## Введение.

**Целями практикума являются:**

1. Использование ООП при проектировании архитектуры программы.
2. Анализ существующих алгоритмов.
3. Разработка своего алгоритма.
4. Оптимизация алгоритма.
5. Анализ сложности алгоритма.
6. Эксперимент и тестирование.
7. Общее представление о графических библотеках (на примере MFC).
8. Архитектура программ с графическим интерфейсом.

**Указания по планированию архтектуры системы:**

Результатом работы должны быть 4 главных класса:

1. Класс, хранящий данные и решающий задачу ПО ШАГАМ.
2. Класс консольного интерфейса пользователя, позволяющий решать задачу целиком или по шагам.
3. Тестирующий класс.
4. Класс графического интерфейса пользователя, позволяющий решать задачу целиком или по шагам.

Консольное приложение должно использовать классы 1,2,3, MFC-приложение – 1,3,4. Таким образом, классы 1 и 3 должны быть написаны таким образом, чтобы уметь взаимодействовать как с консольным, так и с графическим интерфейсом пользователя. Взаимодействие между классами осуществляется с помощью public-методов.

**Работа делится на этапы, каждый из которых заканчивается сдачей работающей программы:**

1. План архитектуры системы: определения 2-х основных классов (алгоритм решения задачи по шагам и интерфейс пользователя), большинство методов пока без реализации.
2. Консольное приложение без оптимизации: реализована 1-я версия решения задачи по шагам и консольный интерфейс пользователя, файловый ввод-вывод.
3. Тестирование: проверка корректности работы алгоритма на различных входных данных.
4. Оптимизация: оптимизированная версия решения задачи, измерение сложности каждого шага и подсчет общей сложности решения задачи.
5. Анализ сложности и эксперимент: набор тестов, позволяющих обнаружить зависимость общей сложности решения от параметров задачи. Результатом этого этапа являются 2 главы в документации: теоретические оценки сложности неоптимизированного и оптимизированного решения (формулы зависимости суммарной сложности решения от параметров задачи); таблица результатов – параметры задачи, физическое время оптимизированного и неоптимизированного решения, отношения физического времени к теоретическим оценкам (должна получиться примерно константа). Дополнительно нужна проверка совпадения неоптимизированного и оптимизированного решений – количество экспериментов, когда их ответы совпали (возможно, с некоторой точностью).
6. Графический интерфейс: MFC-приложение для визуализации хода решения по шагам.

## Условия задач.

### 1. *Упаковка предметов в контейнеры.*

Дано:

N предметов , каждому сопоставлен размер p(i) (вектор действительных чисел длины N.)

S ящиков, в которые можно класть предметы (все ящики имеют одинаковую вместимость V > p(i))

Числа N,S фиксированы.

Программа из входного файла должна получать вектор размеров предметов и значение вместительности ящиков.

Требуется реализовать алгоритм упаковки в предметов в ящики, минимизирующий количество использованных ящиков.

Вывести ответ в выходной файл.

### 2. *Задача о рюкзаке.*

По данному набору из n предметов стоимостями v1,v2,...,vn и весами w1,w2,...,wn (действительные числа, получаемые из входного файла) найти поднабор (с учетом того, что нельзя брать один предмет несколько раз) такой, что его стоимость будет максимальна, среди всех поднаборов веса не более W.

Вывести ответ в выходной файл.

### 3. *Пары чисел.*

Пусть нам дана пара чисел (a, b). Мы можем получить из нее новую пару прибавив одно из этих чисел к другому. В результате у нас будет либо пара (a+b, b), либо (a, a+b). Эта операция занимает одну секунду.

Пусть изначально у нас имеется пара (1, 1). Требуется реализовать алгоритм, позволяющий за минимальное время получить в качестве хотя бы одного из элементов пары число N.

### 4. *Покрытие таблицы из 0 и 1.*

Во входном файле задана матрица $A$ размера NxM из 0 и 1. Известно, что $A$ не содержит нулевых столбцов. Покрытием матрицы $A$ называется такое подмножество строк $C=\{i1,\ldots,ik\}\subset\{1,2,\ldots,m\}$, что для каждого $j=1,\ldots,n$ найдется $s=s(j)\in C$ для которого $asj=1$. Другими словами, надо выбрать строки так, чтобы в полученной подматрице не было нулевых столбцов. Требуется построить кратчайшее покрытие, т.е. покрытие, содержащее наименьшее число строк.

