Задача A. Выбор вершин взвешенного дерева

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Дан граф, являющийся деревом. В вершинах графа написаны целые числа. Множество вершин графа называется *допустимым*, если никакие две вершины этого множества не соединены ребром.

Рассмотрим все допустимые множества вершин графа. Для каждого такого множества вычислим сумму чисел, написанных в его вершинах. Какова максимальная из этих сумм?

Формат входных данных

Граф в этой задаче задан в виде *корневого дерева*. В графе выделена вершина — *корень дерева*. Для каждой вершины i, не являющейся корнем, задан номер вершины-предка p_i в корневом дереве. Дерево, заданное таким образом, состоит из рёбер $i - p_i$ для всех вершин i, кроме корня.

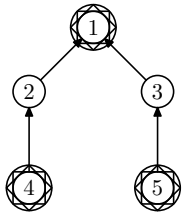
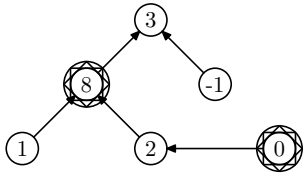
В первой строке записано целое число n — количество вершин в графе ($1 \leqslant n \leqslant 100$). В следующих n строках задан граф. В i-й из этих строк записаны через пробел два целых числа p_i и q_i; здесь p_i — номер вершины-предка i-ой вершины, а q_i — число, записанное в этой вершине. Для корня дерева $p_i = 0$; для всех остальных вершин $1 \leqslant p_i \leqslant n$. Числа q_i не превосходят $10\,000$ по абсолютной величине.

Гарантируется, что заданный граф является деревом.

Формат выходных данных

В первой строке выведите одно число — максимальную сумму чисел в допустимом множестве.

Примеры

стандартный ввод	*стандартный вывод*	*пояснение*
5 0 1 1 2 1 3 2 4 3 5	10	
6 5 8 6 0 5 -1 1 1 0 3 1 2	8	

На рисунках показаны графы, заданные в примерах. В каждом графе выделено допустимое множество с максимальной суммой чисел в вершинах.

Задача B. Красно-чёрное дерево

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Рассмотрим двоичное корневое дерево: в этом дереве выделена вершина-корень, все рёбра ведут в направлении от корня, и у каждой вершины от нуля до двух детей. Будем считать, что из каждой вершины выходят ровно два ребра: если при этом у неё меньше двух детей, оставшиеся рёбра ведут в пустоту.

Будем называть дерево *красно-чёрным*, если выполнены следующие условия:

- Каждая вершина покрашена либо в красный, либо в чёрный цвет.
- В дереве нет ребра с двумя красными концами. Пустота считается чёрной.
- Рассмотрим все пути по рёбрам, идущие из корня в пустоту. Количество чёрных вершин на всех таких путях одинаково.

Аналогичную раскраску можно использовать в двоичных деревьях поиска для их балансировки.

Дано непустое двоичное корневое дерево. Покрасьте его так, чтобы оно стало красно-чёрным, или выясните, что это невозможно.

Формат входных данных

В первой строке задано целое число n ($1 \leqslant n \leqslant 500$) — количество вершин в дереве. Вершины пронумерованы числами от 1 до n.

В следующей строке записаны через пробел n целых чисел: p_1, p_2, …, p_n ($0 \leqslant p_i \leqslant n$). Число $p_i > 0$ означает, что вершина i — ребёнок вершины p_i. Если же $p_i = 0$, то i — корень дерева.

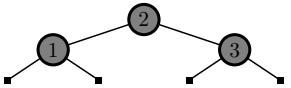
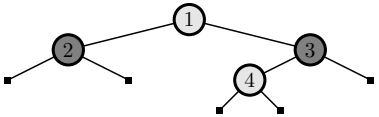
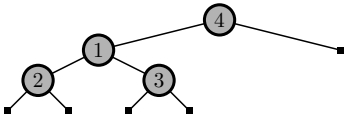
Гарантируется, что во входных данных корректно задано двоичное корневое дерево: корень ровно один, у каждой вершины от нуля до двух детей, а кроме того, из корня можно, двигаясь по рёбрам, попасть во все остальные вершины.

Формат выходных данных

Если можно покрасить заданное дерево так, чтобы оно стало красно-чёрным, выведите любую такую раскраску в виде строки из n символов. Символ на i-й позиции должен быть равен «`R`», если вершина i красная, и «`B`», если она чёрная.

Если же покрасить дерево невозможно, выведите слово «`Impossible`».

