A. Разные фломастеры

Автор задачи: Денис Остапенко
Количество сдавших команд: 15 из 23 (65%)

Для решения данной задачи необходимо в первую очередь понять, какие условия должны быть выполнены, чтобы можно было сделать заключение о том, что гипотеза подтверждена, опровергнута, либо данных недостаточно. Отсутствие части или всех данных про некоторые фломастеры не всегда означает недостаточность данных для подтверждения или опровержения гипотезы. Например, если про два фломастера известны их цвета и вкусы, и они совпадают, то гипотезу можно признать опровергнутой, даже если про какие-то другие фломастеры вообще ничего не известно. С другой стороны, если, например, известны цвета всех фломастеров, и они все разные, то гипотезу можно признать подтверждённой, даже если вкусы части или всех фломастеров не известны.

Сформулируем условие, которое должно выполниться, чтобы можно было признать гипотезу опровергнутой. Такое условие всего одно: должна найтись такая пара фломастеров, что у обоих известны и цвет, и вкус, и при этом цвет и вкус первого фломастера равны цвету и вкусу второго. Для того, чтобы сделать заключение о недостаточности данных, должно выполниться одно из следующих условий:

- должен найтись фломастер, у которого не известен ни цвет, ни вкус
- должна найтись такая пара фломастеров, что
  - их цвета совпадают, а вкус как минимум одного из них неизвестен
  - их вкусы совпадают, а цвет как минимум одного из них неизвестен
  - у первого не известен цвет, а у второго — вкус
  - у первого не известен вкус, а у второго — цвет

Если ни условие для признания гипотезы опровергнутой, ни условия для заключения о недостаточности данных, не выполнены, то можно признать гипотезу подтверждённой.

Реализовать проверку описанных выше условий можно следующим образом. Создадим массив размером N, элементами которого будут структуры с двумя строковыми полями. В этом массиве мы будем хранить полученную в ходе экспериментов информацию про фломастеры, каждый элемент массива — это фломастер с соответствующим номером. Пустая строка в одном из полей будет обозначать, что соответствующее свойство (цвет или вкус) соответствующего фломастера неизвестно. При чтении информации о каждом эксперименте следует записывать его результат в соответствующее поле соответствующего элемента массива. После завершения обработки результатов всех экспериментов, можно использовать получившийся массив для проверки описанных выше условий.

Сначала запустим двойной цикл по массиву фломастеров (внешний — i от 0 до $N-2$, внутренний — j от $i+1$ до $N-1$) и для каждой пары фломастеров проверим описанное выше условие опровержения гипотезы. Если оно выполнилось хотя бы для одной пары, то можно сразу выводить «`Busted`». Затем, запустим одиночный цикл по массиву фломастеров и проверим первое из описанных выше условий недостаточности данных. Если для одного из фломастеров оно выполнилось, то можно сразу выводить «`Not enough data`». Наконец, снова запустим двойной цикл по массиву фломастеров и для каждой пары проверим четыре последних условия недостаточности данных. Если хотя бы одно из них выполнится для хотя бы одной пары фломастеров, то можно тоже сразу выводить «`Not enough data`». Если после завершения всех циклов вывод так и не был сделан, то следует вывести «`Confirmed`».

При желании можно выполнить все проверки с помощью одного двойного цикла. Но в любом случае важно помнить, что вывод «`Busted`» более приоритетный, чем вывод «`Not enough data`». Поэтому, если было обнаружено, что выполняется одно из условий недостаточности данных, нельзя сразу выводить ответ, ведь позднее может обнаружиться, что также выполняется условие опровержения гипотезы.

B. Найти перестановку

Автор задачи: Виктория Рувинская
Количество сдавших команд: 13 из 23 (57%)

Задача решается перебором различных перестановок, удовлетворяющих заданным условиям. Конкретнее, перебираем все возможные значения первого числа перестановки от 1 до N и для каждого такого значения пробуем подсчитать все остальные элементы слева направо. Не сложно убедиться, что значение первого элемента однозначно определяет значение всех остальных.

