**Muhammad Abdullah**

**SE(3A) | 19F-0916**

DS Assignment 7

Hashing

**Question # 1:**

**PROGRAM:**

#include <iostream>

#include <string>

#include <queue>

#include <iomanip>

#include <fstream>

#include <algorithm>

#include <cstdlib>

#include <ctime>

using namespace std;

void initialize();

struct node

{

string data;

struct node\* left;

struct node\* right;

};

class avlTree

{

node\* root;

public:

int height(node\*);

int diff(node\*);

node\* SearchNode(string, node\*);

node\* FindLargest(node\*);

node\* FindSmallest(node\*);

node\* rr\_rotation(node\*);

node\* ll\_rotation(node\*);

node\* lr\_rotation(node\*);

node\* rl\_rotation(node\*);

node\* balance(node\*);

node\* insert(node\*, string);

void display(node\*);

node\* getRoot()

{

return root;

}

avlTree()

{

root = NULL;

}

};

node\* arr[26];

avlTree obj[26];

void insert(string num)

{

node\* new\_node = new node;

new\_node->data = num;

char c = num[0];

int index = (int)(c - 97);

arr[index] = obj[index].insert(arr[index], num);

}

void display()

{

for (int i = 0; i < 26; i++)

{

cout << "Char: " << char(i + 97) << " -> ";

obj[i].display(arr[i]);

cout << endl;

}

}

void search(string num)

{

char c = num[0];

int index = (int)(c - 97);

node\* ptr = obj[index].SearchNode(num, arr[index]);

if (ptr)

cout << num << " is Found in the Dictionary" << endl;

else

cout << num << " is NOT FOUND in the Dictionary" << endl;

}

int main()

{

srand(time(0));

initialize();

bool fh = true;

int choice;

string word;

while (fh)

{

system("cls");

cout << endl << " Hashing by Separate Chaining" << endl << "====================================" << endl << endl

<< " 1) Insert a Word" << endl

<< " 2) Display Complete Dictionary" << endl

<< " 3) Search a Word" << endl

<< " 4) Exit your Program" << endl << endl << "====================================" << endl << endl

<< " Your Choice: ";

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

cout << "Enter word to insert: ";

getline(cin, word);

insert(word);

break;

case 2:

display();

break;

case 3:

cout << "Enter word to search: ";

getline(cin, word);

search(word);

break;

case 4:

fh = false;

break;

default:

cout << "Invalid Choice Entered...";

}

cout << endl << endl << "Press Enter to Continue....";

getchar();

}

system("pause");

return 0;

}

void initialize()

{

for (int i = 0; i < 26; i++)

{

arr[i] = NULL;

}

int i = 0;

fstream obj;

obj.open("text.txt", ios::in);

string t;

int nums[100];

for (int j = 0; j < 100; j++)

nums[j] = rand() % 10000;

while (!obj.eof())

{

getline(obj, t);

for (int j = 0; j < 100; j++)

{

if (i == nums[j])

insert(t);

}

i++;

}

obj.close();

}

int avlTree::height(node \*temp)

{

int h = 0;

if (temp != NULL)

{

int l\_height = height(temp->left);

int r\_height = height(temp->right);

int max\_height = max(l\_height, r\_height);

h = max\_height + 1;

}

return h;

}

int avlTree::diff(node \*temp)

{

int l\_height = height(temp->left);

int r\_height = height(temp->right);

int b\_factor = l\_height - r\_height;

return b\_factor;

}

node \*avlTree::rr\_rotation(node \*parent)

{

node \*temp;

temp = parent->right;

parent->right = temp->left;

temp->left = parent;

return temp;

}

node \*avlTree::ll\_rotation(node \*parent)

{

node \*temp;

temp = parent->left;

parent->left = temp->right;

temp->right = parent;

return temp;

}

node \*avlTree::lr\_rotation(node \*parent)

