1. **Предсказание поведения сервиса. *Весьма умеренно зверская) задача***

В файле predict.csv (https://drive.google.com/open?id=1hc9-ziEgJs5q_H9LvnIp1obRGsCEV7iW) содержится набор метрик, характеризующих работу сервиса, обслуживающих запросы клиентов.

Описание колонок в файле:

timestamp - календарное время снятия показаний работы сервиса в формате time_t.

conns - текущее количество соединений клиентов.

count - полное количество обработанных запросов клиентов с момента старта сервиса.

read_bytes_total - полное количество байт, прочитанных сервисом с момента старта.

latency_ms_sum - полное время обработки всех запросов клиентов с момента старта сервиса.

Ваша задача:

1. Построить одну или несколько моделей, предсказывающих следующее значение целевой метрики latency_ms_sum в зависимости от предыдущих значений всех остальных метрик.
2. Построить на одном графике реальное значение метрики latency_ms_sum и предсказанное.
3. Построить график ошибки предсказаний в зависимости от календарного времени.
4. Найти все интервалы времени аномального поведения сервиса и отобразить их на графике метрики latency_ms_sum в зависимости от календарного времени.

2. **Оценка среднего значения и дисперсии для времени обработки запросов сервисом. *Не зверская) задача***

В файле ms_bucket.csv (https://drive.google.com/open?id=1LZvOz6iMKjzng9jl7EbNhcsS-E7c2LUc) содержатся данные по распределению времени обработки запросов сервисом.

Описание колонок в файле:

timestamp - календарное время снятия показаний работы сервиса в формате time_t.

Inf - Суммарное количество обработанных запросов сервисом с момента старта.

1 - Суммарное количество обработанных запросов, каждый из которых занял меньше чем 1 миллисекунду.

2 - Суммарное количество обработанных запросов, каждый из которых занял меньше чем 2 миллисекунды.

...

90000 - Суммарное количество обработанных запросов, каждый из которых занял меньше чем 90 секунд.

Ваша задача:

1. Рассчитать среднее значение и дисперсию для времени обработки запросов сервисом в зависимости от календарного времени.
2. Отобразить среднее значение времени обработки запроса и дисперсию на графиках в зависимости от календарного времени.

3. Классификация вредоносного поведения. *Не зверская) задача*

В файле behavior.csv (https://drive.google.com/open?id=1fJ5VNX_6xY5yI7g1FkqrVNcFRzLecDy9) содержится статистика, описывающая поведение обычных и вредоносных процессов.

Описание колонок в файле:

tag - классификатор, имеет значение 0, если процесс обычный, и 1, если процесс вредоносный.

Имена остальных колонок (их более 800!) совпадают с именами изменяемых ключей реестра, файлов, загружаемых библиотек и системных вызовов.
В строках для этих колонок содержится количество действий (изменений/загрузок/вызовов).

Ваша задача:

1. Оценить важность различных признаков, оставить наиболее информативные.
2. Реализовать одну или несколько моделей, классифицирующих вредоносное поведение. Оценить точность модели.

4. Поиск секретной информации. *Не зверская) задача*

Имеется файловое хранилище с секретной информацией. В хранилище хранятся миллионы файлов.

Ваша задача:

Реализовать на любом удобном вам языке программирования систему поиска в произвольном файле последовательности байт длиной более 4 Кбайт, совпадающей с любой последовательностью байт в хранилище секретных файлов. Поиск должен быть очень быстрым.

5. NBD-сервер. *Весьма умеренно зверская) задача*

Реализуйте NBD-сервер, раздающий одно блочное устройство. Содержимое блочного устройства должно храниться в файле, который передаётся как аргумент командной строки.

Описание протокола NBD доступно здесь: https://github.com/NetworkBlockDevice/nbd/blob/master/doc/proto.md.

Достаточно реализовать следующие части протокола:

- handshake с поддержкой опций NBD_OPT_STRUCTURED_REPLY и NBD_OPT_GO,
- запросы NBD_CMD_READ и NBD_CMD_DISC.

Для проверки программы создайте файл с образом какой-либо ФС, подключите его с помощью nbdclient в другой Linux-машине и смонтируйте там этот образ.

