## Описание проблемы

Для того, чтобы получить список совпадающих видеороликов в входящем видеопотоке реального времени, необходимо провести сравнение входящих сигнатур кадров с базой роликов. При работе с базой, состоящей из 10 000 роликов и частотой кадров 25 , простого сравнения сигнатур недостаточно. Характерное время сравнения 2 – ух сигнатур составляет 5 мкс. Соответственно за время равное: и средней длине ролика в 1 минуту, необходимо сравнить сигнатуру из входящего видеопотока с более чем сигнатур из базы роликов. Однако за время возможно сравнить только: сигнатур. Для того что бы решить эту проблему, необходимо сократить число сравнений.

В различных видео существуют последовательности похожих кадров. Они могут быть разной длинный и разной степени схожести. Однако, выделяя такие схожие последовательности кадров, можно объединить видеофрагмент в сегменты похожих кадров. Таким образом, если сравнивать сигнатуру входящего видеопотока не со всеми сигнатурами подряд, а только с ключевыми сигнатурами видео сегмента, можно достичь сокращение количества сравнений в 100 раз.

Сегменты формируется путем сравнения соседних сигнатур видео ролика, и при достижении различных порогов при сравнении, формирование сегмента считается оконченным. После разделения, сегменты содержат в себе похожие сигнатуры, таким образом среди них можно выделить ключевую сигнатуру, с которой и будет происходить сравнение во время поиска. В данной дипломной работе, ключевой сигнатурой является первая сигнатура в видео сегменте.

После получения сигнатуры кадра из входящего видеопотока, необходимо выполнить уже поиск не похожих сигнатур, а поиск похожих видео сегментов. Как наиболее оптимальным способом искать совпадающие видео сегменты выходит за рамки данной дипломной работы. Время, затраченное на эту процедуру, не будет учитываться. Далее набор похожих видео сегментов необходимо обработать. Т.е. отсеять ошибочно выбранные, расширить зону искомого кадра и продолжить работу с найденными ранее. Работа с входящем видеопоток формирует конечный автомат. Берется очередной кадр из входящей видеопоследовательности, считается сигнатура и подается на вход алгоритму поиска похожих видео сегментов. Алгоритм поиска выполняет сравнение с ключевыми сигнатурами видео сегментов используя поданных извне порог. Если результат сравнения ниже данного порога, этот видео сегмент помечается как похожий и является результатом поиска. Результатом работы алгоритма поиска, могут быть несколько похожих сегментов, следовательно далее необходимо используя более низкий порог чтобы отсеять неподходящие. Так же возможны случаи, когда искомая сигнатура находится внутри сегмента или сегмент, состоит из абсолютно одинаковых кадров (несколько секунд черного экрана). Что бы допустить минимум ошибок при работе с такими сегментами, необходимо использовать окно поиска. Окно поиска заключается в расширении зоны поиска кадра из входящего видеопотока, за счет нахождения сильно похожих кадров внутри сегмента видеоролика. Сохраняя найденные сегменты на каждой итерации конечного автомата, можно с высокой достоверностью отсеивать ошибочно выбранные, вести подсчет ошибок, например: пропуски кадров. После того как работа с видео сегментом закончена, например кадры в нем закончились, если он подходит под условия поиска, т.е. количество ошибок не превышает заданного, можно с высоким уровнем достоверности утверждать, что этот видео сегмент и есть искомый. Таким образом выполняется достоверный поиск совпавших видео сегментов в входящем видеопотоке реального времени.

# Архитектура алгоритма

# Введение

Далее будет описан алгоритм анализа бесконечного входного видеопотока (последовательность изображений) с целью поиска в нем любых частей из массива заранее подготовленных роликов. Для достижения поставленной цели, необходимо создать базу роликов, в которой будет производиться поиск. База хранит специально подготовленные сегменты видеороликов, благодаря которым ожидается уменьшение времени поиска видео сегмента. При подготовке каждый ролик разделяется на сегменты похожих друг на друга кадров. Например, когда в фильме начинается новая сцена, то первый кадр новой сцены сильно отличается от предыдущего кадра (последнего кадра предыдущей сцены). Далее в фильме могут идти изображения, похожие на первый кадр в сцене. Соответственно, при сравнении входного изображения с кадрами ролика можно ограничить сравнение только с первыми кадрами таких сегментов. Сегменты формируются из подготавливаемого видеоролика, путем разбиения ролика по определенным правилам и формируется сегмент видеоролика, в начале которого находится наиболее значимая сигнатура, с которой и происходит сравнение во время поиска. При этом сигнатуры сравниваются достаточно грубо (с большим порогом сравнения), чтобы отсеять подавляющее большинство ненужных роликов. А затем можно провести более точное сравнение с очень маленьким числом похожих видео сегментов.

