

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №6
по курсу «Алгоритмы и структуры данных»
Тема: Деревья. Пирамида, пирамидальная сортировка.
Очередь с приоритетами
Вариант 27

Выполнил:
Филиппов А.Э.
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург 2022 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3-12
Задача №1. Куча ли?	3-5
Задача №5. Планировщик заданий	5-12
Вывод	13

Задачи по варианту

Задача №1. Куча ли?

Структуру данных «куча», или, более конкретно, «неубывающая пирамида», можно реализовать на основе массива.

Для этого должно выполняться 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 ($1 \leq n \leq 106$). 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

Код:

```
import time
import tracemalloc
```

```

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()

n = int(lines[0])
a = [-1] + list(map(int, lines[1].split()))
out = open("output.txt", "w")

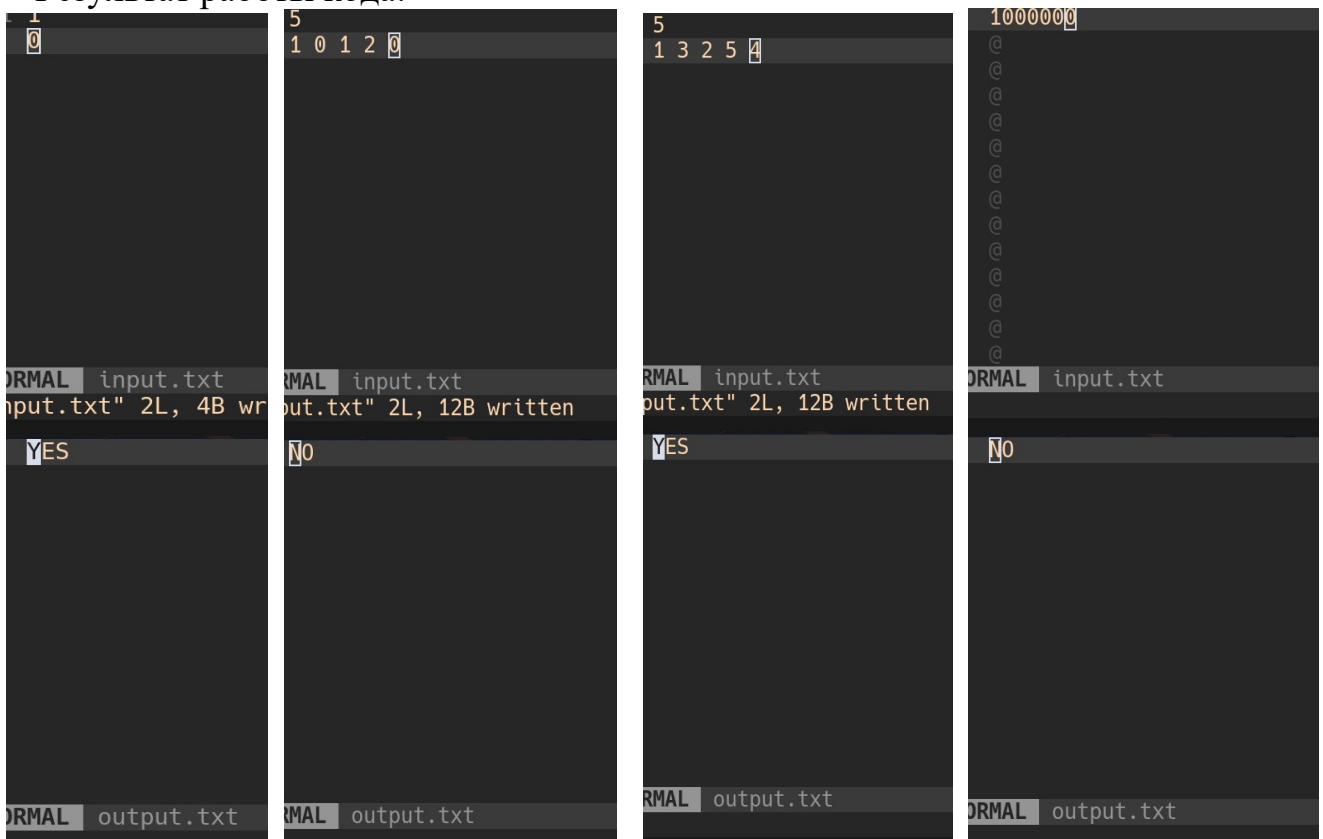
for i in range(1, n+1):
    if i % 2 == 0:
        parent = a[i//2]
    else:
        parent = a[(i-1)//2]
    if a[i] <= parent:
        out.write("NO")
        print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
        print("Использование памяти: " +
              str(tracemalloc.get_traced_memory()[1]) + " байт")
        tracemalloc.stop()
        exit()

out.write("YES")
print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для каждого элемента кроме корневого проверяется условие, что элемент меньше родителя. Таким образом это можно сделать одним циклом.

Результат работы кода:



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00014038600011190283	13939
Пример из задачи	0.00015002100008132402	13947
Пример из задачи	0.00015819300006114645	13947
Верхняя граница диапазона значений входных данных из текста задачи	0.7735960230002092	104621340

Вывод по задаче: Проверка массива на кучу — простая задача. Ведь ее сложность всего лишь $O(n)$.

Задача №5. 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 ($1 \leq n \leq 105$, $1 \leq m \leq 105$). Вторая строка содержит m целых чисел t_i - время в секундах, которое требуется для выполнения i -ой задания любым потоком ($0 \leq t_i \leq 109$). 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, при чем i -ая строка (начиная с 0) должна содержать два 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	0 0
1 2 3 4 5	1 0
	0 1
	1 2
	0 4

1. Два потока пытаются одновременно взять задания из списка, поэто му 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 за дание 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.

Код:

```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

class Heap:
    memory = []
    key = 0
    val = 1

    def __init__(self, key=0, val=1):
        self.key = key
        self.val = val

    @property
    def size(self):
        return len(self.memory) - 1

    def push(self, num):
        key = self.key
        val = self.val
```



```

self.memory.append(num)
parent = (self.size - 1) // 2
index = self.size
while index > 0:
    if self.memory[parent][key] > self.memory[index][key]:
        self.memory[parent], self.memory[index] = self.memory[
            index], self.memory[parent]
        index = parent
        parent = (index - 1) // 2
    elif self.memory[parent][key] == self.memory[index][key]:
        if self.memory[parent][val] > self.memory[index][val]:
            self.memory[parent], self.memory[index] = self.memory[
                index], self.memory[parent]
            index = parent
            parent = (index - 1) // 2
        else:
            break
    else:
        break

@property
def min(self):
    if self.memory:
        return self.memory[0]
    else:
        return None

def heapify(self):
    key = self.key
    val = self.val
    index = 0
    while 2 * index + 2 <= self.size:
        root = self.memory[index][key]
        root_val = self.memory[index][val]
        if self.memory[2 * index + 1][key] < self.memory[2 * index + 2]
[key]:
            minimum = 2 * index + 1
        elif self.memory[2 * index + 1][key] == self.memory[2 * index +
2][key]:
            if self.memory[2 * index + 1][val] < self.memory[2 * index
+ 2][val]:
                minimum = 2 * index + 1
            else:
                minimum = 2 * index + 2
        else:
            minimum = 2 * index + 2
        if self.memory[minimum][key] < root:
            self.memory[minimum], self.memory[index] = self.memory[
                index], self.memory[minimum]
            index = minimum
        elif self.memory[minimum][key] == root:
            if self.memory[minimum][val] < root_val:
                self.memory[minimum], self.memory[index] = self.memory[

```

