# A. Конфеты

(Автор задачи — *Киндер М.И.*)

Подсчитаем количество конфет, которое должно быть одинаковым в каждой коробке. Для этого найдём общее количество конфет во всех коробках $sum = a_1 + a_2 + \ldots + a_n$ и разделим на число коробок $n$. Если значение этого *среднего арифметического* — целое, то есть остаток от деления $sum$ на $n$ равен нулю, то Карлсону ничего съедать и перекладывать не придётся. В общем случае, остаток $eat := sum \bmod n$ определяет число конфет, которое нужно съесть Карлсону. Теперь оставшиеся конфеты можно разложить поровну между коробками. Для этого из каждой коробки, где число конфет $a_i$ больше среднего арифметического $mean := sum \operatorname{div} n$, нужно переложить $a_i - mean$ конфет в коробки с меньшим числом конфет. Общее число перекладываемых конфет равно сумме по всем таким $i$.

Приведём основной фрагмент кода на языке Pascal:

```
eat := sum mod n;
mean := sum div n;

sum := 0;
for i := 1 to n do
    if a[i] > mean then Inc(sum, a[i] – mean);

write(eat, ' ', sum – eat);
```

# B. Штабеля

(Автор задачи — *Киндер М.И.*)

Все числа каждого набора отсортируем по возрастанию. Теперь осталось проверить, что каждое число набора не меньше суммы всех предыдущих. Если это условие выполняется для всех $k$ чисел набора (кроме первого), выводим yes; иначе — выводим no.
Оценим сложность этого алгоритма. Для сортировки массива из $k$ элементов понадобится $O(k \log k)$ операций, для проверки «прочности» штабеля нужно еще $O(k)$ операций. Действительно, когда проверяется условие «прочности» для очередного числа $weight[i+1]$ набора, нам нужно вычислить сумму $sum[i]$ всех стоящих слева от $weight[i+1]$ чисел, причём $sum[i]$ получается из предыдущего значение суммы $sum[i-1]$ добавлением единственного слагаемого $weight[i]$. Значит, для проверки условия $sum[i] \le weight[i+1]$ на очередном шаге потребуется всего две дополнительные операции. Поэтому проверка прочности штабеля (после сортировки чисел) потребует еще $O(k)$ операций. Поскольку таких наборов $n$, общая сложность алгоритма $O(n \cdot k \log k)$.

Приведём основной фрагмент кода на языке Pascal:

```
flag := true;
sum := weight[1];
for i := 2 to k do
    if weight[i] < sum then begin
       flag := false; break;
       end
    else Inc(sum, weight[i]);
if flag then writeln('yes') else writeln('no');
```

## C. Цепочка вычитаний

(Автор задачи — *Киндер М.И.*)

Прежде всего, отметим, что на любом шаге среди чисел ровно одно нечётное. Вначале — это число 1. На очередном шаге вычитаются либо два чётных числа, либо чётное и нечётное числа, так что количество нечётных снова остаётся прежним (и равным одному). Поэтому последнее оставшееся число всегда будет нечётным. Значит, если требуемое число $k$ — чётное, выводим –1.

Покажем теперь, как получить *любое нечётное* число $k < 2^n$.

Предположим, что мы уже умеем получать *все нечётные* $k < 2^{n-1}$ с помощью чисел 1, 2, $2^2$, … , $2^{n-1}$, и пусть теперь задан набор 1, 2, $2^2$, …, $2^{n-1}$, $2^n$. Рассмотрим нечётные числа в промежутке от 1 до $2^n$, отметим его середину — число $2^{n-1}$.

Если заданное нечётное $k < 2^{n-1}$, то по предположению его можно получить с помощью чисел 1, 2, $2^2$, … , $2^{n-1}$. Но тогда его можно получить и с помощью исходного набора 1, 2, $2^2$, …, $2^{n-1}$, $2^n$. Для этого достаточно на первом шаге заменить пару чисел $2^{n-1}$ и $2^n$ их разностью, равной $2^n - 2^{n-1} = 2^{n-1}$, и мы вновь приходим к набору 1, 2, $2^2$, … , $2^{n-1}$, из которого число $k$ уже получается.

