РЕШЕНИЕ ЗАДАНИЙ 19-21 ПРОГРАММИРОВАНИЕМ

суммарное время решения задач 19-21: 22 минуты

ПОРЯДОК РЕШЕНИЯ

Если Вы собираетесь решать задания 19-21 через программирование, то есть 2 варианта:

- начинать решать не с 19-го задания, а с 20-го или с 21-го;
- решить 19-е задание на бумажке, 20 и 21 – через программу.

В любом случае полученные программой ответы нужно проверить вручную, т.к. в программе нельзя задать условия вот такого типа:

- Петя не может выиграть за один ход, но может выиграть своим вторым ходом независимо от того, как будет ходить Ваня;
- у Вани нет стратегии, которая позволит ему гарантированно выиграть первым ходом, но есть выигрышная стратегия, позволяющая ему выиграть первым или вторым ходом при любой игре Пети.

И в том, и в другом случае программа выдаёт одинаковые числа (2 хода), т.е. **проверять возможность выигрыша за 1 ход нужно самостоятельно**.

О ПРОГРАММЕ

Есть несколько способов написать программу для решения задачи, наиболее оптимальной по времени будет программа, решающая задачу с помощью динамического программирования и сохраняющая результаты в таблицу. **В программе оба игрока не допускают ошибок при игре**, т.е. её можно использовать как для определения выигрышных позиций как Вани, так и Пети.

Те места в программе, которые нужно будет менять в зависимости от условий задачи, будут выделены жёлтым маркером. Всё остальное остаётся неизменным (по крайней мере для решения задач 20-21).

Значения чисел в итоговой таблице:

X – выигрыш Пети за X ходов (например, 2 это выигрыш Пети за 2 хода);

-X – выигрыш Вани за X ходов (например, -3 это выигрыш Вани за 3 хода).

СТРУКТУРА ПРОГРАММЫ

1) создание таблицы, в которой будут храниться результаты игры;
2) рекурсивное заполнение таблицы с результатами (при этом смысл чисел в таблице меняется: если в итоговом выводе число 5 означает победу Пети на 5 ходе, то при рекурсивном заполнении таблицы число 5 **в проверке** будет означать выигрыш Вани на 5 ходе, а не Пети);
3) вывод результатов (при этом никак не помечается, мог ли игрок выиграть за меньшее количество ходов при другой игре соперника: **результаты нужно перепроверять вручную**, в ответ попадает самая длинная игра).

ТИП 1: КОЛИЧЕСТВО КАМНЕЙ В КУЧАХ УВЕЛИЧИВАЕТСЯ

1: УСЛОВИЕ

Два игрока, Петя и Ваня, играют в следующую игру. Перед игроками лежат две кучи камней. Игроки ходят по очереди, первый ход делает Петя. За один ход игрок может **добавить в одну из куч (по своему выбору) один камень или увеличить количество камней в куче в два раза**. Игра завершается в тот момент, когда суммарное количество камней в кучах становится не менее **77**. Победителем считается игрок, сделавший последний ход, т.е. первым получивший такую позицию, при которой в кучах будет 77 или больше камней. В начальный момент в первой куче было **семь** камней, во второй куче – S камней; **$1 \leq S \leq 69$**.

Задание 19.

Известно, что Ваня выиграл своим первым ходом после неудачного первого хода Пети. Укажите минимальное значение S, когда такая ситуация возможна.

Задание 20.

Найдите два таких значения S, при которых у Пети есть выигрышная стратегия, причём одновременно выполняются два условия:

− Петя не может выиграть за один ход;

− Петя может выиграть своим вторым ходом независимо от того, как будет ходить Ваня.

Найденные значения запишите в ответе в порядке возрастания.

Задание 21

Найдите минимальное значение S, при котором 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.

1: СОЗДАНИЕ ТАБЛИЦЫ

19-е задание пропускаем, сначала решаем 20-е задание.

Код ниже создаёт таблицу из SUMMA*2 строк и SUMMA*2 колонок:

```
SUMMA = 77
t = [ [0]*2*SUMMA for i in range(2*SUMMA) ]
```

SUMMA – сумма, при которой завершается игра.

Нумерация колонок и строк таблице начинается с 0.

Если в ячейке таблицы t[x][y] записано число 2, то это означает, что если в начале игры в первой куче было X камней, а во второй куче было У камней, то Петя гарантированно побеждает на 2 ходу.

