САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет: Программной инженерии и компьютерной техники
Направление подготовки: 09.03.04 (Программная инженерия)

Дисциплина «Алгоритмы и структуры данных»

Отчёт по лабораторной работе Неделя №5

OpenEdu

Группа: P3217
Выполнил: Минин Александр

г. Санкт-Петербург
2018г.

Задача №1

Условие

Структуру данных «куча», или, более конкретно, «неубывающая пирамида», можно реализовать на основе массива.

Для этого должно выполнятся основное свойство неубывающей пирамиды, которое заключается в том, что для каждого $1 \leq i \leq n$ выполняются условия:

- если $2i \leq n$, то $a[i] \leq a[2i]$;
- если $2i + 1 \leq n$, то $a[i] \leq a[2i + 1]$.

Дан массив целых чисел. Определите, является ли он неубывающей пирамидой.

Формат входного файла

Первая строка входного файла содержит целое число n ($1 \leq n \leq 10^6$). Вторая строка содержит n целых чисел, по модулю не превосходящих $2 \cdot 10^9$.

Формат выходного файла

Выведите «YES», если массив является неубывающей пирамидой, и «NO» в противном случае.

Решение

```
package week5;

import mooc.EdxIO;

public class Week5_1 {
    private static EdxIO edxIO;

    public static void main(String[] args) {
        edxIO = EdxIO.create();

        int n = edxIO.nextInt();
        int[] a = new int[n];
        a[0] = edxIO.nextInt();
        for (int i = 1; i < n; i++) {
            a[i] = edxIO.nextInt();
            if (i % 2 == 0 && a[i] < a[i / 2 - 1] || i % 2 == 1 && a[i] < a[i / 2]) {
                edxIO.println("NO");
                edxIO.close();
                return;
            }
        }

        edxIO.println("YES");
        edxIO.close();
    }
}
```

Результаты

№ теста	Результат	Время, мс	Память	Размер входного файла	Размер выходного файла
Max		281	43122688	10945420	5
1	OK	156	25563136	14	4
2	OK	109	25595904	14	5
3	OK	125	25579520	1092	5
4	OK	125	25583616	889	5
5	OK	140	25587712	1099	4
6	OK	140	25559040	1100	5
7	OK	125	25600000	1098	5
8	OK	109	25600000	1093	5
9	OK	125	25616384	1105	4
10	OK	109	25550848	1095	4
11	OK	125	25784320	10931	5
12	OK	125	25837568	8837	5
13	OK	109	25554944	10928	4
14	OK	125	25800704	10934	5
15	OK	140	25763840	10989	5
16	OK	109	25808896	10934	5
17	OK	125	25726976	10978	4
18	OK	109	25563136	10960	4
19	OK	156	26865664	109474	5
20	OK	125	26800128	89095	5
21	OK	156	25587712	109362	4
22	OK	109	26968064	109479	5
23	OK	140	26910720	109486	5
24	OK	125	26660864	109443	4
25	OK	109	25866240	109565	4
26	OK	156	25632768	109493	4

27	OK	171	28487680	1094387	5
28	OK	171	28319744	886879	5
29	OK	125	25980928	1094726	4
30	OK	156	28561408	1094117	5
31	OK	187	28540928	1094308	5
32	OK	171	28545024	1094215	5
33	OK	156	28233728	1094084	4
34	OK	109	26038272	1094403	4
35	OK	281	42151936	10944156	5
36	OK	281	40914944	8876466	5
37	OK	125	29618176	10945179	4
38	OK	250	42143744	10945420	5
39	OK	234	42676224	10943533	5
40	OK	265	43122688	10944594	5
41	OK	234	38334464	10944330	4
42	OK	140	29777920	10944738	4

Задача №2

Условие

Реализуйте очередь с приоритетами. Ваша очередь должна поддерживать следующие операции: добавить элемент, извлечь минимальный элемент, уменьшить элемент, добавленный во время одной из операций.

