Розбір IV етапу Всеукраїнської олімпіади з інформатики 2020

Матвій Асландуков, Роман Білий, Антон Ципко

Ми дякуємо усім, хто допомагав з тестуванням задач:

Станіславу Безкоровайному, Ігорю Баренблату, Олегу Валласу, Марку Гришечкіну, Іллі Ковалю, В'ячеславу Очеретному, Даніїлу Смельському, Євгену Соболєву, Матвію Стреченю

Окрема подяка Ігорю Баренблату за вичитування умов

Задача A1. Хмарочоси

Автор задачі: **Антон Ципко**
Задачу підготував: **Марко Гришечкін**
Розбір написали: **Марко Гришечкін та Антон Ципко**

Блоки 1-3

Відсортуємо хмарочоси у порядку зростання (незростання) краси поверхів. Будемо розв'язувати задачу за допомогою динамічного програмування. Нехай dp_i — максимальна можлива сума, якщо розглядати лише перші i хмарочоси. Тоді

$$dp_i = \max_{j<i}(dp_j + (a_i - a_j) \cdot b_i)$$

Відповідь буде у dp_n.

Блок 4

Відсортуємо хмарочоси у порядку зростання (незростання) краси поверхів. Якщо побудувати хмарочоси саме в такому порядку, то сумарна краса поверхів буде максимально можливою.

/textbfДоведення: нехай m — максимальна висота серед усіх хмарочосів. Для будь-якої висоти від 1 до m з хмарочоса Вуса на такій висоті буде видно рівно один поверх одного з n хмарочосів. Доведемо, що при такому порядку хмарочосів, для довільної висоти i, з хмарочоса Вуса буде видно поверх з максимальною красою. Цей поверх належить першому зліва хмарочосу з висотою більшою або рівною i. Через те, що ми відсортували хмарочоси за красою, то цей хмарочос має найбільшу красу серед усіх хмарочосів з висотою більшою або рівною i.

Задача A2. Золоте поле

Автор задачі: **Антон Ципко**
Задачу підготував: **Антон Ципко**
Розбір написав: **Антон Ципко**

Блоки 1 та 4

Якщо сума усіх чисел парна, то можемо усі монетки перемістити в одну клітинку, наприклад, $(1, 1)$.

Блок 5

Можемо знайти найменше непарне число, після чого перемістити усі монетки, крім тієї купки, в одну.

Нехай координати (x, y) цієї купки такі, що $x > 1$ та $y > 1$. Тоді ми можемо кожну купку зсунути або ліворуч (тобто зменшити y), або угорі (тобто зменшити x). Ми це не можемо зробити лише для $(1, 1)$. Проте там усі наші монетки і залишаться.

Якщо ж $x = 1$ або $y = 1$, то ми можемо схожим методом перемістити усі монетки у (n, m).

Блок 6

Якщо кількість монеток парна, то застосовуємо алгоритм блоку 4, інакше — алгоритм блоку 5.

Блоки 2 та 3

Якщо m парне, то застосовуємо алгоритм блоку 1. Інакше для кожної позиції y ми знаходимо відповідь, якби ми зсунули усі монетки ліворуч у позицію $(1, y - 1)$, а усі праворуч — у $(1, y + 1)$. Серед усіх можливих відповідей — знаходимо найкращу.

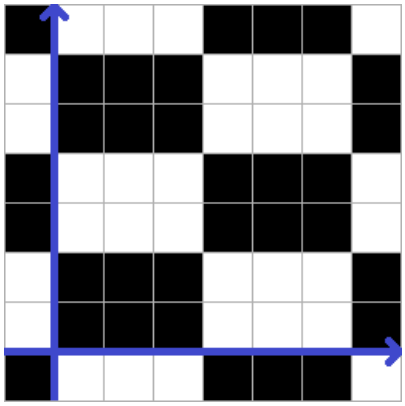
Тут потрібно звернути увагу, що якщо з певної сторони виходить непарне число, то його не можна додавати до відповіді.

Задача B1. Шахове поле

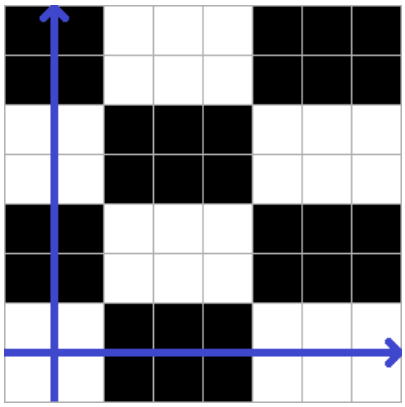
Автор задачі: **Антон Ципко**
Задачу підготував: **Станіслав Безкоровайний**
Розбір написав: **Станіслав Безкоровайний**

Примітка: На усіх ілюстрація до розв'язку вважатимемо, що $n = 2,\ m = 3$.

Нехай базовим полем є таке поле, що у ньому є чорний прямокутник, такий, що його лівий нижній кінець має координати $(1, 1)$. Тобто, таке поле виглядатиме так:



Тоді будь-яке інше поле можна отримати за допомогою деякого зсуву базового поля. Наприклад, якщо зсунути його на 2 клітинки вліво і на 1 вгору, отримаємо такий малюнок:



Твердження. Всього різних полів $2nm$. Усі види полів можна отримати зсунувши поле на деяке число a вліво та деяке число b вниз, де $0 \leqslant a \leqslant 2m - 1,\ 0 \leqslant b \leqslant n - 1$.

Якщо вас не цікавить доведення, перегорніть на 2 сторінку, де буде описуватись сам розв'язок.

Доведення. Нехай у нас є деяке поле, і деяка точка на ньому. У цієї точки є координати (x, y). Тоді нехай трійка чисел (c, x', y') — це її *шахові координати*, які формуються таким чином:

- c — дорівнює 1, якщо ця клітинка знаходиться в білому квадраті, і 0, якщо у чорному.
- x — відстань по осі OX від лівого нижнього кута прямокутника, в якому ця клітинка знаходиться.
- y — відстань по осі OY від лівого нижнього кута прямокутника, в якому ця клітинка знаходиться.

Якщо ми зафіксуємо деяку точку та знатимемо її шахові координати, то ми зможемо однозначно побудувати поле, дізнавшись координати лівого нижнього кута прямокутника, в якому ця клітинка знаходиться та колір цього прямокутника. Очевидно, що для деякої фіксованої точки для кожної трійки будуть різні поля.

Можливих значень c — 2, x' — m, y' — n. Отже, всього різних полів $2nm$.

