Информатика. Методическое пособие по 19-21 заданиям.

За 19 – 21 *задания ты можешь получить* 3 *балла (по* 1 *баллу за каждое задание). Примерное суммарное время выполнения:* 25 *минут. Уровни сложности:* 19 *задача — базовый,* 20 *задача — повышенный,* 21 *задача — высокий. Ответом к каждому заданию может быть цифра (число) или слово (набор букв).*

Содержание

1 Теория для решения руками

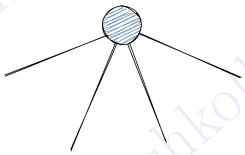
1.1 Правила игры

В математических и информатических задачах на теорию игр существуют некоторые стандартные правила, которые позволяют продумать игру. Давайте рассмотрим такие правила для $19 - 21$ задач:

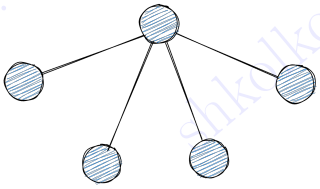
- В игре не должно быть никаких случайностей. Другими словами, **игра должна быть детерминирована;**
- В процессе игры не может возникнуть бесконечного цикла или безвыходной ситуации. Значит, **игра должна рано или поздно заканчиваться;**
- У игры **обязательно должно быть условие победы.**

1.2 Дерево игры

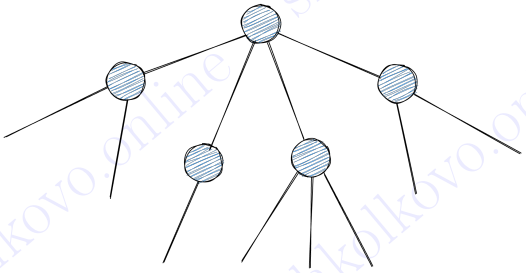
Давайте поиграем! Я предлагаю изобразить такую игру в виде дерева. Первый игрок начинает в какой-то позиции, из которой он может сделать одно из четырёх действий.



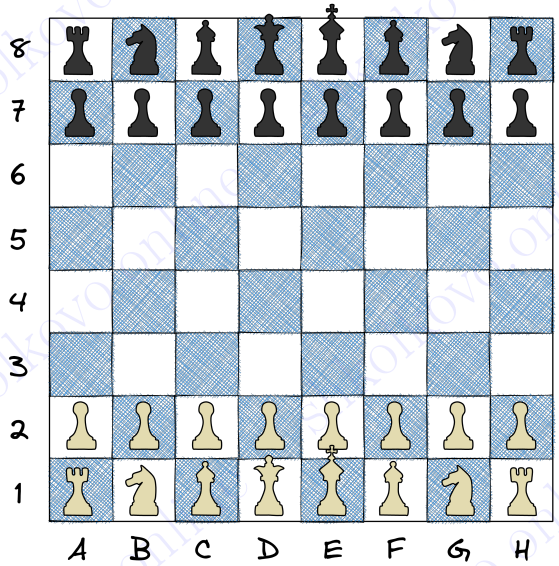
Каждый раз, когда игрок делает определенные действия, он переводит игру в другую позицию.



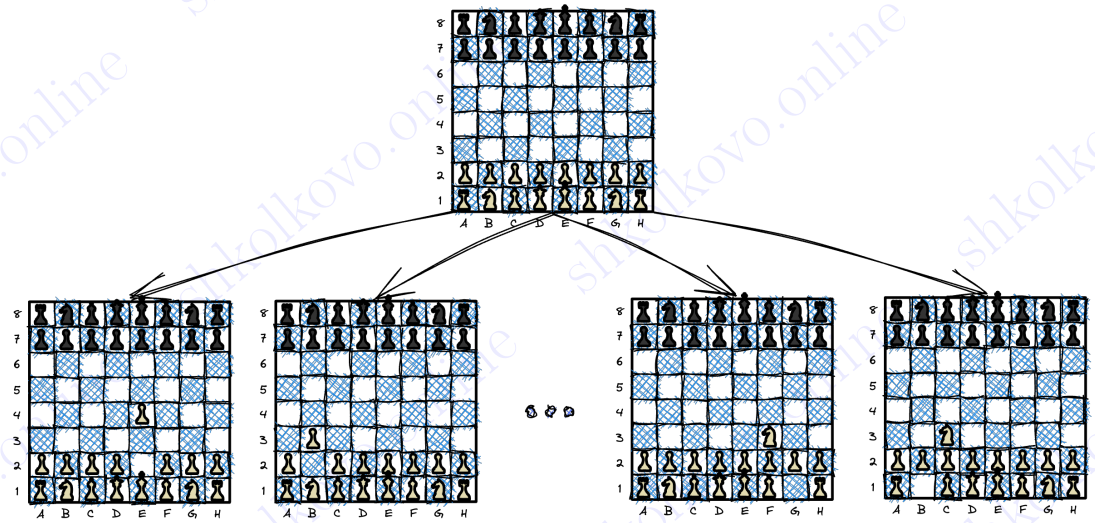
После чего ходит второй игрок. Например, у него есть из первой (крайней левой) позиции два хода, из второй — только один, из третьей — три хода и из четвертой — еще два.



Для каждого хода игрока мы можем продолжать рисовать дерево. Представьте себе игру в шахматы. Исходной позицией будет являться позиция, в которой все фигуры стоят на двух крайних рядах с противоположных сторон.



Из этой позиции есть несколько допустимых ходов для первого игрока. Можно сходить пешкой на одну/две клетки вперед или вообще сходить конем.



Шахматы удовлетворяют всем вышеописанным правилам. Эта игра пошаговая: белые, черные, белые, черные. Эта игра детерминирована, в ней нет никаких случайностей. Эта игра конечна, если учитывать, что трехразовое повторение определенной позиции - это ничья. С помощью комбинаторики можно посчитать количество всех возможных позиций шахматных фигур на доске. Их очень много, но не бесконечно. Значит, если игра будет длиться достаточно долго, какая-то позиция повторится три раза.

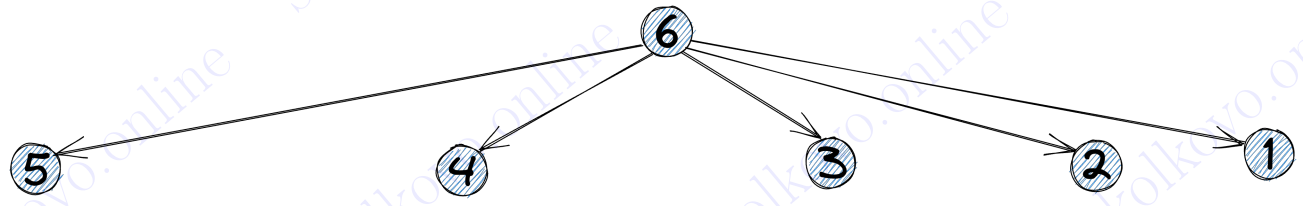
Думайте о шахматах как об очень огромном дереве, в котором на каждом шаге можно делать огромное количество ходов. Из-за этого шахматы достаточно сложно просчитать, в отличие от некоторых других игр. Но что значит «просчитать»? Здесь мы плавно подходим к понятию выигрышной стратегии.