Примеры

стандартный ввод	*стандартный вывод*	*пояснение*
3 2 0 2	BBB	
4 0 1 1 3	RBBR	
4 4 1 1 0	Impossible	

Задача C. Простые пути в дереве

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Дан неориентированный связный граф из n вершин и $n-1$ ребра. Требуется для каждого ребра посчитать суммарную длину простых путей, проходящих через это ребро. Длиной пути здесь называется количество рёбер в пути.

Формат входных данных

В первой строке задано целое число n ($2 \leqslant n \leqslant 300\,000$). Следующие $n-1$ строк содержат пары чисел от 1 до n — рёбра графа.

Формат выходных данных

Выведите $n-1$ строку: i-я строка должна содержать целое число — ответ для i-го ребра.

Пример

стандартный ввод	*стандартный вывод*
5 1 2 2 3 2 4 5 1	13 8 8 9

Задача D. Гиперкуб

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Гиперкуб — это обобщение понятия трёхмерного куба на N измерений. Нуль-мерным гиперкубом является точка, одномерным — отрезок, двумерным — квадрат. В общем же случае N-мерный гиперкуб — это правильный N-мерный многогранник, каждая из $2 \cdot N$ граней которого является $(N-1)$-мерным гиперкубом. Например, для $N = 2$ квадрат — это правильный многоугольник, каждая из $2 \cdot 2 = 4$ сторон которого — отрезок, то есть одномерный гиперкуб. Отметим, что N-мерный гиперкуб имеет 2^N вершин.

Старшеклассник Петя долго разбирался, что же такое гиперкуб, но наконец понял, как этот объект устроен, и ему настолько понравилось, что он даже придумал свою собственную игру на гиперкубе. Игра заключается в следующем.

Рассмотрим N-мерный единичный гиперкуб. Расположим его таким образом, чтобы одна из вершин находилась в начале координат — точке $(0, 0, \ldots, 0)$ в N-мерном пространстве, а для любой из остальных вершин каждая координата равнялась бы нулю или единице. В каждой из 2^N вершин запишем по целому неотрицательному числу. Игрок начинает свой путь в начале координат. За один ход можно переместиться из текущей вершины по любому ребру при условии, что сумма координат новой вершины строго больше суммы координат старой. Игра заканчивается, когда игрок попадает в вершину $(1, 1, \ldots, 1)$, имеющую максимальную сумму координат — N. Результат игры — сумма чисел во всех посещённых игроком вершинах. Цель игры — пройти по гиперкубу таким образом, чтобы эта сумма (количество очков за игру) оказалась как можно больше.

Петя довольно быстро понял, что между двумя вершинами гиперкуба A и B ребро есть тогда и только тогда, когда все координаты этих вершин $(A_1, A_2, \ldots, A_N)$ и $(B_1, B_2, \ldots, B_N)$ совпадают, кроме одной, которая равна нулю у одной из вершин (скажем, A) и единице у другой (B). Поскольку при этом $A_1 + A_2 + \ldots + A_N + 1 = B_1 + B_2 + \ldots + B_N$, то по такому ребру можно перемещаться из A в B, но не наоборот. Однако, сыграв в свою игру, Петя не может с уверенностью сказать, является ли полученная им сумма максимальной, или можно на данном гиперкубе сыграть по-другому и набрать больше очков.

Напишите программу, которая по данному гиперкубу находит максимальную сумму, которую можно получить, сыграв в эту игру.

Формат входных данных

В первой строке записано число N ($1 \leqslant N \leqslant 10$) — размерность гиперкуба. В следующих 2^N строках содержится по одному числу в каждой; в $(k+2)$-й строке записано C_k ($0 \leqslant C_k \leqslant 1000$) — число в вершине с номером k.

Номер вершины вычисляется так: вершина A с координатами $(A_1, A_2, \ldots, A_N)$ имеет номер, равный $A_1 \cdot 2^{N-1} + A_2 \cdot 2^{N-2} + \ldots + A_{N-1} \cdot 2 + A_N$, то есть координаты просто интерпретируются как двоичная запись номера вершины. В этой нумерации начальная вершина имеет номер 0, а конечная — номер $2^N - 1$.

Формат выходных данных

Выведите одно число — максимальную сумму, которую можно получить при игре на данном гиперкубе.

