Так, собственно — условие https://coderun.yandex.ru/problem/work-schedule

```  
 Пете на работе выдали *n* задач.

Каждый день, начиная с первого, Петя может выполнить ровно одну задачу. Про каждую задачу известен последний день *di*​, когда её можно выполнить, и величина стресса *wi*​, который Петя испытает, если задача не будет выполнена в срок и надо будет просить помощи коллег.

Помогите Пете решить, в каком порядке выполнять задачи, чтобы уменьшить его суммарный стресс.  
```

посмотрим на примере

5  
2 10  
1 5  
2 8  
1 7  
3 4

в задаче идёт речь о приоритетах и «наименьшем стрессе». Звучит как что-то связанное с дп, но — можно и эффективнее

приоритет, а какая структура данных нам может помочь с приоритетом? — приоритетная очередь

наша СД это приоритетная очередь, хорошо… а как приоритет выбирать

наш приоритет будет отображаться в стрессе. Но вопрос — кто первее выйдет — самое стрессовое задание, или всё же самое Не стрессовое?

Тк у нас задаче написано

```  
Помогите Пете решить, в каком порядке выполнять задачи, чтобы уменьшить его суммарный стресс.  
```

то нас **интересует** закрыть только самые **тяжёлые** задачи, а лёгкие — пусть уходят

определились с СД, как в ней определяется очередь. А что теперь? Как в Очередь то добавлять? В каком случае удалять из очереди?

Вообще очередь, во многих языках (если не во всех) — это бинарная куча  
тк нас интересует минимальных заборт кидать, и + это очередь — используется min-heap

было бы круто добавить все элементы в кучу и там — удалить сколько то первых элементов. Но нам мешает дедлайн

если я решу задачу с дедлайном 5 дней, хотя у меня есть задача до завтра, да и +

Алгоритм

итак — получаем данные  
затем — сортируем по Дедлайнам. Вот в таком порядке мы будем добавлять наши задачи в очередь

1 5  
1 7  
2 10  
2 8  
3 4

затем — проходимся по задачам и добавляем их в очередь  
 далее рассмотрим процесс через пример  
 у нас нулевой день, Петя планирует своё развлечение на несколько дней  
 - очередь пуста. Добавляем первый по списку : 1 5  
 теперь очередь : 1 5  
 так, в рамки дедлайна проверяем (размер очереди)=1 > (1 5).day=1? нет — значит уместимся в дедлайн, если добавим эту задачу в очередь  
 - круто, идём дальше : 1 7  
 добавляем в очередь С ПРИОРИТЕТОМ :  
 как добавлять в min-heap я не буду, думаю процесс и так понятен. Главное что теперь очередь : (1 5 , 1 7)  
 проверяем (1 5, 1 7).size=2 > (1 7).day=1 ? Да, больше. Следовательно удаляем наш (1 5), и теперь очередь у нас 1 7  
 - далее добавим 2 10  
 очередь : 1 7, 2 10  
 queue.size=2 > 2? нет — всё круто  
 - далее 2 8  
 очередь: 1 7, 2 8, 2 10  
 размер=3 > 2? да — удаляем 1 7  
 теперь очередь: 2 8, 2 10

- 3 4  
 очередь : 3 4, 2 8, 2 10   
 размер=3 > 3? нет — всё круто  
итого — итоговая очередь — список задач, которые нам нужно решить, чтобы получить наименьший emotional damage  
а чтобы получить totalStress — нужно было иметь какую-то переменную, которая будет += удаляемая задача.w

как ещё можно словесно описать алгоритм:  
 так, вот мои задачи. Хм, мне бы стоило их сначала по дням просортировать, а то — добавил за пятый день И решил её, хотя у меня были задачи за 1 день, которые я мог бы решить Сегодня, а вот завтра уже 5 дня, если там нет такого же казуса со 2 днём.  
 Хорошо, у меня максимум 3 дня, я могу решить 3 задачи суммарно  
 так, 1 5 — ну в принципе пока норм, добавлю  
 так, 1 7 — дедлайн такой же, как и с уже имеющейся 1 5, могу по сути то и только одну из них решить, поэтому — решу 1 7, она мне более стрессовая  
 2 10, у меня пока на два дня 1 7 и 2 10 — умещаюсь в эти два дня, пока пусть будет  
 2 8 — 1 7, 2 8, 2 10. т. е., у меня как бы два дня. 1 7 я могу решить только в первый день, хотя -- если бы я решил вместо неё 2 8, а потом на следующий день 2 10, вместо 1 7 И 2 10, то банально бы меньше стресса получил. Поэтому лучше я 2 8 решу в первый день, а потом 2 10  
 3 4 — так, ну тут у меня уже 3 4, 2 8, 2 10, 3 дня , всё по дедлайном прохожу — в первый день 2 8, потом 2 10, а тут можно и 3 4  
 и ведь — если бы я выбрал 1 7, 2 8, 2 10 — то пропустил бы дедлайн по 2 10 или 2 8, поэтому стоило бы 1 7, 2 8, 3 4, а это такой стресс и неэффективно.

Это

что характеризует собой Очередь? — это задачи, да, Они имеют свой порядок. И вот есть каждый день решать по задаче оттуда, то В ЭТОМ ПОРЯДКЕ (ладно, нужно идти справа на налево в ней), то всё будет в порядке и стресс будет минимальным

что собой моделирует предсортировка по дедлайну и потом такое добавление. — помогает нам не пропустить задачи по дням И вот эта проверка помогает нам «не брать лишнее на день».  
 т. е., в ячейку очереди по номеру i (если читать слева направо), может находится задача с дедлайном i и выше

вот я добавил две задачи 1 5 и 1 7, но день последней отсортированной задачи нам показывает, что -- у нас пока дедлайн не позволяет.

Может тут как раз и вылазит Динамическое Программирование? — мы постепенно добавляем, типо постепенно строим свой план, как бы основываясь на прошлом результате? — наверное. Но это пример жадного алгоритма

код на Java

package org.shuman;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.util.Arrays;

import java.util.Comparator;

import java.util.PriorityQueue;

public class Main436 {

public static void main(String[] args) throws IOException {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(System.out));

int n = Integer.parseInt(reader.readLine());

Task[] tasks = new Task[n];

String[] line;

for (int i = 0; i < n; ++i) {

line = reader.readLine().split(" ");

tasks[i] = new Task(

Integer.parseInt(line[0]),

Integer.parseInt(line[1])

);

}

Arrays.sort(tasks, Comparator.comparingInt(t -> t.d));

PriorityQueue<Integer> maxBinaryHeap = new PriorityQueue<>();

long totalStress = 0;

for (Task task : tasks) {

maxBinaryHeap.offer(task.w);

if (maxBinaryHeap.size() > task.d) {

totalStress += maxBinaryHeap.poll();

}

}

writer.write(Long.toString(totalStress));

reader.close();

writer.close();

}

static class Task {

public int d;

public int w;

Task(int d, int w) {

this.d = d;

this.w = w;

}

}

}