Họ và tên: Phạm Thành An

Lớp: CNTT3-K63

Mã sinh viên: 221230727

Lớp học phần: N03

**BÁO CÁO BÀI THI**

- Số bài tập đã làm 22/23

- Số bài tập chưa làm 1/23 (bài số 23)

**Bài 1:**

\* Class IndexMaxPQ bao gồm:

- Các thuộc tính:

+ private int n: số lượng phần tử có trong PriorityQueue.

+ private int[] pq: dùng để lưu chỉ mục của một phần tử.

+ private int[] qp: dùng để lưu nghịch đảo của pq chỉ ra là phần tử thứ mấy trong PriorityQueue.

+ private Key[] keys: dùng để lưu giá trị của key ở chỉ mục i.

- Các phương thức:

+ public boolean isEmpty(): kiểm tra xem PriorityQueue có rỗng không.

+ public boolean contains(int i): kiểm tra xem PriorityQueue có chứa chỉ mục i không.

+ public int size(): trả về kích thước của hàng đợi ưu tiên.

+ public void insert(int i, Key key): chèn một phần tử có chỉ mục i và có giá trị key vào hàng đợi ưu tiên.

+ private void swim(int k): giúp xây dựng một cấu trúc max heap (phần tử cha luôn lớn hơn phần tử con).

+ private void exch(int i, int j): giúp hoán vị hai phần tử có chỉ mục i và j.

+ private boolean less(int i, int j): giúp so sánh hai key ở hai chỉ mục i và j.

+ public int maxIndex(): trả về chỉ mục với khóa là lớn nhất.

+ public Key maxKey(): trả về key lớn nhất.

+ public int delMax(): xóa đi khóa lớn nhất, trả về chỉ mục của khóa lớn nhất đó.

+ private void sink(int k): giúp xây dựng một cấu trúc max heap (phần tử cha luôn lớn hơn phần tử con).

+ public Key keyOf(int i): trả về giá trị của key với chỉ mục i.

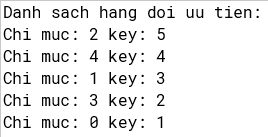
+ public void changeKey(int i, Key key): thay đổi giá trị của key tại chỉ mục i.

+ public void increaseKey(int i, Key key): tăng key được liên kết với chỉ mục i lên giá trị key được chỉ định.

+ public void decreaseKey(int i, Key key): giảm key được liên kết với chỉ mục i lên giá trị key được chỉ định.

+ public void delete(int i): xóa key trên hàng đợi ưu tiên được liên kết với chỉ mục i.

\* Kết quả:



**Bài 2:**

\* Class DijkstraSP\_T bao gồm:

- Các thuộc tính:

+ private double[] distTo: distTo[v] = đường đi ngắn nhất từ đỉnh s -> v.

+ private DirectedEdge[] edgeTo: edgeTo[v] = cạnh cuối ngắn nhất từ đỉnh s -> v.

+ private IndexMinPQ<Double> pq: hàng đợi ưu tiên lưu đỉnh và đường đi ngắn nhất đến chúng.

- Các phương thức:

+ public DijkstraSP\_T(EdgeWeightedDigraph\_T G, int s): tìm đường đi ngắn nhất từ mọi đỉnh đến đỉnh đích.

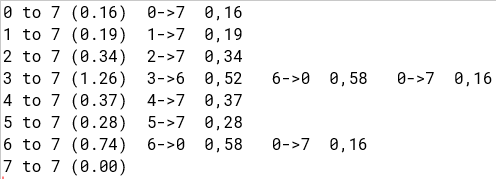
+ private void relax(DirectedEdge e): relax cạnh e và update lại hàng đợi.

+ public double distTo(int v): độ dài đường đi ngắn nhất đển đỉnh v.

+ public boolean hasPathTo(int v): check xem có đường đi từ đỉnh v không.

+ public Iterable<DirectedEdge> pathTo(int v): các đường đi ngắn từ đỉnh v đến đích.

\* Kết quả:



**Bài 3:**

\* Class DijkstraAllPairsSP bao gồm:

- Các thuộc tính:

+ private DijkstraSP[] all: đường đi ngắn nhất từ một đỉnh đến mọi đỉnh.