Если это удалось сделать, то есть найти каждый элемент, начиная со второго, так, чтобы все элементы были в диапазоне от 1 до N, никакие элементы не повторялись в текущей перестановке, а также, чтобы суммы соседних элементов были равны заданным в условии, то завершаем перебор и выводим полученную перестановку. Контролировать, что каждое число от 1 до N встретилось в полученной перестановке только один раз, можно с помощью массива флагов размера N, в котором i-ому элементу будет присваиваться `true`, как только число $i+1$ встретится в перестановке. Перед присвоением следует проверить, что соответствующий элемент был равен `false`.

Если для очередного значения первого элемента не удалось создать перестановку, переходим к следующему значению первого элементу. Если к концу перебора не удалось построить перестановку, выводим -1.

C. Межпланетный интернет

Автор задачи: Владимир Мшанецкий
Количество сдавших команд: 18 из 23 (78%)

Алгоритм решения данной задачи предельно прост:

1. Считать размер в байтах и скорость в битах
2. Перевести размер в биты, умножив его на 8
3. Определить общее время загрузки в секундах, разделив размер на скорость с округлением к ближайшему целому по правилам математики (то есть, округляя 0.5 вверх)
4. Перевести общее время в секундах в часы, минуты и секунды путём деления на 60
5. Вывести часы, минуты и секунды в описанном в задаче формате

Тем не менее, реализация приведённых выше действий вызвала трудности даже у некоторых команд, которые в остальном показали достаточно высокий результат. Рассмотрим основные моменты, которые могут вызвать затруднения.

Самым главным при реализации решения данной задачи является правильный выбор типов переменных, используемых для хранения входных данных, промежуточных результатов вычислений и окончательного ответа. Согласно условию, размер файла S и скорость интернета T не превышают 10^{17}. Для хранения чисел порядка 10^{17} необходимо использовать 64-битные целочисленные переменные (`long long` в C++, `long` в Java). В 32-битные типы (`int`) такие большие числа не помещаются. В случае, когда скорость интернета T равна 1, общее время загрузки в секундах $8 \cdot S \div T$ может достигать $8 \cdot 10^{17}$, поэтому для его хранения тоже нужна 64-битная переменная. Поскольку $8 \cdot 10^{17}$ секунд — это $8 \cdot 10^{17} \div 3600 \approx 2.2 \cdot 10^{14}$ часов, для хранения часовой компоненты времени загрузки тоже необходимо использовать 64-битую переменную. Кроме того, если в в процессе перевода секунд в минуты и часы в переменную для минут сохраняется промежуточный результат до вычисления остатка от деления на 60, то эта переменная тоже должна быть 64-битной. Проще говоря, можно просто все переменные сделать 64-битными целыми, хотя это может не являться необходимым для некоторых из них.

Отдельно следует рассмотреть использование чисел с плавающей точкой. Хотя переменные типа чисел с плавающей точкой потенциально могут хранить числа в широких диапазонах, эти типы имеют ограниченную точность. Точности типов `double` и `float` не достаточно для хранения чисел порядка 10^{17}, поэтому при приведении таких чисел к указанным типам, они округляются, теряя точные значения нескольких последних десятичных цифр. В результате, вычисленный ответ получается с погрешностью в несколько секунд и поэтому является неправильным. Тем не менее, эту задачу можно решить с использованием чисел с плавающей точкой, если использовать тип `long double` в C++. При использовании компилятора GCC на процессорах x86 тип `long double` содержит 64 бита мантиссы (против 52 бит у `double`), чего достаточно для хранения чисел порядка 10^{17} без погрешности.

Кроме того, поскольку числа порядка 10^{17} находятся близко к верхней границе диапазона значений переменных, следует проявлять осторожность при написании математических выражений, чтобы промежуточные результаты вычислений не вышли за допустимый диапазон значений. Например, если попытаться вычислить количество часов как $(8 \cdot S) \div (3600 \cdot T)$, то промежуточный результат выражения $3600 \cdot T$ может достигнуть $3600 \cdot 10^{17} = 3.6 \cdot 10^{20}$, что не помещается ни в один из стандартных целочисленных типов.

Вторым важным моментом при реализации решения этой задачи является правильная реализация деления с округлением к ближайшему целому. При использовании переменных типа `long double` в C++ округление можно реализовать с помощью стандартных функций `round()` или `llround()`. При использовании целочисленных переменных округление реализуется немного сложнее.