Отже, ми знаємо, що всього полів $2nm$. Для того, щоб довести друге твердження достатньо довести, що при будь-якому зсуві поля на деяке число a вліво та деяке число b вниз, де $0 \leqslant a \leqslant 2m-1,\ 0 \leqslant b \leqslant n-1$, ми будемо отримувати різні поля.

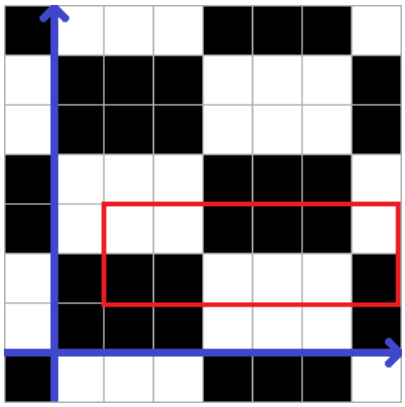
Перше, що варто помітити, що при зсуванні поля на a клітинок вліво і на b вниз, шахові координати клітинок змінюються так само, як якби x-координату збільшили на a, а y-координату на b.

Зафіксуємо точку з шаховими координатами (c, x', y'). Без обмеження загальності скажемо, що $c = 0$ (клітинка у чорному прямокутнику). Також скажемо, що $x' \neq 0, y' \neq 0$ (ці випадки доводяться ще легше).

Тоді можливі такі випадки:

- $0 \leqslant a \leqslant m-x'-1,\ 0 \leqslant b \leqslant n-y'-1$. Тоді ми будемо залишатись всередині одного прямокутника, отже шахові координаті будуть дорівнювати $(0, x'+a, y'+b)$.
- $0 \leqslant a \leqslant m-x'-1,\ n-y' \leqslant b \leqslant n$. Тоді ми перейдемо в новий білий квадратик при зсуві вниз, і наші координати будуть $(1, x'+a, b-n+y')$
- І т.д.... Продовжити цей список читачу на домашнє завдання =). В кінці маємо отримати всі можливі види шахових координат.

Для інтуїтивного розуміння ось ілюстрація (червоним виділено ті місця, в які ми можемо потрапити з точки $(2, 2)$ при зсуві):



Можна бачити, як зі шматочків білих та чорних квадратиків, на які можна потрапити можна зібрати білий прямокутник та чорний прямокутник.

А тепер власне розв'язок:

Це була центральна лема цієї задачі. Ось, що треба зробити, щоб отримати бали за блоки:

Нагадаємо, що при зсуві поля на a клітинок вліво та b клітинок вниз, кольори клітинок змінюються также само, як і при додаванні до відповідних координат клітинок чисел a і b.

Блок 1

У першому блоці можна скласти велике поле, наприклад, 5000×5000, (двовимірний масив символів або чисел). І перевіряючи на цьому полі чи можна зсунути точку на кожен з можливих a і b, будемо рахувати кількість зсувів, які підходять для всіх точок.

Асимптотика $O(S^2 + 2nm \cdot k)$, де $S \times S$ — розмір вашого поля.

Розв'язок: https://pastebin.com/NWqwwG3B

Блок 2

Тепер ми будемо робити те саме, що у першому прикладі, але тепер замість того, щоб явно будувати поле, навчимося за $O(1)$ шукати колір клітинки за її координатами.

Підсказка. Для того, що знайти колір клітинки за її координатами, варто вважати, що спочатку у нас базове поле. Також дуже зручно вважати, що координати клітинок починаються не з 1, а з 0. Спробуйте вивести формулу. Якщо не зможете, то погляньте на функцію *getColor* у розв'язку.

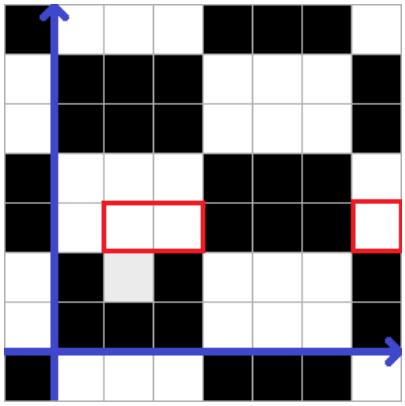
Асимптотика $O(2nm \cdot k)$.

Розв'язок: https://pastebin.com/2q72v6TL

Блок 3

У розв'язку третього блоку я буду вважати, що $n \leqslant m$. Якщо це не так, то робимо аналогічно, але тепер знаходимо відрізки не для зсувів по вертикалі, а для зсувів по горизонталі.

Давайте для кожного зсуву точок вгору від 0 до $n-1$ знайдемо для кожної точки відрізки, які їй підходять. Ось наприклад, для точки з координатами $(2, 2)$, потрібним їй білим кольором та вертикальним зсувом 1, їй підходять такі зсуви вправо (світло-сірим виділено саму клітинку, червоним обведено можливі зсуви):



Потім окремо для кожного з n вертикальних зсувів знаходимо множину таких зсувів по горизонталі, що підходять нам. Для цього можна використати scanline, щоб знайти множину таких точок, які покриваються k відрізками. Зауважте, що це не перетин відрізків, бо для одного вертикального зсуву для точки може бути не один, а два відрізка.

Для тих, хто ніколи не використовував метод scanline, почитайте про нього більше тут: https://informatics.mccme.ru/mod/resource/view.php?id=39111

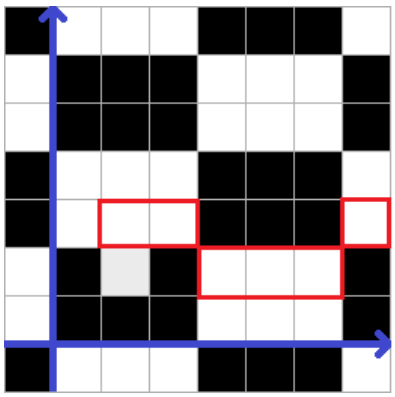
Сума кількістей таких точок для кожного з n зсувів точок вгору і є нашою відповіддю.

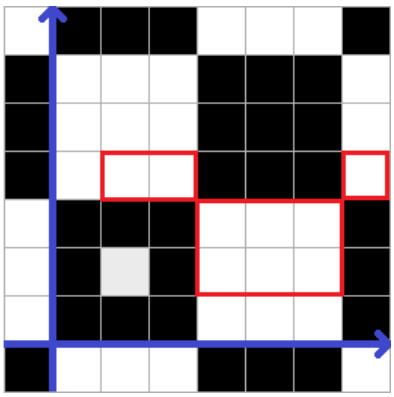
Асимптотика $O(k \cdot min(n, m))$.