- Các phương thức:

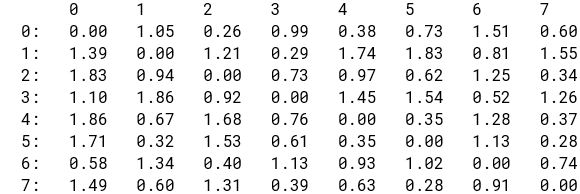
+ public Iterable<DirectedEdge> path(int s, int t): đường đi ngắn nhất từ đỉnh s đến t.

+ public boolean hasPath(int s, int t): kiểm tra xem có đường đi từ đỉnh s đến đỉnh t không.

+ public double dist(int s, int t): độ dài đường đi ngắn nhất từ đỉnh s đến đỉnh t.

+ private void validateVertex(int v): kiểm tra xem có tồn tài đỉnh v không.

\* Kết quả:



**Bài 4:**

\* Class PrimMST bao gồm:

- Các thuộc tính:

+ private Edge[] edgeTo: dùng để lưu cạnh ngắn nhất từ một đỉnh nằm trong tập MST đến đỉnh chưa nằm trong tập MST.

+ private double[] distTo: dùng để lưu trọng số ngắn nhất đến đỉnh v.

+ private boolean[] marked: đánh xấu đỉnh đấy đã nằm trong tập MST chưa.

+ private IndexMinPQ<Double> pq: lưu các trọng số theo thứ tự từ bé đến lớn.

- Các phương thức:

+ public PrimMST(EdgeWeightedGraph G): khởi tạo đồ thị vô hướng để tìm cây khung nhỏ nhất.

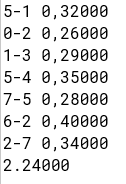
+ private void prim(EdgeWeightedGraph G, int s): chạy thuật toán Prim trên đồ thị G bắt đầu từ đỉnh S.

+ private void scan(EdgeWeightedGraph G, int v): duyệt và xử lí các cạnh kề với đỉnh v trong đồ thị G.

+ public Iterable<Edge> edges(): trả về danh sách các cạnh xây đựng được để tạo nên cây khung nhỏ nhất.

+ public double weight(): trả về tổng trọng số của các cạnh trong cây khung nhỏ nhất.

\* Kết quả:



**Bài 5:**

\* Class BellmanFordSP\_T bao gồm:

- Các thuộc tính:

+ private double[] distTo: distTo[v] = đường đi ngắn nhất từ đỉnh s -> v.

+ private DirectedEdge[] edgeTo: edgeTo[v] = cạnh cuối ngắn nhất từ đỉnh s -> v.

+ private boolean[] onQueue: onQueue[v] kiểm tra xem đỉnh v có ở trong hàng đợi hay không.

+ private Queue<Integer> queue: số đỉnh có trong hàng đợi để relax.

+ private int cost: số lần gọi relax.

+ private Iterable<DirectedEdge> cycle: dùng để lưu các cạnh tạo thành chu trình.

- Các phương thức:

+ public BellmanFordSP\_T(EdgeWeightedDigraph\_T G, int s): tìm đường đi ngắn nhất từ mọi đỉnh đến đỉnh đích.

+ private void relax(EdgeWeightedDigraph\_T G, int v): relax cạnh e và update lại hàng đợi.

+ public boolean hasNegativeCycle(): kiểm tra xem có chu trình âm không.

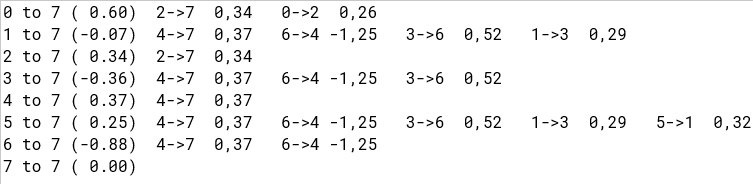
+ private void findNegativeCycle(): tìm ra chu trình âm nếu có.

+ public double distTo(int v): độ dài đường đi ngắn nhất đển đỉnh v.

+ public boolean hasPathTo(int v): check xem có đường đi từ đỉnh v không.