Если в ячейке таблицы t[x][y] записано число -3, то это означает, что если в начале игры в первой куче было X камней, а во второй куче было У камней, то Ваня гарантированно побеждает на 3 ходу.

Таблица большая, т.к. разбираются все возможные ходы и есть ходы типа (1; 76) -> (1; 76*2)

1: РЕКУРСИВНОЕ ЗАПОЛНЕНИЕ ТАБЛИЦЫ

Перебираем все возможные сочетания из X камней в первой куче и У камней во второй куче. Сумма X и У не может быть больше 76 (это конец игры). Начинаем перебирать с конца:

```
for x in range(SUMMA-1, 0, -1):
  for y in range(SUMMA-x-1, 0, -1):
    ....
```

Почему перебор начинается именно с конца, мы разбирать не будем: это нужно для того, чтобы задачу можно было решить рекурсивно. Изменять порядок перебора нельзя.

1: РЕКУРСИВНОЕ ЗАПОЛНЕНИЕ ТАБЛИЦЫ

Внутри цикла в отдельную переменную сохраним все возможные ходы Пети:

```
for x in range(SUMMA-1, 0, -1):
  for y in range(SUMMA-x-1, 0, -1):
    PetyaHod = [ t[x+1][y], t[2*x][y], t[x][y+1], t[x][2*y] ]
```

1: РЕКУРСИВНОЕ ЗАПОЛНЕНИЕ ТАБЛИЦЫ

Если в таблице в t[x][y] стоит положительное число, это значит, что при таких X и У Петя побеждает. Но в нашем случае X и У – это числа не ДО Петиного хода, это числа в ПОСЛЕ Петиного хода, т.е. это начальные условия для Вани. Поэтому если в PetyaHod оказывается положительное число, Петя не выигрывает, а, наоборот, проигрывает! Петя будет выигрывать, если в PetyaHod оказалось отрицательное число. Нули тоже считаем выигрышными для Пети (красивого объяснения здесь нет, это нужно, чтобы таблицу можно было рекурсивно заполнить):

```
for x in range(SUMMA-1, 0, -1):
  for y in range(SUMMA-x-1, 0, -1):
    PetyaHod = [ t[x+1][y], t[2*x][y], t[x][y+1], t[x][2*y] ]
    PetyaWin = [V for V in PetyaHod if V <= 0]    # выбираем отрицательные числа из PetyaHod
```

1: РЕКУРСИВНОЕ ЗАПОЛНЕНИЕ ТАБЛИЦЫ

Программа ищет наиболее оптимальные стратегии и для Пети, и для Вани, поэтому если Петя может выиграть, он будет стараться выиграть и сделать это за наименьшее количество ходов. Это значит, что он возьмёт самое маленькое отрицательное число из Ваниного результата (отрицательное – значит, Ваня проиграл, маленькое – значит, проиграл за маленькое число ходов).

.... # продолжение программы с предыдущего слайда

```
if PetyaWin:
  t[x][y] = -max(PetyaWin) + 1
```

Если Ваня проиграл за N ходов, то Петя выиграл за N + 1 ход, поэтому к ответу добавляем единицу. Минус нам нужен, т.к. Ванин проигрыш (для Вани обозначается отрицательным числом) это Петин выигрыш (для Пети обозначается положительным числом).

1: РЕКУРСИВНОЕ ЗАПОЛНЕНИЕ ТАБЛИЦЫ

Если Петя не может победить, значит, побеждает Ваня. Программа будет играть за Петю наиболее оптимальным образом (т.е. Петя будет стараться максимально растянуть игру, надеясь на ошибку противника). Если Ваня победил за N ходов, то Петя проиграл тоже за N ходов, поэтому:

```
.... # продолжение программы
    if PetyaWin:
      t[x][y] = -max(PetyaWin) + 1
    else:
      t[x][y] = -max(PetyaHod)
```

Берём максимальное положительное число из PetyaHod, т.к. Петя будет вести максимально длинную игру при проигрыше, берём со знаком минус, т.к. при безошибочной игре Вани Петя проигрывает.

1: ВЫВОД ОТВЕТА

.... # продолжение программы

```
x = 7  # в первой куче 7 камней
for y in range(1, SUMMA-x, 1):
  if t[x][y] == 2: # Петя победил на втором ходе
    print(y)   # выводим начальное количество камней во 2-й куче
```

Программа выводит всего 2 значения: 31 и 34. В условии просили найти два значения S, при которых Петя выигрывает вторым своим ходом, но не может выиграть первым. Это значит, что в данном случае значения 31 и 34 можно не перепроверять: других вариантов у нас всё равно нет.

