**Тема 1. Разработка web-приложений на языке Python**

***Задание 1*. Разработать web-приложение на языке Python шифрующее и дешифрующее сообщение посредством алгоритма RSA.**

Требования:

1) Приложение должно бы выполнено в клиент-серверной архитектуре с применением фреймворка Flask.

2) Интерфейс должен иметь форму ввода параметров для формирования закрытого и открытого ключа. После отправки формы приложению показывать страницу с сформированными ключами.

3) Интерфейс должен иметь форму ввода открытого ключа, сообщения для шифрования. После отправки формы приложению показать страницу с зашифрованным текстом.

4) Интерфейс должен иметь форму ввода закрытого ключа, зашифрованного сообщения. После отправки формы приложению показать страницу с расшифрованным текстом.

5)**\*** В приложении необходимо предусмотреть распараллеливание процессов шифрования и дешифрования с помощью потоков и отдельно — с помощью мультипроцессинга. Сравнить скорость работы в разных вариантах.

Описание алгоритма RSA:

1. Формируем закрытый и открытый ключи:

Выбираем простые числа *p* и *q*

Вычисляем *n = p \* q*

Вычисляем *φ(n) = (p - 1) \* (q - 1)*

Выбираем простое число e из диапазона [*3, φ(n) - 1*]*,* взаимно простое с *φ(n)*

Вычисляем *d ≡ e-1 (mod φ(n)) (это такое число d, для которого d\*e ≡ 1 (mod φ(n)), т. е. остаток от деления d\*e на φ(n) равен 1)*

Пара *(e, n)* — открытый ключ, *d —* закрытый ключ.

Шифрование:

Пусть целое число *m < n - 1* (***m*** — сообщение для зашифровки), тогда шифрование заключается в вычислении (с помощью открытого ключа *(e, n)*)  
***с = me mod n***

***c* — сообщение в зашифрованном виде.**

Для расшифровки сообщения необходимо вычислить (с помощью закрытого ключа *d* и части открытого ключа *n*):

***m = cd mod n***

2. Пример:

Пусть ***p*** = 3, ***q*** = 7

Вычисляем ***n*** = p\*q = 21

Вычисляем ***φ(n)*** *= (p - 1) \* (q - 1) = 12*

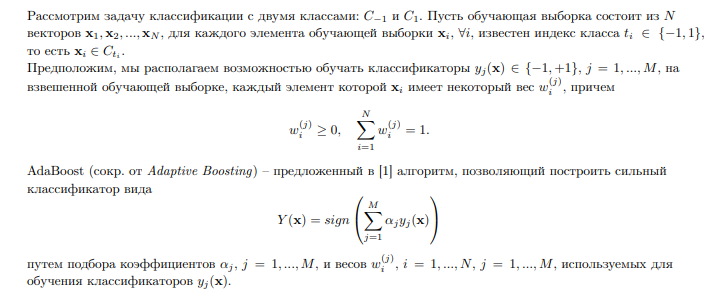
Выбираем простое число ***e*** из диапазона [*3, 11*] , взаимно простое с числом 12, вариантов здесь несколько: 5, 7, 11. Пусть будет ***e*** *= 5*.

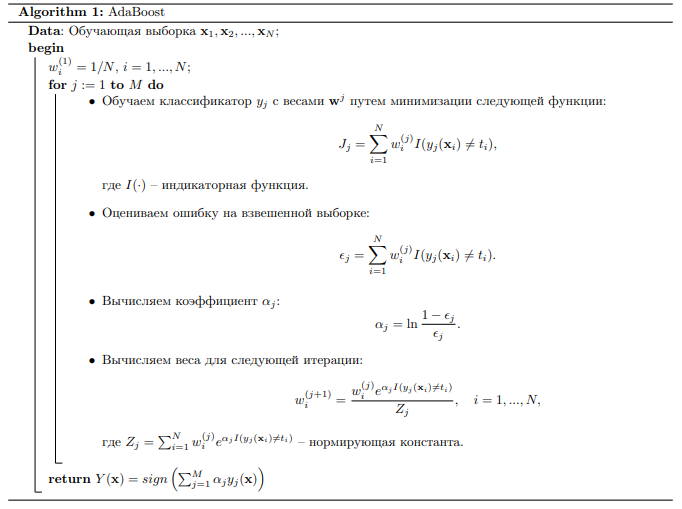
Теперь **открытый ключ** — пара (e,n) = **(5, 21)** , а закрытый ключ

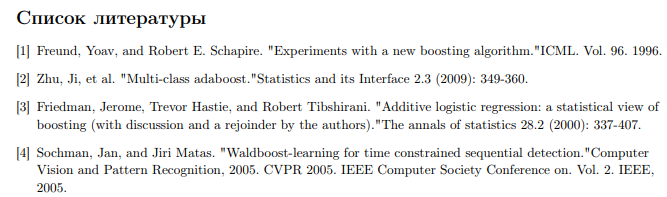
*d = e-1 mod φ(n)* = *5-1 mod 12* = 5

**Тема 2: «Параллельные вычисления».**

Рассмотрим алгоритм AdaBoost для бинарной классификации.