{

node \*temp;

temp = parent->left;

parent->left = rr\_rotation(temp);

return ll\_rotation(parent);

}

node \*avlTree::rl\_rotation(node \*parent)

{

node \*temp;

temp = parent->right;

parent->right = ll\_rotation(temp);

return rr\_rotation(parent);

}

node \*avlTree::balance(node \*temp)

{

int bal\_factor = diff(temp);

if (bal\_factor > 1)

{

if (diff(temp->left) > 0)

temp = ll\_rotation(temp);

else

temp = lr\_rotation(temp);

}

else if (bal\_factor < -1)

{

if (diff(temp->right) > 0)

temp = rl\_rotation(temp);

else

temp = rr\_rotation(temp);

}

return temp;

}

node \*avlTree::insert(node \*root, string value)

{

if (root == NULL)

{

root = new node;

root->data = value;

root->left = NULL;

root->right = NULL;

return root;

}

else if (value < root->data)

{

root->left = insert(root->left, value);

root = balance(root);

}

else if (value >= root->data)

{

root->right = insert(root->right, value);

root = balance(root);

}

return root;

}

void avlTree::display(node \*root)

{

if (!root)

{

cout << "Nothing to Display";

return;

}

queue<node\*> obj;

obj.push(root);

while (!obj.empty())

{

node\* temp = obj.front();

obj.pop();

cout << temp->data << " ";

if (temp->left)

obj.push(temp->left);

if (temp->right)

obj.push(temp->right);

}

}

node\* avlTree::SearchNode(string data, node\* Root)

{

if (!Root)

return NULL;

else if (data == Root->data)

return Root;

else if (data > Root->data)

return SearchNode(data, Root->right);

else if (data < Root->data)

return SearchNode(data, Root->left);

}

node\* avlTree::FindLargest(node\* Root)

{

if (!Root)

{

return NULL;

}

if (Root->right)

return FindLargest(Root->right);

else

return Root;

}

node\* avlTree::FindSmallest(node\* Root)

{

if (!Root)

{

return NULL;

}

if (Root->left)

return FindSmallest(Root->left);

else

return Root;

}

**MAIN CODE MENU**

A picture containing text, monitor, screenshot, electronics

Description automatically generated

**DISPLAYING DICTIONERY**

A picture containing text, screenshot, computer, electronics

Description automatically generated

**Question # 2: ( LINEAR PROBING)**

**PROGRAM:**

#include <iostream>

#include <string>

#include <windows.h>

using namespace std;

struct node

{

string name;

float marks;

string address;

};

node arr[20];

void initialize();

int insert(node, int, int);

int hash\_fun(int, int);

void display(int);

void search(int);

void delete\_num(int);

int main()

{

initialize();

bool fh = true;

int choice, Size;

while (fh)

{

system("cls");

cout << endl << " Hashing by Linear Probing" << endl << "----------------------------------------" << endl<< endl

<< " 1) Insert a Student" << endl

<< " 2) Display the List" << endl

<< " 3) Search a Student" << endl

<< " 4) Delete a Student" << endl

<< " 5) Exit your Program" << endl << endl << "----------------------------------------" << endl

<< " Your Choice: ";

cin >> choice;

cin.ignore();

node temp;

switch (choice)

{

case 1:

cout << "Enter Student Name: ";

getline(cin, temp.name);

cout << "Enter Student Marks: ";

cin >> temp.marks;

while (temp.marks < 0 || temp.marks > 100)

{

cout << "Invalid marks entered.. Enter again: ";

cin >> temp.marks;

}

cin.ignore();

cout << "Enter Student Address: ";

getline(cin, temp.address);

