Задача A. Двоичные векторы

Данная задача является частным случаем задачи, рассмотренной на лекции. Пример решения задачи на языке Python приведен ниже:

```
def gen(n, s):
    if len(s) == n:
        print(s)
        return

    gen(n, s + '0')
    gen(n, s + '1')

n = int(input())
gen(n, '')
```

Задача B. Ровно k единиц

В данной задаче при выполнении перебора достаточно поддерживать количество поставленных ранее единиц (в коде это переменная `ones`). На текущую позицию можно поставить `0` только в том случае, если оставшихся позиций хватит, чтобы дополнить количество единиц до k. Таким образом, можно поставить `0` только в случае, если количество оставшихся позиций $n - |s| - 1$ не меньше, чем $k - \text{ones}$.

Пример решения задачи на языке Python приведен ниже:

```
def gen(n, k, ones, s):
    if len(s) == n:
        print(s)
        return

    if n - len(s) - 1 >= k - ones:
        gen(n, k, ones, s + '0')
    if ones < k:
        gen(n, k, ones + 1, s + '1')

n, k = map(int, input().split())
gen(n, k, 0, '')
```

Задача C. Все перестановки

Данная задача была рассмотрена на лекции. Пример решения задачи на языке Python приведен ниже:

```
def gen(n, used, p):
    if len(p) == n:
        print(*p)
        return

    for i in range(n):
        if not used[i]:
            used[i] = True
            p.append(i + 1)
            gen(n, used, p)
            p.pop()
            used[i] = False

n = int(input())
gen(n, [False for _ in range(n)], [])
```

Задача D. Все сочетания из n по k

Данная задача была рассмотрена на лекции. Пример решения задачи на языке Python приведен ниже:

```
def gen(n, k, last, p):
    if len(p) == k:
        print(*p)
        return

    l = last + 1
    r = n - k + len(p) + 1
    for i in range(l, r + 1):
        p.append(i)
        gen(n, k, i, p)
        p.pop()

n, k = map(int, input().split())
gen(n, k, 0, [])
```

Задача E. Все разбиения на слагаемые

Данная задача была рассмотрена на лекции. Пример решения задачи на языке Python приведен ниже:

```
def gen(n, last, p):
    if n == 0:
        print(*p, sep='+')
        return

    l = last
    r = n // 2
    for i in range(l, r + 1):
        p.append(i)
        gen(n - i, i, p)
        p.pop()

    p.append(n)
    gen(0, n, p)
    p.pop()

n = int(input())
gen(n, 1, [])
```

Задача F. Все правильные скобочные последовательности

Будем поддерживать разницу в количестве поставленных открывающих и закрывающих скобок (переменная `balance` в коде). Поставить открывающую скобку можно только в том случае, если оставшихся позиций (которых $2n - |p|$) хватит, чтобы закрыть все открытые скобки (которых будет $\text{balance} + 1$). Поставить закрывающую скобку можно только в том случае, если сейчас открыта хотя бы одна скобка.

Пример решения задачи на языке Python приведен ниже:

```
def gen(n, balance, p):
    if len(p) == 2 * n:
        print(p)
        return
```

```
    if balance + 1 <= 2 * n - len(p):
        gen(n, balance + 1, p + '(')
    if balance > 0:
        gen(n, balance - 1, p + ')')

n = int(input())
gen(n, 0, '')
```

Задача G. Шуткасперестановкой

Заметим, что в условии сказано, что перестановка содержит не более 50 чисел. Соответственно, каждое число может быть однозначным или двузначным.

Будем перебирать все способы разрезать строку на однозначные и двузначные числа таким. В процессе перебора можно проверять, встречалось ли данное число ранее. Если такое число уже встречалось, то добавить его еще раз нельзя, так как перестановка содержит каждый элемент ровно один раз. Также в конце необходимо проверить, что если размер перестановки равен n, в ней встречаются все числа от 1 до n.

Пример решения задачи на языке Python:

```
import sys

def gen(pref, s, p, used):
    if pref == len(s):
        cnt = len(p)
        ok = True
        for i in range(1, cnt + 1):
            if not used[i]:
                ok = False
                break

        if ok:
            print(*p)
            sys.exit(0)

        return

    d1 = ord(s[pref]) - ord('0')
    if not used[d1]:
        used[d1] = True
        p.append(d1)
        gen(pref + 1, s, p, used)
        used[d1] = False
        p.pop()

    if pref + 1 < len(s):
        d2 = ord(s[pref + 1]) - ord('0')
        d = d1 * 10 + d2
        if d <= 50 and not used[d]:
            used[d] = True
            p.append(d)
            gen(pref + 2, s, p, used)
            used[d] = False
            p.pop()
```

```
s = input()
gen(0, s, [], [False for _ in range(51)])
```

Задача H. Самый лучший робот

На каждом шаге будем перебирать две детали i и j, которые мы будем объединять. Также будем перебирать способ объединения деталей. В конце, когда останется ровно одна деталь, вернем в качестве ответа ее жесткость.

