ADJACENCY LIST MATRIX :

package day21;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class task1 {

public static void main(String args[])

{

Scanner s = new Scanner(System.***in***);

int v , e , sn , dn;

System.***out***.println("enter the No of vertex :");

v=s.nextInt();

System.***out***.println("enter the no of edges :");

e = s.nextInt();

List<List<Integer>> graph = new ArrayList<List<Integer>>();

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

{

graph.add(new ArrayList<Integer>());

}

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

{

sn = s.nextInt();

dn = s.nextInt();

//int wt = s.nextInt();

graph.get(sn).add(dn); // Add edge from dn to sn

graph.get(dn).add(sn); // If it's an undirected graph, add reverse edge

}

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

{

System.***out***.println(i);

for(int neigh : graph.get(i))

{

System.***out***.println("-->"+neigh);

}

}

}

}

package day21;

import java.util.\*;

class Graph{

static class Edge{

int dest;

int weight;

public Edge(int dest, int weight) {

this.dest = dest;

this.weight = weight;

}

}

int vertex;

Map<Integer,List<Edge>> adj;

Graph( int vertex){

this.vertex=vertex;

this.adj=new HashMap<>();

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

adj.put(i, new LinkedList<>());

}

}

public void addEdge(int sr, int ds, int weight) {

adj.get(sr).add(new Edge(ds,weight));

}

public void printGraph() {

for(int node : adj.keySet()) {

System.***out***.print("node "+node + "->");

for(Edge edge : adj.get(node)) {

System.***out***.print("{"+"Dest: "+edge.dest+" Weight: "+edge.weight+"}");

}

System.***out***.println();

}

}

}

public class task3 {

public static void main(String[] args) {

Graph g=new Graph(5);

g.addEdge(0,1,4);

g.addEdge(0,2,1);

g.addEdge(2,3,1);

g.addEdge(2,1,3);

g.addEdge(3,4,7);

g.printGraph();

}

}

node 0->{Dest: 1 Weight: 4}{Dest: 2 Weight: 1}

node 1->

node 2->{Dest: 3 Weight: 1}{Dest: 1 Weight: 3}

node 3->{Dest: 4 Weight: 7}

node 4->

**BFS :**

package day21;

import java.util.\*;

class GraphBfs

{

int ver;

LinkedList<Integer> adj[];

public GraphBfs(int ver)

{

this.ver=ver;

this.adj= new LinkedList[ver+1];

for(int i=1;i<=ver;i++)

{

adj[i] = new LinkedList();

}

}

public void addEdge(int src , int des)

{

adj[src].add(des);

adj[des].add(src);

}

public void BFS(int start)

{

boolean visted[] = new boolean[ver+1];

System.***out***.println(Arrays.*toString*(visted));

Queue<Integer> q = new LinkedList();

visted[start]=true;

q.offer(start);

while(!q.isEmpty())

{

int node = q.poll();

System.***out***.println(node);

for(int n : adj[node])

{

if(!visted[n])

{

visted[n]=true;

q.offer(n);

}

}

}

}

}

public class task4 {

public static void main(String args[])

{

GraphBfs g = new GraphBfs(6);

g.addEdge(1,2);

g.addEdge(1,3);

g.addEdge(2,4);

g.addEdge(2,5);

g.addEdge(3,6);

g.BFS(1);

}

}

**DFS :**

package day21;

import java.util.Iterator;

import java.util.LinkedList;

class GraphDfs

{

int ver;

LinkedList<Integer> adj[];

public GraphDfs(int ver)

{

this.ver=ver;

this.adj= new LinkedList[ver+1];

for(int i=1;i<=ver;i++)

{

adj[i] = new LinkedList();

}

}

void addEdges(int src , int des)

{

adj[src].add(des);

}

void DFS(int start)

{

boolean visted[] = new boolean[ver+1];

Iterator<Integer> it = adj[start].listIterator();

while(it.hasNext())

{

visted[start]=true;

int node=it.next();

System.***out***.println(node);

if(!visted[node])

{

DFS(node);

//System.out.println(node);

}

}

}

}

public class task5 {

public static void main(String args[])

{

GraphDfs g = new GraphDfs(4);

g.addEdges(1,2);

g.addEdges(1,3);

g.addEdges(2,3);

g.addEdges(3,4);

g.DFS(2);

}

}

HASHING :

 Hashing is a technique used to map data in to fixed size memory location using a hash function.

