Question Bank

Subject: Advanced Data Structures

Class: SE Computer

Question1: Write a non recursive pseudo code for post order traversal of binary tree

Answer: Pseudo Code:

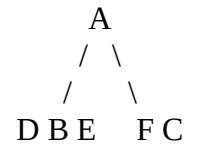
1. Push root into Stack_One.
2. while(Stack_One is not empty)
    1. Pop the node from Stack_One and push it into Stack_Two.
    2. Push the left and right child nodes of popped node into Stack_One.
3. End Loop
4. Pop out all the nodes from Stack_Two and print it.

Question 2: Construct Tree from given Inorder and Preorder traversals

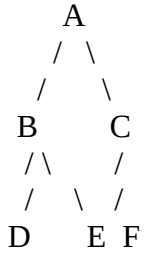
Inorder sequence: D B E A F C

Preorder sequence: A B D E C F

Answer: In a Preorder sequence, leftmost element is the root of the tree. So we know 'A' is root for given sequences. By searching 'A' in Inorder sequence, we can find out all elements on left side of 'A' are in left subtree and elements on right are in right subtree. So we know below structure now.



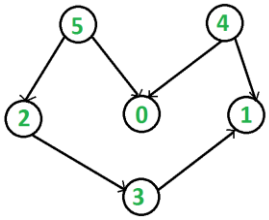
We recursively follow above steps and get the following tree.



Question 3: Write short note on Topological Sort?

Answer: Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

For example, a topological sorting of the following graph is "5 4 2 3 1 0". There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is "4 5 2 3 1 0". The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no in-coming edges).

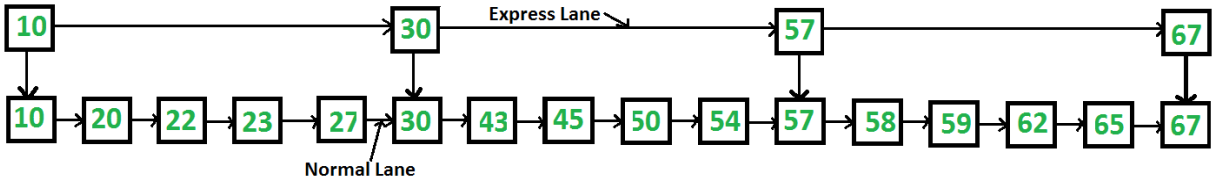


Topological Sorting vs Depth First Traversal (DFS):
In DFS, we print a vertex and then recursively call DFS for its adjacent vertices. In topological sorting, we need to print a vertex before its adjacent vertices. For example, in the given graph, the vertex '5' should be printed before vertex '0', but unlike DFS, the vertex '4' should also be printed before vertex '0'. So Topological sorting is different from DFS. For example, a DFS of the shown graph is "5 2 3 1 0 4", but it is not a topological sorting

Question 4: Write a Short note on Skip List

Answer: Can we search in a sorted linked list in better than O(n) time?
The worst case search time for a sorted linked list is O(n) as we can only linearly traverse the list and cannot skip nodes while searching. For a Balanced Binary Search Tree, we skip almost half of the nodes after one comparison with root. For a sorted array, we have random access and we can apply Binary Search on arrays.
Can we augment sorted linked lists to make the search faster? The answer is Skip List. The idea is simple, we create multiple layers so that we can skip some nodes. See the following example

list with 16 nodes and two layers. The upper layer works as an “express lane” which connects only main outer stations, and the lower layer works as a “normal lane” which connects every station. Suppose we want to search for 50, we start from first node of “express lane” and keep moving on “express lane” till we find a node whose next is greater than 50. Once we find such a node (30 is the node in following example) on “express lane”, we move to “normal lane” using pointer from this node, and linearly search for 50 on “normal lane”. In following example, we start from 30 on “normal lane” and with linear search, we find 50.



What is the time complexity with two layers? The worst case time complexity is number of nodes on “express lane” plus number of nodes in a segment (A segment is number of “normal lane” nodes between two “express lane” nodes) of “normal lane”. So if we have n nodes on “normal lane”, √n (square root of n) nodes on “express lane” and we equally divide the “normal lane”, then there will be √n nodes in every segment of “normal lane” . √n is actually optimal division with two layers. With this arrangement, the number of nodes traversed for a search will be O(√n). Therefore, with O(√n) extra space, we are able to reduce the time complexity to O(√n).

Can we do better?