Пример решения задачи на языке Python приведен ниже:

```
def gen(a, d):
    if len(a) == 1:
        return abs(a[0] - d)

    res = min(abs(i - d) for i in a)
    for i, ai in enumerate(a):
        for j in range(i):
            newa = a[:j].copy() + a[j + 1:i].copy() + a[i + 1:].copy()

            newa.append(a[i] + a[j])
            res = min(res, gen(newa, d))

            newa[-1] = a[i] * a[j]
            res = min(res, gen(newa, d))
    return res

n, d = map(int, input().split())
a = [int(i) for i in input().split()]

print(gen(a, d))
```

Задача I. Ханойские башни

Заметим, что нам нужно положить самый большой блок на третий стержень. Но это возможно только тогда, когда все остальные блоки лежат на втором стержне, ведь иначе если какой-то лежит на первом, то мы не сможем взять больший блок. Аналогично со случаем, когда какой-то блок лежит на третьем стержне. Но это сводит задачу к тому, что нужно положить пирамиду из $n-1$ блоков на второй блок. А это просто уменьшает задачу в размерности количества блоков (ведь на самый большой блок можно класть любой из оставшихся, поэтому можно считать, что большого блока вообще нет). Но так же мы можем продолжать дальше, спускаясь рекурсивно вниз. А базовый случай происходит тогда, когда остается только один блок — мы просто перекладываем его. Теперь, когда мы выполнили рекурсивный вызов для $n-1$ блоков, мы переставляем n-ый блок. И нам нужно опять повесить эту пирамиду из $n-1$ блоков на этот n-й блок.

Это решение можно реализовать с помощью рекурсивной функции от трех параметров: размер пирамиды, которую мы собираемся перевесить, и два номера-номера начального и конечного стержней.

Пример решения задачи на языке Python приведен ниже:

```
def gen(n, fr, to, other):
    if n == 1:
        print(n, fr, to)
        return

    gen(n - 1, fr, other, to)
    print(n, fr, to)
    gen(n - 1, other, to, fr)
```

```
n = int(input())
gen(n, 1, 3, 2)
```

Задача J. Сортировка мусора

(Автор задачи — *Киндер М.И.*)

Пусть $a[1][1]$, $a[2][1]$, $\dots a[n][1]$ — количество отходов вида 1 в контейнерах с номерами 1, 2, …, n соответственно. Общее количество отходов вида 1, очевидно, равно

$$s[1] = a[1][1] + a[2][1] + \ldots + a[n][1] = \sum_{k=1}^{n} a[k][1].$$

Предположим, что после сортировки все отходы вида 1 окажутся в контейнере с номером i_1. Тогда количество операций для перемещения отходов вида 1 в этот контейнер будет равно $s[1]-a[i_1][1]$. Аналогичным образом, подсчитаем количество операций для перемещения отходов вида 2, 3, …, n в контейнеры с номерами i_2, i_3, …, i_n соответственно. Тогда общее число операций, необходимое для сортировки всего мусора, равно

$$s = (s[1] - a[i_1][1]) + (s[2] - a[i_2][2]) + \ldots + (s[n] - a[i_n][n]) = \sum_{i=1}^{n}\sum_{k=1}^{n} a[i][k] - \sum_{k=1}^{n} a[i_k][k].$$

Двойная сумма в правой части этого равенства — это сумма всех чисел исходного массива $a[n][n]$, и она не зависит от выбора контейнеров i_1, i_2, …, i_n. Число операций s будет наименьшим только, если сумма $\sum_{k=1}^{n} a[i_k][k]$ принимает наибольшее значение. Другими словами, в двумерной таблице-массиве $a[n][n]$ необходимо выбрать n элементов $a[i_1][1]$, $a[i_2][2]$, …, $a[i_n][n]$ — по одному из каждой строки и из каждого столбца — так, чтобы сумма выбранных чисел была *наибольшей*.

Для оценки сложности алгоритма подсчитаем количество вариантов выбора таких n элементов в массиве $a[n][n]$. Это количество совпадает с количеством вариантов расстановки n ладей на шахматной доске $n \times n$, в которых ни одна из них не угрожает другой, то есть равно $n! = 1 \cdot 2 \cdot \ldots \cdot n$.

Подзадача 1. При $n = 2$ количество вариантов в переборном алгоритме равно $2! = 2$. Другими словами, достаточно выбрать наибольшее число из двух сумм $a[1][1] + a[2][2]$ и $a[1][2] + a[2][1]$. Это решение оценивается в 20 баллов.

Подзадача 2. При $n = 3$ количество вариантов в переборном алгоритме равно $3! = 6$ и необходимо выбрать наибольшую из шести сумм, каждая из которых состоит из трёх слагаемых. Это решение оценивается ещё в 20 баллов.

Подзадача 3. При $n = 4$ задача также решается аналогичным перебором.

Подзадача 4. Для реализации переборного алгоритма в общей ситуации сгенерируем все $n!$ перестановок n-элементного множества $\{1, 2, \ldots, n\}$. Затем для каждой перестановки $(i_1, i_2, \ldots, i_n)$ вычисляем сумму вида $\sum_{k=1}^{n} a[i_k][k]$. Теперь осталось выбрать наибольшую среди $n!$ таких сумм.

При $n \leqslant 10$ количество вариантов не превосходит $10! < 4 \cdot 10^6$.

Такое переборное решение оценивается в 100 баллов.

Пример решения задачи на языке Python:

```
ans = 10 ** 18
a = []
s = []

def check(p):
    global a

    n = len(p)
    cur = 0
    for i in range(n):
```

```
        cur += s[i] - a[p[i]][i]
    return cur

def gen(n, used, p):
    global ans

    if len(p) == n:
        ans = min(ans, check(p))
        return

    for i in range(n):
        if not used[i]:
            used[i] = True
            p.append(i)
            gen(n, used, p)
            used[i] = False
            p.pop()

n = int(input())
s = [0 for _ in range(n)]
for i in range(n):
    a.append([int(i) for i in input().split()])
    for j in range(n):
        s[j] += a[i][j]
gen(n, [False for _ in range(n)], [])
print(ans)
```