Вывести номера строк покрытия в выходной файл.

В качестве приближенного решения можно использовать градиентный алгоритм:

Инициализация: $A1=A$

k-й шаг:

· В матрице $Ak$ выбирается строка $ik$, содержащая наибольшее число единиц;

· Из матрицы $Ak$ вычеркивается эта строка и все столбцы, на пересечении которых со строкой $ik$стоят единицы; получается матрица $Ak+1$.

Алгоритм заканчивает работу, если матрица $Ak$ не содержит единиц.

### 5. *Раскраска планарного графа*

Реализовать алгоритм, раскрашивающий вершины любого планарного графа в 5 цветов так, что любое ребро соединяет вершины разного цвета.

Входные данные – число вершин планарного графа и список ребер.

Выходные данные – цвета вершин.

### 6. *Эйлеров граф*

Дан неэйлеров граф, как набор вершин (общее число - F) и ребер ( пар вершин ) в файле.

Дополнить его до Эйлерового графа и найти в нем эйлеров цикл. Оптимизировать структуру хранения графа для решения данной задачи.

Вывести ответ в выходной файл.

### 7. *Гамильтонов цикл в гиперкубе.*

В подмножестве ребер гиперкуба размерности N, построить гамильтонов цикл.

Входные данные – подмножество ребер гиперкуба.

Вывести ответ в выходной файл. Выходные данные – координаты вершин в порядке их обхода.

### 8. *Обход графа методом ближайшего соседа.*

Дан граф, как набор вершин (общее число - F) и ребер ( пар вершин ) в файле.

Используя алгоритм "ближайшего соседа" построить его обход по вершинам.

Оптимизировать структуру хранения графа для решения данной задачи.

Вывести ответ в выходной файл.

### 9. *Гамильтонов обход графа.*

Дан полный граф, для которого выполнено неравенство треугольника, как набор вершин (общее число - F) и ребер ( пар вершин ) в файле. Построить гамильтонов обход графа. Найти его вес. Оптимизировать структуру хранения графа для решения данной задачи.

Вывести ответ в выходной файл.

### 10. *Минимальное остовное дерево.*

Дан граф, как набор вершин (общее число - F) и ребер ( пар вершин ) в файле.

Построить минимальное остовное дерево графа. Оптимизировать структуру хранения графа для решения данной задачи.

Вывести ответ в выходной файл.

### 11. *Эволюция слов.*

В файле задано слово W длины M из алфавита {0,1,..,N}.N<M. также в файле задана функция F:{0,1,…,N}^M -> {0,1,…,N}^M, в виде слова (это слово - результат применения функции к набору 0,1,..N). Для произвольного j построить F^j(W). Проверить, нет ли в получившемся слове подряд идущих букв алфавита. Вывести ответ в выходной файл.

### 12. *"Крестики-нолики" на доске NxN.*

Задано число N>10. Реализовать игру "крестики-нолики". Предполагается, что пользователь играет крестиками  и может подавать на вход программы ходы с помощью

задания буквы и числа - координат крестика в матрице NxN. Выигравшим считается тот, кто первым построил ряд из 5 крестиков (или ноликов).

**Консольный интерфейс**: вывод на экран текущей позиции в виде

```
 |a|b|c|d|e|f|
-+-+-+-+-+-+-|
1| | | | | | |
-+-+-+-+-+-+-|
2| |X|O|O| | |
-+-+-+-+-+-+-|
3| | |X|X|O| |
-------------
```

, запрос с клавиатуры следующего хода человека.

**Оптимизация алгоритма**: функция оценки позиции, сокращение перебора таким образом, чтобы обеспечить оценку позиции на 3 хода вперед (2 хода программы и 1 ход человека).

**Усложнение задачи**: реализовать игру на бесконечной доске (структуру данных для хранения позиции, обеспечивающую быструю оценку позиции).

### *13. Задача о раскройке.*

Дан прямоугольник PxS (длины сторон - P и S задаются на входе программы пользователем). Построить рациональную раскройку прямоугольника на фиксированные одинакового размера многоугольники вида "Г"так, чтобы коэффициент

v=(Площадь остатка прямоугольника после вырезания)/(PxS) был минимальным из всех возможных случаев. Многоугольники вида "Г"задаются как 4 вещественных числа: размер описанного прямоугольника и размер вырезанной части.

