MINISTRY OF EDUCATION AND TRAINING

DONG A UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

-----📖---S--

BÀI TẬP LỚN

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giáo viên hướng dẫn: Lý Quỳnh Trân

Sinh viên thực hiện: Đỗ Thái Dương

Lớp: IT19A2B

PHÚ YÊN, 2021

Table of Contents

Chương 1: Mảng động

Câu 0:

- Class Student:

```
package Cau00;

public class Student {
	private int ID;
	private String name;
	private int age;
	private String gender;

	public Student(int id, String name, int age, String gender) {
		super();
		this.ID = id;
		this.name = name;
		this.age = age;
		this.gender = gender;
	}

	public int getID() {
		return ID;
	}

	public void setID(int IDSV) {
		this.ID = ID;
	}

	public String getName() {
		return name;
	}
```

```
	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getGender() {
		return gender;
	}

	public void setGender(String gender) {
		this.gender = gender;
	}

	@Override
	public String toString() {
		return "ID= " + ID + ", name= " + name + ", age= " + age
+ ", gender= " + gender;
	}
}
```

- Class Main:

```
package Cau00;

import java.util.ArrayList;
import java.util.Scanner;

public class Main {
	public static Scanner in = new Scanner(System.in);

	public static Student getNewStudent(Scanner in) {
		System.out.print("Enter ID: ");
		String id = in.nextLine();

		System.out.print("Enter name: ");
		String name = in.nextLine();

		System.out.print("Enter age: ");
		String age = in.nextLine();

		System.out.print("Enter gender: ");
		String subject = in.nextLine();
```

```
            return new Student(Integer.parseInt(id), name,
Integer.parseInt(age), subject);
      }

      public static void choices() {
            System.out.println("Options:");
            System.out.println("1. Add students." + "\n2. Display
list of students." + "\n3. Remove students."
                        + "\n4. Update students." + "\n5. Exit.");
            System.out.print("Options 1-5: ");
      }

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            ArrayList<Student> list = new ArrayList<>();
            int choice;
            do {
                  choices();
                  choice = Integer.parseInt(in.nextLine());
                  switch (choice) {
                  case 1:
                        var newStudent = getNewStudent(in);
                        list.add(newStudent);
                        break;

                  case 2:
                        System.out.println("List of students: ");
                        for (Student student : list)
                              System.out.println(student.toString());
                        break;
                  case 3:
                        System.out.print("Remove at num: ");
                        int removeNum =
Integer.parseInt(in.nextLine());
                        list.remove(removeNum);
                        break;

                  case 4:
                        System.out.print("Update at num: ");
                        int updateNum =
Integer.parseInt(in.nextLine());
                        list.set(updateNum, getNewStudent(in));
                        break;

                  case 5:
                        System.out.println("Exit");
                        break;

                  default:
                        System.err.println("Invalid choice, please
choose again");
```

```
                        return;
                    }
                } while (choice != 5);
        }
}
```

Chương 2: Danh sách liên kết LinkedList

Câu 1:

- Class Node:

```
package Cau01;

public class Node {
        public int value;
        public Node next;

        public Node(int value) {
                this.value = value;
        }

}
```

- Class SinglyLinkedList:

```
package Cau01;

import java.util.NoSuchElementException;

public class SinglyLinkedList {
        private Node initNode;
        private int total;

        public boolean isEmpty() {
                return initNode == null;
        }

        public void addFirst(int x) {
                Node newNode = new Node(x);
                if (isEmpty()) {
                        initNode = newNode;
                        initNode.next = null;
                } else {
                        newNode.next = initNode;
                        initNode = newNode;
                }
                total++;
        }

        public int removeFromFront() {
                if (isEmpty())
                        throw new NoSuchElementException();
```

```
            var result = initNode.value;
            initNode = initNode.next;
            total--;
            return result;
      }

      public int count() {
            return total;
      }

      public void printList() {
            var current = initNode;
            while (current != null) {
                  System.out.print(current.value + "  ");
                  current = current.next;
            }
            System.out.println();
      }
}
```

- Class Main:

```
package Cau01;

public class Main {

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            SinglyLinkedList list = new SinglyLinkedList();
            list.addFirst(1);
            list.addFirst(2);
            list.addFirst(3);
            list.addFirst(4);
            System.out.print("Data list: ");
            list.printList();
            System.out.println("Removed from front: " +
list.removeFromFront());
            System.out.print("New data after removed from front: ");
            list.printList();
            System.out.println("Lenght of list: " + list.count());
      }

}
```

Câu 2:

- Class Node:

```
package Cau02;

public class Node {
```

```
    public Patient value;
    public Node next;
    public Node previous;

    public Node(Patient value) {
        this.value = value;
    }
}
```

- Class DoublyLinkedList:

```
package Cau02;

import java.util.NoSuchElementException;

public class DoublyLinkedList {
      private Node headNode;
      private Node tailNode;
      private int count;

      private boolean isEmpty() {
            return headNode == null && tailNode == null;
      }

      public void insertAtHead(Patient p) {
            var newNode = new Node(p);
            count++;
            if (isEmpty()) {
                  headNode = tailNode = newNode;
                  return;
            }
            newNode.next = headNode;
            headNode.previous = newNode;
            headNode = newNode;
      }

      public void insertAtTail(Patient p) {
            var newNode = new Node(p);
            count++;
            if (isEmpty())
                  headNode = tailNode = newNode;
            else {
                  tailNode.next = newNode;
                  newNode.previous = tailNode;
                  tailNode = newNode;
            }
      }

      public Node removeFromHead() {
            if (isEmpty())
```

```
                throw new NoSuchElementException();

            var result = headNode;
            if (headNode == tailNode)
                headNode = tailNode = null;
            else {
                headNode.next.previous = null;
                headNode = headNode.next;
            }
            count--;
            return result;
        }

        public Node removeFromEnd() {
            if (isEmpty())
                throw new NoSuchElementException();

            var result = tailNode;
            if (tailNode == headNode)
                tailNode = headNode = null;
            else {
                tailNode.previous.next = null;
                tailNode = tailNode.previous;
            }
            count--;
            return result;
        }

        public int count() {
            return count;
        }

        public void printList() {
            var currentNode = headNode;
            while (currentNode != null) {
                System.out.println(currentNode.value + "   ");
                currentNode = currentNode.next;
            }
            System.out.println();
        }

}
```

- Class Patient:

```
package Cau02;

public class Patient {
    private int id;
    private String name;
    private String medicalHistory;
```

```
	private Double hospitalFee;

	public Patient(int id, String name, String medicalHistory,
Double hospitalFee) {
		this.id = id;
		this.name = name;
		this.medicalHistory = medicalHistory;
		this.hospitalFee = hospitalFee;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getMedicalHistory() {
		return medicalHistory;
	}

	public void setMedicalHistory(String medicalHistory) {
		this.medicalHistory = medicalHistory;
	}

	public Double getHospitalFee() {
		return hospitalFee;
	}

	public void setHospitalFee(Double hospitalFee) {
		this.hospitalFee = hospitalFee;
	}

	@Override
	public String toString() {
		return "Patient{" + "id=" + id + ", name='" + name + '\''
+ ", medicalHistory='" + medicalHistory + '\''
				+ ", hospitalFee=" + hospitalFee + '}';
	}

}
```

- Class sortByName:

```
package Cau02;

public class sortByName {
	public void sort(Patient[] patients, int count) {
		for (int i = 0; i < count - 1; i++) {
			for (int j = i + 1; j < count; j++) {
				if
(patients[i].getName().compareTo(patients[j].getName()) > 0) {
					Patient temp = patients[i];
					patients[i] = patients[j];
					patients[j] = temp;
				}
			}
		}
	}
}
```

- Class Main:

```
package Cau02;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Patient p1 = new Patient(111, "John", "Flu", 1000.31);
		Patient p2 = new Patient(222, "Kim", "High Hypertension",
2000.38);
		Patient p3 = new Patient(333, "Jack", "Headache",
3000.14);
		Patient p4 = new Patient(444, "William", "Psycho",
4000.15);
		Patient p5 = new Patient(555, "Kenny", "Overworking",
5000.96);
		DoublyLinkedList list = new DoublyLinkedList();
		list.insertAtHead(p1);
		System.out.println("Removed from end: " +
list.removeFromEnd().value);
		list.insertAtHead(p2);
		list.insertAtHead(p3);
		list.insertAtTail(p4);
		list.insertAtTail(p5);
		list.printList();
		System.out.println("Removed from end: " +
list.removeFromEnd().value);
		System.out.println("Removed from head: " +
list.removeFromHead().value);
		list.printList();
		System.out.println("Lenght of list: " + list.count());
```

```
	}
}
```

Câu 3:

- Class Node:

```
package Cau03;

public class Node {
	private Product Product;
	private Node next;
	private Node previous;

	public Node(Product Product) {
		this.Product = Product;
	}

	public Product getProduct() {
		return Product;
	}

	public void setProduct(Product Product) {
		this.Product = Product;
	}

	public Node getNext() {
		return next;
	}

	public void setNext(Node next) {
		this.next = next;
	}

	public Node getPrevious() {
		return previous;
	}

	public void setPrevious(Node previous) {
		this.previous = previous;
	}
}
```

- Class Product:

```
package Cau03;

public class Product {
	private String name;
	private int price;
```

```
	private int quantity;

	public Product(String name, int price, int quantity) {
		this.name = name;
		this.price = price;
		this.quantity = quantity;
	}

	public String getNameProduct() {
		return name;
	}

	public void setNameProduct(String name) {
		this.name = name;
	}

	public int getPrice() {
		return price;
	}

	public int getQuantity() {
		return quantity;
	}

	public void setQuantity(int quantity) {
		this.quantity = quantity;
	}

	@Override
	public String toString() {
		return "Product {name=" + name + ", price=" + price + ",
quantity=" + quantity + "}";
	}

}
```