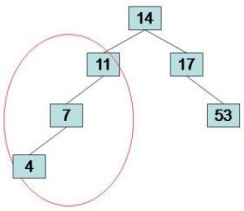
The time complexity of skip lists can be reduced further by adding more layers. In fact, the time complexity of search, insert and delete can become O(Logn) in average case. We will soon be publishing more posts on Skip Lists.

Question 5: Build AVL Tree for following data. show step by step construction.

Answer:

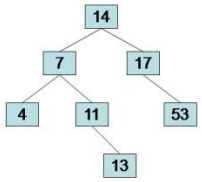
AVL Tree Example:

- **Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree**



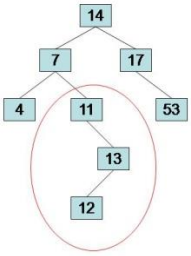
AVL Tree Example:

- **Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree**



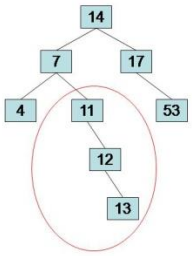
AVL Tree Example:

- **Now insert 12**



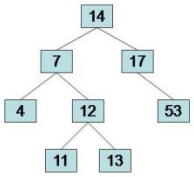
AVL Tree Example:

- **Now insert 12**



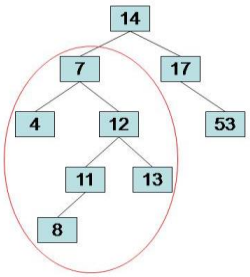
AVL Tree Example:

- **Now the AVL tree is balanced.**



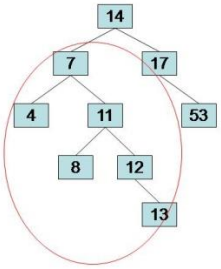
AVL Tree Example:

- **Now insert 8**



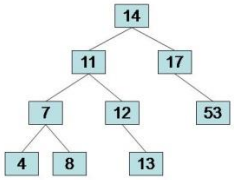
AVL Tree Example:

- **Now insert 8**



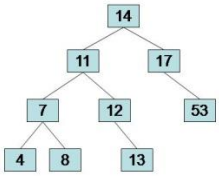
AVL Tree Example:

- **Now the AVL tree is balanced.**



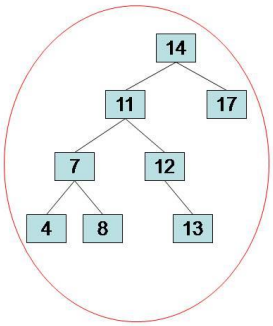
AVL Tree Example:

- **Now remove 53**



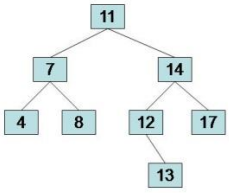
AVL Tree Example:

- **Now remove 53, unbalanced**



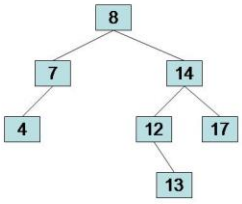
AVL Tree Example:

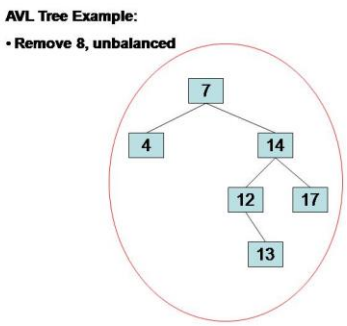
- **Balanced! Remove 11**

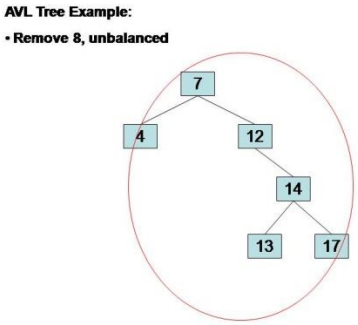


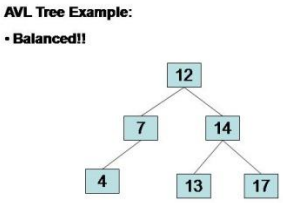
AVL Tree Example:

- **Remove 11, replace it with the largest in its left branch**









Question 6: Explain Linked organization with respect to file Handling

Answer: LINKED ORGANIZATION

- In linked organization, the physical sequence of records is different from the logical sequence of records
- The next logical record is obtained by following a link value from the present record
- Multilist Files
- Coral Rings
- Inverted Files
- Cellular Partitions