Формат входного файла

В первой строке входного файла содержится число n ($1 \le n \le 10^6$) - число операций с очередью.

Следующие n строк содержат описание операций с очередью, по одному описанию в строке. Операции могут быть следующими:

- `A` x — требуется добавить элемент x в очередь.
- `X` — требуется удалить из очереди минимальный элемент и вывести его в выходной файл. Если очередь пуста, в выходной файл требуется вывести звездочку «*».
- `D` x y — требуется заменить значение элемента, добавленного в очередь операцией `A` в строке входного файла номер $x + 1$, на y. Гарантируется, что в строке $x + 1$ действительно находится операция `A`, что этот элемент не был ранее удален операцией `X`, и что y меньше, чем предыдущее значение этого элемента.

В очередь помещаются и извлекаются только целые числа, не превышающие по модулю 10^9.

Формат выходного файла

Выведите последовательно результат выполнения всех операций `X`, по одному в каждой строке выходного файла. Если перед очередной операцией `X` очередь пуста, выведите вместо числа звездочку «*».

Решение

```
package week5;

import mooc.EdxIO;

import java.math.BigDecimal;
import java.math.RoundingMode;

public class Week5_2 {
    private static EdxIO edxIO;

    public static void main(String[] args) {
        edxIO = EdxIO.create();

        int n = edxIO.nextInt();
        Heap<HeapNode<Integer>> heap = new Heap(n);
        HeapNode[] heapNodeMapping = new HeapNode[n];

        for (int i = 0; i < n; i++) {
            switch (edxIO.nextChar()) {
                case 'A':
                    int el = edxIO.nextInt();
                    HeapNode newEl = new HeapNode<>(el, el);
                    heap.insert(newEl);
                    heapNodeMapping[i] = newEl;
                    break;
                case 'X':
                    if (heap.heapSize == 0) {
```

```
                edxIO.println('*');
            } else {
                int min = heap.getMin().getKey();
                heap.removeMin();
                edxIO.println(min);
            }
            break;
        case 'D':
            HeapNode heapNode = heapNodeMapping[edxIO.nextInt() - 1];
            int replaceEl = edxIO.nextInt();
            heapNode.setKey(replaceEl);
            heap.siftUp(heapNode.getPos());
            break;
        default:
            throw new IllegalStateException();
      }
   }

   edxIO.close();
}

private interface Keyable {

   public int getKey();

   public void setKey(int key);

   public int getPos();

   public void setPos(int pos);

}

public static class HeapNode<T> implements Keyable {
   private T obj;
   private int key;
   private int pos;

   HeapNode(T obj, int key) {
      this.obj = obj;
      this.key = key;
   }

   private T getObject() {
      return obj;
   }

   public int getKey() {
      return key;
   }

   public void setKey(int key) {
      this.key = key;
   }

   public int getPos() {
      return pos;
   }

   public void setPos(int pos) {
```

```
        this.pos = pos;
    }

    public void setObj(T obj) {
        this.obj = obj;
    }

}

public static class Heap<T extends Keyable> {
    private Object[] h;
    private int heapSize = 0;

    public Heap(int capacity) {
        h = new Object[capacity];
    }

    /**
     * Constructs a HEAP from the array O(N)
     *
     * @param array
     */
    public Heap(T[] array) {
        h = new Object[array.length];
        heapSize = array.length;
        for (int i = 0; i < array.length; i++) {
            h[i] = array[i];
        }
        for (int i = heapSize / 2; i >= 0; i--) {
            siftDown(i);
        }
    }

    public void insert(T t) {
        t.setPos(heapSize);
        h[heapSize] = t;
        heapSize++;
        siftUp(heapSize - 1);
    }

    public T getMin() {
        return get(0);
    }

    public void removeMin() {
        swap(0, --heapSize);
        siftDown(0);
    }

    private void siftUp(int i) {
        while (i > 0 && get(i).getKey() < get((i - 1) / 2).getKey()) {
            swap(i, (i - 1) / 2);
            i = (i - 1) / 2;
        }
    }

    private void siftDown(int i) {
        while (2 * i + 1 < heapSize) {
            int j = 2 * i + 1;
            if (j + 1 < heapSize && get(j).getKey() > get(2 * i + 2).getKey()) {
```

```
                j += 1;
            }
            if (get(j).getKey() < get(i).getKey()) {
                swap(i, j);
                i = j;
            } else {
                break;
            }
        }
    }

    private T get(int i) {
        return (T) h[i];
    }


    private void swap(int i, int j) {
        get(i).setPos(j);
        get(j).setPos(i);

        T tmp = get(i);
        h[i] = get(j);
        h[j] = tmp;
    }

    public void changeKey(int i, int newKey) {
        get(i).setKey(newKey);
        siftDown(i);
        siftUp(i);
    }

    public void print() {
        BigDecimal dec = new BigDecimal(Math.log(heapSize) / Math.log(2));
        dec = dec.setScale(1, RoundingMode.UP);
        int levels = dec.intValue();
        int lastLevel = (int) Math.pow(2, levels - 1);
        System.out.println("*** HEAP *** SIZE(" + heapSize + ")");
        for (int i = 0, k = 0, lev = 0; i <= (heapSize - 1); i++) {
            if (lev == 4) {
                System.out.println(" ");
            } else {
                for (int l = 0; l < (2 * lastLevel - 1) / Math.pow(2, lev); l++) {
                    System.out.print(" ");
                }
            }
            System.out.print(get(i).getKey());
            if (i == k) {
                k = 2 * k + 2;
                lev += 1;
                System.out.println();
            }
        }
    }
  }

}
```

