Java SOS Sheet

import java.util.*; //ไม่ว่าโจทย์ให้ทำไรบรรทัดแรกใส่คำสั่งนี้ก่อน (import ทุกอย่าง)

```
/* private เข้าได้แค่ Class มันเอง
   public เข้าได้จากทุก Class
   protected เข้าได้แค่จาก Class ที่ extends (Sub class) และ class ตัวเอง
*/
public abstract class ClassName {
   private int field; // field

   public ClassName(int field) { // Constructer
      this.field = field;
   }
   public void concreteMethod() { // Method ปกติ
      System.out.println("This is a concrete method.");
   }
   public abstract void abstractMethod(); //Abstract Method รอ implementation ในsubClass
}
```

ประกาศ subclass ที่สืบทอด abstract class

```
public class SubClass extends ClassName {
   public SubClass(int field) { //ต้องมี constructor เรียกsuper(...)
      super(field); //ส่งข้อมูลไปให้ parent class
   }
   @Override//เขียนทับ method ที่ implement ไปแล้ว
   public void abstractMethod() {
      System.out.println("Implemented แม่งชะ");
      /*ถ้า subclass ไม่ override ทุก abstract method
       subclass ต้องประกาศเป็น abstract ต่อไป*/
   }
}
```

วิธีนำไปใช้

```
public class Main {
   public static void main(String[] args) {
      // ไม่สามารถ new Abstract Class ได้ตรง ๆ
      // ClassName obj = new ClassName(10);
      // อย่าหาทำนะไอชห

      // สร้างผ่าน SubClass ที่ implement ครบแล้ว
      ClassName obj = new SubClass(10);    // ทำแบบนี้
      obj.abstractMethod();    // เรียกเมธอดที่ implement แล้ว
      obj.concreteMethod();    // เรียกเมธอดธรรมดา
   }
}
```

ประกาศ Scanner ไว้รับ input

Vguin

```
Scanner sc = new Scanner(System.in); //Instantiate
Object myObj = sc.next(); //รับ input
int number = sc.nextInt(); //รับ int เท่านั้น
sc.close(); //ปิดScanner อารมณ์ free() ใน c
```

Java Hashmap

เป็น Data structure ที่เก็บข้อมูลเป็นคู่คือ key และ value ประกาศงี้

```
Map<KeyType, ValueType> ObjMap = new Hashmap<>();
```

มี method คร่าวๆ ปมนี้

```
ObjMap.get(key); //return value from inserted key
ObjMap.put(key, value); //add or update key and value
ObjMap.remove(key); //remove key from the map
ObjMap.containsKey(key) or ObjMap.containsValue(value);
//Check if a key or value exists
ObjMap.size(); //return map size
ObjMap.isEmpty(); // เช็คว่า map ว่างหรือไม่
ObjMap.replace(key, newValue);  // เปลี่ยนค่า value ใน map
ObjMap.keyset(); // return key ทั้งหมดที่มี
```

--- Java Array ---

```
Type[][] array = new Type[size][size];
```

--- Java StringBuffer ---

เป็น String ที่แก้ไขได้ด้วย method ต่าง ๆ ประกาศงี้

```
StringBuffer sb = new StringBuffer(); //ข้างในจะเป็นว่าง ๆ งี้หรือ text อะไรก็ได้
sb.append(String s); //เพิ่ม string ไปต่อ ใช้ซ้อน ๆ ได้ append(s).append(“ ”);
sb.reverse(); //reverse text
sb.ToString(); //change StringBuffer to String
```

--- Java List ---

คล้าย Array แต่ว่าเก็บค่าข้างในเป็น Object และไม่ต้องประกาศขนาด

```
List<Integer> ObjList = new ArrayList<>(); //ArrayList
List<String> ObjList = new LinkedList<>(); // LinkedList
ObjList.get(index); //return element from inserted index ถ้า index เป็น list อีกทีนึง จะ print ของใน list นั้นทั้งหมด
ObjList.set(index, element); //update element at given index
ObjList.add(element); //add element at the end
ObjList.add(index, element); //add element at given index
ObjList.remove(index); //remove element at given index
list.remove("o");  // ลบ element แรกที่ตรงกับ o
ObjList.contains(element); //Check if an element exists
ObjList.size(); //return list size
ObjList.isEmpty(); // เช็คว่า list ว่างหรือไม่
```

