COM 201 – Data Structures and Algorithms

Linked Lists

Assist. Prof. Özge ÖZTİMUR KARADAĞ

Department of Computer Engineering – ALKÜ

Alanya

Last Week

- Linear / Nonlinear Data Structures
- Array, Pointer
- Sorting
  - Bubble Sort
- Searching
  - Linear Search
  - Binary Search

Linked Lists

- List: a linear collection of data items.

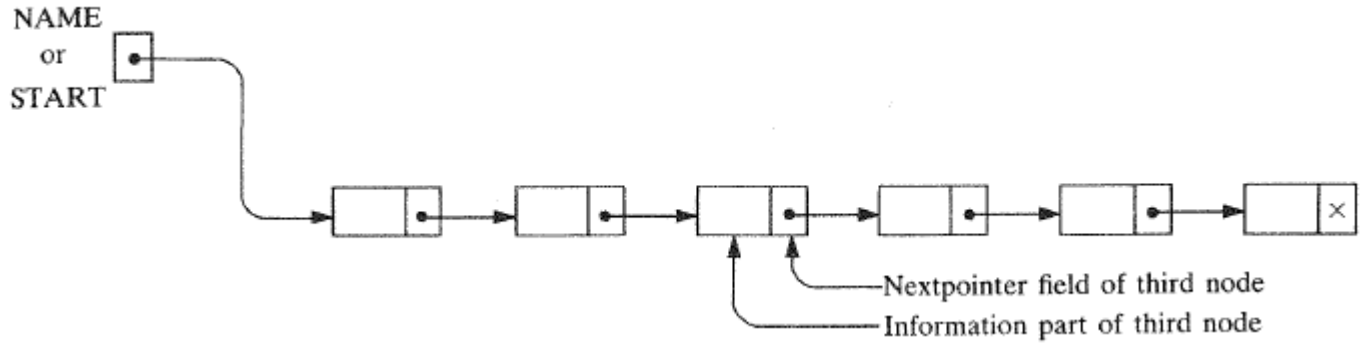
- Previously we stored linear data in array.
  - What is the advantage?
    - Easy to compute the address of an element in an array.
  - What are the disadvantages?
    - Insertion/deletion is relatively expensive
    - Occupies a block of memory space, not easy to expand the array such as by doubling or tripling when additional space is required. For this reason arrays are called as dense list and are said to be static data structures.

Linked Lists

- Each element in the list contain a field, called link or pointer, which contains the address of the next element in the list.
- Successive elements in the list need not occupy adjacent space in memory.
- Easier to insert and delete elements in the list.

Linked Lists

- A linked-list or one-way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers.
- Each node is divided into two parts:
  - First part contains the information of the element
  - Second part, link field or nextpointer field, contains the address of the next node in the list.
  - Ex. Linked list with 6 nodes:



START or NAME is the list pointer which contains the address of the first node in the list.

Representation of Linked Lists in Memory

- Let LIST be a linked list.
- LIST requires two linear arrays.
  - INFO[K]: information part
  - LINK[K]: nextpointer field
    of a node of LIST.
  - A variable name, such as START, which contains the location of the beginning of the list.
  - A nextpointer sentinel- denoted by NULL- which indicates the end of the list, which is usually set as NULL=0.

Representation of Linked List in Memory

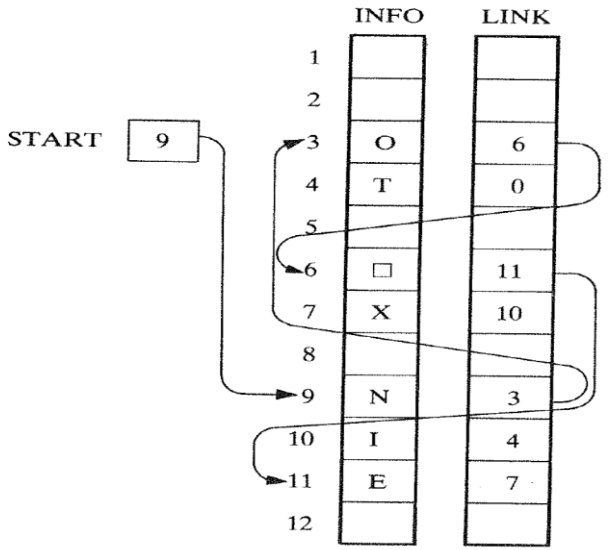
- The nodes of a list need not occupy adjacent elements in the arrays INFO and LINK.