Пример

стандартный ввод	*стандартный вывод*
3 1 2 3 4 5 6 7 8	21

Пояснение к примеру

Наш маршрут таков:

- вершина 0 (число 1, координаты $(0, 0, 0)$) — начальная
- вершина 4 (число 5, координаты $(1, 0, 0)$)
- вершина 6 (число 7, координаты $(1, 1, 0)$)
- вершина 7 (число 8, координаты $(1, 1, 1)$) — конечная

Наше количество очков: $1 + 5 + 7 + 8 = 21$.

Любой другой маршрут с соблюдением правил игры даёт меньшее количество очков.

Задача E. Магические кристаллы

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Маг и волшебник Фнарф имеет в своём распоряжении несколько магических кристаллов. Каждый кристалл характеризуется магической силой — целым неотрицательным числом, меньшим магического числа m. Если несколько кристаллов собрать вместе, то они сольются в новый магический кристалл, сила которого будет равна $(p_1 \cdot p_2 \cdot \ldots \cdot p_k) \mod m$, где p_1, p_2, …, p_k — силы исходных кристаллов.

Для заклинания фейерверка, которое собирается произнести Фнарф, ему потребуется два магических кристалла. Чем больше будет сумма магических сил этих кристаллов, тем грандиознее будет фейерверк. Какова максимальная суммарная сила двух кристаллов, полученных из исходных? Имейте в виду, что после слияния нескольких кристаллов их нельзя отделить от полученного кристалла и использовать повторно.

Формат входных данных

В первой строке заданы целые числа n и m через пробел — количество кристаллов и магическое число ($2 \leqslant n \leqslant 16$, $1 \leqslant m \leqslant 10^9$). Во второй строке даны n чисел p_1, p_2, …, p_n через пробел — силы имеющихся у Фнарфа магических кристаллов ($0 \leqslant p_i < m$).

Формат выходных данных

В первой строке выведите одно число — максимальную суммарную силу двух кристаллов, полученных из исходных.

Примеры

стандартный ввод	*стандартный вывод*
2 10 3 3	6
4 2 0 0 1 1	2
3 8 5 7 3	14

Задача F. Коммивояжёр возвращается!

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	5 секунд
Ограничение по памяти:	512 мебибайт

Коммивояжёр возвращается в систему Альфы Центавра! Население системы с нетерпением ждёт его прибытия — каждый хочет приобрести что-нибудь с далёких планет!

Как обычно, коммивояжёр хочет минимизировать транспортные расходы. Он выбирает начальную планету, прилетает туда на межгалактическом корабле, после чего посещает все остальные планеты системы в порядке, минимизирующем суммарную стоимость посещения, и на другом межгалактическом корабле улетает обратно. Естественно, коммивояжёр не хочет летать ни на какую планету дважды.

Найдите оптимальный маршрут для коммивояжёра. Массы больше не могут ждать!

Формат входных данных

В системе Альфы Центавра n планет. Это число записано в первой строке входных данных ($1 \leqslant n \leqslant 19$). Следующие n строк содержат по n чисел каждая: j-е число на i-й из этих строк — стоимость перемещения a_{ij} от i-й планеты до j-й. Числа в каждой строке разделены пробелами. Числа a_{ii} не несут полезной информации. Все числа во входных данных положительны и не превосходят 10^8.

Формат выходных данных

В первой строке выведите минимальную суммарную стоимость посещения всех планет. Во второй строке выведите n чисел через пробел — номера планет системы в порядке их посещения. Если оптимальных маршрутов несколько, можно вывести любой из них.

Пример

стандартный ввод	*стандартный вывод*
3 8 1 6 3 5 7 4 9 2	5 3 1 2

Задача G. Замощение доминошками 1

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Доминошка на клетчатой плоскости — это фигура из двух клеток, имеющих общую сторону. Замощение прямоугольника из $m \times n$ клеток доминошками — это такой набор доминошек, целиком лежащих внутри прямоугольника, что каждая клетка прямоугольника покрыта ровно одной из них. Два замощения P и Q считаются различными, если существуют доминошки $p \in P$ и $q \in Q$ такие, что одна из клеток p и q — общая, а другая клетка p отличается от другой клетки q.

Сколько существует способов замостить прямоугольник $m \times n$ доминошками?

Формат входных данных

Первая строка ввода содержит два целых числа m и n — высота и ширина прямоугольника, соответственно ($1 \leqslant m, n \leqslant 12$).