Ответ: 31 34

1: ВСЯ ПРОГРАММА (20-Е ЗАДАНИЕ)

```
SUMMA = 77 # сумма, необходимая для выигрыша
t = [ [0]*2*SUMMA for i in range(2*SUMMA) ]


for x in range(SUMMA-1, 0, -1):
  for y in range(SUMMA-x-1, 0, -1):
    PetyaHod = [ t[x+1][y], t[2*x][y], t[x][y+1], t[x][2*y] ]  # возможные ходы из текущей позиции
    PetyaWin = [V for V in PetyaHod if V <= 0]
    if PetyaWin:
      t[x][y] = -max(PetyaWin) + 1
    else:
      t[x][y] = -max(PetyaHod)


x = 7  # в первой куче 7 камней
for y in range(1, SUMMA-x, 1):
  if t[x][y] == 2: # Петя победил на втором ходе
    print(y)
```

1: РЕШЕНИЕ 21-ГО ЗАДАНИЯ

Т.к. программа играет оптимально и за Ваню, и за Петю, то чтобы найти значения S, при которых Ваня выигрывает 2-м ходом, достаточно поменять условие в выводе:

```
.... # продолжение программы

x = 7  # в первой куче 7 камней
for y in range(1, SUMMA-x, 1):
  if t[x][y] == -2: # Ваня победил на втором ходу
    print(y)   # выводим начальное количество камней во 2-й куче
```

Программа выводит числа 30 и 33. Минимальное число – 30, но нужно проверить, что в таком случае есть какой-то Петин ход, позволяющий выиграть Ване за 1 ход.

Проверяем: (7, 30) -> (7, 60) -> (7, 120). Если Петя первым своим ходом умножит количество камней во 2-й куче в 2 раза, Ваня выиграет за 1 ход. Значение 30 подходит в качестве ответа. **Ответ: 30**

Готовая программа находится на следующем слайде.

1: РЕШЕНИЕ 21-ГО ЗАДАНИЯ

```
SUMMA = 77 # сумма, необходимая для выигрыша
t = [ [0]*2*SUMMA for i in range(2*SUMMA) ]


for x in range(SUMMA-1, 0, -1):
  for y in range(SUMMA-x-1, 0, -1):
    PetyaHod = [ t[x+1][y], t[2*x][y], t[x][y+1], t[x][2*y] ]  # возможные ходы из текущей позиции
    PetyaWin = [V for V in PetyaHod if V <= 0]
    if PetyaWin:
      t[x][y] = -max(PetyaWin) + 1
    else:
      t[x][y] = -max(PetyaHod)


x = 7  # в первой куче 7 камней
for y in range(1, SUMMA-x, 1):
  if t[x][y] == -2: # Ваня победил на втором ходе (изменения только здесь)
    print(y)
```

1: РЕШЕНИЕ 19-ГО ЗАДАНИЯ

Программа для 20 и 21 задания не совсем подходит для решения 19-го задания, т.к. в ней **Петя должен играть с ошибкой**. Чтобы найти ответ, программу можно немного изменить: оставить таблицу t с правильными решениями и добавить таблицу d для вычисления результатов при неправильном ходе Пети.

```
SUMMA = 77
t = [ [0]*2*SUMMA for i in range(2*SUMMA) ]  # таблица ОПТИМАЛЬНОЙ игры Пети
d = [ [0]*2*SUMMA for i in range(2*SUMMA) ]  # таблица для игры Пети с ошибкой
```

1: РЕШЕНИЕ 19-ГО ЗАДАНИЯ

После заполнения таблицы t добавляем кусок кода, который заполняет таблицу d. В таблице d Ваня должен выигрывать за 1 ход, а Петя должен проигрывать за 1 ход:

```
# СТРОИМ ТАБЛИЦУ, В КОТОРОЙ ПЕТЯ ДОПУСКАЕТ ОШИБКУ И ПРОИГРЫВАЕТ ЗА 1 ХОД
for x in range(SUMMA-1, 0, -1):
  for y in range(SUMMA-x-1, 0, -1):
    PetyaHod = [ t[x+1][y], t[2*x][y], t[x][y+1], t[x][2*y] ]
    PetyaLose = [V for V in PetyaHod if V == 1]   # Ваня выигрывает за 1 ход
    if PetyaLose:
      d[x][y] = -1 # Петя проигрывает за 1 ход
```

Остальные варианты нас просто не интересуют, поэтому мы их не заполняем.