Розв'язок: https://pastebin.com/w4TRJwu5

Блок 4

Якщо ми ще сильніше подивимось на ілюстрації, то зрозуміємо, що допустимі зсуви являють собою не випадковий набір відрізків, а прямокутнички, ось наприклад прямокутнички, для попередньої ілюстрації і для більшої зрозумілості, ілюстрація для випадку $n = m = 3$ (координати і потрібний колір точки ті самі):





Тепер ми знову можемо використати scanline, але тепер замість того, щоб додавати в scanline точки, будемо додавати відрізки. Наша задача тепер знайти площу, яка покрита k квадратиками.

Для того, щоб знаходити перетин відрізків у сканлайні вам знадобиться будь-яке збалансоване дерево пошуку. У мові програмування C++ для цього можна використати `set`.

Асимптотика $O(klog(k))$.

Розв'язок: https://pastebin.com/cpmL7TXX

Задача B2. Пари камінців

Автор задачі: **Роман Білий**
Задачу підготував: **Роман Білий**
Розбір написав: **Роман Білий**

Назвемо однією операцією сумарний хід Аліси та Боба. Якщо вони вибирають всі 4 купки різні, то вони фактично вибирають 4 числа (x, x, y, y) та змінюють їх на $(x-1, x-1, y-1, y-1)$. А якщо купки збігаються, то нехай Аліса вибрала (x, y), то щоб Боб міг взяти одну з купок, яку взяла Аліса, потрібно, щоб було $y = x-1$. В результаті вийде, що з $(x, x, x-1)$ утворились купки $(x-2, x-2, x-1)$.

Тобто є 2 можливі операції:

- $(x, x, y, y) \rightarrow (x, x, y, y)$
- $(x, x, x-1) \rightarrow (x-2, x-2, x-1)$

.

Тепер ми хочемо зробити якнайбільше операцій цих 2 типів і якщо в кінці залишаться хоча б 2 ненульові купки, то Аліса зможе зробити ще 1 свій хід.

Блок 1

У блоці 1 можна зробити повний перебір всіх можливих операцій. Щоб зменшити кількість станів, будемо тримати масив розмірів купок посортованим.

Блок 2

Розіб'ємо всі значення на пари. Нехай ми робимо операції тільки першого типу. За одну операцію можна вибрати 2 пари та зменшити їх на 1. Така задача розв'язується наступним чином. Нехай b_i — значення в парах, k — кількість пар(в даному блоці $k = n/2$). Посортуємо їх по спаданню, тобто b_1 — найбільше значення. Якщо $b_1 \leqslant \sum_{i=2}^{k} b_i$, то ми зможемо всі пари звести до 0, крім можливо одної, яка буде рівна 1, залежно від парності. Інакше ми можемо звести всі пари до 0, а найбільшу до $b_1 - \sum_{i=2}^{k} b_i$.

З цього також слідує, що використовувати операцію 2 типу не вигідно. Якщо є пари x та $x-1$ і жодна з них не найбільша, то ми тільки зменшимо $\sum_{i=2}^{k} b_i$. Якщо пара x — найбільша, то і операціями першого типу ми отримаємо найкращий результат.

Блоки 3-5

З опису операцій видно, що операції виконуються з парами. Тому якщо є непарна кількість значень x, то одне з них таким ж і залишиться. Назвемо такі купки стоячими. Після цього знову розіб'ємо всі інші купки на пари. Як і в розв'язку до попереднього блоку поганий є випадок, коли $b_1 > \sum_{i=2}^{k} b_i$. Тепер спробуємо використати стоячі купки і 2 операцію. 2 операцію вигідно використовувати тільки для найбільшої купки. Тобто якщо зараз найбільша купка рівна x і є стояче значення $x-1$, то перестрибнемо найбільшою купкою через це стояче значення і найбільша купка тепер буде рівна $x-2$. Тобто знайдемо, яке кінцеве найменше значення зможе мати найбільша купка. Спочатку вона буде рівна $r = b_1 - \sum_{i=2}^{k} b_i$. Далі будемо пробувати перестрибувати через стоячі купки. Якщо є стояча купка в межах $[r-1, b_1-1]$, то через неї можна перестрибнути і зменшити r на 2. Але не можна перестрибнути через стоячі купки з сусідними значеннями, тобто через y та $y-1$. У блоці 3 таких сусідних значень бути не могло.

Тобто розв'язок такий:

- Розіб'ємо всі значення на пари.
- Якщо $b_1 \leqslant \sum_{i=2}^{k} b_i$, то у всіх парах залишиться максимум одна пара зі значеннями 1 залежно від парності.

- Інакше знайдемо значення найбільшої купки, яке ми можемо отримати використовуючи тільки операції 1 типу $r = b_1 - \sum_{i=2}^{k} b_i$. Посортуємо всі стоячі купки до спаданню і спробуємо перестрибувати через них. Але перестрибувати через дві стоячі купки з сусідніми значеннями не можна.
- Знайдемо, чи залишились 2 непусті купки і в такому разі Аліса може зробити ще 1 хід.

Складність розв'язку $O(n \log(n))$.

Задача C1. Вгадайте колір

Автор задачі: **Роман Білий**
Задачу підготував: **Роман Білий**
Розбір написали: **Роман Білий та Ігор Баренблат**

Блок 1

Проітеруємось по всім кулькам і виконуватимемо по запиту з кожною з нею. Рівно один раз як результат отримаємо значення 2. Це відбудеться при запиті з одною з кульок, що повторюється. Іншу можна знайти виконавши ще по одному запиту з кожною з кульок.

Блок 2

Не обмежуючи загальності вважатимемо, що перша кулька — кулька кольору 1. Далі проітеруємося по всім іншим кулькам та виконаємо запит з кожною з них. Якщо значення, яке отримали рівне (останньому відомому значенню кульок кольору 1) + 1, то колір цієї кульки також 1, інакше 2. Для того, щоб однозначно визначати колір при черговому запиті, після кожного запиту виконуватимемо запит з кулькою номер 1.

Блок 5

Проітеруємось по всім кулькам та виконаємо запит з кожною з них. Відсортуємо отримані пари (значення результату запиту, номер кульки) по незростанню першого параметру. Тепер у порядку встановленому цим списком знову виконуватимемо по запиту з кожною кулькою — отримуючи результат запиту, можна однозначно визначити колір кульки.

Інший варіант розв'язати цей блок. Пройдемось двічі по всіх кульках зліва направо. Нехай a_i — значення, яке отримали при першому проході при запиті до i-ої кульки, b_i — при другому проході. Тоді є рівно $b_i - a_i$ кульок такого ж кольору, які такі ж, як і i-та кулька. Враховуючи, що кількість кульок кожного кольору різна, ця інформація також дозволяє відновити всі кольори.