Улучшения:

- Протокол NBD поддерживает отправку ответов вне очереди. Реализуйте исполнение IO в пуле потоков. Как проверить, что мы действительно получаем ускорение от переупорядочивания ответов? Объясните механизм увеличения производительности.
- Замените пул потоков на асинхронный ввод-вывод с помощью io_uring.
- Поддержите structured replies. NBD-клиент в Linux не поддерживает это расширение, им пользуется qemu.

Оцениваться будет не только решение, но и стиль кода, его архитектура, а также то, как и насколько онпротестирован.

Допустимые языки решения: C, Rust, Erlang, Go.

6. Archronis: Архиватор. *Умеренно зверская) задача*

Реализуйте программу сжатия данных (архиватор). В зависимости от уровня сложности задачи, Вы можете применять различные алгоритмы для сжатия (см., напр., Ватолин Д. и др., “Методы сжатия данных” - http://www.compression.ru/book/pdf/compression_methods_full_scanned.pdf), а также разные стратегии компоновки нескольких файлов в архив:

1. Программа читает из одного или нескольких входных файлов и пишет в один или несколько выходных файлов, имена файлов задаются из командной строки. При этом каждый входной файл архивируется независимо и порождает один выходной файл. Подумайте о валидности: что будет, если архив испорчен, как это эффективно определять?
2. (Предпочтительный) Программа читает одного или нескольких входных файлов, имена файлов задаются из командной строки, при этом выходной

файл один - архив, содержащий все входные файлы в сжатом виде. Надо будет подумать о заголовке архивного файла. О валидности (см. выше) тоже надо подумать. Задача может занять больше времени, чем вы думаете, спланируйте Ваши временные ресурсы.

Примитивные методы кодирования (RLE-алгоритм и близкие к нему) не будут считаться достаточными для решения. Как ориентир можно взять алгоритмы группы LZ, чуть попроще - алгоритм Хаффмана, статический или динамический (оптимальный или субоптимальный).

Усиленный вариант этой задачи - архиватор с клиент-серверной архитектурой.

Оцениваться будет не только решение, но и стиль кода, его архитектура.

Язык программирования - C++. Все контейнерные структуры данных надо будет реализовать самостоятельно, без использования STL.

7. AEMU: Бинарный транслятор. *Весьма зверская) задача, но с умеренно зверским) вариантом*

Реализуйте бинарный транслятор с байт-кода какого-либо софт-процессора (предпочтительно -, Lua без поддержки таблиц, или, как вариант, байт-кода самостоятельно разработанного софт-процессора) с компиляцией в машинный код семейства процессоров Intel x86.

Варианты выходных данных:

1. (Предпочтительный) Исполняемый файл для ОС Linux (ELF), Windows (MZ) или MacOS.
2. Исполняемый файл с машинным кодом семейства x86, исполняемый с помощью написанного Вами загрузчика.

Оцениваться будет не только решение, но и стиль кода, его архитектура.

Язык программирования - C++. Все контейнерные структуры данных надо будет реализовать самостоятельно, без использования STL.

8. Acram Alpha: Символьный дифференциатор. *Умеренно зверская) задача*

Реализуйте программу, принимающую формулу в традиционной математической нотации, с учетом естественного приоритета операций и без лишних скобок, и выдающую ее полную производную в виде формулы в PDF-файле (с припиской “Утрем нос Стивену Вольфраму!”). Должна поддерживаться большая часть элементарных функций и умеренные упрощения выражения (умножение на 0 и 1, сложение с 0 и т.п.).

Оцениваться будет не только решение, но и стиль кода, его архитектура.

Язык программирования - C++. Все контейнерные структуры данных надо будет реализовать самостоятельно, без использования STL.

9. Golang -- бесконечные циклы в горутинах. *Немного зверская) задача*

В golang код исполняется в горутине (goroutine), -- легковесном аналоге потока, который полностью реализуется в userspace. Одна программа на Go может запускать много горутин, чтобы выполнять много задач параллельно.