1.3 Выигрышная стратегия

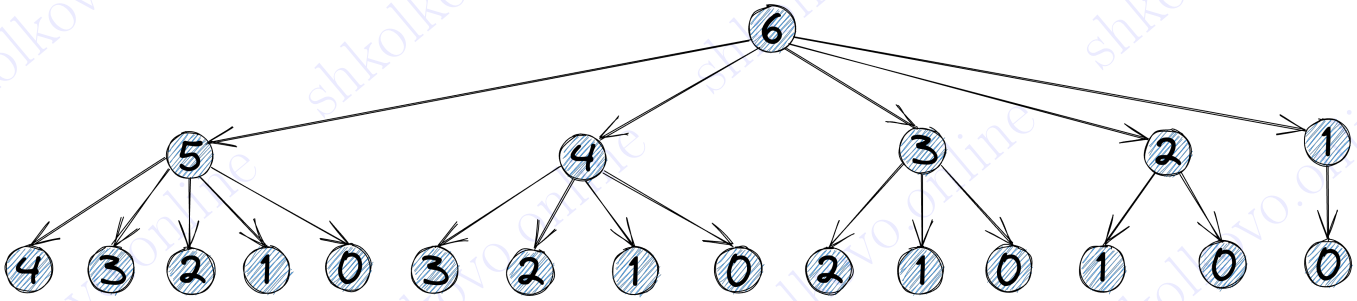
Выигрышная стратегия — последовательность ходов, которая приведет игрока к победе при любых ходах соперника

Два игрока играют в следующую игру. Перед игроками лежит куча из 6 камней. Игроки ходят по очереди. Первый ход делает первый игрок. За один ход игрок может убрать из кучи $1, 2, 3, 4$ или 5 камней. Игра завершается в тот момент, когда количество камней в куче становится равным 0. Кто выигрывает при правильной игре?

Давайте нарисуем дерево для этой игры. Начальная позиция равна 6. Первый игрок может сходить в 5, если уберет из кучи 1 камень, в 4, если уберет из кучи 2 камня, в 3, если уберет из кучи 3 камня, в 2 если уберет из кучи 4 камня, в 1, если уберет из кучи 5 камней.



Второй игрок может: выиграть из позиции 1, убрав 1 камень; выиграть из позиции 2, убрав 2 камня, или получить позицию 1, убрав 1 камень; выиграть из позиции 3, убрав 3 камня, получить позицию 2, убрав 1 камень, или получить позицию 1, убрав 2 камня и так далее с остальными позициями.



Надо ли нам дорисовывать следующие ходы первого игрока?

...

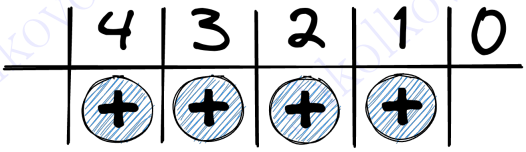
Нет, не надо!

Мы понимаем, что у второго игрока есть выигрышная стратегия: куда бы ни сходил первый игрок, второй всегда будет убирать все камни из кучи и попадать в 0. Значит, второй игрок выиграет в этой игре.

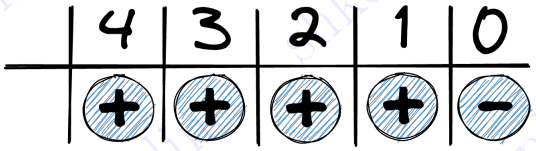
1.4 Принцип дополнения

Два игрока, Петя и Ваня, играют в следующую игру. Перед игроками лежит куча из 42 камней. Игроки ходят по очереди. Первый ход делает Петя. За один ход игрок может убрать из кучи $1, 2, 3$ или 4 камня. Игра завершается в тот момент, когда количество камней в куче становится равным 0. Побеждает тот, кто сделает последний ход. Кто выигрывает при правильной игре?

Посмотрим на эту игру с конца. Если в куче $1, 2, 3$ или 4 камня, тогда такая позиция называется **выигрышной**. Потому что, если перед Петей лежит куча из $1, 2, 3$ или 4 камней, он убирает $1, 2, 3$ или 4 камня соответственно и выигрывает.



Тогда Ваня попадает в позицию 0, откуда он выиграть уже никак не может. То есть, позиция, в которой изначально было 4 камня, для Вани будет **проигрышной**.

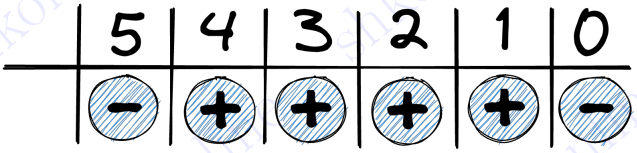


Может ли Петя, имея кучу из 5 камней, выиграть с помощью одного хода?

...

Нет!

При этом любой ход Пети из позиции 5 приведет Ваню к **гарантированной** победе. Получается, что 5 — **проигрышная** позиция.

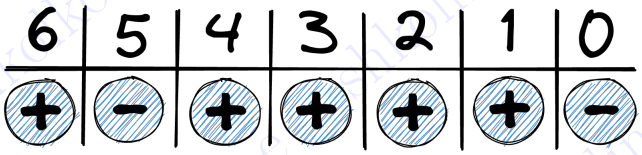


Позиция 6 является выигрышной или проигрышной для Пети?

...

Выигрышной!

Мы уже знаем, что позиция 5 — проигрышная. Тогда Петя из позиции 6 уберет 1 камень и поставит Ваню в позицию 5, откуда Ваня **гарантированно** проиграет.

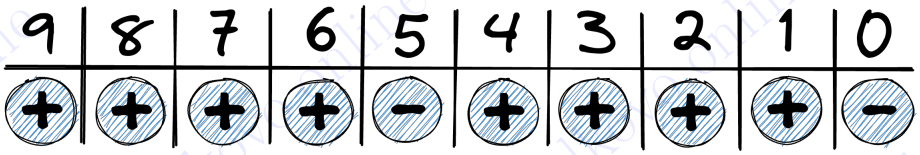


Из каких еще позиций можно попасть в 5?

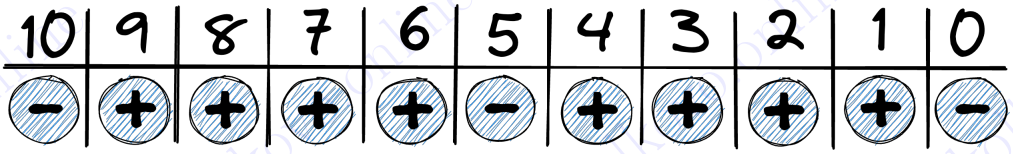
...

Из 7, 8 и 9!

Значит, все эти позиции тоже являются выигрышными.



Позиция 10 является проигрышной по той же причине, что и позиция 5.



Продолжим заполнять нашу табличку.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
−	+	+	+	+	−	+	+	+	+	−	+	+	+	+	−

Заметили закономерность?

...

Любое значение, которое делится на 5 — проигрышное!

Как так получилось? Когда игрок берет 1 камень, соперник берет 4. Когда игрок берет 2 камня, соперник берет 3. Когда игрок берет 3 камня, соперник берет 2. Когда игрок берет 4 камня, соперник берет 1. Таким образом, за два хода игроки берут ровно 5 камней. Игрок дополняет ход соперника до 5. Но зачем?

Чтобы ответить на этот вопрос, ответим на другой:

Позиция 42 является выигрышной или проигрышной для Пети?

...

Выигрышной!

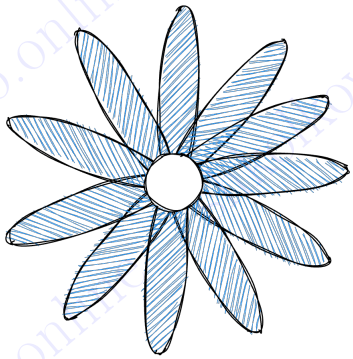
Петя из 42 сходит в 40, чтобы поставить Ваню в проигрышную позицию.

42	41	40	...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+	+	−		−	+	+	+	+	−	+	+	+	+	−	+	+	+	+	−

А дальше Петя будет просто дополнять ходы Вани до 5. Петя понимает, что позиции, кратные 5 — проигрышные. Значит, он будет ходить так, чтобы поставить Ваню позиции, кратные 5.