Всі інші блоки

Зробимо один прохід по кульках зліва направо та знайдемо всі позиції, де колір зустрічається вперше. Назвемо їх базовими кульками. Розставимо на ці позиції кольори від 1 до k. Надалі нас буде цікавити тільки парність результату, тому зробимо ще один запит до всіх кульок, чим фактично обнулимо всі значення.

Тепер зробимо $log_2(k)$ ітерації, на i-ій ітерації знайдемо, в кольорах яких кульок на місці i-го біта стоїть 1. Зробимо по одному запиту до всіх базових кульок, в кольорах яких є i-ий біт. Тобто зараз на кульки цих кольорів ми дивились непарну кількість разів, а на всі інші парну. Тепер будемо проходитися до всіх інших кульках. Робимо запит до кульки, якщо результат парний, то цей біт 1, інакше 0. Щоб зберегти інваріант з парністю, робимо ще один запит до цієї ж кульки та переходимо далі до наступної кульки. В кінці ітерації знову робимо запит до базових кульок з одиничним бітом, щоб зробити всі значення парними.

Такий розв'язок використовує $2n(log_2(k) + 1)$ операцій.

https://ideone.com/MV6rw1

Альтернативне рішення

Реалізуємо функцію $solve(a)$, що приймає масив індексів кульок та встановлює їх відносні кольори — іншими словами розв'язує поставлену задачу. Складність функції $solve$ визначимо як $T_1(n)$ запитів, де $n = len(a)$.

Скористаємося методом «розділяй та володарюй». Розділимо a на два менших масиви L та R. Виконаємо $solve(L)$ та $solve(R)$. Для множин кульок L та R можемо залишити в них до розгляду лише по одній кульці кожного кольору. Залишається встановити пари кульок однакового кольору з L та R. Знайдемо множини таких кульок з L та R: для прикладу, знайти множину кульок з L що мають однаковий колір з кулькою з множини R можна проітерувавшись по L та виконавши по

запиту з кожною кулькою (також запам'ятавши значення результатів запитів), після чого необхідно виконати по запиту з кожною кулькою множини R, та знову виконати по запиту з кожною кулькою з множини L — тепер легко відрізнити кульки, колір який зустрічається серед кольорів кульок з R. Отримані множини назвемо $LToMerge$ та $RToMerge$.

Реалізуємо функцію $merge(L, R)$, що приймає два масиви однакової довжини індексів кульок, та визначає пари кульок однакового кольору, при цьому вважає що кольори кульок з L попарно різні, кольори кульок з R попарно різні та множини кольорів кульок з L та R рівні. Тоді для коректного виконання $solve$ достатньо буде викликати $merge(LToMerge, RToMerge)$. Складність функції $merge$ визначимо як $T_2(n)$ запитів, де $n = len(L) + len(R)$.

Реалізуємо $merge(L, R)$ за допомогою метода «розділяй та володарюй». Розділимо L на 2 масиви L_1 та L_2. Розділимо R на 2 масиви R_1 та R_2 такі, що кольори кульок з L_1 відповідатимуть кольорам кульок з R_1, а кольори кульок з L_2 відповідатимуть кольорам кульок з R_2. Для прикладу спочатку виконаємо по одному запиту з кожною з кульок з R (також запам'ятавши значення результатів запитів), після чого виконаємо по запиту з кожною з кульок з L_1 та знову виконаємо по запиту з кожною з кульок з R — тепер легко відрізнити кульки, колір який зустрічається серед кольорів кульок з L_1. Тоді для коректного виконання $merge$ достатньо буде викликати $merge(L_1, R_1)$ та $merge(L_2, R_2)$.

Складність рішення визначимо як $T(n)$ запитів.

Нехай $T_1(n) = C_1 \cdot n + 2 \cdot T_1(\frac{n}{2})$, $T_2(n) = C_2 \cdot n + 2 \cdot T_2(\frac{n}{2})$. Нескладно помітити, що сумарна довжина масивів що є вхідними параметрами функції $merge$ не більша за $2 \cdot n$. Іншими словами $T(n) \leqslant T_1(n) + T_2(2 \cdot n) = (C_1 + 2 \cdot C_2) \cdot n \cdot log(n)$.

Найменші значення C_1 та C_2 які журі змогло отримати при реалізації такого рішення рівні 1.5 та 0.75 відповідно. Рішення з такими параметрами C_1 та C_2 можна реалізувати враховуючи при вході у рекурсивні виклики з якими кульками виконувались останні запити. Реальна кількість запитів такого рішення звісно менша ніж $3 \cdot n \cdot log(n)$.

Задача C2. Топологічні сортування дерева

Автор задачі: **Матвій Асландуков**
Задачу підготував: **Матвій Асландуков**
Розбір написав: **Матвій Асландуков**

Блок 1

Оскільки усі числа від 1 до n повинні зустрічатися у масиві a рівно один раз, існує всього $n!$ можливих варіантів записати числа у вершинах дерева. Усі їх можна перебрати та просто перевірити, що виконуються усі $(n-1)$ необхідних умов. Складність такого розв'язку — $O(n \cdot n!)$.

Розв'язок — https://pastebin.com/M2VXx76u

Блок 2

Для розв'язання цієї групи можна було використати динамічне програмування по підмножинам. А саме, нехай `dp[mask]` — це кількість способів записати числа від 1 до m у вершинах дерева, номера яких присутні в масці `mask`. Тут m позначає кількість одиничних бітів в масці $mask$. При цьому враховуються тількі ті способи, для яких виконуються усі умови для тих вершин, що вже присутні в масці.

База такої динаміки — `dp[0] = 1`. Для того, щоб зробити перехід, необхідно перебрати вершину v, в якій буде записано значення $(m+1)$. Якщо для чергової вершини виконуються усі необхідні умови, необхідно зробити перехід `dp[mask | (1 <<̇ v)] += dp[mask]`. Після підрахунку усіх значень динаміки, відповіддю на задачу буде `dp[(1 <<̇ n) - 1]`, що відповідає повній масці з усіми обраними вершинами. Складність такого розв'язку — $O(2^n n^2)$ чи $O(2^n n)$ в залежності від реалізації.