***Задание 2.* Создайте программу, реализующую элементы метода AdaBoost, с использованием распараллеливания вычислений. На данном этапе вместо методов обучения используйте «заглушки». Предусмотрите методы для вычисления ошибок слабых классификаторов, пересчёта весов объектов выборки.**

*Необходимые уточнения формулировки задания.*

Обозначим через v – номер Вашего варианта. В Вашем распоряжении (v mod 5 +5) потоков. Распределите оптимальным образом обучение 10v слабых классификаторов на потоках. Предполагается, что Вы можете оценить продолжительности обучений и на данном этапе задайте их произвольным образом. Составьте сетевой график реализации AdaBoost. Вычислите временные параметры графика. Оптимизируйте график, по возможности уменьшив длину критического пути. Результат запрограммируйте.

*Справочный материал по моделям сетевого планирования и управления в учебнике*

Основные элементы: события и работы.

Термин **работа** используется в широком смысле.

Во-первых, это действительная работа — протяженный во времени процесс, требующий затрат ресурсов.

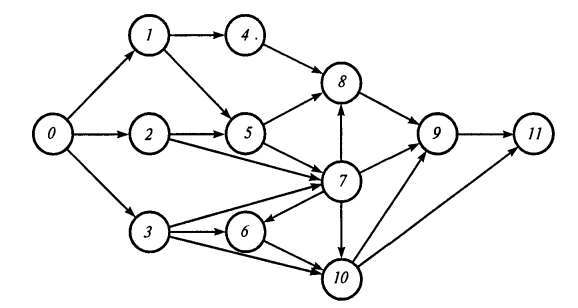
Во-вторых, это ожидание — протяженный во времени процесс, не требующий затрат труда.

В-третьих, это зависимость, или фиктивная работа — логическая связь между двумя или несколькими работами (событиями), не требующими затрат труда, материальных ресурсов или времени. Она указывает, что возможность одной работы непосредственно зависит от результатов другой.

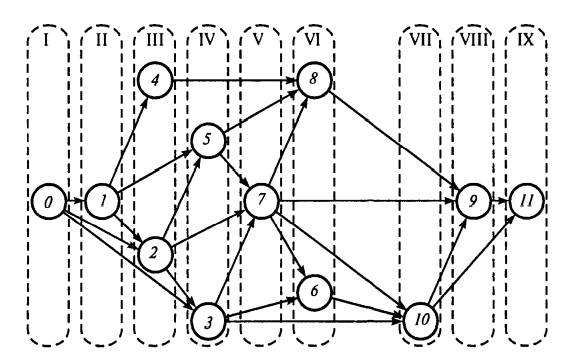
**Событие** — это момент начала или завершения какого-либо процесса, отражающий отдельный этап выполнения проекта. Предполагается, что событие не имеет продолжительности и свершается как бы мгновенно.

Пример 1. Предположим, что при составлении некоторого проекта выделено 12 событий: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 и 24 связывающие их работы: (0, 1), (0, 2), (0, 3), (1, 2), (1, 4), (1, 5), (2, 3), (2, 5), (2, 7),

(3, 6), (3, 7), (3, 10), (4, 8), (5, 8), (5, 7), (6, 10), (7, 6), (7, 8), (7, 9), (7, 10), (8, 9), (9, II), (10, 9), (10, II). Составим и сетевой график:



Упорядоченный вариант:



Путь — любая последовательность работ, в которой конечное событие каждой работы совпадает с начальным событием следующей за ней работы.

Среди различных путей сетевого графика наибольший интерес представляет полный путь L — любой путь, начало которого совпадает с исходным событием сети, а конец — с завершающим. Наиболее продолжительный полный путь в сетевом графике называется критическим. Критическими называются также работы и события, расположенные на этом пути.

Критический путь имеет особое значение, так как работы этого пути определяют общий цикл завершения всего комплекса работ, планируемых при помощи сетевого графика. И для сокращения продолжительности проекта необходимо в первую очередь сокращать продолжительность работ, лежащих на критическом пути.

Резерв времени события показывает, на какой допустимый период времени можно задержать наступление этого события, не вызывая при этом увеличения срока выполнения комплекса работ. Критические события резервов времени не имеют, так как любая задержка в свершении события, лежащего на критическом пути, вызовет такую же задержку в свершении завершающего события. Из этого следует, что для того чтобы определить длину и топологию критического пути, вовсе не обязательно перебирать все полные пути сетевого графика и определять их длины. Определив ранний срок наступления завершающего события сети, мы тем самым определяем длину критического пути, а выявив события с нулевыми резервами времени, определяем его топологию

Временные параметры сетевых графиков.

Ранний срок наступления события:



Поздний срок наступления события:



Резерв времени:



Пример 2.

