Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики
Мегафакультет компьютерных технологий и управления

Дисциплина
«Алгоритмы и структуры данных»

Лабораторная работа №5

Выполнил:
Студент группы P3218
Рябов Сергей Витальевич
Преподаватель:
Муромцев Дмитрий Ильич

Санкт-Петербург,
2018

1. Куча ли?

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 $1 \le i \le n$ выполняются условия:

если $2i \le n$, то $a[i] \le a[2i]$;

если $2i+1 \le n$, то $a[i] \le a[2i+1]$.

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

Формат входного файла

Первая строка входного файла содержит целое число n ($1 \le n \le 10^6$). 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.

Исходный код (C#):

```
using System;
using System.IO;

namespace ADS.Week5
{
    public class Task1
    {
        public static void Main(string[] args)
        {
            int n;
            int[] array;
            using (StreamReader streamReader = new StreamReader("input.txt"))
            {
                n = int.Parse(streamReader.ReadLine());
                array = new int[n];
                string[] str = streamReader.ReadLine().Split(' ');
                for (int i = 0; i < n; i++)
                {
                    array[i] = int.Parse(str[i]);
                }
            }
            bool result = true;
            for (int i = n - 1; i > 0; i--)
            {
                if (array[(i + 1) / 2 - 1] > array[i])
                {
                    result = false;
                    break;
                }
            }
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                streamWriter.WriteLine(result ? "YES" : "NO");
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.421	105127936	10945420	5
1	OK	0.015	10035200	14	4
2	OK	0.062	10088448	14	5
3	OK	0.031	10076160	1092	5
4	OK	0.031	10051584	889	5
5	OK	0.015	10125312	1099	4
6	OK	0.015	10076160	1100	5
7	OK	0.015	10117120	1098	5
8	OK	0.015	10125312	1093	5
9	OK	0.015	10055680	1105	4
10	OK	0.015	10063872	1095	4
11	OK	0.015	10317824	10931	5
12	OK	0.015	10289152	8837	5
13	OK	0.015	10240000	10928	4
14	OK	0.031	10199040	10934	5
15	OK	0.015	10190848	10989	5
16	OK	0.031	10186752	10934	5
17	OK	0.031	10260480	10978	4
18	OK	0.031	10211328	10960	4
19	OK	0.015	11153408	109474	5
20	OK	0.031	11112448	89095	5

2. Очередь с приоритетами

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 «*».

Исходный код (C#):

```
using System;
using System.IO;
using System.Collections.Generic;

namespace ADS.Week5
{
    public class PriorityQueue
    {
        public class Element
        {
            public int Value { get; set; }
            public int Index { get; set; }
            public Element(int value, int index)
            {
                Value = value;
                Index = index;
            }
        }

        public int Size { get => elements.Count; }
        private List<Element> elements;
        private Dictionary<int, Element> operations;

        public PriorityQueue(int baseSize = 1000000)
        {
            elements = new List<Element>(baseSize);
            operations = new Dictionary<int, Element>();
        }

        public void Enqueue(int operationNumber, int value)
        {
            Element element = new Element(value, elements.Count);
            elements.Add(element);
            operations.Add(operationNumber, element);
            LiftElement(Size - 1);
        }
        public int Dequeue()
        {
            int value = elements[0].Value;
            Swap(0, Size - 1);
            elements.RemoveAt(Size - 1);
            Heapify(0);
            return value;
        }
        public void DecreaseValue(int operationNumber, int value)
        {
            Element element = operations[operationNumber];
            element.Value = value;
            LiftElement(element.Index);
        }
        private void LiftElement(int index)
        {
            int parentIndex = (index + 1) / 2 - 1;
            while (index > 0 && elements[parentIndex].Value > elements[index].Value)
            {
```

```
            Swap(index, parentIndex);
            index = parentIndex;
            parentIndex = (index + 1) / 2 - 1;
        }
    }

    private void Heapify(int index)
    {
        int leftIndex = (index + 1) * 2 - 1;
        int rightIndex = leftIndex + 1;
        int minIndex;
        if (leftIndex < elements.Count && elements[leftIndex].Value < elements[index].Value)
            minIndex = leftIndex;
        else
            minIndex = index;
        if (rightIndex < elements.Count && elements[rightIndex].Value < elements[minIndex].Value)
            minIndex = rightIndex;
        if (index != minIndex)
        {
            Swap(index, minIndex);
            Heapify(minIndex);
        }
    }

    private void Swap(int index1, int index2)
    {
        elements[index1].Index = index2;
        elements[index2].Index = index1;

        Element buf = elements[index1];
        elements[index1] = elements[index2];
        elements[index2] = buf;
    }
}
public class Task2
{
    public static void Main(string[] args)
    {
        using (StreamReader streamReader = new StreamReader("input.txt"))
        using (StreamWriter streamWriter = new StreamWriter("output.txt"))
        {
            int n = int.Parse(streamReader.ReadLine());
            PriorityQueue queue = new PriorityQueue(n);
            for (int i = 0; i < n; i++)
            {
                string[] cmd = streamReader.ReadLine().Split(' ');

                switch(cmd[0][0])
                {
                    case 'A':
                        queue.Enqueue(i + 1, int.Parse(cmd[1]));
                        break;
                    case 'X':
                        if (queue.Size > 0)
                            streamWriter.WriteLine(queue.Dequeue());
                        else
                            streamWriter.WriteLine('*');
                        break;
                    case 'D':
                        queue.DecreaseValue(int.Parse(cmd[1]), int.Parse(cmd[2]));
                        break;
                }
            }
        }
```

```
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.765	110915584	12083657	5694235
1	OK	0.031	10477568	37	12
2	OK	0.015	10498048	6	3
3	OK	0.031	10510336	11	3
4	OK	0.031	10563584	22	4
5	OK	0.031	10473472	19	6
6	OK	0.031	10551296	19	6
7	OK	0.078	10498048	19	6
8	OK	0.031	10526720	48	19
9	OK	0.031	10567680	58	29
10	OK	0.015	10518528	57	28
11	OK	0.031	10543104	48	19
12	OK	0.031	10534912	58	29
13	OK	0.031	10547200	57	28
14	OK	0.031	10579968	828	573
15	OK	0.031	10571776	1037	369
16	OK	0.031	10502144	828	573
17	OK	0.015	10600448	988	404
18	OK	0.031	10551296	1082	300
19	OK	0.015	10543104	1139	240
20	OK	0.031	10539008	930	377