1: РЕШЕНИЕ 19-ГО ЗАДАНИЯ

Результат выводим, естественно, для таблицы d:

```
# ВЫВОД ОТВЕТА
x = 7  # в первой куче 7 камней
for y in range(1, SUMMA-x, 1):
  if d[x][y] == -1: # Ваня выигрывает за 1 ход после Петиной ошибки
    print(y)
```

Вывод программы: числа 18, 19, 20, 21 … Минимальное число – 18. Ответ: 18

Готовая программа находится на следующем слайде.

1: РЕШЕНИЕ 19-ГО ЗАДАНИЯ

```
SUMMA = 77
t = [ [0]*2*SUMMA for i in range(2*SUMMA) ]  # таблица ОПТИМАЛЬНОЙ игры Пети
d = [ [0]*2*SUMMA for i in range(2*SUMMA) ]  # таблица для игры Пети с ошибкой

# СТРОИМ ТАБЛИЦУ, В КОТОРОЙ ПЕТЯ НЕ ДОПУСКАЕТ ОШИБОК
for x in range(SUMMA-1, 0, -1):            # x - количество камней в первой куче
  for y in range(SUMMA-x-1, 0, -1):        # y - количество камней во второй куче
    PetyaHod = [ t[x+1][y], t[2*x][y], t[x][y+1], t[x][2*y] ]  # возможные ходы из текущей позиции
    PetyaWin = [c for c in PetyaHod if c <= 0]
    if PetyaWin:                  # если Петя может выиграть, он будет стараться выиграть
      t[x][y] = -max(PetyaWin) + 1
    else:                         # если Петя проигрывает
      t[x][y] = -max(PetyaHod)

# СТРОИМ ТАБЛИЦУ, В КОТОРОЙ ПЕТЯ ДОПУСКАЕТ ОШИБКУ И ПРОИГРЫВАЕТ ЗА 1 ХОД
for x in range(SUMMA-1, 0, -1):            # x - количество камней в первой куче
  for y in range(SUMMA-x-1, 0, -1):        # y - количество камней во второй куче
    PetyaHod = [ t[x+1][y], t[2*x][y], t[x][y+1], t[x][2*y] ]  # возможные ходы из текущей позиции
    PetyaLose = [V for V in PetyaHod if V == 1]   # Ваня выигрывает за 1 ход
    if PetyaLose:                  # если Петя может проиграть Ване за 1 ход, он должен проиграть
      d[x][y] = -1                 # Петя проигрывает

# ВЫВОД ОТВЕТА
x = 7  # в первой куче 7 камней
for y in range(1, SUMMA-x, 1):
  if d[x][y] == -1: # Ваня выигрывает за 1 ход после Петиной ошибки
    print(y, d[x][y])
```

1: О ПОСТРОЕНИИ ТАБЛИЦЫ t

Если Вы запускаете программу в IDLE, то t у Вас сохраняется после первого решения (20-го задания) и перестраивать t не нужно. Тогда при решении 21-го и 19-го заданий t можно не строить заново. Т.е. если Вы уже решили 20-е задание, для решения 21-го достаточно запустить только код для вывода ответа:

```
x = 7  # в первой куче 7 камней
for y in range(1, SUMMA-x, 1):
  if t[x][y] == -2: # Ваня победил на втором ходе (изменения только здесь)
    print(y)
```

1: О ПОСТРОЕНИИ ТАБЛИЦЫ t

Программа для решения 19-го задания **при построенной таблице t**:

```
d = [ [0]*2*SUMMA for i in range(2*SUMMA) ]  # таблица для игры Пети с ошибкой

# СТРОИМ ТАБЛИЦУ, В КОТОРОЙ ПЕТЯ ДОПУСКАЕТ ОШИБКУ И ПРОИГРЫВАЕТ ЗА 1 ХОД
for x in range(SUMMA-1, 0, -1):
  for y in range(SUMMA-x-1, 0, -1):
    PetyaHod = [ t[x+1][y], t[2*x][y], t[x][y+1], t[x][2*y] ]  # возможные ходы из текущей позиции
    PetyaLose = [V for V in PetyaHod if V == 1]
    if PetyaLose:
      d[x][y] = -1

# ВЫВОД ОТВЕТА
x = 7  # в первой куче 7 камней
for y in range(1, SUMMA-x, 1):
  if d[x][y] == -1: # Ваня
```

1: ИТОГ

Даже если Вы не до конца понимаете отдельные участки кода (например, строчку создания таблицы t), эти отдельные строчки всегда можно выучить. Смотрите сами, что Вам проще: выучить наизусть несколько строчек кода (но написать их на экзамене нужно будет без единой ошибки) или решить задачу математически.