```

        index], self.memory[minimum]
        index = minimum
    else:
        break
    else:
        break
    if 2 * index + 1 <= self.size:
        root = self.memory[index][key]
        if self.memory[2 * index + 1][key] < root:
            self.memory[
                2 * index +
                1], self.memory[index] = self.memory[index],
self.memory[
                2 * index + 1]
        elif self.memory[2 * index + 1][key] == root:
            if self.memory[2 * index + 1][val] < root_val:
                self.memory[
                    2 * index +
                    1], self.memory[index] = self.memory[index],
self.memory[
                    2 * index + 1]

    def del_min(self):
        self.memory[0], self.memory[self.size] = self.memory[
            self.size], self.memory[0]
        self.memory.pop()
        self.heapify()

n, m = map(int, lines[0].split())
tasks = list(map(int, lines[1].split()))

threads = Heap(key="time", val="index")

for i in range(n):
    threads.push({
        "time": 0,
        "index": i
    })
count = 0
for task_time in tasks:
    thread = threads.min # minimal available thread
    count += 1
    out.write(f"{thread['index']} {thread['time']}"+"\n")
    threads.del_min()

# current thread will be available after completing the task
threads.push({
    "time": thread["time"]+task_time,
    "index": thread["index"]
})

```

```

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для решения данной задачи использовалась структура данных «Куча». Куча использует в качестве ключа время доступности потока.

Результат работы кода:

```

2 5
1 2 3 4 5

RMAL input.txt

0 0
1 0
0 1
1 2
0 4

RMAL output.txt

2 652924150 946299544 531502802 892267892 204548210 690664844 396132264 151566055 440500777 467814454 702355496 5914860
3 28530424 258359962 111172044 951228804 104185090 14059660 511022062 565405469 824842936 652291719 743049781 579698359 4
3344134 486379423 210867097 849529200 98886195 820702466 914364575 245228189 15894489 85726988 79609640 936888749 7977824
9 44225452 918898541 178500612 787184551 487282920 888276435 438084849 312036247 530063180 813360093 539105096 784420013
07903029 829204545 570876300 961419798 117747053 342555815 474749715 892900587 758457782 187076326 709253237 739492843 66
279454 471872374 342730018 926626796 322543269 272339693 314385812 701058478 12989307 929175302 599558680 16248449 541630
63 565489968 430895459 916385062 983889122 296020349 434921011 377102961 887248152 995735631 797713276 340646001 60644839
83573549 973569251 526964792 859109136 945111130 985012858 89332187 857822443 588664508 538901338 852041013 445504849 155
6494 749826745 738497179 96714994 663895961 856393000 77438388 276353620 791579059 988500751 806093397 240334207 34187368
648559478 549730296 843377989 681075246 550909206 574416918 25070927 465807676 405816147 303330842 636998004 616308008 4
8463544 902609399 991501272 578989395 827553922 167598912 420205695 824905746 349613825 892008043 291963468 507263534 199
15247 489291073 433194958 621064623 197482370 497391308 769084471 999374298 800090815 418186138 929478705 637490912 51369
951 687708771 40699233 386989971 894880796 906059123 625210409 659765048 468260630 72452085 960697688 243820754 736480865
982036785 237818098 913150534 920861483 474345882 986047270 524238210 359709257 972239555 834728581 60712011 441967395 91
RMAL input.txt text utf-8[unix] 100,002 words 100% in :2/23:1 [2]trailing
0 0
1 1 0
2 2 0
3 3 0
4 4 0
5 5 0
6 6 0
7 7 0
8 8 0
9 9 0
10 10 0
11 11 0
12 12 0
13 13 0

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001967490006791195	16875
Пример из задачи	0.00019508200057316571	17425
Верхняя граница диапазона значений входных данных из текста задачи	5.819984706000014	34267843

Вывод по задаче: С помощью структуры данных heap, можно сделать оптимизацию. Куча сводит сложность алгоритма нахождения минимального элемента к $\log n$

Вывод

Деревья, пирамида и очередь с приоритетами широко используются в прикладных задачах. Без этих структур нельзя представить современные операционные системы и сервисы. Ведь на больших массивах данных оптимизация играет большую роль.