+ public Iterable<DirectedEdge> pathTo(int v): các đường đi ngắn từ đỉnh v đến đích.

\* Kết quả:



**Bài 6:**

\* Class KnapsackImp bao gồm:

- Các thuộc tính:

+ private int W: dung tích tối đa cho phép.

+ private int realWeight = 0: dung tích thực tế.

+ private int totalProfit = 0: tổng giá trị của các đồ vật.

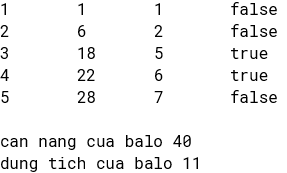
+ private Bag<Item> bag = new Bag<Item>(): nơi lưu trữ các đồ vật.

- Các phương thức:

+ public KnapsackImp(ArrayList<Item> items,int W): chọn ra các đồ vật sao cho balo không vượt quá dung tích W và tổng các giá trị đồ vật là lớn nhất.

+ Các hàm getter và setter.

\* Kết quả:



**Bài 7:**

\* Class Nqueen bao gồm:

- Các thuộc tính:

+ static int N = 8: một biến thể hiện bàn cờ vua cỡ N x N mặc định N = 8 và N cũng là số quân hậu cần đặt vào bàn cờ.

+ static int k = 1: số phép giải cho bài toán Nqueen mặc định là 1.

- Các phương thức:

+ public NQueen(int n): constructor giúp khởi tạo bàn cờ vua cỡ n x n và n cũng là số quân hậu cần đặt vào bàn cờ.

+ static void print(int board[][]): in ra vị trí của N quân hậu theo hàng.

+ static void printByCol(int board[][]): in ra vị trí của N quân hậu theo cột.

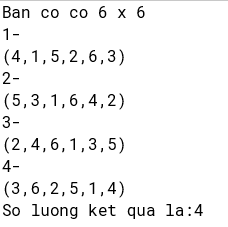
+ static boolean isSafe(int board[][], int row, int col): Một hàm tiện ích để kiểm tra xem quân hậu có thể được đặt ở vị trí hàng row và cột col không (Lưu ý rằng điều này hàm được gọi khi có các Queen "col" đã được đặt trong các cột từ 0 đến col -1.Vì vậy chúng ta chỉ cần kiểm tra bên trái tấn công Queen).

+ static boolean solveNQUtil(int board[][], int col): hàm đệ quy giúp giải quyết bài toán N quân hậu.

+ static void inKetQua(): in ra tất cả các trường hợp đặt n quân hậu thỏa mãn vào bàn cờ n x n.

+ static void inSoLuongKQ(): in ra số lượng kết quả của bài toán.

\* Kết quả:



**Bài 8:** Giải thích các chương trình ứng dụng

**Bài 9:**

\* Class FileFrequencyIndex bao gồm:

- Các thuộc tính:

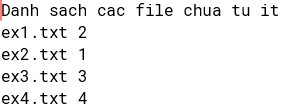
+ private ST<String, ST<File,Integer>> st: là một simple table cho biết với một từ thì có những file nào chứa từ đó và file đó chứa từ số lượng từ đó là bao nhiêu.

- Các phương thức:

+ public void addFile(File file): thêm một file vào trong.

+ public void query(String word): truy xuất danh sách các file chứa từ đó cùng với đó là in ra trong file đó chứa từ đó bao nhiêu lần.

\* Kết quả:



**Bài 10:**

\* Class Student gồm có:

- Các thuộc tính:

+ private String ho, hodem, ten, ns, ma, sdt.

+ private Double dtb: điểm trung bình của sinh viên.

+ private ST<Mon,Double> st = new ST<>(): môn và điểm từng môn của sinh viên.

- Các phương thức:

+ Các phương thức getter và setter.

+ public Student(String other): khởi tạo một đối tượng sinh viên thông qua tham số là một chuỗi.

+ public double tinhDiemTBC(int kythu): tính điểm trung bình của một kì.

