Задача A. Комиссия за банковский перевод

Автор и разработчик задачи: Андрей Станкевич

Самая простая задача в соревновании решается короткой формулой. Приведем код её решения:

```
int k;
cin >> k;
double v = 25.0 + k / 100.0;
if (v < 100) {
    v = 100;
}
if (v > 2000) {
    v = 2000;
}
cout.precision(20);
cout << fixed << v << endl;
```

Можно обойтись и без вещественных чисел:

```
int x;
scanf("%d", &x);
x += 2500;
x = max(x, 10000);
x = min(x, 200000);
printf("%d.%02d\n", x / 100, x % 100);
```

Задача B. Бактерии

Автор задачи: Ильдар Гайнуллин
Разработчик: Иван Волков

Во-первых, отметим, что ДНК бактерий, получившихся на i-м шаге, это просто результаты разбиения ДНК исходной бактерии на 2^{i-1} равных частей. Так что зная (при зафиксированном порядке) ДНК бактерий из i-го шага, легко восстановить ДНК бактерий на последующих и предыдущих шагах, в частности, ДНК исходной бактерии.

Далее заметим, что на каждом шаге у нас есть два ограничения на число различных ДНК: с одной стороны, их не больше чем всего бактерий на этом шаге (то есть 2^{i-1}), а с другой стороны, их не больше чем различных двоичных строк соответствующей длины (то есть не больше чем 2^{len_i}, где $len_i = 2^{n-i+1}$). При этом первое ограничение на каждом шаге слабеет, а второе — наоборот, становится только сильнее.

Чтобы построить ответ, посмотрим на минимальный шаг, на котором $2^{len_i} \leqslant 2^{i-1}$, то есть на котором число бактерий не меньше числа в принципе возможных ДНК. Тогда нам достаточно так «расставить» ДНК бактериям на этом шаге чтобы среди них, во-первых, были все различные двоичные строки длины len_i, а во-вторых, чтобы среди бактерий на $i-1$-м шаге (их мы умеем восстанавливать) все ДНК были попарно различны. Тогда на всех шагах перед i-м мы достигаем максимума, так как «упираемся» в число имеющихся бактерий. А на i-м и следующих шагах мы достигаем максимума, так как «упираемся» в число различных возможных ДНК соответствующей длины.

Но расставить нужным образом ДНК на найденном шаге — это уже довольно техническая задача. Можно, например, сначала «выдать» бактериям всевозможные различные ДНК, а дальше перейти к предыдущему шагу и перебирать еще не выданные ДНК.

Задача C. Проверка маркеров

Автор и разработчик задачи: Николай Калинин

Рассмотрим ситуацию, когда Александр Маркович не найдет два хороших маркера различных цветов. Тогда все маркеры кроме тех, что останутся в куче, можно будет разделить на пары маркеров

различных цветов, хотя бы один из которых не пишет. Также заметим, что если в какой-то момент среди *хороших* маркеров останутся только маркеры одного цвета x, то профессор не сможет начать лекцию независимо от дальнейших действий. Это значит, что мы можем «вернуть» в кучу все такие пары маркеров, в которых оба маркера плохие, это не поможет профессору. Однако это позволяет нам упростить критерий возможного срыва лекции: если можно выбрать несколько пар маркеров различных цветов, где в каждой паре *ровно* один маркер будет плохой, так, что из хороших маркеров в куче останутся только маркеры цвета x, то ответ «YES».

Сформулируем критерий по-другому: если можно каждому хорошему маркеру, кроме маркеров цвета x, сопоставить какой-нибудь плохой маркер другого цвета (и все такие плохие маркеры будут различны), то, возможно, лекция сорвется. Представим двудольный граф, в котором каждая вершина левой доли будет соответствовать хорошему маркеру, а каждая вершина правой доли — плохому. Проведем ребра между вершинами из разных долей, если это маркеры разного цвета (то есть профессор может их достать одновременно). Тогда критерий выглядит следующим образом: если после удаления всех вершин левой доли, соответствующих маркерам цвета x, некоторым паросочетанием можно накрыть все вершины левой доли (все хорошие маркеры), то может произойти такое, что в конце из хороших маркеров останутся только маркеры цвета x. Таким образом, если хотя бы для одного цвета x выполняется этот критерий, то ответ «YES».

Для решения задачи осталось научиться проверять критерий для заданного цвета x. Так как паросочетание должно накрывать все вершины левой доли, то можно воспользоваться леммой Холла: паросочетание, накрывающее все вершины левой доли, существует тогда и только тогда, когда для любого подмножества вершин A из левой доли объединение множеств их соседей в правой доле содержит хотя бы столько же вершин, сколько множество A. Заметим, что в данной задаче можно проверять не все подмножества A, а лишь некоторые из них. Действительно, пусть критерий выполняется для множества всех вершин левой доли (то есть плохих маркеров не меньше, чем хороших). Тогда для любого множества A, в которое входят хотя бы два маркера разных цветов, множество их соседей — все плохие маркеры, а их точно не меньше, чем маркеров в A. Если же в A входят только маркеры одного цвета, то множество их соседей не зависит от их количества, и достаточно проверить только множество из всех маркеров одного цвета.

Таким образом, для фиксированного цвета x достаточно проверить, что:

- всего хороших маркеров, кроме цвета x, не больше, чем всего плохих: $(\sum b_i) - b_x \leqslant \sum a_i$;
- для любого другого цвета y хороших маркеров такого цвета не больше, чем плохих маркеров других цветов: $b_y \leqslant (\sum a_i) - a_y$.

Критерий для цвета y не зависит от выбора цвета x, поэтому их все можно проверить заранее. Тогда при переборе цвета x можно проверить оба критерия за $O(1)$. Итоговая сложность решения: $O(n)$.

Задача D. Междисциплинарные уроки

Автор и разработчик задачи: Андрей Станкевич

Вспомним алгоритм подсчета количества разбиений на слагаемые. Парадоксальным образом, нам понадобится менее эффективный алгоритм, который работает за $O(n^3)$.

Обозначим как $d[n][p]$ количество разбиений числа n на слагаемые не больше p. Для получения рекуррентной формулы переберем, сколько слагаемых равны ровно p, пусть их количество равно t. Тогда

$$d[n][p] = \sum_{t=0}^{\lfloor n/p \rfloor} d[n - tp][p - 1].$$

Теперь нам надо раскрасить эти t слагаемых. Для каждого из t слагаемых надо выбрать один из k цветов, порядок не важен. Это число равно количеству сочетаний с повторениями из k по t, которое, в свою очередь, равно $\binom{k+t-1}{t}$.

Для подсчета биномиальных коэффициентов можно воспользоваться, например, формулой треугольника паскаля или формулой через факториалы.

Итого, получаем решение динамическим программированием. Пусть $a[n][p]$ — количество разбиений n на раскрашенные в k цветов слагаемые не больше p. Тогда

$$a[n][p] = \sum_{t=0}^{\lfloor n/p \rfloor} \binom{k+t-1}{t} \cdot a[n-tp][p-1].$$

Задача E. Пранк в IKEA

Автор и разработчик задачи: Николай Будин

Назовем две клетки, которые диван дополнительно занимает в разложенном состоянии, *необходимыми клетками* для этого дивана.

Будем рассматривать только те диваны, которые потенциально можно разложить. То есть те, у которых свободны обе необходимые клетки. Назовем такие диваны *интересными*.

Построим граф, вершинами которого будут являться интересные диваны. Проведем ребро между двумя вершинами, если два соответствующих дивана являются конфликтующими — если нельзя разложить их оба одновременно. То есть если множества их необходимых клеток пересекаются. Несложно заметить, что нам требуется найти максимальное независимое множество в таком графе (независимое множество — это множество вершин, никакие два их которых не соединены ребром).

Заметим, что каждая свободная клетка может быть необходимой максимум для двух интересных диванов. Если она является необходимой для трех или более интересных диванов, то есть два дивана, находящиеся по разные стороны от этой клетки. Не умаляя общности, они находятся слева и справа от пустой клетки. Тогда ни диван сверху от клетки, раскрывающийся вниз, ни диван снизу от клетки, раскрывающийся вверх, не могут быть интересными, потому что одна из их необходимых клеток точно будет занята диваном.

У каждого интересного дивана есть ровно две необходимые клетки, каждая клетка является необходимой максимум у двух интересных диванов, следовательно каждый интересный диван конфликтует максимум с двумя другими интересными диванами. Поэтому в построенном нами графе все вершины имеют степень не больше 2. А значит, каждая компонента связности графа является простым путем или циклом. В обоих случаях, поиск максимального независимого множества является тривиальной задачей: нужно взять вершины через одну, в случае пути нечетной длины — обязательно взяв концы пути.

Можно доказать еще несколько свойств построенного графа. Например, что возможны только циклы длины 2 или 4, однако задача решается и без этих знаний.

Итоговая асимптотика времени работы решения: $O(n \cdot m)$.

Задача F. СМС от МЧС

Автор задачи: Андрей Станкевич
Разработчик: Михаил Анопренко

В этой задаче требовалось проверить три простых условия и вывести необходимые сообщения. Пример кода, который решает задачу:

```
t1, v1 = map(int, input().split())
t2, v2 = map(int, input().split())

if t2 < 0 and v2 >= 10:
    print('A storm warning for tomorrow!')
    print('Be careful and stay home if possible!')
elif t2 < t1:
    print('MCHS warns! Low temperature is expected tomorrow.')
elif v2 > v1:
    print('MCHS warns! Strong wind is expected tomorrow.')
else:
    print('No message')
```

Задача G. Приготовление еды

Автор задачи: Ильдар Гайнуллин
Разработчик: Сергей Харгелия

Заметим, что ответ -1 тогда и только тогда, когда $\sum_{i=1}^{n} a_i$ нечётна (в таком случае просто никак нельзя разбить блюда на пары).

Пусть ответ не -1 и оптимальное разбиение содержит пары блюд $(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)$. Тогда в каждой паре надо выбрать порядок элементов так, чтобы i-е блюдо встречалось в своих парах на первой позиции либо $\lceil \frac{a_i}{2} \rceil$, либо $\lfloor \frac{a_i}{2} \rfloor$ раз.

Это утверждение эквивалентно следующему: рассмотрим неориентированный граф на n вершинах с рёбрами $\{(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)\} \setminus \{(a_i, b_i) : a_i = b_i\}$. Нужно для каждого ребра выбрать ориентацию так, чтобы выполнялось $\forall v : |indeg(v) - outdeg(v)| \leqslant 1$. Вершин с нечётной степенью чётное количество, пусть это вершины $v_1, v_2, \ldots, v_{2k}$. Добавим в граф фиктивные рёбра $(v_1, v_2), (v_3, v_4), \ldots, (v_{2k-1}, v_{2k})$. В получившемся графе степени всех вершин чётны, следовательно, его можно разбить на простые циклы (это следует из алгоритма построения Эйлерова цикла). Каждый неориентированный простой цикл превратим в ориентированный, после этого $\forall v : indeg(v) = outdeg(v)$. Так как каждой вершине инцидентно не более одного фиктивного ребра, то после удаления фиктивных рёбер будет выполняться $\forall v : |indeg(v) - outdeg(v)| \leqslant 1$.

Переберём для каждого блюда, $\lceil \frac{a_i}{2} \rceil$ или $\lfloor \frac{a_i}{2} \rfloor$ раз оно будет встречаться на первой позиции в своих парах в оптимальном ответе, обозначим эту величину для i-го блюда за b_i. Рассмотрим двудольный граф, где каждому блюду соответствует две вершины — это блюдо на первой/второй позиции в упорядоченных парах. Проведём рёбра вида $(i, 1) \to (j, 2)$ веса $c_{i,j}$, пропускной способностью $+\infty$. Также добавим в граф фиктивные вершины S и T, проведём рёбра вида $S \to (i, 1)$ веса 0 и пропускной способностью b_i и рёбра вида $(i, 2) \to T$ веса 0 и пропускной способностью $a_i - b_i$.

Тогда при фиксированных b_i оптимальный ответ равен минимальной стоимости максимального потока из S в T, при этом подходят только те наборы b, на которых величина максимального потока равна $\frac{1}{2} \sum_{i=1}^{n} a_i$. Взяв минимум по всем подходящим наборам b, получим ответ.

Максимальный поток минимальной стоимости можно искать, например, с помощью алгоритма Дейкстры с потенциалами. Рёбер в графе $O(n^2)$, также обозначим за U величину $\sum_{i=1}^{n} a_i$. Тогда при фиксированных b_i решение работает за $O(n^3 + Un^2) = O(n^3 \max a_i)$. Всего возможных наборов b не более, чем 2^n, то есть итоговая сложность решения $O(2^n n^3 \max a_i)$.

Задача H. Тяжкий труд

Автор задачи: Андрей Станкевич
Разработчик: Даниил Орешников

Для того, чтобы решить эту задачу, давайте разобьем числа в отрезке от l до r на подотрезки, каждый из которых имеет вид «от $\overline{\texttt{pref}0\ldots0}$ до $\overline{\texttt{pref}9\ldots9}$». Так, в частности, отрезок с 3400 до 3499 обозначим как `34**`.

Для примера рассмотрим $l = 3378$ и $r = 3621$. Отрезок с l по r разобьется на следующие подотрезки: `3378`, `3379`, `338*`, `339*`, `34**`, `35**`, `360*`, `361*`, `3620`, `3621`.

Не теряя общности, считаем, что l и r состоят из одинакового числа цифр (иначе дополним l ведущими нулями). Заметим, что (как видно на примере выше) сначала количество «`*`» в описаниях отрезков увеличивается, а затем уменьшается. Будем выписывать отрезки по очереди:

- пусть текущий отрезок должен начинаться в x, тогда обозначим за `pref` префикс x, полученный выкидыванием нулей из конца x;
- если $y = \overline{\texttt{pref}9\ldots9} \leqslant r$, добавим отрезок `pref*...*` и перейдем к $x = y + 1$;
- иначе мы находимся на этапе уменьшения числа звезд: рекурсивно рассмотрим префиксы $\overline{\texttt{pref}0}, \overline{\texttt{pref}1}, \ldots, \overline{\texttt{pref}9}$ и повторим предыдущий шаг для каждого из них.

Всего таких отрезков не более $2 \cdot 10 \cdot 18 < 400$, потому что в части с неуменьшающимся числом звезд в каждом следующем отрезке либо *1.* количество звезд такое же, как и в предыдущем, тогда предыдущая перед звездами цифра на 1 больше, либо *2.* количество звезд на один больше; и симметрично для части с неувеличивающимся числом звезд. Переходов второго типа не более 18, так как это максимальная длина числа, а переходов первого не более 9 в каждой цифре, потому что цифра не может увеличиться на 10.

Для каждого отрезка найти ответ независимо несложно — достаточно рассмотреть два случая:

1. ответ находится в фиксированном префиксе, для этого достаточно пройтись по префиксу и найти максимальную последовательность подряд идущих одинаковых цифр (и количество чисел с такой последовательностью будет равно длине отрезка);

2. ответ находится в «хвосте», тогда оптимум достигается тогда, когда хвост состоит из цифр, равных последней цифре из фиксированного префикса — длину ответа можно посчитать циклом, количество чисел будет равно 1;

После чего достаточно посмотреть на ответы в каждом отрезке и просуммировать найденные количества чисел, на которых достигается ответ, в отрезках, ответ в которых максимален.

Задача I. Точки и отрезки

Автор задачи: Андрей Чулков
Разработчик: Андрей Станкевич

Удивительно, но оказывается, что в описанной игре от действий игроков ничего не зависит. Количество ходов в игре будет одним и тем же, какие бы ходы ни делали игроки.

Докажем это, для этого нам необходимо воспользоваться двумя фактами из геометрии, которые мы приведем без доказательства:

1. Формула Эйлера. Для любого связного графа, уложенного на плоскости, число граней F, число вершин V и число ребер E связаны формулой $F + V - E = 2$.

2. В любом многоугольнике, содержащем хотя бы четыре вершины, есть внутренняя диагональ.

Рассмотрим ситуацию, в которой очередной игрок не может сделать ход. Рассмотрим граф: вершинами будут исходные точки, ребрами — проведенные отрезки. Так как проведенные отрезки не пересекаются, граф уложен на плоскости. По свойству (2) все грани, кроме возможно внешней, являются треугольниками (иначе можно сделать ход, проведя внутреннюю диагональ нетреугольной грани).

Значит число граней и число ребер удовлетворяют двум линейным уравнениям: $F + V - E = 2$ и $3(F - 1) + C = 2E$, где C — число сторон у внешней грани, они же — число вершин на выпуклой оболочке заданного множества точек.

Выражая из первого равенства $F = E + 2 - V$ и подставляя во второе, получаем: $E = 3V - 3 - C$. Таким образом, число ребер всегда одно и то же и не зависит от графа.

Из этого следует, что для решения задачи достаточно выяснить четность числа ребер. Если ребер нечетное число, выгодно ходить первым, а если четное — то вторым. Ходы же не важны, можно делать любой допустимый ход. Для этого, например, можно поддерживать множество допустимых отрезков, оно будет только уменьшаться. Чтобы сделать ход, перебираем их и если отрезок не подходит — удаляем его из множества, а если подходит — делаем ход.

Время работы этого решения $O(n^3)$, поскольку существует $O(n^2)$ потенциально проводимых отрезков, которые мы поддерживаем в множестве, и каждый из них проверяется на пересечение с $O(n)$ проведенными за время игры отрезками.

Задача J. Стрит

Автор задачи: Нияз Нигматуллин
Разработчик: Дмитрий Гнатюк

Отсортируем карты, лежащие на столе, и удалим повторы. Назовём полученный массив a.