Multilist Files:

- To make searching easy, several indexes are maintained as per primary key and secondary keys, one index per key
- The record may be present in different lists as per key
- Consider the following file of office staff

Staff ID	Occupation	Salary	Record
106	Clerk	5000	A
150	Account	4000	B
360	clerk	3000	C
400	Account	3500	D
700	Clerk	2000	E

- We can maintain index on the staff ID
- We can group staff ID with range 101–300, 301–600, 601–900, etc
- Now all the records with staff ID in the same range will be linked together as shown in Figure

Staff range	Link
101-300	
301-600	
601-900	

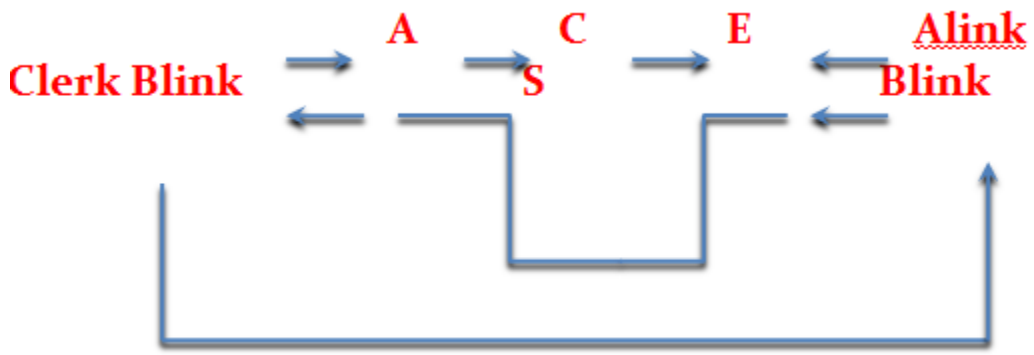
rec A
rec C
rec E

rec B
rec D

Sample multilist file

Coral Rings

- In this doubly linked multilist structure is used. Each list is circular list with headnode



Sample doubly linked list

- Alink' field is used to link all records with same key value
- 'Blink' is used for some records back pointer and for others it is pointer to head node
- Owing to these back pointers, deletion is easy without going to start.

Indexes are maintained as per multilists

Inverted Files

- With the same key value are linked together and links are kept in each record
- But in the inverted files, the link information is kept in the index itself
- The indices for fully inverted file are shown in Figure

106	A
150	B
360	C
400	D
700	E

Accountant	B,D
Clerk	A,C,E

Salary Index	
2000	E
4000	B,C,D
6000	A

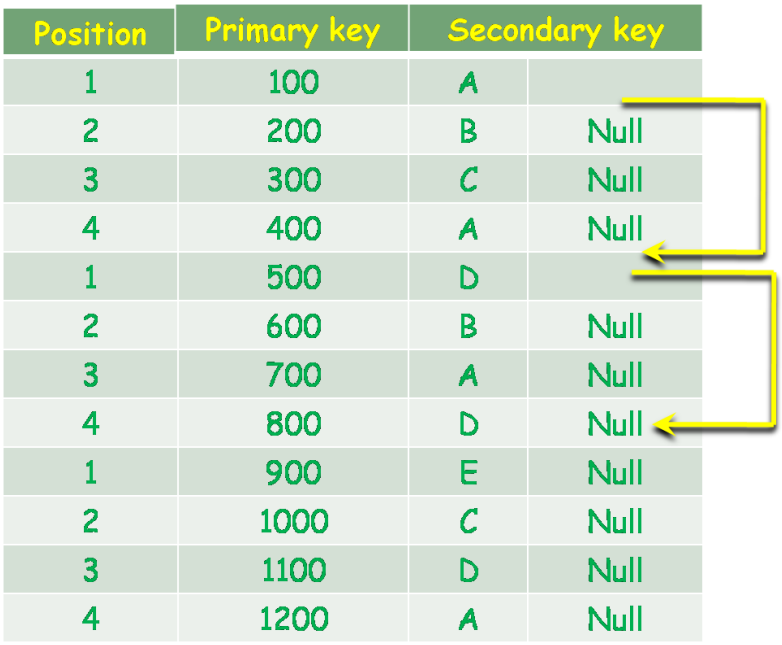
- The inversion process is associated with the information of inverted list
- In inverted files, the record is accessed in two steps. First, the indices are searched to obtain a list of required records and then second, records are retrieved using these lists
- The number of disk accesses required is equal to the number of records being retrieved plus the number to process the indices
- In inverted files, only the index structures are important