Example: Figure pictures a linked-list in memory, where each node of the list contains a single character. We can obtain the string as follows:

START=9, INFO[9]=N is the first character

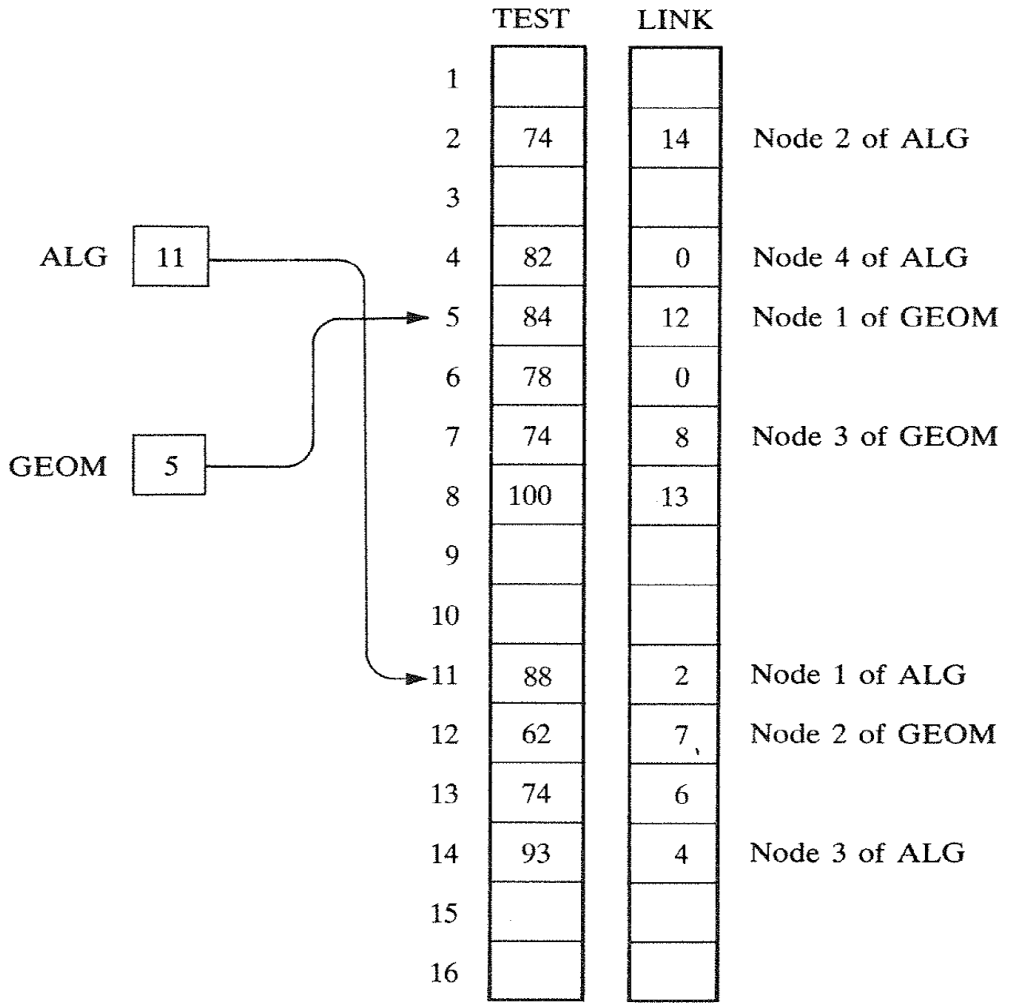
LINK[9]=3, INFO[3]=O is the second character