Анализ входящего видеопотока сводится к разработке алгоритма работы с несколькими видео сегментами одновременно, а также способов достоверно подтвердить наличие того или иного видео сегмента из базы роликов в входящем видеопотоке. Это достигается за счет детектирования смены сцены в входящем видеопотоке, а также отсеивания неподходящих видео сегментов.

Так как в входящем видеопотоке возможны шумы по типу: воздействие алгоритмов сжатия, шумы вещания и т.п., требовать полного совпадения сигнатур невозможно. Для решения этой проблемы вводится эмпирический порог – порог сжатия. Если результат сравнение сигнатур ниже данного порога, они считаются идентичными. Данный порог является самым низким из представленных далее.

### Разбиение ролика на сегменты

Для подготовки ролика и разделения на сегменты, необходимо определять: резкую смену сцены и плавную смену сцены. Резкая смена сцены характеризуется большим значением отличия соседних сигнатур. Для определения резкой смены сцены вводится порог . Но кадры в роликах не всегда имеют характерные границы разделения по порогу . Возможны случаи, когда сцены сменяются плавно, т.е. отличие сигнатур соседних кадров ⋘ . Такой случай характеризует плавную смену сцены. При этом, если разделять видео по порогу , будут образовываться сегменты больших размеров, что негативно влияет на скорость поиска необходимого сегмента. Что бы корректно детектировать и избегать такие случаи, вводится порог  – порог сцены. Данный порог используется как ограничитель между первым и последним кадром в сегменте, т.е. если сравнение первой и последней сигнатуры больше порога , считается что данный сегмент сформирован. Порог настраивается пользователем, однако в данной дипломной работе будут даны рекомендации по настройке порога.

Для работы с сигнатурами, были введены 3 основных порога сравнения:

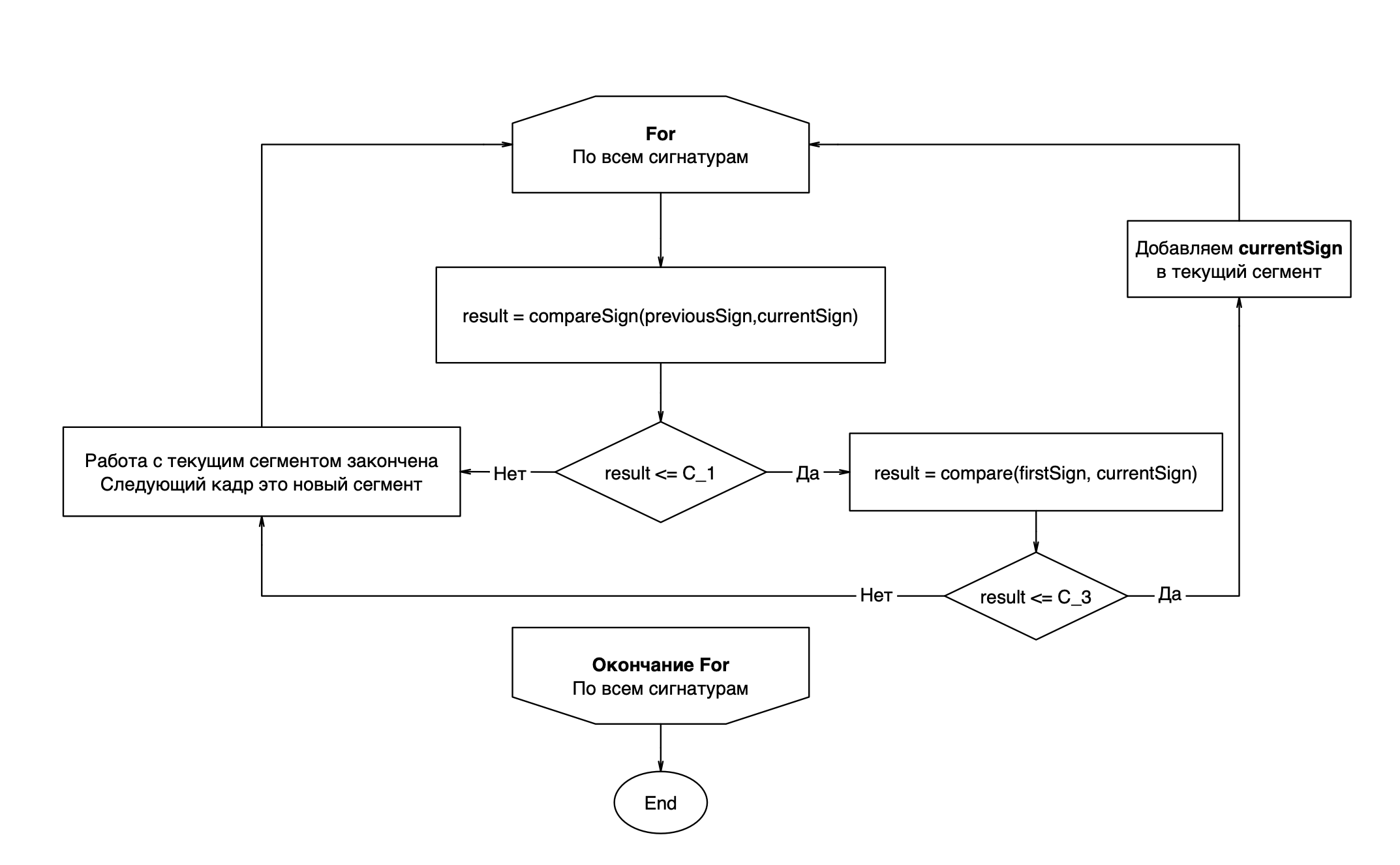
* – порог резкой смены сцены.

