Университет ИТМО

Факультет ПИиКТ

Алгоритмы и структуры данных

Лабораторная работа №1

Выполнила: Наумова Надежда

Группа P3201

Преподаватель: Тропченко А. А.

Санкт-Петербург

2020 г.

## **2025. Стенка на стенку**

## Постановка задачи:

Бокс, каратэ, самбо… Классические боевые единоборства пресытили аудиторию. Поэтому известный спортивный канал запускает новый формат соревнований, основанный на традиционной русской забаве — боях стенка на стенку. В соревновании могут участвовать от двух до *k* команд, каждая из которых будет соперничать с остальными. Всего в соревновании примут участие *n* бойцов. Перед началом боя они должны разделиться на команды, каждый боец должен войти ровно в одну команду. За время боя два бойца сразятся, если они состоят в разных командах. Организаторы считают, что популярность соревнований будет тем выше, чем больше будет количество схваток между бойцами. Помогите распределить бойцов по командам так, чтобы максимизировать количество схваток между бойцами, и выведите это количество.

### Исходные данные

В первой строке дано количество тестов *T* (1 ≤ *T* ≤ 10). В следующих *T* строках перечислены тесты. В каждой из них записаны целые числа *n* и *k* через пробел (2 ≤ *k* ≤ *n* ≤ 104).

### Результат

Для каждого теста в отдельной строке выведите одно целое число — ответ на задачу.

## Описание решения

Каждый боец участвует по разу в схватке с каждым, кто не состоит в его команде. Чтобы в итоге было как можно больше схваток, каждый боец должен биться с как можно большим количеством участников - должно быть как можно меньше бойцов, с которыми он не бьётся. Исходя из этого, равномерно «размазываем» участников по командам, а если N (кол-во бойцов) не кратно K (кол-ву команд), то оставшихся также распределяем по одному в команду, добиваясь того, чтобы количество участников любых команд отличалось на 1 либо не отличалось вовсе.

Предположим, игроки разбиты не поровну, тогда существуют команды, в которых количество участников отличается минимум на 2. Пусть в первой такой команде n человек, тогда во второй n + x (причем x ≥ 2) человек. Докажем, что перенос человек из большей команды в меньшую увеличит суммарное количество боев:

vs

vs

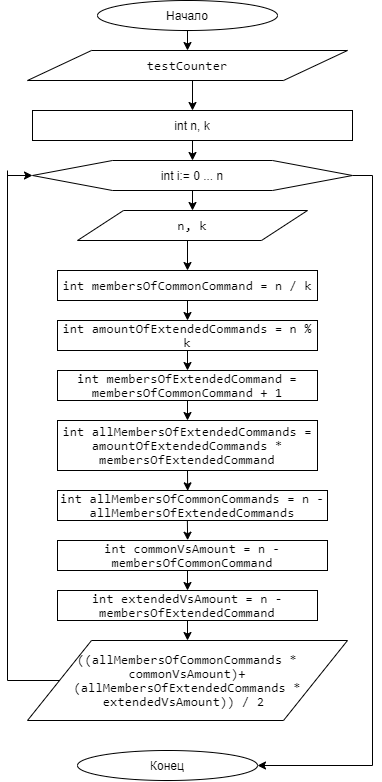
0 vs

0 < (т.к. x ≥ 2)

Верно, значит количество боев увеличилось, при этом количество команд и игроков осталось неизменным.

После этого просто формулами считаем количество схваток.