…



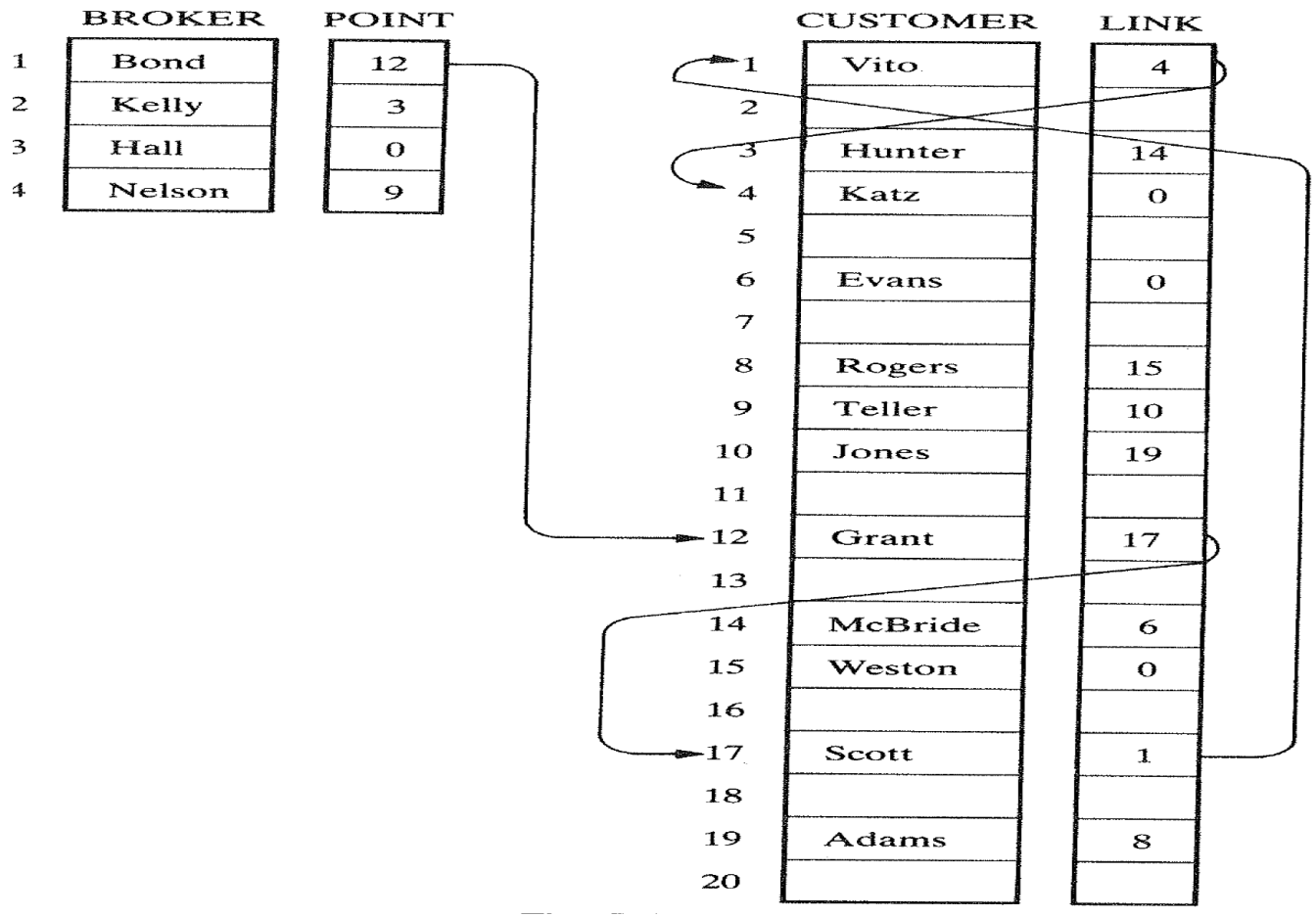
Representation of Linked Lists in Memory

- Ex: Two lists of test scores, ALG and GEOM, are maintained in memory where the nodes of both lists are stored in the arrays TEST and LINK. Names of the lists are also used as the list pointer variables. ALG contains 11, the location of the first node, and GEOM contains 5, the location of its first node.
  - ALG consists of the test scores:
    - 88, 74, 93, 82.
  - GEOM consists of the test scores
    - 84,62, 74, 100, 74, 78.