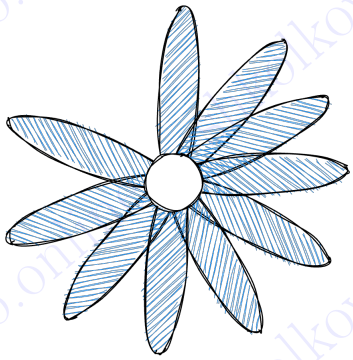
1.5 Симметрия в играх

Два игрока, Петя и Ваня, играют в следующую игру. Перед игроками растет цветок с 11 лепестками. Игроки ходят по очереди. Первый ход делает Петя. За один ход игрок может вырвать 1 лепесток или вырвать 2 лепестка, изначально расположенных рядом. Игра завершается в тот момент, когда количество лепестков на цветке становится равным 0. Побеждает тот, кто вырвет последний лепесток. Кто выиграет в этой игре?

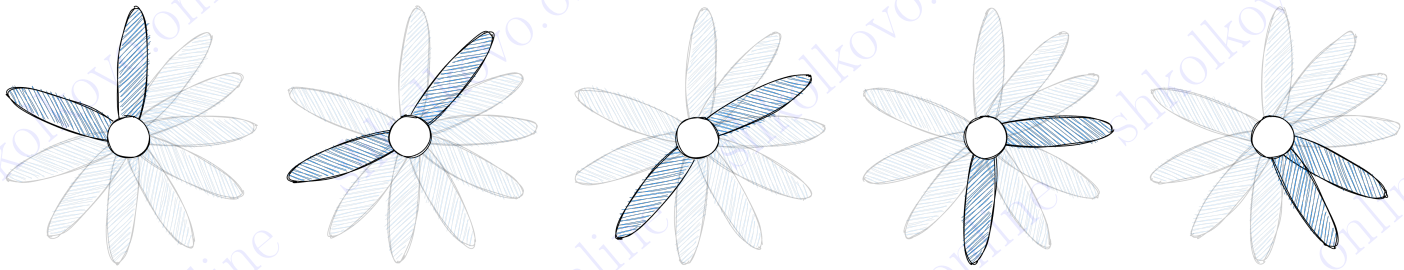


Казалось бы, по принципу дополнения можно всегда дополнять до трёх. Однако проблема заключается в том, что в задаче присутствует ход **вырвать 2 лепестка, изначально расположенных рядом**. Значит, положение лепестков важно.

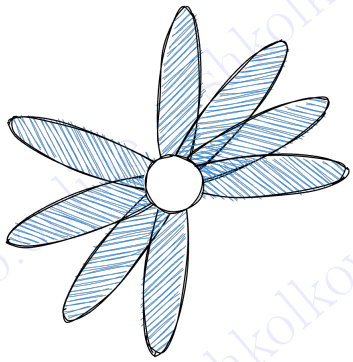
Допустим, Петя решил вырвать 1 любой лепесток.



Тогда Ваня начинает попарно считать лепестки, начиная с того места, где вырвал лепесток Петя.



Последние два посчитанных лепестка Ваня вырывает.

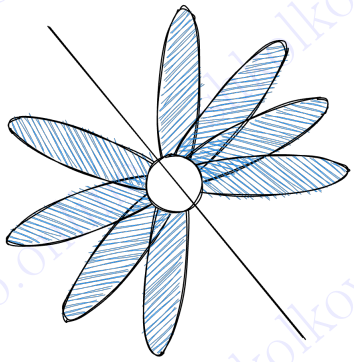


Что уникального можно заметить в текущем цветке?

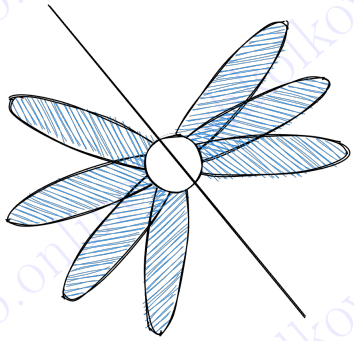
...

Цветок симметричен!

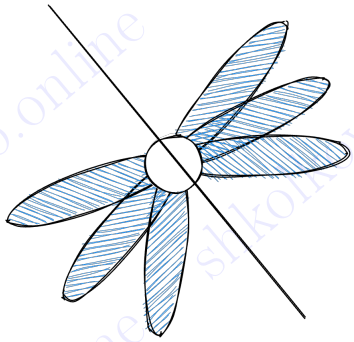
Давайте нарисуем ось симметрии.



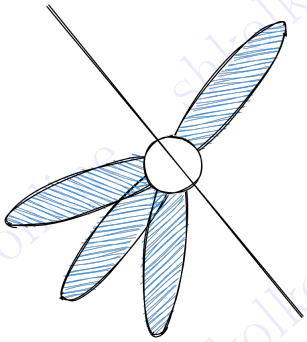
Что нам это дает? Теперь у Вани есть выигрышная стратегия: как бы Петя не ходил, Ваня просто дублирует ход Пети с другой стороны от оси симметрии. Например, Петя вырвал еще 1 лепесток.



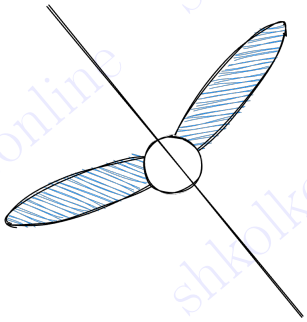
Тогда Ваня просто вырывает симметричный лепесток.



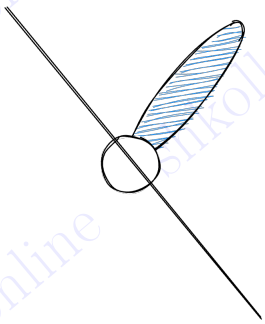
Петя решает вырвать 2 рядом расположенных лепестка.



Тогда Ваня вырывает такие же 2 рядом расположенных симметричных лепестка.



Петя делает свой последний ход, вырывая 1 лепесток.



Тогда Ваня вырывает последний лепесток и выигрывает игру. Принцип симметрии помог Ване выиграть эту игру. Давайте посмотрим на данный принцип в другой игре.

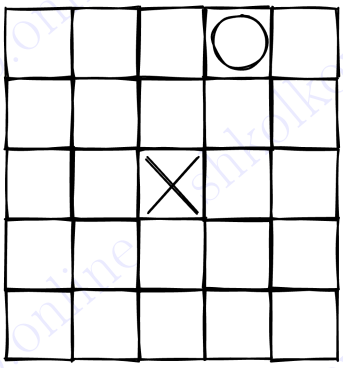
Два игрока, Петя и Ваня, играют в следующую игру. Перед игроками находится таблица 5×5 клеток. Игроки ходят по очереди. Первый ход делает Петя. За один ход игрок может поставить в одну клетку либо крестик, либо нолик (Петя может поставить крестик, а Ваня — нолик). Игра завершается в тот момент, когда количество пустых клеток становится равным 0.

Когда игра завершится, для каждой строки и для каждого столбца подводится итог по следующим правилам: если в строке больше крестиков, чем ноликов, значит в данной строке выиграли крестики, в противном случае — нолики. То же самое проделываем со столбцами.

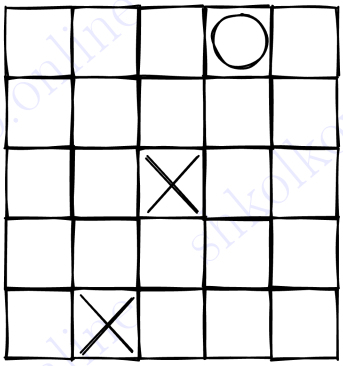
После этого считаем количество строк и столбцов, в которых победили крестики, далее считаем количество строк и столбцов, в которых победили нолики. Побеждает тот, у кого больше выигрышных строк и столбцов.

Кто выигрывает при правильной игре?