PriorityQueue

Comparator.comparingInt (p -> p.second) //บอกว่าคิดลำดับความสำคัญจาก pair ตัวที่ 2 เรียงจากน้อยไปมาก

.peek() ดึงตัวแรกมาดูแต่ไม่เอาออก

--- Tree node structure ---

```
class TreeNode { // ประกาศ Tree
  int data;
  //int height; for AVL Tree
  TreeNode leftChild;
  TreeNode rightChild;

  public TreeNode(int data) {
    this.data = data;
    //this.height = height แล้วรับ height ใน parameter ด้วย
  }
}
```

--- Node structure ---

```
class Node{ // ประกาศ Node
  private int value;
  private Node next;

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

Adjacency List (GraphL)

เป็นการ represent graph ในรูปแบบ map ของ <Node, List of neighbour(s)> บอกว่า Node มี edge ไปไหนบ้าง เป็น Pair<>(dest, weight)

GraphL(constructor), input: isDirected, numVertices, numEdges

- set the graph's isDirected property to be isDirected
- set the graph's numVertices property to be numVertices
- set the graph's numEdges property to be numEdges

addEdge, input: src, dest, weight

- if adjacencyList doesn't have src THEN put src and its new ArrayList<>() into the list
- if adjacencyList doesn't have dest THEN put dest and its new ArrayList<>() into the list
- add a new pair of dest-weight into the src key in adjacencyList
- if the graph isn't a directed graph THEN add a new pair of src-weight into the dest key in adjacencyList

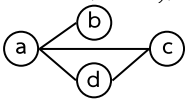
---

Adjacency Matrix (GraphM)

การ represent graph แบบ matrix(2D Array)โดยมีขนาดเป็น n * n | (n = #vertrices)

เช่น