Вывести ответ в выходной файл.

### *14. Триангуляция Делоне*

Триангуляцией некоторого набора точек на плоскости называется набор невырожденных треугольников, удовлетворяющий следующим свойствам:

1. Вершинами треугольников являются только точки исходного набора. Каждая точка исходного набора является вершиной хотя бы одного треугольника.
2. Два различных треугольника либо не имеют общих точек, либо имеют общую вершину, либо имеют общую сторону (но площадь их пересечения всегда равна 0).
3. Любая точка, лежащая внутри выпуклой оболочки исходного набора точек, принадлежит хотя бы одному треугольнику (она может принадлежать нескольким треугольникам, если является их общей вершиной или принадлежит их общей стороне). (Выпуклой оболочкой некоторого набора точек называется наименьший выпуклый многоугольник, содержащий все эти точки).

Триангуляция называется триангуляцией *Делоне*, если кроме того для нее выполняется следующее условие:

4. Внутри окружности, описанной около любого треугольника из триангуляции, не лежит ни одна из исходных точек (точки могут лежать на окружности, в частности на ней, очевидно, лежат вершины рассматриваемого треугольника)

Для заданного набора точек найдите количество его триангуляций Делоне (две триангуляции считаются различными, если они отличаются хотя бы одним треугольником)

***Входные данные***

На первой строке входного файла находится число $N$ – количество точек ($3<=N<=30$) исходного набора. Следующие $N$ строк содержат по одной паре вещественных чисел – координаты соответствующей точки. Никакие три точки не лежат на одной прямой.

***Выходные данные***

Выведите в выходной файл количество различных триангуляций Делоне указанного набора точек.

### *15. Нахождение подслов.*

Даны $n$+1 слово: A и слова $B_1,\ldots, B_n$ причем длина слова A намного превосходит длину слов $B_i$. Требуется найти все вхождения $B_i$, $i=1,\ldots,n$ в A. Программа должна считать из входного файла слова A и $B_1,\ldots, B_n$ и напечатать ответ в выходной файл.

### *16. Строка-Подстрока*

Дана строка $S$ длины $L$ в алфавите $A=\{a_1,\ldots,a_n\}$ . Найти количество строк длины $N$ в алфавите $A$, не содержащих $S$ в качестве подстроки.

Входные данные: $L$, $N$, $n$ – первая строка. Последующие $L$ строчек – номера символов, образующих строку $S$.

Выходные данные: количество строк, не содержащих $S$.

### *17. Быстрое умножение двоичных чисел.*

Даны два n-битовых числа x и y. Если n не степень двойки, то надо эти числа доопределить слева нулями. Требуется реализовать алгоритм быстрого умножения двоичных чисел.

Алгоритм быстрого умножения состоит в следующем:

1. представляем каждое число x и y в виде x = x1x2 и y = y1y2, где x1,x2,y1,y2 - n/2-битовые числа;
2. вычисляем x1y1, x2y2,(x1+x2)(y1+y2);
3. вычисляем xy по формуле:

xy = (x1*2^{n-1}+x2)(y1*2^{n-1}+y2)= x1y1*2^n+(x1y2+x2y1)*2^{n-1}+x2y2,
где x1y2+x2y1 = (x1+x2)(y1+y2)-x1y1-x2y2.

Программа должна считать из входного файла два n-битовым числа и напечатать в выходной файл результат их умножения и количество операций.

### *18. Быстрое умножение матриц.*

Даны две матрицы A и B размера nxn. Если n не степень двойки, то надо доопределить их нулями, например, начиная с верхнего левого угла. Требуется реализовать алгоритм быстрого умножения матриц.