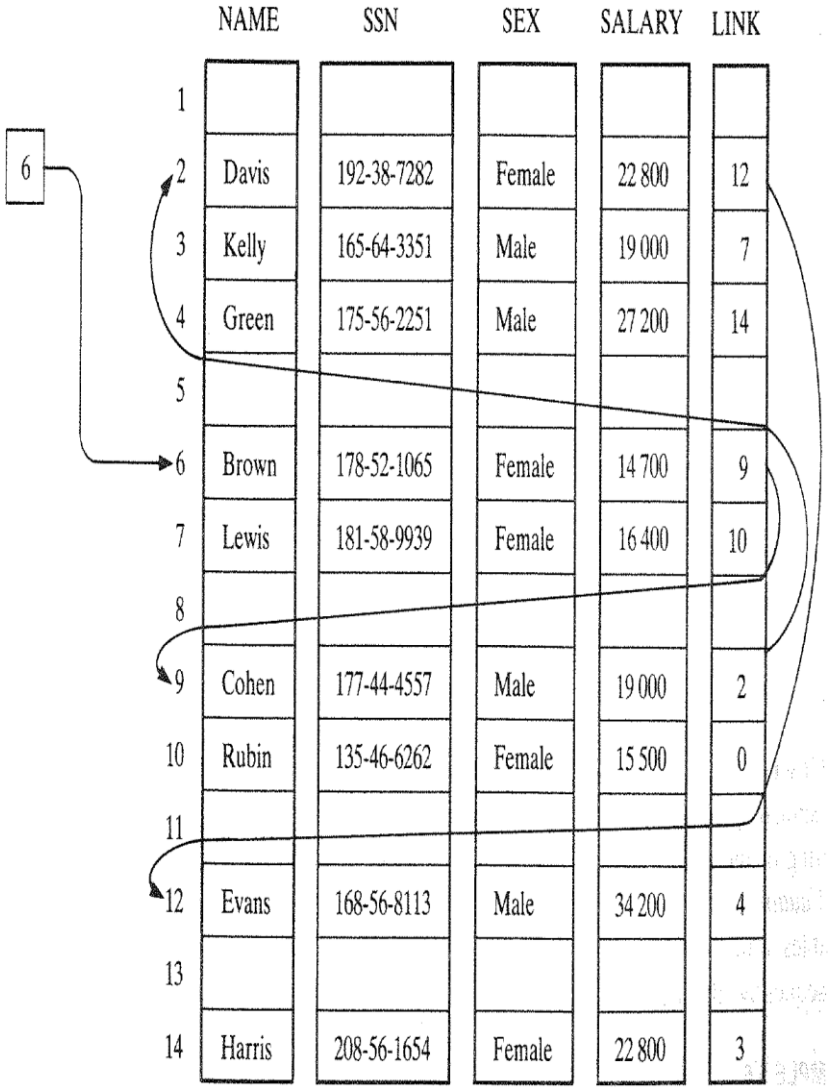
Representation of Linked Lists in Memory

- Suppose a brokerage firm has four brokers and each broker has his own list of customers.
  - Bond's list of customers:
  - Kelly's
  - Nelson's
  - Hall's



Representation of Linked Lists in Memory

- Ex. Personnel file of a small company contains the following data on its nine employees: Name, Social Security Number, Sex, Monthly Salary.
- How to represent this data?
  - Structure or parallel arrays



	NAME	SSN	SEX	SALARY	LINK
1					
2	Davis	192-38-7282	Female	22 800	12
3	Kelly	165-64-3351	Male	19 000	7
4	Green	175-56-2251	Male	27 200	14
5					
6	Brown	178-52-1065	Female	14 700	9
7	Lewis	181-58-9939	Female	16 400	10
8					
9	Cohen	177-44-4557	Male	19 000	2
10	Rubin	135-46-6262	Female	15 500	0
11					
12	Evans	168-56-8113	Male	34 200	4
13					
14	Harris	208-56-1654	Female	22 800	3

6

Traversing A Linked List

- Assume that LIST is a linked list in memory stored in linear arrays INFO and LINK with START pointing to the first element and NULL indicating the end of LIST.
- We want to traverse LIST in order to process each node exactly once.
- The traversing algorithm uses a pointer variable PTR which points to the node that is currently being processed. LINK[PTR] points to the next node to be processed.
- PTR:=LINK[PTR] moves the pointer to the next node in the list

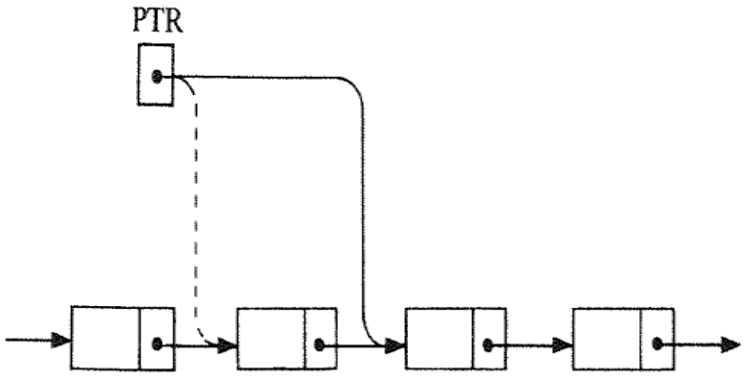


Fig. 5-8 PTR := LINK[PTR].

Traversing A Linked List

- Let LIST be a linked list in memory. Write the algorithm that traverses LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to the node currently being processed.

