DATA STRUCTURE AND ALGORITHM

Dr. Khine Thin Zar
Professor
Computer Engineering and Information Technology Dept.
Yangon Technological University

Lecture 9

Searching

Outlines of Class (Lecture 9)

- Introduction
- Searching Unsorted and Sorted Arrays
- Sequential Search/ Linear Search Algorithm
- Binary Search Algorithm
- Quadratic Binary Search Algorithm
- Self-Organizing Lists
- Bit Vectors for Representing Sets
- Hashing

Introduction

- Searching
  - is the most frequently performed of all computing tasks.
  - is an attempt to find the record within a collection of records that has a particular key value
    - ✓ a collection L of n records of the form $(k_1; I_1); (k_2; I_2);...; (k_n; I_n)$
    - ✓ I_j is information associated with key k_j from record j for $1 \leq j \leq n$.
  - the search problem is to locate a record $(k_j ; I_j)$ in L such that $k_j = K$ (if one exists).
  - Searching is a systematic method for locating the record (or records) with key value $k_j = K$.

Introduction (Cont.)

- Successful Search
  - a record with key $k_j = K$ is found
- Unsuccessful Search
  - no record with $k_j = K$ is found (no such record exists)
- Exact-match query
  - is a search for the record whose key value matches a specified key value.
- Range query
  - is a search for all records whose key value falls within a specified range of key values.

Introduction (Cont.)

- Categories of search algorithms :
  - ✓ Sequential and list methods
  - ✓ Direct access by key value (hashing)
  - ✓ Tree indexing methods

Searching Unsorted and Sorted Arrays

- Sequential Search/ Linear Search Algorithm
- Jump Search Algorithm
- Exponential Search Algorithm
- Binary Search Algorithm
- Quadratic Binary Search (QBS) Algorithm

Searching Unsorted and Sorted Arrays (Cont.)

- Sequential search on an unsorted list requires θ(n) time in the worst case.
- How many comparisons does linear search do on average?
- A major consideration is whether K is in list L at all.
- K is in one of positions 0 to n-1 in L (each position having its own probability)
- K is not in L at all.
- The probability that K is not in L as:

$$\mathbf{P}(K \notin \mathbf{L}) = 1 - \sum_{i=1}^{n} \mathbf{P}(K = \mathbf{L}[i])$$

Where P(x) is the probability of event x

Searching Unsorted and Sorted Arrays (Cont.)

- p_i be the probability that K is in position i of L (indexed from 0 to n-1).
- For any position i in the list, we must look at i + 1 records to reach it.
- the cost when K is in position i is i + 1. When K is not in L, sequential search will require n comparisons.
- p_n be the probability that K is not in L.
- The average cost T(n):

$$\mathbf{T}(n) = np_n + \sum_{i=0}^{n-1}(i+1)p_i.$$

Searching Unsorted and Sorted Arrays (Cont.)

- Assume all the p_i's are equal (except p_0)

$$\begin{aligned}
\mathbf{T}(n) &= p_n n + \sum_{i=0}^{n-1} (i+1)p \\
&= p_n n + p \sum_{i=1}^{n} i \\
&= p_n n + p \frac{n(n+1)}{2} \\
&= p_n n + \frac{1-p_n}{n} \frac{n(n+1)}{2} \\
&= \frac{n+1+p_n(n-1)}{2}
\end{aligned}$$

- Depending on the value of p_n $\frac{n+1}{2} \leq \mathbf{T}(n) \leq n.$

Searching Unsorted and Sorted Arrays (Cont.)

- For large collections of records that are searched repeatedly, sequential search is unacceptably slow.
- One way to reduce search time is to preprocess the records by sorting them.
- Jump search algorithm
  - ✓ What is the right amount to jump?
  - ✓ For some value j, we check every j'th element in L, that is, check elements L[j], L[2j], and so on.
  - ✓ So long as K is greater than the values we are checking, we continue on.
  - ✓ If we reach the value in L greater than K, we do a linear search on the piece of length j-1that we know brackets K if it is in the list.
  - ✓ If we define m such that $mj \leq n < (m + 1)j$, then the total cost of this algorithm is at most m + j -1 3-way comparisons.

Sequential Search/ Linear Search Algorithm

- Starts at the first record and moves through each record until a match is found, or not found.
- Easy to write and efficient for short lists
- Does not require sorted data
- A major consideration is whether K is in list L at all.
- K is in one of positions 0 to n-1 in L or K is not in L at all.
- A simple approach is to do sequential/ linear search, i.e
  - ✓ Start from the leftmost element of arr[ ] and one by one compare x with each element of arr[ ]
  - ✓ If x matches with an element, return the index.
  - ✓ If x doesn't match with any of elements, return -1.

