Q1:

/\*\*

\*

\* @author Thaycacac

\*/

/\*Table content to search\*/

||\_\_\_\_ADD\_\_\_\_\_\_//add last

|| ||\_\_\_\_\_\_//add first

|| ||\_\_\_\_\_\_//add many

|| ||\_\_\_\_\_\_//add if age > 4

|| ||\_\_\_\_\_\_//add element have max age

||

||\_\_\_INSERT\_\_\_\_//insert position k

|| ||\_\_\_\_//insert after position k

||

||\_\_\_SEARCH\_\_\_\_//search node

|| ||\_\_\_\_//search node with string

|| ||\_\_\_\_//search node with integer/double

||

||\_\_\_DELETE\_\_\_\_//delete node

|| ||\_\_\_\_//delete all

|| ||\_\_\_\_//delete at position k

|| ||\_\_\_\_//delete first no age < 6

|| ||\_\_\_\_//delete a node after position k

|| ||\_\_\_\_//delete node after 2 node have age < 9

|| ||\_\_\_\_//delete node thirdth have age < 6

|| ||\_\_\_\_//delete node first after node have position k

|| ||\_\_\_\_//delete node after node have price > xPrice

|| ||\_\_\_\_//delete 2 node last have age > 5

|| ||\_\_\_\_//delete second biggest

||

||\_\_\_SORT\_\_\_\_\_\_//sort by string

|| ||\_\_\_\_\_\_//sort by integer/ double

|| ||\_\_\_\_\_\_//sort by for

|| ||\_\_\_\_\_\_//sort 3 first element

||

||\_\_\_SWAP\_\_\_\_\_\_//swap min max

|| ||\_\_\_\_\_\_//swap node max second with node min first

||

||\_\_\_GET\_\_\_\_\_\_\_//get node at index k

|| ||\_\_\_\_\_\_\_//get node max

||

||\_\_\_ORTHER\_\_\_\_//traverse

|| ||\_\_\_\_//replace a node

|| ||\_\_\_\_//count number of node

|| ||\_\_\_\_//reverse list

|| ||\_\_\_\_//append another list

|| ||\_\_\_\_//change name first

//add last

public void addLast(Person x){

Node q = new Node(x);

if(isEmpty()){head=tail=q;return;}

tail.next=q;tail=q;

}

//add first

public void addFirst(Person x){

Node p = new Node(x);

if(isEmpty()) {head = tail = p;}

else {p.next = head; head = p;}

}

//add many

public void addMany(String [] a, int [] b){

int n,i; n=a.length;

for(i=0;i<n;i++) addLast(new Person(a[i],b[i]));

}

//add if age > 4

public void addLastCondition(){

Node p=head;

while(p!=null){

if(p.info.age>4) h.addLast(p.info);

p=p.next;

}

}

//add element have max age

public void addMaxAgeInH(){

Person k = getMaxColor();

Node p = head;

while(p != null){

if(p.info.age == k.age) h.addLast(p.info);

p = p.next;

}

}

//insert position k

public void insertPositionK(Person x, int position) {

if (isEmpty()) head = tail = new Node(x);

int count = 1;

Node p = head;

while (p!= null && count < position) {

p = p.next;

count ++;

}

Node temp = p.next;

p.next = new Node(x, temp);

}

//insert after position k

public void addAfterPositionK(int k, Person c){

Node p = new Node(c);

if(k == -1){

addFirst(c); return;

}

int count = 0;

Node p1 = head;

while(p1 != null && count < k){

p1 = p1.next; count++;

}

if(p1.next == null && count == k){

addLast(c); return;

}

p.next = p1.next;

p1.next = p;

}

//search node

public Node search(int x{

Node p = head;

while(p != null && p.info != x) p = p.next;

return p;

}

//search node with string

public Node search(String colorName){

Node p = head;

while(p != null){

if(p.info.name.equalsIgnoreCase(colorName)){

break;

}

p = p.next;

}

return p;

}

//search node with integer/double

public Node search(int xAge){

Node p=head;

while(p!=null){

if(p.info.age == xAge) return(p);

p=p.next;

}

return(null);

}

//delete node

public void dele(Node q){

Node f,p; f=null;p=head;

while(p!=null){

if(p==q) break;

f=p;p=p.next;

}

if(p==null) return;

if(f==null){

head=head.next;

if(head==null) tail=null;

return;

}

f.next=p.next;

if(f.next==null) tail=f;

}

//delete all

public void deleAll(int xAge){

Node q;

while(true){

q = searchByAge(xAge);

if(q==null) break;

dele(q);

}

}

//delete at position k

public void deleteAt(int k) {

if (isEmpty()) return;

if (k == 0) {//check if node is head

Node p = head;

head = head.next;

p.next = null;