Cellular Partitions

- To decrease file search time, the storage media may be divided into cells

- ❖ A cell may be an entire disk or a cylinder. Lists are localized to lie within a cell
- ❖ If a cylinder is used as a cell, then all records on the same cylinder may be accessed without moving the read/write heads
- ❖ We divide multilists organized on several different cylinders into several small lists which are stored on the same cylinder

Multilist structure with cellular partitioning



Position	Primary key	Secondary key	
1	100	A	
2	200	B	Null
3	300	C	Null
4	400	A	Null
1	500	D	
2	600	B	Null
3	700	A	Null
4	800	D	Null
1	900	E	Null
2	1000	C	Null
3	1100	D	Null
4	1200	A	Null

Question 7: Explain Sequential file organization and Direct Access file Organization

Answer: F**ile Organization**

File organization ensures that records are available for processing. It is used to determine an efficient file organization for each base relation.

For example, if we want to retrieve employee records in alphabetical order of name. Sorting the

file by employee name is a good file organization. However, if we want to retrieve all employees whose marks are in a certain range, a file is ordered by employee name would not be a good file organization.

Types of File Organization

There are three types of organizing the file:

1. Sequential access file organization
2. Direct access file organization
3. Indexed sequential access file organization

1. Sequential access file organization

- Storing and sorting in contiguous block within files on tape or disk is called as **sequential access file organization**.
- In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.
- Sequential file search starts from the beginning of the file and the records can be added at the end of the file.
- In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.

Advantages of sequential file

- It is simple to program and easy to design.
- Sequential file is best use if storage space.

Disadvantages of sequential file

- Sequential file is time consuming process.
- It has high data redundancy.
- Random searching is not possible.

2. Direct access file organization

- Direct access file is also known as random access or relative file organization.

- In direct access file, all records are stored in direct access storage device (DASD), such as hard disk. The records are randomly placed throughout the file.
- The records does not need to be in sequence because they are updated directly and rewritten back in the same location.
- This file organization is useful for immediate access to large amount of information. It is used in accessing large databases.
- It is also called as hashing.

Advantages of direct access file organization

- Direct access file helps in online transaction processing system (OLTP) like online railway reservation system.
- In direct access file, sorting of the records are not required.
- It accesses the desired records immediately.
- It updates several files quickly.
- It has better control over record allocation.

Disadvantages of direct access file organization

- Direct access file does not provide back up facility.
- It is expensive.
- It has less storage space as compared to sequential file.

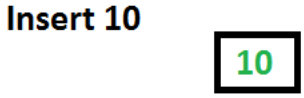
Question 8: write short note on External Sort.

Answer: External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory (usually a hard drive). External sorting typically uses a hybrid sort-merge strategy. In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file. In the merge phase, the sorted sub-files are combined into a single larger file.

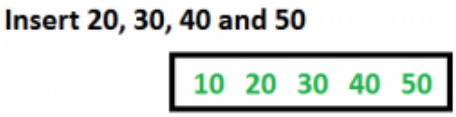
One example of external sorting is the external merge sort algorithm, which sorts chunks that each fit in RAM, then merges the sorted chunks together. We first divide the file into **runs** such that the size of a run is small enough to fit into main memory. Then sort each run in main memory using merge sort sorting algorithm. Finally merge the resulting runs together into successively bigger runs, until the file is sorted.

Question 9: Construct Btree of order 3: 10, 20, 30, 40, 50, 60, 70, 80 and 90

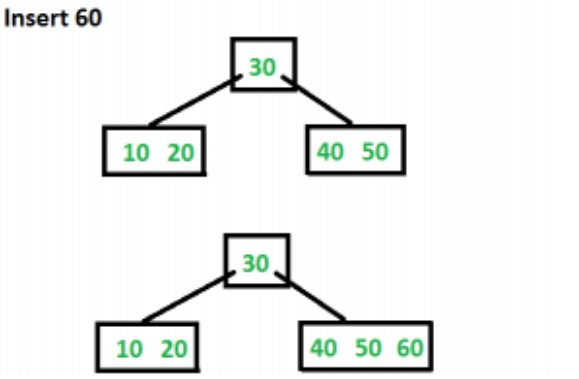
Answer: Initially root is NULL. Let us first insert 10.



Let us now insert 20, 30, 40 and 50. They all will be inserted in root because maximum number of keys a node can accommodate is 2*t – 1 which is 5.