```
   a b c d
a [ 0 1 1 1 ]
b [ 1 0 0 0 ]
c [ 1 0 0 1 ]
d [ 1 0 1 0 ]
```



GraphM(constructor), input: n, isDirected

- Send n to the mother class
- set the graph's isDirected property to be isDirected
- set the graph's numVertices property to be n
- iteration of the graph's matrix
  - fill all positions in the matrix to be 0

addEdge, input: i, j, weight

- set the position i-j in the graph's matrix to be the weight
- if the graph isn't a directed graph THEN also set the position j-i to be the weight

---

BST (Binary Search Tree)

insertNode, input: Value

- create a new BinNode(We'll call it newNode) with Value as it's value
- if the tree's root is null THEN the tree's root is newNode ELSE call insertNode, input: tree's root, newNode

insertNode, input: parent, newNode

- if the value of newNode is lower than the value of parent THEN
  - if the parent's leftChild is null THEN the parent's leftChild is newNode
  - ELSE insertNode, input: parent's leftChild, newNode
- ELSE if the value of newNode is higher than the value of parent THEN
  - if the parent's rightChild is null THEN the parent's rightChild is newNode
  - ELSE insertNode, input: parent's rightChild, newNode
- ELSE return

---Find Balance Factor in BST---

getBalance, type: integer, input: node(BinNode)

- if node is null THEN return 0
- assign a new integer named LeftHeight to be the result of Height, input: node's leftChild
- assign a new integer named RightHeight to be the result of Height, input: node's rightChild
- return the value of LeftHeight - RightHeight

Height, type: integer, input: root(BinNode)

- if root is null THEN return 0
- assign a new integer named LeftHeight to be the result of Height, input: root's leftChild
- assign a new integer named RightHeight to be the result of Height, input: root's rightChild
- return 1 + (the one that is higher: LeftHeight or RightHeight)

---AVL Tree (setup)---

- ก่อนจะเริ่มทำ AVL Tree อย่าลืม ใส่ค่า height - integer เพิ่มเข้าไปใน BinNode รวมไปถึง constructor ของ BinNode ให้ height มีค่าเป็น 1 โดย default ด้วยนะจ๊ะ
- สร้าง AVL.java ขึ้นมา ให้ extends ออกมาจาก BinTree เหมือนกับ BST.java
- constructor AVL ทำเหมือน BST เป๊ะๆ

rotateRight, type: BinNode, input: y(BinNode)

- declare a new BinNode(we'll call it x) and assign it with y's leftChild
- declare a new BinNode(we'll call it T2) and assign it with x's rightChild
- y becomes x's rightChild and T2 becomes y's leftChild
- y's height becomes 1 + (the one that is higher: y's leftChild's Height or y's rightChild's Height)
- x's height becomes 1 + (the one that is higher: x's leftChild's Height or x's rightChild's Height)
- return x

insertNode, type: void, input: Value

- create a new BinNode(We'll call it newNode) with Value as its value
- call insertNodeHelper, input: parent(BinNode), newNode(BinNode) and assign the result to the tree's root(Because the root may change in AVL's insertion)

insertNodeHelper, type: BinNode, input: parent(BinNode), newNode(BinNode)

- if parent is null THEN return newNode
- if newNode's value is lower than parent's value THEN call insertNodeHelper, input: parent's leftChild, newNode and assign the result to parent's leftChild
- ELSE if newNode's value is higher than parent's value THEN call insertNodeHelper, input: parent's rightChild, newNode and assign the result to parent's rightChild
- ELSE return parent (It's a duplication, so we do nothing)
- set the parent's height to be 1 + (the one that is higher: parent's leftChild's Height or parent's rightChild's Height)
- declare a new integer named Balance and assign it with (parent's leftChild's Height - parent's rightChild's Height)
- if Balance > 1 AND newNode's value is lower than parent's leftChild's value THEN return the result of rotateRight, input: parent
- if Balance < -1 AND newNode's value is higher than parent's rightChild's value THEN return the result of rotateLeft, input: parent
- if Balance > 1 AND newNode's value is higher than parent's leftChild's value THEN assign the result of rotateLeft, input: parent's leftChild to parent'sleftChild, THEN return the result of rotateRight, input: parent
- if Balance < -1 AND newNode's value is lower than parent's rightChild's value THEN assign the result of rotateRight, input: parent's rightChild to parent's rightChild, THEN return the result of rotateLeft, input: parent
- return parent

Height, type: integer, input: node(BinNode)

- -if node is null THEN return 0 ELSE return node's height

rotateLeft, type: BinNode, input: x(BinNode)

- do everything just like rotateRight but switch y <-> x and switch left <-> right
- return y

--- Java Stack (Last In First Out) ---

Stack<ObjectType> stack = new Stack<>(); //support Java V.เก่า

Deque<ObjectType> stack = new ArrayDeque<>(); //support Java V.ใหม่

stack.push(elements); //push an element at the top of the stack

type name = stack.pop(); //pop the element out of the stack

type name = stack.peek(); //return the top element but doesn't remove it