* – 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. Ориентировочные характеристики: , n = 2,3,4,5…

Далее представлена принципиальная схема работы алгоритма разбиения видеоролика на сегменты. На вход данный алгоритм принимает массив сигнатур ролика. **firstSign** – первая сигнатура в сегменте.

* **lastSign** – последняя сигнатура в сегменте.
* **previousSign** и **currentSign** – предыдущая и соответственно текущая сигнатура в массиве сигнатур.



Для хранения информации о сегменте, используется структура **VideoPart**.

Поля структуры хранения части ролика:

* **mainSignatureIndex** – индекс ключевой сигнатуры. Ключевая сигнатура – как правило первая сигнатура в **VideoPart**, которая используется при поиске по сегментам.
* **lastSignatureIndex** **-** индекс последнего кадра, входящего в промежуток текущего VideoPart.
* **selfIndex** – собственный индекс в массиве состоящем из **VideoPart**.

# Анализ входящего видеопотока

### Поиск очередного кадра по уже найденным сегментам.

После получения очередного кадра из входящего видеопотока, выполняется поиск по базе данных подготовленных видеороликов. Данный алгоритм производит поиск с порогом . Результатом поиска является массив **VideoPart**, которые подходят под условия поиска. Далее необходимо обработать как новые **VideoPart** (с которыми не производилась работа), так и **VideoPart**, с которыми производилась работа на предыдущем шаге.

Порог выбран в алгоритме поиска для того, чтобы корректно распознать ситуацию, когда искомый кадр находится внутри **VideoPart**. Ведь сравнение в алгоритме поиска производится только с первым кадром **VideoPart**. **VideoPart** устроен так, что в нем находятся похожие кадры, в связи с этим не всегда можно утверждать что найденный кадр и есть искомый, возможны ситуации когда таких кадров несколько, в связи с этим возникает необходимость создания “окна” поиска – **SearchWindow**. Данная структура, хранит данные о размере окна. Так же к отдельной категории относятся ситуации пропуска кадра в входящем видеопотоке, для этого, стандартным размером окна является +- 1 кадр, от кадра с которым ведется работа. В других случаях размер может быть произвольным.

Поля структуры **SearchWindow**:

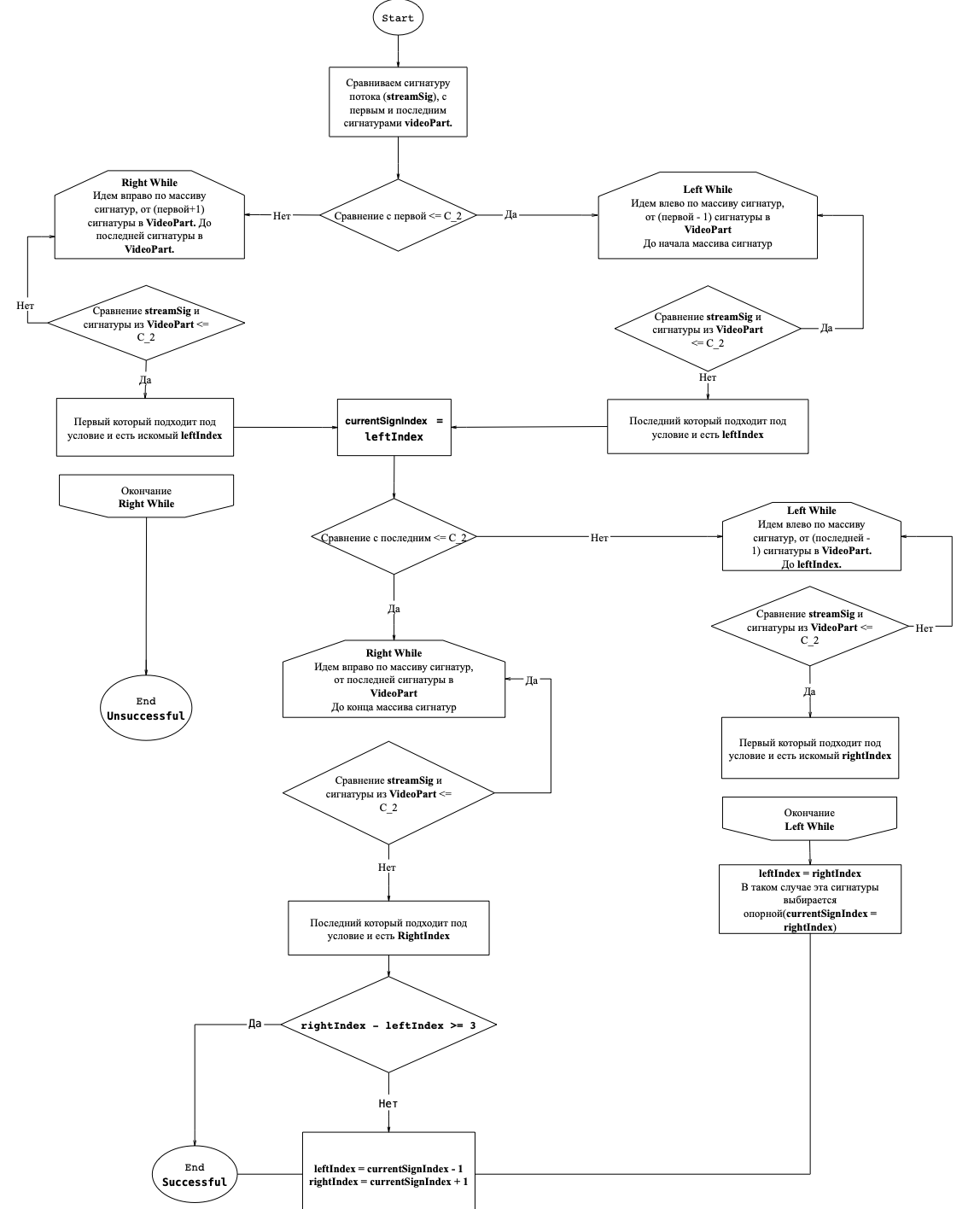
* **leftIndex** – абсолютный индекс левого края окна.
* **rightIndex** – абсолютный индекс правого края окна.

Алгоритм определения окна поиска выходит за рамки **VideoPart**. Такой прием используется для исключения ситуации, когда искомый кадр находится в соседнем **VideoPart**. Производится поиск сигнатур, результат сравнения с которыми будет ниже порога C\_2. Первый такой кадр выбирается опорным, а размеры окна поиска ограничиваются последней сигнатурой, сравнение с которой ниже порога С\_2.

Далее представленная схема алгоритма определения окна поиска **SearchWindow**.

Данная функция позволяет выбрать кадр, с которого начать работать с текущим **VideoPart**, а также определить окно поиска в рамках текущего **VideoPart.**

* **leftIndex** – абсолютный индекс сигнатуры в массиве сигнатур ролика. Обозначает левый край окна поиска.
* **rightIndex** - абсолютный индекс сигнатуры в массиве сигнатур ролика. Обозначает правый край окна поиска.
* **currentSignIndex** – индекс опорного кадра. Предполагается что именно этот кадр и есть искомый (или с него нужно начать поиск). Указывает на сигнатуру в массиве сигнатур.