1. Set PTR := START. [Initializes pointer PTR.]
2. Repeat Steps 3 and 4 while PTR ≠ NULL.
3. Apply PROCESS to INFO[PTR].
4. Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of Step 2 loop.]
5. Exit.

Searching A Linked List

- Let LIST be a linked list in memory. Suppose a specific ITEM of information in given, find the location LOC of the node where ITEM first appears in the LIST.
  - LIST is unsorted
    SEARCH(INFO,LINK,START,ITEM,LOC): List is a linked list in memory. This algorithm finds the location of the node where ITEM first appears in LIST, or sets LOC=NULL

1. Set PTR := START.
2. Repeat Step 3 while PTR ≠ NULL:
3. If ITEM = INFO[PTR], then:
   Set LOC := PTR, and Exit.
   Else:
   Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of If structure.]
   [End of Step 2 loop.]
4. [Search is unsuccessful.] Set LOC := NULL.
5. Exit.

Searching A Linked List

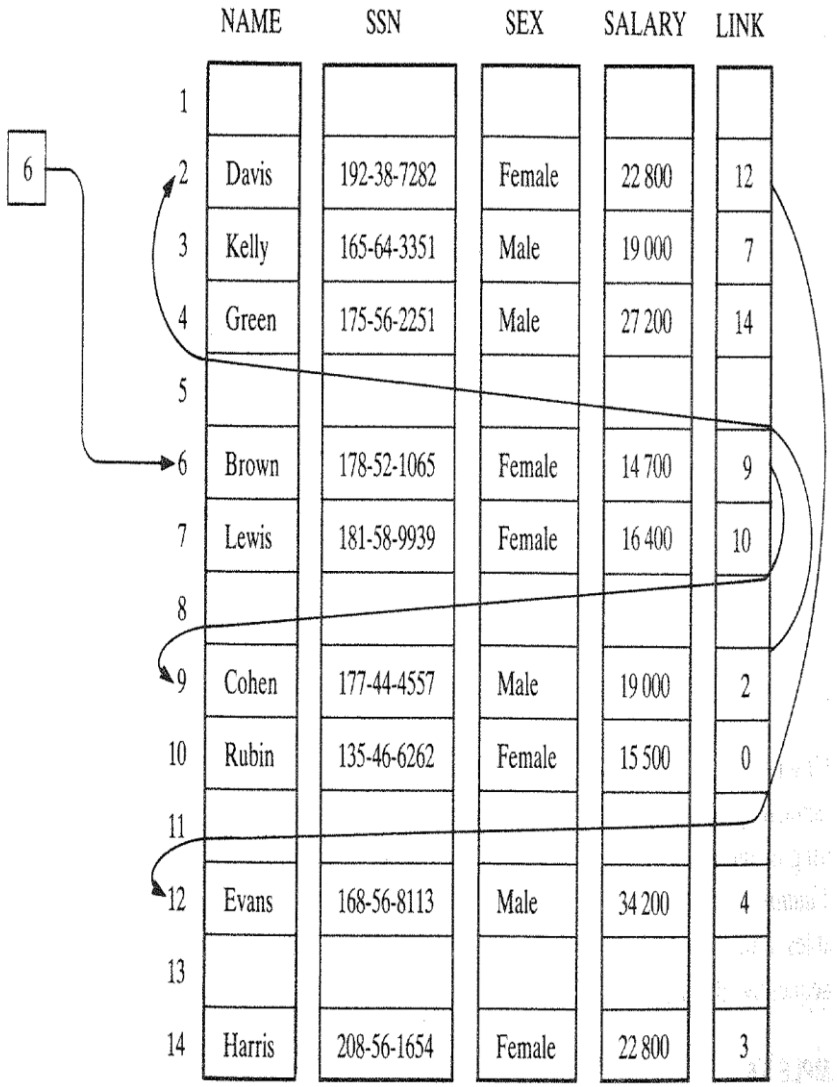
- What is the complexity?
  - Same as linear search
  - Worst case is proportional to the number of elements in LIST
  - Average case is approximately proportional to n/2

```
1. Set PTR := START.
2. Repeat Step 3 while PTR ≠ NULL:
3.     If ITEM = INFO[PTR], then:
           Set LOC := PTR, and Exit.
       Else:
           Set PTR := LINK[PTR]. [PTR now points to the next node.]
       [End of If structure.]
   [End of Step 2 loop.]
4. [Search is unsuccessful.] Set LOC := NULL.
5. Exit.
```

Searching A Linked List

- Example: For the personnel data given, write code to read the social security number NNN of an employee and then gives the employee a 5 percent increase in salary.