Analysis of Sequential Search

The list of items is not ordered;

- Best case: find the item in the first place, at the beginning of the list. need only one comparison.
- Worst case: not discover the item until the very last comparison, the n^{th} comparison.
- Average case: find the item about halfway into the list.
- The complexity of the sequential search: O(n).

Case	Best Case	Worst Case	Average Case
item is present	1	n	$\frac{n}{2}$
item is not present	n	n	n

Analysis of Sequential Search (Cont.)

If the items were ordered in some way;

- Best case: discover that the item is not in the list by looking at only one item.
- Worst case: not discover the item until the very last comparison
- Average case: know after looking through only n/2 items.
- The complexity of the sequential search: O(n).

Case	Best Case	Worst Case	Average Case
item is present	1	n	$\frac{n}{2}$
item not present	1	n	$\frac{n}{2}$

Sequential Search Algorithm (Cont.)

- For large collections of records that are searched repeatedly, sequential search is unacceptably slow.
- One way to reduce search time is to preprocess the records by sorting them.
- Given a sorted array, an obvious improvement is to test if the current element in L is greater than K.
- If it is, then we know that K cannot appear later in the array, and we can quit the search early.
- If we look first at position 1 in sorted array L and find that K is bigger, then we rule out position 0 as well as position 1.
- Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster searching comparison to Linear search.

Binary Search Algorithm

- Searching a sorted array by repeatedly dividing the search interval in half.
- Start with an interval covering the whole array.
- If the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.
- If we know nothing about the distribution of key values, then binary search is the best algorithm available for searching a sorted array.

Dictionary Search or interpolation Search

- The "computed" binary search is called a dictionary search or interpolation search, where additional information about the data is used to achieve better time complexity.
- Binary Search goes to middle element to check.
- Interpolation search may go to different locations according the value of key being searched.
- For example if the value of key is closer to the last element, interpolation search is likely to start search toward the end side.
- In a dictionary search, we search L at a position p that is appropriate to the value of K as follows.

$$p = \frac{K - \mathbf{L}[1]}{\mathbf{L}[n] - \mathbf{L}[1]}$$

- This equation is computing the position of K as a fraction of the distance between the smallest and largest key values.

Quadratic Binary Search Algorithm

- A variation on dictionary search is known as Quadratic Binary Search (QBS)
- In Quadratic Algorithm, first calculate the middle element,1/4th element and 3/4th element .
- Compare the key with the item in the middle ,1/4th and 3/4th positions of the array.

Comparison Between Binary and Quadratic Binary Search Algorithm

- Is QBS better than binary search?
- Theoretically yes,
- First we compare, the running time : lg lg n (for QBS) to lg n(for BS).

n	$\lg n$	$\lg \lg n$	Factor Difference
16	4	2	2
256	8	3	2.7
2^{16}	16	4	4
2^{32}	32	5	6.4

Comparison Between Binary and Quadratic Binary Search Algorithm (Cont.)

- let us look and check the actual number of comparisons used.
- For binary search, log n -1 total comparisons are required.
- Quadratic binary search requires about 2.4 lg lg n comparis

n	$\lg n - 1$	$2.4 \lg \lg n$	Factor Difference
16	3	4.8	worse
256	7	7.2	$\approx$ same
$64K$	15	9.6	1.6
2^{32}	31	12	2.6

Self-Organizing Lists

- Assume that know for each key k_i, the probability p_i that the record with key k_i will be requested.
- Assume also that the list is ordered.
- Search in the list will be done sequentially, beginning with the first position.
- Over the course of many searches, the expected number of comparisons required for one search is:

$$\overline{C}_n = 1p_0 + 2p_1 + \ldots + np_{n-1}.$$

  - ✓ The cost to access the record in L[0] is 1 (because one key value is looked at), and the probability of this occurring is p_0.
  - ✓ The cost to access the record in L[1] is 2 (because we must look at the first and the second records' key values), with probability p1, and so on.

Self-Organizing Lists (Cont.)

- As an example of applying self-organizing lists, consider an algorithm for compressing and transmitting messages. The list is self-organized by the move-to-front rule. Transmission is in the form of words and numbers, by the following rules:
  - ✓ If the word has been seen before, transmit the current position of the word in the list. Move the word to the front of the list.
  - ✓ If the word is seen for the first time, transmit the word. Place the word at the front of the list.
- Both the sender and the receiver keep track of the position of words in the list in the same way (using the move-to-front rule), so they agree on the meaning of the numbers that encode repeated occurrences of words.