Если же нечётное $k > 2^{n-1}$, то заменим его на число $k' = 2^n - k < 2^{n-1}$, которое — опять же по предположению — можно получить из набора 1, 2, $2^2$, … , $2^{n-1}$. Поэтому сначала с помощью $(n-1)$ операций вычитания получаем число $k' = 2^n - k$, а затем последней операцией заменяем пару чисел $2^n$ и $k'$ их разностью $2^n - k' = 2^n - (2^n - k) = k$.

Осталось правильно реализовать указанный алгоритм. Это можно сделать, например, с помощью стека.

## D. Скобочки

(Автор задачи — *Киндер М.И.*)

Решение задачи основано на использовании динамического программирования.

Пусть $m[n]$ — количество правильных скобочных последовательностей, содержащих $n$ пар скобок. Произвольная правильная последовательность скобок длины $2n$ получается по правилу $(ST)$, причём $S$ и $T$ — правильные последовательности.

Если $S$ — пустая строка, то правильная скобочная последовательность $T$ состоит из $(n-1)$ пар скобок. Количество скобочных последовательностей этого типа, очевидно, равно $m[n-1]$.

Пусть теперь $S$ — непустая строка и содержит $k \ge 1$ пар скобок. Строка $S$ обязательно начинается с открывающейся скобки «(», которой соответствует некоторая закрывающаяся скобка «)». Между этими скобками расположена правильная скобочная последовательность из $(k-1)$ пар скобок; их количество равно $m[k-1]$. После строки $S$ записана правильная скобочная последовательностей $T$ из $n-k-1$ пар скобок. (Здесь мы учли пару внешних скобок в исходной строке $(ST)$.) Количество строк $T$ равно $m[n-k-1]$. По правилу произведения общее число всех таких комбинаций равно $m[k-1] \cdot m[n-k-1]$. Суммируя по всем $k$ от 1 до $n-1$, получим окончательную формулу для подсчета чисел $m[n]$:

$$m[n] = m[n-1] + \sum_{k=1}^{n-1} m[k-1] \cdot m[n-k-1],$$

с начальным условием $m[0] = 1$. Поскольку числа $m[n]$ растут достаточно быстро, вычисления по этой формуле нужно делать по модулю $(10^9 + 9)$.

# E. Эх, дороги…

(Автор задачи — *Киндер М.И.*)

Математическая формулировка задачи выглядит следующим образом.
Дан неориентированный граф. Требуется найти такой его подграф, который содержит наименьшее число рёбер и в котором все его вершины, достижимые *до* удаления рёбер, остаются достижимыми и после удаления «лишних» рёбер.

Ясно, что вершины графа разбиваются на несколько групп так, что из любой вершины одной группы можно попасть в любую другую вершину этой группы, и между разными группами пути не существует. Такая группа вершин называется *компонентой связности*. У исходного графа может быть, конечно, несколько таких компонент. После удаления рёбер в каждой компоненте связности не должно быть циклов, иначе можно будет удалить по крайней мере ещё одно ребро с сохранением условия достижимости. Связный граф без циклов называется *деревом*, а их совокупность — *лесом*.
Таким образом, задача сводится к построению для каждой компоненты связности соответствующего дерева, которое соединяет все вершины этой компоненты. Такое дерево называется *минимальным остовом*.
Для решения подзадачи 1 подходят переборные алгоритмы.
Для решения подзадач 2 и 3 можно воспользоваться алгоритмом Крускала для построения минимального остовного дерева.
Вначале каждую вершину из одной компоненты помещаем в своё дерево, а затем соединяем эти деревья, объединяя на каждой итерации два дерева некоторым ребром. В процессе объединения перебираются все рёбра от первого до последнего, и если у текущего ребра его концы принадлежат разным поддеревьям, эти поддеревья объединяются, а ребро добавляется к ответу. По окончании перебора всех рёбер все вершины окажутся принадлежащими одному остову, и ответ получен.
Простейшая реализация этого алгоритма имеет асимптотику $O(m^2)$. Используя структуру данных «система непересекающихся множеств», можно получить более быструю реализацию алгоритма Крускала с асимптотикой $O(n + m)$. Более подробно познакомиться с деталями этой реализации и структурой данных «система непересекающихся множеств» можно на сайте:
http://e-maxx.ru/algo/mst_kruskal_with_dsu

*Председатель жюри М.И. Киндер*