Задача 1. Выбор зала

Пусть длина меньшей стороны равна x, а большей — y. Тогда заметим, что должны выполняться следующие ограничения:

$y \geq x$

$xy \geq A$, следовательно $y \geq A / x$

$2(x + y) \geq C$, следовательно $y \geq C / 2 - x$

Таким образом, минимальное подходящее значение y равно

$ymin = \max(x, A / x, C / 2 - x)$,

причем последние два значения необходимо округлить вверх.

$xy \leq B$, следовательно $y \leq B / x$

$2(x + y) \leq D$, следовательно $y \leq D / 2 - x$

Таким образом, максимальное подходящее значение y равно

$ymax = \min(B / x, D / 2 - x)$,

причем на этот раз значения необходимо округлить вниз.

Переберем меньшую сторону зала, заметим, что перебор можно проводить до квадратного корня из B. Для каждого значения x вычислим минимальное и максимальное значение y и прибавим к ответу $\max(0, ymax - ymin + 1)$.

Приведем соответствующий код на паскале.

```
read(a, b, c, d);
ans := 0;
for x := 1 to d div 2 do begin
    if x * x > b then
        break;
    miny := x;
    if (c + 1) div 2 - x > miny then
        miny := (c + 1) div 2 - x;
    if (a + x - 1) div x > miny then
        miny := (a + x - 1) div x;
    maxy := d div 2 - x;
    if b div x < maxy then
        maxy := b div x;
    if maxy >= miny then
        ans := ans + (maxy - miny + 1);
end;
writeln(ans);
```

Аналогичный подход, но с более простыми формулами, получается, если применить следующий прием. Откажемся от нижних ограничений и решим задачу для числа залов с верхней границей на площадь B и верхней границей на периметр D. Обозначим число таких залов как f(B, D). Теперь, чтобы получить правильное число залов, можно применить формулу включения-исключения.

$ans = f(B, D) - f(B, C - 1) - f(A - 1, D) + f(A - 1, C - 1)$

Подзадача 1 (50 баллов)

$1 \leq A \leq B \leq 1000$, $4 \leq C \leq D \leq 1000$.

Баллы за подзадачу начисляются только в случае, если все тесты успешно пройдены.

В этой подзадаче 20 тестов, они находятся в каталоге tests/subtask1 в файлах 01 - 20

При решении этой подзадачи можно перебирать обе стороны зала.

Также в этой подзадаче решения, которые округляют C/2 вниз, а не вверх при вычислении ymin, проходят все тесты.

Подзадача 2 (50 баллов)

$1 \le A \le B \le 10^9$, $4 \le C \le D \le 10^9$.

В этой подзадаче 25 тестов, каждый тест оценивается в 2 балла. Баллы за каждый тест начисляются независимо.

Тесты к этой подзадаче находятся в каталоге tests/subtask2 в файлах 21 - 45

Для получения баллов в этой подзадаче необходимо было реализовать полное решение. Возможные ошибки, которые приводили к потере баллов:

- округление C / 2 вниз, а не вверх при вычислении ymin приводит к тому, что не проходит 15 тестов в этой подзадаче, такое решение набирает 70 баллов
- использование 32-битного типа данных для хранения ответа приводит к тому, что не проходит 10 тестов в этой подзадаче, такое решение набирает 80 баллов.

Задача 2. Призы

Для решения этой подзадачи воспользуемся префиксными суммами и префиксными максимумами.

Сначала научимся за O(1) находить сумму значений ценности для любого отрезка номеров. Для этого вычислим значения

$s[i] = a_1 + a_2 + \ldots + a_i$

для всех i от 0 до n. Это можно сделать одним линейным проходом.

Теперь сумма значений ai на отрезке от L до R вычисляется как s[R] – s[L – 1].

Второй шаг решения: пусть Алиса выбрала отрезок номеров от A до A + k – 1. Теперь Боб может выбрать отрезок длины k с началом от 1 до A – k или с началом от A + k до n – k + 1. Для всех i от 1 до n вычислим следующую величину

pref[i] = max(s[k] – s[0], s[k+1] – s[1], …, s[i] – s[i – k])

и величину

suff[i] = max(s[i + k – 1] – s[i – 1], s[i + k] – s[i], …, s[n] – s[n – k])

Обе этих величины также можно вычислить одним линейным проходом.