Розв'язок — https://pastebin.com/tQYcGQwG

Блок 3

Нехай sz_v — кількість вершин у піддереві вершини v. Оскільки усі записані символи дорівнюють «<», то використовуючи транзитівність можна неважко довести, що значення у вершині повинно бути менше усіх значеннь вершин свого піддерева. Це означає, що значення 1 обов'язково повинно бути записано у корені дерева. Усі інші значення від 2 до n можна незалежно розподілити по піддеревам кореня дерева. Це можна зробити $cnt_v = \frac{sz_v!}{\prod_{to \in g_v} sz_{to}!}$ способами, де g_v — це список вершин to таких, що $p_{to} = v$. Після того, як ми визначали які числа будуть присутніми у кожному піддереві першої вершини, можна незалежно розв'язати задачу для кожного такого піддерева. Саме тому відповідь на задачу дорівнює $\prod_{v=1} cnt_v$. Складність такого розв'язку — $O(n)$ чи $O(n^2)$ в залежності від реалізації.

Розв'язок — https://pastebin.com/xsv6URJ3

Блок 4

Оскільки в даній групі усі $p_i = 1$, то усі некореневі вершини відрізняються між собою тільки символом c_v. Нехай x — кількість ребер, на яких записано символ «>», а $y = (n-1-x)$ — кількість ребер, на яких записано символ «<». Неважко побачити, що значення a_1 повинно дорівнювати $(x+1)$. Більш того, усі значення від 1 до x можуть бути записані в будь-якому порядку в x вершинах, на ребрах до яких написаний символ «>». Аналогічно усі значення від $(x+2)$ до n можуть бути записані в будь-якому порядку в y вершинах, на ребрах до яких написаний символ «<». Тому відповідь на задачу дорівнює $x!\,y!$ та може бути підрахована за $O(n)$.

Розв'язок — https://pastebin.com/apWgXHKH

Блок 5

У наступних двох групах для усіх вершин v виконується рівність $p_v = (v-1)$. Це означає, що дерево являє собою звичайний масив довжиною n, а символи c_v задають результат порівняння кожної пари сусідніх елементів масиву.

Спробуємо розв'язати задачу методом динамічного програмування. Нехай `dp[l][r]` — кількість способів розставити $(r-l+1)$ число на позиціях від l до r включно так, щоб виконувались усі необхідні відношення. База динаміки — `dp[l][r] = 1` для усіх пустих підвідрізків $(1 \leqslant l \leqslant n+1, r = l-1)$. Для того щоб підрахувати значення `dp[l][r]` для $1 \leqslant l \leqslant r \leqslant n$, переберемо позицію $i = l \dots r$, на якій буде знаходитися максимальне число серед елементів з l по r. Для такої позиції повинно виконуватися дві умови:

- $i = l$ або $c_i =$ «<». Дійсно, якщо ця умова не виконується, елемент на позиції $(i-1)$ буде більший ніж a_i, а тому a_i не може бути максимальним числом серед елементів з l по r.
- $i = r$ або $c_{i+1} =$ «>». Дійсно, якщо ця умова не виконується, елемент на позиції $(i+1)$ буде більший ніж a_i, а тому a_i не може бути максимальним числом серед елементів з l по r.

Якщо ж ці дві умови виконуються, необхідно розподілити $(r-l)$ чисел що залишилися на дві половини, і незалежно розв'язати дві підзадачі на менших підвідрізках. Розподілити числа на дві половини можна C_{r-l}^{i-l} способами, а тому до значення `dp[l][r]` необхідно додати значення `C[r - l][i - l] * dp[l][i - 1] * dp[i + 1][r]`. Тут C_n^k позначає кількість способів обрати k предметів з n. Ці значення можна підрахувати за $O(n^2)$ за допомогою рекурентної формули $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$ перед підрахунком основної динаміки. Після підрахунку усіх значень динамічного програмування, відповіддю на задачу буде значення `dp[1][n]`.

Оскільки всього підвідрізків масиву $O(n^2)$, та для кожного з них значення динаміки знаходиться за $O(n)$, загальна складність такого розв'язку — $O(n^3)$.

Розв'язок — https://pastebin.com/d9NsgVHA

Блок 6

Спробуємо розв'язати задачу за допомогою трохи іншої динаміки. Нехай `dp[r][x]` $(1 \leqslant r \leqslant n, 1 \leqslant x \leqslant r)$ — кількість способів розставити числа від 1 до r на перших r елементах масиву так, щоб виконувалися усі необхідні відношення, і значення a_r дорівнювало x.

База динаміки — `dp[1][1] = 1`. Подивимось, як можна підрахувати значення динаміки `dp[r][x]` при $r > 1$. Не обмежуючи загальності будемо вважати, що $c_r =$ «<». Оскільки повинно виконуватись відношення $a_{r-1} < a_r$, можемо перебрати, чому дорівнює $a_{r-1} = y = 1 \dots x-1$. Отримуємо, що $dp[r][x] = \sum_{y=1}^{x-1} dp[r-1][y]$.

Можна побачити, що $\sum_{y=1}^{x-1} dp[r-1][y] = dp[r-1][x-1] + \sum_{y=1}^{x-2} dp[r-1][y] = dp[r-1][x-1] + dp[r][x-1]$. Використовуючи цю формулу, можна підрахувати усі значення динаміки за $O(n^2)$. Після підрахунку усіх значень динамічного програмування, відповіддю на задачу буде $\sum_{i=1}^{n} dp[n][i]$.

Розв'язок — https://pastebin.com/Pw84hhBM

Блок 7

Спробуємо модифікувати розв'язок, описаний у попередньому пункті, для довільного дерева. Нехай `dp[v][x]` $(1 \leqslant v \leqslant n, 0 \leqslant x < sz_v)$ — кількість способів розставити числа від 1 до sz_v в усіх вершинах піддерева v так, щоб виконувалися усі необхідні відношення, а також існувало рівно x значень в піддереві v менших за a_v (іншими словами, $a_v = x+1$).

Будемо рахувати $dp[v]$ ітеративно, на кожній ітерації додаючи до вершини v його чергове піддерево to, та збільшуючи sz_v на sz_{to}. База динаміки — `dp[v][0] = 1, sz[v] = 1`. Для того щоб додати чергове піддерво to та перейти у наступний шар динаміки, переберемо x $(0 \leqslant x < sz_v)$ — поточну кількість вершин менших за a_v та cnt — кількість вершин у піддеріві to менших за a_v. Не обмежуючи загальності будемо вважати, що $c_{to} =$ «>». Оскільки a_v повинно бути більше ніж a_{to}, у піддереві to буде хоча б одна вершина менша ніж a_v, а тому $1 \leqslant cnt \leqslant sz_{to}$.