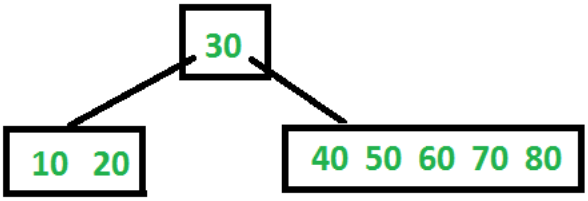


Let us now insert 60. Since root node is full, it will first split into two, then 60 will be inserted into the appropriate child.



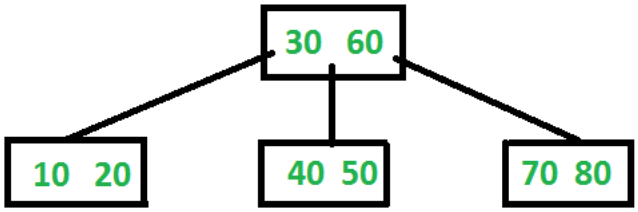
Let us now insert 70 and 80. These new keys will be inserted into the appropriate leaf without any split.

Insert 70 and 80



Let us now insert 90. This insertion will cause a split. The middle key will go up to the parent.

Insert 90



Question 10: Explain Collision Resolution Techniques in Hashing.

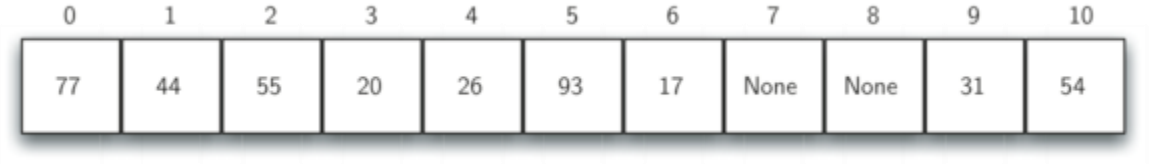
Answer: When two items hash to the same slot, we must have a systematic method for placing the second item in the hash table. This process is called collision resolution. As we stated earlier, if the hash function is perfect, collisions will never occur. However, since this is often not possible, collision resolution becomes a very important part of hashing.

One method for resolving collisions looks into the hash table and tries to find another open slot to hold the item that caused the collision. A simple way to do this is to start at the original hash value position and then move in a sequential manner through the slots until we encounter the first slot that is empty. Note that we may need to go back to the first slot (circularly) to cover the entire hash table. This collision resolution process is referred to as open addressing in that it tries to find the next open slot or address in the hash table. By systematically visiting each slot one at a time, we are performing an open addressing technique called linear probing.

Figure 8 shows an extended set of integer items under the simple remainder method hash function (54,26,93,17,77,31,44,55,20). Table 4 above shows the hash values for the original items. Figure 5 shows the original contents. When we attempt to place 44 into slot 0, a collision

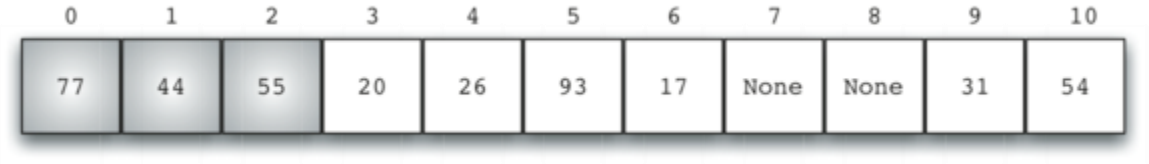
occurs. Under linear probing, we look sequentially, slot by slot, until we find an open position. In this case, we find slot 1.

Again, 55 should go in slot 0 but must be placed in slot 2 since it is the next open position. The final value of 20 hashes to slot 9. Since slot 9 is full, we begin to do linear probing. We visit slots 10, 0, 1, and 2, and finally find an empty slot at position 3.