Теперь для решения задачи достаточно перебрать ход Алисы, если Алиса выбрала отрезок призов от A до A + k – 1, то максимальное значение, которое может достаться Бобу это

max(pref[A – 1], suff[A + k])

Среди этих значений надо выбрать минимум.

Приведем код решения на C++

```
for (int i = 1; i <= n; i++) {
    scanf("%d", &a[i]);
    s[i] = s[i - 1] + a[i];
}

for (int i = k; i <= n; i++) {
    pref[i] = max(pref[i - 1], s[i] - s[i - k]);
}
for (int i = n - k + 1; i >= 1; i--) {
    suff[i] = max(suff[i + 1], s[i + k - 1] - s[i - 1]);
}

long long best = 2e18;
for (int i = 1; i <= n - k + 1; i++) {
    best = min(best, max(pref[i - 1], suff[i + k]));
}
printf("%I64d\n", best);
```

Сложность получившегося решения O(n).

Система оценки и описание подзадач

В этой задаче три подзадачи. Баллы за подзадачу начисляются только в случае, если все тесты для данной подзадачи успешно пройдены.

Подзадача 1 (30 баллов)

$3 \le n \le 50$, $1 \le a_i \le 10^5$

В этой подзадаче 13 тестов, они находятся в каталоге tests/subtask1 в файлах 01-13

При решении этой подзадачи можно непосредственно перебрать все возможные способы выбрать призы для Алисы и для Боба, а также за O(k) посчитать для каждого возможного выбора суммарную ценность призов Боба. Получается решение за $O(n^3)$.

Подзадача 2 (30 баллов)

$3 \le n \le 5000$, $1 \le a_i \le 10^5$

В этой подзадаче 17 тестов, они находятся в каталоге tests/subtask2 в файлах 14-30

При решении этой подзадачи достаточно применить одну из двух описанных в полном решении идей: находить суммарную ценность призов на отрезке с помощью префиксных сумм. При этом можно перебирать по-отдельности выбор Алисы и выбор Боба, получая решение за $O(n^2)$.

Подзадача 3 (40 баллов)

$3 \le n \le 100\,000$, $1 \le a_i \le 10^9$

В этой подзадаче 17 тестов, они находятся в каталоге tests/subtask3 в файлах 31-47.

Для решения этой подзадачи необходимо реализовать полноценное решение, кроме того, следует обратить внимание, что ответ может не поместиться в 32-битный тип данных, поэтому необходимо использовать 64-битный типа.

Задача 3. Река

Основную трудность при решении этой задачи представляет собой поиск предприятия, с которым происходит событие.

Мы опишем общий подход к хранению информации о предприятиях. используемый в решении всех подзадач, а затем в описании решения каждой подзадачи опишем структуру данных, которая позволяет решить данную подзадачу.

Будем хранить предприятия в виде двусвязного списка. По мере появления новых предприятий будем присваивать им глобальные номера, и для предприятия v будем хранить next[v] – номер следующего предприятия и prev[v] – номер предыдущего предприятия вдоль реки.

При банкротстве предприятия соответствующий ему элемент удаляется из списка, а при разделении оно удаляется и в соответствующее место списка вставляется два предприятия.

Приведем фрагмент кода на языке C++, который обрабатывает событие с типом e в предположении, что глобальный номер предприятия v, с которым происходит событие, был каким-либо образом получен из запроса.

Счетчик в переменной t используется для присваивания глобальных номеров.

```
        if (e == 1) {
            if (prev[v] == -1) {
                int u = next[v];
                ans -= a[u] * a[u];
                a[u] += a[v];
                ans += a[u] * a[u];
                prev[u] = -1;
            } else if (next[v] == -1) {
                int w = prev[v];
                ans -= a[w] * a[w];
                a[w] += a[v];
                ans += a[w] * a[w];
                next[w] = -1;
            } else {
                int w = prev[v];
                int u = next[v];
                ans -= a[u] * a[u];
                ans -= a[w] * a[w];
                a[w] += a[v] / 2;
                a[u] += (a[v] + 1) / 2;
                ans += a[w] * a[w];
                ans += a[u] * a[u];
                next[w] = u;
                prev[u] = w;
            }
            a[v] = 0;
        } else { // e = 2
            int w = prev[v];
            int u = next[v];
            a[t] = a[v] / 2;
            a[t + 1] = (a[v] + 1) / 2;
            ans += a[t] * a[t];
            ans += a[t + 1] * a[t + 1];
            if (w != -1) {
                next[w] = t;
            }
            next[t] = t + 1;
            next[t + 1] = u;
            prev[t] = w;
            prev[t + 1] = t;
            if (u != -1) {
                prev[u] = t + 1;
            }
            t += 2;
        }
```

Осталось подобрать структуру данных, позволяющую реализовать поиск глобального номера предприятия по его порядковому номеру от истока реки.