## Схема решения



## Исходный код

// подключение стандартной библиотеки  
#include <iostream>  
  
// использование пространства имен std  
using namespace std;  
  
int main() {  
 //объявление переменной, в которую кладется кол-во тестов  
 int testCounter;  
 //кладем кол-во тестов  
 cin >> testCounter;  
 //объявление переменных, в которые кладутся кол-ва бойцов и команд соответственно  
 int n, k;  
 //запускаем цикл на количество тестов  
 for (int i = 0; i < testCounter; i ++) {  
 //объявление переменных, в которые кладутся кол-ва бойцов и команд соответственно  
 cin >> n >> k;  
 // количество участников обычной команды  
 int membersOfCommonCommand = n / k;  
  
 //количество расширенных команд  
 int amountOfExtendedCommands = n % k;  
  
 //количество участников расширенной команды  
 int membersOfExtendedCommand = membersOfCommonCommand + 1;  
  
 //всего участников в расширенных командах  
 int allMembersOfExtendedCommands = amountOfExtendedCommands \* membersOfExtendedCommand;  
  
 //всего участников в обычных командах  
 int allMembersOfCommonCommands = n - allMembersOfExtendedCommands;  
  
 //соперников одной обычной команды  
 int commonVsAmount = n - membersOfCommonCommand;  
  
 //соперников одной расширенной команды  
 int extendedVsAmount = n - membersOfExtendedCommand;  
  
 // вывод ответа  
 cout << ((allMembersOfCommonCommands \*  
commonVsAmount)+(allMembersOfExtendedCommands \* extendedVsAmount)) / 2 << endl;  
 }  
 return 0;  
}

## **1005. Куча камней**

## Постановка задачи:

У вас есть несколько камней известного веса *w*1, …, *wn*. Напишите программу, которая распределит камни в две кучи так, что разность весов этих двух куч будет минимальной.

### Исходные данные

Ввод содержит количество камней *n* (1 ≤ *n* ≤ 20) и веса камней *w*1, …, *wn* (1 ≤ *wi* ≤ 100 000) — целые, разделённые пробельными символами.

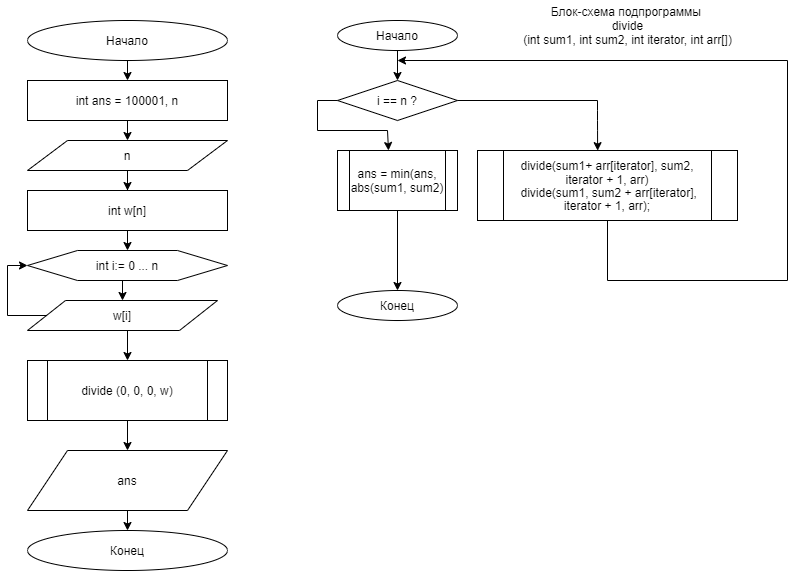
### Результат

Ваша программа должна вывести одно число — минимальную разность весов двух куч.

## Описание решения

Количество камней не превысит 20, поэтому сделаем рекурсивный полный перебор, добавляя в одну кучу и вызывая рекурсию, затем во вторую и вызывая рекурсивный алгоритм.