  "The car on the left hit the car I left."
- The entire transmission would be:

  "The car on 3 left hit 3 5 I 5."
- Ziv-Lempel coding
  - is a class of coding algorithms commonly used in file compression utilities.

Bit Vectors for Representing Sets

- Determining whether a value is a member of a particular set is a special case of searching for keys in a sequence of records.
- The set using a bit array with a bit position allocated for each potential member.
- Those members actually in the set store a value of 1 in their corresponding bit; those members not in the set store a value of 0 in their corresponding bit.
- This representation scheme is called a bit vector or a bitmap.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0

Figure: The bit array for the set of primes in the range 0 to 15. The bit at position i is set to 1 if and only if i is prime.

- to compute the set of numbers between 0 and 15 that are both prime and odd numbers

0011010100010100 & 0101010101010101 : -------- ?

Hashing

- Hashing
  - ✓ The process of finding a record using some computation to map its key value to a position in the array.
- Hash function
  - ✓ The function that maps key values to positions is called a hash function (h).
- Hash Table
  - ✓ The array that holds the records (HT).
- Slot
  - ✓ A position in the hash table
  - ✓ The number of slots in hash table HT will be denoted by the variable M, with slots numbered from 0 to M-1

Hashing (Cont.)

- Applications
  - ✓ Is not suitable for applications where multiple records with the same key value are permitted.
  - ✓ Is not suitable for answering range queries.
  - ✓ Is not suitable for finding the record with the minimum or maximum key value.
  - ✓ Is most appropriate for answering the question (only exact-match queries).
  - ✓ Is suitable for both in-memory and disk-based searching.
  - ✓ Is suitable for organizing large databases stored on disk.

Hashing (Cont.)

- Given a hash function h and two keys k_1 and k_2, if $h(k_1) = \beta = h(k_2)$
  - ✓ where β is a slot in the table, then we say that k_1 and k_2 have a collision at slot under hash function h.
- Finding a record with key value K in a database organized by hashing follows a two-step procedure:

  1. Compute the table location h(K).

  2. Starting with slot h(K), locate the record containing key K using (if necessary) a collision resolution policy.

- Choosing a hash function
  - ✓ Main objectives
    - ✓ Choose an easy to compute hash function
    - ✓ Minimize number of collisions

Searching vs. Hashing

- Searching methods: key comparisons
  - ✓ Time complexity: O(size) or O(log n)
- Hashing methods: hash functions
  - ✓ Expected time: O(1)

Hash Table

- A hash table is a data structure that is used to store keys/value pairs.
- An array of fixed size
- Uses a hash function to compute an index into an array in which an element will be inserted or searched.
- Mapping (hash function) h from key to index

Hash Table Operations

- Insert
- Delete
- Search

Hash Function

- If key range too large, use hash table with fewer buckets and a hash function which maps multiple keys to same bucket:
  - $h(k_1) = \beta = h(k_2)$: k_1 and k_2 have collision at slot β
- Popular hash functions: hashing by division

  $h(k) = k\%D,$

  where D number of buckets in hash table
- Example: hash table with 12 buckets

  $h(k) = k\%12$

  $80 \rightarrow 6$ ($80\%12= 6$), $50 \rightarrow 2$, $64 \rightarrow 4$

  $62 \rightarrow 2$ collision!
  - ✓ If the keys are strings, convert it into a numeric value.

  Could use String keys each ASCII character equals some unique integer

  "label" = 108+97 + 98 + 101+108 == 512

Open Hashing (separate chaining)

- One of the most commonly used collision resolution techniques.
- Implement using linked lists.
- To store an element in the hash table you must insert it into a specific linked list. If there is any collision (i.e. two different elements have same hash value) then store both the elements in the same linked list.
  - ✓ Collisions are stored outside the table (open hashing)
  - ✓ Data organized in linked lists
  - ✓ Hash table: array of pointers to the linked lists
  - ✓ The simplest form of open hashing defines each slot in the hash table to be the head of a linked list.

Open Hashing (separate chaining) (Cont.)

- Better space utilization for large items.
- Simple collision handling: searching linked list.
- Overflow: we can store more items than the hash table size.
- Deletion is quick and easy: deletion from the linked list.

Open Hashing (separate chaining) (Cont.)