Тоді у новому шарі динаміки, кількість значень менших за a_v буде дорівнювати $(x+cnt)$. Кількість способів розподілити ці $(x+cnt)$ значень на вершини поточного піддерева та вершини піддерева to дорівнює C_{x+cnt}^{x}. У свою чергу, кількість значень більших за a_v дорівнює $(sz_v - 1 - x)$ у поточному

піддереві та $(sz_{to} - cnt)$ у піддереві вершини to. Загальна кількість значень більших за a_v дорівнює $(sz_v - 1 - x + sz_{to} - cnt)$, а тому кількість способів розподілити їх на два піддерева — $C^{sz_{to}-cnt}_{sz_v-1-x+sz_{to}-cnt}$. Оскільки $a_v > a_{to}$ та існує рівно cnt значень у піддереві to менших за a_v, то кількість значень у піддереві to менших за a_{to} повинна не перевищувати $cnt - 1$.

Тому можемо зробити перехід у наступний шар динаміки:

$$dp'[v][x + cnt] \mathrel{+}= C^{x}_{x+cnt} C^{sz_{to}-cnt}_{sz_v-1-x+sz_{to}-cnt} \sum_{y=0}^{cnt-1} dp[to][y]$$

.

Суму $\sum\limits_{y=0}^{cnt-1} dp[to][y]$ можна знаходити за $O(1)$ якщо підтримувати масив префіксних сум $sum[v][x] = \sum\limits_{y=0}^{x} dp[v][x]$. Цей масив можна дуже просто підтримувати за допомогою рекурентної формули $sum[v][x] = sum[v][x-1] + dp[v][x]$. Саме тому кожен перехід динаміки можна робити за $O(1)$.

Спробуємо оцінити складність описаного розв'язку. Для цього необхідно оцінити сумарну кількість операцій C наступного псевдокоду, де $g[v]$ — список усіх вершин to таких, що $p[to] = v$:

```
for (int v = n; v >= 1; --v) {
    sz[v] = 1;
    for (int to : g[v]) {
        for (int x = 0; x < sz[v]; ++x) {
            for (int y = 0; y < sz[to]; ++y) {
                ++C;
            }
        }
        sz[v] += sz[to];
    }
}
```

Оскільки сумарний розмір усіх $g[v]$ дорівнює $(n - 1)$ (кількість ребер дерева), перші два цикли сумарно виконаються за $O(n)$. Тому загальну кількість операцій C, на перший погляд, можна оцінити як $O(n^3)$, оскільки $sz[v]$ та $sz[to]$ завжди не більші n.

Але насправді C можна оцінити як $O(n^2)$. Дійсно, цикл по x можна розглядити як перегляд усіх вершин з піддерева v, що знаходяться лівіше піддерева to. Цикл по y можна розглядати як перегляд усіх вершин з піддерева to. Неважко побачити, що у такому випадку кожна пара вершин (x, y) буде розглядатися рівно один раз при $v = lca(x, y)$, оскільки при всіх більш високих вершинах v, вершини x та y вже будуть знаходитися в одному піддереві. Тут $lca(x, y)$ позначає найменшого спільного предка вершин x та y. Оскільки загальна кількіста пар вершин x, y дорівнює $\frac{n(n-1)}{2}$, сумарну кількість операцій C можна оцінити як $O(n^2)$.

Розв'язок — https://pastebin.com/Za5MtZ99

Якщо написати цей розв'язок неефективно (наприклад перебирати x не до поточного значення $sz[v]$, а до загального розміру піддерева), то рішення буде працювати за $O(n^3)$. Також за $O(n^3)$ буде працювати рішення, що не використовує масив префіксних сум, а рахує суму окремим циклом. Саме через наявність таких рішень була зроблена сьома група тестів з обмеженнями $n \leqslant 200$.

Задача D1. Зменшення масиву

Автори задачі: **Матвій Асландуков, Роман Білий та Антон Ципко**
Задачу підготував: **Матвій Асландуков**
Розбір написав: **Матвій Асландуков**

Спочатку необхідно окремо розглянути випадок, коли $k = 0$. У такому випадку ми не можемо зменшити жодне число, а тому якщо є хоча б одне число $a_i > 0$, то відповідь дорівнює -1. Інакше усі числа вже спочатку є недодатними, тому можно зробити 0 операцій.

Тепер розглянемо послідовно рішення на кожну групу тестів для $k > 0$:

Блок 1

У першій групі значення $t = 0$, а тому за одну операцію можна просто зменшити будь-яке число k. Для того, щоб зробити значення a_i недодатним, необхідно виконати з ним хоча б $\left\lceil \frac{max(0,a_i)}{k} \right\rceil$ операцій. А тому мінімальна кількість операцій, за яку можна зробити усі числа недодатними, дорівнює $\sum_{i=1}^{n} \left\lceil \frac{max(0,a_i)}{k} \right\rceil$.

Розв'язок: https://pastebin.com/9H9Z2txE

Блок 2

Для розв'язання цієї групи потрібно було визначити критерій того, що можливо зробити усі числа недодатніми рівно за x операцій. Для цього спробуємо звести задачу до випадку $t = 0$. А саме, скажемо, що за одну операцію ми збільшуємо усі n чисел на t, та зменшуємо одне число a_i на $(k + t)$. Неважко побачити, що така операція еквівалентна операції з умови. З таким визначенням можна сказать, що після виконання x операцій усі a_i збільшаться на xt. Після цього для того, щоб робити число $(a_i + xt)$ недодатним, потрібно виконати з ним хоча б $\left\lceil \frac{max(0,a_i+xt)}{k+t} \right\rceil$ операцій. А тому отримуємо доволі простий критерій: усі числа можна зробити недодатними рівно за x операцій тоді і тільки тоді, коли $\sum_{i=1}^{n} \left\lceil \frac{max(0,a_i+xt)}{k+t} \right\rceil \leqslant x$.

Маючи цей критерій може здатися, що можна зробити звичайний бінарний пошук по відповіді. Але таке рішення є неправильним, оскільки не завжди існує відповідь, що використовує рівно $(x+1)$ операцій навіть тоді коли, існує відповідь, що використовує рівно x операцій. Наприклад на тесті $n = 2, k = 100, t = 99, a_1 = 1, a_2 = 1$, можна зробити усі числа недодатними за 2 операції, але не можна за 3.

Можна довести, що коли відповідь існує, вона не перевищує nm, де $m = \max(0, \max_{i=1}^{n} a_i)$. А тому можна перевірити усі можливі варіанти відповіді від 0 до nm, використовуючи описаний вище критерій. Для перевірки критерію необхідно зробити n операцій, а тому загальна складність такого рішення — $O(n^2 m)$.