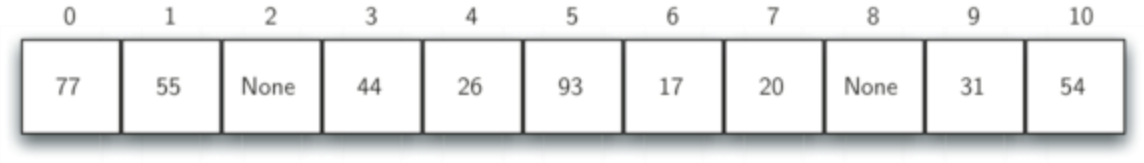


Once we have built a hash table using open addressing and linear probing, it is essential that we utilize the same methods to search for items. Assume we want to look up the item 93. When we compute the hash value, we get 5. Looking in slot 5 reveals 93, and we can return True. What if we are looking for 20? Now the hash value is 9, and slot 9 is currently holding 31. We cannot simply return False since we know that there could have been collisions. We are now forced to do a sequential search, starting at position 10, looking until either we find the item 20 or we find an empty slot.

A disadvantage to linear probing is the tendency for **clustering**; items become clustered in the table. This means that if many collisions occur at the same hash value, a number of surrounding slots will be filled by the linear probing resolution. This will have an impact on other items that are being inserted, as we saw when we tried to add the item 20 above. A cluster of values hashing to 0 had to be skipped to finally find an open position. This cluster is shown in Figure 9.

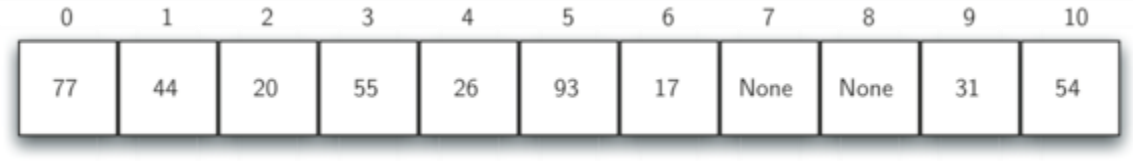


One way to deal with clustering is to extend the linear probing technique so that instead of looking sequentially for the next open slot, we skip slots, thereby more evenly distributing the items that have caused collisions. This will potentially reduce the clustering that occurs. Figure 10 shows the items when collision resolution is done with a “plus 3” probe. This means that once a collision occurs, we will look at every third slot until we find one that is empty.



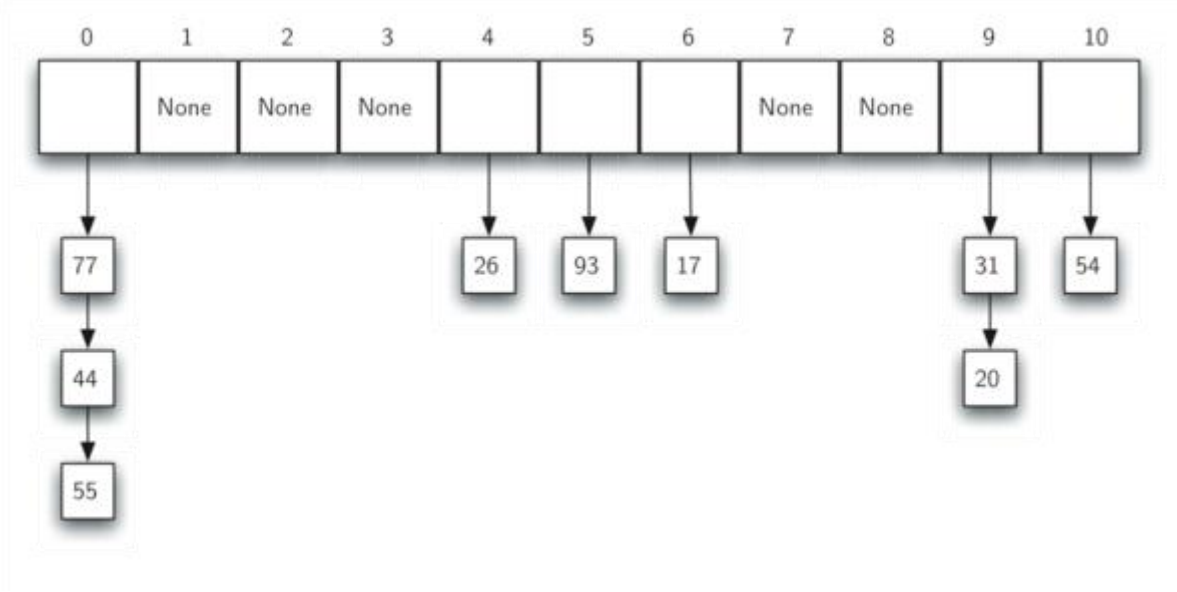
The general name for this process of looking for another slot after a collision is rehashing. With simple linear probing, the rehash function is newhashvalue=rehash(oldhashvalue)newhashvalue=rehash(oldhashvalue) where rehash(pos)=(pos+1)%sizeoftablerehash(pos)=(pos+1)%sizeoftable. The "plus 3" rehash can be defined as rehash(pos)=(pos+3)%sizeoftablerehash(pos)=(pos+3)%sizeoftable. In general, rehash(pos)=(pos+skip)%sizeoftablerehash(pos)=(pos+skip)%sizeoftable. It is important to note that the size of the "skip" must be such that all the slots in the table will eventually be visited. Otherwise, part of the table will be unused. To ensure this, it is often suggested that the table size be a prime number. This is the reason we have been using 11 in our examples.