int name = stack.size(); //return stack size

boolean name = stack.isEmpty(); //check if stack is empty, return boolean

-- Java Queue (First In First Out) ---

Queue<Integer> queue = new LinkedList<>();

queue.add(element); //add element at the last index

queue.offer(element); // เหมือน add แต่โค้ดจะทำงานต่อ ถ้าระเบิด เช่น queue เต็ม

queue.remove(); //remove the front(first element)

type name = queue.poll(); //remove and return front

type name = queue.peek(); //return front without removing

boolean name = queue.isEmpty(); //check if queue is empty

BFS, type: List<Integer>, input: GraphL, startNode

- create an array(**isVisited** - boolean) the size of number of vertices, fill it with false
- create an empty list(**Result** - integer)
- create an empty queue(**q** - integer)
- mark the **startNode** in isVisited as **true**
- add the **startNode** into **q**
- main iteration **while** q is **not empty**
  - create **currNode** as the first member in q and dequeue (.poll)
  - add **currNode** to **Result**
  - create a **List pair**, fill it with get currNode(**neighbors-**integer,integer) *//edge list of currNode*
  - if **neighbors** isn't empty Then
    - sub iteration of Pair **List of neighbors**(pair-integer,integer): neighbors *//for-each*
      - create **Adj** as the first of pair
      - if **Adj isn't visited**, THEN
        - mark Adj in isVisited as true
        - add Adj into q
- return **Result**

DFS, type: List<Integer>, input: GraphM, startNode

- create an array(isVisited - boolean) the size of number of vertices, fill it with false
- create an empty list(Result - integer)
- call DFSUtil, input: GraphM, startNode, isVisited, Result
- return Result

DFSUtil, type: void, input: GraphM, currNode, isVisited, Result

- if currNode hasn't been visited yet THEN
  - mark currNode as already visited
  - add currNode to result
  - main iteration of all nodes, we'll call it loop "i"
    - if there's an edge between "currNode" - "i" in the matrix AND "i" hasn't been visited yet THEN call DFSUtil, input: GraphM, i, isVisited, Result

Floyd, type: integer, input: GraphM, startNode, destNode

- create a 2D matrix(**Shortest** - integer) with the same size of GraphM
- setup iteration *//Nested i, j | position is index*
  - **If** it to itself (i==j: 0-0, 1-1, ...), set those positions in **Shortest to 0**
  - **ELSE IF GraphM at position i-j isn't empty** (is not 0) THEN set the position i-j of **Shortest** with the **value from position i-j** in **GraphM matrix**
  - **ELSE**, set position i-j of **Shortest** to infinity/2 (prevent overflow)
- 3 nested iterations of all nodes in Shortest, we will call it loop "k", "i", "j"
  - **IF** both **Shortest i-k** AND **k-j isn't infinity AND Shortest** i-k + k-j < i-j THEN update the **Shortest** i-j to **i-k + k-j**
- if the position of startNode-destNode in Shortest is still infinity THEN return -1 (or whatever you want to define as unreachable)
- **return** the **Shortest** of startNode-destNode

Warshall, type: boolean, input: GraphM, startNode, destNode

- create a 2D matrix (isConnected - boolean)
- for all positions in GraphM that have length(edge), fill those positions in isConnected i,j as (graph of matrix i,j isn't zero)
- 3 nested iterations of all nodes, we will call it loop "k", loop "i", loop "j"
  - if "i" is connected to "j" OR "i" is connected to "k" AND "k" is connected to "j" THEN mark i connecting to j as true
- return the boolean of startNode, destNode from the isConnected 2Dmatrix

---

Dijkstra, type: integer, input: GraphL, startNode, destNode

- create 2 arrays(leastDistance - integer, isVisited - boolean) and fill it with infinity, false (size is amount of vertices of gr
- mark the startNode in leastDistance as 0
- main iteration of all nodes
  - create currNode as -1 (or anything that marks it as not found)
  - create Min as infinity
  - sub iteration of all nodes (To find the starting point)
    - find the node with the lowest distance in leastDistance (if a node in leastDistance has a lower distance than Min AND this node hasn't been visited yet, THEN mark the currNode as that node, change the Min to that distance. Repeating this checking method for all nodes)
  - if the currNode is still -1, THEN end the main loop
  - otherwise, mark the currNode in isVisited as true
  - sub iteration of all pairs of neighbor,weight in the currNode (To update the shortest path)
    - create neighborNode and fill it with first of neighbor
    - create weight and fill it with second of neighbor
    - if the neighbor of the currNode isn't visited AND the (pair's weight + the currNode's distance in leastDistance) is lower than the neighbor's distance in leastDistance, THEN change the neighbor's distance in leastDistance to the (pair's weight + the currNode's distance in leastDistance) Repeating this checking method for all pairs of the currNode
- return the distance of destNode in leastDistance

Java Fucking Sheet