Алгоритм быстрого умножения состоит в следующем:

1. представляем матрицы A и B в виде:

```
A = A11 A12
    A21 A22

B = B11 B12
    B21 B22
```

где Aij, Bij - матрицы размера (n/2)x(n/2);

2. вычисляем

```
p1 = (A11+A22)(B11+B22)
p2 = (A21+A22)B11
p3 = A11(B12-B22)
```

```
p4 = A22(B21-B11)
p5 = (A11+A12)B22
p6 = (A21-A11)(B11+B12)
p7 = (A12-A22)(B21+B22)
```

3. вычисляем AB по формуле

```
AB = C11 C12
     C21 C22
```

где

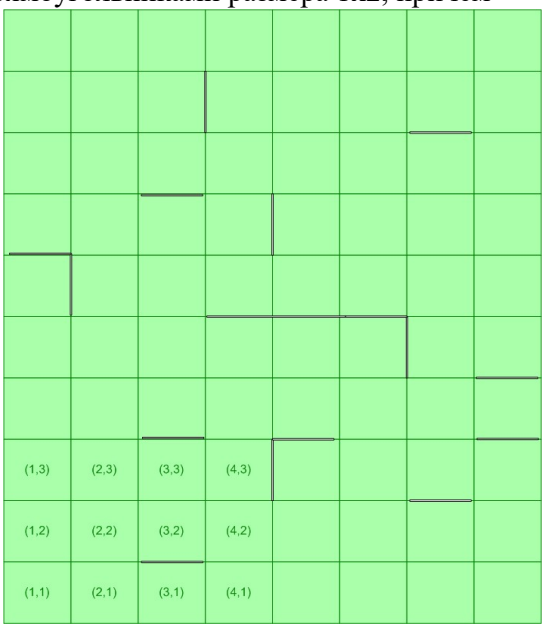
```
C11 = p1+p4-p5+p7,
C12 = p3+p5,
C21 = p2+p4,
C22 = p1+p3-p2+p6.
```

Программа должна считать из входного файла две матрицам размера nxn и напечатать в выходной файл результат их умножения и количество операций.

### *19. Домино.*

На доске размером m x n между некоторыми полями стоят перегородки.

Надо покрыть ее костяшками домино, т.е. прямоугольниками размера 1x2, причем нельзя класть их на перегородки. Если такое расположение невозможно, то программа должна выдавать соответствующее сообщение.




Входные данные: первая строка – размеры строки и количество перегородок. Каждая следующая строка – номера клеток, между которыми перегородка стоит. Клетки нумеруются так (x,y), где x – номер столбца, y – строки.

Выходные данные – пары клеток, покрытые одной костяшкой домино.

### *20. Игра «Угадай число»*

Первый игрок задумывает число от 1 до *N*. Второй может задавать вопросы вида «делится ли задуманное число на …». Надо отгадать задуманное число за наименьшее число вопросов.

Программа имитирует действия второго игрока и вычисляет делимость для ответов на вопросы.

Входные данные – *N* и задуманное число *n*

Выходные данные список вопросов, которые задает 2 игрок.

### *21. Реализация алгоритма Хаффмана построения кода с минимальной избыточностью.*

См. https://en.wikipedia.org/wiki/Huffman_coding

**Усложнение задачи**: задать несколько распределений вероятностей – для разных областей и минимизировать среднюю длину для наихудшего случая.

### *22. Бал*

Расположить N точек на сторонах $M$-угольника $A_1A_2\ldots A_M$, так, чтобы на любой стороне было одинаковое количество точек. Если точка расположена в вершине угла, то она считается принадлежащей обеим сторонам, образующим этот угол. Вдоль стороны может размещаться любое количество точек, а в углу не больше одной.

Напишите программу, находящую требуемое расположение точек.

**Входные данные**

Во входном файле содержатся два натуральных числа $M$ и $N$ ($3 \le M \le 1000$, $1 \le N \le 10$).

**Выходные данные**

В выходной файл для каждого угла требуется вывести число точек, стоящих в этом углу (0 или 1), а для каждой стены — количество точек, стоящих вдоль нее (не считая тех, что стоят в углах). Таким образом, в файле должно быть $2M$ чисел в соответствии со следующим порядком: сторона $A_MA_1$, угол $A_1$, сторона $A_1A_2$, угол $A_2$, …, сторона $A_{M-1}A_M$, угол $A_M$.

Числа в файле разделяются пробелами и/или переводами строки. В случае, если требуемым образом разместить точки невозможно, выходной файл должен содержать единственную строку NO.