1. Read: NNN.
2. Call SEARCH(SSN, LINK, START, NNN, LOC).
3. If LOC ≠ NULL, then:
   Set SALARY[LOC] := SALARY[LOC] + 0.05 * SALARY[LOC],
   Else:
   Write: NNN is not in file.
   [End of If structure.]
4. Return.



Searching A Linked List

- List is sorted: Search for an ITEM in LIST by traversing the list using a pointer variable PTR and comparing ITEM with the contents INFO[PTR] of each node, one by one, of LIST.
- We can stop once ITEM exceeds INFO[PTR]
- SRCHSL(INFO,LINK,START,ITEM,LOC): LIST is a sorted list in memory. This algorithm finds the location LOC of the node where ITEM first appears in LIST, or sets LOC=NULL.

1. Set PTR := START.
2. Repeat Step 3 while PTR ≠ NULL:
3. If ITEM < INFO[PTR], then:
   Set PTR := LINK[PTR]. [PTR now points to next node.]
   Else if ITEM = INFO[PTR], then:
   Set LOC := PTR, and Exit. [Search is successful.]
   Else:
   Set LOC := NULL, and Exit. [ITEM now exceeds INFO[PTR].]
   [End of If structure.]
   [End of Step 2 loop.]
4. Set LOC := NULL.
5. Exit.

Searching A Linked List

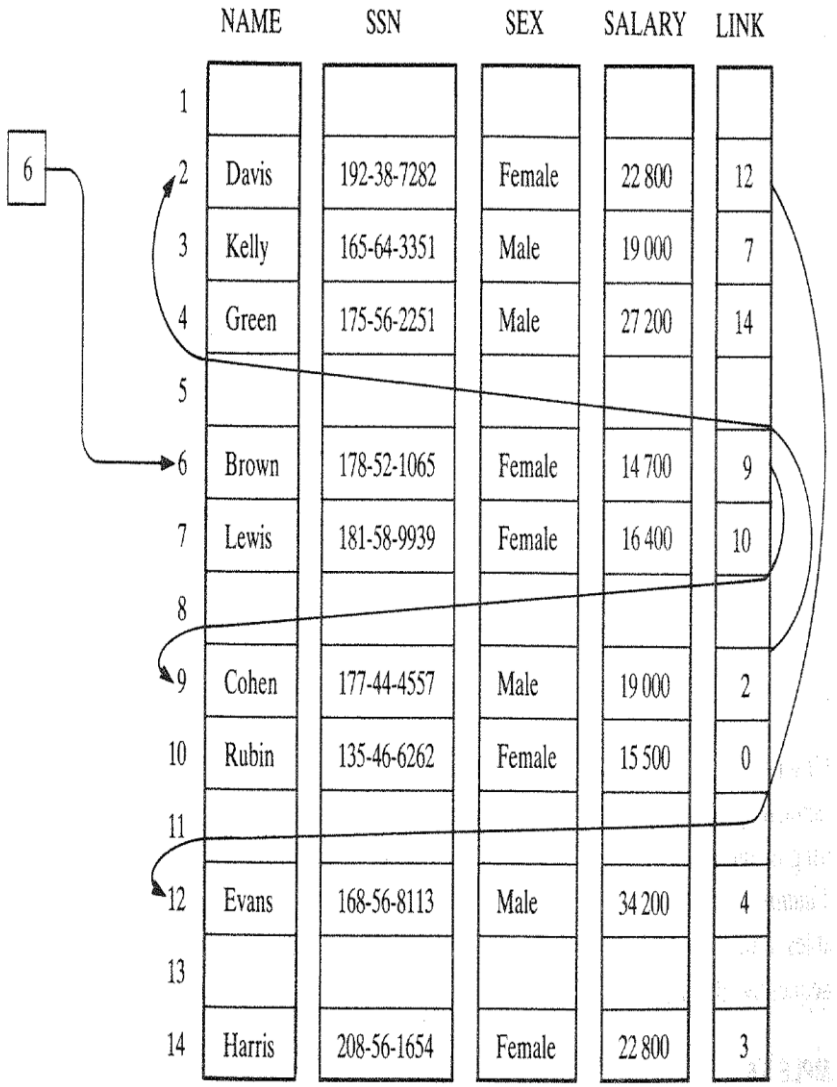
- List is sorted
- What is the complexity?
  - Worst case: proportional to the number of elements in LIST,
  - Average case: proportional to n/2
- Can we apply Binary search on a linked list?