Код программ для заданий 19-21 находится в курсе в файле "Пример 19-21.txt".

ТИП 2: КОЛИЧЕСТВО КАМНЕЙ В КУЧАХ УМЕНЬШАЕТСЯ

2: УСЛОВИЕ

Два игрока, Петя и Ваня, играют в следующую игру. Перед игроками лежат две кучи камней. Игроки ходят по очереди, первый ход делает Петя. 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 больше, чем убирается). Игра завершается в тот момент, когда суммарное количество камней в кучах становится не более 40. Победителем считается игрок, сделавший последний ход, то есть первым получивший позицию, в которой в кучах будет 40 или меньше камней. В начальный момент в первой куче было 20 камней, во второй куче — S камней, S > 20.

Задание 19.

Известно, что Ваня выиграл своим первым ходом после неудачного первого хода Пети. Укажите максимальное значение S, когда такая ситуация возможна.

Задание 20.

Найдите три наименьших значения S, при которых у Пети есть выигрышная стратегия, причём одновременно выполняются два условия:

— Петя не может выиграть за один ход;

— Петя может выиграть своим вторым ходом независимо от того, как будет ходить Ваня.

Найденные значения запишите в ответе в порядке возрастания без разделительных знаков.

Задание 21

Найдите максимальное значение S, при котором 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.

2: ИЗМЕНЕНИЕ ПРОГРАММЫ ПОД 2-Й ТИП

Для того, чтобы рекурсивно заполнить таблицу при уменьшении количества камней в кучах, требуется:

1) определиться с максимальным размером таблицы (теперь мы сверху не ограничены)
2) понять, как запустить циклы в обратном порядке;
3) понять, как изменить формулы (в вычислении следующего хода).

Максимальный размер возьмём таким: SUMMA * 16. Такой размер таблицы позволяет вычислить игру минимум в 4 хода.

```
SUMMA = 40
MAXS = 16*SUMMA
t = [ [0]*MAXS for i in range(MAXS) ]
```

Максимальный размер таблицы во всех задачах на уменьшение берём хотя бы 16*SUMMA.

2: ИЗМЕНЕНИЕ ПРОГРАММЫ ПОД 2-Й ТИП

Запуск циклов в обратном порядке:

```
# СТРОИМ ТАБЛИЦУ ИГРЫ
for x in range(1, MAXS, 1):
  for y in range(max(SUMMA-x+1, 1), MAXS, 1):
    .....

# ВЫВОД ОТВЕТА
x = 20
for y in range(max(SUMMA-x+1, 1), MAXS, 1):
  ....
```

Если не очень понятно, почему нужно именно так запустить циклы, остаётся одно: зазубрить эти три строчки кода.

2: ИЗМЕНЕНИЕ ПРОГРАММЫ ПОД 2-Й ТИП

Теперь изменим формулы в PetyaHod.

Убрать камень из одной кучи легко:

PetyaHod = [t[x-1][y], t[???][y], t[x][y-1], t[x][???]]

Теперь уменьшим количество камней в куче в 2 раза, причём если камней – нечётное количество, **лишний камень заберём**:

PetyaHod = [t[x-1][y], t[x//2][y], t[x][y-1], t[x][y//2]]

Но по условию ведь нам нельзя забирать лишний камень, он должен остаться в куче, поэтому:

PetyaHod = [t[x-1][y], t[x//2 + x%2][y], t[x][y-1], t[x][y//2 + y%2]]

2: ИЗМЕНЕНИЕ ПРОГРАММЫ ПОД 2-Й ТИП

Всё. Больше никаких изменений нет.

Программы для решения заданий 20, 21 и 19 на следующих слайдах.