## Схема решения



## Исходный код

// подключение стандартной библиотеки  
#include <iostream>  
  
// использование пространства имен std  
using namespace std;  
  
//объявление рекурсивной подпрограммы  
void divide (int sum1, int sum2, int iterator, int arr[]);  
// объявление переменных для хранения результата и количества элементов введенного

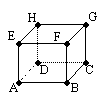
// массива соответственно  
int ans = 100001;  
int n;  
  
int main() {  
 //считывание кол-ва эл-тов массива  
 cin >> n;  
 //объявление массива на n эл-тов  
 int w[n];  
 //запускаем цикл от 0 до n для считывания элементов массива  
 for (int i = 0; i < n; i ++) {  
 cin >> w[i];  
 }  
 //вызов рекурсивной подпрограммы  
 divide(0, 0, 0, w);  
   
 //вывод результата работы программы  
 cout << ans << endl;  
  
 return 0;  
}  
  
//подпрограмма, принимающая на вход суммы двух куч, счетчик - номер операции, массив, // с которым ей предстоит работать  
void divide (int sum1, int sum2, int iterator, int arr[]) {  
   
 //проверка, дошли ли до последнего уровня рекурсии  
 if (iterator == n) {  
 //если дошли, то ответом станет минимум из ответа (в котором хранится минимум из текущих операций) и разности сумм 2-х куч  
 ans = min(ans, abs(sum1 - sum2));  
 }  
 else {  
 //иначе вызываем функцию рекурсивно, добавляя камень то в одну кучу, то в другую и увеличивая счетчик  
 divide(sum1 + arr[iterator], sum2, iterator + 1, arr);  
 divide(sum1, sum2 + arr[iterator], iterator + 1, arr);  
 }  
  
}

## **1155. Дуоны**

## Постановка задачи:

*Архангел по науке докладывает:  
— Господи, эти физики там, внизу, — они открыли ещё одну элементарную частицу!  
— Хорошо, добавим параметр в Общее Уравнение Вселенной.*

С развитием техники физики находят всё новые и новые элементарные частицы, с непонятными и даже загадочными свойствами. Многие слышали про мюоны, глюоны, странные кварки и прочую нечисть. Недавно были обнаружены элементарные частицы дуоны. Эти частицы названы так потому, что учёным удаётся создавать или аннигилировать их только парами. Кстати, от дуонов одни неприятности, поэтому от них стараются избавляться до начала экспериментов. Помогите физикам избавиться от дуонов в их установке.



Экспериментальная установка состоит из восьми камер, которые расположены в вершинах куба. Камеры промаркированы латинскими буквами A, B, C, …, H. Технически возможно создать, или наоборот, аннигилировать, два дуона, находящихся в смежных камерах. Вам нужно автоматизировать процесс удаления дуонов из установки.

### Исходные данные

В единственной строке даны восемь целых чисел в пределах от 0 до 100, описывающих количество дуонов в камерах установки (сначала в камере A, потом в B, и т.д.).

