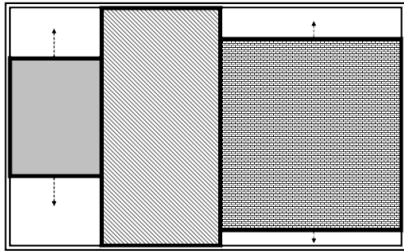


Задача А. Транспортировка артефактов

Авторы задачи: Даниил Орешников, Андрей Станкевич

Заметим, что прямоугольники, с точностью до поворота плоскости на 90° , поворота каждого прямоугольника на 90° и их перестановки, могут быть расположены относительно друг друга двумя способами:

- правая сторона первого касается левой стороны второго, а правая сторона второго — левой стороны третьего

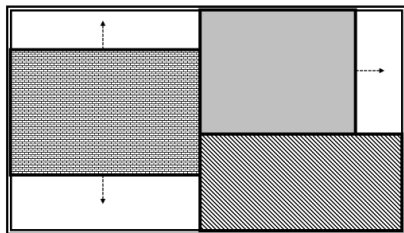


Как видно на картинке, все три прямоугольника расположены в ряд. Стрелками показано, что на самом деле прямоугольники можно двигать друг относительно друга в пределах максимума из их высот.

В этом случае площадь баржи равна

$$(a_1 + a_2 + a_3) \cdot \max(b_1, b_2, b_3)$$

- слева находится первый прямоугольник, справа — блок из второго и третьего, при чем нижняя сторона второго касается верхней стороны третьего



Если прямоугольники расположены не в ряд, мы приходим к этой конфигурации. Не обязательно все три должны иметь общую точку, но если два касаются горизонтальными сторонами, и два — вертикальными, их можно сдвинуть, не увеличив площадь баржи.

В этом случае площадь баржи равна

$$(a_1 + \max(a_2, a_3)) \cdot \max(b_1, b_2 + b_3)$$

Осталось только перебрать все возможные повороты прямоугольников и их перестановки. Суммарно надо перебрать $8 \cdot 6 \cdot 2 = 96$ вариантов. Это можно сделать с помощью нескольких циклов и/или ручного перебора вариантов.

Отметим, что благодаря потестовой оценке, если участник забывает рассмотреть некоторые случаи, он лишается только части баллов.

Задача В. Скоростной транспорт

Автор задачи: Иван Сафонов

Для решения первой подзадачи можно перебрать все варианты новых границ отрезка и посчитать количество подходящих вариантов. Получится решение за $O(n^4)$.

```
long long tot = r1 - l1 + r2 - l2;  
long long ans = 0;  
for (long long ln1 = l1; ln1 <= r1; ln1++) {  
    for (long long rn1 = ln1; rn1 <= r1; rn1++) {  
        for (long long ln2 = rn1 + 1; ln2 <= l2; ln2++) {  
            for (long long rn2 = r2; rn2 <= ln2 + tot; rn2++) {  
                if (rn1 - ln1 + rn2 - ln2 == tot) {  
                    ans++;  
                }  
            }  
        }  
    }  
}
```

```
    }  
  }  
}
```

Заметим, что количество подходящих значений $rn2$ не более одного, вычислим его по формуле и проверим, что оно подходит. Получится решение за $O(n^3)$. Оно подходит также для второй подзадачи.

```
long long tot = r1 - l1 + r2 - l2;  
long long ans = 0;  
for (long long ln1 = l1; ln1 <= r1; ln1++) {  
  for (long long rn1 = ln1; rn1 <= r1; rn1++) {  
    for (long long ln2 = rn1 + 1; ln2 <= l2; ln2++) {  
      long long rn2 = tot - (rn1 - ln1) + ln2;  
      if (rn2 >= r2) {  
        ans++;  
      }  
    }  
  }  
}
```

Для дальнейшей оптимизации зафиксируем $rn1$ и $ln2$. Заметим, что число подходящих пар левых границ $ln1$ и правых границ $rn2$ выражается простой формулой. Получим решение за $O(n^2)$, которое укладывается в ограничения по времени также для третьей подзадачи.

```
long long tot = r1 - l1 + r2 - l2;  
long long ans = 0;  
for (long long rn1 = l1; rn1 <= r1; rn1++) {  
  for (long long ln2 = rn1 + 1; ln2 <= l2; ln2++) {  
    ans += max(0LL, rn1 - max(l1, rn1 + l2 + l1 - r1 - ln2) + 1);  
  }  
}
```

Следующий уровень оптимизации, который позволяет решить все подзадачи и набрать 100 баллов: избавиться от внутреннего цикла по $ln2$. Заметим, что число, которое прибавляется к ans внутри цикла кусочно-линейно зависит от $ln2$. Суммируя арифметические прогрессии, получаем значение, которое необходимо прибавить к ответу для фиксированного $rn1$.

```
long long tot = r1 - l1 + r2 - l2;  
long long ans = 0;  
for (long long rn1 = l1; rn1 <= r1; rn1++) {  
  ans += (rn1 - l1 + 1) * (l2 + 1 -  
    max(max(rn1 + 1, l2 + l1 - r1), rn1 + l2 - r1));  
  long long f = max(rn1 + 1, l2 + l1 - r1);  
  long long t = min(rn1 + l2 - r1, l2 + 1);  
  ans += (-l2 - l1 + r1 + 1) * (t - f);  
  ans += (f + (t - 1)) * (t - f) / 2;  
}
```

Полученное решение работает за $O(n)$ и проходит все подзадачи.