- Class public class ProductLinkedList {

```
package Cau03;

public class ProductLinkedList {
	private Node head;
	private int size;

	public boolean isEmpty() {
		return this.head == null;
	}

	public int getSize() {
		return size;
	}
```

```
	public void addFirst(Product Product) {
		var newNode = new Node(Product);
		newNode.setNext(this.head);
		this.head = newNode;
		size++;
	}

	public Product[] toArray() {
		Product[] Product = new Product[size];
		var currentNode = this.head;
		int index = 0;
		while (currentNode != null) {
			Product result = currentNode.getProduct();
			Product[index] = result;
			currentNode = currentNode.getNext();
			index++;
		}
		return Product;
	}

	public void printList() {
		for (int i = 0; i < toArray().length; i++) {
			System.out.println(toArray()[i].toString());
		}
	}

	public void printName() {
		for (int i = 0; i < toArray().length; i++) {
			System.out.printf("%d. %s\n", i,
this.toArray()[i].getNameProduct());
		}
	}
}
```

- Class Main:

```
package Cau03;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Product products1 = new Product("Shoe", 110, 42);
		Product products2 = new Product("T-Shirt", 270, 15);
		Product products3 = new Product("Watch", 128, 32);
		Product products4 = new Product("TV", 240, 112);
		ProductLinkedList productsLinkedList = new
ProductLinkedList();
		productsLinkedList.addFirst(products1);
		productsLinkedList.addFirst(products2);
		productsLinkedList.addFirst(products3);
```

```
            productsLinkedList.addFirst(products4);
            System.out.println("List products: ");
            productsLinkedList.printName();
            System.out.println("List quantity products: ");
            productsLinkedList.printList();
      }
}
```

Câu 4:

- Class Node:

```
package Cau04;

public class Node {
      public Product value;
      public Node next;
      public Node previous;

      public Node(Product value) {
            this.value = value;
      }

}
```

- Class Product:

```
package Cau04;

public class Product {
      private int id;
      private String name;
      private String type;
      private int yearOfManufacturer;
      private int yearWarranty;

      public Product(int id, String name, String type, int
yearOfManufacturer, int yearWarranty) {
            this.id = id;
            this.name = name;
            this.type = type;
            this.yearOfManufacturer = yearOfManufacturer;
            this.yearWarranty = yearWarranty;
      }

      public int getId() {
            return id;
      }

      public void setId(int id) {
            this.id = id;
```

```
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getYearOfManufacturer() {
		return yearOfManufacturer;
	}

	public int getYearWarranty() {
		return yearWarranty;
	}

	@Override
	public String toString() {
		return "IDSP: " + id + ", Name: '" + name + '\'' + ",
Type: '" + type + '\'' + ", Year of manufacture: "
				+ yearOfManufacturer + ", Num of year of
warranty: " + yearWarranty;
	}

}
```

- Class ProductLinkedList:

```
package Cau04;

import java.time.LocalDate;
import java.util.NoSuchElementException;

public class ProductLinkedList {
	private Node head;
	private Node tail;
	private int size;

	public boolean isEmpty() {
		return head == null && tail == null;
	}

	public int getSize() {
		return size;
	}

	public void insertAtHead(Product p) {
		var newNode = new Node(p);
		size++;
```

```
            if (isEmpty()) {
                  head = tail = newNode;
                  return;
            }
            newNode.next = head;
            head.previous = newNode;
            head = newNode;
      }

      public void insertAtTail(Product p) {
            var newNode = new Node(p);
            size++;
            if (isEmpty()) {
                  head = tail = newNode;
                  return;
            }
            tail.next = newNode;
            newNode.previous = tail;
            tail = newNode;
      }

      public void removeExpired() {
            if (isEmpty())
                  throw new NoSuchElementException();

            var node = head.next;
            while (node != null) {
                  var checkExpired =
(node.value.getYearOfManufacturer() +
node.value.getYearWarranty()) > LocalDate.now()
                              .getYear();
                  if (checkExpired == false) {
                        node.next.previous = node.previous;
                        node.previous.next = node.next;
                        node = node.next;
                        size--;
                  }
                  node = node.next;
            }

            if (head.value.getYearWarranty() +
head.value.getYearOfManufacturer() < LocalDate.now().getYear()) {
                  head.next.previous = null;
                  head = head.next;
                  size--;
            }
            if (tail.value.getYearWarranty() +
tail.value.getYearOfManufacturer() < LocalDate.now().getYear()) {
                  tail.previous.next = null;
                  tail = tail.previous;
                  size--;
```

```
            }
        }

        public void printList() {
            var currentNode = head;
            while (currentNode != null) {
                System.out.println(currentNode.value + "  ");
                currentNode = currentNode.next;
            }
            System.out.println();
        }
}
```

- Class Main:

```
package Cau04;

public class Main {

        public static void main(String[] args) {
            // TODO Auto-generated method stub
          Product product1 = new Product(1, "Ghế", "Nhựa", 2017, 8);
           Product product2 = new Product(2, "Búa", "Kim loại", 2016,
3);
           Product product3 = new Product(3, "Giấy A4", "Giấy", 2013,
5);
           Product product4 = new Product(4, "Thanh thước", "Nhựa",
2012, 8);

           ProductLinkedList productLinkedList = new
ProductLinkedList();

           productLinkedList.insertAtHead(product1);
           productLinkedList.insertAtHead(product2);
           productLinkedList.insertAtHead(product3);
           productLinkedList.insertAtHead(product4);
           productLinkedList.printList();
           productLinkedList.removeExpired();
           System.out.println("After remove expired year: ");
           productLinkedList.printList();
           System.out.println("Lenght of list: " +
productLinkedList.getSize());
       }
}
```

Chương 3: Ngăn xếp (Stack)

Câu 5:

- Class Main:

```
package Cau05;

import java.util.Stack;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Stack<String> stack = new Stack<>();
		// Example: EAS*Y**QUE***ST***I*ON
		stack.push("E");
		stack.push("A");
		stack.push("S");
		System.out.print(stack.peek() + " ");
		stack.push("Y");
		System.out.print(stack.peek() + " ");
		System.out.print(stack.peek() + " ");
		stack.push("Q");
		stack.push("U");
		stack.push("E");
		System.out.print(stack.peek() + " ");
		System.out.print(stack.peek() + " ");
		System.out.print(stack.peek() + " ");
		stack.push("S");
		stack.push("T");
		System.out.print(stack.peek() + " ");
		System.out.print(stack.peek() + " ");
		System.out.print(stack.peek() + " ");
		stack.push("I");
		System.out.print(stack.peek() + " ");
		stack.push("O");
		stack.push("N");
	}

}
```

Output: `S Y Y E E E T T T I`

Câu 6:

- Class Main:

```
package Cau06;

import java.util.Stack;
import java.util.EmptyStackException;
```

```
public class Main {

	public static void main(String[] args) {
		Stack<Integer> stack = new Stack<>();
		// Process stacking
		// Push
		System.out.println(stack.push(5) + " pushed to stack");
		System.out.println(stack.push(3) + " pushed to stack");
		System.out.println("Current stack: " + stack);
		// Pop
		System.out.println(stack.pop() + " pop from stack\n");
		System.out.println("Current stack: " + stack);
		System.out.println("First element: " + stack.peek());
		// Push
		System.out.println(stack.push(2) + " pushed to stack");
		System.out.println(stack.push(8) + " pushed to stack");
		System.out.println("Current stack: " + stack);
		// 3-time stack pop
		// stack.pop();
		// stack.pop();
		// stack.pop();
		for (int i = 1; i <= 3; i++) {
			System.out.println(stack.pop() + " pop from
stack\n");
			System.out.println("Current stack: " + stack);
			System.out.println("First element: " +
stack.peek());
			System.out.println("");
		}
		// Push
		System.out.println(stack.push(9) + " pushed to stack");
		System.out.println(stack.push(1) + " pushed to stack");
		System.out.println("Current stack: " + stack);
		// Pop
		System.out.println(stack.pop() + " pop from stack\n");
		System.out.println("Current stack: " + stack);
		System.out.println("First element: " + stack.peek());
		// Push
		System.out.println(stack.push(7) + " pushed to stack");
		System.out.println(stack.push(6) + " pushed to stack");
		System.out.println("Current stack: " + stack);
		// stack pop twice
		// stack.pop();
		// stack.pop();
		for (int i = 1; i < 2; i++) {
			System.out.println(stack.pop() + " pop from
stack\n");
			System.out.println("Current stack: " + stack);
			System.out.println("First element: " +
stack.peek());
		}
```

```
            System.out.println(stack.push(4) + " pushed to stack");
            System.out.println("Current stack: " + stack);
            // stack pop twice
            // stack.pop();
            // stack.pop();
            for (int i = 1; i < 2; i++) {
                  System.out.println(stack.pop() + " pop from
stack\n");
                  System.out.println("Current stack: " + stack);
                  System.out.println("First element: " +
stack.peek());

            }
      }
}
```

Câu 7:

- Class MyStack:

```
package Cau07;

import java.util.EmptyStackException;

public class MyStack {
      private int[] stack;
      private int top;

      public MyStack(int capacity) {
            stack = new int[capacity];
      }

      public void push(int value) {
            if (top == stack.length) {
                  int[] newArray = new int[2 * stack.length];
                  System.arraycopy(stack, 0, newArray, 0,
stack.length);
                  stack = newArray;
            }
            stack[top++] = value;
      }

      public boolean isEmpty() {
            return top == 0;
      }
```

```
	public int pop() {
		if (isEmpty()) {
			throw new EmptyStackException();
		}
		int value = stack[top - 1];
		stack[top - 1] = 0;
		top--;
		return value;
	}

	public void printStack() {
		for (int i = top - 1; i >= 0; i--) {
			System.out.println("" + stack[i]);
		}
	}
}
```