A variation of the linear probing idea is called quadratic probing. Instead of using a constant "skip" value, we use a rehash function that increments the hash value by 1, 3, 5, 7, 9, and so on. This means that if the first hash value is *h*, the successive values are h+1h+1, h+4h+4, h+9h+9, h+16h+16, and so on. In other words, quadratic probing uses a skip consisting of successive perfect squares. Figure 11 shows our example values after they are placed using this technique.



An alternative method for handling the collision problem is to allow each slot to hold a reference to a collection (or chain) of items. Chaining allows many items to exist at the same location in the hash table. When collisions happen, the item is still placed in the proper slot of the hash table. As more and more items hash to the same location, the difficulty of searching for the item in the collection increases. Figure 12shows the items as they are added to a hash table that uses

chaining to resolve collisions.



When we want to search for an item, we use the hash function to generate the slot where it should reside. Since each slot holds a collection, we use a searching technique to decide whether the item is present. The advantage is that on the average there are likely to be many fewer items in each slot, so the search is perhaps more efficient.

Question11: Explain Different types of Hash functions.

Answer:

1. **Division Method:**

   One of the required features of the hash function is that the resultant index must be within the table index range

   One simple choice for a hash function is to use the modulus division indicated as MOD (the operator % in C/C++)

   The function returns an integer

   If any parameter is NULL, the result is NULL

   Hash(Key) = Key % M

2. **Multiplication Method:**

The multiplication method works as:

1. Multiply the key 'Key' by a constant A in the range $0 < A < 1$ and extract the fractional part of Key ´ A
2. Then multiply this value by M and take the floor of the result

$$Hash(Key) = M (Key XA MOD 1),$$

where Key ´ A MOD 1 means the fractional part of Key ´ A,

that is,

Key X A − Key X A and A = (sqrt (5) − 1/2 = 0.6180339887)

3. Extraction Method:

When a portion of the key is used for the address calculation, the technique is called as the extraction method

In digit extraction, few digits are selected and extracted from the key which are used as the address

Key	Hashed Address
345678	**357**
234137	**243**
952671	**927**

4. Mid-Square Hashing:

The mid-square hashing suggests to take square of the key and extract the middle digits of the squared key as address

The difficulty is when the key is large. As the entire key participates in the address calculation, if the key is large, then it is very difficult to store the square of it as the square of key should not exceed the storage limit

So mid-square is used when the key size is less than or equal to 4 digits

5. **Folding Technique:**

In folding technique, the key is subdivided into subparts that are combined or folded and then combined to form the address

For the key with digits, we can subdivide the digits in three parts, add them up, and use the result as an address.

Here the size of subparts of key could be as that of the address

There are two types of folding methods:

Fold shift — Key value is divided into several parts of that of the size of the address. Left, right, and middle parts are added

Fold boundary — Key value is divided into parts of that of the size of the address

Left and right parts are folded on fixed boundary between them and the centre part.

6. **Rotation:**

When keys are serial, they vary in only last digit and this leads to the creation of synonyms. Rotating key would minimize this problem. This method is used along with other methods. Here, the key is rotated right by one digit and then use of folding would avoid synonym.

For example,

let the key be 120605, when it is rotated we get 512060

Then further the address is calculated using any other hash function

7. **Universal Hashing:** The main idea behind universal hashing is to select the hash function at random at run time from a carefully designed set of functions

Because of randomization, the algorithm can behave differently on each execution; even for the same input

This approach guarantees good average case performance, no matter what keys are provided as input

Question15: Write Short note on Hashing.

Answer: Hashing is finding an address where the data is to be stored as well as located using a key with the help of the algorithmic function. Hashing is a method of directly computing the address of the record with the help of a key by using a suitable mathematical function called the hash function. A hash table is an array-based structure used to store <key, information> pairs. The resulting address is used as the basis for storing and retrieving records and this address is called as home address of the record. For array to store a record in a hash table, hash function is applied to the key of the record being stored, returning an index within the range of the hash table. The item is then stored in the table of that index position.