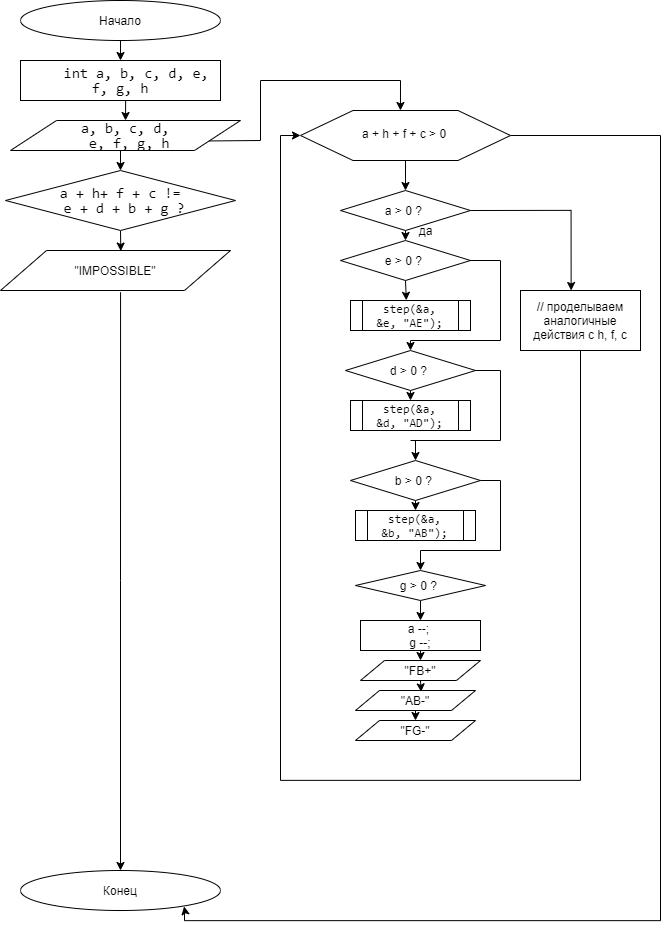
### Результат

Выведите последовательность действий для удаления всех дуонов или слово «IMPOSSIBLE», если это невозможно. Каждое действие должно быть описано в отдельной строке, в следующем формате: маркер первой камеры, маркер второй (смежной с первой), далее плюс либо минус (создать или аннигилировать пару дуонов). Количество действий в последовательности не должно превосходить 1000.

## Описание решения

Для начала определим, в какой ситуации удаление всех дуонов невозможно. Согласно условию задачи, можно создать или аннигилировать два дуона, находящихся в смежных камерах (вершины должны принадлежать одному ребру). Однако, существуют вершины, наиболее удаленные друг от друга, которые не имеют соединяющего их ребра. Разделим эти вершины на две группы (A, C, F, H) и (B, D, E, G). Заметим, что если сумма всех дуонов первой группы точек не равна сумме всех дуонов второй группы, то убрать все дуоны будет невозможно. В случае, если задача имеет решение, то сначала избавимся от дуонов на вершине А, аннигилируя все дуоны на смежных с ней вершинах. Если дуоны на вершине А не обнулились, а дуоны на смежных вершинах закончились, то смотрим на наиболее удаленную точку от А – G и находим 2 плоскости, в одной из которых будет лежать А, в другой G, добавляем по дуону на отрезок прямой, по которой эти плоскости пересекаются, а затем аннигилируем эти пары дуонов, потому что добавленный дуон и дуон в вершине окажется смежным (аналогичная ситуация с другим добавленным дуоном и вершиной G). После проделаем то же с вершинами C, F, H.

## Схема решения



## Исходный код

// подключение стандартной библиотеки  
#include <iostream>  
  
// использование пространства имен std  
using namespace std;  
  
//подпрограмма, которая удалит дуоны на двух ребрах и выведет сообщение о том, на каком ребре сработала  
void step(int\* x, int\* y, string msg) {  
 (\*x)--;  
 (\*y)--;  
 cout << msg << '-' << endl;  
};  
  
int main() {  
 // переменные, для хранения кол-ва дуонов в ребрах куба  
 int a, b, c, d, e, f, g, h;  
  
 // считывание кол-ва дуонов в ребрах куба  
 cin >> a >> b >> c >> d >> e >> f >> g >> h;  
  
 // если сумма дуонов на несмежных вершинах  
 // не совпадает - решения нет  
 if (a + h + f + c != e + d + b + g) {  
 //вывод соответствующего сообщения и выход из программы  
 cout << "IMPOSSIBLE" << endl;  
 return 0;  
 }  
  