- Class Main:

```
package Cau07;

public class Main {
	public static void decToBin(int k) {
		MyStack s = new MyStack(20);
		System.out.print(k + " in binary system is: ");
		while (k > 0) {
			s.push(new Integer(k % 2));
			k = k / 2;
		}
		while (!s.isEmpty()) {
			System.out.print(s.pop());
		}
		System.out.println();
	}

	public static void decToOctal(int k) {
		MyStack s = new MyStack(20);
		System.out.print(k + " in Octal system is: ");
		while (k > 0) {
			s.push(new Integer(k % 8));
			k = k / 8;
		}
		while (!s.isEmpty()) {
			System.out.print(s.pop());
		}
		System.out.println();
	}

	public static void decToHex(int k) {
		String digits = "0123456789ABCDEF";
		MyStack s = new MyStack(20);
```

```
		System.out.print(k + " in Hexa system is: ");
		while (k > 0) {
			int digit = k % 16;
			s.push(digits.charAt(digit));
			k = k / 16;
		}
		while (!s.isEmpty()) {
			System.out.print((char) s.pop());
		}
		System.out.println();
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		decToBin(25);
		decToOctal(41);
		decToHex(18);
		System.out.println();
	}

}
```

Câu 8:

Implement by Stack:

- Class reverseNumByStack:

```
package Cau08;

import java.util.Stack;
import java.util.Scanner;

public class reverseNumByStack {

	static Stack<Integer> stack = new Stack<>();

	static void pushDigits(int number) {

		while (number != 0) {
			stack.push(number % 10);
			number = number / 10;
		}
	}

	static int reverse(int number) {
		pushDigits(number);
		int reverse = 0;
		int i = 1;

		while (!stack.isEmpty()) {
			reverse = reverse + (stack.peek() * i);
```

```
                stack.pop();
                i = i * 10;
            }
            return reverse;
        }

        public static void main(String[] args) {
            System.out.println("Input the number:");
            Scanner sc = new Scanner(System.in);
            int number = sc.nextInt();
            sc.close();
            System.out.println("Initial number: " + number);
            System.out.println("Number reversed by stack: " +
reverse(number));
        }
}
```

Implement by Queue:

- Class Main:

```
package Cau08;

import java.util.Queue;
import java.util.Scanner;
import java.util.Stack;
import java.util.LinkedList;

public class reverseNumByQueue {
    static Queue<Integer> queue = new LinkedList<>();

    static void pushDigits(int number) {

        while (number != 0) {
            queue.add(number % 10);
            number = number / 10;
        }
    }

    static void reverseQueue(Queue<Integer> queue) {
        int n = queue.size();
        Stack<Integer> stack = new Stack<>();
        // Remove all the elements from queue and push them to
stack
        for (int i = 0; i < n; i++) {
            int curr = queue.poll();
            stack.push(curr);
        }
        // Pop out elements from the stack and push them back to
queue
        for (int i = 0; i < n; i++) {
            int curr = stack.pop();
```

```
                queue.add(curr);
            }
        }

        static int reverse(int number) {
            pushDigits(number);
            int reverse = 0;
            int i = 1;
            reverseQueue(queue);
            while (!queue.isEmpty()) {
                reverse = reverse + (queue.peek() * i);
                queue.poll();
                i = i * 10;

            }
            return reverse;
        }

        public static void main(String[] args) {
            // TODO Auto-generated method stub
            System.out.println("Input the number:");
            Scanner sc = new Scanner(System.in);
            int number = sc.nextInt();
            sc.close();
            System.out.println("Initial number: " + number);
            System.out.println("Number reversed by queue: " +
reverse(number));
        }

}
```

Câu 9:

- Class MyStack:

```
package Cau09;

import java.util.EmptyStackException;

public class MyStack {
    private int[] stack;
    private int top;

    public MyStack() {
    }

    public MyStack(int capacity) {
        stack = new int[capacity];
    }
```

```
	public void push(int value) {
		// check the stack is full
		if (top == stack.length) {
			int[] newArray = new int[stack.length * 2];
			System.arraycopy(stack, 0, newArray, 0,
stack.length);
			stack = newArray;
		}
		stack[top++] = value;
	}

	public int pop() {
		if (isEmpty()) {
			throw new EmptyStackException();
		}
		int value = stack[top - 1];
		stack[top - 1] = 0;
		top--;
		return value;

	}

	public int peek() {
		if (isEmpty()) {
			throw new EmptyStackException();
		}
		return stack[top - 1];
	}

	public int size() {
		return top;
	}

	public boolean isEmpty() {
		return top == 0;

	}

	// clear stack
	public void makeEmpty() {
		top = 0;
	}

	public void printStack() {
		for (int i = top - 1; i >= 0; i--) {
			System.out.println("" + stack[i]);
		}
	}

}
```

- Class Main:

```
package Cau09;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MyStack s = new MyStack(10);
		int x = 3;
		int y = 5;
		int z = 2;
		s.makeEmpty();
		s.push(x);
		s.push(4);
		s.pop();
		s.push(y);
		s.push(3);
		s.push(z);
		s.pop();
		s.push(2);
		s.push(x);

		while (!s.isEmpty())
			System.out.print(s.pop() + " ");
	}

}
```

Output: `3 2 3 5 3`

Câu 10:

- Class MyStack:

```
package Cau10;

import java.util.EmptyStackException;

public class MyStack {
	private int[] stack;
	private int top;

	public MyStack() {
	}

	public MyStack(int capacity) {
		stack = new int[capacity];
	}

	public void push(int value) {
		// check the stack is full
```

```
            if (top == stack.length) {
                  int[] newArray = new int[stack.length * 2];
                  System.arraycopy(stack, 0, newArray, 0,
stack.length);
                  stack = newArray;
            }
            stack[top++] = value;
      }

      public int pop() {
            if (isEmpty()) {
                  throw new EmptyStackException();
            }
            int value = stack[top - 1];
            stack[top - 1] = 0;
            top--;
            return value;

      }

      public int peek() {
            if (isEmpty()) {
                  throw new EmptyStackException();
            }
            return stack[top - 1];
      }

      public int size() {
            return top;
      }

      public boolean isEmpty() {
            return top == 0;

      }

      // clear stack
      public void makeEmpty() {
            top = 0;
      }

      public void printStack() {
            for (int i = top - 1; i >= 0; i--) {
                  System.out.println("" + stack[i]);
            }
      }

}
```

- Class Main:

```
package Cau10;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MyStack s = new MyStack(10);
		int x = 3;
		int y = 1;
		s.makeEmpty();
		s.push(5);
		s.push(7);
		s.pop();
		x += y;
		s.pop();
		s.push(x);
		s.push(y);
		s.push(2);
		s.pop();
		s.pop();

		while (!s.isEmpty()) {
			y = s.pop();
			System.out.println(y);
		}

		System.out.println("x = " + x);
		System.out.println("y = " + y);
	}

}
```

Output:

```
4
x = 4
y = 4
```

Câu 11:

- Class Node:

```
package Cau11;

public class Node {
	public int value;
    public Node next;

    public Node(int value) {
        this.value = value;
    }
}
```

- Class MyStack:

```
package Cau11;

import java.util.EmptyStackException;

public class MyStack {
	private Node top;
	public int size;

	public boolean isEmpty() {
		return top == null;
	}

	public void clear() {
		top = null;
	}

	public void push(int x) {
		var newNode = new Node(x);
		size++;
		if (isEmpty()) {
			top = newNode;
			return;
		}
		newNode.next = top;
		top = newNode;
	}

	public int pop() {
		if (isEmpty())
			throw new EmptyStackException();
		var result = top;
		top = top.next;
		size--;
		return result.value;
	}

	public int top() {
		if (isEmpty())
			throw new EmptyStackException();
		return top.value;
	}

	public void traverse() {
		var temp = top;
		while (temp != null) {
			System.out.print(temp.value + " ");
			temp = temp.next;
		}
		System.out.println();
	}
```

```
	public int size() {
		return size;
	}
}
```

- Class Main:

```
package Cau11;

import java.util.Scanner;

public class Main {
	public static int input(Scanner sc) {
		System.out.println("Input an positive integer number: ");
		int num = Integer.parseInt(sc.nextLine());
		while (num < 0) {
			input(sc);
		}
		return num;
	}

	public static String decimalToBinary(int number) {
		MyStack sta = new MyStack();
		while (number != 0) {
			sta.push(number % 2);
			number /= 2;
		}
		String result = "";
		int length = sta.size();
		for (int i = 0; i < length; i++) {
			result += sta.pop();
		}
		return result;
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MyStack stack = new MyStack();
		stack.push(2);
		stack.push(5);
		stack.clear();
		stack.push(8);
		stack.push(11);
		stack.push(29);
		stack.push(31);
		stack.traverse();
		stack.pop();
		stack.traverse();
		Scanner sc = new Scanner(System.in);
		int decimal = input(sc);
```

```
            System.out.println("Convert decimal to Binary: " +
decimalToBinary(decimal));
      }

}
```

Chương 4: Hàng đợi (queue)

Câu 12:

- Class MyQueue:

```
package Cau12;

public class MyQueue {
      int front, rear, size;
      int capacity;
      int array[];

      public MyQueue(int capacity) {
            this.capacity = capacity;
            front = this.size = 0;
            rear = capacity - 1;
            array = new int[this.capacity];
      }

      // Check if the queue is full
      boolean isFull(MyQueue queue) {
            return (queue.size == queue.capacity);
      }

      // Queue is empty when size is 0
      boolean isEmpty(MyQueue queue) {
            return (queue.size == 0);
      }


      void enqueue(int item) {
            if (isFull(this))
                  return;
            this.rear = (this.rear + 1) % this.capacity;
            this.array[this.rear] = item;
            this.size = this.size + 1;
            System.out.println(item + " enqueued to queue");
      }

      int dequeue() {
            if (isEmpty(this))
                  return Integer.MIN_VALUE;

            int item = this.array[this.front];
```

```
            this.front = (this.front + 1) % this.capacity;
            this.size = this.size - 1;
            return item;
      }
      int front() {
            if (isEmpty(this))
                  return Integer.MIN_VALUE;

            return this.array[this.front];
      }

      int rear() {
            if (isEmpty(this))
                  return Integer.MIN_VALUE;

            return this.array[this.rear];
      }
}
```

- Class Main:

```
package Cau12;

public class Main {

      public static void main(String[] args) {
            MyQueue q = new MyQueue(20);
            q.enqueue(5);
            q.enqueue(3);
            System.out.println(q.dequeue() + " dequeued from
queue\n");
            System.out.println("Front item is " + q.front());
            System.out.println("Rear item is " + q.rear());
            q.enqueue(2);
            q.enqueue(8);
            System.out.println(q.dequeue() + " dequeued from
queue\n");
            System.out.println("Front item is " + q.front());
            System.out.println("Rear item is " + q.rear());
            q.enqueue(9);
            q.enqueue(1);
            System.out.println(q.dequeue() + " dequeued from
queue\n");
            System.out.println("Front item is " + q.front());
            System.out.println("Rear item is " + q.rear());
            q.enqueue(7);
            q.enqueue(6);
            System.out.println(q.dequeue() + " dequeued from
queue\n");
```

```
            System.out.println("Front item is " + q.front());
            System.out.println("Rear item is " + q.rear());
            // double dequeue
            System.out.println(q.dequeue() + " dequeued from
queue\n");
            System.out.println("Front item is " + q.front());
            System.out.println("Rear item is " + q.rear());
            q.enqueue(4);
            System.out.println(q.dequeue() + " dequeued from
queue\n");
            System.out.println("Front item is " + q.front());
            System.out.println("Rear item is " + q.rear());
            // double dequeue
            System.out.println(q.dequeue() + " dequeued from
queue\n");
            System.out.println("Front item is " + q.front());
            System.out.println("Rear item is " + q.rear());
      }
}
```

Câu 13:

- Class Main:

```
package Cau13;

import java.util.Queue;
import java.util.Scanner;
import java.util.Stack;
import java.util.LinkedList;

public class Main {
      static Queue<Integer> queue = new LinkedList<>();

      static void pushDigits(int number) {

            while (number != 0) {
                  queue.add(number % 10);
                  number = number / 10;
            }
      }

      static void reverseQueue(Queue<Integer> queue) {
            int n = queue.size();
            Stack<Integer> stack = new Stack<>();
            // Remove all the elements from queue and push them to
stack
            for (int i = 0; i < n; i++) {
                  int curr = queue.poll();
                  stack.push(curr);
```

```
            }
            // Pop out elements from the stack and push them back to
queue
            for (int i = 0; i < n; i++) {
                  int curr = stack.pop();
                  queue.add(curr);
            }
      }

      static int reverse(int number) {
            pushDigits(number);
            int reverse = 0;
            int i = 1;
            reverseQueue(queue);
            while (!queue.isEmpty()) {
                  reverse = reverse + (queue.peek() * i);
                  queue.poll();
                  i = i * 10;

            }
            return reverse;
      }

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            System.out.println("Input the number:");
            Scanner sc = new Scanner(System.in);
            int number = sc.nextInt();
            sc.close();
            System.out.println("Initial number: " + number);
            System.out.println("Number reversed by queue: " +
reverse(number));
      }

}
```

Câu 14:

- Class Node:

```
package Cau14;

public class Node {
      public int value;
    public Node next;
    public Node previous;

    public Node(int d) {
        this.value = d;
    }
}
```

- Class Queue:

```
package Cau14;

import java.util.EmptyStackException;

public class Queue {
	private Node first;
	private Node last;
	public int size;

	public boolean isEmpty() {
		return first == null && last == null;
	}

	public void clear() {
		first = last = null;
	}

	public void enqueue(int d) {
		Node newNode = new Node(d);
		size++;
		if (isEmpty()) {
			first = last = newNode;
			return;
		}

		newNode.previous = last;
		last.next = newNode;
		last = newNode;
	}

	public int dequeue() {
		if (isEmpty())
			throw new EmptyStackException();

		var result = first;
		if (first == last)
			first = last = null;
		else {
			first.next.previous = null;
			first = first.next;
		}
		size--;
		return result.value;
	}

	public int first() {
		if (isEmpty())
			throw new EmptyStackException();
```

```
            return first.value;
      }

      public void traverse() {
            var current = first;
            while (current != null) {
                  System.out.print(current.value + " ");
                  current = current.next;
            }
            System.out.println();
      }

      public String toStringReverse() {
            var end = last;
            var result = "";
            while (end != null) {
                  result += end.value;
                  end = end.previous;
            }
            return result;
      }

      public int size() {
            return size;
      }


}
```

- Class Main:

```
package Cau14;

import java.util.Scanner;

public class Main {

      public static int input(Scanner scanner) {
            System.out.print("Input an real number that less than 1:
");
            int num = Integer.parseInt(scanner.nextLine());
            while (num > 1) {
                  input(scanner);
            }
            return num;
      }

      public static String decimalToBinary(int num) {
            Queue q = new Queue();
            while (num != 0) {
                  q.enqueue(num % 2);
```

```
                num = num / 2;
            }
            return q.toStringReverse();
      }

      public static void main(String[] args) {
            Queue q = new Queue();
            q.enqueue(5);
            q.enqueue(16);
            q.enqueue(21);
            q.clear();
            q.enqueue(9);
            q.enqueue(29);
            q.enqueue(41);
            q.enqueue(3);
            q.traverse();
            System.out.println(q.dequeue() + " dequeued from queue");
            q.traverse();
            System.out.println("First value: " + q.first());
            Scanner in = new Scanner(System.in);
            int number = input(in);
            System.out.println("Convert decimal to Binary: " +
decimalToBinary(number));
      }
}
```

Chương 5: Đệ quy

Câu 15:

```
package Cau15;

public class Main {
      public static double sum(int n) {
            if (n == 1)
                  return 1;
            return (double) 1 / n + sum(n - 1);
      }

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            System.out.println("Sum: " + sum(8));
      }

}
```

Câu 16:

- Class Main:

```
package Cau16;

public class Main {

	public static long fact(int n) {
		if (n <= 1)
			return 1;
		return n * fact(n - 1);
	}

	public static void main(String[] args) {
		System.out.println("Factorial function recursively: " +
fact(5));
	}
}
```

Câu 17:

- Class Main:

```
package Cau17;

public class Main {
	public static long fib(int n) {
		if (n <= 2)
			return 1;
		return fib(n - 1) + fib(n - 2);
	}

	public static void main(String[] args) {
		System.out.println("Fibonacci recursively: " + fib(4));
	}
}
```

Câu 18:

- Class Main:

```
package Cau18;

public class Main {

	public static int findsum(int a[], int n) {
		// base case
		if (n == 1)
			return a[0];
		else
			return Math.addExact(a[n - 1], findsum(a, n - 1));
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
```

```
            int arr[] = { 1, 9, 26, 6, -50, -120, 25, 54, -31 };
            System.out.println("The initial array is: ");
            for (int i = 0; i < arr.length; i++) {
                  System.out.print(+arr[i] + " ");
            }
            System.out.println();
            int n = arr.length;
            System.out.println("The sum of all elements in the array
is: " + findsum(arr, n));
      }

}
```

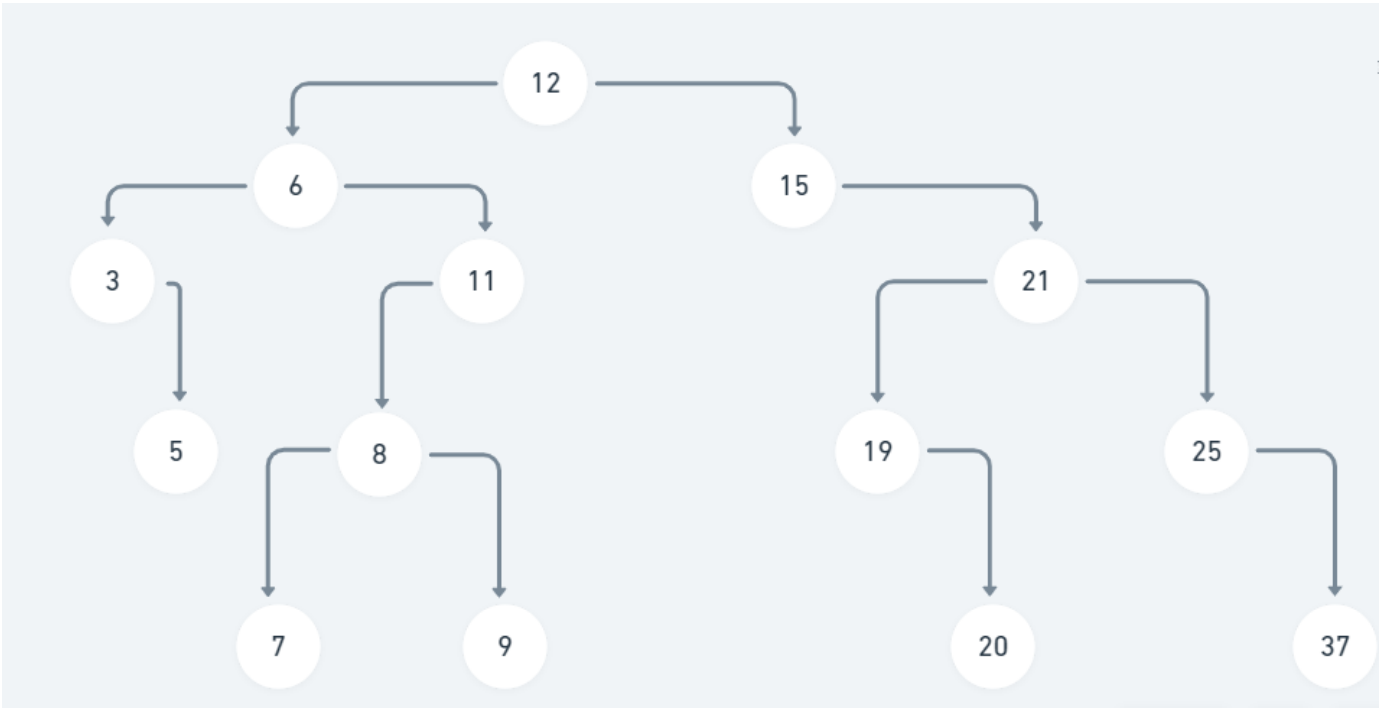
Câu 19:

- Class Main:

```
package Cau19;

public class Main {

      public static int isPalindrome(char a[], int n) {
            if (n == 0)
                  return 1;
            if (a[a.length - n] == a[n - 1])
                  return isPalindrome(a, n - 1);
            return 0;
      }

      public static void main(String[] args) {
            String s1 = "geeg";
            String s2 = "abcawawof";
            // return 1 if a[] is a palindrome, 0 otherwise
            System.out.println(s1 + ": " +
isPalindrome(s1.toCharArray(), s1.toCharArray().length - 1));
            System.out.println(s2 + ": " +
isPalindrome(s2.toCharArray(), s2.toCharArray().length - 1));
      }
}
```

Chương 6: Cây

Câu 20: Đồ thị cây nhị phân được vẽ lại như sau: (Hình được vẽ trên Whimsical)



Theo cách duyệt Post Order, ta có thứ tự duyệt trên hình:

5, 3, 7, 9, 8, 11, 6, 20, 19, 37, 25, 21, 15, 12

Cấu trúc dữ liệu và các hàm được dựng như sau:

Class TreeNode:

```
package Cau20;

public class TreeNode {
	private int data;
	private TreeNode leftChild;
	private TreeNode rightChild;

	public TreeNode(int data) {
		this.data = data;
	}

	public void insert(int value) {
		// TODO Auto-generated method stub
		if (value == data) {
			return;
		}
		if (value < data) {
			if (leftChild == null) {
				leftChild = new TreeNode(value);
			} else {
```

```
                leftChild.insert(value);
            }
        } else {
            if (rightChild == null) {
                rightChild = new TreeNode(value);
            } else {
                rightChild.insert(value);
            }
        }
    }

    public boolean isLeaf() {
        return leftChild == null ? rightChild == null : false;
    }

    public void traversePreOrder() {
        System.out.print(data + ", ");
        if (leftChild != null) {
            leftChild.traversePreOrder();
        }
        if (rightChild != null) {
            rightChild.traversePreOrder();
        }
    }

    public void traverseInOrder() {
        // TODO Auto-generated method stub
        if (leftChild != null) {
            leftChild.traverseInOrder();
        }
        System.out.print(data + ", ");
        if (rightChild != null) {
            rightChild.traverseInOrder();
        }
    }

    public void traversePostOrder() {
        if (leftChild != null) {
            leftChild.traversePostOrder();
        }
        if (rightChild != null) {
            rightChild.traversePostOrder();
        }
        System.out.print(data + ", ");
    }

    public TreeNode get(int value) {
        if (value == data) {
            return this;
        }
        if (value < data) {
```

```
            if (leftChild != null) {
                  return leftChild.get(value);
            }
      } else {
            if (rightChild != null) {
                  return rightChild.get(value);
            }
      }
      return null;
}

public int min() {
      if (leftChild == null) {
            return data;
      } else {
            return leftChild.min();
      }
}

public int max() {
      if (rightChild == null) {
            return data;
      } else {
            return rightChild.max();
      }
}

public int getData() {
      return data;
}

public void setData(int data) {
      this.data = data;
}

public TreeNode getLeftChild() {
      return leftChild;
}

public void setLeftChild(TreeNode leftChild) {
      this.leftChild = leftChild;
}

public TreeNode getRightChild() {
      return rightChild;
}

public void setRightChild(TreeNode rightChild) {
      this.rightChild = rightChild;
}
```

```
	@Override
	public String toString() {
		return "TreeNode {data=" + data + ", leftChild=" +
leftChild + ", rightChild=" + rightChild + "}";
	}

}
```

Class Tree:

```
package Cau20;

public class Tree {
	private TreeNode root;

	//insert
	public void insert(int value) {
		if (root == null) {
			root = new TreeNode(value);
		} else {
			root.insert(value);
		}
	}

	//traverse depth-search
	//preorder
	public void traversePreOrder() {
		if (root != null) {
			root.traversePreOrder();
		}
	}

	//inorder
	public void traverseInOrder() {
		if (root != null) {
			root.traverseInOrder();
		}
	}

	//postorder
	public void traversePostOrder() {
		if (root != null) {
			root.traversePostOrder();
		}
	}

	// get root
	public TreeNode getRoot() {
		return root;
```

```
	}

	// set root
	public void setRoot(TreeNode root) {
		this.root = root;
	}

	//get value info
	public TreeNode get(int value) {
		if (root != null) {
			return root.get(value);
		}
		return null;
	}

	//count leaf node in binary tree
	public int countLeafNodes() {
		return countLeaves(root);
	}

	private int countLeaves(TreeNode node) {
		if (node == null) {
			return 0;
		}
		if (node.isLeaf()) {
			return 1;
		} else {
			return countLeaves(node.getLeftChild()) +
countLeaves(node.getRightChild());
		}
	}

	//print leaf node in binary tree
	public void printLeafNodes(TreeNode node) {
		if (node == null) {
			return;
		} else if (node.getLeftChild() == null &&
node.getRightChild() == null) {
			System.out.print(node.getData() + " ");
		}
		printLeafNodes(node.getLeftChild());
		printLeafNodes(node.getRightChild());
	}
	//get min element
	public int min() {
		if(root==null) {
			return Integer.MIN_VALUE;
		} else {
			return root.min();
		}
	}
```

```
	//get max element
	public int max() {
		if(root==null) {
			return Integer.MAX_VALUE;
		} else {
			return root.max();
		}

	}

	//binary search on binary tree
	public TreeNode binarySearch(TreeNode root, int value) {
		if (root == null) {
			return (null);
		}
		if (root.getData() == value) {
			return root;
		}
		if (value < root.getData()) {
			return (binarySearch(root.getLeftChild(), value));
		} else {
			return (binarySearch(root.getRightChild(), value));
		}

	}
}
```

Class Main:

```
package Cau20;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Tree tree = new Tree();
		tree.insert(12);
		TreeNode root = tree.get(12);
		tree.insert(6);
		tree.insert(3);
		tree.insert(5);
		tree.insert(11);
		tree.insert(8);
		tree.insert(7);
		tree.insert(9);
		tree.insert(15);
		tree.insert(21);
		tree.insert(19);
		tree.insert(20);
		tree.insert(25);
		tree.insert(37);
```

```
            System.out.println("Binary search Pre-order:");
            tree.traversePreOrder();
            System.out.println();
            System.out.println("Binary search In-order:");
            tree.traverseInOrder();
            System.out.println();
            System.out.println("Binary search Post-order:");
            tree.traversePostOrder();
            System.out.println();
            System.out.println("
"+tree.binarySearch(tree.getRoot(),15));
            System.out.println("The min element of the binary tree: "
+ tree.min());
            System.out.println("The max element of the binary tree: "
+ tree.max());
            System.out.println("Total leaf node of the binary tree: "
+ tree.countLeafNodes());
            System.out.print("Leaves of the binary tree: ");
            tree.printLeafNodes(root);
      }

}
```

Câu 21: Thuật toán duyệt cây theo chiều sâu sử dụng đệ quy:

Có 3 cách duyệt mà chúng ta sẽ đi vào tìm hiểu chi tiết trong bài này đó là: Preorder, Inorder, Postorder.

Các cách duyệt cây này khác nhau ở thứ tự mà chúng ta duyệt qua các cây con bên trái, bên phải và node gốc.

Cách thức duyệt như sau:

Preorder: Gốc – Trái – Phải

Inorder: Trái – Gốc – Phải

Postorder: Trái – Phải – Gốc

Với Preorder:

Bước 1: Thăm node gốc.

Bước 2: Thăm cây con bên trái (thăm tất cả các node của cây con bên trái), sử dụng đệ quy gọi lại hàm để thăm tất cả cây con bên trái.

Bước 3: Thăm cây con bên phải (thăm tất cả các node của cây con bên phải), sử dụng đệ quy gọi lại hàm để thăm tất cả cây con bên phải

Xây dựng trong Eclipse:

```
public void traversePreOrder() {
            System.out.print(data + ", ");
```

```
            if (leftChild != null) {
                  leftChild.traversePreOrder();
            }
            if (rightChild != null) {
                  rightChild.traversePreOrder();
            }

      }
```

Với Inorder:

Bước 1: Thăm con bên trái (thăm tất cả các node của cây con bên trái), sử dụng đệ quy gọi lại hàm để thăm tất cả cây con bên trái.

Bước 2: Thăm node gốc.

Bước 3: Thăm con bên phải (thăm tất cả các node của cây con bên phải), sử dụng đệ quy gọi lại hàm để thăm tất cả cây con bên phải.

