~~D~~SA (Fall 2022) 25-01-23

Important operations on Data Structures

We can do the following using the search algorithm

* Determine whether a particular item exists in a list/DS.
* If the is organized (e.g. sorted), find the location in the ~~location~~ list for new insertion.
* Find the location of an item to be deleted.

→ Performance of search algorithm is crucial.

(A) Sequential search

Compare searchItem with 1st element in the list & continue till list end or ~~at local~~ stop when item is found.

* works the same for array-based or linked lists.

```
int loc; bool found=false;
seqSearch(item)
for(loc=0; loc<length; loc++)
    if (list[loc]==item)
    { found = true;
      break;
    }
if (found)
    return loc
else
    return -1;
```

* Statements before/after loop executed once hence computer time negligible.
* Statements in for loop are repeated ~~until~~ several times.
* key/imp repetition is comparison
  list[loc] == item
* No. of comparison done remains the same in any implementation/computer.
* If the list size is 'n'
  - if search item not in list
    n comparisons (unsuccessful case)
  - if search item in list
    - if search item is at 1st location
      1 comparison (successful scenario)
      Best case.
    - if search item in the last of list
      n comparisons (successful case)
      Worst Case

Not likely to occur all the time.

Look for average # of comparisons, in the successful case. For this

1- Consider all possible cases.
2- Find the number of comparisons in each case.
3- Add the number of comparisons and divide by the number of cases.

* if search item at 1^{st} location, comparisons $= 1$

" " 2^{nd} " " $= 2$

⋮

" " n^{th} " " $= n$

$$\text{Average \# of comparisons} = \frac{1+2+\cdots+n}{n}$$

we know that $1+2+\cdots+n = \frac{n(n+1)}{2}$

$$S = 1+2+3+\cdots+(n-2)+(n-1)+n \quad -(1)$$

write in Reverse Order

$$S = n+(n-1)+(n-2)+\cdots+(3)+(2)+1 \quad -(2)$$

Add (1) & (2)

$$2S = (n+1)+(n-1+2)+(n-2+3) + \cdots + (n-2+3)+(n-1+2)+(n+1)$$

$$2S = \underbrace{(n+1)+(n+1)+(n+1)+\cdots+(n-1)+(n-1)}_{\text{total } n \text{ terms.}}$$

$$2S = n(n+1)$$

$$\boxed{S = \frac{n(n+1)}{2}}$$

$$\text{Average \# of comparisons} = \frac{n(n+1)}{2} \times \frac{1}{n} = \frac{n+1}{2}$$

for large $n \longrightarrow O(n)$

~~Itm~~ Sequential Search Not ~~ef~~efficient for large lists.

SA (Fall 2022) 25-01-23

Ordered Lists

A list is ordered if its elements are ordered according to some criteria. Usually ascending order. Most operations performed on unordered lists are same for ordered lists.

Binary Search

Performed on ordered list. Divide & conquer strategy.

* First search item is compared with middle element of the lists. If found search terminates.
* If SI (searchItem) < middle element search space is Left half of middle element otherwise right half.
* Search space shrinks to half at every comparison.

	first=0	1	2	3	4	5	6	7	8	9	10	11=last=length-1
list	4	8	19	25	34	39	45	48	66	75	89	95
						↑			↑			

List length = 12 list[5] $\frac{first+last}{2} = \frac{11+0}{2} = 5 = mid$

search Item ≠ list[5], SI > list[5]

first = mid+1 = 6, last = length-1 = 11

else first = ~~mid~~ 0, last = mid-1 same as 64

$\frac{6+11}{2} = 8$

Page 504 & 505 of Text Book

Fall 2022)

25-01-23

5

* Suppose ~~list~~ L is a sorted list of size 1024
  - Searching sequentially $\frac{1024+1}{2} = 512$ comparisons.
  - $2^{10} = 1024$ $2^k = \text{Length}$ then at most $k+1$ comparisons. i.e 11 comparisons at most,
  - Binary search makes 2 comparisons, so at most 22 comparisons.

Now if the list size $n = 2^m \Rightarrow m = \log_2 n$

* total $m+1$ iterations, every iteration of binary search involves 2 key comparisons

i.e. $2(m+1) = 2(\log_2 n + 1) = O(\log_2 n)$

Hashing Search algorithm, also requires data to be specially organized.

* In hashing, data is organized with the help of a table, called hash table. HT
HT is stored in an array.
* To determine item key, say X in Table. we apply a function h, called the hash function to the key X, i.e. h(X)
* h(X) is typically an arithmetic function & gives the address the address of the item.
* Suppose that the size of the hash table is m. Then $0 \le h(X) < m$.

* Thus to determine whether the item with key X is in the table, we look for entry HT[h(X)] in hashtable.

* As address of item is computed with the help of a function, it follows that the items are stored in no particular order.