} else {

Node p = getNode(k);//get node position k

if (p == null) return;

Node q = getNode(k - 1);//q is node before of p

// Remove p

q.next = p.next;

p.next = null;

if (p == tail) tail = q;

}

}

//delete first no age < 6

public void deleteFirstCondition(){

Node p = head;

while(p != null){

if(p.info.age<6) dele(p);

p=p.next;

}

}

//delete a node after position k

public void deleteAfterPosK(int k){

if(isEmpty()) return;

//remove head

if(k == -1) {

Node p = head;

head = head.next;

p.next = null;

}else {

Node p = getNode(k + 1);

if(p == null) return;

Node q = getNode(k);

//remove p

q.next = p.next;

p.next = null;

if(p == tail) tail = q;

}

}

//delete node after 2 node have age < 9

public void deleteAfterTwoNodeAgeSmallerNine(){

Node p = head;

while(p != null && p.info.age >= 9){

p = p.next;

}

if(p == null && p.next == null){

return;

}

remove(p.next.next);

}

//delete node thirdth have age < 6

public void removeThirthAgeSmallerSix(){

Node p = head;

int count = 0;

while(p != null){

if(p.info.age < 6 && count != 3) count ++;

else if(p.info.age < 6 && count == 3) break;

p = p.next;

}

remove(p);

}

//delete node first after node have position k

public void deleteFirstAfterPosition(int k){

Node p = getNode(k);

if(p == null || p.next == null){//if p is tail then donothing

return;

}

Node q = p.next;//if p is before tail then remove tail

if(q.next == null){

tail = p;

p.next = null;

}

else{

p.next = q.next;

q.next = null;

}

}

//delete node after node have price > xPrice

public void deleteAfterCondition(double xPrice){

Node p = head;

while (p != null && p < xPrice) {

p = p.next;

}

if (p == null || p.next == null) return;

Node q = p.next;

if (q == tail) tail = p;

p.next = q.next;

q.next = null;

}

//delete 2 node last have age > 5

public void removeTwoLastNodeCondition(){

int c = 0; int sz = size();

for(int i = sz - 1; i >= 0; i--){

Node p = getNode(i);

if(p.info.age > 5){

c++;

remove(p);

if(c >= 2) break;

}

}

}

//delete second biggest

public void removeSecond() {

Person firstMax = getMaxColor();

if(firstMax == null) return;

int n = size();

if(n <= 1 ) return;

int imax = 0; Node p = head;

while(p != null && p.info.age == firstMax.age){

imax++;

p = p.next;

}

// Find second max starting from imax

Person secondMax = (p != null) ? p.info : null;

for (int i = imax + 1; i < n; i++) {

Node pi = getNode(i);

if(pi.info.age > secondMax.age && pi.info.age != firstMax.age){

imax = i;

secondMax = pi.info;

}

}

if(imax < n) remove(imax);

}

//sort by string

public void sort(){

Node pi,pj; Person x;

pi=head;

while(pi!=null){

pj=pi.next;

while(pj!=null){

if(pj.info.name.compareTo(pi.info.name)<0){

x=pi.info;pi.info=pj.info;pj.info=x;

}

pj=pj.next;

}

pi=pi.next;

}

}

//sort by integer/ double

public void sort() {

Node pi,pj; Car x;

pi=head;

while(pi!=null){

pj=pi.next;

while(pj!=null){

if(pj.info.price<pi.info.price){

x=pi.info;pi.info=pj.info;pj.info=x;

}

pj=pj.next;

}

pi=pi.next;

}

}

//sort by for

public void sortByFor(){

int n = (size() > 3) ? 3 : size();

int n = size();

// for(int i = n-3; i < n; i++) //last 3 element

// for(int j = i+1; j < n; j++) //first 3 element

for(int i = 0; i < n - 1; i++){

for(int j = i+1; j < n; j++){

Node pi = getNode(i);

Node pj = getNode(j);

if(pi.info.name.compareToIgnoreCase(pj.info.name) > 0){

Person temp = pi.info;

pi.info = pj.info;

pj.info = temp;

}

}

}

}

//sort 3 first element

public void sortThird() {

Node pi, pj; pi = head; int count = 0;

while(pi != null) {

count++; pj = pi.next; int count1 = 0;

while(pj != null) {

count1++;

if(pi.info.name.compareToIgnoreCase(pj.info.name) < 0) {

Person t = pi.info; pi.info = pj.info;pj.info = t;

}

pj = pj.next; if(count1 == 3 - count) break;

}

pi = pi.next; if(count == 2) break;

}

}

//swap min max

public void swapMinMax(){

Node min = getMin();