 //пока не закончатся дуоны в одной из групп  
 while (a + h + f + c > 0) {  
 // если дуоны в вершине А не закончились, проверяем сначала 3 смежные с ней вершины,  
 // а затем наиболее удаленную от нее - G, добавляем ребро, у которого одна вершина   
 // смежная с А, а другая с G, затем удаляем эти отрезки.  
 if (a > 0) {  
 if (e > 0) step(&a, &e, "AE");  
 else if (d > 0) step(&a, &d, "AD");  
 else if (b > 0) step (&a, &b, "AB");  
 else if (g > 0) {  
 a--;  
 g--;  
 cout << "FB+" << endl;  
 cout << "AB-" << endl;  
 cout << "FG-" << endl;  
 }  
 }  
 //аналогичные действия с вершиной H  
 if (h > 0) {  
 if (e > 0) step(&h, &e, "HE");  
 else if (d > 0) step(&h, &d, "HD");  
 else if (g > 0) step(&h, &g, "HG");  
 else if (b > 0) {  
 h--;  
 b--;  
 cout << "DC+" << endl;  
 cout << "HD-" << endl;  
 cout << "BC-" << endl;  
 }  
 }  
 //аналогичные действия с вершиной F  
 if (f > 0) {  
 if (e > 0) step(&f, &e, "EF");  
 else if (b > 0) step(&f, &b, "BF");  
 else if (g > 0) step(&f, &g, "FG");  
 else if (d > 0) {  
 d--;  
 f--;  
 cout << "AB+" << endl;  
 cout << "AD-" << endl;  
 cout << "BF-" << endl;  
 }  
 }  
 //аналогичные действия с вершиной C  
 if (c > 0) {  
 if (b > 0) step(&b, &c, "BC");  
 else if (d > 0) step(&c, &d, "CD");  
 else if (g > 0) step(&c, &g, "CG");  
 else if (e > 0) {  
 c--;  
 e--;  
 cout << "FB+" << endl;  
 cout << "EF-" << endl;  
 cout << "BC-" << endl;  
 }  
 }  
 }  
 return 0;  
}

## **1296. Гиперпереход**

## Постановка задачи:

Гиперпереход, открытый ещё в начале XXI-го века, и сейчас остаётся основным способом перемещения на расстояния до сотен тысяч парсеков. Но совсем недавно физиками открыто новое явление. Оказывается, длительностью альфа-фазы перехода можно легко управлять. Корабль, находящийся в альфа-фазе перехода, накапливает гравитационный потенциал. Чем больше накопленный гравитационный потенциал корабля, тем меньше энергии потребуется ему на прыжок сквозь пространство. Ваша цель — написать программу, которая позволит кораблю за счёт выбора времени начала альфа-фазы и её длительности накопить максимальный гравитационный потенциал.

В самой грубой модели грави-интенсивность — это последовательность целых чисел *pi*. Будем считать, что если альфа-фаза началась в момент *i* и закончилась в момент *j*, то накопленный в течение альфа-фазы потенциал — это сумма всех чисел, стоящих в последовательности на местах от *i* до *j*.

### Исходные данные

В первой строке входа записано целое число *N* — длина последовательности, отвечающей за грави-интенсивность (0 ≤ *N* ≤ 60000). Далее идут *N* строк, в каждой записано целое число *pi* (−30000 ≤ *pi* ≤ 30000).

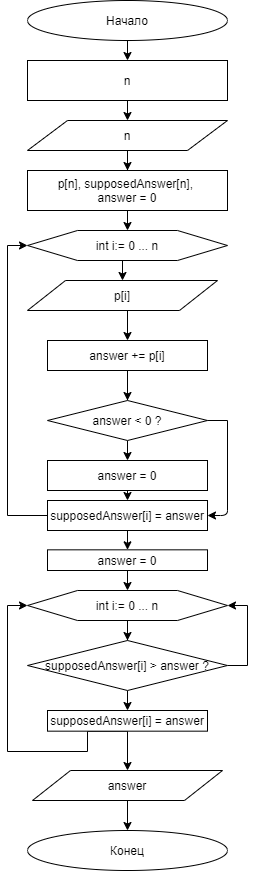
### Результат

Максимальный гравитационный потенциал, который может накопить корабль в альфа-фазе прыжка. Считается, что потенциал корабля в начальный момент времени равен нулю.