Формат выходных данных

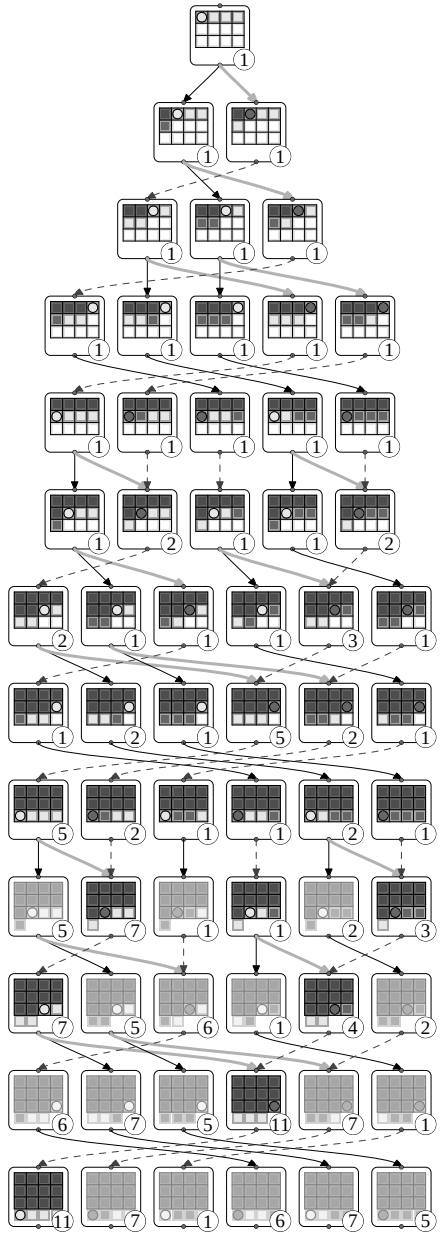
Выведите одно число — количество способов замостить прямоугольник $m \times n$ доминошками.

Пример

стандартный ввод	*стандартный вывод*
3 4	11

Пояснение к примеру

На диаграмме справа представлен ход решения этой задачи динамическим программированием по изломанному профилю. Профиль состоит из n клеток. В каждом прямоугольнике показаны закрашенные клетки и текущий профиль. Тёмные клетки — заполненные, светлые — пустые. Клетки профиля имеют серые рамки, клетки над профилем — тёмные, а клетки под профилем — светлые. Текущая рассматриваемая клетка — первая клетка профиля — обозначена кружком. Каждому уровню диаграммы соответствует одна и та же текущая клетка. Число в левом нижнем углу — количество способов получить такое состояние. Стрелками обозначены переходы между состояниями. Жирной серой стрелке соответствует укладка горизонтальной доминошки, тонкой чёрной стрелке — укладка вертикальной доминошки. Пунктирная стрелка означает, что первая клетка профиля уже занята, и доминошка в этом переходе не укладывается. Состояния, в которых профиль выходит за границы прямоугольника, затенены.



Задача H. Замощение доминошками 2

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Доминошка на клетчатой плоскости — это фигура из двух клеток, имеющих общую сторону. Замощение прямоугольника из $m \times n$ клеток доминошками — это такой набор доминошек, целиком лежащих внутри прямоугольника, что каждая клетка прямоугольника покрыта ровно одной из них. Два замощения P и Q считаются различными, если существуют доминошки $p \in P$ и $q \in Q$ такие, что одна из клеток p и q — общая, а другая клетка p отличается от другой клетки q.

Сколько существует способов замостить прямоугольник $m \times n$ доминошками? В этой версии задачи некоторые клетки прямоугольника могут быть вырезаны.

Формат входных данных

Первая строка ввода содержит два целых числа m и n — высота и ширина прямоугольника, соответственно ($1 \leqslant m, n \leqslant 12$). Следующие m строк содержат по n символов каждая и описывают прямоугольник. Символ '`.`' (точка) соответствует обычной клетке, а символ '`#`' (решётка) — вырезанной. Гарантируется, что в этих m строках ввода никакие другие символы не встречаются.

Формат выходных данных

Выведите одно число — количество способов замостить заданный прямоугольник $m \times n$ доминошками.

Пример

стандартный ввод	*стандартный вывод*
3 4	11
2 2 .. .#	0
2 3 #.. ..#	1

Пояснение к примеру

Первый пример совпадает с примером из первой версии задачи.

Во втором примере три невырезанные клетки невозможно покрыть доминошками.

В третьем примере единственный способ покрытия — положить две доминошки горизонтально.