Система оценки и описание подзадач

В этой задаче четыре подзадачи. Баллы за каждую подзадачу начисляются только в случае, если все тесты для данной подзадачи успешно пройдены.

Подзадача 1 (30 баллов)

$2 \le n \le 100$, $1 \le k \le 100$, $1 \le a_i \le 100$, $p = 1$

В этой подзадаче 10 тестов, они находятся в каталоге tests/subtask1 в файлах 01 - 10

Для решения этой подзадачи достаточно любым способом реализовать описанные в условии действия. Например, можно просто хранить информацию обо всех предприятиях в

массиве в том порядке, в котором они находятся вдоль реки и при изменениях сдвигать часть массива. Либо можно применить описанную идею со списком и непосредственно пробегать указателем каждый раз от начала списка до соответствующей позиции.

Подзадача 2 (30 баллов)

$2 \le n \le 100\,000$, $1 \le k \le 100\,000$, $1 \le a_i \le 10^4$, $p = 2$
Для всех i от 1 до $k - 1$ выполнено условие: $|v_i - v_{i+1}| \le 10$

В этой подзадаче 5 тестов, они находятся в каталоге tests/subtask2 в файлах 11 - 15

При решении этой подзадачи уже обязательно реализовать двусвязный список. При этом, благодаря дополнительному условию, предприятия, с которыми происходят события, находятся недалеко друг от друга в списке. Поэтому, поддерживая указатель на последнее предприятие, с которым произошло событие, можно быстро перемещаться вдоль списка к очередному предприятию.

Подзадача 3 (20 баллов)

$2 \le n \le 100\,000$, $1 \le k \le 100\,000$, $1 \le a_i \le 10^4$, $p = 3$
Все события имеют тип $e_i = 1$ (нет разделений предприятий, только банкротства).

В этой подзадаче 5 тестов, они находятся в каталоге tests/subtask3 в файлах 16 - 20

В случае, когда с предприятиями происходит только банкротства, новых предприятий не появляется и число предприятий только уменьшается. Поэтому можно считать, что все предприятия имеют глобальные номера, равные исходному номеру вдоль реки. При решении этой подзадачи для поиска исходного номера предприятия можно использовать структуру данных дерево отрезков для операции «сумма». Создадим дерево отрезков для операции «сумма» на n элементов и присвоим всем элементам значение 1.

При удалении предприятия будем присваивать соответствующему этому предприятию элементу значение 0 и проводить обновление в дереве отрезков.

Для поиска i-го предприятия от начала реки, будем спускаться по дереву отрезков. Находясь в очередной вершине, проверяем: если сумма в левом поддереве больше или равна i, то переходим в левого сына текущей вершины, иначе — вычитаем из i сумму значений в левом поддереве и переходим в правого сына. Лист, в котором в результате спуска окажется указатель, соответствует i-му предприятию от начала реки.

Обе операции выполняются за O(log n), поэтому общая сложность решения O((n + k) log n).

Отметим, что решение для подзадачи 3 не решает подзадачи 1 и 2, поэтому для получения 80 баллов необходимо реализовать как идею со списком для подзадач 1 и 2, так и дерево отрезков для подзадачи 3. Для удобства участников, которые реализуют в своей программе несколько вариантов решения для различных подзадач, во входных данных указывается номер подзадачи.

Подзадача 4 (20 баллов)

$2 \le n \le 100\,000$, $1 \le k \le 100\,000$, $1 \le a_i \le 10^4$, $p = 4$

В этой подзадаче 6 тестов, они находятся в каталоге tests/subtask4 в файлах 21 - 26

В отличие от удаления предприятий, добавление новых предприятий в середину списка нельзя просто реализовать с помощью дерева отрезков. Здесь на помощь может прийти структура данных, поддерживающая следующие операции:

- получение элемента по номеру;
- удаление элемента на заданной позиции;
- добавление элемента в заданную позицию.

Есть две достаточно распространенных структуры, которые подходят под эти требования.