1. Set PTR := START.
2. Repeat Step 3 while PTR ≠ NULL:
3. If ITEM < INFO[PTR], then:
   Set PTR := LINK[PTR]. [PTR now points to next node.]
   Else if ITEM = INFO[PTR], then:
   Set LOC := PTR, and Exit. [Search is successful.]
   Else:
   Set LOC := NULL, and Exit. [ITEM now exceeds INFO[PTR].]
   [End of If structure.]
   [End of Step 2 loop.]
4. Set LOC := NULL.
5. Exit.

Searching A Linked List

- Example: For the personnel data given, write code to read the name EMP of an employee and then gives the employee a 5 percent increase in salary.
  - Since the data is sorted alphabetically, we can use the second search algorithm.

1. Read: EMPNAME.
2. Call SRCHSL(NAME, LINK, START, EMPNAME, LOC).
3. If LOC ≠ NULL, then:
   Set SALARY[LOC] := SALARY[LOC] + 0.05 * SALARY[LOC].
   Else:
   Write: EMPNAME is not in list.
   [End of If structure.]
4. Return.

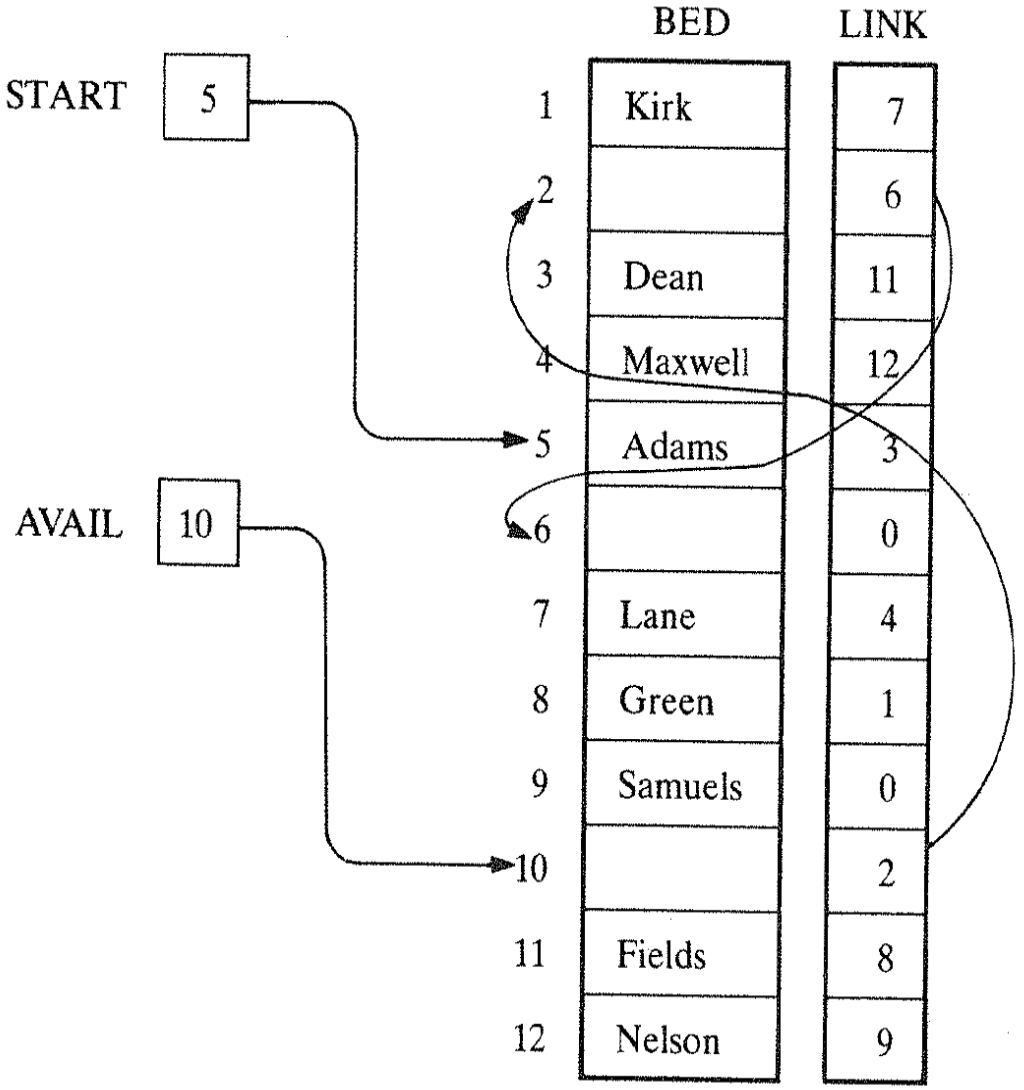


Memory Allocation; Garbage Collection

- Need to keep track of free memory to be used for insertion.
- Together with the linked list in memory, a special list is maintained which consists of unused memory cells. This list which has its own pointer, is called the list of available space or the free storage list or the free pool.
- Suppose our linked lists are implemented by parallel arrays.
- Unused memory cells in the arrays will also be linked together to form a linked list using AVAIL as its list pointer variable.
- LIST(INFO,LINK, START,AVAIL)