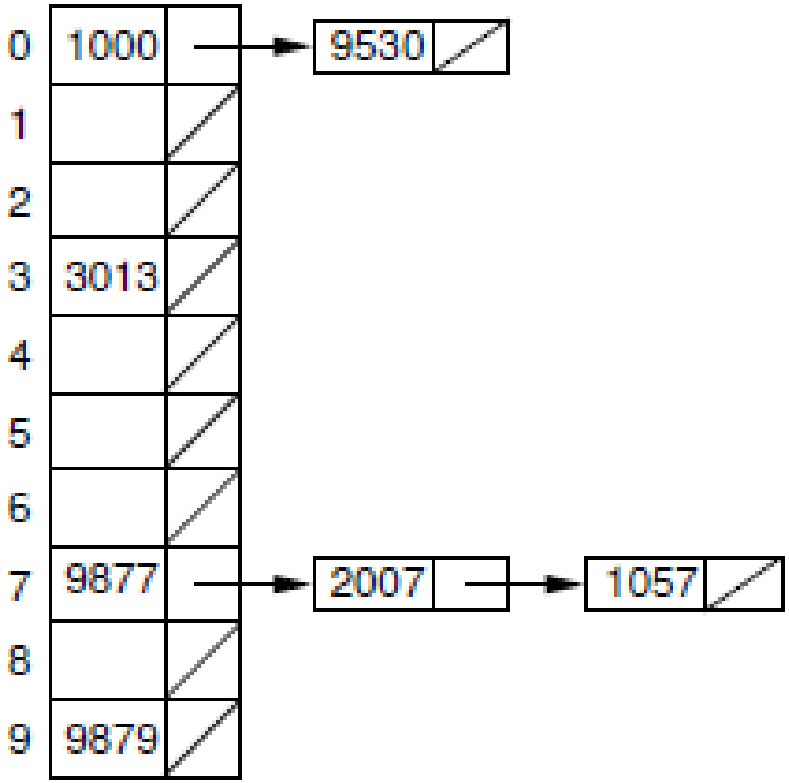

Figure: An illustration of open hashing for seven numbers stored in a ten-slot hash table using the hash function h(K) = K mod 10. The numbers are inserted in the order 9877, 2007, 1000, 9530, 3013, 9879, and 1057. Two of the values hash to slot 0, one value hashes to slot 2, three of the values hash to slot 7, and one value hashes to slot 9.

Close Hashing (open addressing)

- Closed hashing stores all records directly in the hash table. Each record R with key value kR has a home position that is h(kR), the slot computed by the hash function.
- One implementation for closed hashing groups hash table slots into buckets. The M slots of the hash table are divided into B buckets.
- The hash function assigns each record to the first slot within one of the buckets.
- If this slot is already occupied, then the bucket slots are searched sequentially until an open slot is found.
- If a bucket is entirely full, then the record is stored in an overflow bucket of infinite capacity at the end of the table.

Close Hashing (open addressing) (Cont.)

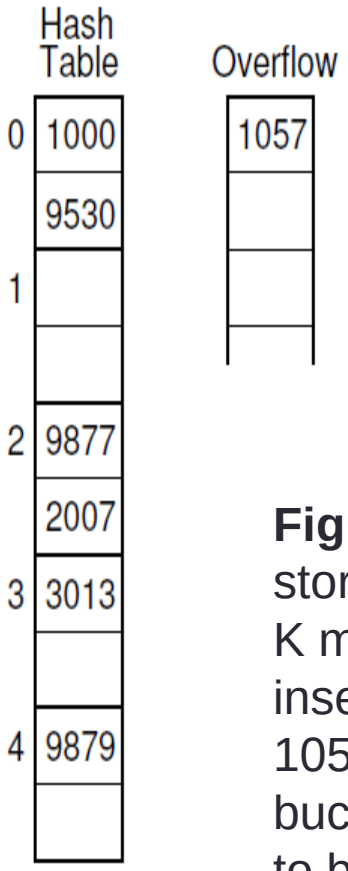

Figure: An illustration of bucket hashing for seven numbers stored in a five bucket hash table using the hash function h(K) = K mod 5. Each bucket contains two slots. The numbers are inserted in the order 9877, 2007, 1000, 9530, 3013, 9879, and 1057. Two of the values hash to bucket 0, three values hash to bucket 2, one value hashes to bucket 3, and one value hashes to bucket 4. Because bucket 2 cannot hold three values, the third one ends up in the overflow bucket.

Next Week Lecture (Week 10)

Lecture 10: Indexing

Thank you!