Задача A. Красиві сторінки

Наївне рішення — обійти всі числа від 1 до N і перевірити для кожного числа, чи складається воно з однакових цифр.

Можливим рішенням для максимальної кількості балів є побудова в циклі всіх чисел типу 1, 11, 111 тощо, поки ми не перевищимо N. Потім аналогічно побудувати числа 2, 22, 222, ... і так далі.

Під час реалізації слід враховувати, що при $N = 10^{18}$ можливе переповнення типу символу long long – якщо число 18 дев'яток ще менше N, то число 19 дев'яток вже дає переповнення. Тому необхідно використовувати тип unsigned long long unsigned.

Задача B. Естетична краса

Це завдання було поставлено багато років тому, а обмеження на кількість дерев N, за спогадами, становило до 1000.

Тоді рішенням була динамічна оптимізація:

Знаходимо зліва направо для кожного дерева, якої довжини найдовший ряд зліва від нього, для якого він найвищий.

Якщо висоти h [i], а довжина наведеної вище послідовності позначена L [i], то алгоритм такий:

```
FOR J=1 TO I-1
        IF h[i]>h[J]
                IF L[I]<L[J]+1
                        L[I]=L[J]+1
```

Складність N*N / 2, тобто. O (N^2).

Аналогічно ми шукаємо зростаючий порядок справа наліво в масиві L [i]:

```
FOR J=I+1 TO N
        IF h[i]>h[J]
                IF R[I]<R[J]+1
                        R[I]=R[J]+1
```

Відповідь: max (L [i], R [i], L [i] + R [i] -1) для кожного i: 1≤i≤N.

Рішення складності O (NlogN) використовує двійкове дерево. Ми подивимося на дерево Фенвіка. Ми знаємо, що він повертає максимум в інтервалі [0; K], тобто. якщо змінити значення масиву, то в будь-який момент ми зможемо відповісти, який є найбільшим елементом від першого до K.

Це саме те, що нам потрібно в задачі.

Нехай висоти дерев становлять h [1], h [2],…, h [N], йдіть ліворуч і праворуч, і ми досягли h [K]. Нас цікавить максимальна вихідна послідовність, що закінчується на число менше K.

Для ілюстрації ми скористаємося наступним прикладом:

Дано наступне:3 1 4 2 6 3 7 5 8 6

У верхньому рядку таблиці розташовані індекси, тобто. числа від 1 до максимальної висоти дерева, яка з меж завдання до 100000.

У другому рядку знаходяться значення масиву дерева Фенвіка - під числом I запишемо довжину максимального зростаючого рядка, який закінчується на I.

1	2	3	4	5	6	7	8	9	……	100000
									……	

Читаємо першу цифру 3 /синім кольором/. З дерева Фенвіка запитуємо, що є максимумом в інтервалі [0; 2] / в жовтому /. Це 0, отже, не існує зростаючої послідовності, яка закінчується на 3. Запишемо під 3 суму 0 + 1 = 1.

3 1 4 2 6 3 7 5 8 6

1	2	3	4	5	6	7	8	9
		1						

Аналогічно для другого числа 1: у [0; 0] максимум дорівнює 0, нижче 1 пишемо число 0 + 1 = 1.

3 **1** 4 2 6 3 7 5 8 6

1	2	3	4	5	6	7	8	9
1		1						

Для третього числа 4 з дерева Фенвіка беремо те, що є максимумом в інтервалі [0; 3], щоб позначити його як max [0; 3]. Відповідь 1, що означає, що перед 4 стоїть послідовність з 1 числа. Тому для 4 заповнюємо 2.