### *23. Проверка однозначности декодирования.*

Схема кодирования. Каждой букве ai из A = {a1,...,ar} ставится в соответствие некоторое слово Bi из алфавита B = {b1,...,bq}. Для каждого кода Bi рассмотрим все его нетривиальные разложения Bi = PBj_1...Bj_kS, где P и S – некоторые слова.

Обозначим через V множество, содержащее пустое слово Z и слова X, встречающиеся в нетривиальном разложениях как в виде начал, так и в виде окончаний. Построим помеченный ориентированный граф G по следующим правилам. Множеством вершин графа G является V. Проводим дугу из вершины X1 в вершину X2, если и только если в некотором нетривиальном разложении X1 является началом, а X2 - концом. При этом дуга (X1,X2) помечается словом Bj_1...Bj_k.

Схема кодирования не обладает свойством однозначности декодирования тогда и только тогда, когда граф G содержит контур, проходящий через вершину Z.

Программа должна считать из входного файла кодовые слова Bi и напечатать в выходной файл либо 0, если граф G содержит контур, проходящий через вершину Z (схема не обладает свойством однозначности декодирования), либо 1 в противном случае (схема обладает свойством однозначности декодирования).

### *24. Задача о куче камней.*

Задано множество положительных чисел (весов) wi, i=1,...,N, и l. Разбиваем множество M = {i | i=1,...,N} на k непересекающихся подмножеств Mi, i=1,..,k, где k<N. Вес W(Mi) подмножества Mi определяется как сумма весов чисел, принадлежащих множеству Mi.

Вводится ограничение max(W(Mi))<=L

Требуется найти разбиение, для которого k минимально.

Программа должна считать из входного файла сначала число L, а затем последовательно числа w1,...,wN, и напечатать в выходной файл для каждого i=1,...,k множество {wj|j из Mi}, начиная каждое множество с новой строки.

### *25. Нахождение ориентированного цикла.*

Дан ориентированный граф. Требуется построить в нем ориентированный цикл наибольшей длины, если он существует.

Программа должна считать из входного файла таблицу смежности графа и напечатать в выходной файл либо последовательность вершин, образующих цикл, либо -1, если ориентированных циклов нет.

### *26. Раскраска графа.*

Дан граф без петель и кратных ребер и натуральное число L. Требуется раскрасить вершины этого графа в L красок.

Программа должна считать из входного файла таблицу смежности графа и напечатать в выходной файл либо последовательность вершин, окрашенных в каждый из цветов, либо написать -1, если данный граф в L цветов не раскрашивается.

### *27. Нахождение кратчайшего пути.*

Дан граф без петель и кратных ребер и выделенная вершина s. Ребрам этого графа приписаны положительные числа. Требуется найти кратчайшие расстояния от данной вершины s до остальных вершин этого графа. Можно использовать любой алгоритм, например, алгоритм Дейкстры.

Программа должна считать из входного файла таблицу смежности графа, в которой вместо единиц стоят веса, приписанные ребрам, и напечатать в выходной файл вектор кратчайших расстояний от вершины s до остальных вершин графа, а также с новой строки вектора предшественников на кратчайших путях, т.е номера тех вершин, которые встречаются в этих путях.

### *28. Раскраска ребер графа.*

Раскрасить ребра графа в k цветов так, чтобы из каждой вершины выходили ребра как одного, так и другого цвета. Если же это невозможно, то программа должна выдавать соответствующее сообщение.

Входные данные: первая строчка – количество вершин N

Остальные строчки – пары вершин, которые соединены ребром. Вершины нумеруются от 1 до N.

Выходные данные – список ребер, окрашенных в первый цвет ( соответственно остальные покрашены во второй).

### 29. Хэш-функция

Для быстрого распознавания идентификаторов в современных трансляторах используются *хэш-функции*. Хэш-функция сопоставляет строке целое число из некоторого промежутка. Критерием оценки хэш-функции является количество *коллизий*, то есть ситуаций, когда различным встретившимся строкам сопоставляется одно и то же число.

Вы должны построить хэш-функцию для заданного множества слов русского языка и реализовать ее в программе, которая будет вычислять хэш-значения из интервала от 1 до 255 для всех слов из входного файла. Ваша функция будет оцениваться в зависимости от максимального количества слов, хэш-значения которых оказались одинаковыми: чем это количество меньше, тем лучше функция.