Первый подход заключается в использовании «корневой декомпозиции». Разобьем множество предприятий на последовательные отрезки длиной B. Внутри каждого из таких отрезков будем решать задачу как в первой подзадаче – хранить последовательность элементов в массиве, сдвигая элементы при необходимости. Если же в процессе добавления элементов длина отрезка станет больше 2B, разобьем его на два. Сами же эти отрезки также будем хранить в массиве, сдвигая конец массива при необходимости.

При такой реализации каждая операция выполняется за время O(B + (n+k) / B), выбирая B порядка квадратного корня из n + k, получаем асимптотику O(k sqrt(n+k))

Второй подход заключается в выборе для хранения предприятий декартового дерева по неявному ключу, в этом случае операции получения элемента по номеру, удаления и вставки выполняются в среднем за O(log n), общее время работы решения получается O(k log n).

Отметим также, что в отличие от подзадачи 3, решение подзадачи 4 решает также все предыдущие подзадачи.

Задача 4. Чемпионат по поиску в сети Меганет

Заметим, что поскольку имя сервера и имя раздела состоят не более чем из 5 частей каждое, то для каждого адреса существует лишь небольшое число фильтров, под которые он потенциально может подходить.

А именно, адрес подходит под фильтры, где фильтр сервера совпадает с его именем сервера, либо содержит некоторое количество его заключительных частей, перед которыми идет звездочка, всего не более 6 вариантов (без звездочки, оставить 1, 2, …, 5 конечных частей). Аналогично, фильтр раздела может либо совпадать с именем сервера, либо содержать некоторое количество его начальных частей, после которых идет звездочка, всего не более 7 вариантов (без звездочки, оставить 0, 1, 2, …, 5 начальных частей).

Получается, что для каждого адреса существует не более 42 различных фильтров, под которые он может подходить.

Разместив фильтры в соответствующей структуре данных, в которой возможен быстрый поиск наличия фильтра, например в боре или в структуре std::set, мы можем за O(1) или O(log n) проверять наличие фильтра.

Отметим также, что условие не гарантирует, что все фильтры различны, и в решении следует учитывать, что может быть несколько одинаковых фильтров.

Система оценки и описание подзадач

Подзадача 1 (27 баллов)

$1 \le n \le 1000$, $1 \le k \le 1000$, $p = 1$

Каждый фильтр начинается с «*.» и заканчивается на «/*».

В этой подзадаче 9 тестов, каждый тест оценивается в 3 балла. Баллы за каждый тест начисляются независимо.

Тесты к этой подзадаче находятся в каталоге tests/subtask1 в файлах 01-09

Для решения этой подзадачи достаточно произвольным образом проверить для каждого фильтра, после удаления концевых звездочек, является ли он подстрокой адреса.

Благодаря потестовой оценке решения различной эффективности получают различное число баллов.

Максимально эффективные должны действовать действуют одним из двух способов: либо быстро проверять вхождения подстроки в строку, например, алгоритмом Кнута-Морриса-Пратта, либо учитывать, что «стык» имени сервера и имени раздела находится однозначно и можно проверять лишь вхождения, где он правильно расположен.

Подзадача 2 (25 баллов)

$1 \le n \le 50\,000$, $1 \le k \le 50\,000$, $p = 2$

Фильтры не содержат символов «*».

В этой подзадаче 5 тестов, каждый тест оценивается в 5 баллов. Баллы за каждый тест начисляются независимо.

Тесты к этой подзадаче находятся в каталоге tests/subtask1 в файлах 10-14

В тестах к этой подзадаче для каждого адреса может быть только один фильтр, под который он подходит, полностью с ним совпадающий. Для решения можно применить любую стандартную структуру данных, например, сложить все фильтры в map<string, int> и искать число подходящих фильтров одним запросом к этой структуре.

Как и в предыдущей подзадаче, благодаря потестовой оценке менее эффективные решения могут набрать частичные баллы.

Подзадача 3 (48 баллов)

$1 \le n \le 50\,000$, $1 \le k \le 50\,000$, $p = 3$

В этой подзадаче 12 тестов, каждый тест оценивается в 4 балла. Баллы за каждый тест начисляются независимо.

Тесты к этой подзадаче находятся в каталоге tests/subtask3 в файлах 15-26.

Для решения этой подзадачи необходимо применить подход, описанный в основном решении. Ограничения по времени в этой задаче достаточно жесткие, различные неасимптотические неточности в реализации, либо неудачное использование хеширования для сравнения строк, приводят к тому, что часть тестов может не пройти, такие решения получают частичную оценку.