Как известно, при делении `a / b`, где `a` и `b` — это целочисленные переменные, результат округляется в меньшую сторону. Это можно использовать, чтобы записать выражении так, чтобы вычислить результат деления с округлением в нужную сторону:

- для округления в меньшую сторону нужно писать `a / b`

- для округления в большую сторону нужно писать `(a + b - 1) / b`
- для округления к ближайшему целому нужно писать `(a + b / 2) / b`

Разобраться, как и почему работают данные выражения, предлагается читателю. Таким образом, в данной задаче общее количество времени в секундах можно вычислить как `(8 * S + T / 2) / T`.

Отдельно следует обратить внимание, что, если $8 \cdot S < T$, то ответ может быть как «`1s`», так и «`0s`». Это зависит от того, выполняется ли неравенство $8 \cdot S < T \div 2$. Вообще говоря, при правильной реализации деления с округлением случай $8 \cdot S < T$ рассматривать отдельно не нужно, но некоторые команды рассматривали его отдельно и всегда выводили «`1s`» или «`0s`».

Также следует отметить, что гораздо проще и надёжнее реализовать округление изначально при вычислении общего количества секунд, которые затем перевести в минуты и часы, чем вычислить часы, минуты и секунды с помощью обычного деления с округлением в меньшую сторону, а затем прибавлять к ним единицу, если необходимо. В последнем варианте проще допустить ошибку.

Третьим важным моментом при решении данной задачи является внимательное чтение условия. Во-первых, следует в точности следовать требуемому формату вывода времени. Согласно описанию формата выходных данных, опускать нужно только начальные компоненты времени, равные нулю. Например, если загрузка будет длиться ровно 42 часа, следует вывести «`42h 0m 0s`», а не просто «`42h`». Кроме того, следует проверить, что программа выводит что-то, если ответ равен нулю.

Во-вторых, нужно обратить внимание, что нигде в условии не говорится, что скорость интернета T всегда делится на 8. То есть, скорость интернета, данная в битах в секунду, может не быть целым количеством байт в секунду. Следовательно, если вместо умножения S на 8 выполнять деление T на 8, можно получить неправильный ответ.

D. Квантовая определённость

Автор задачи: Владимир Мшанецкий
Количество сдавших команд: 3 из 23 (13%)

Задача решается с помощью обхода в ширину графа состояний, который строится по исходному графу следующим образом. Каждой паре вершин u и v (в тому числе, когда $u=v$) исходного графа в графе состояний соответствует вершина, обозначим её (u,v), которая обозначает состояние, когда Женя находится в вершине u, а Коля — в вершине v. Всего в графе состояний будет B^2 вершин. Каждому ребру (a,b) исходного графа в графе состояний соответствует множество рёбер $((a,v),(b,v))$ и $((v,a),(v,b))$ для всех вершин v исходного графа. Каждое из этих рёбер обозначает возможность перехода между соответствующими состояниями, причём для совершения этого перехода один из ребят стоит на месте в вершине v исходного графа, а второй — движется из вершины a в вершину b (или наоборот) исходного графа по ребру (a,b). Всего в графе состояний будет $2BT$ рёбер. Теперь удалим из графа состояний все недопустимые состояния — такие состояния (u,v), у которых расстояние между точками u и v больше, чем D. Теперь длина пути между состояниями (S_1,S_2) и (F,F) в получившемся графе состояний будет ответом задачи. Длину пути можно определить обходом в ширину из состояния (S_1,S_2).

На практике явное построение описанного выше графа состояний не является обязательным. Более того, решение получается значительно проще, если этого не делать. Можно взять за основу обычный обход в ширину по исходному графу и модифицировать его таким образом, что в очередь будут помещаться не отдельные вершины v, а пары вершин (u,v). В процессе обработки извлечённой из очереди пары вершин (u,v) следует сначала проанализировать все рёбра (u,w) из вершины u, а затем — все рёбра (v,w) из вершины v. При этом в первом случае переход будет осуществляться в пару вершин (w,v), а во втором — в пару вершин (u,w). Перед переходом следует проверить, что расстояние между точками из новой пары не превышает D. При этом длину пути следует хранить не в одномерном массиве, как для обычного обхода в ширину, а в двумерном.