*Замечание: Запрещается использовать стандартные алгоритмы – типа SHA, MD5 и др.*

### *30. Триангуляция .*

Выпуклый $n$-угольник задан списком координат вершин в порядке их обхода против часовой стрелки $(n < 100)$.

Требуется выполнить триангуляцию этого многоугольника, то есть разбить его на треугольники, проведя *n*-2 непересекающиеся хорды, причем так, чтобы сумма длин проведенных хорд была минимальной.

Считая, что вершины нумеруются от 1 до *n*, вывести суммарную длину хорд и список хорд, задавая каждую хорду парой номеров вершин. Записав результаты в файл, ваша программа должна изобразить триангулированный многоугольник на графическом мониторе, подходящим образом его отмасштабировав. Обеспечьте возможность рассмотреть вывод вашей программы.

### *31. Распределение последовательностей букв в тексте.*

Во входном файле записан текст. Пусть V – множество всех непробельных символов этого текста. *Словом* текста будем называть всякую последовательность символов из алфавита V (сам текст, при этом, разбивается пробельными символами на слова). Пусть L - длина длиннейшего слова текста. В выходной файл последовательно для каждого натурального n не превосходящего L следует записать все слова длины n (в алфавите V), упорядоченные в порядке убывания частоты встречаемости в качестве подслова в каком-то слове текста.

### *32. Прибить листочки*

На прямоугольный стол выложены *N* одинаковых квадратных листков бумаги со сторонами параллельными краям стола. Необходимо прибить листки к столу гвоздями так, чтобы в каждый листик был вбит ровно один гвоздь, а каждым гвоздем прикреплялся к столу хотя бы один листок.

В файле исходных данных записано количество листков *N* и длина стороны одного листка *L*. Далее следуют описания *N* листков, заданных координатами левого нижнего угла. Оси координат пущены по краям стола, начало координат находится в левом нижнем углу стола. Координаты точек - пары неотрицательных целых чисел, не превосходящих 1000. Все числа в исходном файле разделяются пробелами и (или) символами перевода строки.

Если вбить гвозди требуемым в условии способом невозможно, то вывести в выходной файл строчку «вбить гвозди невозможно». Если искомый способ существует, то ваша программа должна вывести в выходной файл в произвольном порядке координаты точек, в которые следует забить гвозди. Гвозди нельзя заколачивать в границы листков бумаги. Координаты точек – вещественные числа. Числа в выходном файле могут разделяться пробелами или символами перевода строки.

### *33. Растяжение/сжатие растрового изображения.*

Дано чёрно-белое растровое изображение размера k0 x m0. Требуется корректно изменить размер растра до k1 x m1 (растянув/сжав изображение на нем): считаем, что пиксель результирующего изображения окрашен в чёрное, если по меньшей мере половина из пикселей-прообразов окрашены в чёрное.

Входной файл содержит матрицу, задающую исходное изображение (k0 строк, в каждой строке m0 чисел из множества {0,1}, разделённых пробелом; число 1, при этом, интерпретируется как чёрный цвет, а 0 - как белый) и два числа k1,m1 (в следующей строке).

В выходной файл следует записать матрицу, задающую результирующее изображение (в том же формате: k1 строк из m1 чисел).

### *34. Наибольшее общее подслово.*

Входной файл содержит два слова, разделённых пробелом. В выходной файл следует записать их общее подслово максимальной длины (в частности, ничего не записать, если единственное общее подслово - пустое слово).

### *35. Задача о растекании жидкости.*

Имеется клетчатая область размера n x k. В начальный момент в каждой клетке имеется некоторое количество <<жидкости>>. Процесс <<растекания>> жидкости происходит в дискретном времени в соответствии с правилом: для каждой ячейки Я области

1. если ячейка Я - правая нижняя ячейка области, то жидкость, находившаяся в ней, остаётся на месте;
2. иначе если ячейка Я - крайняя правая ячейка области, то вся находившаяся в ней жидкость перетекает в соседнюю ячейку снизу;
3. иначе если ячейка Я - крайняя нижняя ячейка области, то вся находившаяся в ней жидкость перетекает в соседнюю ячейку справа;
4. иначе P частей жидкости из ячейки перетекает в соседнюю ячейку справа, а оставшиеся (1-P) частей - в соседнюю ячейку снизу.

Требуется смоделировать этот процесс.

