Алгоритмы и структуры данных

Задание к лабораторной работе №5. Деревья. Пирамида, пирамидальная сортировка. Очередь с приоритетами.

Лабораторная работа посвящена разбору следующих структур данных: деревья, пирамида или двоичная куча, очередь с приоритетами, а также еще одному виду сортировки за $n \log n$: пирамидальной (heapsort). Аналогично предыдущим лабораторным работам, есть два способа ее выполнения и защиты:

Читать внимательно: изменения!

1 **Базовый уровень.** Решается 2 задачи по вариантам. Варианты в табличке внизу, номер вашего варианта соответствует вашему номеру в списке группы. **Посмотреть свой номер нужно, например, в журнале успеваемости по дисциплине**. В этом случае максимум за защиту можно получить 4 балла. В сумме с самой работой (0,5 балла) и отчетом (1 балл) получается 5,5 балла, что достаточно для зачета.

2 **Продвинутый уровень.** Решаются все задачи или минимум 4 задачи, причем 2 из них - исходя из вашего варианта. В этом случае вы сможете получить максимальные 7,5 баллов.

Вариант	Номера задач	Вариант	Номера задач
1	1,2	16	4,5
2	1,3	17	4,6
3	1,4	18	4,7
4	1,5	19	5,6
5	1,6	20	5,7
6	1,7	21	6,7
7	2,3	22	1,7
8	2,4	23	2,7
9	2,5	24	6,7
10	2,6	25	3,5
11	2,7	26	5,6
12	3,4	27	1,5
13	3,5	28	1,3
14	3,6	29	4,5
15	3,7	30	4,6

Если одна задача вашего варианта вас по каким-то причинам не устраивает, вы можете выбрать вместо нее другую задачу.

1 задача. 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 $1 \leq i \leq n$ выполняются условия:

1. если $2i \leq n$, то $a_i \leq a_{2i}$,
2. 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 (input.txt).** 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 (output.txt).** Выведите «YES», если массив является неубывающей пирамидой, и «NO» в противном случае.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

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

2 задача. Высота дерева

В этой задаче ваша цель - привыкнуть к деревьям. Вам нужно будет прочитать описание дерева из входных данных, реализовать структуру данных, сохранить дерево и вычислить его высоту.

- Вам дается корневое дерево. Ваша задача - вычислить и вывести его высоту. Напомним, что высота (корневого) дерева - это максимальная глубина узла или максимальное расстояние от листа до корня. Вам дано произвольное дерево, не обязательно бинарное дерево.
- **Формат ввода или входного файла (input.txt).** Первая строка содержит число узлов n ($1 \leq n \leq 10^5$). Вторая строка содержит n целых чисел от -1 до $n-1$ – указание на родительский узел. Если i-ое значение равно -1, значит, что узел i - корневой, иначе это число является обозначением индекса родительского узла этого i-го узла ($0 \leq i \leq n-1$). **Индексы считать с 0.** Гарантируется, что дан только один корневой узел, и что входные данные представляют дерево.
- **Формат вывода или выходного файла (output.txt).** Выведите целое число – высоту данного дерева.
- Ограничение по времени. 3 сек.
- Ограничение по памяти. 512 мб.
- Пример 1:

input.txt	output.txt
5 4 -1 4 1 1	3

- **Объяснение примера**. Данный входной файл задает 5 узлов дерева с числами от 0 до 4. Узел под индексом 0 является дочерним узлом узла с индексом 4. Узел под индексом 1 - корневой узел. Узел с индексом 2 - тоже дочерний узел четвертого узла, а узлы с индексами 3 и 4 - дочерние узлы первого (корневого) узла. Можно записать данные узлы с соответствующми индексами, чтобы увидеть наглядно:

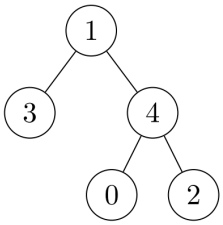
0	1	2	3	4
4	-1	4	1	1

  Давайте построим это дерево: Узел «1» – корневой, у него двое дочерних узла: «3» и «4», и у узла «4» – тоже два дочерних: «0» и «2».

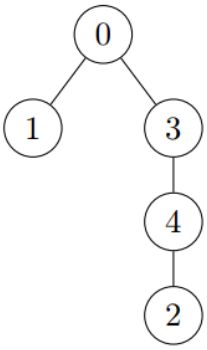
  Таким образом, высота этого дерева равна 3, т.к. количество вершин на пути от корня 1 к листу 2 равно 3.