Size = insert(temp, 0, 0);

if (Size == 20)

cout << "List is Full. New Number can't be Inserted...\n";

else

cout << "\nNumber has been Inserted...\n";

cout << endl << endl;

break;

case 2:

display(-1);

cout << "\n";

break;

case 3:

cout << "Enter Marks to Search: ";

cin >> choice;

cin.ignore();

search(choice);

break;

case 4:

cout << "Enter Marks to Delete: ";

cin >> choice;

cin.ignore();

delete\_num(choice);

break;

case 5:

fh = false;

break;

default:

cout << "Invalid Choice Entered...";

}

cout << endl << endl << "Press Enter to Continue....";

getchar();

}

return 0;

}

void initialize()

{

for (int i = 0; i < 20; i++)

arr[i].marks = -1;

}

int insert(node num, int f, int Size)

{

if (Size < 10)

{

Size++;

int index = hash\_fun(num.marks, f);

if (arr[index].marks == -1)

{

arr[index] = num;

}

else

return insert(num, ++f, Size);

}

return Size;

}

void display(int index = -1)

{

cout << "(-1's represent empty indexes)\nStudent Database: \n\n";

for (int i = 0; i < 20; i++)

{

cout << "Index: " << i << "\t\t";

if (arr[i].marks == -1)

cout << arr[i].marks;

else

{

cout << "Student Name: " << arr[i].name << endl

<< "\t\t\tStudent Marks: " << arr[i].marks << endl

<< "\t\t\tStudent Address: " << arr[i].address;

}

cout << endl << endl;

}

}

int hash\_fun(int num, int f)

{

return (num + f) % 10;

}

void search(int num)

{

int i = 0;

int index = hash\_fun(num, i++);

while ((arr[index].marks != num) && (arr[index].marks != -1) && (i < 20))

index = hash\_fun(num, i++);

if (arr[index].marks == -1 || i > 20)

cout << "Student not found in the List\n";

else

{

cout << "\tStudent with " << num << " marks found at Index Number: " << index << "\n";

cout << "\nStudent Name: " << arr[index].name << endl

<< "Student Marks: " << arr[index].marks << endl

<< "Student Address: " << arr[index].address << endl;

}

}

void delete\_num(int num)

{

int i = 0;

int index = hash\_fun(num, i++);

while ((arr[index].marks != num) && (arr[index].marks != -1) && (i < 20))

index = hash\_fun(num, i++);

if (arr[index].marks == -1 || i > 20)

cout << "Number not found in the list\n";

else

{

int zzz = arr[index].marks;

arr[index].marks = -1;

cout << "\tStudent with " << num << " marks has been deleted" << "\n";

cout << "\nStudent Name: " << arr[index].name << endl

<< "Student Marks: " << zzz << endl

<< "Student Address: " << arr[index].address << endl;

}

}

A picture containing text, monitor, screenshot, indoor

Description automatically generated

**DISPLAYING VALUES**

A picture containing text, screenshot, monitor, computer

Description automatically generated

**Question # 2: ( QUADRATIC PROBING)**

**PROGRAM:**

#include <iostream>

#include <string>

#include <windows.h>

using namespace std;

struct node

{

string name;

float marks;

string address;

};

node arr[20];

void initialize();

int insert(node, int, int);

int hash\_fun(int, int);

void display(int);

void search(int);

void delete\_num(int);

int main()

{

initialize();

bool fh = true;

int choice, Size;

while (fh)

{

system("cls");

cout << endl << " Hashing by Quadratic Probing" << endl << "----------------------------------------" << endl << endl

<< " 1) Insert a Student" << endl

<< " 2) Display the List" << endl

<< " 3) Search a Student" << endl

<< " 4) Delete a Student" << endl

<< " 5) Exit your Program" << endl << endl << "----------------------------------------" << endl

<< " Your Choice: ";

cin >> choice;

cin.ignore();

node temp;

switch (choice)

{

case 1:

cout << "Enter Student Name: ";

getline(cin, temp.name);

cout << "Enter Student Marks: ";

cin >> temp.marks;

while (temp.marks < 0 || temp.marks > 100)

{

cout << "Invalid marks entered.. Enter again: ";

cin >> temp.marks;