Fig. Hashing Concept

Hash Function: A function that maps a key into the range [0 to Max − 1], the result of which is used as an index (or address) to hash table for storing and retrieving record.

Question16: Write a Short note Index Sequential File organization.

Answer:

File—Records are stored sequentially but the index file is prepared for accessing the record directly

An index file contains records ordered by a record key

The record key uniquely identifies the record and determines the sequence in which it is accessed with respect to other records.

A file that is loaded in key sequence but can be accessed directly by use of one or more indices is known as an indexed sequential file. A sequential data file that is indexed is called as indexed sequential file. A solution to improve speed of retrieving target is index sequential file.An indexed file contains records ordered by a record key. Each record contains a field that contains the record key.

Advantages:

- ❖ Accessing any record is more efficient than sequential file organization
- ❖ Large amount of data can be stored using this type of file organization

Disadvantages:

- ❖ Often more than one indices are needed which occupies large storage area

Question17: Explain types of Indexes.

Answer:

Primary index: It is an index ordered in the same way as the data file, which is sequentially ordered according to a key. The indexing field is equal to this key

Secondary index: An index that is defined on a non-ordering field of the data file.In this case, the indexing field need not contain unique values

Clustering indexes: A data file can associate with utmost one primary index and several secondary indexes. The single-level indexing structure is the simplest one where a file, whose records are pairs, contains a key and a pointer.

Question18: Explain Heap Data Structure.

Answer:

A heap is a binary tree having the following properties:

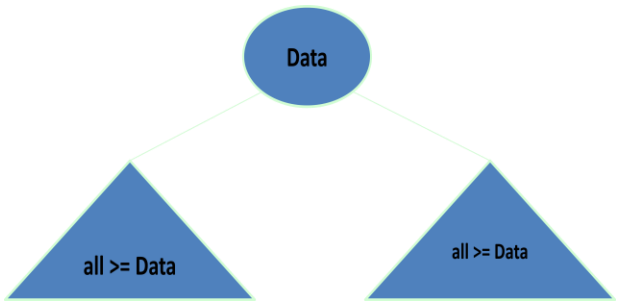
It is a complete binary tree, that is, each level of the tree is completely filled, except at the bottom level

At this level, it is filled from left to right

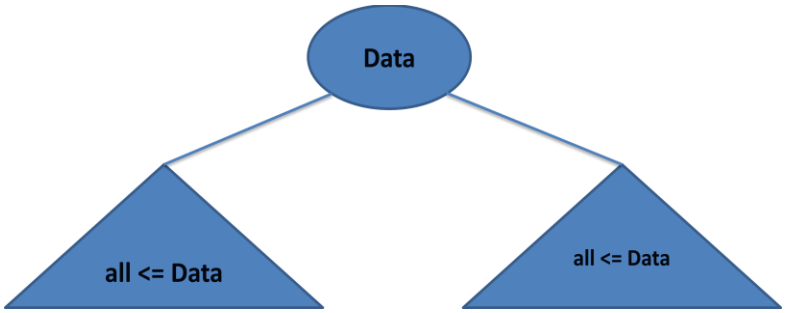
It satisfies the heap-order property, that is, the key value of each node is greater than or equal to the key value of its children, or the key value of each node is lesser than or equal to the key value of its children

Types of Heap:

1. Min-heap : In min-heap, the key value of each node is lesser than or equal to the key value of its children. In addition, every path from root to leaf should be sorted in ascending order



2. Max-heap: A max-heap is where the key value of a node is greater than the key values in all of its sub trees. In general, whenever the term 'heap' is used by itself



❖ **Structure Property:**

A binary tree is complete if it is of height h and has $2^{h+1} - 1$ nodes.

A binary tree of height h is complete if

- ❖ it is empty, or
- ❖ its left sub tree is complete of height h − 1 and its right sub tree is completely full of height h − 2, or
- ❖ its left sub tree is completely full of height h − 1 and its right sub tree is complete of height h − 1.

A complete tree is filled from the left when all the leaves are on the same level or two adjacent ones , all nodes at the lowest level are as far to the left as possible

Heap-order Property:A binary tree has the heap property if :

- ❖ it is empty or
- ❖ the key in the root is larger than either children and both sub trees have the heap property.