Розв'язок: https://pastebin.com/E2Q58fqX

Блок 3

Будемо підтримувати поточну відповідь x, яка спочатку дорівнює нулю, а також масив cnt, де cnt_i позначає кількість операцій, зроблених з позицією i. На черговій ітерації алгоритму переберемо усі позиції $i = 1 \dots n$. Якщо для чергової позиції виконується нерівність $\left\lceil \frac{max(0,a_i+xt)}{k+t} \right\rceil > cnt_i$, необхідно збільшити cnt_i та x на $(\left\lceil \frac{max(0,a_i+xt)}{k+t} \right\rceil - cnt_i)$. Якщо на черговій ітерації алгоритму значення x не збільшилося, відповідь була знайдена. Якщо ж x стало більше ніж nm, необхідно вивести -1. Можна довести, що таке рішення працює за $O(nm)$. Також можна помітити, що таке рішення зробить рівно одну ітерацію на тестах першої групи і спрацює за $O(n)$.

Розв'язок: https://pastebin.com/f0risSD7

Блок 4

Для початку зрозуміємо, як допомогає обмеження $1 \leqslant a_i$. По-перше, трохи спрощується критерій перевірки, описаний у другому пункті: $\sum\limits_{i=1}^{n} \left\lceil \frac{a_i+xt}{k+t} \right\rceil \leqslant x$. По-друге, можна помітити, що коли усі $a_i \geqslant 1$, з кожною позицією i необхідно буде зробити хоча б одну операцію. Це означає, що у випадку, коли $k \leqslant (n-1)t$, відповідь буде -1. Дійсно, припустимо що існує якась оптимальна відповідь. Оскільки с кожною позицією була виконана хоч одна операція, можемо зменшити кількість операцій з кожною позицією на 1. Відповідь залишиться валідною, оскільки кожне a_i не збільшиться. Тому ми отримали рішення з меншою кількістю операцій, що протиречить припущенню оптимальності відповіді.

Тепер розглянемо випадок коли $k > (n-1)t$. Припустимо, що рівно за x операцій можна зробити усі числа недодатними. Тоді неважко побачити, що за $(x+n)$ операцій усі числа також можна зробити недодатними. Дійсно, можна просто зробити ще по одній операції з кожною позицією i, після чого значення на усіх позиціях зменшаться на $(k-(n-1)t)$. Саме тому можна перебрати остачу від відповіді при діленні на n, і зробити бінарний пошук на числах x з такою остачею. Складність такого рішення $O(n^2 \log ans)$.

Розв'язок: https://pastebin.com/kgUTa61U

Блок 5

На відміну від попереднього рішення, будемо перебирати остачі rem $(0 \leqslant rem < n)$ від відповіді при діленні на n у випадковому порядку. Також будемо підтримувати мінімальну знайдену відповідь x серед усіх остач, що були розглянути попередньо. Тоді перед запуском бінарного пошуку на черговій остачі перевірими, чи покращуються взагалі відповідь. Для цього необхідно перевірити описаний раніше критерій на максимальному числі меншим за x з остачею rem. Тільки у випадку, коли критерій виконується, треба запускати бінарний пошук. Можна довести, у такому рішенні бінарний пошук буде запускатися $O(\log n)$ разів, а тому загальна складність — $O(n^2 + n \log n \log ans)$.

Розв'язок: https://pastebin.com/D8KpR3ZX

Блок 6

Для прискорення попереднього рішення навчимося рахувати суму $\sum\limits_{i=1}^{n} \left\lceil \frac{a_i+xt}{k+t} \right\rceil$ швидше ніж за $O(n)$.

Для цього помітимо, що $\sum\limits_{i=1}^{n} \left\lceil \frac{a_i+xt}{k+t} \right\rceil = \sum\limits_{i=1}^{n} \left\lfloor \frac{a_i}{k+t} \right\rfloor + \sum\limits_{i=1}^{n} \left\lfloor \frac{xt}{k+t} \right\rfloor + \sum\limits_{i=1}^{n} \left\lceil \frac{xt \mod (k+t) + a_i \mod (k+t)}{k+t} \right\rceil = A+B+C$. Значення A не залежить від x, а тому його можна підрахувати один раз до виконання всіх перевірок. Значення B можна підрахувати як $B = n \cdot \left\lfloor \frac{xt}{k+t} \right\rfloor$. Нехай $xt \mod (k+t) = r$, та $a_i \mod (k+t) = b_i$. Тоді неважко побачити, що значення $\left\lceil \frac{r+b_i}{k+t} \right\rceil$ буде дорівнювати 0 у випадку, коли $r = b_i = 0$, дорівнювати 1, коли $1 \leqslant r + b_i \leqslant (k+t)$, та 2 інакше. Тому значення C можна підрахувати як кількість чисел у масиві b, що більше або дорівнюють $(1-r)$ плюс кількість чисел у масиві b, що більше $(k+t-r)$. Ці кількості можна знайти за допомогою бінарного пошуку, якщо попередньо відсортувати масив b.

Таким чином, ми навчилися перевіряти критерій за $O(\log n)$, що набагато швидше $O(n)$. Загальна складність рішення — $O(n \log n \log ans)$.

Розв'язок: https://pastebin.com/9wHi9fdg

Блок 7

Для прискорення попереднього рішення достатньо використати ідею, що була описана у п'ятому пункті. А саме необхідно запускати бінарний пошук тільки у випадку, коли відповідь для чергової остачі покращується. Складність такого рішення — $O(n \log n + \log^2 n \log ans)$.

Розв'язок: https://pastebin.com/N7yFLqYE

Блок 8

Для зручності будемо вважати, що усі числа a_i відсортовані за спаданням. В останніх чотирьох блоках числа a_i могли бути від'ємними, але насправді це не дуже сильно ускладнює задачу. Дійсно, позначимо через n' — максимальне число не більше за n таке, що $k > (n'-1)t$. Тоді неважко довести, що хоча б одна операція могла бути виконана максимум з n' позиціями (можна провести доведення, аналогічне доведенню з пункту 4). Це означає, що з останніми $(n-n')$ позиціями не було проведено жодної операції.

Тоді для перших n' чисел можна розв'язати задачу так само, як і у групі 4, а далі необхідно буде перевірити, що останні $(n-n')$ чисел залишаться недодатними після виконання знайденної кількості операцій.

Розв'язок: https://pastebin.com/WXaDSk8n

Блок 9

Для прискорення попереднього рішення достатньо використати ідею, що була описана у п'ятому пункті. Складність такого рішення — $O(n^2 + n \log n \log ans)$.