## Описание решения

Наша задача сводится к поиску подмассива с максимальной суммой. Заметим, что даже в худшем случае – когда все элементы массива отрицательны, сумма никогда не станет меньше 0. Начиная с первого элемента, прибавляем его к текущей сумме, на каждому шаге цикла записывая результат в отдельный массив (1). Если получившееся число больше 0, то в дальнейшем оно может нам только увеличить результат, все хорошо, продолжаем проход по массиву. Если текущая сумма оказывается меньше 0, то смысла в ней нет, поэтому сумму обнуляем и в (1) записываем 0 . Затем делаем проход по массиву (1) и отыскиваем там максимальный результат, он и будет результатом.

## Схема решения



## Исходный код

// подключение стандартной библиотеки  
#include <iostream>  
  
// использование пространства имен std  
using namespace std;  
  
int main() {  
  
 // создание переменной для хранения длины последовательности,  
 // отвечающей за грави-интенсивность  
 int n;  
 // считывание длины последовательности, отвечающей за грави-интенсивность  
 cin >> n;  
  
 // создание массива для хранения значений грави-интенсивности  
 int p[n];  
  
 // создание массива для хранения всех возможных ответов  
 int supposedAnswer[n];  
  
 //переменная для хранения результата работы программы  
 int answer = 0; //даже для худшего случая это так, меньше уже точно некуда  
  
 // запуск цикла на количество зн-й грави-интенсивности  
 for (int i = 0; i < n; i++) {  
 // считывание значений грави-интенсивности  
 cin >> p[i];  
 //добавляем к текущему значению считанное значение  
 answer += p[i];  
 // если получилось отрицательное число, то обнуляем переменную  
 if (answer < 0)  
 answer = 0;  
 // складываем все текущие значения переменной в массив  
 supposedAnswer[i] = answer;  
 }  
 // обнуляем переменную результата  
 answer = 0;  
 // запуск цикла на количество зн-й грави-интенсивности для   
 // поиска максимума в массиве всех возможных ответов, таким образом, в переменной   
 // ответа лежит максимальное значение массива, что от нас и требовалось  
 for (int i = 0; i < n; i ++) {  
 if (supposedAnswer[i] > answer)  
 answer = supposedAnswer[i];  
 }  
 // вывод результата работы программы  
 cout << answer << endl;  
 return 0;  
}

**1401. Игроки**

## Постановка задачи:

Известно, что господин Чичиков зарабатывал свой капитал и таким способом: он спорил со всякими недотёпами, что сможет доказать, что квадратную доску размера 512 × 512 нельзя замостить следующими фигурами:

|  |
| --- |
| X XX X XX  XX X XX X |

и всегда выигрывал. Однако один из недотёп оказался не так уж глуп, и сказал, что сможет замостить такими фигурами доску размера 512 × 512 без правой верхней клетки. Чичиков, не подумав, ляпнул, что он вообще может любую доску размера 2*n* × 2*n* без одной произвольной клетки замостить такими фигурами. Слово за слово, они поспорили. Чичиков чувствует, что сам он не докажет свою правоту. Помогите же ему!

### Исходные данные

В первой строке записано целое число *n* (1 ≤ *n* ≤ 9). Во второй строке через пробел даны два целых числа *x*, *y*: координаты «выколотой» клетки доски (1 ≤ *x*, *y* ≤ 2*n*), *x* — номер строки, *y* — номер столбца. Левый верхний угол доски имеет координаты (1, 1).