До версии 1.14 golang использовал кооперативную многозадачность при планировании исполнения горутин. Это означает, что переключение между горутинами происходит только в точках, где они делают вызовы go runtime. Для горутин, которые занимаются в основном вводом-выводом, это не проблема, так как они часто зовут функции, чтобы прочесть или записать что-либо в сокет или файл.

Если же горутина занята вычислительной задачей, она может неопределённо долго не вызывать никакие функции из go runtime, из-за чего другие горутины не смогут выполняться.

Простое “решение” этой проблемы -- это регулярно звать функцию runtime.Gosched().

Ваша задача -- получить на вход исходный файл программы на Go и автоматически расставить вызовы runtime.Gosched() перед всеми обратными рёбрами в CFG (control flow graph).

Стандартная библиотека Go экспортирует значительную часть компилятора golang. Используйте пакеты go/parser, go/ast и go/format для построения AST, его изменения и сохранения AST назад в текст.

Оцениваться будет не только решение, но и стиль кода, его архитектура, а также то, как и насколько он протестирован.

Вопрос для обсуждения на собеседовании: чем плохо решение расставить runtime.Gosched() перед всеми обратными рёбрами?

10. Сравнение реестров Windows разных компьютеров. *Внезапно халявная задача)*

Конфигурация в Windows хранится в реестре. Реестр представляет собой дерево, каждый узел которого имеет ноль или больше узлов-потомков, а также ноль или больше конфигурационных значений. Каждый узел и значение имеют имя. Для аналогии можно считать, что узлы – это каталоги в ФС, а значения -- файлы.

Напишите программу, которая покажет, как отличаются два реестра, взятые из разных экземпляров Windows: какие значение были добавлены, какие удалены, какие -- изменены. Поскольку реестр большой, распечатайте только отличающиеся части.

Пример вывода:

```
 ControlSet002
   Control
     Class
       {1ed2bbf9-11f0-4084-b21f-ad83a8e6dcdc}
       - 0000
       {4d36e972-e325-11ce-bfc1-08002be10318}
         0001
           Ndi
             Params
             - *LsoV1IPv4
             + *LsoV2IPv4
             + *LsoV2IPv6
```

Строчка “- *LsoV1IPv4” означает, что этот узел присутствует в первой версии реестра, но отсутствует во второй. Строчка “+ *LsoV2IpV4” означает, что узла нет в первой версии, но есть во второй. Строчка “Params” (родитель “*LsoV1IPv4” и “*LsoV2IPV4”) не имеет знаков “+” и “-” и означает, что узел “Params” есть в обоих версиях, но различается между ними, в данном случае -- из-за изменившегося списка узлов-потомков.

Пример показывает только добавленные и удалённые узлы. Придумайте, как наглядно изобразить изменения в значениях.

Для разбора реестра Windows используйте библиотеку hivex.

Допустимые языки решения: C, Rust, Go.

11. Непропорциональное уменьшение изображения. *Не зверская) задача*

Пусть имеется изображение размером *MxN* точек. Мы хотим уменьшить его до размера *M’ x N* точек, где *M’ < M*. При этом мы хотим не портить пропорции объектов на картинке, а удалить менее существенные её части.

Менее существенные части можно определять, например, как те, которые слабее всего отличаются от соседей. Например, на фотографии дома и пустого поля вокруг него вертикальная прямая, проходящая по полю далеко от дома, почти не отличается от другой вертикальной прямой, проведённой рядом. Удалить её будет предпочтительнее, чем деталь дома.

Определим вес точки как сумму квадратов расстояний от цвета этой точки до цветов соседей в RGB-координатах. Рассмотрим (связный) путь от нижней границы изображения до верхней (x_0, 0), (x_1, 1), (x_2, 2), … Связность пути означает, что $| x_{i+1} - x_i | <= 1$ для всех *i*. Весом пути назовём сумму весов точек в нём.

Для непропорционального уменьшения ширины картинки на единицу удалим путь наименьшего веса.

Реализуйте программу, которая путём удаления наименее существенных путей уменьшает ширину изображения. Используйте libpng или аналогичную библиотеку, чтобы загрузить исходное изображение и сохранить результат.

