

Data structures

Lab-14



**Name: Talib Husain**

**Roll# 21F-9070**

**Task-1**

#include <iostream>

#include<queue>

#include<stack>

using namespace std;

//creating a Node for queue

struct Node {

    int data;

    Node\* next;

};

//Making a Queue ADT

class Queue {

    Node\* front;

    Node\* rear;

public:

//Constructor to initialze with NULL

    Queue() {

        front = NULL;

        rear = NULL;

    }

    //Functon to Insert a data in a queue

    void enqueue(int x)

    {

        // Create new node

        Node\* temp = new Node;

        temp->data = x;

        temp->next = NULL;

        // new node is front and rear both If queue is empty

        if (rear == NULL) {

            front = temp;

            rear = temp;

        }

        else {

            // Add the new node at end

            rear->next = temp;

            rear = temp;

        }

    }

    // Function to remove

    int dequeue()

    {

        // If queue is empty, return NULL

        if (front == NULL) {

            cout << "Queue is Empty";

            return NULL;

        }

        else{

        Node\* temp = front;

        front = front->next;

        if (front == NULL)

            rear = NULL;

        int data = temp->data;

        delete temp;

        temp = NULL;

        return data;

        }

    }

    void display(){

        Node \*temp = front;

        while (temp!=NULL)

        {

            cout<<temp->data<<" ";

            temp = temp->next ;

        }

        cout<<endl;

    }

    bool isEmpty(){

        if(front == NULL){

            return true;

        }

        else{

            return false;

        }

    }

};

//class graph

class Graph{

    char\*\*arr;

    int count;

    int no\_vertex = 0;

public:

  //constructor

    Graph(int no\_vertex = 9){

        count = 0;

        this->no\_vertex = no\_vertex;

        //Dynamically allocating 2-d array

        arr = new char\*[no\_vertex];

        for(int i=0;i<no\_vertex;i++){

            \*(arr+i) = new char[no\_vertex];

        }

        //initialising array with 0

        for(int i=0;i<no\_vertex;i++){

            for(int j =0;j<no\_vertex;j++){

                arr[i][j]  = '-';

            }

        }

    }

  //Code for addinf vertex

    void Addvertex(char x){

        if(count<no\_vertex){

        arr[count][0] = x;

        count++;

        }

        else

        cout<<"Vertex can't be added\n";

    }

  //code for adding a connection between two vertex

    void addedge(char vertex1,char vertex2){

        for(int i=0;i<no\_vertex;i++){

            if(arr[i][0] == vertex1){

                for(int j = 0;j<no\_vertex;j++){

                    if(arr[i][j] == '-'){

                        arr[i][j] = vertex2;

                        break;

                    }

                }

            }

        }

    }

  //Code for printig

    void print(){

        for(int i=0;i<no\_vertex;i++){

            for(int j =0;j<no\_vertex;j++){

                if(arr[i][j] !='-')

                cout<<arr[i][j];

                if(j==0)

                    cout<<": ";

                else

                    if(arr[i][j] !='-')

                cout<<" -> ";

            }

                cout<<"\n";

        }

    }

  //BFS for fining a path

    void BFS(char start, char goal){

      //queue to  mark

        queue<char> q;

        //visit array to mark visited ones

        char visit[256];

        bool x = true;

        q.push(start);

        visit[0] = start;

        bool check = true;

        //Loop while queue is not emty

        while (!q.empty()){

            for(int i=0;i<no\_vertex;i++){

                if(arr[i][0] == q.front()){

                    check = false;

                    for(int j=1;j<no\_vertex;j++){

                        if (arr[i][j] != '-') {

                            for(int k=0;k<no\_vertex;k++){

                                if (visit[k] == arr[i][j]) {

                                    x = false;

                                }

                            }

                                if(x)

                                {

                                    visit[i] = arr[i][j];

                                    q.push(arr[i][j]);

                                }

                                x = true;

                        }

                        else

                            break;

                    }

                }

                if (i == no\_vertex || !check) {

                    check = true;

                    break;

                }

            }

                cout<<q.front()<<"->";

                if (q.front() == goal)

                    break;

                q.pop();

         }

    }

//dfs

    void DFS(char start, char goal){

        stack<char> q;

        char visit[256];

        bool x = true;

        q.push(start);

        visit[0] = start;

        bool check = true;

        while (!q.empty()){

            cout << q.top() << "->";

            if (q.top() == goal)

                break;

            for(int i=0;i<no\_vertex;i++){

                if(arr[i][0] == q.top()){

                    q.pop();

                    check = false;

                    for(int j=1;j<no\_vertex;j++){

                        if (arr[i][j] != '-') {

                            for(int k=0;k<no\_vertex;k++){

                                if (visit[k] == arr[i][j]) {

                                    x = false;

                                }

                            }

                                if(x)

                                {

                                    visit[i] = arr[i][j];

                                    q.push(arr[i][j]);