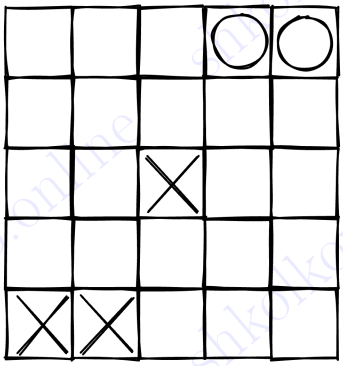
Допустим, Петя поставил в центр таблицы крестик, а Ваня — нолик в любую другую клетку.



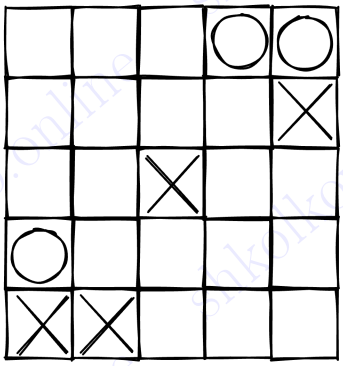
Тогда Петя ставит крестик в клетку, симметричную ходу Вани относительно центра.



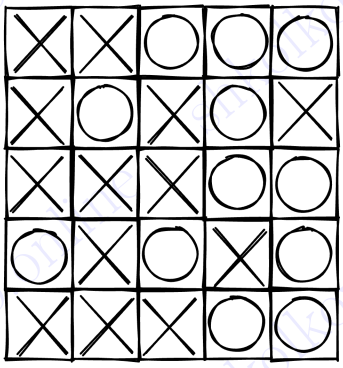
Ваня ставит нолик в любую другую клетку таблицы. Петя ходит симметрично.



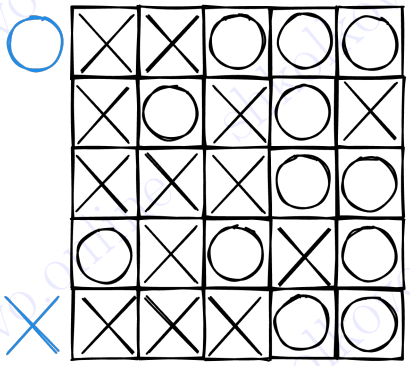
Ваня ставит нолик в любую другую клетку таблицы. Петя ходит симметрично.



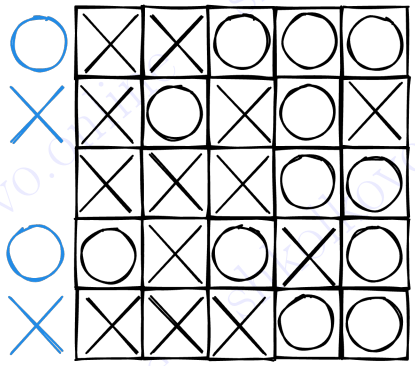
Заполним доску до конца.



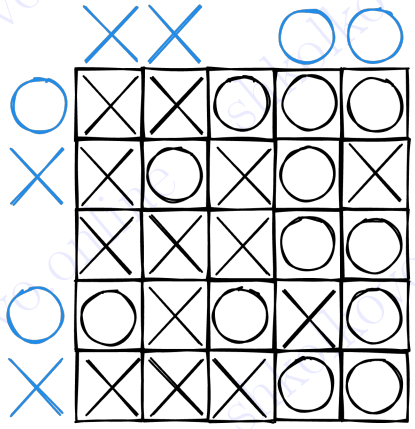
А теперь подводим итоги. В первой строке 3 нолика и 2 крестика. Значит, в этой строке победили нолики. Но у этих трёх ноликов есть три симметричных крестика в нижней строке, которые обеспечили победу крестикам.



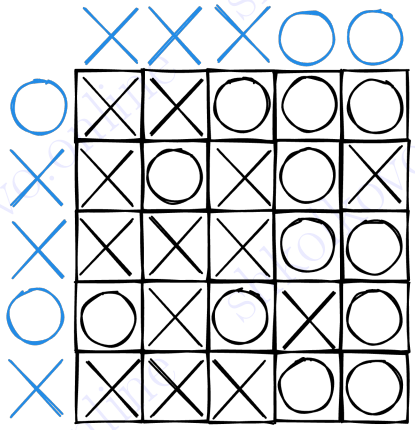
Во второй снизу строке 3 нолика и 2 крестика. Значит, в этой строке победили нолики. Но у этих трёх ноликов есть три симметричных крестика во второй сверху строке, которые обеспечили победу крестикам.



Подведем итоги, не считая центральную строку и столбец.



Для каждой выигрышной строки/столбца ноликов у нас нашлась симметричная выигрышная строка/столбец крестиков. Теперь смотрим на центральное перекрестие. Благодаря тому, что крестик у нас является центром симметрии, в центральной строке/столбце крестиков на 1 больше, чем ноликов.



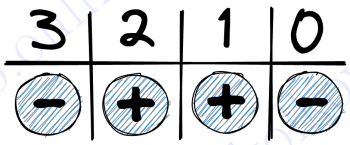
Значит, Ваня победил Петю со счетом $6 - 4$.

1.6 Счёт позиций

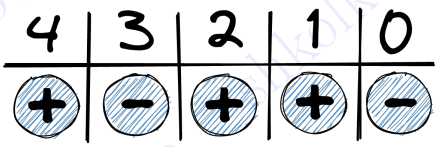
Два игрока, Петя и Ваня, играют в следующую игру. Перед игроками лежит куча из 23 камней. Игроки ходят по очереди. Первый ход делает Петя. За один ход игрок может убрать из кучи $1, 2$ или 5 камней. Игра завершается в тот момент, когда количество камней в куче становится равным 0. Побеждает игрок, который сделал последний ход. Кто выигрывает при правильной игре?

Принцип дополнения тут не работает, потому что у нас отсутствуют ходы **убрать из кучи 3 или 4 камня**. В противном случае игрок дополнял бы ходы соперника до 6 и выигрывал.

Рассмотрим несколько последних позиций. Если перед игроком лежит куча из 0 камней, игрок не может сделать ход, а значит, он проиграл. Из позиций 1 и 2 игрок может попасть в 0, убрав 1 или 2 камня соответственно, значит, это выигрышные позиции. Из позиции 3 игрок не может выиграть своим первым ходом, но любой его ход ставит противника в выигрышную позицию (1 или 2). Значит, 3 — проигрышная позиция.



Если перед игроком лежит куча из 4 камней, он может поставить соперника в проигрышную позицию 3, откуда любой ход соперника приведет игрока к победе. Значит, из позиции 4 игрок может выиграть за два хода.



Из позиции 5 игрок может убрать все 5 камней и выиграть за один ход.

5	4	3	2	1	0
+	+	−	+	+	−

Если перед игроком куча из 6 камней, он может убрать из кучи $1, 2$ или 5 камней и поставить соперника в выигрышную позицию $5, 4$ или 1. Значит, 6 — проигрышная позиция.

6	5	4	3	2	1	0
−	+	+	−	+	+	−

В позицию 6 можно попасть из $7, 8$ или 11. Значит, все эти позиции — выигрышные.

11	10	9	8	7	6	5	4	3	2	1	0
+			+	+	−	+	+	−	+	+	−

Все возможные выигрышные позиции на основе всех возможных проигрышных записаны в таблицу. Значит, позиция 9 точно будет проигрышной.

Почему позиция 9 точно будет проигрышной?

...

Потому что все ходы из 9 ставят противника в выигрышную позицию?

Почему из 9 все ходы ведут в выигрышную позицию?

Потому что все позиции, из которых можно попасть в проигрышные, мы уже отметили!

11	10	9	8	7	6	5	4	3	2	1	0
+		−	+	+	−	+	+	−	+	+	−

Теперь, опираясь на то, что 9 — это проигрышная позиция, мы заполняем выигрышные.

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+			+	+	−	+	+	−	+	+	−	+	+	−

Какая позиция точно будет проигрышной?