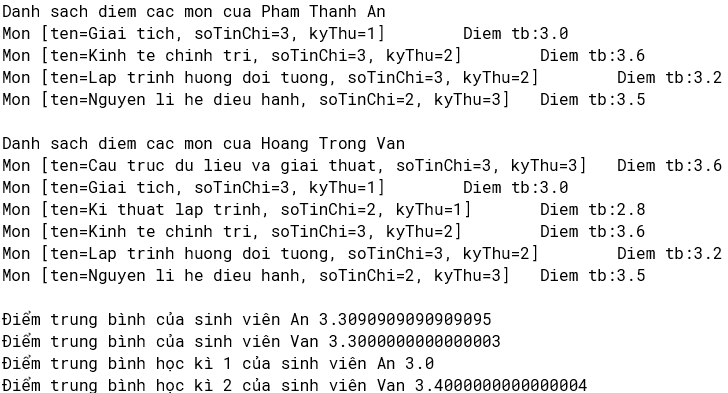
+ public double tinhDiemTBC(): tính điểm trung bình cộng tích lũy của sinh viên đó.

+ public void nhapDiemMonSv(Mon m,Double diem): nhập điểm một môn của sinh viên.

+ public void inHoSoSinhVien(): in ra điểm trung bình tích lũy của sinh viên đó và điểm các môn học của sinh viên đó.

+ public int compareTo(Student o): hàm giúp so sánh giữa hai đối tượng Student.

\* Kết quả:



**Bài 11:**

\* Class DSLop gồm có:

- Các thuộc tính:

+ ST<Student,ST<Mon,Double>> st: lưu trữ sinh viên trong một lớp và điểm các môn của sinh viên đó.

- Các phương thức:

+ DSLop(String[] args): constructor giúp khởi tạo sinh viên trong một lớp.

+ public static void xetThiDuaHocKy(int hocKy): xét thi đua học kì in ra 10 sinh viên có điểm trung bình cao nhất.

+ public static void danhSachDiemCuaTungSinhVien(): danh sách điểm của từng sinh viên trong một lớp.

+ public void DanhSachLopTheoTen(): danh sách sinh viên theo tên.

+ public void DanhSachLopTheoNgaySinh(): danh sách sinh viên theo ngày sinh.

+ public void DanhSachLopTheoQue(String quecantim): danh sách sinh viên có cùng quê.

\* Kết quả:

|  |  |
| --- | --- |
|  |  |
|  |  |

**Bài 12:**

\* Class Student gồm có:

- Các thuộc tính: giống bài 11.

- Các phương thức:

+ giống bài 11 có thêm các phương thức ở dưới.

+ public int compareDTBC(Student o1, Student o2): so sánh điểm trung bình của 2 sinh viên.

+ public static class DTBC implements Comparator<Student>: class kế thừa comparator giúp sắp xếp hai sinh viên theo điểm trung bình.

\* Kết quả:

|  |  |
| --- | --- |
|  |  |
|  |  |

**Bài 13:**

\* Class HashMapSinhVien gồm có:

- Các thuộc tính:

+ private static int i=1: đếm số lượng sinh viên.

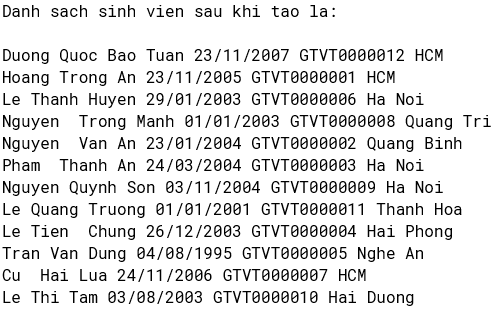
+ private static HashMap<Student,String> hs = new HashMap<>(): lưu danh sách sinh viên và mã của sinh viên mới tạo.

- Các phương thức:

+ private static int demSoLuongChuSo(int u): đếm số lượng số chữ số của một số.

+ public static String generateStudentID(): tạo mã sinh viên cho sinh viên mới vào trường.

\* Kết quả:



**Bài 14:**

\* Class DSLop bao gồm:

- Các thuộc tính:

+ private static ArrayList<Student> arr: lưu trữ danh sách lớp.

- Các phương thức:

+ public DSLop(String[] args): constructor giúp tạo danh sách lớp.

+ public static void inDanhSachLop(): giúp in danh sách lớp.

+ public static void inDanhSachDongHuong(String ten): in danh sách đồng hương.