### Результат

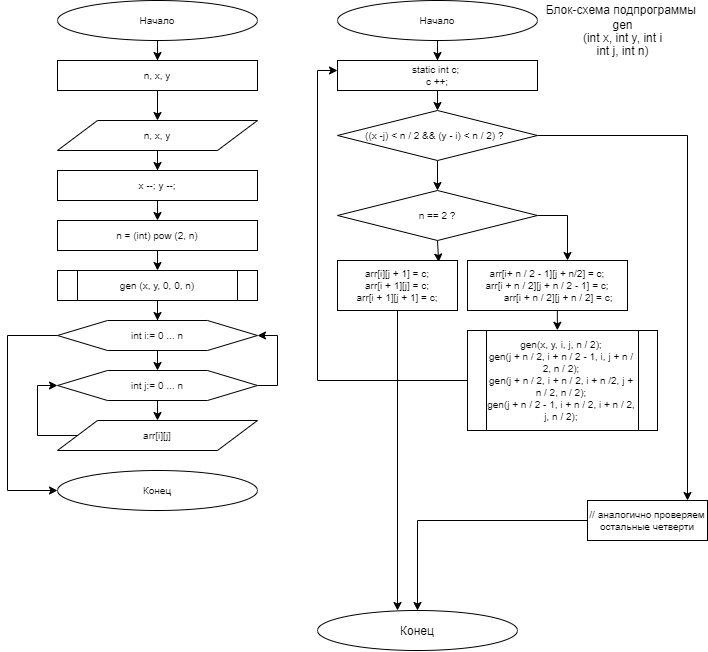
Ваша программа должна выдать 2*n* строчек по 2*n* чисел в каждой строке. На месте выбитой клетки должно стоять число 0. На месте остальных клеток должны стоять числа от 1 до (22*n* − 1) / 3 — номер фигуры, закрывающей данную клетку. Разумеется, одинаковые номера должны образовывать фигуры. Если же такую доску нельзя покрыть фигурами, выведите «−1».

## Описание решения

Заметим, что для квадрата 2 х 2 задача решена. Докажем, что мы можем решить данную задачу для любых квадратов размера 2n x 2n. Допустим мы умеем решать задачу для 2n x 2n, решим для 2n+1 x 2n+1.

Во-первых, разобьём наш квадрат на 4 квадрата размера 2n x 2n. Один из получившихся квадратов содержит в себе дырку и его мы можем разрезать согласно предположению. Докажем, что оставшиеся квадраты мы также можем разрезать, возьмём оставшийся треугольник из центра большого квадрата 2n+1 x 2n+1 отдельно. Теперь в оставшихся 3х маленьких квадратах, у нас ровно по одной вырезанной точке, а такую задачу мы решать умеем – далее решаем рекурсивно.

## Схема решения



## Исходный код

// подключение библиотек  
#include <iostream>  
#include <cmath>  
  
//массив, который нужно замостить  
int arr[512][512];  
  
// i, j - координаты начала квадрата  
// x, y - координаты дырки  
// n - координаты дырки  
void gen(int x, int y, int i, int j, int n) {  
 // счетчик для фигур  
 static int c;  
 c++;  
 //если дырка находится в 4 четверти  
 if ((x - j) < n / 2 && (y - i) < n / 2) {  
 // если мы пришли к задаче 2х2, следует сразу замостить и выйти  
 if (n == 2) {  
 arr[i][j + 1] = c;  
 arr[i + 1][j] = c;  
 arr[i + 1][j + 1] = c;  
 return;  
 }  
 // замащиваем фигурой так, чтобы в каждом получившемся  
 // квадрате меньшего размера была ровно одна "дырка"  
 arr[i + n / 2 - 1][j + n / 2] = c;  
 arr[i + n / 2][j + n / 2 - 1] = c;  
 arr[i + n / 2][j + n / 2] = c;  
 // вызываем функцию для получившихся квадратов, в каждом из которых

// по одной дырке  
 gen(x, y, i, j, n / 2);  
 gen(j + n / 2, i + n / 2 - 1, i, j + n / 2, n / 2);  
 gen(j + n / 2, i + n / 2, i + n / 2, j + n / 2, n / 2);  
 gen(j + n / 2 - 1, i + n / 2, i + n / 2, j, n / 2);  
  