Задача I. Длинные домино

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Найдите количество способов замостить прямоугольник размера $m \times n$ длинными домино — прямоугольниками размера 3×1.

Каждое домино должно полностью находиться внутри прямоугольника, домино не должны накладываться.

Формат входных данных

Входной файл содержит m и n ($1 \leqslant m \leqslant 9$, $1 \leqslant n \leqslant 30$).

Формат выходных данных

Выведите количество способов замостить прямоугольник $m \times n$ длинными домино.

Примеры

стандартный ввод	*стандартный вывод*
3 3	2
3 10	28

Задача J. Щупальца Дэйви Джонса

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мебибайт

Капитан Дэйви Джонс (на картинке справа) хранит свои сокровища в сундуке. Замок на сундуке представляет собой клетчатую прямоугольную панель размера $h \times w$ клеток. В каждой клетке этой панели расположена одна клавиша. Кодовый замок открывается, если нажать на определённые клавиши в определённом порядке.

Чтобы не тратить время зря, последовательность нужно набирать как можно быстрее, поэтому Дэйви Джонс использует для открывания сундука свои k щупалец. Изначально все щупальца находятся над левой верхней клавишей.

Щупальца Дэйви Джонса существуют в четырёхмерном пространстве, поэтому можно считать, что они способны перемещаться независимо. За одну секунду щупальце может либо остаться на месте, либо переместиться таким образом, чтобы оказаться над одной из восьми соседних клавиш, либо нажать на клавишу, которая находится под ним. Одно нажатие занимает целую секунду, и в эту секунду нажимать на другие клавиши не следует, а щупальце, которое осуществляет нажатие, не может перемещаться.

Напишите программу, которая по последовательности из n нажатий, требуемой для открывания сундука, выяснит, за какое минимальное время сундук можно открыть, а также выдаст информацию о том, какие щупальца и в какие моменты должны производить эти нажатия.

Формат входных данных

В первой строке ввода заданы через пробел четыре целых числа h, w, k и n: высота и ширина панели, количество щупалец и длина требуемой последовательности нажатий ($2 \leqslant h, w \leqslant 10$, $2 \leqslant k \leqslant 10$ и $1 \leqslant n \leqslant 1500$). Следующие n строк описывают требуемую последовательность нажатий. Каждая из этих строк содержит два числа r_i и c_i, разделённых пробелом: номер ряда и номер столбца, в которых расположена клавиша, которую необходимо нажать ($1 \leqslant r_i \leqslant h$, $1 \leqslant c_i \leqslant w$).

Формат выходных данных

В первой строке выведите одно целое число: минимальное общее время t в секундах, за которое можно сделать все нажатия. Далее выведите n строк. На i-й из этих строк выведите два целых числа s_i и t_i: номер щупальца, которое нажимает на i-ю клавишу в последовательности, и номер секунды, в которую это происходит ($1 \leqslant s_i \leqslant k$). Секунды нумеруются с единицы. Помните, что моменты времени t_i должны строго возрастать. Если оптимальных ответов несколько, можно вывести любой из них.

Примеры

стандартный ввод	*стандартный вывод*
3 3 2 4 1 1 2 2 3 3 1 1	5 1 1 2 2 2 4 1 5
3 4 2 4 3 3 1 4 3 2 1 2	7 2 3 1 4 2 6 1 7

Пояснения к примерам

В первом примере один из оптимальных сценариев таков. В первую секунду первое щупальце нажимает на клавишу в клетке $(1, 1)$. Второе щупальце тем временем движется в клетку $(2, 2)$, во вторую секунду производя там второе нажатие. После этого третья секунда уходит на перемещение второго щупальца из $(2, 2)$ в $(3, 3)$, а четвёртая — на нажатие клавиши в этой клетке. Первое щупальце тем временем ожидает в клетке $(1, 1)$ с тем, чтобы сразу после третьего нажатия — на пятой секунде — вновь нажать на клавишу, расположенную в этой клетке.

Во втором примере один из оптимальных сценариев таков. Второе щупальце движется к клетке $(3, 3)$ и нажимает на расположенную там клавишу на третьей секунде. Первое щупальце движется к клетке $(1, 4)$ и нажимает на расположенную там клавишу на четвёртой секунде. После этого первому щупальцу, чтобы добраться до клетки $(1, 2)$ и осуществить там нажатие, нужно ещё три секунды. Второе же щупальце может переместиться в $(3, 2)$ уже к началу пятой секунды и нажать на клавишу на пятой или на шестой секунде.