Assignment No. 2

Title :-

Implementation of Dictionary ADT using hashing with collision Handling (chaining with / without replacement)

Objectives :-

1. To understand the concept of hashing & its applications in data storage & retrieval.
2. To implement dictionary ADT operations such as insert, find & delete using hashing technique.
3. To compare different collision handling technique. (chaining with & without replacement)
4. To develop programming skills for implementing data structure using object-oriented principles.

Learning Objectives :-

1. To learn how to handle collisions effectively using chaining.
2. To gain knowledge of hashing collisions efficiently using chaining methods.
3. To implement key-value pair operations in a hash table based dictionary.

Learning Outcomes :-

- Understand & explain the fundamental of hashing & collision resolution.
- Successfully implement a dictionary ADT hash tables & chaining.

Theory :-

The Dictionary Abstract Data Type (ADT) is a data structure that stores key-value pairs allowing efficient retrieval, insertion & deletion of values based on unique keys.

Operations :- The primary operations include

1. insert (key, value) : Adds a new key-value pair.
2. find (key) : Retrieves the value associated with a given key.
3. Delete (key) : Removes key-value pair from the dictionary.

Hashing :-

Hashing is a technique used to map data (keys) to a fixed size array called a hash table using a hash function.

A hash function transforms an input key into an index (hash code) within the table.

Eg. hash (key) = key % table size.

Collision Handling Technique :-

When 2 keys produce the same hash index, a collision occurs.

Method :-

1] Chaining :-

Here, each table index stores a linked list / to hold multiple key-value pair.

Types of chaining :-

a] with replacement - if collision occurs, a new key-value pair replace the existing entry if it matches otherwise its added.

b] without replacement - The new entry is simply appended to existing chain without replacing any element.

Insertion operation :-

Without Replacement :-

```
index = hash (key) % size
if table [index] is empty :
    table [index] = (key, value)
else :
    add_chain ( table [index], (key, value))
```

With replacement :-

if table [index] has same key :
    table [index] = (key, value)
else :
    add_chain (table [index] , (key, value))

2] Open addressing :-
    Instead of using linked list handles collisions by finding next ava position within table.

Common strategies include :-

- Linear Probing.
- Quadratic Probing.
- Double Hashing.

• Dictionary ADT operations using Hashing

1] Insertion :-
    Compute Hash index using hash function, if slot is empty insert key value pair. If occupied, resolve the collision using chaining or open add

2] Search (Find) :-

Compute hash index, traverse the list or probe for correct key, return the associated value if key is found.

3] Deletion :-

Compute hash index, locate & remove key-value pair. Adjust chain or rehash in case of open addressing.

Conclusion :-

In this way, we have implemented a hash table in Python for quick data lookup. This helps in efficiently storing & retrieving key-value pairs.

Assignment No. 3.

Title :-

A book consists of chapters, chapter consists of sections & sections consist of subsections. Construct a tree & print the nodes. Find the time & space requirement of your methods.

Objectives :-

- To understand concept of tree data structure.
- To understand concept & features of object oriented programming.

Learning Objectives :-

- To understand concept of class.
- To understand concept & features of OOP.
- To understand concept of tree data structure.

Learning Outcomes :-

- Define class for structure using OOP.
- Analyze tree data structure.

Theory :-

Introduction to Tree :-

A tree T is a set of nodes storing elements such that nodes have parent-child relationship. that satisfies if T is not empty.

- T is either empty.
- It consist of a node r (the root) and possibly empty set of whose roots are children of r.

A node may contain a value or a condition or represent separate data structu or a tree of its own. Each node in a tree has zero or more child nodes which are one level lower in the tree heiron

A node has atmost one parent. A node that has no childs is called a leaf that node is ofcourse at the bottommost tree.

1. The height of tree with no element is
2. The height of tree with 1 element is 1
3. The height of a tree with >1 element equal to 1 + height o fits tallest subtree.

The depth of a node is length of th to its root. Every child node is alway level lower than his parent.
The topmost node in a tree is cal root node. Being the topmost node, th root node will not have parents.

The subtree corresponding to the any other node is called a proper sub-tree.

Every node in a tree can be seen as the root node of the sub-tree rooted at the node.

