Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа № 12

по дисциплине «Алгоритмы и структуры данных»

Stepik –жадные алгоритмы (разделы 4-5)

Подготовил:

студент группы P3217

Бураков Илья Алексеевич

Преподаватели:

Романов Алексей Андреевич

Волчек Дмитрий Геннадьевич

Санкт-Петербург, 2019

Задача на программирование: покрыть отрезки точками

По данным n отрезкам необходимо найти множество точек минимального размера, для которого каждый из отрезков содержит хотя бы одну из точек.

В первой строке дано число $1 \le n \le 100$ отрезков. Каждая из последующих n строк содержит по два числа $0 \le l \le r \le 10^9$, задающих начало и конец отрезка. Выведите оптимальное число m точек и сами m точек. Если таких множеств точек несколько, выведите любое из них.

Sample Input 1:

```
3
1 3
2 5
3 6
```

Sample Output 1:

```
1
3
```

Sample Input 2:

```
4
4 7
1 3
2 5
5 6
```

Sample Output 2:

```
2
3 6
```

Решение

```
from operator import itemgetter

n = int(input())
areas = sorted(map(lambda s: list(map(int, s.split(" "))), (input() for i in
range(n))), key=itemgetter(1))
dots = []
for s, e in areas:
    if not dots or s > dots[-1]:
        dots.append(e)

print(len(dots))
print(" ".join(map(str, dots)))
```

Задача на программирование: непрерывный рюкзак

Первая строка содержит количество предметов $1 \le n \le 10^3$ и вместимость рюкзака $0 \le W \le 2 \cdot 10^6$. Каждая из следующих n строк задаёт стоимость $0 \le c_i \le 2 \cdot 10^6$ и объём $0 < w_i \le 2 \cdot 10^6$ предмета (n, W, c_i, w_i — целые числа). Выведите максимальную стоимость частей предметов (от каждого предмета можно отделить любую часть, стоимость и объём при этом пропорционально уменьшатся), помещающихся в данный рюкзак, с точностью не менее трёх знаков после запятой.

Sample Input:

```
3 50
60 20
100 50
120 30
```

Sample Output:

```
180.000
```

Решение

```
n, tw = map(int, input().split())


class Product:
    def __init__(self, cost, weight):
        self.cost = cost
        self.weight = weight

    @property
    def wcost(self):
        return self.cost / self.weight


goods = sorted(map(lambda s: Product(*map(int, s.split())), (input() for i in
range(n))),
               key=lambda p: p.wcost)

tc = 0
while tw and goods:
    nxt = goods.pop()
    taken = min(tw, nxt.weight)
    tc += taken * nxt.wcost
    tw -= taken

print(tc)
```

Задача на программирование: различные слагаемые

По данному числу $1 \le n \le 10^9$ найдите максимальное число k, для которого n можно представить как сумму k различных натуральных слагаемых. Выведите в первой строке число k, во второй — k слагаемых.

Sample Input 1:

```
4
```

Sample Output 1:

```
2
1 3
```

Sample Input 2:

```
6
```

Sample Output 2:

```
3
1 2 3
```

Решение

```
n = int(input())
k = int((((8 * n + 1) ** 0.5) - 1) / 2)
print(k)
seq_nums = list(range(1, k))
seq_nums_sum = int((k - 1) / 2 * k)
print(" ".join(map(str, seq_nums + [n - seq_nums_sum])))
```

Задача на программирование: кодирование Хаффмана

По данной непустой строке Ss длины не более 10^4, состоящей из строчных букв латинского алфавита, постройте оптимальный беспрефиксный код. В первой строке выведите количество различных букв k, встречающихся в строке, и размер получившейся закодированной строки. В следующих k строках запишите коды букв в формате "letter: code". В последней строке выведите закодированную строку.

