ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА № 5

ПО ДИСЦИПЛИНЕ «Алгоритмы и структуры данных»

Выполнил: Мещеряков Владимир Алевтинович

Группа: P3218

Преподаватель: Романов Алексей Андреевич

Волчек Дмитрий Геннадьевич

Санкт-Петербург

2019 г.

Week 5

1)Куча ли?

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

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 ($1 \le n \le 106$). Вторая строка содержит n целых чисел, по модулю не превосходящих $2 \cdot 109$.

Формат выходного файла

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

Примеры

input.txt	output.txt
5 1 0 1 2 0	NO
5 1 3 2 5 4	YES

```
#include "edx-io.hpp"
using namespace std;

int main() {

	long N;
	io >> N;
	long* array = new long[N];


	for (long i = 0; i < N; i++) {
		io >> array[i];
	}
	bool isHeap = true;
	//На каждом узле проверяем что его дети больше него
	//Если хоть в 1 false - то это не куча
	for (long i = 1; i < N / 2 + N % 2; i++) {
		if (array[i - 1] > array[2 * i - 1] || array[i - 1] > array[2 * i]) {
			isHeap = false;
			break;
		}
	}
	if (isHeap) {
		io << "YES";
	}
	else {
		io << "NO";
	}
	return 0;
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.062	16760832	10945420	3
1	OK	0.000	2445312	14	2
2	OK	0.015	2445312	14	3
3	OK	0.000	2445312	1092	3
4	OK	0.015	2453504	889	3
5	OK	0.000	2445312	1099	2
6	OK	0.000	2441216	1100	3
7	OK	0.000	2445312	1098	3
8	OK	0.015	2445312	1093	3
9	OK	0.000	2445312	1105	2
10	OK	0.015	2449408	1095	2
11	OK	0.000	2449408	10931	3
12	OK	0.000	2232320	8837	3
13	OK	0.015	2244608	10928	2
14	OK	0.000	2244608	10934	3
15	OK	0.000	2232320	10989	3
16	OK	0.015	2232320	10934	3
17	OK	0.015	2232320	10978	2
18	OK	0.000	2232320	10960	2
19	OK	0.000	2265088	109474	3
20	OK	0.015	2265088	89095	3
21	OK	0.000	2265088	109362	2
22	OK	0.015	2265088	109479	3
23	OK	0.000	2265088	109486	3
24	OK	0.000	2260992	109443	2
25	OK	0.015	2265088	109565	2
26	OK	0.000	2260992	109493	2
27	OK	0.015	3309568	1094387	3
28	OK	0.000	3096576	886879	3
29	OK	0.015	3309568	1094726	2
30	OK	0.015	3309568	1094117	3
31	OK	0.000	3309568	1094308	3
32	OK	0.015	3309568	1094215	3
33	OK	0.000	3309568	1094084	2
34	OK	0.000	3309568	1094403	2
35	OK	0.046	16756736	10944156	3
36	OK	0.046	14692352	8876466	3
37	OK	0.062	16760832	10945179	2
38	OK	0.046	16756736	10945420	3
39	OK	0.046	16752640	10943533	3
40	OK	0.062	16760832	10944594	3
41	OK	0.046	16756736	10944330	2
42	OK	0.046	16760832	10944738	2

2)Очередь с приоритетами

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

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 «$*$».

Пример

input.txt	output.txt

8 A 3 A 4 A 2 X D 2 1 X X X	2 1 3 *

```
#include "edx-io.hpp"
using namespace std;

//Структура хранит в себе введённое значение(приоритет) и номер строки, в которой оно было введено
struct number {
	long value;
	long input_time;
};

void swop(number* a, number* b) {
	number temp = *a;
	*a = *b;
	*b = temp;
}

//Обход кучи снизу с i-того элемента, нахождение его места в куче
void heapifyUp(number* array, long i, long* array_of_positions) {
	//Позиция родительского для i-того элемента в куче
	long parent = i / 2 - (1 - i % 2);
	//Пока не добрались до корня и пока родитель больше
	while (i != 0 && array[i].value < array[parent].value) {
		//Меняем местами ребёнка и родителя, сохраняя их позиции в массиве позиций
		array_of_positions[array[i].input_time] = parent;
		array_of_positions[array[parent].input_time] = i;
		swop(&array[i], &array[parent]);
		//Переопределяем местоположение элемента и его родителя
		i = parent;
		parent = i / 2 - (1 - i % 2);
	}
}

