# АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
# НАЙ-ДОБРА ПРОИЗВОДИТЕЛНОСТ

Подзадача 1

Ограниченията на подзадача 1 са малки, така че каквото и да е изчерпване на всички възможни назначения на работници към линии ще мине.

Подзадача 2

Решението на подзадача 2 използва динамично програмиране по подмножества. Нека с F(p, W) означим максималната сумарна производителност с p линии и множество работници W. F(p, W) е минус безкрайност ако няма валидно разпределение на множеството работници W на p линии.

F(0, { }) = 0
F(p, W) = MAX(F(p - 1, A) + work_together(W \ A)), където A ≠ W, A ⊂W.
work_together(B) е максималното време, за което може работниците от множество B да работят на една линия (минималното крайно време минус максималното начално време на работник в B), или минус безкрайност ако работниците от това множество не могат да работят заедно.

Подмножествата W, A може да представим като числа x между 0 и $2^N$ - 1. Ако битът на позиция i в x е единица, това означава, че работник i е в множеството, което x представя. Като избираме A подмножество на W можем да избегнем елементи, които не са действителни подмножества със следния трик:
**for (int submask = mask; submask; submask = (submask - 1) & mask)**

Сложността на решението ще бъде O(P * $3^N$).
Оставяме като упражнение за читателя да се убеди в сложността и трика с подмаските.
Повече информация за динамично програмиране с битови маски може да намерите на:
http://codeforces.com/blog/entry/337

Подзадача 3

Основната стъпка към решението за 100 точки е подзадача 3. Разбиваме множеството работници на “добри” и “лоши”. Един работник X е добър, ако съществува работник Y, такъв че интервалът време, в който Y е на работа се съдържа изцяло в интервала време, в което X е на работа. Тоест, ако сме назначили всички работници освен X на линии, ние винаги ще можем да назначим X на линията, на която работи Y и решението ще остане валидно.
Всички работници, които не са добри наричаме “лоши”.
Нека подредим лошите работници по момент на пристигане на работа в нарастващ ред. Лошите работници са {(A1, B1), (A2, B2), ...,} : Ai <= A(i+1). Тогава лошите работници ще бъдат подредени и по момент на заминаване в нарастващ ред.

Ако това не беше вярно и имаме i < j : Ai <= Aj, но Bi >= Bj. Тогава работното време на (Aj, Bj) се съдържа в работното време на (Ai, Bi). Тоест (Ai, Bi) е добър работник. Нещо повече, нямаме лоши работници, които започват или свършват работа в един и същи момент. Тоест: A1 < A2 < … и B1 < B2 < … (*)

Как да назначим лошите работници на линии?
Трябва да има линия, на която работи (A1, B1). Нека лошият работник с най-голям номер, който работи на тази линия е (Ai, Bi). Тази линия ще работи от време Ai до време B1. Без да загубим време, в което линията работи, можем да назначим работници (Aj, Bj) на същата линия за всяко j : 1 <= j <= i (заради (*)). С подобни разсъждения може да се убедим, че на всяка линия ще работят последователен интервал лоши работници от сортирания списък.
Нека с G(q, k) означим максималното сумарно работно време на q линии, на които сме назначили лоши работници (A1, B1), (A2, B2), …, (Ak, Bk).
G(q, k) може да се смята с динамично програмиране:
G(0, 0) = 0
G(q, k) = MAX(G(q - 1, j) + work_together(j + 1, k)) за 1 <= j < k. work_together е времето, за което работниците j + 1, j + 2, …, k могат да работят заедно: B(j + 1) - Ak
Тази формула може да се смята за $O(P * N^2)$.

Ами добрите работници?
Както вече установихме, добрите работници изцяло покриват лошите, и могат да бъдат добавени без да влошат решението. Обаче, добрите работници могат да подобрят решението. Ако назначим лошите работници на q линии, на останалите P - q линии можем да сложим добри работници. Най-доброто решение би било да сложим P-q добри работници, които работят най-дълго и да ги оставим сами на линии. Ако добавим друг работник, няма да подобрим решението.

Накрая, за да получим отговора, трябва да вземем максимума от {G(q, #BAD) + top_good(P-q)} - стойност от динамичното плюс най-големите добри работници.

Директна имплементация с $O(P * N^2)$ е достатъчна, за да решим подзадача 3.

Подзадача 4

За подзадача 4 е необходимо да свалим сложността. Да забележим, че стойностите j, които итерираме, за да намерим G(q, k), образуват интервал. Когато j е прекалено малко, няма да може да назначим всички лоши работници от j + 1 до k на една линия.

G(q, k) = MAX{G(q - 1, j) + B(j + 1) - Ak} = MAX{G(q - 1, j) + B(j + 1)} - Ak: j0 <= j < k
j0: min такова, че B(j0 + 1) >= Ak.
Нещо повече J0(q, k) <= J0(q, k + 1) (B(j0), Ak са сортирани в нарастващ ред)
Тоест, може да смятаме формулата на динамичното програмиране G(q, k) итеративно и да държим всички кандидати за подобрение на G(q, k) в опашка, от която можем да

добавяме елемент накрая, премахваме елемент от началото и да взимаме максимум с константно време. В опашката държим (G(q - 1, j) + B(j+1)) за j0 <= j < k. j0 и k само растат. Един начин да реализираме такава опашка е описан в http://stackoverflow.com/questions/4802038/implement-a-queue-in-which-push-rear-pop-front-and-get-min-are-all-consta - може да използваме два стека, всеки от които поддържа взимане на максимум.
Друг начин за реализиране е да пазим в опашка само тези елементи, които ще станат максимум ако спрем да добавяме елементи. Това е стандартна техника, която се среща в други задачи и не е основния фокус на тази задача.

Решението става със сложност O(N * P)

Подзадача 5

За последната подзадача е необходимо да пазим само 2 реда от динамичното, защото паметта не е достатъчна да пазим цялата таблица с размер N * P.
Забележете, че G(q, k) зависи само от елементи G(q - 1, .). Това ни позволява да пазим само два реда от таблицата в паметта.

Сложността на решението остава същата, но консумираната памет вече е O(N).

*Автор: Йордан Чапъров*