Кроме того, можно заметить, что состояния (u,v) и (v,u) в данной задаче равнозначны. Действительно, если мы уже были в состоянии, когда Женя находится в вершине u, а Коля — в вершине v, то, если они поменяются местами, ничего не изменится. Следовательно, можно перед осуществлением перехода всегда делать так, что, например, $u \geq v$. Это позволит сократить в два раза максимальное количество посещаемых состояний и требуемый объём памяти для хранения длин пути. Но для успешной сдачи данной задачи эта оптимизация необязательна.

При реализации решения следует обратить отдельное внимание на следующие частные случаи:

- какие-то две или все три из вершин S_1, S_2 и F совпадают
- могут быть достигнуты некоторые состояния (F,u) и (v,F), но состояние (F,F) — недостижимо (говоря иными словами, Женя может попасть в F, и Коля может попасть в F, но оба одновременно — не могут)
- после достижения некоторого состояния (F,v) или (u,F) может быть необходимо совершить переход в некоторое состояние (u,v), такое, что $u \neq F$ и $v \neq F$ (говоря иными словами, после того, как один из ребят в первый раз попадёт в F, ему может быть необходимо снова уехать из F, чтобы вновь вернуться туда позже)

E. Точка зрения

Автор задачи: Владимир Мшанецкий
Количество сдавших команд: 21 из 23 (91%)

Данная задача задумывалась как задача на внимательность при чтении условия и на пространственное мышление. Женя и Илья сидят за столом друг напротив друга, и строчка печенек лежит между ними таким образом, что каждый из них видит её со своей стороны (со своей «точки зрения»). Если принять за основу точку зрения Жени, то получается, что Илья Вичсайдап видит строчку вверх ногами (прочтите ещё раз внимательно его фамилию). А, как известно, если перевернуть вверх ногами цифру 6, то получится цифра 9, и наоборот. В этом и кроется причина спора ребят, ведь каждый из них подсчитывает печеньки, смотря на строчку со своей стороны. Обратите внимание, что в задаче не спрашивалось, прав ли кто-то из них (ведь это — философский вопрос), а выполнил ли кто-то из них подсчёт правильно.

На вход дана строчка циферок так, как её видит Женя. При этом возможны три варианта:

1. В ней цифр 6 больше, чем цифр 9. Следовательно, Женя посчитал правильно. При этом, поскольку Илья на месте каждой цифры 6 видит цифру 9 и наоборот, он тоже посчитал правильно. Таким образом, ответ — «да».
2. В ней одинаковое количество цифр 6 и 9. Женя говорит, что цифр 6 — больше, а следовательно он ошибается. Но, поскольку в перевёрнутой строке цифр тоже будет поровну, Илья тоже ошибается. Таким образом, ответ — «нет».
3. В ней цифр 9 больше, чем цифр 6. Очевидно, что в этом случае Женя посчитал неправильно. Но в перевёрнутой строке больше будет как раз цифр 6, а не цифр 9, как говорит Илья, поэтому он тоже посчитал неправильно. Таким образом, ответ — опять «нет».

Получается, что ответ — «да» тогда и только тогда, когда в данной на вход строке цифр 6 строго больше, чем цифр 9. Возможен вариант, что одна из цифр или даже обе ни разу не встречаются в строке. Тогда получается, что количество соответствующих цифр равно нулю, и логика остаётся прежней.

При реализации следует обратить внимание на то, что строка, подаваемая на вход, может иметь длину до 100 символов. Диапазона значений и точности всех стандартных числовых типов не достаточно для хранения 100-значного числа. Поэтому входные данные нужно считывать не как число, а как строку текста. Затем следует пройти циклом по всем её символам и посчитать количество символов «6» и «9».

F. Отдых для программистов

Автор задачи: Леонид Беркович
Количество сдавших команд: 9 из 23 (39%)

Разделим всех программистов на группы таким образом, чтобы в каждой отдельной группе дни отпуска не совпадали, и сумма дней не превышала значения d. Максимальное число программистов, которые при этом могут быть свободны от работы в один и тот же день, равно минимально возможному числу таких групп (`grcount`).