3 1 **4** 2 6 3 7 5 8 6

1	2	3	4	5	6	7	8	9
1		1	2					

Для числа 2: до 2 ми маємо max [0; 1] = 1, тому під 2 пишемо 2:

3 1 4 **2** 6 3 7 5 8 6

1	2	3	4	5	6	7	8	9
1	2	1	2					

Тоді число 6. max [1; 5] = 2, збільшити на 1 і записати результат 3 під шісткою:

3 1 4 2 **6** 3 7 5 8 6

1	2	3	4	5	6	7	8	9
1	2	1	2		3			

Число 3 в порядку: max [0; 2] = 2, ми пишемо під 3 число 2 + 1 = 3:

3 1 4 2 6 **3** 7 5 8 6

1	2	3	4	5	6	7	8	9
1	2	3	2		3			

Для наступного числа 7: до 6 максимум - 3, тому нижче 7 - це 4:

3 1 4 2 6 3 **7** 5 8 6

1	2	3	4	5	6	7	8	9
1	2	3	2		3	4		

Число 5 наступне, аналогічно в інтервалі [1; 4] від Фенвіка ми отримаємо максимум 3, а нижче 5 буде 3 + 1 = 4.

3 1 4 2 6 3 7 **5** 8 6

1	2	3	4	5	6	7	8	9
1	2	3	2	4	3	4		

Для передостаннього числа 8: max [1; 7] = 4, то нижче 8 дорівнює 5:

3 1 4 2 6 3 7 5 **8** 6

1	2	3	4	5	6	7	8	9
1	2	3	2	4	3	4	5	

І для останнього числа 6: у [1; 5] максимум 4, нижче 6 змінюємо на 5.

3 1 4 2 6 3 7 5 8 **6**

1	2	3	4	5	6	7	8	9
1	2	3	2	4	5	4	5	

Позначимо через L [i] масив з другого рядка таблиці. Це відповіді на зростаючу кількість зліва направо.

Знайдемо аналогічний R [i] - висоти для зростаючого ряду справа наліво.

Тоді відповідь на задачу макс (max (L [i]), max (R [i]), max (L [i [+ R] [i] -1)),

У estet1.cpp є розв'язок на 100 балів, а в estet2.cpp - рішення зі складністю O (N2).

Задача C. Зима прийшла, коли не чекали

Зрозуміло, що маємо неорієнтований граф з N вершин і M ребрами між ними. Ми шукаємо декілька різних невпорядкованих пар вершин, які залишаються зв’язаними при видаленні будь-якого ребра.

Перше підзавдання – 30 балів. Ми робимо безпосередньо, як описано в умові. Для кожної пари вершин перевіряємо, чи після видалення ребра воно залишиться незв’язаним. Складність е $O(\boldsymbol{N^2 * M * (N + M)})$.

Друге підзавдання – на 20 балів. Тут нам потрібно зробити наведене вище рішення трохи більш оптимальним. Легко помітити, що немає сенсу видаляти кожне ребро, перевіряючи кожну пару, але ми можемо видалити ребро і побачити, які пари стають непов’язаними. Тому спочатку знімаємо ребро, дивимося, які компоненти зв’язності залишаються і для кожні дві вершини різних компонентів зазначаємо, що їх немає в остаточній відповіді. Безпосередня реалізація цієї ідеї складна $O(\boldsymbol{M * (N + M + N^2)})$.

Третє підзавдання – на 20 балів. Тепер ми повинні перейти до основної думки. Зрозуміло, що нас цікавлять лише ребра, у яких граф розпадається на більше ніж одну зв’язну складову. Досвідчені учасники повинні знати, що такі ребра називають мостами (або

артикуляційними ребрами). Для наступних розв'язків ми спочатку з'ясовуємо, які є мостами відліку. Наведене вище рішення можна оптимізувати, перевіряючи при знятті лише мостів (легко помітити, що їх не більше N-1), але це не підзавдання, тому що це не особливе поліпшення. Ми можемо зробити наступне. З кожної вершини пускаємо обхід. Пограємо з вершиною з номером $\boldsymbol{x}$. Під час руху по ребрах, які не є мостами, вершини, яких ми досягаємо, розглядаються як пара з x, є частиною відповіді. А коли ми досягаємо мосту, ми знаємо, що вершини внизу залишаються нез'єднаними при знятті мосту і ми їх не будемо рахувати, тобто. ми можемо перестати повзати вниз. Нарешті, нам доведеться розділити отримане число на 2, оскільки кожну пару ми порахували двічі. Тут рішення буде трохи складніше, ніж відповідь або в гіршому випадку: $O(\boldsymbol{N} * (\boldsymbol{N} + \boldsymbol{M}))$.