+ public static void inDanhSachSinhVien(Double diem): in danh sách sinh viên theo điểm.

\* Kết quả:

|  |  |
| --- | --- |
|  |  |

**Bài 15:**

\* Class FordFulkerson bao gồm:

- Các thuộc tính:

+ private final int V: số lượng đỉnh.

+ private boolean[] marked: marked[v] = true nếu có đường đi từ s->v trên đồ thị phần dư.

+ private FlowEdge[] edgeTo: edgeTo[v] = cạnh cuối cùng trên đường đi s->v ở đồ thị phần dư là ngắn nhất.

+ private double value: giá trị luồng max.

- Các phương thức:

+ public FordFulkerson(FlowNetwork G, int s, int t): tính toán luồng max tối đa và lát cắt tổi thiểu.

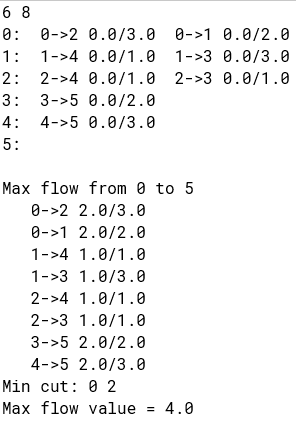
+ public double value(): trả về giá trị max của luồng.

+ public boolean inCut(int v): trả về true nếu đỉnh v nằm trong lát cắt tối thiểu thuộc bên chứa đỉnh xuất phát.

+ private boolean hasAugmentingPath(FlowNetwork G, int s, int t): kiểm tra xem có đường tăng luồng từ s đến t không.

+ private double excess(FlowNetwork G, int v): trả lại giá trị luồng phần dư tại đỉnh v.

\* Kết quả:



**Bài 16:**

\* Class Date bao gồm:

- Các thuộc tính:

+ private static final int[] DAYS = { 0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }: lưu số ngày trong một tháng của năm nhuận.

+ private final int month: lưu tháng

+ private final int day: lưu ngày

+ private final int year: lưu năm

- Các phương thức:

+ public Date(String date): hàm tạo constructor theo định dạng chuỗi.

+ public String[] convertInputVietNam(String a): trả về một mảng String giúp chuyển đổi từ định dạng chuỗi về dạng ngày tháng năm Việt Nam.

+ các hàm get và set lấy ra thuộc tính có trong lớp và hàm toString(): in ra Date.

+ private static boolean isValid(int d, int m, int y): kiểm tra ngày tháng năm có hợp lệ không.

+ private static boolean isLeapYear(int y): kiểm tra xem có phải năm nhuận không.

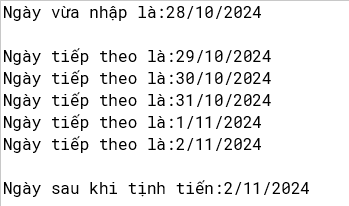
+ public Date next(): trả về ngày kế tiếp.

+ public boolean equals(Object other): kiểm tra xem 2 ngày có giống nhau không.

+ public boolean isAfter(Date that): trả về true nếu ngày này sau ngày đó, ngược lại trả về false.

+ public boolean isBefore(Date that): trả về true nếu ngày này trước ngày đó, ngược lại trả về false.

\* Kết quả:



**Bài 17:**

\* Class Inversions bao gồm:

- Các phương thức:

+ private static <Key extends Comparable<Key>> long merge(Key[] a, Key[] aux, int lo, int mid, int hi): giúp merge hai mảng lại với nhau đồng thời cập nhật lại số nghịch thế.

+ private static <Key extends Comparable<Key>> long count(Key[] a, Key[] b, Key[] aux, int lo, int hi): trả về số nghịch thế trong mảng con b đồng thời các phần tử trong mảng b cũng được sắp xếp lại.

+ public static <Key extends Comparable<Key>> long count(Key[] a): trả về số lần nghịch thế trong mảng.

+ private static <Key extends Comparable<Key>> boolean less(Key v, Key w): giúp so sánh hai phần tử với nhau.

\* Kết quả:



**Bài 18:**

\* Class ComplexExpression bao gồm:

- Các thuộc tính:

+ private String[] tokenArray: dùng để lưu các phần tử sau khi tách ra bởi dấu cách.

