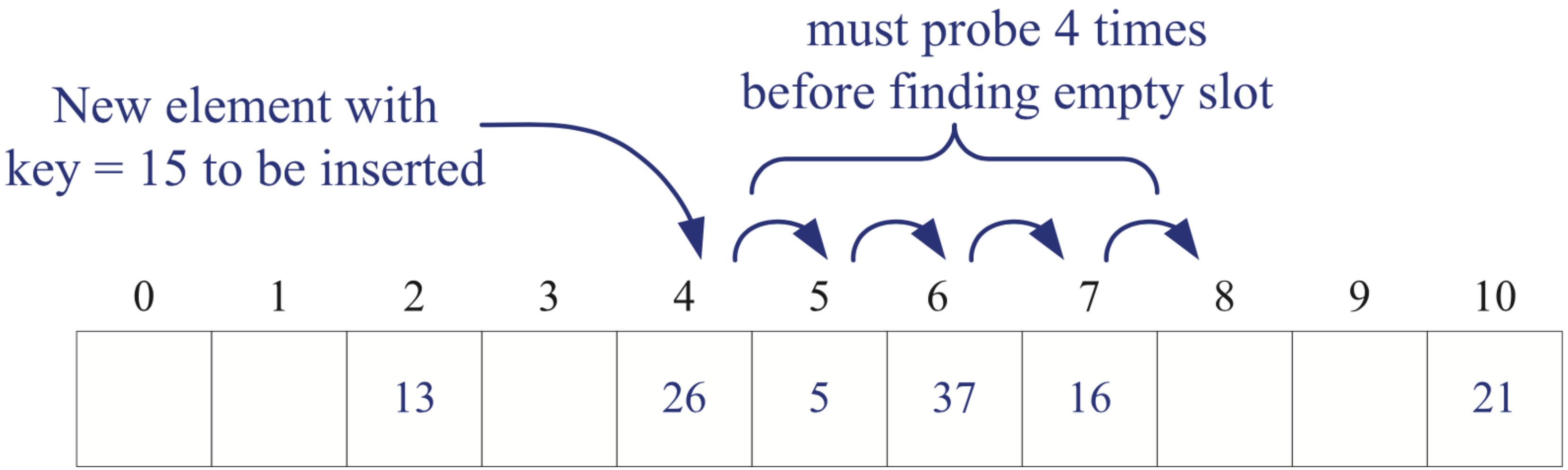
جلسه ۲۳

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

Open Addressing: Linear Probing

- Open addressing: the colliding item is placed in a different cell of the table
- Linear probing: handles collisions by placing the colliding item in the next (circularly) available table cell
- Each table cell inspected is referred to as a “probe”
- Colliding items lump together, causing future collisions to cause a longer sequence of probes

Open Addressing: Linear Probing



$h(x) = x \bmod 11$

Linear Probing: Example

- Example:
  - $h(x) = x \bmod 13$
  - Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order

0	1	2	3	4	5	6	7	8	9	10	11	12

⇩

		41			18	44	59	32	22	31	73	
0	1	2	3	4	5	6	7	8	9	10	11	12

Search with Linear Probing

- Consider a hash table ***A*** that uses linear probing
- find(***k***)
  - We start at cell ***h***(***k***)
  - We probe consecutive locations until one of the following occurs
    - An item with key ***k*** is found, or
    - An empty cell is found, or
    - ***N*** cells have been unsuccessfully probed

Algorithm *find*(k)
    $i \leftarrow h(k)$
    $p \leftarrow 0$
    repeat
        $c \leftarrow A[i]$
        if $c = \varnothing$
            return *null*
        else if $c.key() = k$
            return $c.value()$
        else
            $i \leftarrow (i + 1) \bmod N$
            $p \leftarrow p + 1$
    until $p = N$
    return *null*

Updates with Linear Probing

IUT-ECE

- To handle insertions and deletions, we introduce a special marker, called ***AVAILABLE***, which replaces deleted elements
  - Avoids a lot of shift operations