Node max = getMax();

Person t = min.info;

min.info = max.info;

max.info = t;

}

//swap node max second with node min first

public void swapMax2Min1(){

Node max = getMax();

Node min = getMin();

Node p = head;

int count = 0;

while(p != null){

if(p.info.age == max.info.age) count++;

if(count == 2) break;

p = p.next;

}

Person temp;

temp = p.info;

p.info = min.info;

min.info= temp;

}

//get node at index k

public Node getNode(int k) {

int c = 0;

Node p = head;

while (p != null && c < k) {

p = p.next;

c++;

}

return p;

}

//get node max

public Person getMaxPerson(){

if(isEmpty()) return null;

Person max = head.info;

Node p = head;

while( p != null){

if(p.info.age > max.age) max = p.info;

p = p.next;

}

return max;

}

//traverse

public void traverse(){

MyList h=new MyList();

Node p=head;Person x;

while(p!=null)

{x=p.info;h.addFirst(x);

p=p.next;

}

head=h.head;tail=h.tail;

}

//replace a node

public void replace{

Node p = head;

while(p != null){

if(p.info.name.equals("xName")){// xName was given

break;

}

p = p.next;

}

if(p != null){

p.info.name = yName;// yName was given

}

}

//count number of node

int count(Node p)

{if(p==null) return(0);

int k,h,r;

k = count(p.left);

h = count(p.right);

r = k+h+1;

return(r);

}

//reverse list

public void reverse() {

int n = size(), i, j;

for (i = 0, j = n - 1; i < j; i++, j--) {

Node pi = getNode(i);// create a node = getnode index i

Node pj = getNode(j);// create a node = getnode index j

Person temp = pi.info;// Note: change value of node, not change node

pi.info = pj.info;

pj.info = temp;

}

}

//append another list

void appendAnotherList(MyList h){

Node p=h.head;

while(p!=null)

{addLast(p.info);

p=p.next;

}

}

//change name first

void changeNameFirst(){

Node p=head;

while(p!=null)

{if(p.info.name.equals("C6")) {p.info.name="CX";break;}

p=p.next;

}

}

}

Q2

/\*\*

\*

\* @author Thaycacac

\*/

||\_\_TRAVERSAL\_\_//breadth-first traversal

|| ||\_\_\_\_\_\_//preorder

|| ||\_\_\_\_\_\_//postorder

|| ||\_\_\_\_\_\_//inorder

|| ||\_\_\_\_\_\_//use BFS change second node have age >=5 to age = 10

|| ||\_\_\_\_\_\_//preorder with condition: 3 <= price <= 5

||

||\_\_\_INSERT\_\_\_\_//insert by string

|| ||\_\_\_\_//insert by integer/double

|| ||\_\_\_\_//insert have age > 4 use BTF

||

||\_\_\_SEARCH\_\_\_\_//search

|| ||\_\_\_\_//search by string

|| ||\_\_\_\_//search by integer/double

||

||\_\_\_COUNT\_\_\_\_\_//count node in tree

|| ||\_\_\_\_//count node have 1 child

|| ||\_\_\_\_//count node have exactly 2 child

|| ||\_\_\_\_//count height of tree

||

||\_\_\_DELETE\_\_\_\_//delete by copy integer/double

|| ||\_\_\_\_\_\_//delete by copy string

|| ||\_\_\_\_\_\_//delete by node p

||

||\_\_\_BALANCE\_\_\_//balance tree

|| ||\_\_\_\_\_\_//balance simple array list

||

||\_\_\_ROTATE\_\_\_\_//rotate left

|| ||\_\_\_\_\_\_\_//rotate right

|| ||\_\_\_\_\_\_\_//rotate any node

|| ||\_\_\_\_\_\_\_//rotate right root

||

||\_\_\_GET\_\_\_\_\_\_\_//get node father

|| ||\_\_\_\_//get node by string

|| ||\_\_\_\_//get node by integer/double

||\_\_\_OTHER\_\_\_\_\_//calculate level of node

|| ||\_\_\_\_\_//calculate factor

|| ||\_\_\_\_\_//copy all node to tree by inorder traversal

|| ||\_\_\_\_\_//Calculate balance factor

|| ||\_\_\_\_\_//Calculate level all node

|| ||\_\_\_\_\_//balance a binary search tree

//breadth-first traversal

public void bfs(Node p){ //input root

if(p == null) return;

MyQueue m = new MyQueue();

m.enqueue(p);

while(!m.isEmpty()){

Node q = (Node)m.dequeue();// get node

if(q.left != null){ // if lever still hava node left

m.enqueue(q.left);

}

if(q.right != null){ //if lever still hava node right

m.enqueue(q.right);