...

Позиция 12! Потому что все позиции, из которых можно попасть в проигрышные, мы уже отметили!

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+		−	+	+	−	+	+	−	+	+	−	+	+	−

Опираясь на то, что 12 — это проигрышная позиция, мы заполняем выигрышные.

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+			+	+	−	+	+	−	+	+	−	+	+	−	+	+	−

Заполним таблицу до конца по аналогичному принципу.

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+	+	−	+	+	−	+	+	−	+	+	−	+	+	−	+	+	−	+	+	−	+	+	−

Получилось, что позиция 23 выигрышная, а значит, в этой игре победит Петя. Его стратегия заключается в том, чтобы каждым своим ходом ставить Ваню в проигрышную позицию, из которой Ваня любым ходом будет ставить Петю в выигрышную позицию.

Самые внимательные могли заметить закономерность в таблице: каждая позиция, которая делится на 3, является проигрышной. Оказалось, что ход **убрать из кучи 5 камней** никак не опровергает принцип дополнения.

На самом деле в этой игре можно было дополнять ход соперника до числа, кратного трём. Но почему? 1 и 2 дают разные остатки от деления на 3, а 5 дает остаток от деления на 3 равным 2. В этом смысле, **убрать из кучи 2 или 5 камней** являются эквивалентными ходами, если говорить по модулю 3.

Получается, что 1 надо дополнять 2 или 5, а 2 и 5 надо дополнять 1. В итоге игрок всегда дополняет либо до 3, либо до 6. Таким образом, игрок всегда получает количество камней, кратное трём.

Если бы Петя увидел принцип дополнения в этой игре, первым действием он бы поставил Ваню в позицию 21 с количеством камней, кратным трём. Дальше, куда бы Ваня ни сходил, Петя всегда будет дополнять ход Вани до количества камней, кратного трём, и в конечном итоге придет в 0, потому что 0 делится на 3.

2 Теория для решения с помощью Excel

2.1 Горячие клавиши

Горячие клавиши — это комбинации на клавиатуре, которые выполняют то или иное действие. Это удивительный способ сэкономить уйму времени, которое может пойти на гораздо более полезные вещи, например, на то же перерешивание с целью нахождения и исправления своих ошибок.

Познакомимся с теми горячими клавишами, которые чаще всего нам понадобятся:

Ctrl + C — для копирования любого объекта (текст, ячейка, формулы);

Ctrl + V — для вставки скопированного объекта;

Ctrl + X — для вырезки выделенного объекта;

Shift ⇑ + ← → ↑ ↓ — для выделения;

Ctrl + Shift ⇑ + ← → ↑ ↓ — для перемещения.

2.2 Формулы

Формулы в Excel — главная причина, которая привлекает пользователей, поскольку формулы ускоряют работу и лишают ее рутинного написания одной и той же информации для некоторой цели. Чтобы начать писать формулу, необходимо лишь нажать на нужную ячейку и поставить знак =. Все, осталось дело за малым — написать саму формулу, которая нам необходима. В стандартных заданиях 19 – 21 не нужны сложные формулы, нам даже не понадобятся встроенные функции Excel, нужны лишь знания арифметики 1 – 3 классов:)

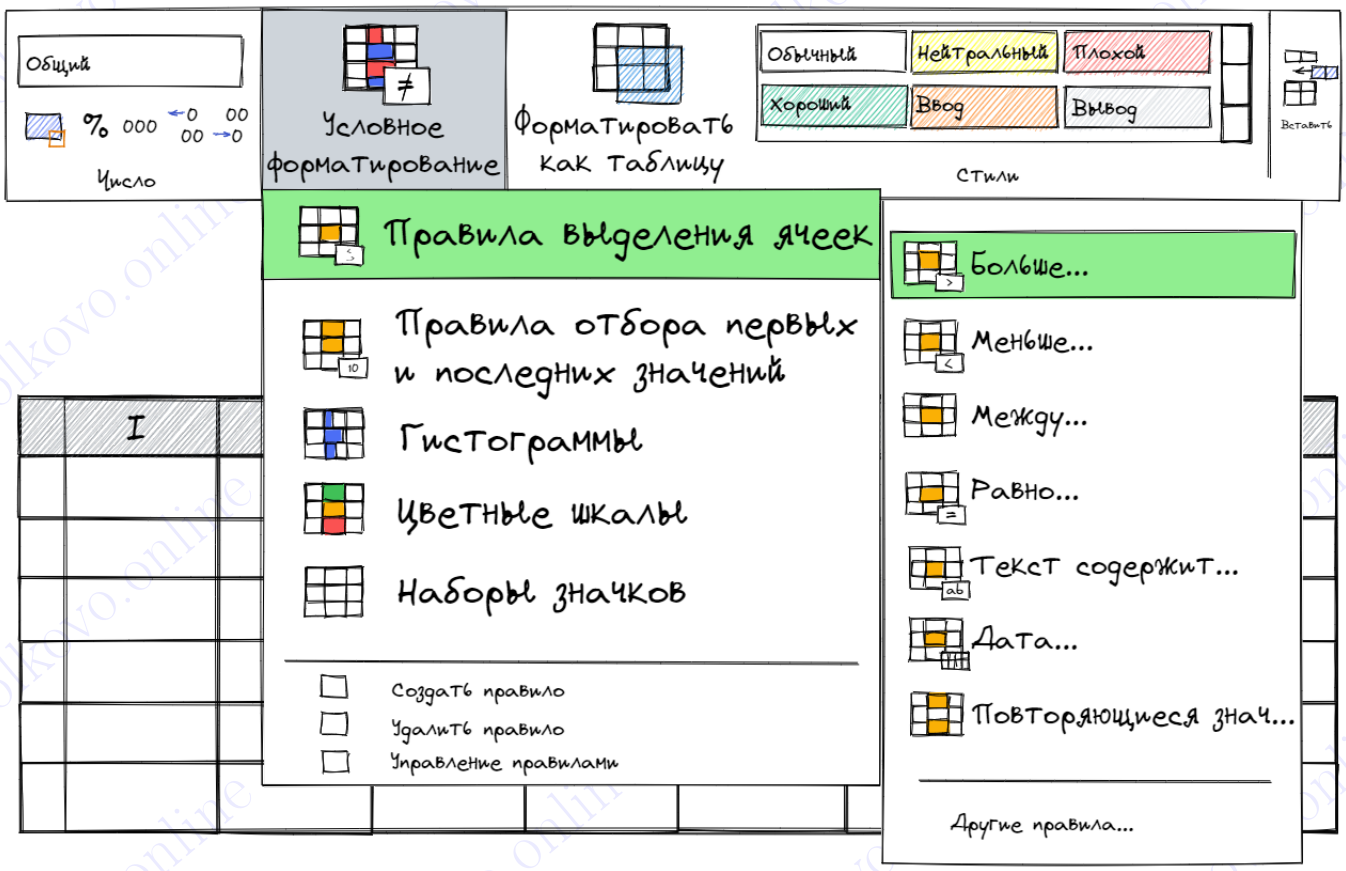
Пример формулы для 19 – 21 заданий:

	A	B	C
1	10	=A1+3	

2.3 Условное форматирование

Условное форматирование — еще одна штука, которая спасет нас при решении игр с помощью Excel. Оно поможет нам определить «хорошие» и «плохие» ячейки, по всем преданиям и традициям зеленый цвет — хорошие клетки, а красный — плохие. Может показаться, что это что-то бесполезное, но, поверьте, смотреть на каждую ячейку и думать: «Больше ли в этой клетке камней, чем наш порог выигрыша? Влияет ли это на победу соперника?» быстро надоест, а условное форматирование моментально отвечает нам на эти вопросы:)

Итак, как же им пользоваться? В шапке Excel переходим в раздел Главная (по умолчанию этот раздел открыт сразу после создания нового файла), в нем находим Условное форматирование, нажимаем на него, появляется выбор «правил», наводим курсор на Правила выделения ячеек и, отталкиваясь от условия задачи, выбираем Больше или Меньше. В появившемся окошке записываем значение, данное в условии, которое обеспечит нам победу, но не забываем про строгость/нестрогость знаков.