//Восстановление кучи сверху
void heapifyDown(number* array, long tail, long* array_of_positions) {
	//Позиция текущего элемента
	long position = 0;
	//Позиции его детей
	long left = 2 * (position + 1) - 1;
	long right = 2 * (position + 1);
	//Пока не дошли до конца очереди, проверяем, что хотя бы один ребёнок меньше родителя и при этом находится в очереди
	while (position != tail && ((array[position].value > array[left].value && left <= tail) || (array[position].value > array[right].value && right <= tail))) {
		//Находим наименьшего, при условии, что правый находится в очереди
		if (array[right].value < array[left].value && right <= tail) {
			//Меняем местами ребёнка и родителя, сохраняя их позиции в массиве позиций
			array_of_positions[array[position].input_time] = right;
			array_of_positions[array[right].input_time] = position;
			swop(&array[right], &array[position]);
			//Переопределяем позицию сортируемого элемента
			position = right;
		}
		else {
			array_of_positions[array[position].input_time] = left;
```

```
                array_of_positions[array[left].input_time] = position;
                swop(&array[left], &array[position]);
                position = left;
            }
            //Переопределяем детей сортируемого элемента
            left = 2 * (position + 1) - 1;
            right = 2 * (position + 1);
        }
}


int main() {
        long N;
        io >> N;
        //Очередь с приоритетами
        number* Queue = new number[N];
        long head = 0;
        long tail = -1;

        //Массив, индекс которого - номер строки, в которой ввели число, а значение -
положение с очереди
        long* array = new long[N];

        char action;
        long a, temp;


        for (long i = 0; i < N; i++) {
            io >> action;
            switch (action)
            {
            case 'A':
                io >> a;
                //Добавляем в очередь структуру (значение, номер строки)
                Queue[++tail] = number{ a, i };
                //Записываем начальную позицию введённого числа
                array[i] = tail;
                //Располагаем наш элемент в куче по приоритету
                heapifyUp(Queue, tail, array);
                break;
            case 'X':
                if (head > tail) {
                    io << '*' << '\n';
                }
                else {
                    //Меняем местами первый и последний элемент, сохраняя при этом
целостность массива array
                    array[Queue[head].input_time] = tail;
                    array[Queue[tail].input_time] = head;
                    swop(&Queue[head], &Queue[tail]);

                    io << Queue[tail--].value << '\n';

                    //Восстанавливаем кучу
                    heapifyDown(Queue, tail, array);
                }
                break;
            case 'D':
                io >> a;
                io >> temp;
                //Позиция элемента введённого в a-той строке сохранена в массиве
array
                Queue[array[a - 1]].value = temp;
                //Располагаем измененный элемент в куче
                heapifyUp(Queue, array[a - 1], array);
                break;
            default:
                break;
            }

        }
```

```
	return 0;
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.390	22732800	12083657	5694235
1	OK	0.015	2457600	37	12
2	OK	0.031	2441216	6	3
3	OK	0.015	2445312	11	3
4	OK	0.015	2457600	22	4
5	OK	0.000	2441216	19	6
6	OK	0.000	2441216	19	6
7	OK	0.015	2449408	19	6
8	OK	0.000	2441216	48	19
9	OK	0.000	2445312	58	29
10	OK	0.000	2445312	57	28
11	OK	0.015	2445312	48	19
12	OK	0.000	2224128	58	29
13	OK	0.000	2224128	57	28
14	OK	0.000	2236416	828	573
15	OK	0.015	2224128	1037	369
16	OK	0.000	2240512	828	573
17	OK	0.015	2232320	988	404
18	OK	0.015	2224128	1082	300
19	OK	0.000	2224128	1139	240
20	OK	0.015	2224128	930	377
21	OK	0.000	2236416	1190	280
22	OK	0.000	2240512	8184	5678
23	OK	0.015	2240512	10768	3637
24	OK	0.015	2240512	8206	5700
25	OK	0.000	2252800	9903	3928
26	OK	0.000	2240512	10814	3000
27	OK	0.000	2240512	11338	2400
28	OK	0.015	2236416	11138	3582
29	OK	0.015	2240512	10904	3851
30	OK	0.000	2338816	81951	56944
31	OK	0.000	2310144	110901	36274
32	OK	0.000	2306048	81971	56964
33	OK	0.000	2318336	99351	39719
34	OK	0.000	2318336	107882	30000
35	OK	0.000	2342912	113181	24000