Таким образом, задача сводится к нахождению минимального числа `mincount` таких групп при данных условиях. Для решения будем использовать неполный перебор всевозможных наборов групп в поиске минимального их числа.

Во входном массиве — число дней отпуска для каждого из N программистов. Если все элементы входного массива равны 0, то `w = grcount = 0`, и в качестве ответа остается вывести `w = N - grcount`.

Первый способ. Изначально присвоим переменной `mincount` максимально возможное значение N. Для хранения информации о том, присоединен ли каждый из программистов к какой-либо группе (0 — нет, 1 — да) используем `bitset`. Каждый раз при создании новой группы будем включать в нее первый из незадействованных элементов входного массива, а если текущая группа не заполнена, добавлять в нее по одному из оставшихся, перебирая в цикле. При этом каждый раз мы решаем задачу нахождения минимального значения `grcount`, постепенно уменьшая число оставшихся элементов, а переход к новой группе происходит, когда не остается места в текущей. Таким образом, можно организовать рекурсию, передавая в каждом вызове функции три параметра: число групп `grcount`, сумму элементов новой группы и `bitset`. Когда в `bitset` остается один свободный элемент, функция вычисляет новое значение `mincount` и завершает работу.

Примечание: можно ускорить вычисления, не обрабатывая в рекурсии значения элементов входного массива, равные 0 или d, а только учитывать число таких элементов при вычислении ответа.

Второй способ, более лаконичный. Будем использовать рекурсивную функцию, в которую при вызове передаются 5 параметров:

- входные данные в виде вектора, в котором хранится число дней отпуска для каждого из N программистов,
- текущий индекс этого вектора (при первом вызове равен 0),
- число используемых групп (при первом вызове равно 0),
- число дней d в сезоне отпусков,
- вспомогательный вектор, содержащий текущее число дней для каждой из групп; в начале размер вектора равен 0, максимально возможный размер равен длине входного вектора.

Функция должна возвратить минимальное возможное число групп (`mincount`). В каждом новом вызове функции сначала присвоим переменной `mincount` максимально возможное значение. Затем в цикле будем пробовать добавлять одно и то же значение из входного вектора к одной из используемых групп и в новую группу. Первый элемент входного вектора войдет в первую группу (таком образом, увеличит значение первого элемента вспомогательного вектора, то есть, текущее количество дней для первой группы, второй элемент может добавиться в одну из первых двух групп и так далее). Таким образом, можно достичь оптимального распределения по группам всех элементов, поскольку в завершение рекурсии, когда достигается конец входного вектора, мы всегда находим минимальное количество групп.

В завершение остается вывести результат, `w = N - mincount`.

G. Самодостаточный стартап

Автор задачи: Максим Молчанов
Автор идеи задачи: Юрий Тимков
Количество сдавших команд: 7 из 23 (30%)

Задача решается с помощью жадного алгоритма. Необходимо отсортировать всех инвесторов по убыванию цены, которую они готовы заплатить за одну акцию, после чего пройти всех инвесторов по порядку, продавая каждому из них максимальное количество акций, которое ему можно продать. Максимальное количество акций, которое можно продать конкретному инвестору, определяется как минимум из:

- максимального количества акций, которое готов купить данный инвестор
- количества акций, которое ещё можно продать группе инвесторов-друзей, к которой принадлежит данный инвестор
- общего количества акций, которые осталось продать

Для определения количества акций, которое ещё можно продать некоторой группе инвесторов, нужно предварительно разбить всех инвесторов на группы инвесторов-друзей и затем поддерживать для каждой группы количество акций, которое уже было продано инвесторам из этой группы. Если представить инвесторов вершинами графа, а дружеские связи — рёбрами в этом графе, то получится, что каждая компонента связанности — это группа инвесторов-друзей. Таким образом, задача разбивки инвесторов на группы сводится к задаче разбивки графа дружеских связей на компоненты связанности. Разбивку можно выполнить, явно построив этот граф и обойдя его, например, поиском в глубину, либо воспользовавшись структурой данных «система непересекающихся множеств».