Заметим, что можно сделать последний шаг и получить решение за $O(1)$, избавившись также от последнего оставшегося цикла. В результате для получения ответа нужно будет вычислить сумму нескольких значений, каждое из которых имеет порядок куба от входных данных. К счастью, для получения полного балла по задаче этого не требовалось.

Задача С. Продукты в экспедиции

Автор задачи: Дмитрий Саютин

В первой подзадаче у есть только один тип продуктов. Нужно просто проверить, можно ли съесть этот тип продукта. За время t_1 все участники экспедиции смогут съесть $c \cdot t_1$ порций, поэтому если $k_1 \leq c \cdot t_1$, то ответ 1, иначе ответ 0. Время работы этого решения $O(1)$.

Как для множества типов продуктов проверить, можно ли его съесть полностью? Пусть множество состоит из m типов продуктов, таких что время, за которое они портятся это t_1, t_2, \dots, t_m , количество порций продуктов k_1, k_2, \dots, k_m , соответственно. Отсортируем их по возрастанию времени, то есть пусть $t_1 \leq t_2 \leq \dots \leq t_m$. Заметим, что для всех $1 \leq i \leq m$ должно быть выполнено, что $k_1 + k_2 + \dots + k_i \leq c \cdot t_i$. Это так, потому что время, за которое мы сможем есть первые i типов продуктов это t_i , при этом за это время мы должны будем съесть как минимум $k_1 + k_2 + \dots + k_i$ порций. Докажем, что это также является достаточным условием того, что мы сможем все съесть. Заметим, что $k_1 \leq c \cdot t_1$, поэтому за первые t_1 дней мы сможем съесть все порции продукта первого типа. Затем, так как $k_1 + k_2 \leq c \cdot t_2$, мы за первые t_2 дней сможем съесть все порции первого и второго типа, так как на продукты первого типа мы потратили ровно k_1 возможностей съесть одну порцию. И так будет для любого i : поскольку $k_1 + k_2 + \dots + k_i \leq c \cdot t_i$, мы сможем съесть за t_i дней все продукты первых i типов, потому что мы съели все продукты первых $i - 1$ типов, потратив на это ровно $k_1 + k_2 + \dots + k_{i-1}$ возможностей съесть одну порцию.

Во второй подзадаче переберем все 2^n подмножеств типов продуктов. Для каждого из них, за время $O(n \log n)$ или $O(n)$ (если предварительно отсортировать все t_i) мы сможем определить, можно ли съесть все порции выбранного множества типов продуктов. Среди всех таких подмножеств найдем подмножество максимального размера, это будет ответом. Время работы этого решения $O(2^n n)$.

Для того, чтобы решить третью подзадачу, отсортируем все t_i по возрастанию. Мы должны выбрать некоторое подмножество типов продуктов, которое удовлетворяет полученному нами критерию. Обозначим за $T = t_n$ — максимальное значение t_i . По ограничениям этой подзадачи $T \leq 2000$. Посчитаем $dp[i][s]$ — максимальное количество типов продуктов, которое можно выбрать, так чтобы его можно было съесть, сумма значений k_j по всем продуктам из множества равна s и продукт i это максимальный номер продукта, взятый в множество. Эти значения мы будем считать для $1 \leq i \leq n$ и $1 \leq s \leq T$. Тогда $dp[i][s] = \max_{1 \leq j < i} dp[j][s - k_i] + 1$, если $s \leq t_i$ и $dp[i][s] = 0$, иначе. Это позволяет нам вычислить все значения за время $O(n^2 T)$, что недостаточно чтобы решить задачу. Для того, чтобы ускорить вычисление, обозначим $pref[i][s] = \max_{1 \leq j \leq i} dp[j][s]$. Тогда $dp[i][s] = pref[i - 1][s - k_i] + 1$ и $pref[i][s] = \max(pref[i - 1][s], dp[i][s])$. Вычислим значения за время $O(nT)$ и ответом будет являться максимум $dp[i][s]$ по всем $1 \leq i \leq n, 1 \leq s \leq T$.