2: ВСЯ ПРОГРАММА (20-Е ЗАДАНИЕ)

```
SUMMA = 40  # минимальное количество камней
MAXS = 16*SUMMA # берём хотя бы 16
t = [ [0]*MAXS for i in range(MAXS) ]

# СТРОИМ ТАБЛИЦУ БЕЗОШИБОЧНОЙ ИГРЫ
for x in range(1, MAXS, 1):
  for y in range(max(SUMMA-x+1, 1), MAXS, 1):
    PetyaHod = [ t[x-1][y], t[x//2 + x%2][y], t[x][y-1], t[x][y//2 + y%2] ]
    PetyaWin = [V for V in PetyaHod if V <= 0]
    if PetyaWin:
      t[x][y] = -max(PetyaWin) + 1
    else:
      t[x][y] = -max(PetyaHod)

# ВЫВОД ОТВЕТА ДЛЯ 20-ГО ЗАДАНИЯ
x = 20
for y in range(max(SUMMA-x+1, 1), MAXS, 1):
  if t[x][y] == 2:   # Петя выигрывает в 2 хода
    print(y)
```

Программа выводит числа 42, 43, 61, 81, 82.
Вывод программы нужно перепроверить (что Петя не может выиграть за 1 ход).
После перепроверки оказывается, что первые три наименьших значения (42, 43, 61) подходят.
Ответ: 42 43 61

2: ВСЯ ПРОГРАММА (21-Е ЗАДАНИЕ)

```
SUMMA = 40  # минимальное количество камней
MAXS = 16*SUMMA # берём хотя бы 16
t = [ [0]*MAXS for i in range(MAXS) ]

# СТРОИМ ТАБЛИЦУ БЕЗОШИБОЧНОЙ ИГРЫ
for x in range(1, MAXS, 1):
  for y in range(max(SUMMA-x+1, 1), MAXS, 1):
    PetyaHod = [ t[x-1][y], t[x//2 + x%2][y], t[x][y-1], t[x][y//2 + y%2] ]
    PetyaWin = [V for V in PetyaHod if V <= 0]
    if PetyaWin:
      t[x][y] = -max(PetyaWin) + 1
    else:
      t[x][y] = -max(PetyaHod)

# ВЫВОД ОТВЕТА ДЛЯ 21-ГО ЗАДАНИЯ
x = 20
for y in range(max(SUMMA-x+1, 1), MAXS, 1):
  if t[x][y] == -2:   # Ваня выигрывает в 2 хода
    print(y)
```

Программа выводит единственное число - 44. Ничего перепроверять не будем, всё равно других вариантов нет.
Ответ: 44

2: ВСЯ ПРОГРАММА (19-Е ЗАДАНИЕ)

```
SUMMA = 40  # минимальное количество камней
MAXS = 16*SUMMA # берём хотя бы 16
t = [ [0]*MAXS for i in range(MAXS) ]

# СТРОИМ ТАБЛИЦУ БЕЗОШИБОЧНОЙ ИГРЫ
for x in range(1, MAXS, 1):
  for y in range(max(SUMMA-x+1, 1), MAXS, 1):
    PetyaHod = [ t[x-1][y], t[x//2 + x%2][y], t[x][y-1], t[x][y//2 + y%2] ]
    PetyaWin = [V for V in PetyaHod if V <= 0]
    if PetyaWin:
      t[x][y] = -max(PetyaWin) + 1
    else:
      t[x][y] = -max(PetyaHod)

# РЕШЕНИЕ 19-ГО ЗАДАНИЯ
d = [ [0]*MAXS for i in range(MAXS) ]

# СТРОИМ ТАБЛИЦУ, В КОТОРОЙ ПЕТЯ ДОПУСКАЕТ ОШИБКУ И ПРОИГРЫВАЕТ ЗА 1 ХОД
for x in range(1, MAXS, 1):
  for y in range(max(SUMMA-x+1, 1), MAXS, 1):
    PetyaHod = [ t[x-1][y], t[x//2 + x%2][y], t[x][y-1], t[x][y//2 + y%2] ]
    PetyaLose = [V for V in PetyaHod if V == 1]
    if PetyaLose:
      d[x][y] = -1

# ВЫВОД ОТВЕТА ДЛЯ 19-ГО ЗАДАНИЯ
x = 20
for y in range(max(SUMMA-x+1, 1), MAXS, 1):
  if d[x][y] == -1:   # Ваня выигрывает за 1 ход после Петиной ошибки
    print(y)
```

Программа выводит много чисел: 22, 23, 24 ... 78, 79, 80. Максимальное - 80.

Ответ: 80