- Пример 2:

input.txt	output.txt
5 -1 0 4 0 3	4



- **Объяснение примера**. Здесь также 5 узлов со значениями от 0 до 4, причем узел «0» – корневой, узел «1» – дочерний узла «0», узел «2» – дочерний от узла «4», узел «3» – также дочерний узла «0» (корневого), и узел «4» – дочерний узла «3». Высота дерева равно 4, т.к. количество узлов на пути от корневого узла к листу «2» равно 4.



- Что делать. Высоту ***бинарного*** дерева можно посчитать рекурсивно:

```
def height(root):
        if root is None:
                return 0

        return max(height(root.left), height(root.right)) + 1
```

Однако дерево в задаче не обязательно бинарное, поэтому вам нужно адаптировать подсчет высоты дерева для произвольного дерева. В этом случае дерево может быть очень глубоким, не допускайте переполения стека, если вы используете рекурсию, и тестируйте ваш алгоритм для максимальных данных и максимально возможной глубины.

Используйте тот факт, что значения каждого узла дерева - это его индекс, т.е. целочисленное значение от 0 до $n-1$, и вы можете хранить значения каждого узла в массиве, при этом доступ к любому узлу будет $O(1)$.

3 задача. Обработка сетевых пакетов

В этой задаче вы реализуете программу для моделирования обработки сетевых пакетов.

- Вам дается серия входящих сетевых пакетов, и ваша задача - смоделировать их обработку. Пакеты приходят в определенном порядке. Для каждого номера пакета i вы знаете время, когда пакет прибыл A_i и время, необходимое процессору для его обработки P_i (в миллисекундах). Есть только один процессор, и он обрабатывает входящие пакеты в порядке их поступления. Если процессор начал обрабатывать какой-либо пакет, он не прерывается и не останавливается, пока не завершит обработку этого пакета, а обработка пакета i занимает ровно P_i миллисекунд.

  Компьютер, обрабатывающий пакеты, имеет сетевой буфер фиксированного размера S. Когда пакеты приходят, они сохраняются в буфере перед обработкой. Однако, если буфер заполнен, когда приходит пакет (есть S пакетов, которые прибыли до этого пакета, и компьютер не завершил обработку ни одного из них), он отбрасывается и не обрабатывается вообще. Если несколько пакетов поступают одновременно, они сначала все сохраняются в буфере (из-за этого некоторые из них могут быть отброшены - те, которые описаны позже во входных данных). Компьютер обрабатывает пакеты в порядке их поступления и начинает обработку следующего доступного пакета из буфера, как только заканчивает обработку предыдущего. Если в какой-то момент компьютер не занят и в буфере нет пакетов, компьютер просто ожидает прибытия следующего пакета. Обратите внимание, что пакет покидает буфер и освобождает пространство в буфере, как только компьютер заканчивает его обработку.

- **Формат ввода или входного файла (input.txt).** Первая строка содержит размер S буфера ($1 \leq S \leq 10^5$) и количество n ($1 \leq n \leq 10^5$) входящих сетевых пакетов. Каждая из следующих n строк содержит два числа, i-ая строка содержит время прибытия пакета A_i ($0 \leq A_i \leq 10^6$) и время его обработки P_i ($0 \leq P_i \leq 10^3$) в миллисекундах. Гарантируется, что последовательность времени прибытия входящих пакетов – неубывающая, однако, она может содержать одинаковые значения времени прибытия нескольких пакетов, в этом случае рассматривается пакет, записанный в входном файле раньше остальных, как прибывший ранее. ($A_i \leq A_{i+1}$ для $1 \leq i \leq n - 1$.)

- **Формат вывода или выходного файла (output.txt).** Для каждого пакета напечатайте время (в миллисекундах), когда процессор начал его обрабатывать; или -1, если пакет был отброшен. Вывести ответ нужно в том же порядке, как как пакеты были описаны во входном файле.

- Ограничение по времени. 10 сек.

- Ограничение по памяти. 512 мб.

- Пример 1:

input.txt	output.txt
1 0	

Если нет пакетов, ничего выводить не нужно.

- Пример 2:

input.txt	output.txt
1 1 0 0	0

Единственный пакет поступил в момент времени 0, и компьютер обработал его сразу.

- Пример 3:

input.txt	output.txt
1 2 0 1 0 1	0 -1

Первый пакет поступил в момент времени 0, второй пакет также, но был отброшен, так как сетевой буффер имеет размер 1 и он был полон (т.к. место занят первый пакет). Первый пакет начал обрабатываться в момент времени 0, а второй не обрабатывался.

- Пример 4:

input.txt	output.txt
1 2 0 1 1 1	0 1

Первый пакет поступил в момент времени 0, компьютер сразу начал его обрабатывать и закончил в момент времени 1. Второй пакет поступил во время 1, и компьютер так же начал его обрабатывать.