                                }

                                x = true;

                        }

                        else

                            break;

                    }

                }

                if (i == no\_vertex || !check) {

                    check = true;

                    break;

                }

            }

         }

    }

};

//Driver COde

int main(){

    Graph g(13);

    //Adding a vertex

    g.Addvertex('a');

    g.Addvertex('b');

    g.Addvertex('c');

    g.Addvertex('d');

    g.Addvertex('e');

    g.Addvertex('f');

    g.Addvertex('g');

    g.Addvertex('h');

    g.Addvertex('i');

    g.Addvertex('j');

    g.Addvertex('k');

    g.Addvertex('l');

    g.Addvertex('m');

    g.addedge('a','b');

    g.addedge('a','c');

    g.addedge('b','d');

    g.addedge('b','e');

    g.addedge('c','f');

    g.addedge('f','j');

    g.addedge('d','g');

    g.addedge('d','h');

    g.addedge('e','i');

    g.addedge('i','m');

    g.addedge('g','j');

    g.addedge('g','l');

    cout << "Adj List Graph\n";

    g.print();

    cout<<"\nBFS Path\n";

    cout << "BFS Path from a--->g\n";

    g.BFS('a','g');

     cout << "\n\nDFS Path from a--->g\n";

     g.DFS('a', 'g');

 }

**output**

**Text

Description automatically generated**

**Task-2**

**a)**

#include <iostream>

using namespace std;

//node for chainng of keys

struct node

{

    int key;

    node \*next;

};

//class hashtable

class HashTable

{

    node \*\*chain;

    int size;

public:

//constructor

    HashTable(int size = 10)

    {

        this->size = size;

        chain = new node \*[size];

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

        {

            chain[i] = NULL;

        }

    }

    //hash function for hashing

    int hashFun(int key)

    {

        return key % size;

    }

    //method to inserting keys

    void insert(int key)

    {

        int index = hashFun(key);

        node \*newNode = new node;

        newNode->key = key;

        newNode->next = NULL;

        node \*temp = chain[index];

        //if first index is NULL set to first

        if (temp == NULL)

        {

            chain[index] = newNode;

            return;

        }

        // move to the end of chained data

        while (temp->next != NULL)

        {

            temp = temp->next;

        }

        temp->next = newNode;

        return;

    }

    // method  to search a key from the chaining

    int search(int key)

    {

        int count = 0;

        node \*current;

        //traverse the full data where data matches return

        while (count < size)

        {

            current = chain[count];

            while (current != NULL)

            {

                if (key == current->key)

                {

                    cout << "Key is Found\n";

                    return count;

                }

                current = current->next;

            }

            count++;

        }

        cout << "Key is not Found\n";

        return -1;

    }

    //method to delete the key

    void Delete(int key)

    {

        int index = search(key);

    //if not foound

        if (index == -1)

        {

            cout << "Key does't exist\n";

            return;

        }

        node \*current = chain[index];

        node \*prev = NULL;

        //find the key and delete it

        while (current != NULL)

        {

            if (key == current->key)

            {

                break;

            }

            prev = current;

            current = current->next;

        }

        node \*temp = current;

        prev->next = temp->next;

        delete temp;

        temp = NULL;

    }

    //mrhtod to display the data

    void Display()

    {

        int count = 0;

        node \*current;

        while (count < size)

        {

            current = chain[count];

            cout << count << ": ";

            while (current != NULL)

            {

                cout << current->key;

                current = current->next;

                if (current)

                    cout << "->";

            }

            cout << endl;

            count++;

        }

    }

};

//driver code

int main()

{

    HashTable h(5);

    int ch = 0, elem = 0;

    while (true)

    {

        cout << "1-Insert\n2-Delete\n3-Search\n4-Display\n5-Exit\nEnter Choice: ";

        cin >> ch;

        switch (ch)

        {

        case 1:

            cout << "Enter Element: ";

            cin >> elem;

            h.insert(elem);

            break;

        case 2:

            cout << "Enter Element: ";

            cin >> elem;

            h.Delete(elem);

            break;

        case 3:

            cout << "Enter Element: ";

            cin >> elem;

            h.search(elem);

            break;

        case 4:

            h.Display();

            break;

        case 5:

            exit(1);

            break;

        default:

            break;

        }

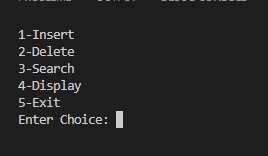
        system("pause");

        system("cls");

    }

}

**Output**

Text

Description automatically generated

**b) Linear Probing**

#include<iostream>

//hash table class

class HashTable{

    int \*arr;

    int size = 0;

public:

    //Constructor

    HashTable(int size = 15){

        this->size = size;

        arr = new int[size];

        for(int i=0;i<size;i++){

            arr[i] = NULL;

        }

    }

    //hash function

    int hash(int key){

        return (key % size);

    }

    //insert function

    void insert(int key){

        int index = hash(key);

        int count = 0;