При реализации решения необходимо учесть, что как суммарный заработок, так и заработок от продажи акций одному инвестору могут не помещаться в диапазон значений 32-битных целых. Кроме того, важно не забыть учесть требование продать не более K акций всего, всем инвесторам в сумме, как это сделали некоторые команды, которые так и не сдали задачу.

H. Есть два ведра

Автор задачи: Денис Остапенко
Количество сдавших команд: 4 из 23 (17%)

Решение задачи можно описать как обход в ширину графа возможных состояний с последующим восстановлением пути. Однако, решение задачи можно придумать, даже не зная теорию графов.

Пусть текущее состояние описывается парой чисел (a,b), где a — текущее количество воды в ведре ёмкостью A, а b — текущее количество воды в ведре ёмкостью B. Тогда начальным состоянием будет $(0,0)$, а получить нужно одно из состояний $(C,0)$, $(C-1,1)$, $(C-2,2)$, ..., $(2,C-2)$, $(1,C-1)$, $(0,C)$ (некоторые из перечисленных состояний могут быть невозможны, если $A<C$ или $B<C$). Рассмотрим, как каждая из доступных операций меняет текущее состояние:

- `fillA`: $(a,b)\rightarrow(A,b)$
- `fillB`: $(a,b)\rightarrow(a,B)$
- `emptyA`: $(a,b)\rightarrow(0,b)$
- `emptyB`: $(a,b)\rightarrow(a,0)$
- `AtoB`: $(a,b)\rightarrow(a-m,b+m), m=min(a,B-b)$
- `BtoA`: $(a,b)\rightarrow(a+m,b-m), m=min(b,A-a)$

Организуем очередь, в которую будем добавлять достигнутые состояния, которые нужно обработать. Изначально поместим в неё состояние $(0,0)$. При обработке очередного состояния, извлечённого из очереди, попробуем применить к нему каждую из доступных операций и поместим полученные в результате этого новые состояния в очередь. Будем продолжать обрабатывать очередь, пока она не опустеет, или пока мы не извлечём из неё одно из целевых состояний, то есть такое состояние (a,b), для которого выполняется равенство $a+b=C$.

Для того, чтобы иметь возможность после достижения целевого состояния восстановить цепочку операций, приведших к нему, организуем двумерный массив размерностью $(A+1)\times(B+1)$. Каждому возможному состоянию (a,b) в этом массиве будет соответствовать элемент с такими же индексами. Элементами массива будут структуры, в которых будет указываться, какая из операций была применена непосредственно перед достижением соответствующего состояния, а также каким было предыдущее состояние, к которому была применена эта операция. Таким образом, эти структуры сформируют подобие связанного списка, двигаясь по которому можно восстановить всю цепочку операций, приведших к конкретному состоянию. Изначально массив нужно инициализировать некоторыми специальными значениями, которые будут обозначать, что соответствующее состояние ещё не было достигнуто. В процессе обработки очереди, после применения некоторой операции и получения некоторого нового состояния, прежде, чем помещать его в очередь, проверим по массиву, не достигалось ли это состояние ранее. Если оно уже достигалось, обрабатывать его повторно не следует. В противном случае следует обновить соответствующий элемент массива и добавить это новое состояние в очередь.

Для восстановления цепочки операций, приведших к найденному в процессе обработки очереди целевому состоянию, нужно воспользоваться описанным выше массивом и пройти по нему всю цепочку от найденного целевого состояния до состояния $(0,0)$. При этом приведшие к этому целевому состоянию операции будут восстановлены в обратном порядке (от последней к первой), поэтому их следует в процессе восстановления складывать в новый массив, а затем — вывести его в обратном порядке.

I. Snake

Автор задачи: Александр Нестерюк
Количество сдавших команд: 15 из 23 (65%)

Задача решается в три этапа. На первом этапе проверяется возможность заполнения всех элементов матрицы по формуле `K == M * N`, где `K` — количество элементов во входном массиве составных частей змеи, `M` и `N` — размерность матрицы, описывающей спираль. В случае невыполнения равенства выводится надпись «`IMPOSSIBLE`» и обработка завершается.

На втором этапе производится сортировка считанного массива по убыванию при помощи одного из методов сортировки, например, пузырька.

