Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Лабораторная работа №5
Дисциплина «Алгоритмы и структуры данных»

Выполнил:
Молодецкий Арсений Алексеевич
Группа P3217

Санкт-Петербург
2018

Задание №1:

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.

Код:

```
#include <iostream>
#include <string>
using namespace std;
#include "edx-io.hpp"


int main() {
   int n;
   io >> n;
   int* heap = new int[n+1];
   for (int i = 1; i <= n; ++i) {
      io >> heap[i];
   }

   bool isHeap = true;
   for (int i = n; isHeap && i > 1; i--) {
      if (heap[i / 2] > heap[i]){
         isHeap = false;
      }
   }
   io << (isHeap ? "YES" : "NO");
   return 0;
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.046	18333696	10945420	3
1	OK	0.046	3440640	14	2
2	OK	0.000	3428352	14	3
3	OK	0.031	3428352	1092	3
4	OK	0.000	3407872	889	3
5	OK	0.015	3420160	1099	2
6	OK	0.015	3428352	1100	3
7	OK	0.000	3416064	1098	3
8	OK	0.000	3432448	1093	3
9	OK	0.000	3424256	1105	2
10	OK	0.015	3424256	1095	2
11	OK	0.015	3448832	10931	3
12	OK	0.000	3432448	8837	3
13	OK	0.000	3416064	10928	2
14	OK	0.000	3411968	10934	3
15	OK	0.000	3416064	10989	3
16	OK	0.000	3416064	10934	3
17	OK	0.015	3436544	10978	2
18	OK	0.000	3411968	10960	2
19	OK	0.000	3534848	109474	3
20	OK	0.015	3489792	89095	3
21	OK	0.000	3543040	109362	2
22	OK	0.000	3526656	109479	3
23	OK	0.015	3534848	109486	3
24	OK	0.000	3506176	109443	2
25	OK	0.000	3522560	109565	2
26	OK	0.015	3510272	109493	2
27	OK	0.015	4841472	1094387	3
28	OK	0.000	4653056	886879	3
29	OK	0.015	4849664	1094726	2
30	OK	0.015	4874240	1094117	3
31	OK	0.000	4853760	1094308	3
32	OK	0.015	4874240	1094215	3
33	OK	0.000	4874240	1094084	2
34	OK	0.015	4849664	1094403	2
35	OK	0.046	18333696	10944156	3
36	OK	0.031	16240640	8876466	3
37	OK	0.031	18313216	10945179	2
38	OK	0.046	18321408	10945420	3
39	OK	0.046	18296832	10943533	3
40	OK	0.046	18333696	10944594	3
41	OK	0.031	18313216	10944330	2
42	OK	0.046	18317312	10944738	2

Задание №2:

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 «*».

Код:

```
#include <iostream>
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace tmp
{
   public class Program
   {
      private static StreamWriter _out;

      private static void Main(string[] args)
      {

         _out = new StreamWriter("output.txt");
         Console.SetOut(_out);
         new PriorityQueue().Run(File.ReadAllLines("input.txt"));
         DisposeIO();
      }

      private static void DisposeIO()
      {
         _out?.Dispose();
      }
   }
```

```
public struct HeapElem
{
   public long value;
   public int number;
}
public class PriorityQueue
{
   public Dictionary<int, int> _links = new Dictionary<int, int>();
   private HeapElem _tmp;
   private HeapElem[] _heap;
   private Dictionary<char, Action<string, int>> _instructions;
   private int _length;
   public void Run(string[] lines)
   {
      DefineInstructions();

      int n = int.Parse(lines[0]);
      _heap = new HeapElem[n + 1];

      _length = 0;

      for (int i = 1; i < lines.Length; i++)
      {
         _instructions[lines[i][0]](lines[i], i);
      }
   }

   private void DefineInstructions()
   {
      _instructions = new Dictionary<char, Action<string, int>>
      {
         ['X'] = (s, i) => { PrintAndRemoveMin(); },
         ['D'] = (s, i) =>
         {
            string[] tokens = s.Split(' ');
            int  number = int.Parse(tokens[1]);
            long  value = long.Parse(tokens[2]);
            ChangeElement(number, value);
         },
         ['A'] = (s, i) =>
         {
            string[] tokens = s.Split(' ');
            long value = long.Parse(tokens[1]);
            AddElement(value, i);
         }