}

cin.ignore();

cout << "Enter Student Address: ";

getline(cin, temp.address);

Size = insert(temp, 0, 0);

if (Size == 20)

cout << "List is Full. New Number can't be Inserted...\n";

else

cout << "\nNumber has been Inserted...\n";

cout << endl << endl;

break;

case 2:

display(-1);

cout << "\n";

break;

case 3:

cout << "Enter Marks to Search: ";

cin >> choice;

cin.ignore();

search(choice);

break;

case 4:

cout << "Enter Marks to Delete: ";

cin >> choice;

cin.ignore();

delete\_num(choice);

break;

case 5:

fh = false;

break;

default:

cout << "Invalid Choice Entered...";

}

cout << endl << endl << "Press Enter to Continue....";

getchar();

}

return 0;

}

void initialize()

{

for (int i = 0; i < 20; i++)

arr[i].marks = -1;

}

int insert(node num, int f, int Size)

{

if (Size < 10)

{

Size++;

int index = hash\_fun(num.marks, (f \* f));

if (arr[index].marks == -1)

{

arr[index] = num;

}

else

return insert(num, ++f, Size);

}

return Size;

}

void display(int index = -1)

{

cout << "(-1's represent empty indexes)\nStudent Database: \n\n";

for (int i = 0; i < 20; i++)

{

cout << "Index: " << i << "\t\t";

if (arr[i].marks == -1)

cout << arr[i].marks;

else

{

cout << "Student Name: " << arr[i].name << endl

<< "\t\t\tStudent Marks: " << arr[i].marks << endl

<< "\t\t\tStudent Address: " << arr[i].address;

}

cout << endl << endl;

}

}

int hash\_fun(int num, int f)

{

return (num + f) % 10;

}

void search(int num)

{

int i = 0;

int index = hash\_fun(num, i++);

while ((arr[index].marks != num) && (arr[index].marks != -1) && (i < 20))

index = hash\_fun(num, i++);

if (arr[index].marks == -1 || i > 20)

cout << "Student not found in the List\n";

else

{

cout << "\tStudent with " << num << " marks found at Index Number: " << index << "\n";

cout << "\nStudent Name: " << arr[index].name << endl

<< "Student Marks: " << arr[index].marks << endl

<< "Student Address: " << arr[index].address << endl;

}

}

void delete\_num(int num)

{

int i = 0;

int index = hash\_fun(num, i++);

while ((arr[index].marks != num) && (arr[index].marks != -1) && (i < 20))

index = hash\_fun(num, i++);

if (arr[index].marks == -1 || i > 20)

cout << "Number not found in the list\n";

else

{

int zzz = arr[index].marks;

arr[index].marks = -1;

cout << "\tStudent with " << num << " marks has been deleted" << "\n";

cout << "\nStudent Name: " << arr[index].name << endl

<< "Student Marks: " << zzz << endl

<< "Student Address: " << arr[index].address << endl;

}

}

**A picture containing text, screenshot, monitor, indoor

Description automatically generated**

**DISPLAYING VALUES**

**A picture containing text, monitor, screenshot, electronics

Description automatically generated**

**Question # 2: ( UNIQUE HASHING FUCNTION)**

**PROGRAM:**

#include <iostream>

#include <string>

#include <windows.h>

using namespace std;

struct node

{

string name;

float marks;

string address;

};

node arr[20];

void initialize();

int insert(node, int, int);

int hash\_fun(int, int);

void display(int);

void search(int);

void delete\_num(int);

int main()

{

initialize();

bool fh = true;

int choice, Size;

while (fh)

{

system("cls");

cout << endl << " Hashing by Unique Funcion of Probing" << endl << "----------------------------------------" << endl << endl

<< " 1) Insert a Student" << endl

<< " 2) Display the List" << endl

<< " 3) Search a Student" << endl

<< " 4) Delete a Student" << endl

<< " 5) Exit your Program" << endl << endl << "----------------------------------------" << endl