Четверте підзавдання – на 30 балів. Повне рішення. Давайте подивимося, що станеться з графом, видаливши всі мости з графа відразу. Він розпадається на деякі пов'язані компоненти. Ми бачимо, що якщо у відповіді є пара вершин, то вони будуть в одній складовій зв'язку і навпаки. Таким чином, як тільки ми знайдемо мости, достатньо виконати обхід без них і підрахувати кількість вершин кожної з'єднаної компонент. Відповідно, якщо з'єднаний компонент має k вершини, він додасть до відповіді $\binom{k}{2}$ – кожна з кожною вершиною. Тут складність вже є лінійною на графі, а саме $O(\boldsymbol{N} + \boldsymbol{M})$.

Задача D. Красиві підрядки

Перше підзавдання – на 11 балів. Тут достатньо повного перебору - ми робимо два вкладених цикли для обходу всіх підрядків і перевіряємо для кожного, чи немає іншого підрядка, розташованого ліворуч, який такий самий і якщо є, то цей вже зараховується до відповіді і вдруге не рахувати. Складність $O(\boldsymbol{N}^4)$, але насправді вона набагато менша, тому що не може бути $\boldsymbol{N}^2$ різних підмножин для більшої довжини, і перевірка для порівняння підмножин буде постійною більшу частину часу.

Друге підзавдання — на 13 балів. Тепер ми трохи оптимізуємо цю ідею. Ми можемо застосувати хешування - знайти хеш-значення кожного підрядка і помістити всі значення в один *unordered_set*. Отже, його розмір буде кількістю різних підмножин. Складність$O(\boldsymbol{N}^2)$.

Третє підзавдання – 10 балів. Тепер ми можемо мати тут запити. Ми знову використовуємо наведене вище рішення, оновлюючи *unordered_set* хешами для кожного запиту. Складність $O(\boldsymbol{N}^2 + \boldsymbol{Q} * (\boldsymbol{N} + \boldsymbol{Q}))$.

Четверте підзавдання – на 12 балів. Це відносно відоме завдання, для якого ми вже повинні використовувати суфіксну структуру. Для кожного запиту давайте знайдемо масив суфіксів поточного рядка. Тоді ми можемо знайти відповідь наступним чином. Перебираємо суфікси в упорядкованій формі. Спочатку ми додаємо кількість літер до першого суфікса (так ми підраховуємо всі підрядки, які є його префіксами). Потім, оскільки ми знаходимося на другому суфіксі, ми знову додаємо його довжину. Але ми, можливо, підрахували деякі підмножини кілька разів. Це загальні префікси першого та другого підрядків. Щоб знайти повтори, ми використовуємо функцію *lcp* (longest common prefix) для цих двох суфіксів. З відповіді беремо результат функції і продовжуємо таким же чином обходити впорядковані суфікси. Зрозуміло, що таким чином, оскільки суфікси розташовані, ми будемо рахувати всі різні підмножини, оскільки в цій постанові сусіди даного суфікса є найближчими лексикографічно суфіксами. Це підзавдання не дає більше балів, оскільки це відносно стандартне завдання. Складність е $O(\boldsymbol{Q} * (\boldsymbol{N} + \boldsymbol{Q}) * \log_2(\boldsymbol{N} + \boldsymbol{Q}))$ – ми маємо $\boldsymbol{N} + \boldsymbol{Q}$, ому що якщо всі операції виконуються шляхом додавання, то довжина рядка тільки збільшується.

