# Архитектура алгоритма при добавлении ролика в базу данных

Перед тем, как добавить новый рекламный ролик в базу данных, его необходимо подготовить. Это делается для ускорения поиска сигнатур по базе данных.

Основная идея, состоит в том, что любой ролик можно разбить на части. Части будут строиться по принципу схожести, т.е. если в ролике подряд идут похожие кадры, они будут добавляться в одну и ту же часть, если порог, при сравнении сигнатур, больше заданного, то будем считать, что началась следующая часть. Порог в текущем исследовании будет носить рекомендательный характер и может быть изменен пользователем.

Далее будут описаны структуры хранения частей ролика, а также алгоритм их наполнения.

Основные понятия и константы:

* VideoPart – структура хранения части ролика.

* – порог сравнения первого и последнего кадра из VideoPart.

* – порог сжатия. Данный порог необходимо вычислить эмпирически, путем сравнения исходного видео и после различных популярных алгоритмов сжатия.

* – порог сцены. Данный порог настраивается в зависимости от категории видео. Так же данный порог подается в алгоритм поиска. Ориентировочные характеристики: , n = 2,3,4,5…
* mainSignature – сигнатура, являющаяся представителем ячейки VideoPart.

## Структура хранения части ролика: “VideoPart”

За хранение набора сигнатур, вычисленных из кадров подготавливаемого ролика, отвечает класс **SignatureHandler**. А именно в нем хранится динамический массив сигнатур. Так как поиск в массиве происходит за константное время O(1), в создаваемой структуре не будет хранится сигнатура, а лишь разметка. При этом у класса **SignatureHandler** будет константный метод **getSignature(size\_t index)**, который будет возвращать необходимую сигнатуру по индексу.

Далее будут описаны поля структуры хранения части ролика:

* Номер ключевого кадра – кадр подготавливаемого ролика, на котором произошел резкий скачок значения сравнения сигнатур соседних кадров (а именно разница оказалась выше задаваемого порога)(mainSignatureIndex).
* Номер последнего кадра, входящего в промежуток этой части(lastFrameIndex).
* Указатель на хранилище сигнатур, а именно класс **SignatureHandler**
* isMain – булева переменная. True, означает что данный videoPart построен на резком переходе видео. False, что текущий videoPart построен на основе ограничений , отличия первого и последнего кадров videoPart.

Описанного выше набора полей структуры достаточно для описания части ролика.

## Структура для хранения данных о поиске: “**SearchInfo**”

Так как необходимо работать с несколькими videoPart одновременно, данная структура содержит необходимую информацию, объединяя данную структуру в массив, появляется возможность работы с несколькими **videoPart** одновременно.

**SearchWindow** – структура используемая для поиска кадров в режиме “окно”. Для стандартных случаев, это +- 1 кадр (стандартная возможная ошибка пропуска кадров), в других случаях данный размер может быть произвольным.

Поля структуры **searchWindow**:

* **leftLimi**t – количество кадров влево от текущего
* **rightLimit** – количество кадров вправо от текущего

**VideoPart** – непосредственно videoPart с которым ведется работа.

**sigIndex** – текущий индекс в массиве сигнатур из **SignatureHandler**, на который указывает VideoPart.

isWorkWith – булевская переменна, отвечающая за то, что мы уже работали с этим блоком.

## Алгоритм сравнения кадра из потока видео с массивом VideoPart

Поток видео: stream,

Текущий кадр stream: streamFrame,

SearchInfoList – лист из SearchInfo;

While(!stream.isEmpty()){

Соответственно для начала работы StateMachine мы получаем кадр

streamFrame = stream.getNextFrame();

currentPartsVector,diffVector = dataBase.find(streamFrame); - возвращает массив videoPart, а так же массив, состоящий из уже вычисленных разниц сигнатуры streamFrame и mainSignature соответствующей videoPart.

if(!currentPartVector.isEmpty()){

for(auto part: currentPartVector, auto diff: diffVector){

currentSearchWindow = SearchInfoVector.find(part);

if(currentSearchWindow != SearchInfoVector.last()){

workWithSearchWindow(currentSearchWindow);

}else(если такого videoPart не найдено){

If(diff < ){

SearchInfoVector.pushback( SearchInfo(part, SearchWindow(1,1) ) )

currentIndex = mainSignature.index()

}else if (diff < ){

currentSearchWindow,index = checkAvailability(); - псевдо реализация ниже.