Xây dựng trong Eclipse:

```
public void traverseInOrder() {
            if (leftChild != null) {
                  leftChild.traverseInOrder();
            }
            System.out.print(data + ", ");
            if (rightChild != null) {
                  rightChild.traverseInOrder();
            }

      }
```

Với Postorder:

Bước 1:Thăm con bên trái (thăm tất cả các node của cây con bên trái), sử dụng đệ quy gọi lại hàm để thăm tất cả cây con bên trái.

Bước 2: Thăm con bên phải (thăm tất cả các node của cây con bên phải), sử dụng đệ quy gọi lại hàm để thăm tất cả cây con bên phải.

Bước 3: Thăm node gốc.

Xây dựng trong Eclipse:

```
public void traversePostOrder() {
            if (leftChild != null) {
                  leftChild.traversePostOrder();
            }
            if (rightChild != null) {
                  rightChild.traversePostOrder();
            }
            System.out.print(data + ", ");

      }
```

Với cây nhị phân cho bởi đề bài, cách duyệt sâu preorder cho thứ tự duyệt: 5, 2, 1, 3, 7, 6, 8.

*Tiến hành code hiển thị kết quả:

Class TreeNode:

```
package Cau21;

public class TreeNode {
	private int data;
	private TreeNode leftChild;
	private TreeNode rightChild;

	public TreeNode(int data) {
		this.data = data;
	}

	public void insert(int value) {
		// TODO Auto-generated method stub
		if (value == data) {
			return;
		}
		if (value < data) {
			if (leftChild == null) {
				leftChild = new TreeNode(value);
			} else {
				leftChild.insert(value);
			}
		} else {
			if (rightChild == null) {
				rightChild = new TreeNode(value);
			} else {
				rightChild.insert(value);
			}
		}
	}

	public void traversePreOrder() {
		System.out.print(data + ", ");
		if (leftChild != null) {
			leftChild.traversePreOrder();
		}
		if (rightChild != null) {
			rightChild.traversePreOrder();
		}
	}

	public void traverseInOrder() {
		// TODO Auto-generated method stub
		if (leftChild != null) {
			leftChild.traverseInOrder();
		}
```

```
        System.out.print(data + ", ");
        if (rightChild != null) {
            rightChild.traverseInOrder();
        }
    }

    public void traversePostOrder() {
        if (leftChild != null) {
            leftChild.traversePostOrder();
        }
        if (rightChild != null) {
            rightChild.traversePostOrder();
        }
        System.out.print(data + ", ");
    }

    public TreeNode get(int value) {
        if (value == data) {
            return this;
        }
        if (value < data) {
            if (leftChild != null) {
                return leftChild.get(value);
            }
        } else {
            if (rightChild != null) {
                return rightChild.get(value);
            }
        }
        return null;
    }

    public int min() {
        if (leftChild == null) {
            return data;
        } else {
            return leftChild.min();
        }
    }

    public int max() {
        if (rightChild == null) {
            return data;
        } else {
            return rightChild.max();
        }
    }

    public int getData() {
        return data;
    }
```

```
	public void setData(int data) {
		this.data = data;
	}

	public TreeNode getLeftChild() {
		return leftChild;
	}

	public void setLeftChild(TreeNode leftChild) {
		this.leftChild = leftChild;
	}

	public TreeNode getRightChild() {
		return rightChild;
	}

	public void setRightChild(TreeNode rightChild) {
		this.rightChild = rightChild;
	}

	@Override
	public String toString() {
		return "TreeNode {data=" + data + ", leftChild=" +
leftChild + ", rightChild=" + rightChild + "}";
	}

}
```

Class Tree:

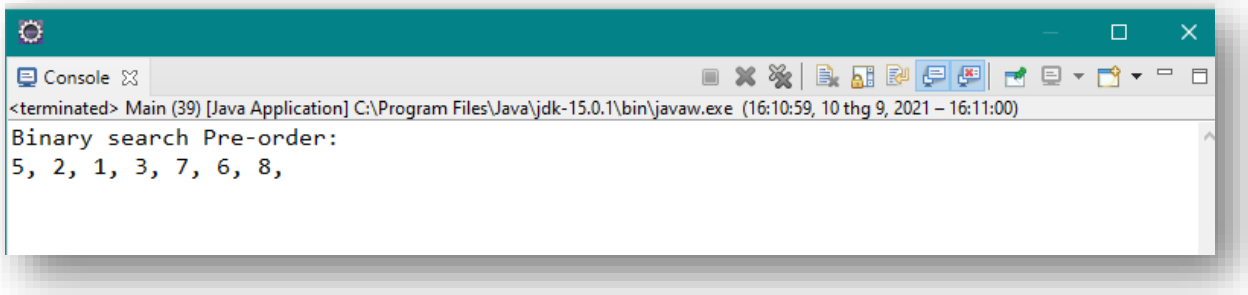
```
package Cau21;

import Cau20.TreeNode;

public class Tree {
	private TreeNode root;

	//insert
	public void insert(int value) {
		if (root == null) {
			root = new TreeNode(value);
		} else {
			root.insert(value);
		}
	}

	//traverse depth-search
	//preorder
	public void traversePreOrder() {
		if (root != null) {
			root.traversePreOrder();
```

```
        }
    }

    //inorder
    public void traverseInOrder() {
        if (root != null) {
            root.traverseInOrder();
        }
    }

    //postorder
    public void traversePostOrder() {
        if (root != null) {
            root.traversePostOrder();
        }
    }

    //ger root
    public TreeNode getRoot() {
        return root;
    }

    // set root
    public void setRoot(TreeNode root) {
        this.root = root;
    }

    //get value info
    public TreeNode get(int value) {
        if (root != null) {
            return root.get(value);
        }
        return null;
    }

    //get min element
    public int min() {
        if(root==null) {
            return Integer.MIN_VALUE;
        } else {
            return root.min();
        }
    }

    //get max element
    public int max() {
        if(root==null) {
            return Integer.MAX_VALUE;
        } else {
            return root.max();
        }
```

```
	}
}
```

Class Main:

```
package Cau21;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Tree tree = new Tree();
	   tree.insert(5);
	   tree.insert(2);
	   tree.insert(1);
	   tree.insert(3);
	   tree.insert(7);
	   tree.insert(6);
	   tree.insert(8);
	   System.out.println("Binary search Pre-order:");
	   tree.traversePreOrder();
	}

}
```



Màn hình kết quả cho kết quả tương tự: 5, 2, 1, 3, 7, 6, 8

Chương 7: Thuật toán tìm kiếm và Thuật toán sắp xếp

Câu 22:

a. Ý tưởng thuật toán Quick Sort trên mảng:

Thuật toán Quick Sort là một thuật toán sắp xếp kiểu chia để trị (divide and conquer), sử dụng đệ quy trong việc sắp xếp. Thuật toán sử dụng một element gọi là pivot, là một phần tử nằm giữa của mảng để thực hiện chia mảng xét thành hai phần.

Sử dụng logic nếu những phần tử trong mảng duyệt tới < pivot, nó được chuyển đổi qua bên trái pivot, và ngược lại với những phần tử trong mảng duyệt tới > pivot.

Thuật toán dừng lại khi chỉ số duyệt thứ i từ trái sang tới pivot lớn hơn chỉ số duyệt thứ j từ phải sang tới pivot. Lúc này, để sắp xếp, ta gọi đệ quy lại 1 lần nữa để duyệt cho đến khi mảng được sắp xếp thành công.

So sánh các thuật toán:

Name	Best	Average	Worst	Memory	Stable	Method
Selection	n^2	n^2	n^2	1	No	Selection
Bubble	n	n^2	n^2	1	Yes	Exchanging
Insertion	n	n^2	n^2	1	Yes	Insertion
Quick	n * log(n)	n * log(n)	n^2	1	Depends	Partitioning

b, c.

Class QuickSort:

```
package Cau22;

public class QuickSort {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int[] intArray = {22, 12, 9, 7, 31, 19, 2, 27};
		System.out.println("The initial array:");
		printArray(intArray);
		System.out.println("The array after quick sort:");
		quickSort(intArray, 0, intArray.length - 1);
		printArray(intArray);
	}

	public static void printArray(int arr[]) {
		for (int i = 0; i < arr.length; i++) {
			System.out.print(arr[i] + " ");
		}
		System.out.println("");
	}

	public static void swap(int[] array, int i, int j) {
		if (i == j) {
			return;
		}
		int temp = array[i];
		array[i] = array[j];
		array[j] = temp;
	}

	public static void quickSort(int[] arr, int left, int right) {
```

```
            if (left >= right) {
                  return;
            } else {
                  int index = partition(arr, left, right);
                  if (left < index - 1) {
                        quickSort(arr, left, index - 1);
                  }
                  if (index < right) {
                        quickSort(arr, index, right);

                  }
            }
      }

      private static int partition(int[] arr, int left, int right) {
            // TODO Auto-generated method stub
            int i = left, j = right;
            int pivot = arr[(left + right) / 2];

            while (i <= j) {
                  while (arr[i] < pivot) {
                        i++;
                  }
                  while (arr[j] > pivot) {
                        j--;
                  }
                  if (i <= j) {
                        swap(arr, i, j);
                        printArray(arr);
                        i++;
                        j--;
                  }

            }
            return i;
      }
}
```

Câu 23:

a. Thuật toán tìm kiếm nhị phân (Binary Search) là thuật toán tiếp kiếm được sử dụng rất nhiều trong thực tế cho phép tìm kiếm vị trí của một phần tử trong một mảng **đã được sắp xếp**.