Задача 5. Кольцевая линия

Полное решение этой задаче основано на следующем наблюдении. Пусть Андрей и Борис живут на различных станциях, причем при движении от Андрея к Борису по и против часовой стрелки необходимо проехать различное число станций. Под условие задачи подходят такие пары станций и только они.

Действительно, если Андрей и Борис живут на одной станции, то соседние по и против часовой стрелки станции обе характеризуются парой [1, 1] и не различимы описанным способом.

Если же они живут на противоположных станциях, то есть от одного друга до другого нужно проехать ровно n / 2 станций в любом из направлений (отметим, что в этом случае n должно быть четно), то аналогично, две соседние с одним из них станций характеризуются обе парой [1, n / 2 – 1].

Пусть теперь описанное условие выполнено. Тогда заметим, что для каждого числа i есть не более двух станций, до которых расстояние от станции Андрея равно i. И если, например, расстояние от Андрея до Бориса по часовой стрелке меньше, чем против, то та из станций, до которой расстояние i получается при движении к Андрею по часовой стрелке, ближе к станции Бориса, чем вторая. Поэтому они имеют разный второй компонент пары.

Итого получаем следующую формулу:

- если n четно, то ответ равен $n^2 - 2n$
- если n нечетно, то ответ равен $n^2 - n$.

Система оценки и описание подзадач

В этой задаче три подзадачи. Баллы за подзадачу начисляются только в случае, если все тесты для данной подзадачи успешно пройдены.

Подзадача 1 (25 баллов)

$3 \le n \le 50$.

В этой подзадаче 10 тестов, они находятся в каталоге tests/subtask1 в файлах 1 – 10

При решении этой подзадачи можно, например, перебрать все пары станций A и B, для каждой пары перебрать все варианты станции X, для каждой из них построить искомую пару чисел и убедиться, что все они различны. Решение за $O(n^4)$.

Отметим, также, что изучение полученных результатов позволяет выдвинуть математическую гипотезу о правильном решении и доказать ее.

Подзадача 2 (25 баллов)

$3 \le n \le 200$.

В этой подзадаче 8 тестов, они находятся в каталоге tests/subtask2 в файлах 11 – 18

При решении этой подзадачи можно, например, заметить, что если для некоторого значения A подходят k станций B, то это же количество станций из соображений симметрии будет подходить и для любого другого значения B, а значит можно перебирать только одну из двух станций. Решение за $O(n^3)$.

Подзадача 3 (50 баллов)

$3 \le n \le 40000$.

В этой подзадаче 9 тестов, они находятся в каталоге tests/subtask2 в файлах 19 – 27

Для решения этой подзадачи необходимо вывести приведенную выше формулу.

Задача 6. Вырубка леса

Для решения этой задачи применим двоичный поиск.

Заметим, что за 0 дней вырубить лес нельзя, а 2X / (A + B), округленное вверх дней заведомо достаточно.

Научимся проверять, можно ли вырубить лес за T дней.

Для этого заметим, что за T дней Дмитрий вырубит

D = (T div K) * (K – 1) * A + (T mod K) * A деревьев,

а Федор вырубит

F = (T div M) * (M – 1) * B + (T mod M) * B деревьев.

Здесь как x div y обозначена целая часть от деления x на y, а как x mod y остаток от деления x на y. Если оба фермера в сумме вырубят за T дней не меньше X деревьев, то этого времени достаточно для вырубки леса, иначе — недостаточно.

Отдельно отметим, что в этой задаче к выбору начального значения правой границы двоичного поиска, чтобы не вызвать переполнения 64-битного целочисленного типа данных при вычислениях.

Приведем реализацию на языке C++.

```
long long left = 0;
long long right = 2 * x / (a + b) + 1;
while (left + 1 < right)
{
    long long c = (left + right) / 2;
    long long cut1 = a * (k - 1) * (c / k) + a * (c % k);
    long long cut2 = b * (m - 1) * (c / m) + b * (c % m);
    if (cut1 + cut2 >= x)
        right = c;
    else
        left = c;
}
```

В конце ответ находится в переменной right.

Заметим, что тот же код на языке программирования Python застрахован от переполнения типов, там аккуратности требуется меньше.

```
l = 0
r = 2 * x
while l < r - 1:
    t = (l + r) // 2
    dmi = (t // k) * (k - 1) * a + (t % k) * a
    fed = (t // m) * (m - 1) * b + (t % m) * b
    if dmi + fed >= x:
        r = t
    else:
        l = t
```

Система оценки и описание подзадач

Подзадача 1 (32 балла)

$1 \le X \le 1000$, $1 \le A, B \le 1000$, $2 \le K, M \le 1000$

Баллы за подзадачу начисляются только в случае, если все тесты успешно пройдены.

В этой подзадаче 14 тестов, они находятся в каталоге tests/subtask1 в файлах 01 - 14

В этой подзадаче можно применить непосредственную симуляцию по дням, поддерживая информацию о том, сколько деревьев уже вырублено.

Подзадача 2 (10 баллов)

$1 \le X \le 10^{18}$

$X < K$

$X < M$

При решении этой подзадачи можно считать, что лесорубы не отдыхают.

Баллы за подзадачу начисляются только в случае, если все тесты успешно пройдены.

В этой подзадаче 5 тестов, они находятся в каталоге tests/subtask2 в файлах 15 - 19

Если лесорубы не отдыхают, то каждый день срубается A + B деревьев, потребуется округленное вверх X / (A + B) дней.

Подзадача 3 (10 баллов)

$1 \le X \le 10^{18}$

Дополнительно к приведенным ограничениям выполняется условие $K = M$.

Баллы за подзадачу начисляются только в случае, если все тесты успешно пройдены.

В этой подзадаче 6 тестов, они находятся в каталоге tests/subtask3 в файлах 20 - 25

В этой задаче лесорубы одинаковые, поэтому можно заметить следующее.

За каждый K дней срубается (K – 1) * (A + B) деревьев. Заметим, что полных блоков по (K – 1) * (A + B) деревьев будет Z = X div ((K – 1) * (A + B)), в этих блоках будет срублено Y = Z * (K – 1) * (A + B) деревьев.

Тогда если Y < X, то ответ равен Z * K + (X – Y + A + B - 1) div (A + B), а иначе он равен Z * K – 1.

Подзадача 4 (48 баллов)

$1 \le X \le 10^{18}$, $1 \le A, B \le 10^9$, $2 \le K, M \le 10^{18}$

В этой подзадаче 16 тестов, каждый тест оценивается в 3 балла. Баллы за каждый тест начисляются независимо.

Тесты к этой подзадаче находятся в каталоге tests/subtask4 в файлах 26 - 41

Для решения этой подзадачи необходимо воспользоваться двоичным поиском, описанным выше. Частичные баллы получают решения, содержащие различные ошибки в двоичном поиске, либо ошибки, связанные с переполнением из-за неверно выбранных типов данных или неверно выбранной правой границы двоичного поиска.

Задача 7. Укладка плитки

Для решения этой задачи необходимо применить динамическое программирование.

Рассмотрим сначала решение для первых трех подзадач.

Будем укладывать плитки, начиная слева, пусть мы полностью заполнили первые k столбцов и все плитки содержат хотя бы один единичный квадрат в одном из этих столбцов. Заметим, что в k+1-м столбце плитками, которые задевают первые k столбцов, могут быть заполнены: ни один из единичных квадратов, только верхний квадрат, только нижний квадрат, оба квадрата.

Обозначим число этих замощений как A[k], B[k], C[k] и D[k], соответственно.

Легко заметить, что

A[k] = 2 * A[k – 1] + B[k – 1] + C[k - 1] + D[k – 1] (если k-й столбец ранее не было заполнен, то можно либо положить одну вертикальную двойную, либо две единичных плитки, в остальных случаях способ один)

B[k] = A[k – 1] + C[k – 1]

C[k] = A[k – 1] + B[k – 1]

D[k] = A[k – 1]

Ответом же на задачу является значение A[n].

Для решения подзадачи 4 осталось заметить, что если информация о существующих единичных плитках противоречит соответствующему варианту перехода в динамическом программировании, то этот переход делать не следует.

Время работы решения O(n).

Система оценки и описание подзадач

Подзадача 1 (20 баллов)

$1 \le n \le 8$, $k = 0$

Баллы за подзадачу начисляются только в случае, если все тесты подзадачи пройдены.

В этой подзадаче 8 тестов, они находятся в каталоге tests/subtask1 в файлах 01 - 08

Для решения этой подзадачи можно использовать рекурсивный перебор всех замощений.

Подзадача 2 (20 баллов)

$1 \le n \le 1000$, $k = 0$

Баллы за подзадачу начисляются только в случае, если все тесты подзадачи пройдены.