}

visit(q);// traversal them

}

}

//preorder

public void preorder(Node p){

if(p == null){

return;

}

visit(p);

preorder(p.left);

preorder(p.right);

}

//postorder

public void postorder(Node p){

if(p == null){

return;

}

postorder(p.left);

postorder(p.right);

visit(p);

}

//inorder

public void inorder(Node p){

if(p == null){

return;

}

inorder(p.left);

visit(p);

inorder(p.right);

}

//use BFS change second node have age >=5 to age = 10

public void changeNodeCondition(Node p){

if(p == null) return;

MyQueue m = new MyQueue();

m.enqueue(p);

while(!m.isEmpty()){

Node q = (Node)m.dequeue();

if(q.left != null){

m.enqueue(q.left);

}

if(q.right != null){

m.enqueue(q.right);

}

if(q.info.age >= 5){ //maybe or not

c++;

if(c == 2){

q.info.age = 10;

break;

}

}

}

}

//preorder with condition: 3 <= price <= 5

void preOrder2(Node p, RandomAccessFile f) throws Exception{

if(p==null) return;

if(p.info.price>=3&&p.info.price<=5){

fvisit(p,f);

}

preOrder2(p.left,f);

preOrder2(p.right,f);

}

//insert by string

public void insert(Person x){

Node p = new Node(x);

if(isEmpty()){root = p; return;}

Node f = null;

Node q = root;

while(q != null){

if(q.info.name.equals(x.name)){

System.out.println("Insertion failed, duplicated key");

return;

}

else if(q.info.name.compareToIgnoreCase(x.name) > 0){f = q; q = q.left;}

else{f = q; q = q.right;}

}

if(f.info.name.compareToIgnoreCase(x.name) > 0) f.left = p;

else f.right = p;

}

//insert by integer/double

public void insert(Car x){

Node q=new Node(x);

if(isEmpty())

{root=q;

return;

}

Node f,p;

f=null;p=root;

while(p!=null)

{if(p.info.price == x.price)

{System.out.println("The key " + x.price + " already exists, no insertion");

return;

}

f=p;

if(x.price < p.info.price) p=p.left; else p=p.right;

}

if(x.price< f.info.price) f.left=q; else f.right=q;

}

//insert have age > 4 use BTF

public void breadthModifier(){

if(root == null){ return; //change

MyQueue m = new MyQueue();

m.enqueue(root);

while(!m.isEmpty()){

Node q = (Node)m.dequeue();

if(q.left != null){

m.enqueue(q.left);

}

if(q.right != null){

m.enqueue(q.right);

}

if(q.info.age > 4){

h.insert(q.info);

}

}

}

}

//search

public Node search(Person x){

return search(root, x);

}

//search by string

public Node search(Node p, Person x){

if(p == null){

return null;

}

if(p.info.name.equals.x.name){

return p;

}

else if(p.info.name.compareToIgnoreCase(x.name) > 0){

return search(p.left,x);

}

else{

return search(p.right,x);

}

}

//search by integer/double

public Node search(Node p, int key){

if(p == null) return null;

if(p.info == key) return p;

else if(p.info > key) return search(p.left, key);

else return search(p.right, key);

}

//count node in tree

public int count(Node p){

if(p==null) return(0);

int k,h,r;

k = count(p.left);

h = count(p.right);

r = k+h+1;

return(r);

}

//count node have 1 child

int countModifer(Node p) {

int n = 0;

MyQueue m = new MyQueue();

m.enqueue(p);

while(!m.isEmpty()) {

Node q = (Node)m.dequeue();

if(q.left != null) m.enqueue(q.left);

if(q.right != null) m.enqueue(q.right);

// If q has only one child, increment c by 1

if(q.left == null && q.right != null) n++;

if(q.right == null && q.left != null) n++;

}

return n;

}

//count node have exactly 2 child

int countNodeCo2con(Node p) {

int n = 0;

MyQueue m = new MyQueue();

m.enqueue(p);

while(!m.isEmpty()) {

Node q = (Node)m.dequeue();

if(q.left != null) m.enqueue(q.left);

if(q.right != null) m.enqueue(q.right);

if(q.left != null && q.right != null) n ++;

}

return n;

}

//count height of tree

public int height(Node p) {

if (p == null) {

return 0;

}

int l = height(p.left) + 1;

int r = height(p.right) + 1;

return (l > r) ? l : r;

}

//delete by copy integer/double

public void deleByCopy(int xPrice) {

if (root == null) {

System.out.println(" The tree is empty, no deletion");

return;

}

Node f, p; // f will be the father of p

p = root;

f = null;

while (p != null) {

if (p.info.price == xPrice) {

break;//Found key x

}

if (xPrice < p.info.price) {

f = p;

p = p.left;