Thuật toán tìm kiếm nhị phân thực hiện tìm kiếm một mảng đã sắp xếp bằng cách liên tục chia các khoảng tìm kiếm thành 1 nửa. Bắt đầu với một khoảng từ phần tử đầu mảng, tới cuối mảng. Nếu giá trị của phần tử cần tìm nhỏ hơn giá trị của phần từ nằm ở giữa khoảng thì thu hẹp phạm vi tìm kiếm từ đầu mảng tới giữa mảng và ngược lại. Cứ thế tiếp tục chia phạm vi thành các nửa cho đến khi tìm thấy hoặc đã duyệt hết.

Ý tưởng thuật toán:

- Xét một đoạn trong mảng arr[left...right]. Lúc này giá trị của left và right lần lượt là 0 và số phần tử của mảng - 1.
- So sánh x với phần tử nằm ở vị trí chính giữa của mảng (mid = (left + right) /2). Nếu x bằng arr[mid] thì trả về vị trí và thoát vòng lặp.
- Nếu x < arr[mid] thì chắc chắn x sẽ nằm ở phía bên trái tức là từ arr[left....mid-1].
- Nếu x > arr[mid] thì chắc chắn x sẽ nằm ở phía bên phải mid tức là ở khoảng arr[mid+1...right].
- Tiếp tục thực hiện chia đôi các khoảng tìm kiếm tới khi nào tìm thấy được vị trí của x trong mảng hoặc khi đã duyệt hết mảng.

Triển khai thuật toán:
1. Theo đệ quy:
Class binarySearch:

```
public class binarySearch {

	public static int recursiveBinarySearch(int[] input, int start,
int end, int value) {
		if (start >= end) {
			return -1;
		}
		int midPoint = (start + end) / 2;
		System.out.println("Mid point: " + input[midPoint]);

		if (value == input[midPoint]) {
			return midPoint;
		}
		if (value > input[midPoint]) {
			return recursiveBinarySearch(input, midPoint + 1,
end, value);
		} else {
			return recursiveBinarySearch(input, start, midPoint,
value);
		}
	}
```

2. Theo Iterative:

Class binarySearch:

```
int iterativeBinarySearch(int arr[], int key, int start, int end) {
		while (end >= start) {
			int mid = (start + end) / 2;
			if (arr[mid] < key)
				start = mid + 1;
			else if (arr[mid] > key)
				end = mid - 1;
			else
				return mid;
		}
```

```
            return KEY_NOT_FOUND;
      }
```

b. Class Main:

```
package Cau23;

public class Main {

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            int[] intArray = { 2, 7, 1, 0, 3, 9, 5, 11, 8 };
            int start = 0;
            int end = intArray.length;
            quickSort(intArray, start, end - 1);
            System.out.println("Array after sorting: ");
            printArray(intArray);
            int rs = recursiveBinarySearch(intArray, start, end, 8);
            if (rs == -1) {
                  System.out.println("Can't found");
            } else {
                  System.out.println("Found the item at position " +
rs);
            }
      }

      public static void printArray(int arr[]) {
            for (int i = 0; i < arr.length; i++) {
                  System.out.print(arr[i] + " ");
            }
            System.out.println("");
      }

      // process to sort array before binarysearch
      public static void swap(int[] array, int i, int j) {
            if (i == j) {
                  return;
            }
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
      }

      public static void quickSort(int[] arr, int left, int right) {
            if (left >= right) {
                  return;
            } else {
                  int index = partition(arr, left, right);
```

```
                if (left < index - 1) {
                        quickSort(arr, left, index - 1);
                }
                if (index < right) {
                        quickSort(arr, index, right);

                }
        }
    }

    private static int partition(int[] arr, int left, int right) {
        // TODO Auto-generated method stub
        int i = left, j = right;
        int pivot = arr[(left + right) / 2];

        while (i <= j) {
            while (arr[i] < pivot) {
                i++;
            }
            while (arr[j] > pivot) {
                j--;
            }
            if (i <= j) {
                swap(arr, i, j);
                i++;
                j--;
            }

        }
        return i;
    }

    // binary search
    private static int recursiveBinarySearch(int[] intArray, int
start, int end, int value) {
        // TODO Auto-generated method stub
        if (start >= end) {
            return -1;
        }
        int midpoint = (start + end) / 2;
        System.out.println("midpoint = " + midpoint);
        if (intArray[midpoint] == value) {
            return midpoint;
        } else if (intArray[midpoint] < value) {
            return recursiveBinarySearch(intArray, midpoint + 1,
end, value);
        } else {
            return recursiveBinarySearch(intArray, start,
midpoint, value);
        }
    }
```

```
}
```

Câu 24:

a. Chúng ta có thể dùng thuật toán Quicksort để tiến hành sắp xếp dãy trên như sau:
Class QuickSort:

```
package Cau24;
public class QuickSort {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int[] intArray = {27, 32, 12, 52, 39, 76, 18, 4, 25, 69,
10, 8, 16};
		System.out.println("The initial array:");
		printArray(intArray);
		System.out.println("The array after quick sort:");
		quickSort(intArray, 0, intArray.length - 1);
		printArray(intArray);
	}

	public static void printArray(int arr[]) {
		for (int i = 0; i < arr.length; i++) {
			System.out.print(arr[i] + " ");
		}
		System.out.println("");
	}

	public static void swap(int[] array, int i, int j) {
		if (i == j) {
			return;
		}
		int temp = array[i];
		array[i] = array[j];
		array[j] = temp;
	}

	public static void quickSort(int[] arr, int left, int right) {
		if (left >= right) {
			return;
		} else {
			int index = partition(arr, left, right);
			if (left < index - 1) {
				quickSort(arr, left, index - 1);
			}
			if (index < right) {
				quickSort(arr, index, right);

			}
		}
	}
```

```
	private static int partition(int[] arr, int left, int right) {
		// TODO Auto-generated method stub
		int i = left, j = right;
		int pivot = arr[(left + right) / 2];

		while (i <= j) {
			while (arr[i] < pivot) {
				i++;
			}
			while (arr[j] > pivot) {
				j--;
			}
			if (i <= j) {
				swap(arr, i, j);
				printArray(arr);
				i++;
				j--;
			}

		}
		return i;
	}
}
```

Input: 27, 32, 12, 52, 39, 76, 18, 4, 25, 69, 10, 8, 16
Output: 4, 8, 10, 12, 16, 18, 25, 27, 32, 39, 52, 69, 76

b. Để thuật toán thực hiện sắp xếp theo thứ tự giảm dần, với đoạn code ở phần a. ta chỉ cần thay đổi nội dung trong phương thức partition:

Từ:

```
while (i <= j) {
			while (arr[i] < pivot) {
				i++;
			}
			while (arr[j] > pivot) {
				j--;
```

Thành:

```
while (i <= j) {
			while (arr[j] < pivot) {
				j--;
			}
			while (arr[i] > pivot) {
				i++;
			}
```

Ta có:

```
private static int partition(int[] arr, int left, int right) {
```

```
            // TODO Auto-generated method stub
            int i = left, j = right;
            int pivot = arr[(left + right) / 2];

            while (i <= j) {
                  while (arr[j] < pivot) {
                        j--;
                  }
                  while (arr[i] > pivot) {
                        i++;
                  }
                  if (i <= j) {
                        swap(arr, i, j);
                        i++;
                        j--;
                  }

            }
            return i;
      }
}
```

Input: 27, 32, 12, 52, 39, 76, 18, 4, 25, 69, 10, 8, 16

Output: 76, 69, 52, 39, 32, 27, 25, 18, 16, 12, 10, 8, 4

c. Độ phức tạp của thuật toán trên: QuickSort có hiệu quả với độ phức tạp thuật toán là O(nlog(n)). Độ phức tạp dữ liệu của thuật toán QuickSort tùy thuộc vào cách hiện thực. Tần suất tối ưu là thỉnh thoảng.

Hoạt động phân chia này diễn ra liên tục và chỉ dừng lại khi độ dài của mỗi phần tử con bằng 1. Để sắp xếp nhanh các phần tử con thu được thành một mảng hoàn chỉnh, người dùng sẽ sử dụng phương pháp đệ quy. Tất cả các thao tác này đều diễn ra trong thời gian tuyến tính. Và hơn nữa, QuickSort vẫn là một lựa chọn tốt vì Time Complexity trung bình = O(n log(n)) và Space Complexity tệ nhất cũng chỉ là O(log(n)).

d. Không có.