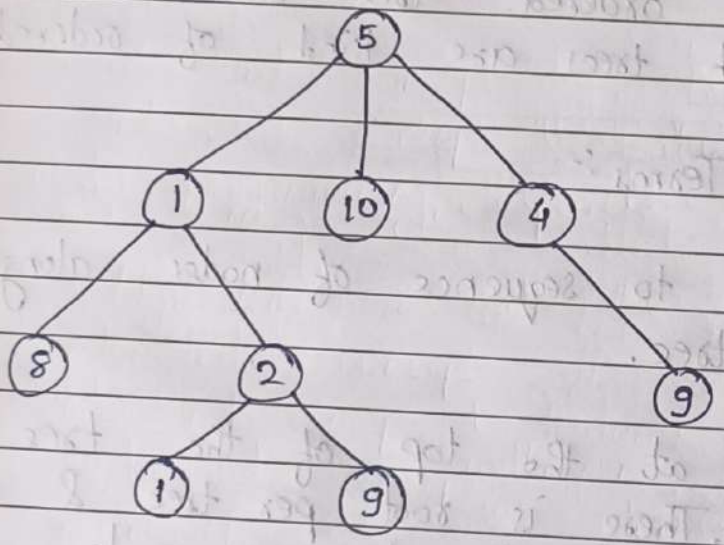


fig. An example of a Tree.

An internal node or inner node is any node of a tree that has child nodes & its thus not a leaf node.

There are 2-basic types of trees :-

1. Unordered Tree.
2. Ordered Tree.

In an unordered tree, a tree is a tree
a purely structured sense that is to say
a node, is no order for children of that

A tree on which order is imposed is ca
an Ordered tree, & structures built on th
are called ordered tree data structures.
Binary search trees are kind of ordered tree.

Important Terms :-

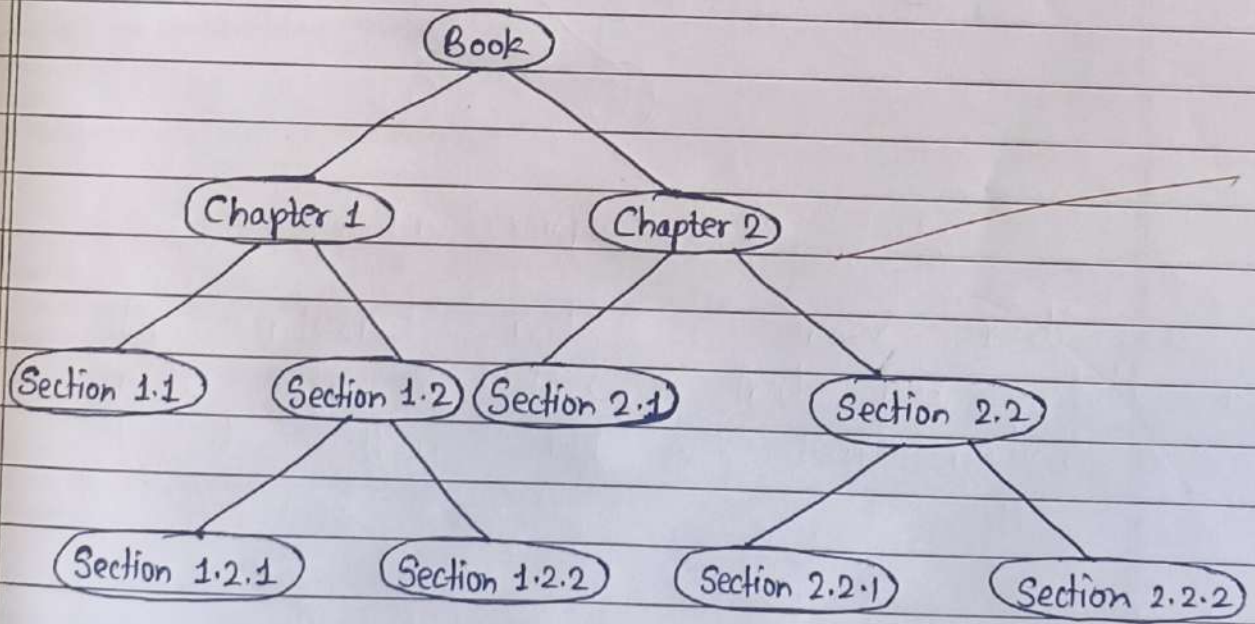
- Path - Refers to sequence of nodes along the
of tree.
- Root - Node at the top of the tree is ca
root. There is root per tree & one path
from root node to any node.
- Parent - Any node except the root node has
one edge upward to a node called pa
- Child - The node below a given node connected
its edge onward called its child node.
- Leaf - The node which does not have any
node is called as leaf node.
- Subtree - Represents the descendants of a node

- Visiting - Refers to checking the value of a node when control is on the node.
- Traversing - Passing through nodes in specific order.
- Level - Represents the generation of a node.

Advantages of Trees :-

1. Trees reflect structural relationships in the data.
2. Trees are used to represent hierarchies.
3. Trees provide an efficient insertion & searching.

For this assignment we consider tree as follows :-



Software Required :-
g++ / gcc compiler - / 64 bit fedora, eclipse

Input :
Book name & its number of sections subsections along with name.

Output :-
Formation of tree structure for book & its sections.

Conclusion :-
This program gives us the knowledge of data structure.