} else {

f = p;

p = p.right;

}

}

if (p == null) {

System.out.println(" The key " + xPrice + " does not exist, no deletion");

return;

}

if (p.left == null && p.right == null) // p is a leaf node

{

if (f == null) // The tree is one node

{

root = null;

} else {

if (f.left == p) {

f.left = null;

} else {

f.right = null;

}

}

return;

}

if (p.left != null && p.right == null) // p has only left child

{

if (f == null) // p is a root

{

root = p.left;

} else {

if (f.left == p) // p is a left child

{

f.left = p.left;

} else {

f.right = p.left;

}

}

return;

}

if (p.left == null && p.right != null) // p has only right child

{

if (f == null) // p is a root

{

root = p.right;

} else {

if (f.left == p) // p is aleft child

{

f.left = p.right;

} else {

f.right = p.right;

}

}

return;

}

if (p.left != null && p.right != null) // p has both left and right children

{

Node q, fr, rp; // p's key will be replaced by rp's one

fr = null;

q = p.left;

rp = q;

while (rp.right != null) {

fr = rp;

rp = rp.right; // Find the right most node on the left sub-tree

}

p.info = rp.info;

if (fr == null) // rp is just a left son of p

{

p.left = rp.left;

} else {

fr.right = rp.left;

}

}

}

//delete by copy string

void deleByCopy(String xName){

Node f,p;

f=null;p=root;

while(p!=null)

{if(p.info.name.equals(xName)) break;

f=p;

if(xName.compareTo(p.info.name)<0) p=p.left; else p=p.right;

}

if(p==null) return; // not found

// p is leaf node

if(p.left==null && p.right==null)

{if(f==null) // p=root

{root=null;

}

else

{if(p==f.left) f.left=null; f.right=null;

}

return;

}

// p has left child only

if(p.left!=null && p.right==null)

{if(f==null) // p=root

{root=p.left;

}

else

{if(p==f.left) f.left=p.left; f.right=p.left;

}

return;

}

// p has right child only

if(p.left==null && p.right!=null)

{if(f==null) // p=root

{root=p.right;

}

else

{if(p==f.left) f.left=p.right; f.right=p.right;

}

return;

}

// p has both 2 children

if(p.left!=null && p.right!=null)

{// find the right most node

Node q=p.left;

Node frp, rp; frp=null;rp=q;

while(rp.right!=null) {frp=rp;rp=rp.right;}

// rp is the right most node on the left child

p.info=rp.info;

if(frp==null) // rp=q

{p.left=q.left;

}

else

{

frp.right=rp.left;

}

}

}

//delete by node p

public void deleteByCopy(Node p) {

if (isEmpty()) {

return;

}

if(p == null){

System.out.println("Key does not exists, deletion failed");

return;

}

// Find Node f where f is father of p

Node f = null;

Node q = root;

while(q != p){

if(q.info.name.compareTo(p.info.name) > 0){ // Changed

f = q;

q = q.left;

}

else{

f = q;

q = q.right;

}

}

// 1. p is a leaf (no right and left child)

if (p.left == null && p.right == null) {

// a BST has a Node only

if (f == null) {

root = null;

}

else if (f.left == p) {

f.left = null;

}

else if(f.right == p){

f.right = null;

}

}

// 2. p has a left child only

else if(p.left != null && p.right == null){

if(f == null){// remove root

root = p.left;

}

else if(f.right == p){

f.right = p.left;

}

else if(f.left == p){

f.left = p.left;

}

}

// 3. p has a right child only

else if(p.right != null && p.left == null){

if(f == null){// remove root

root = p.right;

}

else if(f.right == p){

f.right = p.right;

}

else if(f.left == p){

f.left = p.right;

}

}

// 4. Both of right and left child

else if(p.left != null && p.right != null){

f = null;

Node rp = p.left;

while(rp.right != null){

f = rp;

rp = rp.right;

}

p.info = rp.info;

if(f == null){// rp has no right child

p.left = rp.left;

}

else{

f.right = rp.left;

}

}

}

//balance tree

public void balance(ArrayList a, int first, int last){

if(first > last) return;

int m = (first + last) /2;

Person x = ((Node)a.get(m)).info;

insert(x);

balance(a, first, m-1);

balance(a, m+1, last);

}

//balance simple array list

public void balance(Node p){

ArrayList a = new ArrayList();

buildArray(a, p);

int first = 0;

int last = a.size() - 1;

BSTree b = new BSTree(); //create new tree

b.balance(a, first, last);

root = b.root; //referen root to root b

}

//rotate left

public Node rotateLeft(Node p){//must be have node right

if(p.right == null){

return p;