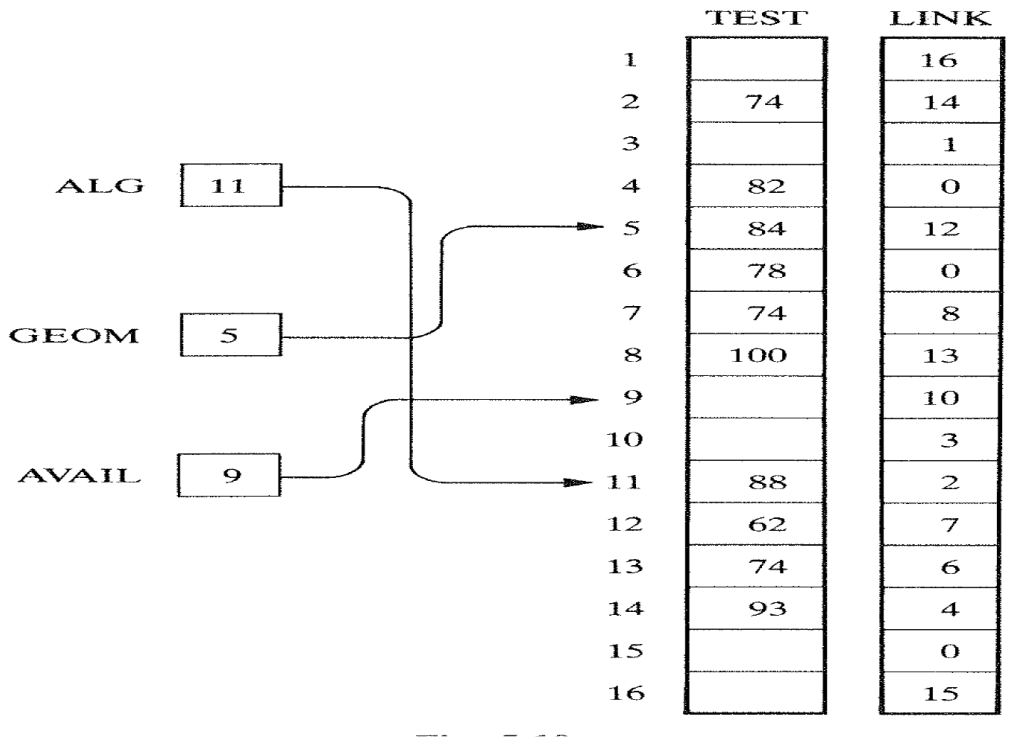
Memory Allocation; Garbage Collection

- Example: List of patients is stored in linear arrays BED and LINK (patient in bed K is assigned to BED[K]). Then the available space in the linear array BED may be linked as follows:
  - BED[10] is the first available bed, BED[2] is the next available bed, and BED[6] is the last available bed.



Memory Allocation; Garbage Collection

- Ex. Both GEOM and ALG linked list can use the AVAIL list.



Memory Allocation; Garbage Collection

- Suppose some memory space becomes reusable because a node is deleted from a list or an entire list is deleted from a program.
- We can immediately insert the space into the free-storage list. –> too time consuming.
- Alternatively, the operating system periodically collect all the deleted space onto the free-storage list. → any technique which does this collection is called garbage collection.
  - First step: run through all lists, tagging those cells which are currently in use,
  - Second step: run through the memory, collect all untagged space onto the free-storage list.
- Garbage collection may take place;
  - when there is only minimum amount of space or no space at all left in the free-storage list,
  - When CPU is idle

Memory Allocation; Garbage Collection

- Overflow: There is an insertion but there is no available space. Program may handle overflow by printing OVERFLOW message.
  - When AVAIL=NULL and there is an insertion → Overflow occurs.

- Underflow: one wants to delete data from a data structure that is empty.
  - When START=NULL and there is a deletion → Underflow occurs.