- Что делать. Для решения этой задачи вы можете использовать массив или очередь (точнее, дек, чтобы у вас был доступ к последнему элементу).

  Одно из возможных решений - сохранить в списке или очереди время, когда компьютер завершит обработку пакетов (`finish_time`), которые в настоящее время хранятся в сетевом буфере, в порядке возрастания. Когда прибывает новый пакет, вам сначала нужно удалить в начале `finish_time` все пакеты, которые уже обработаны к моменту прибытия нового пакета. Затем вы добавляете время окончания для нового пакета в `finish_time`. Если буфер заполнен (в *очереди* `finish_time` уже S элементов), пакет отбрасывается. В противном случае время завершения его обработки добавляется к `finish_time`.

  Если при поступлении нового пакета очередь `finish_time` пуста, компьютер начнет обработку нового пакета немедленно, как только он поступит.

В противном случае компьютер начнет обработку нового пакета, как только он закончит обработку последнего из пакетов, находящихся в настоящее время в `finish_time` (здесь вам нужно получить доступ к последнему элементу `finish_time`, чтобы определить, когда компьютер начнет обрабатывать новый пакет). Вам также нужно будет вычислить время окончания обработки, добавив P_i к времени начала обработки и поместив его в конец `finish_time`.

- Еще примеры.

№	input.txt	output.txt
5	1 1 0 1	0
6	1 1 1 0	1
7	1 2 0 0 0 0	0 0
8	1 2 0 0 0 1	0 0
9	1 2 0 1 0 0	0 -1

№	input.txt	output.txt
10	1 2 0 1 0 1	0 -1
11	1 2 0 1 1 1	0 1
12	1 2 0 1 2 1	0 2
13	2 3 0 1 3 1 10 1	0 3 10

№	input.txt	output.txt
14	3 6 0 2 1 2 2 2 3 2 4 2 5 2	0 2 4 6 8 -1

4 задача. Построение пирамиды

В этой задаче вы преобразуете массив целых чисел в пирамиду. Это важнейший шаг алгоритма сортировки под названием `HeapSort`. Гарантированное время работы в худшем случае составляет $O(n \log n)$, в отличие от ***среднего*** времени работы `QuickSort`, равного $O(n \log n)$. `QuickSort` обычно используется на практике, потому что обычно он быстрее, но HeapSort используется для внешней сортировки, когда вам нужно отсортировать огромные файлы, которые не помещаются в памяти вашего компьютера.

Первым шагом алгоритма `HeapSort` является создание пирамиды (heap) из массива, который вы хотите отсортировать.

Ваша задача - реализовать этот первый шаг и преобразовать заданный массив целых чисел в пирамиду. Вы сделаете это, применив к массиву определенное количество перестановок (swaps). Перестановка - это операция, как вы помните, при которой элементы a_i и a_j массива меняются местами для некоторых i и j. Вам нужно будет преобразовать массив в пирамиду, используя только $O(n)$ перестановок. Обратите внимание, что в этой задаче вам нужно будет использовать `min-heap` вместо `max-heap`.

- **Формат ввода или входного файла (input.txt).** Первая строка содержит целое число n ($1 \leq n \leq 10^5$), вторая содержит n целых чисел a_i входного массива, разделенных пробелом ($0 \leq a_i \leq 10^9$, все a_i - различны.)

- **Формат выходного файла (output.txt).** Первая строка ответа должна содержать целое число m - количество сделанных свопов. Число m должно удовлетворять условию $0 \leq m \leq 4n$. Следующие m строк должны содержать по 2 числа: индексы i и j сделанной перестановки двух элементов, **индексы считаются с 0**. После всех перестановок в нужном порядке массив должен стать пирамидой, то есть для каждого i при $0 \leq i \leq n-1$ должны выполняться условия:

  1. если $2i + 1 \leq n - 1$, то $a_i < a_{2i+1}$,
  2. если $2i + 2 \leq n - 1$, то $a_i < a_{2i+2}$.

  Обратите внимание, что все элементы входного массива различны. Любая последовательность свопов, которая менее $4n$ и после которой входной массив становится корректной пирамидой, считается верной.

- Ограничение по времени. 3 сек.

- Ограничение по памяти. 512 мб.

- Пример 1:

input.txt	output.txt
5 5 4 3 2 1	3 1 4 0 1 1 3

  После перестановки элементов в позициях 1 и 4 массив становится следующим: 5 1 3 2 4.

  Далее, перестановка элементов с индексами 0 и 1: 1 5 3 2 4. И напоследок, переставим 1 и 3: 1 2 3 5 4, и теперь это корректная неубывающая пирамида.