Входной файл содержит матрицу, задающую исходное распределение жидкости (n строк, в каждой строке l чисел, разделённых пробелом) и число P. В выходной файл следует последовательно записать все состояния системы, возникающие в ходе процесса растекания жидкости, начиная с начального распределения вплоть до состояния стабилизации (когда вся жидкость перетечёт в правый нижний угол).

### *36. Гири*

Имеются гири с массами: 1 г, 2 г, ..., *N* г (*N*<=500000). Написать программу, распределяющую эти гири на максимально возможное количество пар так, чтобы суммарный вес гирь в каждой паре выражался простым числом.

### *37. Выпуклая оболочка.*

Дано конечное множество M={(x1,y1),…,(xn,yn)} точек на плоскости. Требуется построить их выпуклую оболочку. При $n >= 3$ можно воспользоваться, например, алгоритмом Грехема:

1. Пусть p0 - точка с наименьшей ординатой среди точек из M, а p1,…,pn - остальные точки множества M, отсортированные в порядке возрастания полярного угла относительно p0.
2. Заносим в стек точки p0,p1,p2.
3. Последовательно перебирая точки p3,…,pn, в зависимости от направления движения к текущей точке (левый или правый поворот) либо добавляем в стек текущую вершину, либо вначале выталкиваем из стека одну вершину, а затем добавляем текущую.

Входной файл состоит из множества строк вида <xk,yk>, разделённых пробелами. В выходной файл следует вывести координаты точек выпуклой оболочки, начиная с p0 в порядке обхода против часовой стрелки.

### *38. Компоненты связности ориентированного графа.*

Дан ориентированный граф (без кратных рёбер). Множество M вершин графа называется *компонентой связности*, если для любых v1,v2 из M существует ориентированный путь в графе из v1 в v2, проходящий через вершины из множества M. Компонента связности M называется максимальной, если для любой вершины v графа,

не лежащей в M, множество M, объединенное с v, не является компонентой связности. Требуется найти все максимальные компоненты связности графа.

Матрица инцидентности графа, содержащего n вершин, - это матрица размера nxn, в клетке (i,j) которой стоит 1, если граф содержит ориентированное ребро из i-ой вершины в j-ую, и 0 в противном случае.

Входной файл содержит матрицу инцидентности графа (n строк, в каждой строке n чисел из множества {0,1}, разделённых пробелом).В выходной файл следует вывести матрицы инцидентности максимальных компонент связности (рассматриваемых как граф с n вершинами, некоторые из которых, возможно, не соединены ребром).

### *39. Счастливые билетики.*

Счастливым называется билетик, в котором сумма первых n цифр совпадает с суммой следующих n. Рассмотреть задачу в d-ичной системе счисления.

Входные данные: n, d.

Выходные данные: количество счастливых билетиков.

### *40. Мультиколлизии.*

Случайная величина $X$ равномерно распределена от 1, до $N$. Производится n независимо распределенных испытаний. Найти вероятность того, что $X$ принимает какое-то значение не менее $k$ раз.

Т.е. мы бросаем n мячиков в N корзин. Какова вероятность, что хотя бы в одну корзину попадет не менее k мячиков.

### *41. Прямоугольники.*

На плоскости задано N прямоугольников. Найти точки, покрытые не менее чем k из них.

Входные данные – координаты прямоугольников ( левый нижний и правый верхний угол) и k.

Выход – координаты прямоугольников, образованные точками, покрытыми не менее чем k прямоугольниками.

### *42. Распределение компьютеров*

В новом учебном году на занятия в компьютерные классы пришли учащиеся, которые были разбиты на $N$ групп. В $i$-й группе оказалось $X_i$ человек. Тут же перед директором встала серьезная проблема: как распределить группы по аудиториям. В институте имеется $M \geq N$ аудиторий, в $j$-й аудитории имеется $Y_j$ компьютеров. Для занятий необходимо, чтобы у каждого учащегося был компьютер и еще один компьютер был у преподавателя. Переносить компьютеры из одной аудитории в другую запрещается.

Напишите программу, которая найдет, какое максимальное количество групп удастся одновременно распределить по аудиториям, чтобы всем учащимся в каждой группе хватило компьютеров, и при этом остался бы еще хотя бы один для учителя.