      };
   }

   private void AddElement(long value, int number)
   {
      _length++;
      _heap[_length].value = value;
      _heap[_length].number = number;
      _links.Add(number, _length);
      DecreaseValue(_length, value);
   }

   private void DecreaseValue(int _i, long value)
```

```
{
    int i = _i;
    _heap[i].value = value;
    while (i > 1 && _heap[i / 2].value > _heap[i].value)
    {
        Swap(i, i / 2);
        i = i / 2;
    }
}

private void ChangeElement(int number, long value)
{
    DecreaseValue(_links[number], value);
}

private void PrintAndRemoveMin()
{
    if (_length <= 0)
    {
        Print('*');
    }
    else
    {
        Print(_heap[1].value);
        RemoveMin();
    }
}

private void RemoveMin()
{
    Swap(1,_length);
    _length--;
    if (_length > 0) Heapify(1);
}

private void Heapify(int index)
{
    int left = this.left(index);
    int right = this.right(index);
    int minIndex = index;

    if (left <= _length && _heap[left].value < _heap[index].value)
    {
        minIndex = left;
    }

    if (right <= _length && _heap[right].value < _heap[minIndex].value)
    {
        minIndex = right;
    }

    if (minIndex != index)
    {
        Swap(minIndex, index);
        Heapify(minIndex);
    }
}

private int left(int i)
{
```

```
            return i*2;
        }

        private int right(int i)
        {
            return i*2 + 1;
        }

        private void Swap(int i1, int i2)
        {
            _links[_heap[i1].number] = i2;
            _links[_heap[i2].number] = i1;
            _tmp.value = _heap[i1].value;
            _tmp.number = _heap[i1].number;
            _heap[i1].value = _heap[i2].value;
            _heap[i1].number = _heap[i2].number;
            _heap[i2].value = _tmp.value;
            _heap[i2].number = _tmp.number;
        }
        private void Print(long number)
        {
            Console.WriteLine(number);
        }
        private void Print(char c)
        {
            Console.WriteLine(c);
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.828	226910208	12083657	5694235
1	OK	0.015	10436608	37	12
2	OK	0.031	10305536	6	3
3	OK	0.031	10407936	11	3
4	OK	0.031	10399744	22	4
5	OK	0.031	10461184	19	6
6	OK	0.015	10448896	19	6
7	OK	0.031	10403840	19	6
8	OK	0.046	10510336	48	19
9	OK	0.031	10485760	58	29
10	OK	0.015	10481664	57	28
11	OK	0.031	10412032	48	19
12	OK	0.031	10469376	58	29
13	OK	0.031	10469376	57	28
14	OK	0.031	10493952	828	573
15	OK	0.015	10518528	1037	369
16	OK	0.046	10489856	828	573
17	OK	0.015	10477568	988	404
18	OK	0.031	10432512	1082	300
19	OK	0.031	10510336	1139	240
20	OK	0.015	10514432	930	377
21	OK	0.031	10514432	1190	280
22	OK	0.031	10694656	8184	5678
23	OK	0.031	10854400	10768	3637
24	OK	0.031	10690560	8206	5700
25	OK	0.031	10776576	9903	3928
26	OK	0.031	10768384	10814	3000
27	OK	0.015	10752000	11338	2400
28	OK	0.015	10809344	11138	3582
29	OK	0.031	10825728	10904	3851
30	OK	0.062	61267968	81951	56944
31	OK	0.062	61505536	110901	36274
32	OK	0.062	61235200	81971	56964
33	OK	0.078	61644800	99351	39719
34	OK	0.062	61837312	107882	30000
35	OK	0.062	61927424	113181	24000
36	OK	0.062	61489152	112799	37474
37	OK	0.078	61517824	114106	37576
38	OK	0.171	76824576	819273	569265
39	OK	0.171	75071488	1143615	361526
40	OK	0.171	76836864	819455	569447
41	OK	0.187	77144064	992441	396009
42	OK	0.187	77598720	1079125	300000
43	OK	0.218	80695296	1131016	240000
44	OK	0.171	75362304	1175194	377350
45	OK	0.171	75358208	1174192	378071
46	OK	1.687	166047744	8194244	5694235
47	OK	1.546	160239616	11753433	3632457
48	OK	1.421	166023168	8193883	5693874
49	OK	1.593	181227520	9926125	3963652
50	OK	1.812	223801344	10792079	3000000
51	OK	1.828	226910208	11312176	2400000
52	OK	1.078	160780288	12078250	3794039
53	OK	1.062	160833536	12083657	3795822