Данное условие говорит о том, что кадры похожи, но недостаточно, чтобы быть уверенным, что это искомый кадр. Мы предполагаем, что кадр может находится внутри ячейки VideoPart.

If(index >= 0){

SearchInfoVector.pushback( SearchInfo(part, currentSearchWindow,index) )

}

}

}

for(auto part: SearchInfoList){

}

}

workWithSearchWindow(){

currentSearchWindow.isWorkWith = true;

curDiff = getDiffOfSignatures (SignatureHandler[currentSearchWindow.sigIndex + 1],streamFrame.signature()]

if(curDiff < ){

считаем что данный кадр и есть искомый.

currentSearchWindow.sigIndex += 1;

continue; (продолжаем сравнение)

}else{

trulyIndex = checkFramesInWindow(); - в этой функции проверяются кадры вошедшие в окно, различие сигнатур с streamFrame окажется меньше , возвращаем его индекс, иначе текущий. Так же произвожится корректировка окна.

If(trulyIndex != currentIndex ){

errorVector.pushback(currentIndex);

currentIndex = trulyIndex;

}else{

Считаем что с текущим VideoPart работа закончена

}

}

if(currentSearchInfo.index == currentSearchInfo.videoPart.lastFrameIndex){

Данный блок означает, что кадры из текущего VideoPart закончились, мы ожидаем следующий VideoPart из текущего SignatureHandler.

Массив структур VideoPart, так же будет храниться в SignatureHandler, поэтому пытаемся получить следующий. Если не получилось, закрываем промежуток:

Иначе продолжаем работу.

currentSearchWindow.videoPart = signatureHandler.nextVideoPart();

}

}

defineSearchWindow(currentIndex, videoPart){

if(videoPart.isMain()){

searchWindow.leftLimit = 2;

searchWindow.rightLimit = 2;

}else{

rightSearch();

leftSearch();

//данные функции выполняют проход по массиву сигнатур( из videoPart.SignatureHangler) вправо и влево, пока порог streamFrame.signature и кадра не больше или не дошли до конца videoPart.

Про концы videoPart:

Предположим, мы попали в mainFrame (videoPart.isMain() = False), тогда идти дальше конца текущего videoPart и начала предыдущего, смысла нет, так как они построены так, что между первым и последним кадром разница .

Если мы попали не в mainframe, оном поиска становится:

searchWindow.leftLimit = getDiff(videoPart.mainFrame.index(),currentIndex)

searchWindow.rightLimit = getDiff(videoPart.lastFrame.index(),currentIndex)

}

}

checkFrameInWindow(){

checkSignatures() – идем слево – направо по окну, ищем кадры у которых порог меньше , среди них выбираем минимум

{

если такой кадр найден:

Оставляем окно текущим.

Если этот кадр следующий, за currentIndex (мы его и ожидали), то обрезаем окно до +- 2;

}

возвращаем этот кадр, иначе currentIndex

}

## Алгоритм разделения ролика на части

Далее будет описан алгоритм построения частей ролика:

Порог принадлежности: threshold. Предполагается  threshold.

Структура для хранения части ролика: newVideoPart.

Текущий индекс в массиве сигнатур: currentIndex = 0.

Массив сигнатур: signatures.

Массив частей ролика: videoPartHandler.

endVideoPart = false; булевая переменная, означающая что работа с текущим VideoPart окончена.

while(currentIndex != signatures.size()){

diff = getDiffOfSignatures(signatures[currentIndex], signatures[currentIndex + 1]);

If(diff <= threshold){

if(getDiffOfSignatures(signatures[currentIndex], newVideoPart.mainFrame) >= ){

endVideoPart = true;

}

}else if(diff == 1){

Данный случай необходимо исследовать отдельно, т.к. в курсовой работе [], был отмечен один из недостатков данного алгоритма, а именно: если после абсолютного черного кадра, идет кадр с слегка увеличенной яркостью, данный алгоритм показывает максимальное значение разницы кадров, что не является истинной.

If(checkIfBlack(signatures[currentIndex], signatures[currentIndex + 1]){

//блок обработки особого случая

}

}else{

endVideoPart = true;

}

if(endVideoPart){

newVideoPart.lastFrame = currentIndex;

videPartHandler.pushback(newVideoPart);

newVideoPart = VideoPart(параметры инициализации) ;

endVideoPart = false;

}

currentIndex ++;

}