Розв'язок: https://pastebin.com/Eh5cfipg

Блок 10

Для прискорення квадратичного розв'язку необхідно навчитися швидко перевіряти критерій, описаній в пункті 2. Відмінність від розв'язку шостої групи полягає в тому, що a_i можуть бути менше нуля, а тому деякі a_i не повинні додавати від'ємні значення $\left\lceil \frac{a_i+xt}{k+t} \right\rceil$ до суми.

Оскільки усі значення a_i відсортовані за спаданням, то в першу чергу можна знайти максимальну позицію i таку, що $a_i + xt \geqslant 0$. Для усіх наступних позицій внесок в шукану суму буде дорівнюавти нулю, а тому про ці позиції можна просто забути. Для знайденого префіксу позицій $1 \dots i$ можна використовувати ідею з пункту 6, але для цього потрібно навчитися швидко рахувати кількість чисел більших даного на префіксі. Це можна зробити за допомогою техніки «часткове каскадування». Ця техніка дозволяє відповідати не необхідні запити за той самий час $O(\log n)$ та потребує $O(n \log n)$ часу та пам'яті на побудову. З її допомогою можна розв'язати задачу за час $O(n \log n + n \log n \log ans)$.

Розв'язок: https://pastebin.com/U4CQse1c

Блок 11

Для прискорення попереднього рішення достатньо використати ідею, що була описана у п'ятому пункті. Складність такого рішення — $O(n \log n + \log^2 n \log ans)$.

Розв'язок: https://pastebin.com/kjFkP61L

У цієї задачі також існує інше, більш просте, рішення, що було знайдено під час тестування задачі Соболєвим Євгеном. Основна ідея полягає в тому, що можна робити звичайний бінарний пошук по відповіді, але перевіряти, що відповідь уснує не у точці mid, а у вікні розміром n на інтервалі $[mid; mid + n)$. Для того, щоб швидко перевіряти критерій, описаний раніше у пункті 2, можна спочатку за час $O(n)$ знайти значення суми при $x = mid$. Для того, щоб перараховувти значення суми при збільшенні x на 1, можна було помітити, що кожен доданок суми може збільшитися максимум один раз на 1. Дійсно, при збільшенні x на 1, чисельник дробу збільшується на t, а знаменник рівен $(k + t)$. Оскільки виконується рівність $k > (n-1)t$, то $k + t > (n-1)t + t = nt$. Саме тому при збільшенні на t лише n разів, значення дробу може збільшитися максимум на 1. Момент часу, у який це значення збільшиться, можна знайти за формулою, і у відповідний момент часу збільшити поточне значення суми на 1. Складність такого рішення — $O(n \log ans)$.

Розв'язок: https://pastebin.com/DTmqmKBM

Зауваження та цікаві факти:

- При реалізації рішення потрібно дуже уважно слідкувати за операціями множення, щоб випадково не вийти за межі 64-бітного типу даних. Найбільш небезпечне місце, де може переповнитися 64-бітний тип — це множення x на t у функції перевірки (наприклад, якщо виставляти межі бінарного пошуку рівними 10^{15}). Але можно помітити, що оскільки

$k > (n-1)t$, то $nt \leqslant 2k = 2 \cdot 10^9$. Оскільки відповідь не перевищує nm ($x \leqslant nm$), то $xt \leqslant ntm \leqslant 2 \cdot 10^9 \cdot 10^9 = 2 \cdot 10^{18}$. Це означає, що у якості правої межі бінарного пошуку можна виставити значення $\frac{2 \cdot 10^{18}}{t}$ та більше не думати про переповнення.

- Спочатку ця задача була запропонована на позицію «`A`» з неправильним авторським рішенням, описаним у пункті 2 (звичайним бінарним пошуком по відповіді). Після знаходження правильного розв'язку складність задачі різко зросла, і задача пересунулася на останню позицію.

Задача D2. Додай ребра

Автори задачі: **Ілля Коваль, Даніїл Осташев та Антон Ципко**
Задачу підготували: **Ілля Коваль та Даніїл Осташев**
Розбір написав: **Ілля Коваль**

Для того, щоб розв'язати цю задачу, необхідно зрозуміти, куди додавати ребра, та які ціни їм надавати. Для того, щоб максимізувати наш результат, будемо з'єднувати вершини, які знаходяться близько у нашому дереві. Нам потрібно додати велику кількість ребер, тому в новому графі у деяких вершин буде багато сусідів. Якщо ми будемо шукати відстані від деякої такої вершини, то нам потрібно, щоб множини сусідів сусідів нашої вершини сильно перетинались, інакше від нашої вершини можна буде за декілька ребер прийти до будь-якої вершини графа. Звідси, можна вибрати декілька вершин та з'єднати їх всіх між собою, отримавши те, що зветься клікою.

У ці кліки будемо додавати вершини, які з'єднані між собою в дереві та відокремлені від інших вершин, щоб додавання цих ребер не сильно змінило відстані між іншими вершинами. Тому, можна брати піддерева та робити з них кліки. Розмір цих клік не може бути більше, ніж $k + 1$, щоб ми не додали дуже багато ребер до вершин. Тому, можливий такий алгоритм для додавання ребер:

Знаходимо розміри всіх піддерев нашого дерева, та шукаємо найбільшу кліку, яку можна зробити (Якщо у нас мало додаткових ребер, то можна взяти такий розмір піддерева, щоб зробити майже кліку та використати всі ребра). Після цього додаємо необхідні ребра та забуваємо про ці вершини, та продовжуємо розбиття.

Тепер поставимо ваги на всі ребра. В кліці дуже багато ребер, тому відстані між вершинами кліки будуть дуже маленькими (якщо якесь ребро у кліці дороге, то є безліч можливостей його обійти). Тому, можна зробити всі ребра у кліці дуже дешевими. Інших ребер у графі буде дуже мало, тому що доданих ребер значно більше, ніж початкових. Тому можна зробити інші ребра найдорожчими у графі. Додатково, у клік є межові вершини — ті, які мають ребра до вершин не з кліки. Їх не дуже багато, та всі мінімальні шляхи, які йдуть через кліку, або між вершиною кліки та іншими вершинами, проходять через ці вершини. Тому, можна зробити ребра між вершинами межі та іншими вершинами кліки дорогими, тому що багато шляхів проходить через них. Також можна не додавати ребра між вершинами межи, щоб шляхи через кліку два рази йшли дорогими ребрами кліки, а не один, як раніше.

Для подальшого покращення результатів можна модифікувати алгоритм пошуку піддерева для перетворення на кліку шляхом пошуку того, наскільки це зменшує відстані між вершинами графа, чи шляхом перебору.