}

Node q = p.right;

p.right = q.left;

q.left = p;

return q;

}

//rotate right

public Node rotateRight(Node p){

if(p.left == null){

return p;

}

Node q = p.left;

p.left = q.right;

q.right = p;

return q;

}

//rotate any node

public void rotateModifier(Node node){

Node nodeRotate = rotateToRight(node);

Node nodeFather = father(node.info.price);

if(nodeFather==null) root = nodeRotate;

else{

if(nodeFather.left==node) nodeFather.left = nodeRotate;

else nodeFather.right = node;

}

}

//rotate right root

Node rotateToRight(Node p){//root = rotateToRight(root)

if(p==null || p.left==null) return(p);

Node q=p.left;

p.left=q.right;

q.right=p;

return(q);

}

//get node father

Node father(int xPrice){

Node f,p;

f=null;p=root;

while(p!=null)

{if(p.info.price == xPrice) break;

f=p;

if(xPrice < p.info.price) p=p.left; else p=p.right;

}

return(f);

}

//get node by string

public Node getNode(String xName) {

Node p = root;

while (p != null) {

if (p.info.name.compareToIgnoreCase(xName) > 0) p = p.left;

else if (p.info.name.compareToIgnoreCase(xName) < 0) p = p.right;

return p;

}

return null;

}

//get node by integer/double

public Node getNode(int x) {

Node p = root;

while (p != null) {

if (p.info > x) p = p.left;

else if (p.info < x) p = p.right;

return p;

}

return null;

}

//calculate level of node

public void calLevel(Node p){

if(p == null){

return;

}

MyQueue m = new MyQueue();

m.enqueue(p);

p.level = 1;//first, leve = 1

while(!m.isEmpty()){

Node q = (Node)m.dequeue();

if(q.left != null){

q.left.level = q.level +1;

m.enqueue(q.left);

}

if(q.right != null){

q.right.level = q.level +1;

m.enqueue(q.right);

}

}

}

//calculate factor

public void calculateBalance(Node p){

if(p == null){

return;

}

MyQueue m = new MyQueue();

m.enqueue(p);

while(!m.isEmpty()){

Node q = (Node)m.dequeue();

if(q.left != null){

m.enqueue(q.left);

}

if(q.right != null){

m.enqueue(q.right);

}

q.bal = height(q.right) - height(q.left);

if(isAVL && q.bal < -1 || q.bal > 1){// De cho thuc hien nhieu

isAVL = false;

}

}

}

//copy all node to tree by inorder traversal

public void buildArray(ArrayList a, Node p){

if(p == null){

return;

}

buildArray(a, p.left);

a.add(p);

buildArray(a, p.right);

}

//Calculate balance factor

void calculateFactorBalance(RandomAccessFile f123) throws Exception{

boolean isAVL = true;

MyQueue q = new MyQueue();

q.enqueue(root);Node r;

while(!q.isEmpty())

{r = q.dequeue();

r.bal = height(r.right) - height(r.left);//int bal in class Node; // balance factor of the node p = height(p.right) - height(p.left)

if(r.bal>=2 || r.bal<=-2) isAVL = false;

if(r.left!=null) q.enqueue(r.left);

if(r.right!=null) q.enqueue(r.right);

}

breadthBal(root,f123);

if(!isAVL)

f123.writeBytes("\r\nThe tree is not an AVL tree\r\n");

else

f123.writeBytes("\r\nThe tree is an AVL tree\r\n");

}

public void breadthBal(Node p, RandomAccessFile f) throws Exception{//use for balance factor

if(p==null) return;

MyQueue q = new MyQueue();

q.enqueue(p); Node r;

while(!q.isEmpty())

{r = q.dequeue();

fvisitBal(r,f);

if(r.left != null) q.enqueue(r.left);

if(r.right != null) q.enqueue(r.right);

}

}

void fvisitBal(Node p, RandomAccessFile f) throws Exception

{if(p != null)

f.writeBytes("("+p.info.name+","+p.info.age+","+p.bal+") ");

}

//Calculate level all node

void calculateLevelAllNode(){

MyQueue q = new MyQueue();

if(isEmpty()) return;

root.level=1;

q.enqueue(root);Node r;

while(!q.isEmpty())

{r = q.dequeue();

if(r.left!=null) r.left.level = r.level +1;//level in class node

if(r.right!=null) r.right.level = r.level +1;

if(r.left!=null) q.enqueue(r.left);

if(r.right!=null) q.enqueue(r.right);

}

}

void fvisitLevel(Node p, RandomAccessFile f) throws Exception

{if(p != null)

f.writeBytes("("+p.info.name+","+p.info.age+","+p.level+") ");