 } //если дырка находится в 1 четверти  
 else if ((x - j) >= n / 2 && (y - i) < n / 2) {  
 // если мы пришли к задаче 2х2, следует сразу замостить и выйти  
 if (n == 2) {  
 arr[i][j] = c;  
 arr[i + 1][j] = c;  
 arr[i + 1][j + 1] = c;  
 return;  
 }  
 // замащиваем фигурой так, чтобы в каждом получившемся  
 // квадрате меньшего размера была ровно одна "дырка"  
 arr[i + n / 2 - 1][j + n / 2 - 1] = c;  
 arr[i + n / 2][j + n / 2 - 1] = c;  
 arr[i + n / 2][j + n / 2] = c;  
 // вызываем функцию для получившихся квадратов, в каждом из которых по

// одной дырке  
 gen(j + n / 2 - 1, i + n / 2 - 1, i, j, n / 2);  
 gen(x, y, i, j + n / 2, n / 2);  
 gen(j + n / 2, i + n / 2, i + n / 2, j + n / 2, n / 2);  
 gen(j + n / 2 - 1, i + n / 2, i + n / 2, j, n / 2);  
 } //если дырка находится в 2 четверти  
 else if ((x - j) >= n / 2 && (y - i) >= n / 2) {  
 // если мы пришли к задаче 2х2, следует сразу замостить и выйти  
 if (n == 2) {  
 arr[i][j] = c;  
 arr[i + 1][j] = c;  
 arr[i][j + 1] = c;  
 return;  
 }  
 // замащиваем фигурой так, чтобы в каждом получившемся  
 // квадрате меньшего размера была ровно одна "дырка"  
 arr[i + n / 2 - 1][j + n / 2 - 1] = c;  
 arr[i + n / 2][j + n / 2 - 1] = c;  
 arr[i + n / 2 - 1][j + n / 2] = c;  
 //вызываем функцию для получившихся квадратов, в каждом из которых по

// одной дырке  
 gen(j + n / 2 - 1, i + n / 2 - 1, i, j, n / 2);  
 gen(j + n / 2, i + n / 2 - 1, i, j + n / 2, n / 2);  
 gen(x, y, i + n / 2, j + n / 2, n / 2);  
 gen(j + n / 2 - 1, i + n / 2, i + n / 2, j, n / 2);  
 } //если дырка находится в 3 четверти  
 else if ((x - j) < n / 2 && (y - i) >= n / 2) {  
 // если мы пришли к задаче 2х2, следует сразу замостить и выйти  
 if (n == 2) {  
 arr[i][j] = c;  
 arr[i][j + 1] = c;  
 arr[i + 1][j + 1] = c;  
 return;  
 }  
 // замащиваем фигурой так, чтобы в каждом получившемся  
 // квадрате меньшего размера была ровно одна "дырка"  
 arr[i + n / 2 - 1][j + n / 2 - 1] = c;  
 arr[i + n / 2][j + n / 2] = c;  
 arr[i + n / 2 - 1][j + n / 2] = c;  
 // вызываем функцию для получившихся квадратов, в каждом из которых по

// одной дырке  
 gen(j + n / 2 - 1, i + n / 2 - 1, i, j, n / 2);  
 gen(j + n / 2, i + n / 2 - 1, i, j + n / 2, n / 2);  
 gen(j + n / 2, i + n / 2, i + n / 2, j + n / 2, n / 2);  
 gen(x, y, i + n / 2, j, n / 2);  
 }  
}  
  
int main() {  
 // объявление переменных для хранения характеристики  
 // размера поля, а также координат дырки  
 int n, x, y;  
 // считывание характеристики размера поля и координат дырки  
 std::cin >> n >> x >> y;  
 // приводим координаты к системе координат с началом в (0;0)  
 x--;  
 y--;  
 // вычисление размера поля  
 n = (int) pow(2, n);  
 // вызов функции для замощения  
 gen(x, y, 0, 0, n);  
 // вывод массива результатов  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 std::cout << arr[i][j] << " ";  
 }  
 std::cout << std::endl;  
 }  
 return 0;  
}