- Các phương thức:

+ public ComplexExpression(String complexExpression): constructor

+ public static boolean isOperator(String s): kiểm tra s có phải là một toán tử hay không.

+ public static Expression getOperator(String s, Expression left, Expression right): giúp tính toán biểu thức expression theo cộng, trừ, nhân, chia.

+ public double interpret(): giúp tính toán biểu thức hậu tố đồng thời trả về kết quả là một số thực.

\* Kết quả:



**Bài 19:**

\* Class FarthestPair bao gồm:

- Các thuộc tính:

+ private Point2D best1, best2: hai điểm xa nhau nhất.

+ private double maxDistance: khoảng cách xa nhất.

- Các phương thức:

+ public FarthestPair(Point2D[] points): tìm ra hai điểm xa nhau nhất.

+ private double farthest(Point2D[] pointsByX, Point2D[] pointsByY, Point2D[] aux, , int lo, int hi): tìm cặp điểm xa nhất trong pointByX[lo..hi].

+ public double distance(): lấy ra khoảng cách xa nhất giữa hai điểm.

+ private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)

\* Kết quả:



**Bài 20:**

\* Class Job bao gồm:

- Các thuộc tính:

+ int start: thời gian bắt đầu của một công việc.

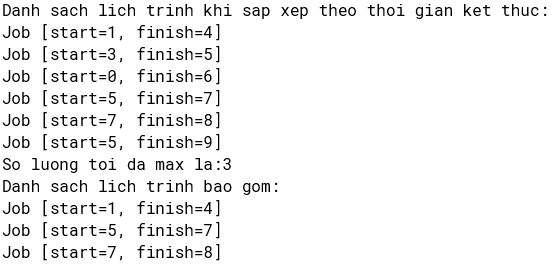
+ int finish: thời gian kết thúc của một công việc.

- Các phương thức:

+ public int compareTo (Job y): phương thức giúp so sánh hai công việc với nhau theo thời gian kết thúc.

+ các hàm get và set lấy ra thuộc tính có trong lớp và hàm toString(): in ra một công việc

\* Kết quả:



**Bài 21:**

\* Class Job bao gồm:

- Các thuộc tính:

+ int start: thời gian bắt đầu của một công việc.

+ int finish: thời gian kết thúc của một công việc.

+ int profit: lợi nhuận của một công việc.

- Các phương thức:

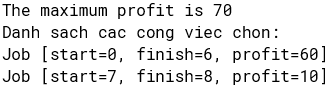
+ public int compareTo (Job y): phương thức giúp so sánh hai công việc với nhau theo thời gian kết thúc.

+ các hàm get và set lấy ra thuộc tính có trong lớp và hàm toString(): in ra một công việc

+ public static int findLastNonConflictingJob(List<Job> jobs, int n): tìm ra vị trí cuối cùng của công việc không bị xung đột với công việc ở vị trí n.

+ public static int maxProfit(List<Job> jobs,ArrayList<Job>[] ans): tìm ra lợi nhuận lớn nhất và cho biết nên làm những công việc nào để đạt được lợi nhuận lớn nhất.

\* Kết quả:



**Bài 22:**

\* Class FileSearch bao gồm:

- Các thuộc tính:

+ private static FileFrequencyIndex ffi: dùng để lưu các từ có ở trong file nào và file đấy chứa bao nhiêu từ đó.

+ private static ST<File,Integer> st: truy xuất file đó có tổng số từ cần tìm là bao nhiêu.

+ private static boolean searchSuccess = true: giúp kiếm tra xem chúng ta có tìm kiếm ra được kết quả không. Giá trị mặc định sẽ là true.

- Các phương thức:

+ public void query(String[] words): giúp truy vấn chuỗi cần tìm

+ public static LinkedHashMap<File,Integer> sortByValue(ST<File, Integer> st): giúp sắp xếp kết quả các file thỏa mãn theo tổng số lượng các từ trong chuỗi cần tìm.

+ public static void inKetQua(): in ra kết quả các file cùng với đó là tổng số lượng từ trong file đó. Nếu không có file nào thỏa mãn in ra không tìm thấy.

\* Kết quả:

