Всероссийская олимпиада школьников по информатике 2020–2021

Региональный этап

Разбор задач

Условия задач, тесты, решения и разбор задач подготовили Николай Будин, Ильдар Гайнуллин, Дмитрий Гнатюк, Арсений Кириллов, Даниил Орешников, Михаил Путилин, Андрей Станкевич.

Ценные замечания по результатам тестирования задач сделали Никита Голиков, Илья Кирпичев, Максим Кузин, Ильдар Латыпов, Семен Степанов, Никита Сычев.

Разбор задачи «Два станка»

Автор задачи: Даниил Орешников

В первых четырех подзадачах достаточно легко найти конкретную формулу, которая позволяет найти ответ. Обозначим искомое количество деталей как r.

Подзадача 1

Если $a = 0$ и $x = 0$, то первый станок вообще нет смысла использовать. Введем в строй второй и получим ответ:

$$r = y \cdot \max(k - b, 0).$$

Важно не забыть, что время на запуск может оказаться больше k, и тогда количество деталей будет равно 0, для чего мы и берем max в формуле.

Подзадача 2

Если a и b равны нулю, то можно сразу запустить оба станка в работу и получить ответ:

$$r = (x + y) \cdot k.$$

Подзадача 3

Когда $a = b$, выгодно сначала ввести в строй наиболее производительный станок, и сразу после этого начать вводить в строй второй. Ответ тогда будет равен:

$$r = \max(x, y) \cdot \max(k - a, 0) + \min(x, y) \cdot \max(k - 2a, 0).$$

Тут снова надо не забыть о случаях, когда время на введение в строй больше доступного нам.

Подзадача 4

Если станки производят детали с одинаковой скоростью, то требуется ввести в строй сначала тот, на запуск которого уйдет меньше времени, чтобы детали раньше начали производиться. Получаем:

$$r = a \cdot \max(k - \min(a, b), 0) + a \cdot \max(k - a - b, 0).$$

Подзадача 5

Теперь посмотрим как решать общий случай. Переберем два варианта: какой станок будем вводить в строй первым. Если мы сначала введем в строй первый станок, ответ будет:

$$r_1 = x \cdot \max(k - a, 0) + y \cdot \max(k - a - b, 0).$$

Наоборот, если сначала ввести в строй второй, то получим:

$$r_2 = y \cdot \max(k - b, 0) + x \cdot \max(k - a - b, 0).$$

Чтобы получить ответ на задачу, достаточно взять максимум из двух этих значений:

$$r = \max(r_1, r_2).$$

Заметим, что, разумеется, решение пятой подзадачи решает также и первые четыре подзадачи.

Разбор задачи «Разбиение таблицы»

Автор задачи: Даниил Орешников

Подзадача 1

Переберем линию разделения. Затем за $n \cdot m$ посчитаем сумму в разделенной таблице. Выберем лучший разрез. В сумме решение работает за $n \cdot m \cdot (n + m)$.

Подзадача 2

Заметим, что при перемещении линии разреза на 1 в каждой из половин добавляется или удаляется только $O(n)$ или $O(m)$ клеток. Значит можно перебрать все варианты за $O(nm)$.

Подзадачи 3 и 4

Зафиксируем какой-нибудь разрез по вертикали перед столбцом k, посчитаем сумму в полученном прямоугольнике, используя сумму арифметической прогрессии:

$$a_1 + a_2 + a_3 + \ldots + a_x = \frac{(a_1 + a_x)}{2} \cdot x$$

Применяя, получаем:

$$1+2+\ldots+k+(m+1)+(m+2)+\ldots+(m+k)+\ldots+(m\cdot(n-1)+1)+(m\cdot(n-1)+2)+\ldots+(m\cdot(n-1)+k) =$$

$$= \frac{1+k}{2} \cdot k + \frac{(m+1)+(m+k))}{2} \cdot k + \ldots + \frac{((m \cdot (n-1)+1)+(m \cdot (n-1)+k)}{2} \cdot k$$

Полученная сумма также является арифметической прогрессией с шагом $m \cdot k$, значит она равна:

$$\frac{\frac{1+k}{2} \cdot k + \frac{((m\cdot(n-1)+1)+(m\cdot(n-1)+k)}{2} \cdot k}{2} \cdot n$$

Сумму в горизонтальном разрезе найдем по более простой формуле:

$$1+2+\ldots+m+(m+1)+(m+2)+\ldots+(m+m)+\ldots+(m\cdot(k-1)+1)+(m\cdot(k-1)+2)+\ldots+(m\cdot(k-1)+m)$$

$$\frac{1+m \cdot k}{2} \cdot mk$$

Таким образом, за $O(1)$ можно посчитать сумму при фиксированном разрезе, а значит, можно найти лучший разрез за $(n + m)$.

Подзадача 5

Здесь арифметическая прогрессия одномерная, и надо найти такое k, чтобы

$$\frac{1+k}{2} \cdot k \approx \frac{1+m}{4} \cdot m$$

Решение 1

Упрощая, получим квадратное уравнение на k, и найдем лучшее k за $O(1)$. Из-за округлений возможна погрешность, поэтому нужно перебрать ответ в диапазоне ± 1.

Решение 2

Можно найти лучшее k, используя двоичный поиск (сумма первых k значений возрастает при увеличении k).

Подзадача 6

Совместим идеи в подзадачах 4 и 5: Посчитаем сумму при фиксированной границе раздела:

$$\frac{\frac{1+k}{2} \cdot k + \frac{((m \cdot (n-1)+1)+(m \cdot (n-1)+k)}{2} \cdot k}{2} \cdot n \approx \frac{1+n \cdot m}{4} \cdot nm$$

Решение 1 за $O(1)$ на запрос

Упрощая, получим квадратное уравнение на k, и найдем лучшее k с точностью до ± 1, так как знак приблизительно равно.

Решение 2 за $\log_2 max(n, m)$ на запрос

Можно найти лучшее k, используя двоичный поиск, так как значение слева возрастает при увеличении k.

Разбор задачи «Изменённая ДНК»

Автор задачи: Олег Христенко

Общая идея

Для того, чтобы получить минимальную длину, всегда выгодно удалить какой-то элемент, а чтобы получить максимальную длину, нужно добавить какой-то элемент. Докажем это.

Для минимума. Пусть мы заменили какой-то символ. Вместо этого удалим этот символ. Длина каждого из соседних блоков не изменится, либо они сольются в один. В любом случае длина закодированной строки получится меньше или такой же, как при замене. Аналогично, вместо добавления символа всегда выгоднее удалить символ перед ним.

Для максимума. Пусть мы заменили какой-то символ. Тогда вместо этого добавим в эту позицию тот символ, на который мы заменили. Длина каждого из соседних блоков не уменьшится, значит длина закодированной версии полученной строки будет не меньше. Аналогично вместо удаления символа всегда выгоднее добавить символ перед ним.

Перейдем теперь к рассмотрению того, как решить, какую именно операцию сделать.

Подзадачи 1 и 2

Для того, чтобы решить первые две подзадачи, достаточно перебрать место, в которое нужно вставить новый элемент, или удалить существующий. После фиксирования изменения, можно посчитать длину закодированной строки полным проходом по ней. Асимптотика этого решения $O(L^2)$.

Подзадача 3

Для того, чтобы решить третью подзадачу, нужно заметить, что каждый блок одинаковых символов изменился не сильно. Поэтому после изменения символа достаточно для каждого исходного блока посчитать новую длину этого блока или новых блоков, которые появились после изменения. Асимптотика решения $O(n \cdot L)$.

Подзадача 4

Для того, чтобы решить четвертую подзадачу, нужно заметить, что после изменения меняется только блок, в котором произошло изменение, или соседние с ним блоки. Для всех остальных блоков длина не меняется, и её можно не пересчитывать. Асимптотика решения $O(L)$.

Подзадача 5

Для того, чтобы решить пятую подзадачу, нужно научиться понимать, как правильно удалять и добавлять символы.

Если в строке есть блок длины 1, соседние блоки которого содержат одинаковые символы, то при удалении этого символа эти блоки объединятся в один, и длина закодированной строки уменьшится. Если в строке есть блок длины 1, то при его удалении длина строки уменьшится на 1. Если в строке есть блок длины 2, то при удалении символа из него, в его кодировке больше не нужно будет писать число, и длина строки уменьшится на 1. Если в строке есть блок длины 10^x, то при удалении символа из него длина числа уменьшится на 1. Это все возможные способы уменьшения длины при удалении символа.

При добавлении символа можно поставить его в начало для создания нового блока длины 1, чтобы длина строки увеличилась на 1. Также можно какой-то блок разделить на два, если добавить другой символ внутри блока. Для блока длины до 20 достаточно перебрать, где именно будет новый символ. Если длина блока от $2 \cdot 10^x$ до 10^{x+1}, то из него можно выделить блок размера 10^x, и увеличить длину строки на $x+3$. Если длина блока от 10^x+10^{x-1} до $2 \cdot 10^x$, то из него можно выделить блок длины 10^x, и увеличить длину строки на $x+2$. Если длина блока от 10^x до $10^x + 10^{x-1}$, то из него можно выделить блок длины $8 \cdot 10^{x-1}$, и увеличить длину строки на $x+1$. Это все возможные способы увеличить длину строки при добавлении символа.

Для каждого блока проверим все оптимальные способы добавления и удаления символа и выберем оптимальный способ среди всех. Так как для каждого блока оптимальных способов всего $O(1)$, то асимптотика решения $O(n)$.

Разбор задачи «Антенна»

Авторы задачи: Даниил Орешников, Геннадий Короткевич

Подзадача 1

Для того, чтобы решить первую подзадачу, можно перебрать, в каком порядке нужно соединить фрагменты. Всего существует $n!$ таких вариантов. Для каждого варианта можно вычислить позиции всех перекладин и проверить, подходит ли такой вариант.

Подзадача 2

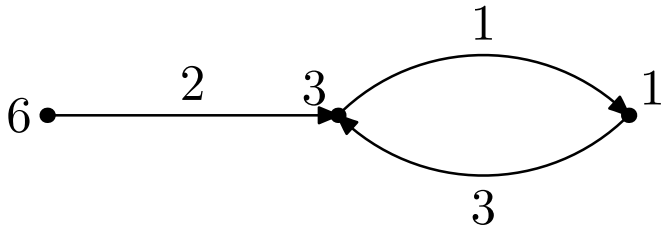
Для того, чтобы решить вторую подзадачу, нужно заметить, что если на одном из фрагментов присутствует более одной перекладины, то, если существует способ правильно собрать антенну, расстояние между соседними перекладинами должно быть равно расстоянию между перекладинами на этом фрагменте. Поэтому можно заранее проверить, что расстояния между соседними перекладинами в каждом отдельно взятом фрагменте равны. После чего можно оставить в каждом фрагменте только первую и последнюю перекладины. Затем, как в предыдущей подзадаче, перебрать порядок фрагментов и проверить, подходит ли он.

Общая идея

Как во второй подзадаче, оставим на каждом фрагменте только первую и последнюю перекладины. Пусть на i-м фрагменте первая перекладина находится на расстоянии l_i от начала, а последняя — на расстоянии r_i от конца. Научимся проверять, существует ли порядок фрагментов, при котором

расстояние между соседними перекладинами будет равно d. Пусть такой порядок существует, обозначим его $q_1, q_2, \ldots, q_n$. Тогда должно выполняться $r_{q_i} + l_{q_{i+1}} = d$ $(1 \leqslant i \leqslant n-1)$, значит $l_{q_{i+1}} = d - r_{q_i}$. Рассмотрим граф, вершинами которого являются все целые числа. Проведем ориентированное ребро из l_i в $d - r_i$ для всех i. Заметим, что искомый порядок существует тогда и только тогда, когда в построенном графе существует Эйлеров путь. Поиск Эйлерова пути (и проверка существования) является стандартным алгоритмом.

Например, в первом тесте из примеров, $l_1 = 3$, $r_1 = 4$, $l_2 = 6$, $r_2 = 2$, $l_3 = 1$, $r_3 = 2$. В ответе на тест, $d = 5$. Построим граф по этим данным:



Рядом с вершинами написаны соответствующие им числа, а рядом с ребрами — номера соответствующих им фрагментов. Видно, что последовательность ребер 2, 1, 3 является Эйлеровым путем.

Для решения оставшихся подзадач, мы будем находить некоторое множество значений d, каждое из которых будем проверять за линейное время алгоритмом из предыдущего абзаца.

Подзадача 4

Чтобы решить четвертую подзадачу, нужно заметить, что по принципу Дирихле всегда будет существовать хотя бы один фрагмент, содержащий хотя бы две перекладины. Поэтому, если искомый порядок существует, расстояние d между соседними перекладинами должно быть равно расстоянию между соседними перекладинами на этом фрагменте. Воспользуемся алгоритмом и проверим, подходит ли данное d.

Подзадача 3

Чтобы решить третью подзадачу, можно перебрать два варианта:

- Если фрагмент номер 1 не будет стоять самым последним, то можно перебрать номер фрагмента i, который будет стоять сразу после фрагмента 1, и проверить $d = r_1 + l_i$.
- Если фрагмент номер 1 будет стоять самым последним, то фрагмент номер 2 не будет стоять последним. Можно перебрать номер фрагмента i, который будет стоять после фрагмента 2, и проверить $d = r_2 + l_i$.

Таким образом, мы проверим $O(n)$ вариантов, каждый за время $O(n)$. Итого, решение работает за $O(n^2)$.

Подзадача 5

Чтобы решить пятую подзадачу, можно проверить все значения d от 0 до 200. Дополнительно ускорить решение можно, проверяя только значения d от 0 до $\lfloor \frac{100 \cdot n}{n-1} \rfloor$.

Подзадача 6

Наконец, чтобы решить задачу на полный балл, нужно было научиться проверять только $O(1)$ различных вариантов d. Рассмотрим пары $(l_{q_i}, r_{q_{i+1}})$ и отсортируем их в порядке возрастания l. Заметим, что так как суммы значений в каждой паре равны, значения r будут убывать. Среди всех

значений l в рассмотренных парах не присутствует ровно одно, соответствующее первому фрагменту. Аналогично, среди всех значений r отсутствует значение, соответствующее последнему фрагменту. Таким образом, можно отсортировать все значения l в порядке возрастания, все значения r в порядке убывания и проверить 4 варианта значения d: сумма одного из двух минимальных значений l и одного из двух максимальных значений r.

Разбор задачи «Календарь на Альфе Центавра»

Автор задачи: Андрей Станкевич

Заметим ключевую идею. Пусть мы нашли x — номер дня от начала летосчисления до заданного во вводе дня. С первого дня первого месяца первого года (который, как мы знаем, имел обозначение «a», прошло $x - 1$ дней. Каждые w дней повторялся день с обозначением «a», поэтому на самом деле нас интересует величина $y = (x - 1) \bmod w$.

Заметим, что если $y = 0$, то день имеет обозначение «a», если $y = 1$, то «b», и так далее. Чтобы получить по сдвигу относительно «a» букву в английском алфавите, можно воспользоваться возможностями языка программирования. В C++ это сделать проще всего:

```
cout << 'a' + y << endl;
```

В других языках могут потребоваться функции приведения типа. Например, в Паскале:

```
writeln(chr(ord('a') + y));
```

В языке Python:

```
print(chr(ord('a') + y))
```

Осталось разобраться, как найти x.

Подзадача 1

В первой подзадаче год состоит из одного дня. Поэтому $i = 1$ и $j = 1$, а значит $x = k$.

Подзадача 2

Во второй подзадаче $m = 1$, то есть месяц всего один. $x = (k - 1) \cdot d + i$. Заметим, что в этой подзадаче благодаря тому, что k небольшое, достаточно 32-битного типа данных.

Подзадача 3

В третьей подзадаче речь все время идет о первом дне первого месяца. $x = (k - 1) \cdot d \cdot m + 1$.

Подзадача 4

В четвертой подзадаче $k = 1$. Год учитывать не надо, надо учесть только день и месяц. $x = (j - 1) \cdot d + i$.

Подзадачи 5 и 6

Наконец получим общую формулу: $x = (k - 1) \cdot m \cdot d + (j - 1) \cdot d + i$. Отличие пятой подзадачи в том, что в ней k невелико и поэтому можно использовать 32-битный тип данных.

В заключение отметим, что различные способы итерациями перебирать дни могли проходить некоторые подзадачи, например подзадачи 1, 2, 3, 5.

Разбор задачи «Числа»

Автор задачи: Андрей Станкевич

Подзадача 1

Заметим, что $k = 0$ означает, что число должно состоять из всех одинаковых цифр. В этой подзадаче можно просто увеличивать x, пока все цифры числа не будут одинаковы. Число 111 111 обладает этим свойством и не меньше всех возможных чисел во вводе, значит будет сделано не больше чем такое количество шагов.

Подзадача 2

В этой подзадаче увеличение шагами по одному уже не приводит к успеху. Для получения баллов за эту подзадачу нужно разобраться, как устроено минимальное число, большее заданного, состоящее из одинаковых цифр.

Первую цифру числа нельзя уменьшить, посмотрим, можно ли её оставить такой же. Пойдем по числу от старших цифр к младшим, пока они равны первой цифре числа f. Если в какой-то момент очередная цифра цифра отличается от первой, посмотрим меньше она или больше. Если она оказывается меньше первой цифры числа, то можно просто заменить все цифры на f, получив большее число из всех одинаковых цифр. Если же она оказывается больше, то придется увеличить первую цифру. Это можно сделать, если она не равна 9. В этом случае заполним число цифрами $f + 1$. Иначе надо взять число из всех единиц, имеющее на одну цифру в своей десятичной записи больше.

Подзадача 3

В этой задаче снова маленькое значение x, и можно увеличивать его, пока все его цифры, кроме, может быть, одной не станут равны. На этот раз верхним порогом выступает число 100 000. Единственная трудность этой подзадачи — техническая, проверить, что все цифры, кроме не более чем k, равны.

Подзадача 4

На самом деле полное решение задачи даже проще разбора частных случаев предыдущих подзадач. Заметим, что искомое число имеет не более 18 десятичных цифр. Значит всего подходящих чисел $18 \cdot 9$ для $k = 0$ и не больше $18 \cdot 10 \cdot 18 \cdot 10$ для $k = 1$ (во втором случае мы перебираем длину, основную цифру, позицию отличающейся и саму отличающуюся). Переберем все потенциальные ответы и из тех, которые не меньше x, выберем минимальный.

Код на C++, который строит число из цифр d длины len, на позиции pos ставится цифра $d2$:

```
long long num(int len, int d, int pos, int d2) {
    long long res = 0;
    for (int i = 0; i < len; i++) {
        if (i == pos) {
            res = res * 10 + d2;
        } else {
            res = res * 10 + d;
        }
    }
    return res;
}
```

Код на C++, который перебирает все подходящие числа для $k = 1$:

```
for (int i = 1; i <= 18; i++) {
    for (int d = 0; d < 10; d++) {
        for (int p = 0; p < i; p++) {
            for (int d2 = 0; d2 < 10; d2++) {
                long long y = num(i, d, p, d2);
                if (y >= x && (ans == -1 || y < ans)) {
                    ans = y;
                }
            }
        }
    }
}
```

Разбор задачи «Хорошие раскраски»

Автор задачи: Ильдар Гайнуллин

Эта задача — довольно нетрадиционная для олимпиад по информатике. Вместо конкретного конструктивного или алгоритмического решения участникам предлагалось поэкспериментировать с эвристиками и перебором.

Прежде чем разобрать основные идеи, отметим, что конструктивного решения в этой задаче, скорее всего, нет. На это наводит следующая мысль: конструктивный паттерн, который решал бы задачу для приведенных ограничений, мог бы быть распространен на бесконечное поле. А можно доказать — кстати, интересное олимпиадное упражнение по математике — что раскрасить таким образом в 2 или 3 цвета бесконечное поле невозможно.

Отметим, что в задаче очень маленькие ограничения и потестовая оценка. В принципе, можно локально запустить решение для всех возможных входов (их 200) и отправить в систему результаты предподсчета. Практически любое продвижение для больших полей вознаграждается баллами.

Перейдем теперь к рассмотрению основных идей переборных решений и решений с помощью локальных оптимизаций.

Перебор

Самое простое решение это перебор за $\mathcal{O}(c^{n \cdot m})$, можно перебирать все раскраски простой рекурсией: перебирать цвета клеток в порядке сортировки по строкам и при равенстве по столбцам, и проверять, подходит ли итоговое поле. Чтобы оптимизировать это решение, в процессе перебора подходящих раскрасок, можно проверять, что среди клеток, которым уже проставлены значения, нет плохих четверок. Это решение уже может пройти все тесты с $c = 2$ и значительное число тестов с $c = 3$. Чтобы еще больше оптимизировать решение, можно перебирать цвета для клеток в случайном порядке.

Дополнительная оптимизация, которая сильно ускоряет перебор, — добавление «*мемоизации*». А именно, давайте запоминать некоторую информацию про текущую раскраску, которая точно не приведет к ответу (так как такая же комбинация перебиралась ранее, и ответ найден не был). Например, можно хранить множество троек (c, y_1, y_2), для которых среди уже покрашенных клеток найдется такое x, что $a_{x,y_1} = a_{x,y_2} = c$, и проверять, что такое множество не встречалось, когда мы начинаем красить первую клетку в определенной строке.

Прошлое решение работает заметно лучше при $m < n$, в противном случае можно поменять местами n и m и транспонировать поле для ответа. С этими оптимизациями можно пройти почти все тесты.

Локальные оптимизации

Другим возможным решением является метод локальных оптимизаций.

Исходно проставим каждой клетке случайный цвет $1 \dots c$. Затем будем выбирать случайную клетку и изменять ей цвет, если при этом уменьшится *стоимость* поля.

Стоимостью поля может быть почти любая функция, которая будет равна нулю для хороших полей. Например, за нее можно взять количество таких четверок x_1, x_2, y_1, y_2, что $1 \leqslant x_1 < x_2 \leqslant n$, $1 \leqslant y_1 < y_2 \leqslant m$, и клетки (x_1, y_1), (x_2, y_1), (x_1, y_2) и (x_2, y_2) покрашены в одинаковый цвет. Когда никакое изменение цвета не приведет к уменьшению ответа, можно завершить процесс.

Для большинства случайных раскрасок стоимость в конечном итоге будет равна маленькому числу. А если с исходной раскраской «*повезёт*», то стоимость будет равна нулю, и поле сойдется к хорошему полю. Отсюда выходит следующее решение: будем случайно генерировать исходные поля, запускать описанный выше процесс, пока стоимость уменьшается, и проверять что в конце получился 0. Если получился, то можно вывести ответ. Иначе генерируем исходное поле заново.

Это решение способно пройти все тесты с ограничениями этой задачи меньше чем за секунду.

Разбор задачи «A+B»

Автор задачи: Михаил Путилин

Подзадача 1

Первая подзадача решается перебором $n!$ перестановок.

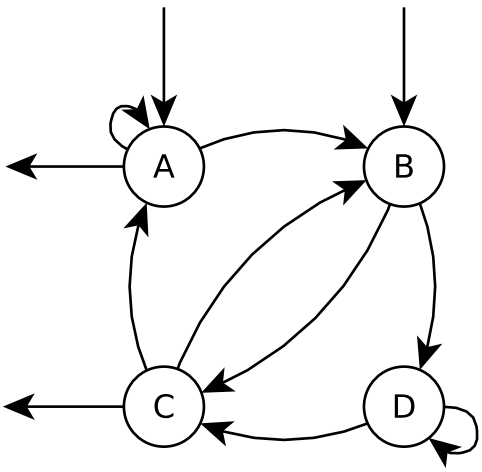
Подзадача 2

Каждый столбец можно охарактеризовать тремя параметрами (обозначим цифры в нём за a_i, b_i, c_i):

1. Есть ли в нём цифра ноль (это влияет на то, может ли он быть первым).
2. Требуется ли ему перенос из столбца справа, в зависимости от того, верно $(a_i+b_i) \mod 10 = c_i$ или $(a_i+b_i+1) \mod 10 = c_i$. Если ни одно из этих равенств не выполнено, то ответ 0. Обозначим этот параметр $needCarry_i \in \{0, 1\}$.
3. Создаёт ли этот столбец перенос через разряд. Это так, когда $a_i + b_i + needCarry_i \geqslant 10$. Обозначим этот столбец $makeCarry_i \in \{0, 1\}$.

Будем выбирать перестановку столбцов слева направо. Первым может идти столбец без нулей и не создающий переноса. Если после столбца i идёт столбец j, то $needCarry_i = makeCarry_j$. Для последнего столбца $needCarry_i = 0$. Получается граф, в котором нужно найти число гамильтоновых путей. Это можно сделать при помощи динамического программирования за $O(2^n n^2)$, решив, таким образом, вторую подзадачу.

Подзадачи 3 и 4



В полученном графе есть четыре вида вершин в зависимости от значений $needCarry_i$ и $makeCarry_i$. Обозначим эти виды следующим образом:

- Тип A: $needCarry_i = 0$, $makeCarry_i = 0$
- Тип B: $needCarry_i = 1$, $makeCarry_i = 0$
- Тип C: $needCarry_i = 0$, $makeCarry_i = 1$
- Тип D: $needCarry_i = 1$, $makeCarry_i = 1$

На рисунке показано, какие рёбра есть в графе, а также какие вершины могут быть начальными и конечными.

Посчитать гамильтоновы пути в таком графе можно динамикой за $O(n^4)$: $d_x[A][B][C][D]$ — число путей, которые заканчиваются в вершине вида x и проходят через соответствующее число вершин каждого вида. Это решает подзадачу 3.

Чтобы решить подзадачу 4, нужно учесть, что нельзя начинать со столбца с нулём. Это влияет только на базу динамики: в $d_A[1][0][0][0]$ и $d_B[0][1][0][0]$ нужно записать количество соответствующих столбцов, в которых нет нулей.

Подзадачи 5 и 6

На любом пути в таком графе, который начинается в A или B, количество вершин вида B не более чем на один превосходит количество вершин вида C. Это позволяет сократить число состояний в динамике до $O(n^3)$, решив, таким образом, подзадачи 5 и 6.

Подзадача 7

Обозначим за A, B, C, D количество вершин каждого вида. Любой интересующий нас путь выглядит следующим образом:

$$\_B\_C\_B\_C\_B\_C\ldots B\_C\_$$

При этом вершины A расположены в промежутках перед B, а также в последнем. Вершины D расположены в промежутках после B. Кроме того, если $B \neq C$, то ответ 0.

Если $B = C = 0$ и $D > 0$, то ответ 0. Если $B = C = 0$ и $D = 0$, то ответ $A!$.

Если $B = C > 0$, то ответом будет произведение следующих чисел:

1. Количество способов расставить вершины B по местам: $B!$
2. Количество способов расставить вершины C по местам: $B!$
3. Количество способов расставить A вершин в $B + 1$ промежуток с учётом порядка: $\frac{(A+B)!}{B!}$.
4. Количество способов расставить D вершин в B промежутков с учётом порядка: $\frac{(D+B-1)!}{(B-1)!}$.

Ответ равен $(A + B)!(D + B - 1)!B$. Это решение за $O(n)$ проходит подзадачу 7.

Подзадача 8

Осталось учесть, что среди вершин вида A и B может быть сколько-то вершин, с которых нельзя начинать (столбцы с нулями). Обозначим эти количества за A_0 и B_0.

Как и в прошлой подзадаче, если $B = C = 0$ и $D > 0$, то ответ 0. Если $B = C = 0$ и $D = 0$, то ответ $(A - 1)! \cdot (A - A_0)$.

Далее $B = C > 0$. Сначала отдельно посчитаем:

- ans_B — число путей, которые начинаются с вершины вида B. Это означает, что промежутков, в которые можно ставить вершины A, не $B + 1$, а B. Тогда $ans_B = B! \cdot B! \cdot \frac{(A+B-1)!}{(B-1)!} \cdot \frac{(D+B-1)!}{(B-1)!} = (A + B - 1)!(D + B - 1)!B^2$

- ans_A — число путей, которые начинаются с вершины вида A. $ans_A = (A + B)!(D + B - 1)!B - ans_B$

Ответ равен $ans_A \cdot \frac{A-A_0}{A} + ans_B \cdot \frac{B-B_0}{B}$.