На третьем этапе производится заполнение результирующей матрицы. Одним из способов выполнения данного этапа является организация одного цикла по всем элементам отсортированного на предыдущем этапе массива с записью его элементов в выходную матрицу. Всего можно выделить четыре участка обхода элементов по уменьшающимся виткам спирали ($b=0,1,2,\ldots$) начиная от начального элемента $(0,0)$:

1. по i-той строке в прямом направлении от начального элемента до последнего (то есть, $(b,b)\ldots(b,N-b-1)$)
2. по j-тому столбцу также в прямом направлении от элемента, находящегося в следующей строке, находящейся после обработанной на предыдущем участке (то есть, $(b+1,N-b-1)\ldots(M-b-1,N-b-1)$)
3. по p-той строке в обратном направлении от столбца с номером на единицу меньше, чем полученном на предыдущем участке, до минимального для данного витка спирали (то есть, $(M-b-1,N-b-2)\ldots(M-b-1,b)$)
4. по r-тому столбцу в обратном направлении от строки с номером на единицу меньше, чем полученном на предыдущем участке, до минимального на данном участке спирали (то есть, $(M-b-2,b)\ldots(b+1,b)$)

Последний этап решения можно немного преобразовать путем введения дополнительных переменных, отвечающих за направление изменений координат, в результате чего получаем код:

```
for(int ii = 0, i = 0, j = 0, dx = 0, dy = 0, di = 0, dj = 1,
    bk = 1; ii < k; ii++) {
 B[i][j] = A[ii];
 if(dj) {
  j += bk ? 1 : -1;
  if(bk && j == n-1-dx) dj = 0, di = 1;
  else if(!bk && j == dx) dj = 0, di = 1, dy++;
 } else if(di) {
  i += bk ? 1 : -1;
  if(bk && i == m-1-dy) di = 0, dj = 1, bk = 0;
  else if(!bk && i == dy) di = 0, dj = 1, bk = 1, dx++;
 }
}
```

Здесь переменная `ii` отвечает за текущий элемент входного массива; `dx` и `dy` — ограничения, накладываемые размерами спирали на текущем витке; `di` и `dj` — переменные, отвечающие за движение по строке либо столбцу; `bk` отвечает за прямое либо обратное движение по строкам/столбцам.

Также необходимо отметить, что имеются еще несколько вариантов реализации последнего этапа, например, с использованием циклов, рекурсии и тому подобное.

В конце выводим полученную матрицу.

J. XOR expression

Автор задачи: Дмитрий Садовый
Количество сдавших команд: 1 из 23 (4%)

Як можна було б вирішити задачу, якщо послідовність була б усього 20 цифр? Повний перебір усіх 2^{N-1} варіантів розташування операцій XOR ($N-1$ бо в нас лише стільки місць, куди можна поставити операції). Для прикладу візьмемо послідовність 1000101 та маску 011010, де 1 означає поставити XOR між i та $i+1$ цифрою. Тобто, вираз буде виглядати як «`10 XOR 0 XOR 0 XOR 01 XOR 01`». І з усіх 2^{N-1} варіантів виберемо той, що підходить краще за всіх.

Але в нашій задачі N може бути більшим, але достатньо маленьким, щоб використати підхід Meet-in-the-middle. Давайте порахуємо відповіді для всіх префіксів та суфіксів, довжина яких не більша за $N/2$. А далі переберемо усі вирази вигляду:

`[prefix] XOR [middle part] XOR [suffix]`

Таких варіантів буде N^2. Тож ми беремо найкращу відповідь для префіксу, для суфіксу, а також додаємо дві операції XOR. Зверніть увагу, що префікс чи суфікс можуть бути пустими. В такому випадку ми додаємо лише одну або нуль додаткових операцій XOR.

Щоб знайти найкращу відповідь з усіх оптимальних, достатньо вибрати найбільшу або найменшу маску — залежить від того як ви рахуєте індекси (зліва направо або навпаки).

Також виявилось, що за умовами даної задачі не існує послідовностей, щоб у відповіді було більше 5 операцій XOR. Тож проходить звичайний перебір із п'яти вкладених циклів. Але, якщо ви дорішуєте задачу, то все ж раджу написати авторський розв'язок, особливо якщо ви не знайомі з meet-in-the-middle (google в допомогу).