}

//balance a binary search tree

void balance(){

ArrayList<Person> t;

t = new ArrayList<Person>();

inOrder(t,root);

int n = t.size();

clear();

balance(t,0,n-1);

}

void inOrder(ArrayList<Person> t, Node p){

if(p==null) return;

inOrder(t,p.left);

t.add(p.info);

inOrder(t,p.right);

}

void balance(ArrayList<Person> t, int i, int j)

{if(i>j) return;

int k=(i+j)/2;

insert(t.get(k));//insert person

balance(t,i,k-1);

balance(t,k+1,j);

}

Q3

/\*\*

\*

\* @author Thaycacac

\*/

||\_\_TRAVERSAL\_\_//breadth-first traversal

|| ||\_\_\_\_\_\_//dept-first traversal

|| ||\_\_\_\_\_\_//example breadth-first

||

||\_\_\_ALGORITHM\_//dijkstra

|| ||\_//euler cycle

||

||\_\_\_OTHER\_\_\_\_\_//count the connectivity parts

//breadth-first traversal

void breadth(boolean [] visited, int k, RandomAccessFile f) throws Exception{

GQueue q = new GQueue();

int r,i;

boolean [] enqueued = new boolean[20];

for(i=0;i<n;i++) enqueued[i]=false;

q.enqueue(k); enqueued[k]=true;

while(!q.isEmpty())

{r = q.dequeue();

if(!visited[r]) {fvisit(r,f);//modifier function fvisit

visited[r] = true;}

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

{if(!visited[i] && !enqueued[i] && a[r][i]>0) {q.enqueue(i);enqueued[i]=true;}

}

}

}

void breadth(int k, RandomAccessFile f) throws Exception{

boolean [] visited = new boolean[20];

int i;

for(i=0;i<n;i++) visited[i]=false;

breadth(visited,k,f);

for(i=0;i<n;i++) if(!visited[i]) breadth(visited,i,f);

}

//breadth first print degree with: A(4) E(3) F(3) G(2) I(3) B(2) C(1) H(2) D(1)

void fvisitDeg(int i, RandomAccessFile f) throws Exception{

f.writeBytes(" "+v[i]+"("+deg[i]+")");

}

//dept-first traversal

void depth(boolean [] visited,int k, RandomAccessFile f) throws Exception{

fvisit(k,f);visited[k]=true;

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

{if(!visited[i] && a[k][i]>0) depth(visited,i,f);

}

}

void depth(int k, RandomAccessFile f) throws Exception{

boolean [] visited = new boolean[20];

int i;

for(i=0;i<n;i++) visited[i]=false;

depth(visited,k,f);

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

if(!visited[i]) depth(visited,i,f);

}

//dijkstra

void dijkstra(int fro, int to, RandomAccessFile f) throws IOException {

boolean [] S = new boolean[n];

int [] d = new int[n];

int [] p = new int[n];

int INF = 999;

int i,j,k, t;

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

S[i] = false;

d[i] = a[fro][i];

p[i] = fro;

}

S[fro] = true;

while(true) {

// find d[k] = min {d[i}}

t = INF; k=-1;

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

if(S[i]==true) continue;

if(d[i]<t) {

t = d[i];

k = i;

}

}

if(k==-1) {

return;

}

// add k to S

S[k] = true;

if(k==to) break;

// update d[i] & p[i]

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

if(S[i]==true) continue;

if(d[i] > d[k] + a[k][i]) {

d[i] = d[k] + a[k][i];

p[i] = k;

}

}

}

// System.out.println("The shortest distance is " + d[to]);

i = to;

GStack s = new GStack();//store vertex

ArrayList points = new ArrayList();//store distance

while(true) {

s.push(i);

if(i==fro) break;

i = p[i];

}

//print line 1: vertex, line2 : distance

while(!s.isEmpty()) {

i = s.pop();

points.add(i);

}

for (int l = 0; l < points.size(); l++) f.writeBytes(v[l]+" ");//print vertex

f.writeBytes("\n");

for (int l = 0; l < points.size(); l++) f.writeBytes(d[(int)points.get(l)]+ " ");//print distance

f.writeBytes("\n");

//print (0)A->(9)C->(2)F->(9)E

for (int l = 0; l < points.size()-1; l++) f.writeBytes("("+d[(int)points.get(l)]+")"+v[l]+" "+"->");

f.writeBytes("("+d[(int)points.get(points.size()-1)]+")"+v[points.size()-1]+" ");

f.writeBytes("\n");

}

//count degree

int deg(int i) {

int s,j;

s=0;

for(j=0;j<n;j++) s += a[i][j];

s += a[i][i];

return(s);

