Санкт-Петербургский национальный исследовательский университет

Информационных технологий, механики и оптики

Отчет по решению задач пятой недели
По курсу «Алгоритмы и структуры данных»

на Openedu

Выполнил: Сыроватский Павел Валентинович

Группа P3218

Санкт-Петербург

2019

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 ($1 \le n \le 106$). 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 <fstream>
using namespace std;


#include <fstream>
#include <string>

using namespace std;

string check_is_heap(long* A, long n) {
	for (long i = n / 2; i > 0; i--)
	{
		if (A[i-1] > A[2 * i-1] )
			return "NO";
```

```
			if (2*i + 1 <= n && A[i-1] > A[2 * i ])
				return "NO";
		}
		return "YES";


}


int main() {
		ifstream input("input.txt");
		ofstream output("output.txt");
		long n;
		input >> n;
		long* A = new long[n];

		for (long i = 0; i < n; i++)
		{
			input >> A[i];
		}
		output << check_is_heap(A, n);



		return 0;
}
```

Результат выполнения первой задачи

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.078	6356992	10945420	3
1	OK	0.000	2355200	14	2
2	OK	0.015	2338816	14	3
3	OK	0.000	2342912	1092	3
4	OK	0.000	2338816	889	3
5	OK	0.015	2338816	1099	2
6	OK	0.015	2338816	1100	3
7	OK	0.000	2342912	1098	3
8	OK	0.000	2355200	1093	3
9	OK	0.015	2338816	1105	2
10	OK	0.000	2334720	1095	2
11	OK	0.000	2347008	10931	3
12	OK	0.000	2347008	8837	3
13	OK	0.000	2359296	10928	2
14	OK	0.000	2347008	10934	3
15	OK	0.000	2347008	10989	3
16	OK	0.015	2359296	10934	3
17	OK	0.000	2347008	10978	2
18	OK	0.000	2363392	10960	2
19	OK	0.031	2400256	109474	3
20	OK	0.015	2379776	89095	3
21	OK	0.000	2383872	109362	2
22	OK	0.015	2383872	109479	3

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 ($1 \le n \le 106$) - 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 109.

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 <fstream>
#include <string>

using namespace std;


struct element {
	long value;
	long position_string;
};

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

void find_place(element* array, long i, long* array_of_positions) {
	//Находим место для элемента, как было сказано в презентации
	long parent = i / 2 - (1 - i % 2);
	while (i != 0 && array[i].value < array[parent].value) {
		array_of_positions[array[i].position_string] = parent;
		array_of_positions[array[parent].position_string] = i;
		swop(&array[i], &array[parent]);
		i = parent;
		parent = i / 2 - (1 - i % 2);
	}
}

void recovery_heap(element* array, long tail, long* array_of_positions) {
	long position = 0;
	long left = 2 * (position + 1) - 1;
	long right = 2 * (position + 1);
	while (position != tail && ((array[position].value > array[left].value && left <=
tail) || (array[position].value > array[right].value && right <= tail))) {
		if (array[right].value < array[left].value && right <= tail) {
			array_of_positions[array[position].position_string] = right;
			array_of_positions[array[right].position_string] = position;
			swop(&array[right], &array[position]);
			position = right;
		}
		else {
			array_of_positions[array[position].position_string] = left;
```

```
                array_of_positions[array[left].position_string] = position;
                swop(&array[left], &array[position]);
                position = left;
            }
            left = 2 * (position + 1) - 1;
            right = 2 * (position + 1);
        }
}


int main() {
        ifstream input("input.txt");
        ofstream output("output.txt");
        long N;
        input >> N;
        element* Queue = new element[N];
        long top = 0;
        long tail = -1;

        long* tree = new long[N];

        char command;
        long a, temp;


        for (long i = 0; i < N; i++) {
            input >> command;
            switch (command)
            {
            case 'A':
                input >> a;
                Queue[++tail] = element{ a, i };
                tree[i] = tail;
                find_place(Queue, tail, tree);
                break;
            case 'X':
                if (top > tail) {
                    output << '*' << '\n';
                }
                else {
                    tree[Queue[top].position_string] = tail;
                    tree[Queue[tail].position_string] = top;
                    swop(&Queue[top], &Queue[tail]);

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

                    recovery_heap(Queue, tail, tree);
                }
                break;
            case 'D':
                input >> a;
                input >> temp;
                Queue[tree[a - 1]].value = temp;
                find_place(Queue, tree[a - 1], tree);
                break;
            default:
                break;
            }

        }

        return 0;
}
```

Результат решения задачи

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.578	12365824	12083657	5694235
1	OK	0.000	2588672	37	12
2	OK	0.015	2592768	6	3
3	OK	0.000	2600960	11	3
4	OK	0.000	2588672	22	4
5	OK	0.015	2588672	19	6
6	OK	0.000	2588672	19	6
7	OK	0.000	2588672	19	6
8	OK	0.000	2600960	48	19
9	OK	0.015	2584576	58	29
10	OK	0.000	2588672	57	28
11	OK	0.000	2588672	48	19
12	OK	0.015	2371584	58	29
13	OK	0.015	2371584	57	28
14	OK	0.031	2367488	828	573
15	OK	0.015	2371584	1037	369
16	OK	0.000	2367488	828	573
17	OK	0.000	2379776	988	404
18	OK	0.000	2371584	1082	300
19	OK	0.000	2383872	1139	240
20	OK	0.000	2371584	930	377
21	OK	0.000	2371584	1190	280
22	OK	0.015	2383872	8184	5678
23	OK	0.015	2400256	10768	3637
24	OK	0.000	2379776	8206	5700
25	OK	0.000	2383872	9903	3928
26	OK	0.015	2392064	10814	3000