Результаты

№ теста	Результат	Время, мс	Память	Размер входного файла	Размер выходного файла
Max		953	98467840	12083657	5694235
1	OK	125	21651456	37	12
2	OK	140	21590016	6	3
3	OK	109	21643264	11	3
4	OK	109	21618688	22	4
5	OK	109	21630976	19	6
6	OK	140	21635072	19	6
7	OK	125	21643264	19	6
8	OK	140	21676032	48	19
9	OK	125	21618688	58	29
10	OK	140	21639168	57	28
11	OK	109	21651456	48	19
12	OK	109	21635072	58	29
13	OK	125	21614592	57	28
14	OK	109	21643264	828	573
15	OK	125	21667840	1037	369
16	OK	171	21684224	828	573
17	OK	125	21659648	988	404
18	OK	109	21676032	1082	300
19	OK	125	21622784	1139	240
20	OK	140	21688320	930	377

21	OK	109	21618688	1190	280
22	OK	125	21856256	8184	5678
23	OK	156	21860352	10768	3637
24	OK	109	22106112	8206	5700
25	OK	109	22073344	9903	3928
26	OK	140	22159360	10814	3000
27	OK	125	22163456	11338	2400
28	OK	125	21942272	11138	3582
29	OK	156	22016000	10904	3851
30	OK	171	23867392	81951	56944
31	OK	171	24018944	110901	36274
32	OK	171	23744512	81971	56964
33	OK	171	23834624	99351	39719
34	OK	187	23920640	107882	30000
35	OK	156	23900160	113181	24000
36	OK	156	23437312	112799	37474
37	OK	171	23523328	114106	37576
38	OK	281	28553216	819273	569265
39	OK	250	27336704	1143615	361526
40	OK	281	28323840	819455	569447
41	OK	296	28467200	992441	396009
42	OK	265	28880896	1079125	300000
43	OK	234	29147136	1131016	240000
44	OK	250	27725824	1175194	377350
45	OK	281	28262400	1174192	378071
46	OK	953	61022208	8194244	5694235
47	OK	859	59600896	11753433	3632457
48	OK	656	61042688	8193883	5693874
49	OK	859	97824768	9926125	3963652

50	OK	875	97632256	10792079	3000000
51	OK	921	98467840	11312176	2400000
52	OK	578	60407808	12078250	3794039
53	OK	546	60022784	12083657	3795822