}

//check has Isolated

boolean hasIsolated() {

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

if(deg(i)==0) return(true);

return(false);

}

//check connect

boolean isConnected() {

boolean [] p = new boolean[n];

int i,j,r;

for(i=0;i<n;i++) p[i]=false;

GStack s = new GStack();

s.push(0); p[0] = true;

while(!s.isEmpty()) {

r = s.pop();

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

if(!p[i] && a[r][i]>0) {

s.push(i); p[i]=true;

}

}

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

if(!p[i]) return(false);

return(true);

}

//check undirected

boolean isUnDirected() {

int i,j;

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

for(j=0;j<n;j++)

if(a[i][j] != a[j][i]) return(false);

return(true);

}

//check all deg even

boolean allDegEven() {

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

if(deg(i)%2 == 1) return(false);

return(true);

}

//check has euler cycle

boolean hasEulerCycle() {

if(!hasIsolated() && isUnDirected() && isConnected() && allDegEven())

return(true);

else

return(false);

}

//euler cycle

void eulerCycle(int fro, RandomAccessFile f) throws IOException {

if(!hasEulerCycle()) {

return;

}

int [] eu = new int[100];

int m,i,j,r;

GStack s = new GStack();

s.push(fro);

j=0;

while(!s.isEmpty()) {

r = s.top();

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

if(a[r][i] > 0) break;

}

if(i==n) { //r is isolated

s.pop();

eu[j++] = r;

}

else {

s.push(i);

a[r][i]--;

a[i][r]--;

}

}

m = j;

for(i=0;i<m;i++) {

f.writeBytes(v[eu[i]]+" ");

}

}

/\*

declare a stack S of characters (a vertex is labeled by a character)

declare an empty array E (which will contain Euler cycle)

push the vertex X to S

while(S is not empty)

{ch = top element of the stack S

if ch is isolated then remove it from the stack and put it to E

else

select the first vertex Y (by alphabet order), which is adjacent

to ch,push Y to S and remove the edge (ch,Y) from the graph

}

the last array E obtained is an Euler cycle of the graph

\*/

void EulerCycle(int k, RandomAccessFile f) throws Exception{

if(!isUndirected() || !isConnected() || !isEvenDegree())

{f.writeBytes("Conditions are not satisfied\r\n");

return;

}

MyStack s = new MyStack();

int [][] b = new int[20][20];

int [] eu = new int[20]; int m;

int i,j,r;

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

for(j=0;j<n;j++) b[i][j]=a[i][j];

s.push(k);//Dua dinh k vao Stack

m = 0;//Ban dau chu trinh chua co phan tu nao

while(!s.isEmpty())

{r = s.top();

i=0;

while(i<n && b[r][i]==0) i++;//Tim i dau tien de b[r][i]#0

if(i==n) //r da la dinh co lap, dua r vao chu trinh Euler

{eu[m++] = r; s.pop();}//Lay dinh co lap ra khoi Stack

else

{s.push(i);b[r][i]--;b[i][r]--;}//Loai canh (i,r) khoi do thi

}

// Display Euler cycle

for(i=0;i<m;i++) f.writeBytes(v[eu[i]] + " ");

f.writeBytes("\r\n");

}

//check conditions for the existence of Euler’s cycle

void checkEulerCycle(RandomAccessFile f) throws Exception{

if(isUndirected())

f.writeBytes("The graph is undirected.\r\n");

else

f.writeBytes("The graph is directed.\r\n");

if(isConnected())

f.writeBytes("The graph is connected.\r\n");

else

f.writeBytes("The graph is not connected.\r\n");

if(isEvenDegree())

f.writeBytes("All vertices have even degree.\r\n");

else

f.writeBytes("The graph has a vertex with odd degree.\r\n");

if(isUndirected() && isConnected() && isEvenDegree())

f.writeBytes("Conditions for Euler's cycle are satisfied.\r\n");

else

f.writeBytes("Conditions for Euler's cycle are not satisfied.\r\n");

}

//count the connectivity parts

public int connectedParts(){ //f123.writeBytes("k = " + k + "\r\n");

boolean [] pushed = new boolean[20];

boolean cont = false;

int i, j, k, r;

for(i=0;i<n;i++) pushed[i]=false;

MyStack s = new MyStack();

k=0;

while(true)

{s.clear();

i = 0;

while(i<n && pushed[i]) i++;

if(i==n) break;

s.push(i); pushed[i] = true;

while(!s.isEmpty())

{r = s.pop();

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

{if(i==r) continue;

if(!pushed[i] && a[r][i]>0) {s.push(i); pushed[i] = true;}

}

}

k++;

}

return(k);

}