            //loop till empty cell not found

            while(arr[index]!=NULL){

                index++;

                count++;

                if(index == this->size){

                    index = 0;

                }

                //return if no space left

                if(size<=count){

                    std::cout<<"No space left\n";

                return;

                }

            }

            //inserting a key

        arr[index] = key;

    }

    //method to search

    int search(int key){

        int index = hash(key);

        int count = 0;

        //find the key until key not found

        while (arr[index]!=key && count<= size){

            index++;

            count++;

            if(index == size){

                index = 0;

            }

            //if key not found return -1

            if(hash(key) == index)

            return -1;

        }

        return index;

    }

    //Method to found

    void Display(){

        std::cout<<"Hash Table\n";

        for(int i=0;i<size;i++){

            std::cout<<i<<"->"<<arr[i]<<std::endl;

        }

    }

    //method to delete key

    void DeleteKey(int key){

        int index = hash(key);

        int found = index;

        int freespace = index;

        while (arr[index]!=key){

            index++;

            if (index >= size)

                index = 0;

            if(key == arr[index]){

                arr[index] = NULL;

                found = index;

                freespace = index;

                // to check others index are at their pos if not move them to their pos

                for(int i=found; i != found-1;i++){

                    if(i==size)

                      i = 0;

                    if(hash(arr[i])!=i){

                        std::swap(arr[freespace],arr[i]);

                        freespace = i;

                    }

                }

        std::cout<<"Key is Deleted\n";

                return;

            }

        }

        std::cout<<"Key not found\n";

    }

};

//Driver code

int main(){

    HashTable ht(15);

    //inserting keys

    ht.insert(17);

    ht.insert(26);

    ht.insert(15);

    ht.insert(9);

    ht.insert(11);

    ht.insert(43);

    ht.insert(75);

    ht.insert(19);

    ht.insert(35);

    ht.insert(45);

    ht.insert(55);

    ht.insert(9);

    ht.insert(10);

    ht.insert(21);

    ht.insert(61);

    ht.insert(23);

    ht.Display();

    std::cout<<std::endl;

    ht.DeleteKey(10);

    std::cout<<"after deleting"<<std::endl;

    ht.Display();

    std::cout<< ht.search(17)<<std::endl;

}

**Text

Description automatically generated**

**Text

Description automatically generated**

**c) Quadratic probing**

#include<iostream>

//hash table

class HashTable{

    int \*arr;

    int size = 0;

    int count = 0;

public:

    //constructor

    HashTable(int size = 15){

        this->size = size;

        arr = new int[size];

        for(int i=0;i<size;i++){

            arr[i] = NULL;

        }

    }

    //hash function

    int hash(int key,int i, char op){

        if(op == '+')

        return (key + (i\*i))%size;

        return (key - (i\*i))%size;

    }

    //method to insert key

    void insert(int key){

        if(count>=size){

            std::cout<<"No space to add\n";

            return;

        }

        int i=0;

        int decide = 0;

        int index = hash(key,i,'+');

    // loop until empty not found

        while (arr[index] != 0){

            decide++;

            if(decide%2==1){

            index = hash(key,i,'-');

            i++;

            }

            else

            index = hash(key,i,'+');

        }

        arr[index] = key;

        count++;

    }

    //method to search key

    int search(int key){

        int i =0;

        int decide = 0;

        int index = hash(key,i,'+');

        int terminate = 0;

        //loop till not found

        while (arr[index]!=key && terminate <= size){

            decide++;

            if(decide%2==1){

            index = hash(key,i,'-');

            i++;

            }

            else

            index = hash(key,i,'+');

            terminate++;

        }

        if(arr[index] == key)

        return index;

        return -1;

    }

    //method to display

    void Display(){

        std::cout<<"Hash Table\n";

        for(int i=0;i<size;i++){

            std::cout<<i<<"->"<<arr[i]<<std::endl;

        }

    }

    //delete key

    void DeleteKey(int key){

        int index = search(key);

        if(index  ==-1){

           std::cout<<"Key not Found\n";

           return;

        }

        std::cout<<"Key is Deleted\n";

        arr[index] = 0;

    }

};

//driver code

int main(){

    HashTable ht(15);

    ht.insert(17);

    ht.insert(26);

    ht.insert(15);

    ht.insert(9);

    ht.insert(11);

    ht.insert(43);

    ht.insert(75);

    ht.insert(19);

    ht.insert(35);

    ht.insert(45);

    ht.insert(55);

    ht.insert(9);

    ht.insert(10);

    ht.insert(21);

    ht.insert(61);

    ht.insert(23);

    ht.Display();

    std::cout<<std::endl;

    ht.DeleteKey(35);

    std::cout<<"After Deleting"<<std::endl;

    ht.Display();

    std::cout<< ht.search(21)<<std::endl

}

**Output**

**Text

Description automatically generated**

**Text

Description automatically generated**