Câu 25: a,b. Hàm sắp xếp nổi bọt và in ra từng bước thực hiện của thuật toán bên dưới code như sau:

Class BubbleSort:

```
package Cau25;

import java.util.Arrays;

public class BubbleSort {

      public static void main(String[] args) {
```

```
		// TODO Auto-generated method stub
		int intArray[] = { 3, 8, 4, 9, 1, 2 };
		System.out.println("Initial Array:");
		System.out.println(Arrays.toString(intArray));
		bubbleSort(intArray);
		System.out.println("Array after bubble sort:");
		printArray(intArray);
		System.out.println(Arrays.toString(intArray));
	}

	private static void bubbleSort(int[] arr) {
		// TODO Auto-generated method stub
		int lastIndex = arr.length - 1;
		for (int last = lastIndex; last >= 0; last--) {
			System.out
					.println("unsortedPartitionIndex = " +
last + " -this is the last index of the unsorted partition");
			for (int i = 0; i < last; i++) {
				System.out.println("i = " + i + " - index used
to traverse the array from left to right");
				System.out.println("");
				if (arr[i] < arr[i + 1]) {
					swap(arr, i, i + 1);
				}
				System.out.println(Arrays.toString(arr));
			}
		}
	}

	private static void swap(int[] arr, int i, int j) {
		// TODO Auto-generated method stub
		if (i == j) {
			return;
		}
		int temp = arr[i];
		arr[i] = arr[j];
		arr[j] = temp;
	}

	private static void printArray(int[] arr) {
		// TODO Auto-generated method stub
		for (int i = 0; i <= arr.length - 1; i++) {
			System.out.print(arr[i] + " ");
		}
		System.out.println("");
	}

}
```

Câu 26:

Class Employee:

```
package Cau26;

public class Employee {
      private String ID;
    private String name;
    private int level;

    public Employee(String id, String name, int level) {
        this.ID = id;
        this.name = name;
        this.level = level;
    }
    public String getID() {
        return ID;
    }
    public String toString() {
        return "{ Id: " + ID + "; Name: " + name + "; Level: " +
level+" }";
    }
}
```

Class EmployeeArr:

```
package Cau26;

public class EmployeesArr {
      public int length;
      private Employee[] employees;

      public EmployeesArr(int size) {
            employees = new Employee[size];
      }

      public void insert(Employee e) {
            if (length == employees.length) {
                  Employee[] temp = new Employee[length * 2];
                  System.arraycopy(employees, 0, temp, 0,
employees.length);
                  employees = temp;
            }
            employees[length] = e;
            length++;
      }

      public Employee[] toArray() {
            return employees;
      }
```

```
	public void printList() {
		for (int i = 0; i < length; i++) {
			System.out.println(employees[i]);
		}
	}
}
```

Class BubbleSortEmployee:

```
package Cau26;

public class BubbleSortEmployee {
	public void sort(Employee[] employees) {
		for (int i = 0; i < employees.length; i++)
			for (int j = 1; j < employees.length - i; j++) {
				int prevId = Integer.parseInt(employees[j -
1].getID().substring(1));
				int currId =
Integer.parseInt(employees[j].getID().substring(1));
				if (prevId > currId)
					swap(employees, j - 1, j);
			}
	}

	public void sortByDescending(Employee[] employees) {
		for (int i = 0; i < employees.length; i++)
			for (int j = 1; j < employees.length - i; j++) {
				int prevId = Integer.parseInt(employees[j -
1].getID().substring(1));
				int currId =
Integer.parseInt(employees[j].getID().substring(1));
				if (prevId < currId)
					swap(employees, j - 1, j);
			}
	}

	public static void swap(Employee[] employees, int index1, int
index2) {
		Employee temp = employees[index1];
		employees[index1] = employees[index2];
		employees[index2] = temp;
	}
}
```

Class Main:

```
package Cau26;
```

```
public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Employee[] employees = new Employee[5];
		employees[0] = new Employee("A05", "Tran Quang", 7);
		employees[1] = new Employee("A03", "Nguyen An", 7);
		employees[2] = new Employee("A01", "Truong Phung", 5);
		employees[3] = new Employee("A04", "Pham Thi Lam", 2);
		employees[4] = new Employee("A02", "Do Trung Ton ", 5);

		System.out.println("Before sort: ");
		printEmployees(employees);

		// 2. Sort the list in ascending and descending order by
ID using different sort
		// algorithms (using bubble).
		// Sorting in ascending order
		System.out.println("\nSorting in ascending order by ID:
");
		sortAscending(employees);
		printEmployees(employees);

		// Sorting in descending order
		System.out.println("\nSorting in descending order by ID:
");
		sortDescending(employees);
		printEmployees(employees);

	}

	public static void printEmployees(Employee[] employees) {
		for (int i = 0; i < employees.length; i++) {
			System.out.println(employees[i]);
		}
	}

	// sorting in ascending method
	public static void sortAscending(Employee[] arr) {
		int lastIndex = arr.length - 1;
		for (int last = lastIndex; last >= 0; last--) {
			for (int i = 0; i < last; i++) {
				int prevId = Integer.parseInt(arr[i +
1].getID().substring(1));
				int currId =
Integer.parseInt(arr[i].getID().substring(1));
				if (prevId > currId) {
					swap(arr, i + 1, i);
				}

			}
```

```
            }
        }

        // sorting in descending method
        public static void sortDescending(Employee[] arr) {
            int lastIndex = arr.length - 1;
            for (int last = lastIndex; last >= 0; last--) {
                for (int i = 0; i < last; i++) {
                    int prevId = Integer.parseInt(arr[i +
1].getID().substring(1));
                    int currId =
Integer.parseInt(arr[i].getID().substring(1));
                    if (prevId < currId) {
                        swap(arr, i + 1, i);
                    }
                }
            }
        }

        public static void swap(Employee[] arr, int id1, int id2) {
            Employee temp = arr[id1];
            arr[id1] = arr[id2];
            arr[id2] = temp;
        }
}
```

Câu 27:

* Hash table (bảng băm), or a hash map, is a data structure that associates keys (names) with values (attributes).

- Look-Up Table
- Dictionary
- Cache
- Extended Array

* General Idea (Ý tưởng chung) :

- The novel concept for a hash table is the use of a hash function to map general keys to corresponding indices in a table.
- Ideally, keys will be well distributed in the range from 0 to N−1 by a hash function, but in practice there may be two or more distinct keys that get mapped to the same index.
- As a result, we will conceptualize our table as a bucket array, in which each bucket may manage a collection of entries that are sent to a specific index by the hash function. (To save space, an empty bucket may be replaced by

a null reference.)

* Solution to solve Collision Handling Schemes

1. Separate chaining (open hashing)

- Separate chaining là một kỹ thuật xử lý và chạm phổ biến nhất. Nó thường được cài đặt với danh sách liên kết. Để lưu giữ một phần tử trong bảng băm, bạn phải thêm nó vào một danh sách liên kết ứng với chỉ mục của nó. Nếu có sự va chạm xảy ra, các phần tử đó sẽ nằm cùng trong 1 danh sách liên kết.

- Như vậy, kỹ thuật này sẽ đạt được tốc độ tìm kiếm O(1) trong trường hợp tối ưu và O(N) nếu tất cả các phần tử ở cùng 1 danh sách liên kết duy nhất. Đó là do có điều kiện 3 trong tiêu chí hàm băm tốt.

2. Linear probing (open addressing or closed hashing)

   Trong kỹ thuật xử lý va chạm này, chúng ta sẽ không dùng linklist để lưu trữ mà chỉ có bản thân array đó thôi.

   Khi thêm vào bảng băm, nếu chỉ mục đó đã có phần tử rồi; Giá trị chỉ mục sẽ được tính toán lại theo cơ chế tuần tự. Giả sử rằng chỉ mục là chỉ số của mảng, khi đó, việc tính toán chỉ mục cho phần tử được tính theo cách sau:

   index = index % hashTableSize
   index = (index + 1) % hashTableSize
   index = (index + 2) % hashTableSize
   index = (index + 3) % hashTableSize

   Và cứ thế theo cách như vậy chừng nào index thu được chưa có phần tử được sử dụng. Tất nhiên, không gian chỉ mục phải được đảm bảo để luôn có chỗ cho phần tử mới.

Ứng dụng Hash vào 1 số chương trình:

1. Define My Dictionary

Class Main:

```
package Cau27;

import java.util.Hashtable;

public class Main {

	public static void main(String[] args) {
		Hashtable<String, String> myDictionary = new
Hashtable<String, String>();
		System.out.println("" + myDictionary);
```

```
            myDictionary.put("happy", "having or showing a feeling of
pleasure or contentment");
            myDictionary.put("lucky", "good things happened in my
life");
            myDictionary.put("one", "a number with the count one");
            System.out.println("" + myDictionary);

            // Print
            for (String key : myDictionary.keySet()) {
                  String value = myDictionary.get(key);
                  System.out.println(key + " means " + value);
            }
      }
}
```

2. Manage Customer buy Fruit:

Class Customer:

```
package HashTable;

public class Customer {
      private int id;
      private String name;
      private String address;

      public Customer(){

      }

      public Customer(int id, String name, String address) {
            this.id = id;
            this.name = name;
            this.address = address;
      }

      public int getId() {
            return id;
      }

      public void setId(int id) {
            this.id = id;
      }

      public String getName() {
            return name;
      }

      public void setName(String name) {
            this.name = name;
```

```
	}

	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}

	@Override
	public String toString() {
		return "Customer {id=" + id + ", name=" + name + ",
address=" + address + "}";
	}

}
```

Class FruitList:

```
package HashTable;

import java.util.ArrayList;

public class FruitList {

	private ArrayList<String> myFruits = new ArrayList<>();

	public FruitList() {
	}

	public void enter(String nameFruit) {
		myFruits.add(nameFruit);
	}

	public ArrayList<String> getMyFruits() {
		return myFruits;
	}

	public void setMyFruits(ArrayList<String> myFruits) {
		this.myFruits = myFruits;
	}

	@Override
	public String toString() {
		return "FruitList{" + "myFruits=" + myFruits + '}';
	}

}
```

Class Main:

```
package HashTable;

import java.util.Hashtable;

public class Main {

	public static void main(String[] args) {

		// Save info
		Hashtable<Customer, FruitList> mySales = new
Hashtable<Customer, FruitList>();

		Customer cus01 = new Customer(111, "Tran", "ABC Street,
Da Nang");
		System.out.println("" + cus01);
		FruitList fruitList01 = new FruitList();
		fruitList01.enter("Apple");
		fruitList01.enter("Mango");
		fruitList01.enter("Cherry");
		System.out.println("" + fruitList01.getMyFruits());
		Customer cus02 = new Customer(222, "B", "XYZ Street, Da
Nang");
		FruitList fruitList02 = new FruitList();
		fruitList02.enter("WaterMelon");
		mySales.put(cus01, fruitList01);
		mySales.put(cus02, fruitList02);
		System.out.println("" + mySales);

		// Print
		for (Customer key : mySales.keySet()) {
			FruitList value = mySales.get(key);
			System.out.println(key + " – BUY - " + value);
		}
	}

}
```