 A hash function converts a input key into a index within a fixed size array called a hash table.

Best use case : fast insertion , deletion , search 🡪 O(1)

Main use case of hashing :

* Fast look up
* Efficient space usage .
* Password storage and catching the data bases .

HASH FOR CHARCHER .

public class task6 {

public static int hashChar(char ch , int pos)

{

static final int P = 31 ;

static final long M=100000007;

int val = ch-'a'+1;

long hash = (val\*(long) Math.*pow*(P, pos))%M;

}

public static void main(String args[])

{

char ch = 'c';

int pos = 2;

System.***out***.println(*hashChar*(ch,pos));

}

}

**Hash funt of string uses**

1.uses the each char ascii or Unicode value

2.apply some math formula to generate the unique numeric key

3.java provide the hash code for the string but less use the custom .

**Simple polynomial rolling hash map**

**H(s)=(s[0]+s[1]….s[n]\*p[n])%m**

package day21;

public class task7 { // Class names should follow PascalCase

static final int ***P*** = 31;

static final long ***M*** = 100000007L;

public static int compute(String s) {

long hash = 0;

long pow = 1;

for (char ch : s.toCharArray()) {

int val = ch - 'a' + 1;

hash = (hash + val \* pow) % ***M***;

pow = (pow \* ***P***) % ***M***;

}

return (int) hash;

}

public static void main(String args[]) {

String str1 = "hello";

String str2 = "world";

System.***out***.println(*compute*(str1));

System.***out***.println(*compute*(str2));

}

}

14222002

4069362

**Collision occur when to key map to the same key .**

Technique :

1.chain 🡪 using linked list

2.open address 🡪 linear , double hashing , quarter

**Insert : collision occurs :**

package day21;

class HashTable

{

static final int ***size*** =10;

Integer table[];

public HashTable()

{

table = new Integer[***size***];

}

public void insert(int key)

{

int index =hashFunc(key);

table[index]=key;

}

public int hashFunc(int key)

{

return key%***size***;

}

public void display()

{

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

{

System.***out***.println("index"+i + "-->"+table[i]);

}

}

}

public class task8 {

public static void main(String args[]) {

HashTable ht = new HashTable();

ht.insert(10);

ht.insert(20);

ht.insert(30);

ht.insert(46);

ht.insert(58);

ht.display();

}

}

index0-->30

index1-->null

index2-->null

index3-->null

index4-->null

index5-->null

index6-->46

index7-->null

index8-->58

index9-->null

HASH CHAINING :

import java.util.LinkedList;

import java.util.LinkedList;

class HashTable1{

static final int ***size*** = 10;

LinkedList<Integer> table[];

public HashTable1() {

table = new LinkedList[***size***];

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

table[i] = new LinkedList();

}

}

public void insert(int data) {

int index = hashFunc(data);

table[index].add(data);

}

public int hashFunc(int data) {

return data % ***size***;

}

public void display() {

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

System.***out***.println(i + " --" + table[i]);

}

}

public boolean search(int data) {

int index = hashFunc(data);

return table[index].contains(data);

}

public void delete(int data) {

int index = hashFunc(data);

table[index].remove((Integer)data);

}

}

public class task9{

public static void main(String[] args) {

HashTable1 ht = new HashTable1();

ht.insert(10);

ht.insert(20);

ht.insert(55);

ht.insert(40);

ht.insert(254);

ht.display();

System.***out***.println(ht.search(10)); // override by the new value so false

System.***out***.println(ht.search(40));

System.***out***.println(ht.search(55));

ht.delete(55);

ht.display();

}

}

0 --[10, 20, 40]

1 --[]

2 --[]

3 --[]

4 --[254]

5 --[55]

6 --[]

7 --[]

8 --[]

9 --[]

true

true

true

0 --[10, 20, 40]

1 --[]

2 --[]

3 --[]

4 --[254]

5 --[]

6 --[]

7 --[]

8 --[]

9 --[]

LINEAR PROBING :

package day21;

class HashTable2{

static final int ***size*** = 10;

Integer table[];

public HashTable2() {

table = new Integer[***size***];

}

public void insert(int data) {

int index = hashFunc(data);

int originalIndex = index;

while(table[index] != null) {

index = (index + 1) % ***size***;

if(index == originalIndex) {

System.***out***.println("Hash Table is Filled");

return;

}

}

table[index] = data;