- Пример 2:

input.txt	output.txt
5 1 2 3 4 5	0

5 задача. Планировщик заданий

В этой задаче вы создадите программу, которая параллельно обрабатывает список заданий. Во всех операционных системах, таких как Linux, MacOS или Windows, есть специальные программы, называемые планировщиками, которые делают именно это с программами на вашем компьютере.

У вас есть программа, которая распараллеливается и использует n независимых потоков для обработки заданного списка m заданий. Потоки берут задания в том порядке, в котором они указаны во входных данных. Если есть свободный поток, он немедленно берет следующее задание из списка. Если поток начал обработку задания, он не прерывается и не останавливается, пока не завершит

обработку задания. Если несколько потоков одновременно пытаются взять задания из списка, поток с меньшим индексом берет задание. Для каждого задания вы точно знаете, сколько времени потребуется любому потоку, чтобы обработать это задание, и это время одинаково для всех потоков.

Вам необходимо определить для каждого задания, какой поток будет его обрабатывать и когда он начнет обработку.

- **Формат ввода или входного файла (input.txt).** Первая строка содержит целые числа n и m ($1 \leq n \leq 10^5$, $1 \leq m \leq 10^5$). Вторая строка содержит m целых чисел t_i - время в секундах, которое требуется для выполнения i-ой задания любым потоком ($0 \leq t_i \leq 10^9$). Все эти значения даны в том порядке, в котором они подаются на выполнение. **Индексы потоков начинаются с 0.**
- **Формат выходного файла (output.txt).** Выведите в точности m строк, причем i-ая строка (начиная с 0) должна содержатьдва целочисленных значения: индекс потока, который выполняет i-ое задание, и время в секундах, когда этот поток начал выполнять задание.
- Ограничение по времени. 6 сек.
- Ограничение по памяти. 512 мб.
- Пример 1:

input.txt	output.txt
2 5 1 2 3 4 5	0 0 1 0 0 1 1 2 0 4

1. Два потока пытаются одновременно взять задания из списка, поэтому поток с индексом 0 фактически берет первое задание и начинает работать над ним в момент 0.
2. Поток с индексом 1 берет второе задание и начинает работать над ним также в момент 0.
3. Через 1 секунду поток 0 завершает первое задание, берет третье задание из списка и сразу же начинает его выполнять в момент времени 1.
4. Через секунду поток 1 завершает второе задание, берет четвертое задание из списка и сразу же начинает его выполнять в момент времени 2.
5. Наконец, еще через 2 секунды поток 0 завершает третье задание, берет пятое задание из списка и сразу же начинает его выполнять в момент времени 4.

- Пример 2:

input.txt
4 20 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

output.txt
0 0 1 0 2 0 3 0 0 1 1 1 2 1 3 1 0 2 1 2 2 2 3 2 0 3 1 3 2 3 3 3 0 4 1 4 2 4 3 4

Задания берутся 4 потоками по 4 штуки за раз, обрабатываются за 1 секунду, а затем приходит следующий набор из 4 заданий. Это происходит 5 раз, начиная с моментов 0, 1, 2, 3 и 4. После этого обрабатываются все $5 \times 4 = 20$ заданий.

- Что делать? Подумайте о последовательности событий, когда один из потоков становится свободным (в самом начале и позже, после завершения некоторого задания). Как применить очередь с приоритетами, чтобы имитировать обработку этих заданий в нужном порядке? Не забудьте рассмотреть случай, когда одновременно освобождаются несколько потоков.

6 задача. 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 (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^6$) - 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 (output.txt).** 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 «*».
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
8	2
A 3	1
A 4	3
A 2	*
X	
D 2 1	
X	
X	
X	

7 задача. Снова сортировка

Напишите программу пирамидальной сортировки на Python для последовательности в **убывающем порядке**. Проверьте ее, создав несколько рандомных массивов, подходящих под параметры:

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, ***по модулю*** не превосходящих 10^9.

- **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным по невозрастанию массивом. Между любыми двумя числами должен стоять ровно один пробел.
- Для проверки можно выбрать случай, когда сортируется массив размера $10^3, 10^4, 10^5$ чисел порядка 10^9, отсортированных в обратном порядке; когда массив уже отсортирован в нужном порядке; когда много одинаковых элементов, всего 4-5 уникальных; средний - случайный. Сравните на данных сетах `Randomized-QuickSort`, `MergeSort`, `HeapSort`, `InsertionSort`.
- Есть ли случай, когда сортировка пирамидой выполнится за $O(n)$?

* Напишите процедуру `Max-Heapify`, в которой вместо рекурсивного вызова использовалась бы итеративная конструкция (цикл).