В этой подзадаче 10 тестов, они находятся в каталоге tests/subtask2 в файлах 09 - 18

Рассмотрим альтернативный подход к динамическому программированию, который приводит к решению за $O(n^2)$, которое решает данную подзадачу.

Будем считать только величину A[k] – число способов замостить полностью первые k столбцов, не задевая других клеток. Рассмотрим последний столбец.

Если в нем лежит одна вертикальная двойная плитка или две единичных, число таких способов равно 2 * A[k – 1].

Если в нем лежит две горизонтальные двойные плитки, число таких способов равно A[k – 2].

Иначе пусть в нем лежит горизонтальная двойная и единичная плитка (есть два таких варианта). Заметим, что в этом случае в предпоследнем столбцы может лежать либо горизонтальная двойная, либо единичная плитка, в первом случае во втором с конца столбце имеет место та же ситуация, и так далее. Таким образом, к результату надо прибавить 2 * (A[k – 2] + A[k – 3] + … + A[0]).

Получаем решение за $O(n^2)$.

Подзадача 3 (20 баллов)

$1 \le n \le 100\,000$, $k = 0$

Баллы за подзадачу начисляются только в случае, если все тесты подзадачи пройдены.

В этой подзадаче 10 тестов, они находятся в каталоге tests/subtask3 в файлах 19 - 28

Для решения этой задачи можно применить либо решение, описанное выше, либо адаптировать решение предыдущей подзадачи. Заметим, что единственное из-за чего время работы составляет $O(n^2)$ — вычисление на каждом шаге суммы (A[k – 2] + A[k – 3] + … + A[0]). Но это вычисление можно не проводить каждый раз, поддерживая сумму префикса массива A. Получается альтернативное решение за O(n).

Подзадача 4 (40 баллов)

$1 \le n \le 100\,000$, $1 \le k \le 2n$

В этой подзадаче 20 тестов, каждый тест оценивается в 2 балла. Баллы за каждый тест начисляются независимо.

Тесты к этой подзадаче находятся в каталоге tests/subtask4 в файлах 29 - 48

Для решение этой подзадачи необходимо применить полноценное решение, описанное выше. Различные ошибки в решении: неверный учет случаев, когда в столбце уже уложены обе плитки, использование предположения, что плитки описаны в порядке возрастания ряда, и т.д. приводят к потере части баллов в этой подзадаче.

Задача 8. Магические порталы

В условии задачи описан граф, который называется *турнир* — ориентированный граф, в котором между каждой парой различных есть ровно одно ребро.

Отметим следующее свойство турнира: в любом турнире есть гамильтонов путь — простой путь, проходящий через каждую вершину ровно один раз. В частности, если турнир является ациклическим, то порядок вершин в гамильтоновом пути соответствует топологической сортировке вершин.

Решим сначала первые три подзадачи, когда требуется предоставлять частичный отчет. Необходимо выяснить, какое число совершенных городов можно получить разворотом ровно одного ребра, если это число строго больше исходного числа совершенных городов.

Найдем компоненты сильной связности заданного графа и построим его конденсацию. Заметим, что конденсация также будет турниром, причем ациклическим, поэтому в ней существует гамильтонов путь, причем он соответствует топологической сортировке компонент. Обозначим компоненты сильной связности в порядке их топологической сортировки как C1, C2, …, Cs, а число вершин в них, соответственно, как c1, c2, …, cs.

Заметим, что совершенными в исходном графе являются в точности вершины компоненты C1, то есть k = c1.

Пусть мы развернули ребро, оба конца которого не принадлежат компоненте C1. Заметим, что ни от одной вершины, от которой ранее нельзя было добраться до компоненты C1, по-прежнему нельзя до нее добраться. Значит, разворот любого такого ребра не меняет число совершенных городов.

Разворот ребра внутри компоненты C1 также не может увеличить число совершенных городов. Правда он может уменьшить это число, мы вернемся к анализу этой ситуации при рассмотрении подзадачи 4.

Пусть теперь мы развернули ребро, которое соединяет вершину из компоненты C1 с вершиной из компоненты Ci. Существует два случая.