### *43. Игра 1.*

Игра начинается с числа N. На первом ходе разрешается уменьшить его на любое целое число от $1$ до $K$, на каждом последующем – на число, которое превосходит предыдущее не более чем на 1. Например, если N=10, K=5, то первый игрок может может уменьшить на 1, 2, 3, 4 или 5, если он уменьшил на 3, то на следующем ходу второй игрок может уменьшить на 1, 2, 3 или 4, и если он выберет 1, то первый затем может взять 1 или 2, и т. д. Проигрывает тот, кто получит 0.

Написать программу, которая играет с человеком и выигрывает, если это возможно.

### *44. Игра 2.*

Игра начинается с числа 1. Игроки по-очереди умножают число на любое число от 2 до k. Выигрывает тот, кто первым получит число не меньшее n.

Входные данные n,k.

Написать программу, которая играет с человеком и выигрывает, если это возможно.

### *45. Микросхема*

Для эффективной аппаратной реализации криптографической схемы RSA была разработана микросхема, позволяющая вычислять степени очень больших чисел. Память этой микросхемы стековая, то есть она организована по принципу «первым вошел – последним вышел». Элементарные операции, исполняемые микросхемой, следующие:

ЗАП X: записать число X на вершину стека; время исполнения 1 такт

УМН: перемножить два самых верхних числа в стеке и записать произведение на их место; время исполнения 1 такт

СТП n: возвести число, находящееся на вершине стека, в степень n и записать результат на его место; время исполнения n-1 такт

Напишите программу, которая бы для данного *n* выдавала последовательность инструкций, вычисляющую *x*^*n* (*n*-ю степень числа *x*), при помощи микросхемы за наименьшее количество тактов. После исполнения этих инструкций на вершине стека должно находиться значение *x*^*n*.

Например, для *n*=15 возможным способом вычисления является следующий.

**ЗАП X (содержимое стека X, время исполнения 1 такт)**
**СТП 5 (содержимое стека X^5, время исполнения 4 такта)**
**СТП 3 (содержимое стека X^15, время исполнения 2 такта)**

### *46. Ближайший вектор*

Задан граф с K вершинами и отметками на ребрах – целыми числами от 1 до K. В базе данных хранится N векторов длины M, компоненты которых – номера вершин графа - числа от 1 до K. Значения разных компонент вектора могут совпадать. Запрос – такой же вектор длины M. Требуется найти в базе данных вектор, ближайший к запросу. Расстояние между двумя векторами $(a_1,..,a_M)$ и $(b_1,..,b_M)$ вычисляется как сумма от 1 до M расстояний $d(a_i,b_i)$ между вершинами графа с номерами $a_i$ и $b_i$.

### *47. Нахождение слов с заданным свойством.*

Дан алфавит, состоящий из букв a1,...,an. На буквах этого алфавита задана метрика R(ai,aj) = |i-j|. Эту метрику можно распространить на конечные слова в этом алфавите следующим образом: пусть есть два слова A = ai_1…ai_l и B = aj_1…aj_l, тогда R(A,B)= min_{k=1,…,l}R(ai_k,aj_k).

Пусть дано слово A = ai_1…ai_s, требуется найти все такие слова B, чтобы R(A,B) = m, где m не превосходит n.

Программа должна считать входного из файла буквы алфавита, с новой строки слово A и, тоже с новой строки, натуральное число m, не превосходящее количества букв в алфавите. После этого она должна напечатать в выходной файл список слов с заданным свойством.

### *48. Доска Гальтона.*

Доска Гальтона - это вертикально расположенная доска, разделённая на N горизонтальных уровней. На каждом уровне в доску вбито несколько гвоздей (на самом верхнем один по центру, на каждом следующем уровне вбито на один гвоздь больше, чем на предыдущем). В середину верхнего кровня помещается шар, который, падая вниз, ударяется о гвоздь верхнего уровня и с вероятностью p отскакивает вправо, а с вероятностью (1-p) - влево, на следующем уровне шар вновь ударяется о гвоздь и с теми же вероятностями отскакивает вправо или влево.

Входной файл содержит число N горизонтальных уровней, вероятность p и количество M экспериментов, которые необходимо провести. В выходной файл следует записать частоту распределения шаров на нижнем уровне (по результате M экспериментов).