2.4 Начинаем игру

Давайте решим стандартный прототип на 1 кучу камней.

Два игрока, Петя и Ваня, играют в следующую игру. Перед игроками лежит куча камней. Игроки ходят по очереди, первый ход делает Петя. За один ход игрок может добавить в кучу два камня или увеличить количество камней в куче в 3 раза. Например, имея кучу из 17 камней, за один ход можно получить кучу из 19 или 51 камня. Чтобы делать ходы, у каждого игрока есть неограниченное количество камней. Игра завершается в тот момент, когда количество камней в куче становится не менее 27.

Победителем считается игрок, сделавший последний ход, то есть первым получивший позицию, в которой в куче будет 27 или более камней. В начальный момент в куче было S камней, $1 \leq S \leq 26$.

Известно, что Петя выигрывает своим первым ходом. Укажите минимальное значение S, при котором такая ситуация возможна.

	A	B	C	D	E
1	`+2				
2	*3				
3	>=27				
4		Старт		Петя	
5		0		=B5*3	
6					

У нас есть клетка, отвечающая за наш ответ, но пока она белая, а должна быть зеленой. Что нужно сделать, чтобы это получить? Перебрать все подходящие значения в стартовой клетке ручками!

	A	B	C	D	E
1	`+2				
2	*3				
3	>=27				
4		Старт		Петя	
5		9		27	
6					

И вот подходящая ячейка стала зеленой, значит, мы нашли ответ! Так как мы перебирали снизу, то значение, которое мы получили, и есть минимально возможное. Если бы мы шли сверху, нужно было бы проверить меньшие значения до того момента, пока мы не получили бы белую клетку, то есть неподходящую нам, так как в ней слишком мало камней для победы. Таким образом, лучше всегда проверять снизу вверх.

Отлично, с первым ходом Пети разобрались, но что делать, если нужно найти значение, при котором первым ходом победит Ваня?

Известно, что Ваня выигрывает первым ходом после **любого** хода Пети. Укажите минимальное значение S, при котором такая ситуация возможна.

Применяем такой же алгоритм, что и в предыдущей задаче, но теперь вырезаем всю таблицу (Ctrl + X), содержащую числовые значения, и вставляем (Ctrl + V) туда, где до этого момента находился ответ на п. а), то есть в ячейку, хранящую количество камней после первого хода Пети.

	A	B	C	D	E	F	G
1	`+2						
2	*3						
3	>=27						
4		Старт		Петя		Ваня	
5				9		27	
6							

Копируем (Ctrl + C) текущую таблицу и вставляем (Ctrl + V) чуть ниже первой.

	A	B	C	D	E	F	G
1	`+2						
2	*3						
3	>=27						
4		Старт		Петя		Ваня	
5				9		27	
6				9		27	

Зачем нам две одинаковые таблицы?

...

Каждая таблица отвечает за свой ход!

Сейчас мы изменим первую ячейку каждой таблицы в соответствии со всеми возможными ходами. Пусть первая таблица отвечает за ход $+1$, а вторая — за ход $\cdot 3$.

	A	B	C	D	E	F	G
1	`+2						
2	*3						
3	>=27						
4		Старт		Петя		Ваня	
5		0		=B5+2		=D5*3	
6				=B5*3		=D6*3	

Теперь подставляем значения в стартовую ячейку, пока все крайние ячейки обеих таблиц не станут зелеными.

	A	B	C	D	E	F	G
1	`+2						
2	*3						
3	>=27						
4		Старт		Петя		Ваня	
5		7		9		27	
6				21		63	

Можно было бы подумать, что для первого хода больше нечего разбирать, но нет. Не зря ведь выделено слово **любого** в условии. Разберемся с еще одним случаем.

Известно, что Ваня выигрывает первым ходом после **неудачного** хода Пети. Укажите минимальное значение S, при котором такая ситуация возможна.

Применяем алгоритм из прошлой задачи, но теперь нам не нужно находить стартовое значение, при котором все крайние значения обеих таблиц будут «хорошими», нам будет достаточно только одной таблицы со всеми крайними зелеными ячейками.

	A	B	C	D	E	F	G
1	`+2						
2	*3						
3	>=27						
4		Старт		Петя		Ваня	
5		3		5		15	
6				9		27	

2.5 Игра продолжается

Если можно сделать два различных хода, то действия абсолютно идентичны. Однако Вас могут испугать размеры подобной таблицы, особенно, если мы говорим про второй ход Вани:

	A	B	C	D	E	F	G	H	I	J	K
1											
2	`+2										
3	*3										
4	>=27										
5		Старт		Петя		Ваня		Петя		Ваня	
6		4		6		8		10		30	
7								24		72	
8											
9						18		20		60	
10								54		162	
11											
12				12		14		16		48	
13								42		126	
14											
15						36		38		114	
16								108		324	
17											
18											
19											
20											

Не пугаемся! Это нормально, ведь у нас очень много ходов, конкретно для данной задачи в конце мы можем получить одно из 8 значений.

Обратим внимание, что, если Ваня может выиграть первым или вторым ходом, но не гарантированно первым, последние ячейки могут быть красными, но ВАЖНО! при этом ячейка, отвечающая за первый ход, должна быть зеленой.

3 Теория для решения с помощью программы

3.1 Что нужно знать?

Для дальнейшего понимания материала необходимо знать базовый синтаксис языка программирования Python и уметь составлять логически осмысленные конструкции, используя такие инструменты, как `any()`, `all()`, `@lru_cache` и списковые включения. Вы можете найти этот материал в методическом пособии по усложненному синтаксису Python.

3.2 Функция moves()

В любой задаче на теорию игр есть какой-то вид ходов. Например: убрать из кучи 1 или 2 камня, добавить в кучу 3 камня, стереть одну букву в слове, вырвать один лепесток и т.д.

Давайте напишем функцию `moves()`, которая будет отвечать за ходы игроков. Например, возьмем задачу про кучу камней с ходами «добавить в кучу два камня или увеличить количество камней в куче в 3 раза». Как будет выглядеть наша функция для таких ходов?

```
def moves(heap):
    return heap + 2, heap * 3
```

Функция принимает в качестве аргумента параметр `heap`, который отвечает за текущее количество камней в куче и возвращает кортеж из двух элементов — допустимых ходов.

Например, в куче было 10 камней. Тогда функция `moves()` вернет кортеж из двух элементов: 12 и 30.