Рассмотрим по очереди все карты, которые лежат на столе. Посчитаем количество стритов, в которых младшая используемая карта со стола это $a[i]$. Чему может быть равна младшая карта в таком стрите?

Во-первых, в этом подсчёте мы не должны использовать более маленькие карты со стола, поэтому младшая карта стрита должна быть хотя бы $a[i-1]+1$, а в случае $i=1$ она должна быть просто хотя бы 1.

Во-вторых, заметим, что со стола должны быть обязательно использовано хотя бы $m-s$ карт. Значит, карты $a[i], a[i+1], \ldots, a[i+m-s-1]$ должны существовать и «влезать» в собираемый стрит. Таким образом, если элемента $a[i+m-s-1]$ не существует, то такое i просто не может использоваться, а в противном случае максимальная карта стрита должна быть не меньше $a[i+m-s-1]$. А значит, минимальная карта стрита должна быть не меньше $a[i+m-s-1]-m+1$.

В-третьих, минимальная карта стрита должна быть не больше $a[i]$, чтобы $a[i]$ можно было использовать в стрите.

Наконец, минимальная карта стрита должна быть не больше $n-m+1$, потому что максимальная карта стрита должна существовать, то есть быть не больше n.

Итак, мы получили нижнюю и верхнюю границу для начала стрита. Если этот отрезок получился пустым, такое i следует пропустить. И ответ на задачу — это сумма длин таких отрезков по всем i.

Задача K. Новый уровень

Автор и разработчик: Александр Морозов

Для данной раскраски оставим только ребра графа, соединяющие пары соседних цветов.

Рассмотрим следующий алгоритм:

1. Если граф связный, то найденная раскраска является ответом, завершим алгоритм.
2. В противном случае, увеличим цвета всех вершин в компоненте связности первой вершины на один (по модулю k). Вернемся к (1).

Утверждение: В любой момент времени текущая раскраска является правильной раскраской.

Доказательство: Если появилась пара вершин u, v соединенных ребром, что $c_u = c_v$, то на прошлой итерации (до последнего увеличения цвета), цвета c_u и c_v отличались на один, а значит эта пара вершин была в одной компоненте связности рассмотренного графа, и их цвет должен был увеличиться одновременно. Противоречие.

Утверждение: Этот алгоритм завершится за конечное время.

Доказательство: Заметим, что через k итераций добавления в граф обязательно добавится новое ребро. А значит, через $\geqslant k(n-1)$ итераций прибавления единицы граф будет связным.

Чтобы соптимизировать алгоритм, будет поддерживать ребра исходящие из текущей компоненты в `std::set`, приоритетом ребра будет разница между цветами. Так можно каждый раз находить минимальное число прибавлений к цветам, через которое появится новое ребро, а затем нужно добавить все минимумы из кучи в граф с помощью СНМ.

Итоговая асимптотика равна $\mathcal{O}((n+m)\log(n+m))$

Задача L. Задача о прочном рюкзаке

Автор и разработчик: Сергей Копелиович

Разделим предметы на маленькие ($w_i \leqslant \frac{W}{2}$) и большие ($w_i > \frac{W}{2}$).

В оптимальном решении к задаче о рюкзаке (обозначим *opt*) максимум один большой предмет (больше не влезет в W). Переберём, какой i-й большой предмет мы возьмём, или что мы не возьмём ни одного большого предмета.

Для каждого такого i будем набирать рюкзак следующим образом. Сперва жадно возьмём i-й предмет, затем жадно в порядке убывания $\frac{cost_j}{w_j}$ будем набирать маленькие предметы.

Чтобы описанная выше идея работала за $O(n \log n)$, будем перебирать большие предметы в порядке убывания w_i, и воспользуемся методом двух указателей — с ростом i уменьшается w_i, а, значит, количество маленьких предметов, которые влезут в рюкзак, только растёт, это и есть второй указатель.

Доказательство корректности.

Пусть мы угадали i (большой предмет из *opt*). Других больших предметов в *opt* нет. Рассмотрим первый момент, когда наша «жадность» не возьмёт какой-то маленький предмет $j \in opt$. Обозначим W_{cur} сумму весов уже выбранных предметов. Мы знаем, что $w_j \leqslant \frac{W}{2}$ (маленький) и $W_{cur} + w_j > \frac{3}{2}W$ (не влез), поэтому имеем $W_{cur} > W$, следовательно, стоимость нашего решения уже строго больше стоимости *opt* (мы взяли все предметы из *opt* средней стоимости лучше чем $\frac{cost_j}{w_j}$ и ещё какие-то средней стоимости не хуже, а вес набрали больше).