П’яте підзавдання – на 15 балів. Тут ми зробимо важливий крок до повного рішення. Ми оптимізуємо наведену вище ідею (зрозуміло, що в попередньому підзавданні ми маємо найтривіальніший спосіб вирішення, оскільки ми щоразу генеруємо масив суфіксів рядка). Давайте розглянемо, як змінюється масив суфіксів у запитах. Якщо ми видалимо передню частину, ми просто видалимо один суфікс (поки що весь рядок) і загалом суфікси залишаться такими ж. Це не так з додаванням ззаду. Коли ми додаємо букву, то разом з нею збільшуються всі суфікси, і відповідно ми можемо мати зсуви (для тих суфіксів, які є сусідніми префіксами), які важко передбачити. Тому ми зробимо стандартний трюк, скориставшись тим, що запити офлайн. Ми розглянемо їх у зворотному порядку, тобто. спочатку ми читаємо їх і виконуємо операції з рядком, і таким чином отримуємо остаточний рядок. Далі, озираючись назад, ми маємо такі операції - операція додавання виконується шляхом видалення задньої частини, а операція видалення виконується шляхом додавання передньої частини. Знову ж таки, зрозуміло, що у нас немає проблем з операцією додавання на фронті. Питання в тому, що робити з іншим. Вона знову змінила всі суфікси, прибравши одну букву зі зворотного боку. Знову ж таки, ми можемо внести зміни в постанову щодо суфіксів, які є префіксом сусіда. Але тут є щось дуже важливе — у цих операціях суфікси лише зменшуються в довжині. І якщо у нас є суфікс, який є префіксом сусіда, він не впливає на відповідь поточного алгоритму. Це означає, що ми можемо просто видалити будь-який суфікс, який стає префіксом сусіда. Таким чином у нас не буде змін в постанові під час операції з видалення! Ми можемо виконати наступний алгоритм - у булевому масиві ми зберігаємо в порядку суфікси, які все ще працюють для нас. Маючи операцію додавання, ми поміщаємо одиницю в цей масив у відповідну позицію, а також скануємо всі суфікси, щоб побачити, чи є який-небудь суфікс, який став префіксом сусіда під час додавання. Під час операції видалення ми ставимо нуль у булевий масив у відповідну позицію і знову обходимо масив, щоб побачити, чи відсутні нові суфікси. Так само, як ми бачимо, чи рядок не є префіксом сусіда, ми порівнюємо довжини та знову використовуємо функцію *lcp*. Оскільки ми хочемо оптимізувати *log* множник, нам потрібно використовувати те, що *lcp* двох суфіксів є запитом *RMQ* в масиві *lcp* для рядка. Таким чином, ми можемо обійтися з *sparse table RMQ* з постійним запитом. Тут складність $O(\boldsymbol{Q} * (\boldsymbol{N} + \boldsymbol{Q}))$.

Шосте підзавдання – на 13 балів. Ми виконуємо лише операції з видалення. Знову ж таки, ми використовуємо прийом, щоб розглядати запити у зворотному порядку. Таким чином, ці операції здійснюються лише шляхом додавання. Як ми зазначали, вони просто додають новий суфікс до масиву суфіксів. Оскільки в розпорядженні немає жодних змін, крім додавання нового суфікса, нам не потрібно стежити за тим, чи стане суфікс префіксом сусіда. Тож ми можемо просто використовувати set, щоб зберегти номери суфіксів для поточного рядка. Потім, додавши новий суфікс, ми дивимося на двох сусідів (або тільки на одного, якщо він на одному кінці) і відповідно змінюємо відповідь. Таким чином, складність $O(\boldsymbol{Q} * \log_2(\boldsymbol{N} + \boldsymbol{Q}))$.

Сьоме підзавдання — на 15 балів. Тут ми повинні спробувати краще реалізувати ідею, про яку ми говорили в четвертому підзавданні. Там ми щоразу проповзали поточні суфікси. Знову ж таки, ми будемо використовувати *set* для збереження чисел (звичайно, номерів у відсортованих розпорядженнях) суфіксів для поточного рядка. Якось доводиться постійно стежити, чи не стає суфікс префіксом сусіда. Тому ми будемо використовувати *multiset*, в якому будемо зберігати впорядковану пару, яка є відношенням між суфіксом і сусідом. Перше число буде числом символів суфікса без спільної частини з його сусідом, а друге — номером суфікса (тобто ми можемо відстежувати, який суфікс випадає). Отже, якщо перше число впорядкованої пари стає 0, це означає, що підрядок став сусіднім префіксом, і нам потрібно його видалити. Друге число буде позицією суфікса в масиві суфіксів, це нам потрібно для того,