Разберем другой пример. Пусть в задаче сказано «за один ход игрок может убрать из кучи один камень или убрать из кучи три камня». Тут мы не можем просто вернуть кортеж из ходов `heap - 1` и `heap - 3`, потому что у нас есть ограничение: количество камней в куче не может быть отрицательным. Значит, чтобы убрать из

кучи 1 камень, мы должны иметь в куче как минимум 1 камень. Аналогично со вторым ходом. Создадим пустой массив, в который будем добавлять ходы только в том случае, если их возможно совершить, не нарушив правил природы. В конце функции вернем этот массив.

```
def moves(heap):
    m = []
    if heap >= 1:
        m.append(heap - 1)
    if heap >= 3:
        m.append(heap - 3)
    return m
```

Что же делать, если в условии не одна куча, а две? Например, в задаче сказано: «За один ход игрок может добавить в одну из куч один камень или увеличить количество камней в куче в два раза». Наша функция будет принимать на вход так же количество камней, но уже не в куче, а в кучах. Значит, `heap` — кортеж из двух элементов — количеств камней в каждой куче. В самом начале функции сделаем **распаковку** элементов кортежа в переменные `a` и `b`, чтобы в дальнейшем с ними было удобно работать. Теперь, для каждой кучи возвращаем все возможные ходы.

```
def moves(heap):
    a, b = heap
    return (a + 1, b), (a, b + 1), (a * 2, b), (a, b * 2)
```

Последний пример. Пусть в задаче сказано: «За один ход игрок может убрать из одной из куч один камень или уменьшить количество камней в куче в два раза (если количество камней в куче нечётно, остаётся на 1 камень меньше, чем убирается)». По аналогии с предыдущей задачей **распаковываем** кортеж в переменные `a` и `b`. Т.к. эта задача на убывание камней, значит, существует нижняя граница. Создаем пустой массив, в который будем добавлять только те ходы, которые возможно совершить, не нарушив правил природы. В конце функции возаращем этот массив.

```
def moves(heap):
    a, b = heap
    m = []
    if a > 0:
        m.append((a - 1, b))
        m.append((a // 2, b))
    if b > 0:
        m.append((a, b - 1))
        m.append((a, b // 2))
    return m
```

Отлично, мы научились задавать в программе ходы. Теперь давайте научимся задавать логику игры.

3.3 Функция game()

Давайте решим следующую задачу.

Два игрока, Петя и Ваня, играют в следующую игру. Перед игроками лежит куча камней. Игроки ходят по очереди, первый ход делает Петя. За один ход игрок может добавить в кучу два камня или увеличить количество камней в куче в 3 раза. Например, имея кучу из 17 камней, за один ход можно получить кучу из 19 или 51 камня. Чтобы делать ходы, у каждого игрока есть неограниченное количество камней. Игра завершается в тот момент, когда количество камней в куче становится не менее 27.

Победителем считается игрок, сделавший последний ход, то есть первым получивший позицию, в которой в куче будет 27 или более камней. В начальный момент в куче было S камней, $1 \leq S \leq 26$.

1. Известно, что Петя выигрывает своим первым ходом. Укажите минимальное значение S, при котором такая ситуация возможна;

2. Найдите два таких значения S, при которых у Пети есть выигрышная стратегия,

причём Петя не может выиграть своим первым ходом, но Петя может выиграть своим вторым ходом независимо от того, как будет ходить Ваня.
В ответе запишите числа в порядке возрастания без пробелов и знаков препинания.

3. Найдите два таких значения S, при которых одновременно выполняются два условия:
– у Вани есть выигрышная стратегия, позволяющая ему выиграть первым или вторым ходом при любой игре Пети;
– у Вани нет стратегии, которая позволит ему гарантированно выиграть первым ходом.
В ответе запишите числа в порядке возрастания без пробелов и знаков препинания.

Наша функция будет принимать на вход переменную `heap` — количество камней в куче — и возвращать наименование позиции, из которой выигрывает игрок.

- «END» — Победа за ноль ходов;
- «P1» — Петя выигрывает за один ход, игра длится один ход;
- «V1» — Ваня выигрывает за один ход, игра длится два хода;
- «P2» — Петя выигрывает за два хода, игра длится три хода;
- «V2» — Ваня выигрывает за два хода, игра длится четыре хода;

Для начала стоит задать условие выигрыша (определить позиции типа «END»). В условии сказано, что если количество камней в куче больше или равно 27, то игра завершается.

```
def game(heap):
    if heap >= 27:
        return 'END'
```

Отлично, теперь давайте определим позиции типа «P1». Если существует такая позиция, из которой с помощью допустимых ходов Петя может попасть в позицию «END», то такая позиция называется «P1». Например, если Петя находится в позиции 25, с помощью хода +2 он может попасть в позицию 27 и выиграть. Значит, позиция 25 имеет тип «P1». Все позиции, начиная с 27 и заканчивая 9, имеют тип «P1», потому что у Пети есть сильный ход, а именно ·3. В коде это будет выглядеть следующим образом:

```
def game(heap):
    if heap >= 27:
        return 'END'
    elif any(game(x) == 'END' for x in moves(heap)):
        return 'P1'
```

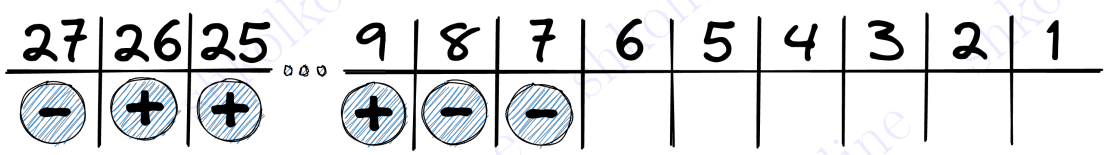
Одновременно с написание программы будем заполнять табличку, чтобы проводить ассоциации.

27	26	25	...	9	8	7	6	5	4	3	2	1
−	+	+		+								

Ответ на пункт 1 — 9.

Идем дальше, зададим позиции «V1». Если существует такая позиция, из которой Петя каждым допустимым ходом попадает в позицию «P1», то такая позиция называется «V1». Другими словами, Петя любым ходом ставит Ваню в выигрышную позицию. Это возможно сделать из позиций 7 и 8.

```
def game(heap):
    if heap >= 27:
        return 'END'
    elif any(game(x) == 'END' for x in moves(heap)):
        return 'P1'
    elif all(game(x) == 'P1' for x in moves(heap)):
        return 'V1'
```

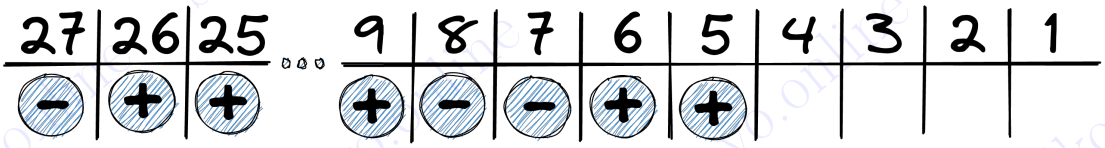


Бывают 19-е задачи, где условие на позицию «V1» меняют. Вместо привычного «Ваня выигрывает своим первым ходом при любой игре Пети» иногда можно встретить «Ваня выиграл своим первым ходом после неудачного первого хода Пети». В таком случае вместо `all()` нужно ставить `any()`, потому что Ване достаточно хотя бы одного проигрышного хода Пети.

```
def game(heap):
    if heap >= 27:
        return 'END'
    elif any(game(x) == 'END' for x in moves(heap)):
        return 'P1'
    elif any(game(x) == 'P1' for x in moves(heap)):
        return 'V1'
```

Зададим позиции «P2». Если существует такая позиция, из которой с помощью допустимых ходов Петя может попасть в позицию «V1», то такая позиция называется «P2». Другими словами, Петя хотя бы одним своим ходом ставит Ваню в проигрышную позицию (7 или 8). Это возможно сделать из позиций 5 и 6.

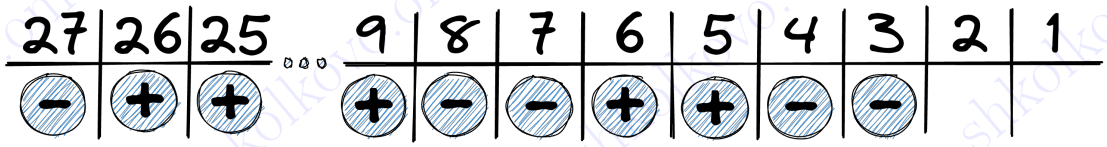
```python
def game(heap):
    if heap >= 27:
        return 'END'
    elif any(game(x) == 'END' for x in moves(heap)):
        return 'P1'
    elif all(game(x) == 'P1' for x in moves(heap)):
        return 'V1'
    elif any(game(x) == 'V1' for x in moves(heap)):
        return 'P2'
```



Ответ на пункт 2 — 56.