Как Вы предложите улучшить определение веса точки в изображении? Евклидово расстояние в RGB-координатах не имеет отношения к тому, как цвет воспринимается глазом: глаз более чувствителен в диапазоне волн, соответствующих “зеленым цветам”, цвета (0, 10, 0) и (0, 20, 0) отличаются заметнее, чем (0, 200, 0) и (0, 210, 0), и т.д.

Допустимые языки решения: C, Rust, Go.

12. Travelling salesman. *Единственная математическая задача)*

Вам дана карта, на которой отмечено N поселений. Поселения находятся на различном расстоянии друг от друга. Торговец отправляется из поселения N_0. Предложите алгоритм, который позволит найти оптимальный маршрут для обхода всех N поселений и вернуться в точку старта.

Ответ должен содержать один или несколько алгоритмов (можно псевдокод) и пояснение о эффективности данных решений.

Картинка для примера:

13. CrackMe. *Зверская) задача*

Вам дана программа crackme [http://tiny.cc/s304nz] (измените расширение файла на .exe), которая ожидает от вас логин и пароль. Патчить программу нельзя. Напишите keygen - консольное или оконное/графическое решение, которая принимает на вход логин (проверяет валидный ли логин) и возвращает минимум один пароль, подходящие для crackme. Оцениваться будет не только решение, но и стиль кода, его архитектура.

14. Mouse Filter Driver Sandbox. *Зверская) задача*

Необходимо написать фильтр драйвер для мыши под операционную систему Windows 7 x64. Драйвер должен после комбинации нажатий мыши LRRL инвертировать ось Y. Так как Вы, вероятно, не знакомы с кернельной разработкой, вот небольшой гайд:

1. Вам потребуется: VMWare Workstation Pro или VirtualBox, виртуальная машина с Windows 7 x64, Visual Studio (рекомендую 2013) установленная на хост-машине, WDK (рекомендую 8.1) установленная на хост-машине.
2. Необходимо настроить Shared Folder между хост-машиной и виртуалкой чтобы передавать бинарь драйвера после сборки в виртуалку.
3. Необходимо настроить kernel debug: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/attaching-to-a-virtual-machine--kernel-mode-
4. Возможно у вас не будут отображаться принты в windbg, вам поможет этот совет: http://www.osronline.com/article.cfm%5Earticle=295.htm
5. Пример драйвера мыши от Microsoft: https://github.com/microsoft/Windows-driver-samples/tree/master/input/moufiltr
6. Полезные материалы, которые могут помочь: https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/ , https://www.youtube.com/watch?v=T5VtaP-wtkk&list=PLZ4EgN7ZCzJyUT-FmgHsW4e9BxfP-VMuo

Оцениваться будет не только решение, но и стиль кода, архитектура, а также то, как глубоко Вы смогли самостоятельно разобраться в разработке драйверов.

15. Предсказание потребления ресурсов системой виртуализации. *Не зверская) задача*

Имеется система виртуализации, которая может запускать несколько гостевых операционных систем (ОС) на хосте, который характеризуется мощностью процессора (CPU), скоростью ввода/вывода дисковой подсистемы (I/O) и объемом памяти (RAM).

Все гостевые операционные системы характеризуются средним значением потребляемых ресурсов (CPU, I/O, RAM).

Система виртуализации ведет себя нелинейно - при увеличении количества запускаемых ОС и приближении к границам имеющихся ресурсов общая эффективность выполнения задач снижается. Поэтому важно предсказать, можно ли запустить еще одну ОС с заданным средним значением потребляемых ресурсов в данном гипервизоре при условии, что известны текущие значения потребляемых ресурсов всеми ОС, выполняемыми гипервизором.

На основе предыдущих наблюдения за работой гипервизора вы собрали 1000 измерений, которые показывают, как меняется загрузка гипервизора от текущей при запуске еще одной ОС с заданными значениями потребляемых ресурсов.

Ваша задача - реализовать систему, которая максимально точно предсказывает изменение потребления ресурсов гипервизором от текущих значений при запуске на нем еще одной ОС.
Текущее потребление ресурсов гипервизором и планируемое потребление ресурсов гостевой ОС может существенно отличаться он собранных ранее данных наблюдений.