Mert Kilic

100334228

Assignment 5

CPSC 2150

**1)**

**Separate Chaining:**

a)

0 → ibex → bat → dog → nullptr

1 → koala → nullptr

2 → hare → nullptr

3 → nullptr

4 → nullptr

5 → nullptr

6 → ape → mud → nullptr

7 → nullptr

8 → carp → stork → nullptr

9 → nullptr

10 → nullptr

b)

6 → ape → mud → nullptr

**Linear Probing:**

a)

|  |  |
| --- | --- |
| 0 | ibex |
| 1 | bat |
| 2 | hare |
| 3 | koala |
| 4 | dog |
| 5 |  |
| 6 | ape |
| 7 | mud |
| 8 | carp |
| 9 | stork |
| 10 |  |

b)

|  |  |
| --- | --- |
| 0 | ibex |
| 1 | bat |
| 2 | hare |
| 3 | koala |
| 4 | dog |
| 5 |  |
| 6 | ape |
| 7 | mud |
| 8 | carp |
| 9 | stork |
| 10 |  |

**Double Hashing:**

a)

|  |  |
| --- | --- |
| 0 | ibex |
| 1 | bat |
| 2 | hare |
| 3 | koala |
| 4 | carp |
| 5 | mud |
| 6 | ape |
| 7 |  |
| 8 | dog |
| 9 |  |
| 10 | stork |

b)

|  |  |
| --- | --- |
| 0 | ibex |
| 1 | bat |
| 2 | hare |
| 3 | koala |
| 4 | carp |
| 5 | mud |
| 6 | ape |
| 7 |  |
| 8 | dog |
| 9 |  |
| 10 | stork |

**2)**

1)

Let M be 10 and N be 10,

If we have a set S = {1,21,31,41,51,61,71,81,91, 101}, all the keys from S will have the same hash value for h(k). Therefore, search will be O(N) which is the worst case.

2)

I would use this hash table. For example, for air traffic, this table can store all the flights that land at the same time as a linked list. That way, we could get all the data faster by iterating through the linked list.

**3)**

insert 5

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 | 5 |
| 6 |  |

insert 28

|  |  |
| --- | --- |
| 0 | 28 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 | 5 |
| 6 |  |

insert 19

|  |  |
| --- | --- |
| 0 | 28 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 | 5 |
| 6 | 19 |

insert 15

|  |  |
| --- | --- |
| 0 | 28 |
| 1 | 15 |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 | 5 |
| 6 | 19 |

insert 20, load factor > 0.7, increase the table size

|  |  |
| --- | --- |
| 0 | 28 |
| 1 | 15 |
| 2 | 20 |
| 3 |  |
| 4 |  |
| 5 | 5 |
| 6 | 19 |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |

insert 33

|  |  |
| --- | --- |
| 0 | 28 |
| 1 | 15 |
| 2 | 20 |
| 3 |  |
| 4 |  |
| 5 | 5 |
| 6 | 19 |
| 7 | 33 |
| 8 |  |
| 9 |  |
| 10 |  |

**4)**

**Array:**

insert = O(N)

search = O(N)

delete = O(N)

Using arrays for implementing a compiler would be highly inefficient because all the operations would be linear. As the array grows, compiling time would be very slow. Also, for deletion, shifting would be a good solution to avoid using too much space. Otherwise, arrays are also inefficient in terms of space complexity.

**Linked List:**

insert = O(1)

search = O(N)

delete = O(N)

Like arrays, linked list would be very inefficient in terms of time as well. Except inserting, all the other operations are slow. In terms of space, linked list only allocates enough memory for the nodes in the list and frees the memory after deletion.

**Balanced Binary Tree:**

insert = O(N)

search = O(N)

delete = O(N)

Since there is no such a rule for inserting searching and deleting, operations are very slow in this data structure. Again, program would go through the entire tree to find out data we are looking for. However, program would use less space since a successful deletion would remove the element from the tree and free the memory that being used for the data. Also, there is no waste of space since the program only allocates enough memory for a new node.

**Hash Table:**

insert = O(1)

search = O(1)

delete = O(1)

This implementation is the fastest one. If we think about the worst case, all the data structures I’ve mentioned, and the hash table are similar. They all have O(N) for the worst case. However, for the average case, hash table is very efficient in terms of all inserting, searching, and deleting. With a good hashing function, compiler would work efficiently. On the other hand, hash table might not be very efficient in terms of space because as the table capacity reaches the load factor, table must grow. In some cases, after the grow, our program might end up with a waste of space. Besides, the space that is not in use, table would not shrink after the deletion. Therefore, this data structure would be inefficient in terms of space.