щоб мати можливість видалити більше одного суфікса, який буде впорядкований, а не хаотично, що створить проблеми. Ми можемо створити функцію, яка бачить, чи перший елемент *multiset-a* не має першого числа 0, і якщо є, то деякий суфікс відкидається, і він обертається до тих пір, поки не з'явиться більше число або не закінчаться суфікси (видаляючи всі літери, наприклад). Відкинути суфікс означає перерахувати відповідь, щоб побачити, хто є його сусідами. Тепер розглянемо операції. Маючи операцію додавання, нам потрібно додати відповідний суфікс до *set*-а суфіксів, а потім викликати функцію, яка перевіряє суфікси, які є сусідніми префіксами. І оскільки у нас є операція видалення, нам потрібно зменшити всі числа в *multiset-i* на одиницю і подивитися, що вийде. Тут може здатися, що ми зробимо кілька ходів, і це так. Але якщо ми дивимося на амортизований, то кожен суфікс, ми дивимося на нього двічі – при додаванні і вилученні, тобто. постійна операція амортизується (за журналом, через роботу з набором) Зрозуміло, що це непросто, тому у нас буде просто змінна, яка контролює кількість видалених літер. Щоб це спрацювало, ми помістимо довжину *lcp* в *multiset*, дивлячись на суфікси так, ніби вони були розширені до кінця рядка (тобто, ніби ми не махали літерами). Алгоритм, описаний таким чином, має складність $O(\boldsymbol{Q} * \log_2(\boldsymbol{N} + \boldsymbol{Q}))$, але з досить великою константою для кожної операції.

Останнє підзавдання – на 11 балів. Тут ми лише покращимо константу. Замість *set*-а суфіксів ми будемо використовувати дерево Фенвіка (це як оновити ідею за допомогою булевого масиву). Для того, щоб знайти сусідів суфікса в дереві Фенвіка, ми повинні вміти знайти найближчі ліворуч і праворуч для даної одиниці. Ми можемо зробити це за допомогою журналу ^ 2 кроки з легким двійковим пошуком, але тут ми будемо використовувати кращу ідею - схожу на *binary lifting*. Давайте знайдемо найменший префікс із сумою $\boldsymbol{s}$ (яка така ж, як у позиції s, яка є суфіксом). Починаємо з початку масиву і з найбільшого степеня, який дорівнює $\leq \boldsymbol{N}+\boldsymbol{Q}$. Перевіряємо, чи не отримаємо, додавши його, суму більшу за $\boldsymbol{s}$, якщо так, то спробуємо в меншій мірі і повторимо цей крок. Цей вид просто перевіряє число в масиві, яке є деревом Фенвіка з позицією, зміщеною на ступінь пари (це пояснюється специфічним способом, яким кожна позиція відповідає заданому інтервалу). Звичайно, тут ми будемо використовувати *RMQ* з постійним запитом, щоб знайти *lcp* двох суфіксів, які нам знадобляться кілька разів. Інша річ, яку ми оптимізуємо, це *multiset*. Знову з деревом Фенвіка (адже ми маємо чистий логарифм для кожної операції). Перше число впорядкованої пари тепер буде індексом у масиві, і кожен елемент буде *multiset*, але набагато меншим, ніж раніше. Не кожен елемент може бути векторним, наприклад, тому що нам потрібно видалити суфікси по порядку. Знову ж таки, ми скористаємося трюком для *lower_bound* в дереві Фенвіка, і тут нам потрібно буде лише знайти суму 1 і перевірити, чи індекс не менше видалених літер. За допомогою цих оптимізацій, про які ми згадували, ми значно зменшуємо константу і, звичайно, знову ж таки складність $O(\boldsymbol{Q} * \log_2(\boldsymbol{N} + \boldsymbol{Q}))$.