Sample Input 1:

```
a
```

Sample Output 1:

```
1 1
a: 0
0
```

Sample Input 2:

```
abacabad
```

Sample Output 2:

```
4 14
a: 0
b: 10
c: 110
d: 111
01001100100111
```

Решение

```
import functools
from collections import Counter, namedtuple
from queue import PriorityQueue

q = PriorityQueue()
s = input()
n = len(s)


class Node(namedtuple("Node", ["left", "right"])):
    def walk(self, code, acc):
        self.left.walk(code, acc + "0")
        self.right.walk(code, acc + "1")


class Leaf(namedtuple("Leaf", ["char"])):
    def walk(self, code, acc):
        code[self.char] = acc or "0"


@functools.total_ordering
class Entry:
    def __init__(self, freq, node):
        self.freq = freq
        self.node = node

    def __lt__(self, other):
        return self.freq < other.freq

    def __eq__(self, other):
        return self.freq == other.freq

    def tuple(self):
        return self.freq, self.node


for c, freq in Counter(s).items():
    q.put(Entry(freq, Leaf(c)))

while q.qsize() > 1:
    f1, ln = q.get_nowait().tuple()
    f2, rn = q.get().tuple()
    q.put(Entry(f1 + f2, Node(ln, rn)))

_f, root = q.get_nowait().tuple()
code = {}
root.walk(code, "")

encoded = "".join(code[c] for c in s)
print(len(code), len(encoded))
for c in sorted(code):
    print(f"{c}: {code[c]}")
print(encoded)
```

Задача на программирование: декодирование Хаффмана

Восстановите строку по её коду и беспрефиксному коду символов.

В первой строке входного файла заданы два целых числа k и l через пробел — количество различных букв, встречающихся в строке, и размер получившейся закодированной строки, соответственно. В следующих k строках записаны коды букв в формате "letter:

code". Ни один код не является префиксом другого. Буквы могут быть перечислены в любом порядке. В качестве букв могут встречаться лишь строчные буквы латинского алфавита; каждая из этих букв встречается в строке хотя бы один раз. Наконец, в последней строке записана закодированная строка. Исходная строка и коды всех букв непусты. Заданный код таков, что закодированная строка имеет минимальный возможный размер.

В первой строке выходного файла выведите строку s. Она должна состоять из строчных букв латинского алфавита. Гарантируется, что длина правильного ответа не превосходит 10^4 символов.

Sample Input 1:

```
1 1
a: 0
0
```

Sample Output 1:

```
a
```

Sample Input 2:

```
4 14
a: 0
b: 10
c: 110
d: 111
01001100100111
```

Sample Output 2:

```
abacabad
```

Решение

```
n, m = map(int, input().split())
rcode = {}
for i in range(n):
    c, seq = input().split(": ")
    rcode[seq] = c

encoded = input()
decoded = []
cc = ""
for i in range(m):
    if cc in rcode:
        decoded.append(rcode[cc])
        cc = encoded[i]
    else:
        cc += encoded[i]
decoded.append(rcode[cc])

print("".join(decoded))
```

Задача на программирование: очередь с приоритетами

Первая строка входа содержит число операций $1 \leq n \leq 10^5$. Каждая из последующих n строк задают операцию одного из следующих двух типов:

- `Insert x`, где $0 \leq x \leq 10^9$ — целое число;
- `ExtractMax`.

Первая операция добавляет число x в очередь с приоритетами, вторая — извлекает максимальное число и выводит его.

Sample Input:

```
6
Insert 200
Insert 10
ExtractMax
Insert 5
Insert 500
ExtractMax
```

Sample Output:

```
200
500
```

Решение

```
heap = ['_']


def swap(i1, i2):
    heap[i1], heap[i2] = heap[i2], heap[i1]


n = int(input())

for i in range(n):
    inp = input().split()
    cmd = inp[0]
    if cmd == "Insert":
        arg = int(inp[1])

        heap.append(arg)
        ni = len(heap) - 1
        while ni // 2 and heap[ni // 2] < arg:
            swap(ni // 2, ni)
            ni = ni // 2
    else:
        print(heap[1])
        nv = heap.pop()
        if len(heap) == 1:
            continue

        heap[1] = nv
        ni = 1
        while ni * 2 + 1 < len(heap):
            c1, c2 = heap[ni * 2], heap[ni * 2 + 1]
            if nv >= max(c1, c2):
                break
            elif c2 >= c1:
                # sinking to right subtree
                nni = ni * 2 + 1
                swap(nni, ni)
                ni = nni
            elif c1 > c2:
                # sinking to the left
```

```
            nni = ni * 2
            swap(nni, ni)
            ni = nni

    if ni * 2 < len(heap) and nv < heap[ni * 2]:
        swap(ni, ni * 2)
```