}

public int hashFunc(int data) {

return data % ***size***;

}

public void display() {

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

System.***out***.println(i + " --" + table[i]);

}

}

public boolean search(int data) {

int index = hashFunc(data);

int org = index;

while(table[index]!=null)

{

if(table[index]==data)

{

return true ;

}

index = (index+1) %***size***;

if(index==org)

{break;}

}

return false;

}

public void delete(int data) {

int index = hashFunc(data);

int org = index;

while(table[index]!=null)

{

if(table[index] ==data)

{

table[index]=null;

}

index = (index+1)%***size***;

if(org==index)

{break;}

}

}

}

public class task10 {

public static void main(String[] args) {

HashTable2 ht = new HashTable2();

ht.insert(10);

ht.insert(20);

ht.insert(55);

ht.insert(40);

ht.insert(254);

ht.display();

System.***out***.println(ht.search(100));

System.***out***.println(ht.search(40));

System.***out***.println(ht.search(55));

ht.delete(20);

ht.display();

}

}

0 --10

1 --20

2 --40

3 --null

4 --254

5 --55

6 --null

7 --null

8 --null

9 --null

false

true

true

0 --10

1 --null

2 --40

3 --null

4 --254

5 --55

6 --null

7 --null

8 --null

9 --null

Quadratic probing :

package day21;

class HashTable3{

static final int ***size*** = 10;

Integer table[];

public HashTable3() {

table = new Integer[***size***];

}

public void insert(int data) {

int index = hashFunc(data);

int i=1;

while(table[index] != null) {

index = (hashFunc(data) + i \* i) % ***size***;

i++;

if(i>=***size***) {

System.***out***.println("Hash Table is Filled");

return;

}

}

table[index]=data;

}

public int hashFunc(int data) {

return data % ***size***;

}

public void display() {

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

System.***out***.println(i + " --" + table[i]);

}

}

public boolean search(int data) {

int index = hashFunc(data);

int i=1;

while(table[index]!=null)

{

if(table[index]==data)

{

return true;

}

index = (hashFunc(data) + i \* i) % ***size***;

i++;

if(i>=***size***)

{break;}

}

return false;

}

public void delete(int data) {

int index = hashFunc(data);

int i=1;

while(table[index]!=null)

{

if(table[index]==data)

{

table[index]=null;

return;

}

index = (hashFunc(data) + i \* i) % ***size***;

i++;

if(i>=***size***)

{break;}

}

}

}

public class task11 {

public static void main(String[] args) {

HashTable3 ht = new HashTable3();

ht.insert(10);

ht.insert(20);

ht.insert(55);

ht.insert(40);

ht.insert(254);

ht.display();

System.***out***.println(ht.search(100));

System.***out***.println(ht.search(40));

System.***out***.println(ht.search(55));

ht.delete(20);

ht.delete(55);

ht.display();

}

}

0 --10

1 --20

2 --null

3 --null

4 --40

5 --55

6 --null

7 --null

8 --254

9 --null

false

true

true

0 --10

1 --null

2 --null

3 --null

4 --40

5 --null

6 --null

7 --null

8 --254

9 --null

DOUBLE HASHING :

package day21;

class HashTable4{

static final int ***size*** = 10;

static final int ***P*** = 7;

Integer table[];

public HashTable4() {

table = new Integer[***size***];

}

int hash2(int data)

{

return ***P*** - data % ***P***;

}

public void insert(int data) {

int index = hashFunc(data);

int i=1;

while(table[index]!=null)

{

index= (hashFunc(data)+i\*hash2(data))%***size***;

i++;

if(i>=***size***)

{

System.***out***.println("table full");

}

}

table[index] = data;

}

public int hashFunc(int data) {

return data % ***size***;

}

public void display() {

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

System.***out***.println(i + " --" + table[i]);

}

}

public boolean search(int data) {

int index = hashFunc(data);

return table[index] != null && table[index] == data;

}

public void delete(int data) {

int index = hashFunc(data);

if(table[index] != null && table[index] == data){

table[index] = null;

}

}

}

public class task12 {

public static void main(String[] args) {

HashTable4 ht = new HashTable4();

ht.insert(10);

ht.insert(20);

ht.insert(55);

ht.insert(40);

ht.insert(254);

ht.display();

}

}

0 --10

1 --20

2 --40

3 --null

4 --254

5 --55

6 --null

7 --null

8 --null

9 --null