**POST – LAB**

**PART 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Identifiers** | **Hash table size** | **Time taken by Linear Probing in milliseconds** | **Time taken by Chaining in milliseconds** |
| **100** | **113** | **0** | **4** |
| **1000** | **1009** | **0** | **4.3** |
| **5000** | **4999** | **0** | **4.5** |
| **8000** | **7013** | **0** | **5** |

**Part 2**

#include<iostream>

#include<Windows.h>

using namespace std;

int const tablesize = 10;

struct hash\_node

{

int val,key;

hash\_node \*next;

hash\_node \*prev;

};

class Hashchain

{

public:

hash\_node \*\*hashtable, \*\*top;

Hashchain()

{

hashtable= new hash\_node\*[tablesize];

top= new hash\_node\*[tablesize];

for(int i=0;i<tablesize;i++)

{

hashtable[i]=NULL;

top[i]=NULL;

}

}

int HashF(int key)

{

key % tablesize;

return key;

}

void search(int key)

{

int hash\_val = HashF(key);

bool B = false;

hash\_node\* entry = hashtable[hash\_val];

if (entry != NULL)

{

while (entry != NULL)

{

if (entry->key == key)

{

B = true;

}

if (B)

{

cout << "Element found at key "<< key << " is ";

cout << entry->val << endl;

}

entry = entry->next;

}

}

if (!B)

cout << "No Element found at key "

<< key << endl;

}

void remove(int key)

{

int hash\_val = HashF(key);

hash\_node\* entry = hashtable[hash\_val];

if (entry->key != key || entry == NULL)

{

cout << "Couldnot find any element at this key "<< key << endl;

return;

}

while (entry != NULL)

{

if (entry->next == NULL)

{

if (entry->prev == NULL)

{

hashtable[hash\_val] = NULL;

top[hash\_val] = NULL;

delete entry; break;

}

else

{

top[hash\_val] = entry->prev;

top[hash\_val]->next = NULL;

delete entry;

entry = top[hash\_val];

}

}

entry = entry->next;

}

cout << "Element was removed at the key "<< key << endl;

}

void insert(int key, int value)

{

int hash\_val = HashF(key);

hash\_node\* entry = hashtable[hash\_val];

if (entry == NULL)

{

entry = new hash\_node;

entry->val = value;

entry->key = key;

entry->next = NULL;

entry->prev = NULL;

hashtable[hash\_val] = entry;

top[hash\_val] = entry;

}

else

{

while (entry != NULL)

entry = entry->next;

entry = new hash\_node;

entry->val = value;

entry->key = key;

entry->next = NULL;

entry->prev = top[hash\_val];

top[hash\_val]->next = entry;

top[hash\_val] = entry;

}

cout << "Value " << value << " was added at key " << key << endl;

}

};

int main()

{

Hashchain H;

H.insert(1,2);

H.insert(2,6);

H.insert(3,5);

H.insert(4,8);

H.insert(5,9);

int find\_key;

cout<<"Enter the key to search for "<<endl;

cin>>find\_key;

DWORD start, end;//measure time

start= GetTickCount();

H.search(find\_key);

end= GetTickCount();

int del;

cout<<"enter the key do you want to remove : ";

cin>>del;

H.remove(del);

return 0; }



**Part 3**

#include<iostream>

#include <list>

using namespace std;

class Hash

{

int BUCKET; // No. of buckets

// Pointer to an array containing buckets

list<int> \*table;

public:

Hash(int V); // Constructor

// inserts a key into hash table

void insertItem(int x);

// deletes a key from hash table

void deleteItem(int key);

// hash function to map values to key

int hashFunction(int x) {

return (x % BUCKET);}

void displayHash();};

Hash::Hash(int b)

{

this->BUCKET = b;

table = new list<int>[BUCKET];

}

void Hash::insertItem(int key)

{

int index = hashFunction(key);

table[index].push\_back(key); }

void Hash::deleteItem(int key)

{

// get the hash index of key

int index = hashFunction(key);

// find the key in (inex)th list

list <int> :: iterator i;

for (i = table[index].begin();

i != table[index].end(); i++) {

if (\*i == key)

break;}

// if key is found in hash table, remove it

if (i != table[index].end())

table[index].erase(i);}

// function to display hash table

void Hash::displayHash() {

for (int i = 0; i < BUCKET; i++) {

cout << i;

for (auto x : table[i])

cout << " --> " << x;

cout << endl; }}

// Driver program

int main()

{

// array that contains keys to be mapped

int a[] = {15, 11, 27, 8, 12};

int n = sizeof(a)/sizeof(a[0]);

// insert the keys into the hash table

Hash h(7); // 7 is count of buckets in

// hash table

for (int i = 0; i < n; i++)

h.insertItem(a[i]);

// delete 12 from hash table

h.deleteItem(12);

// display the Hash table

h.displayHash();

return 0;}