1. i = 2, причем обе компоненты C1 и C2 имеют ровно по одной вершине. В этом случае эти компоненты просто «меняются ролями» и в графе остается одна совершенная вершина (именно такой граф приведен в примере).
2. i > 2 или одна из компонент C1 или Ci имеет хотя бы две вершины. В этом случае по прежнему можно добраться от любой вершины компоненты C1 до любой вершины в графе, кроме того, теперь для всех компонент с номерами от 2 до i, включительно, также выполняется это условие, поскольку от них можно добраться до компоненты C1 (сначала по гамильтонову пути, затем по развернутому ребру).

Во втором случае к ответу для суммы c1 + c2 + … + ci следует добавить 1. Легко заметить, что всего к этому ответу будет прибавлено число c1*ci.

Компоненты сильной связности и их топологическую сортировку можно найти за $O(n^2)$, остальная часть решения работает за O(n), таким образом получаем решение первых трех подзадач за время $O(n^2)$.

Рассмотрим теперь составление полного отчета. Для этого необходимо выяснить, разворот каких ребер приводит к уменьшению размера компоненты, из которой достижимы все вершины. Ясно, что это должно быть одно из ребер, оба конца которого лежат внутри этой компоненты.

Известно, что в сильно связном турнире существует гамильтонов цикл, причем его можно найти за $O(n^2)$. Найдем гамильтонов цикл в компоненте C1. Разворот любого ребра, не принадлежащего этому циклу, сохраняет C1 сильно связной (по гамильтонову циклу) и не приводит к изменению числа совершенных городов.

Заметим, что если мы развернем ребро uv гамильтонова цикла, то из v в u можно добраться как по новому ребру, так и по остатку гамильтонова цикла, поэтому можно считать, что мы не разворачиваем, а удаляем по одному ребра гамильтонова цикла, и для каждого ребра надо сказать, из какого числа вершин по прежнему достижимы все.

Зафиксируем порядок вершин v1, v2, …, vk вдоль гамильтонова цикла, удалим ребро vk → v1. Пройдем вдоль оставшегося пути от v1 до vk и для каждой вершины vi найдем

величину x[i], равную максимальному j, такому что есть ребро из vj в vi. Рассмотрим отрезки [i, x[i]], пусть y – максимальный номер вершины, такой отрезок от 1 до y полностью покрыт этими отрезками. Тогда совершенными остались в точности вершины от v1 до vy.

Заметим, что число y можно найти за O(n), изначальное построение массива x[i] выполнено за $O(n^2)$.

Теперь вернем на место ребро vk → v1 и удалим ребро v1 → v2. Заметим, что это эквивалентно тому, что мы перенесли вершину v1 в конец пути. За O(n) мы можем обновить значения x[i], учитывая теперь, что вершина v1 имеет «максимальный» номер. Найдем теперь новое значение y, совершенными остались вершины от v2 до vy. Проделав это со всеми ребрами из гамильтонова цикла в C1, мы найдем все способы строго уменьшить число совершенных вершин, время работы алгоритма составляет $O(n^2)$.

Система оценки и описание подзадач

В этой задаче четыре подзадачи. Баллы за каждую подзадачу начисляются только в случае, если все тесты подзадачи пройдены.

Подзадача 1 (20 баллов)

$2 \le n \le 50$, $p = 0$

В этой подзадаче 19 тестов, они находятся в каталоге tests/subtask1 в файлах 01 - 19

Для решения этой подзадачи можно перебрать все ребра, в явном виде развернуть каждое ребро и с помощью обхода в глубину найти все совершенные города. Это решение работает за $O(n^4)$.

Подзадача 2 (30 баллов)

$2 \le n \le 300$, $p = 0$

В этой подзадаче 16 тестов, они находятся в каталоге tests/subtask2 в файлах 20 - 35

Для решения этой подзадачи необходимо провести рассуждения, проведенные выше для частичного отчета, но не требуется знать эффективного алгоритма поиска компонент сильной связности и топологической сортировки. Компоненты сильной связности можно построить за $O(n^3)$, например, алгоритмом Флойда.

Подзадача 3 (20 баллов)

$2 \le n \le 2000$, $p = 0$

В этой подзадаче 14 тестов, они находятся в каталоге tests/subtask3 в файлах 36 - 49

Для решения этой подзадачи необходимо реализовать полноценное решение для частичного отчета, описанное выше.

Подзадача 4 (30 баллов)

$2 \le n \le 2000$, $p = 1$

В этой подзадаче 28 тестов, они находятся в каталоге tests/subtask4 в файлах 50 - 77

Для решения этой подзадачи необходимо реализовать полноценное решение для полного отчета, описанное выше.