Зададим позиции «V2». Если существует такая позиция, из которой Петя каждым допустимым ходом попадает в позицию «P1» или «P2», то такая позиция называется «V2». Другими словами, Петя любым ходом ставит Ваню в выигрышную позицию. Это возможно сделать из позиций 3 и 4.

```
def game(heap):
    if heap >= 27:
        return 'END'
    elif any(game(x) == 'END' for x in moves(heap)):
        return 'P1'
    elif all(game(x) == 'P1' for x in moves(heap)):
        return 'V1'
    elif any(game(x) == 'V1' for x in moves(heap)):
        return 'P2'
    elif all(game(x) == 'P1' or game(x) == 'P2' for x in moves(heap)):
        return 'V2'
```



Ответ на пункт 3 — 34.

Данная конструкция универсальна для большинства прототипов со стандартным условием на кучи и камни. Меняется только функция `moves()` и условие победы. Давайте рассмотрим другое представление функции `game()` на примере этой же задачи.

Теперь функция `game()` будет возвращать не наименование позиции, а количество ходов до завершения игры. Если в куче больше 26 камней, то игра длится 0 ходов.

```
def game(heap):
    if heap >= 27:
        return 0
```

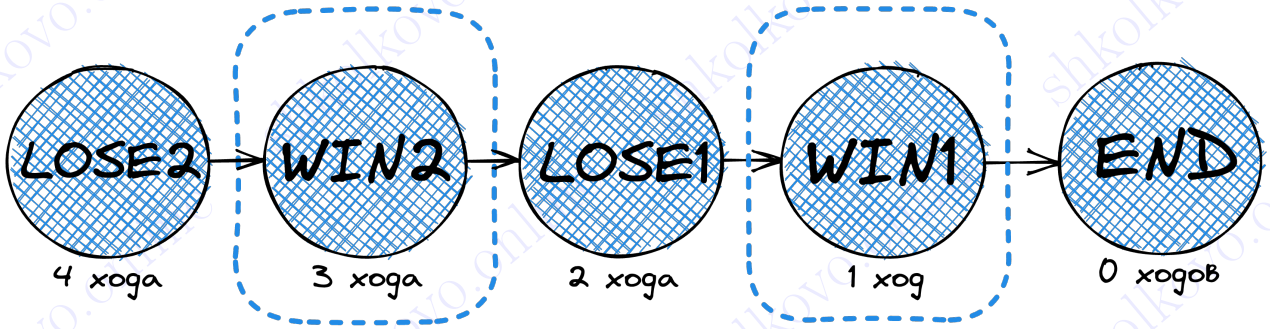
Создадим переменную `steps`, в которой будет содержаться количество ходов до победы.

```python
def game(heap):
    if heap >= 27:
        return 0
    steps = [game(x) for x in moves(heap)]
```

Теперь давайте поподробнее.

- Если игрок находится в позиции «END», то он проиграл. Такая игра длится **ноль ходов**;
- Позицию, из которой игрок выигрывает за **один ход**, назовем «WIN1».
- Позицию, из которой игрок проигрывает за один ход, ставя соперника в выигрышную позицию «WIN1», назовем «LOSE1». Такая игра длится **два хода**.
- Позицию, из которой игрок выигрывает за два хода, ставя противника в проигрышную позицию «LOSE1», назовем «WIN2». Такая игра длится **три хода**.
- Позицию, из которой игрок проигрывает за два хода, ставя противника в выигрышную позицию «WIN2», назовем «LOSE2». Такая игра длится **четыре хода**.

Заметим, что выигрышные позиции имеют нечетное количество ходов в игре, а проигрышные — четное.



Дальше все просто, если хотя бы один ход ставит соперника в проигрышную позицию, то игрок находится в выигрышной позиции, а функция возвращает минимальное число ходов для гарантированной победы. В противном случае, если игрок не может поставить соперника в проигрышную позицию, то он сам находится в проигрышной позиции, а функция возвращает максимальное число ходов, на которое игрок может растянуть игру при поражении.

```
def game(heap):
    if heap >= 27:
        return 0
    steps = [game(x) for x in moves(heap)]
    if any(x % 2 == 0 for x in steps):
        return min(x for x in steps if x % 2 == 0) + 1
    return max(steps) + 1
```

Не забываем поставить $+1$ в последних двух строках, потому что мы смотрим «на ход вперед». То есть, если игрок нашел такую проигрышную позицию для соперника, в которой игра длится четное количество ходов, то ему надо сделать гарантированно один ход, чтобы поставить соперника в проигрышную позицию, и только после этого выиграть за минимальное количество ходов.

3.4 Что такое @lru_cache(None)?

Вы уже могли заметить, что функция `game()` получилась рекурсивной. Рекурсия — это, конечно, красиво, но не всегда быстро. Если мы попробуем решить какую-то задачу с кодом, приведенным выше, он будет работать довольно долго. Потому что функции `game()` нужно будет проанализировать все возможные ходы игроков или, другими словами, построить полное дерево игры.

Что же тогда делать? Как ускорить код? Именно на этом моменте в игру вступает `@lru_cache(None)`. Эта магическая команда ускоряет наш код в десятки, а то и в сотни раз. Но каким образом?

Внимание! Сейчас будет сложно!

`@lru_cache(None)` создает словарь значений, где ключом является аргумент функции, а значением — результат работы функции. При повторном вызове функции с одним и тем же аргументом вместо того, чтобы тело функции выполнялось заново, функция просто возвращает значение из словаря.

Пусть программа вызывает `game()` с некоторым аргументом, например 5. Если раньше в программе уже вызывалась функция `game()` с аргументом 5, то функция «запомнила» результат своей работы. Вместо того, чтобы заново считать результат,

она просто вернет уже готовый ответ.

В программе это выглядит следующим образом:

```
from functools import lru_cache

def moves(heap):
    ...

@lru_cache(None)
def game(heap):
    ...
```

Мы просто подключаем `@lru_cache()` из модуля `itertools` и перед определением функции `game()` ставим `@lru_cache(None)`. Умными словами, данная операция называется **мемоизацией**.

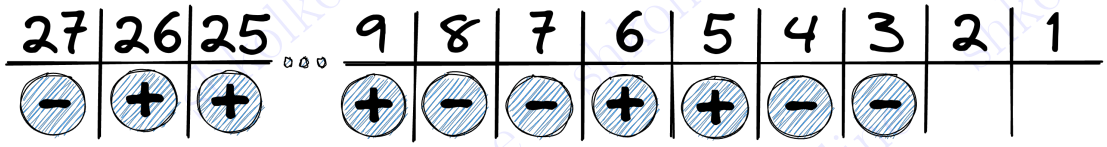
3.5 Вывод значений и их анализ

После написания функций `moves()` и `game()`, нам надо вывести на экран результат наших трудов. Для этого запустим цикл `for` по допустимым значениям для S и для каждого значения выведем соответствующий результат.

Например, для задачи выше это будет выглядедть следующим образом:

```python
from functools import lru_cache

def moves(heap):
    ...

@lru_cache(None)
def game(heap):
    ...

for x in range(1, 27):
    print(x, game(x))

# 1 None, 2 None, 3 V2, 4 V2, 5 P2,
# 6 P2, 7 V1, 8 V1, 9 P1, 10 P1,
# 11 P1, 12 P1, 13 P1, 14 P1, 15 P1
# 16 P1, 17 P1, 18 P1, 19 P1, 20 P1
# 21 P1, 22 P1, 23 P1, 24 P1, 25 P1, 26 P1
```

Сравните это с табличкой, которую мы рисовали ранее.



Классно, да? Но что такое происходит в позициях 1 и 2? Что значит `None`? Наша программа не смогла проанализировать эти позиции, потому что она может считать в «глубину» только на четыре хода. Из позиций 1 и 2 игра идет больше 4-х ходов, поэтому там и стоит `None`.