<< " Your Choice: ";

cin >> choice;

cin.ignore();

node temp;

switch (choice)

{

case 1:

cout << "Enter Student Name: ";

getline(cin, temp.name);

cout << "Enter Student Marks: ";

cin >> temp.marks;

while (temp.marks < 0 || temp.marks > 100)

{

cout << "Invalid marks entered.. Enter again: ";

cin >> temp.marks;

}

cin.ignore();

cout << "Enter Student Address: ";

getline(cin, temp.address);

Size = insert(temp, 0, 0);

if (Size == 20)

cout << "List is Full. New Number can't be Inserted...\n";

else

cout << "\nNumber has been Inserted...\n";

cout << endl << endl;

break;

case 2:

display(-1);

cout << "\n";

break;

case 3:

cout << "Enter Marks to Search: ";

cin >> choice;

cin.ignore();

search(choice);

break;

case 4:

cout << "Enter Marks to Delete: ";

cin >> choice;

cin.ignore();

delete\_num(choice);

break;

case 5:

fh = false;

break;

default:

cout << "Invalid Choice Entered...";

}

cout << endl << endl << "Press Enter to Continue....";

getchar();

}

return 0;

}

void initialize()

{

for (int i = 0; i < 20; i++)

arr[i].marks = -1;

}

int insert(node num, int f, int Size)

{

if (Size < 10)

{

Size++;

int index = hash\_fun(num.marks, (f \* f \* f));

if (arr[index].marks == -1)

{

arr[index] = num;

}

else

return insert(num, ++f, Size);

}

return Size;

}

void display(int index = -1)

{

cout << "(-1's represent empty indexes)\nStudent Database: \n\n";

for (int i = 0; i < 20; i++)

{

cout << "Index: " << i << "\t\t";

if (arr[i].marks == -1)

cout << arr[i].marks;

else

{

cout << "Student Name: " << arr[i].name << endl

<< "\t\t\tStudent Marks: " << arr[i].marks << endl

<< "\t\t\tStudent Address: " << arr[i].address;

}

cout << endl << endl;

}

}

int hash\_fun(int num, int f)

{

return (num + f) % 10;

}

void search(int num)

{

int i = 0;

int index = hash\_fun(num, i++);

while ((arr[index].marks != num) && (arr[index].marks != -1) && (i < 20))

index = hash\_fun(num, i++);

if (arr[index].marks == -1 || i > 20)

cout << "Student not found in the List\n";

else

{

cout << "\tStudent with " << num << " marks found at Index Number: " << index << "\n";

cout << "\nStudent Name: " << arr[index].name << endl

<< "Student Marks: " << arr[index].marks << endl

<< "Student Address: " << arr[index].address << endl;

}

}

void delete\_num(int num)

{

int i = 0;

int index = hash\_fun(num, i++);

while ((arr[index].marks != num) && (arr[index].marks != -1) && (i < 20))

index = hash\_fun(num, i++);

if (arr[index].marks == -1 || i > 20)

cout << "Number not found in the list\n";

else

{

int zzz = arr[index].marks;

arr[index].marks = -1;

cout << "\tStudent with " << num << " marks has been deleted" << "\n";

cout << "\nStudent Name: " << arr[index].name << endl

<< "Student Marks: " << zzz << endl

<< "Student Address: " << arr[index].address << endl;

}

}

**MAIN MENU OF PROGRAM**

A picture containing text, screenshot, indoor, monitor

Description automatically generated

**DISPLAYING VALUES**

A picture containing text, monitor, screenshot, indoor

Description automatically generated

**Question # 3:**

**PROGRAM:**

**BY LINEAR PROBING**

Table

Description automatically generated

**Question # 3:**

**PROGRAM:**

**BY QUADRATIC PROBING**

**Table

Description automatically generated**