Пропустим четвертую подзадачу и разберем сначала пятую. В этой подзадаче все значения t_i совпадают. Обозначим их за t . Тогда в этом случае критерием того, что мы можем съесть все продукты, количества которых k_1, k_2, \dots, k_m является одно неравенство, а именно $k_1 + k_2 + \dots + k_m \leq c \cdot t$. То есть мы хотим взять как можно больше типов продуктов, таких, что сумма их количеств не превышает $c \cdot t$. Для этого можно воспользоваться жадным алгоритмом: отсортируем все значения k_i и будем набирать их от меньшего к большему, пока сумма не превысит $c \cdot t$. Асимптотика решения этой подзадачи $O(n \log n)$.

Далее приведем концепцию полного решения. Отсортируем все типы продуктов по убыванию времени, за которое они портятся, то есть $t_1 \geq t_2 \geq \dots \geq t_n$. Будем перебирать i от 1 до n и смотреть, какие продукты надо есть в обратном порядке времени. Будем хранить множество типов продуктов и для каждого типа сколько еще порций этого типа осталось. Перебирая i добавим тип продукта i и то, что осталось k_i порций этого типа. Теперь до следующего добавления типа продукта мы можем съесть $c \cdot (t_i - t_{i-1})$ порций (положим $t_0 = 0$). Заметим, что теперь выгодно начать есть те типы продуктов, которых осталось меньше всего, потому что разные типы продуктов к этому моменту равны между собой и невыгодно есть тип, которого больше. Поэтому будем есть продукты в порядке возрастания оставшихся порций этого типа, пока количество порций, которые мы можем сейчас съесть не будет равно 0 или пока все типы продуктов не кончатся. Если мы доедаем какой-

то тип продуктов, то добавим его индекс в ответ. Таким образом мы получим множество типов продуктов максимального размера, которое можно съесть.

Для решения четвертой подгруппы можно наивно реализовать это решение — будем хранить массив типов продуктов и количество оставшихся порций. При выборе типа продукта, который надо съесть, линейным поиском найдем минимальное количество и будем есть порцию этого типа. Тогда реализация этого решения будет работать за время $O(n^2)$.

Для полного решения задачи будем хранить типы продуктов в структуре данных s типа, например, `std::set`, сортируя их по количеству оставшихся порций этого типа. При добавлении нового типа, просто положим его в s , если мы хотим съесть $c \cdot (t_i - t_{i-1})$ порций, будем вынимать из s тип продукта с минимальным количеством оставшихся порций и есть. Если количество оставшихся порций будет равно 0, то не будем возвращать этот тип в s , иначе вернем с новым количеством оставшихся порций. Суммарное время работы такого решения будет равно $O(n \log n)$, потому что каждый тип продуктов будет удален из s не более одного раза и мы n раз будем добавлять тип в s и n раз обновлять количество оставшихся порций.

Задача D. Доставка почты

Авторы задачи: Михаил Ютман, Николай Будин

Немного переформулируем задачу. Дано дерево, подвешенное за вершину номер 1. У каждой вершины можно выбрать произвольный порядок детей, после чего, выполняется обход дерева из вершины 1, и все вершины выписываются в порядке первого посещения. Требуется посчитать количество способов выбрать порядок детей у всех вершин, чтобы вершина a_i была выписана раньше вершины b_i для всех i .

Пусть вершина a должна быть выписана раньше, чем b . Если a является предком b , это произойдет в любом случае. Аналогично, если b является предком a , это не произойдет никогда.

Иначе, рассмотрим c — наименьшего общего предка вершин a и b — наиболее глубокую вершину, являющуюся одновременно предком и a , и b . Обозначим за a' ребёнка вершины c , который является предком вершины a , и за b' — ребёнка c , являющегося предком b . Несложно заметить, что a' и b' существуют, единственны и не равны. Тогда a будет выписано до b , если из вершины c переход в a' будет выполнен раньше, чем в b' . Поэтому, требуется посчитать количество способов упорядочить детей вершин, чтобы не нарушалось ни одно из таких условий. Так как условия между вершинами независимы, можно посчитать количество способов упорядочить детей отдельно для каждой вершины, и затем перемножить результаты. Для того, чтобы решить задачу для отдельной вершины, нужно посчитать количество топологических сортировок в ориентированном графе, вершинами которого являются дети данной вершины, а рёбрами — наложенные ограничения.

Для нахождения наименьшего общего предка двух вершин, можно использовать либо наивный алгоритм, работающий за линейное время, что соответствует подгруппам 3–5. Либо любой оптимальный алгоритм, например, отвечающий на один запрос за время $O(\log n)$, что соответствует подгруппам 6–7.

Для нахождения количества топологических сортировок графа, содержащего не более D вершин, можно либо разобрать вручную все случаи для $D = 3$, либо перебрать все перестановки вершин для $D = 7$, либо использовать метод динамического программирования по подмножествам для $D = 12$.