Imp Questions

* How do we choose a hash function?
* How do we organize data with the help of the hash table?

Page 510, 511

Collision

Resolve Collision

1. open Hashing (closed addressing) → solution → chaining Method or separate chaining
2. closed Hashing (open addressing)
   - Linear Probing
   - Quadratic Probing
   - Double Hashing

A (Fall 2022)

Chaining

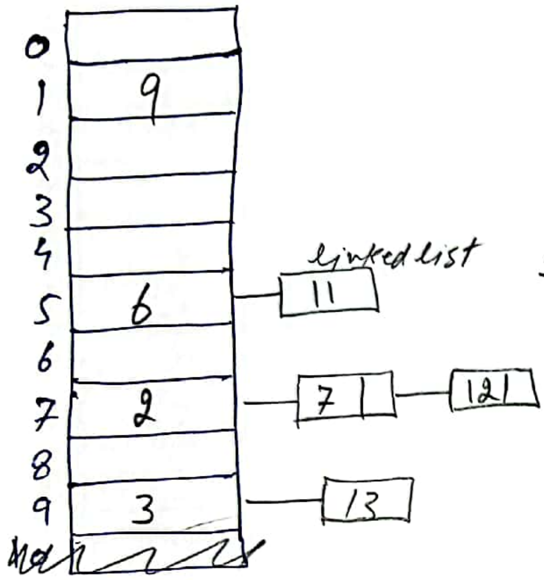
key values 3, 2, 9, 6, 11, 13, 7, 12

$h(k) = 2k+3$

$m = 10$

if this not given use directly → $h(k_i) = k_i \% m$ & Use division method & closed addressing (open hashing)/chaining

o/w ↓



Key	Location
3	$(2\times\underset{k}{3}+3)\ \%\ 10 = 9$
2	$(2\times2+3)\ \%\ 10 = 7$
9	$(2\times9+3)\ \%\ 10 = 21$
6	$(2\times6+3)\ \%\ 10 = 5$
11	$(2\times11+3)\ \%\ 10 = 5$ already occupied
13	$2\times13+3\ \%\ 10 = 9$ using chaining.
7	$2\times7+3\ \%\ 10 = 7$
12	$2\times12+3\ \%\ 10 = 7$

i.e $h(k_i)\ \%\ m$ or $(2k_i+3)\ \%\ m$

~~Closed Hashing~~ Open addressing

Linear probing. (By default for open addressing)

0	13
1	9
2	7
3	
4	
5	6
6	5
7	2
8	7
9	3

Probe Next free

Probe Next

Key	Location	Probes
3	9	1
2	7	1
9	1	1
6	5	1
11	5	2
13	9	2
7	7	2
12	7	6

Total Probes = 16

Order of table

13, 9, 7, _, _, 6, 5, 2, 7, 3

Insert k_i at 1^{st} free location from $(u+i)\ \%\ m$ where $i = 0$ to $m-1$ & u = location calculated.

A (Fall 2022)

25-01-23

(8)

Same problem with Quadratic Probing.

Key values 3, 2, 9, 6, 11, 13, 7, 12

$h(k) = 2k + 3$, $m = 10$

$h(k_i) = (2k_i + 3) \% 10$

Division Method + Quadratic Probing to store values.

0	13
1	9
2	
3	12
4	
5	6
6	11
7	2
8	7
9	3

Key	Location (u)	Probe
3	9	1
2	7	1
9	1	1
6	5	1
11	5 → 6	2
13	9 → 0	2
7	7 → 8	2
12	7 → 3	5

Total Probes = 15

$(5+1^2)\%10 = 6$

$(9+1^2)\%10 = 0$

$(7+1^2)\%10 = 8$

$(7+1)\%10 = 8, (7+2^2)\%10 = 01$

$7+3^2\%10 = 6, (7+4^2)\%10 = 3$

* Insert k_i at 1^{st} free location from $(u+i^2)\% m$ for $i = 1$ to $m-1$

Order of table = 13, 9, –, 12, –, 6, 11, 2, 7, 3

Same Problem with Double Hashing

$h_1(k) = 2k+3 = u$, $h_2(k) = 3k+1 = v$

Insert at $(u+v*i)\%m$

0	13
1	9
2	
3	11
4	12
5	6
6	
7	2
8	
9	3

Key	Location (u)	Location (v)	Probes
3	9		1
2	7		1
9	1		1
6	5		1
11	5 → 3	$(3 \times 11+1)\%10 = 4$	3
13	9	$(3 \times 13+1)\%10 = 0$	
7	7	$(3 \times 7+1)\%10 = 2$	
12	7 → 4	$(3 \times 12+1)\%10 = 7$	2

$5+4\%10 = 9$, $5+8\%10 = 3$

$(9+0*i)\%10 = 0$ 13 Not insertable

Check all 7 Not //