- erase(***k***)
  - We search for an entry with key ***k***
  - If such an entry (***k, o***) is found, we replace it with the special item ***AVAILABLE*** and we return element ***o***
  - Else, we return ***null***

- put(***k, o***)
  - We throw an exception if the table is full
  - We start at cell ***h***(***k***)
  - We probe consecutive cells until one of the following occurs
    - A cell ***i*** is found that is either empty or stores ***AVAILABLE***, or
    - ***N*** cells have been unsuccessfully probed
  - We store (***k, o***) in cell ***i***

Theorems

IUT-ECE

Theorem 11.6

Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming independent uniform permutation hashing and no deletions.

$$1/(1-\alpha) = 1 + \alpha + \alpha^2 + \alpha^3 + \cdots$$

$\alpha = 0.5 \quad \rightarrow \quad 2$

$\alpha = 0.9 \quad \rightarrow \quad 10$

Theorems

Theorem 11.8

Given an open-address hash table with load factor $\alpha < 1$, the expected number of probes in a successful search is at most

$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha} ,$$

assuming independent uniform permutation hashing with no deletions and assuming that each key in the table is equally likely to be searched for.

$\alpha = 0.5 \quad \rightarrow \quad 1.4$

$\alpha = 0.9 \quad \rightarrow \quad 2.5$

Other Issues

- Search with Linear Probing
  - Clustering problem
- Other open addressing method
  - Quadratic Probing, Double Hashing (the details in the book)

Probing

- Quadratic Probing, Double Hashing (the details in the book)

$$i = h(k),$$

$$A[(i + f(j)) \bmod N], \text{ for } j = 0, 1, 2, \ldots, \text{ where } f(j) = j$$

$$\text{where } f(j) = j^2,$$

***secondary clustering*,**

may not find an empty slot

Probing

- Quadratic Probing, Double Hashing (the details in the book)

$$i = h(k),$$

$$A[(i + f(j)) \bmod N], \text{ for } j = 0, 1, 2, \ldots, \text{ where } f(j) = j$$

$$\text{where } f(j) = j^2,$$

$$\text{where } f(j) = j \cdot h'(k).$$

Other Issues

- The load factor $\boldsymbol{a} = \boldsymbol{n}/\boldsymbol{N}$ affects the performance of a hash table

- Keeping the load factor below a certain threshold is vital
  - Open addressing (requires $\boldsymbol{a} < 0.5$)
  - Separate-chaining (requires $\boldsymbol{a} < 0.9$)
  - Resize the hash table, i.e., rehashing a new table

Performance of Hashing

IUT-ECE

- In the worst case, searches, insertions and removals on a hash table take $O(n)$ time
- The worst case occurs when all the keys inserted into the map collide
- The load factor $a = n/N$
- Assuming that the hash values are like random numbers, it can be shown that the expected number of probes for an insertion with open addressing is
  $1 / (1 - a)$
- But, when well designed, the expected running time of all the MAP ADT operations in a hash table is $O(1)$
- In practice, hashing is very fast provided the load factor is not close to 100%

پیاده‌سازی Map

- برای یک map با n جفت (key, value)

	`insert`	`find`	`delete`	`find max/min`
Unsorted linked-list	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Unsorted array	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Sorted linked list	$O(n)$	$O(n)$	$O(n)$	$O(1)$
Sorted array	$O(n)$	$O(logn)$	$O(n)$	$O(1)$
AVL/RB tree	$O(logn)$	$O(logn)$	$O(logn)$	$O(logn)$
Hash table	$O(1)^*$	$O(1)^*$	$O(1)^*$	$O(n)$

پیاده‌سازی Map

- Skip list

Operation	*Time*
size, empty	$O(1)$
firstEntry, lastEntry	$O(1)$
find, insert, erase	$O(\log n)$ (expected)
ceilingEntry, floorEntry, lowerEntry, higherEntry	$O(\log n)$ (expected)

Table 9.3: Performance of an ordered map implemented with a skip list.