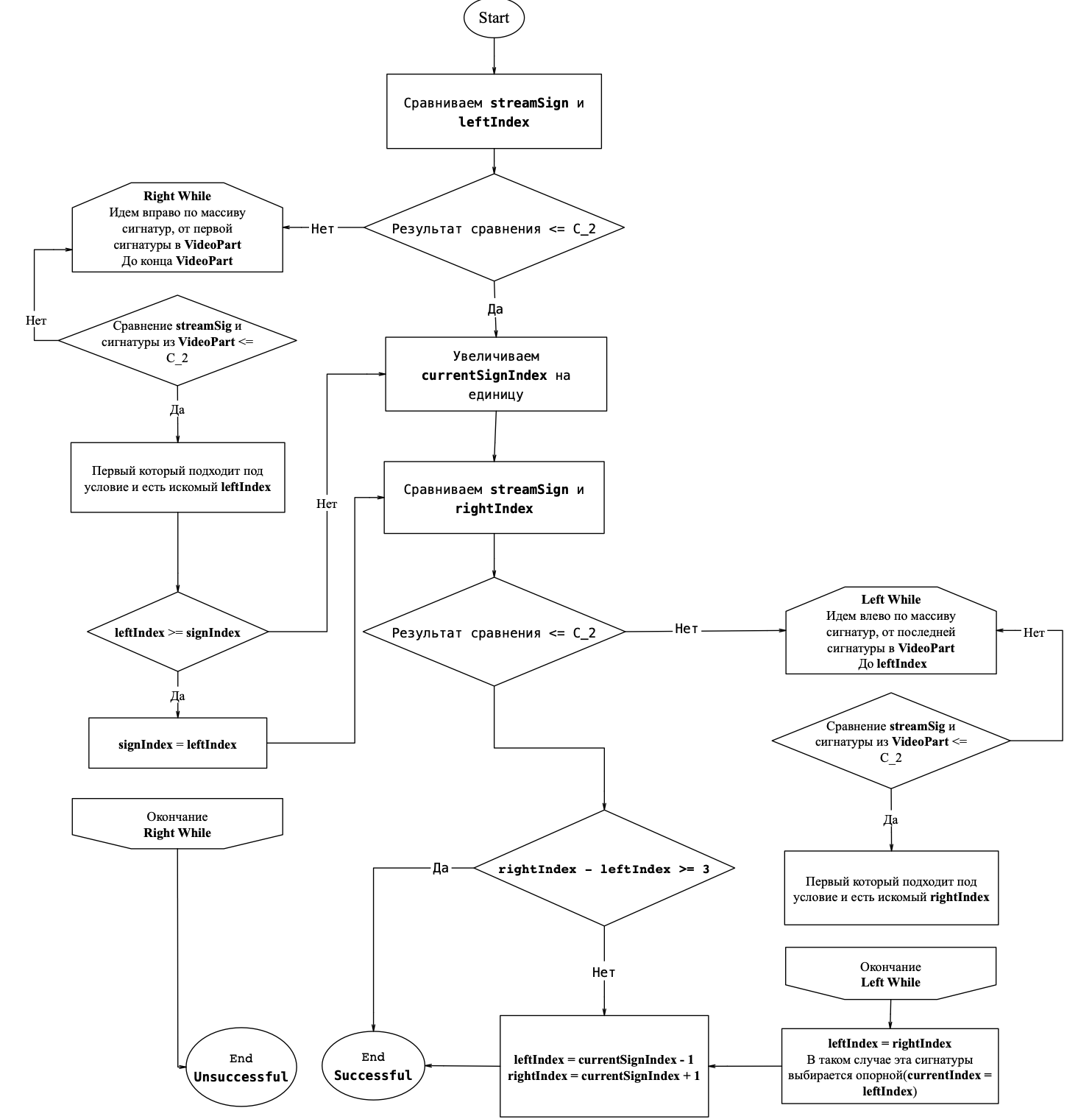


## Работа с сигнатурами внутри окна

На следующих шагах алгоритма поиска, необходимо обработать кадры, которые находятся в окне поиска, а также произвести корректировку окна поиска. Данный шаг выполняется в целях сужения окна поиска, а также определения корректности ранее найденной информации. Алгоритм выполняет поиск сигнатуры с левого края **SearchWindow** до правого, выполняя поиск кадра, результат сравнение сигнатуры которого с сигнатурой из входящего видеопотока будет < C\_2. Такая сигнатура становится опорной, а размеры окна корректируются. Если размер окна не изменились, индекс ожидаемой сигнатуры увеличивается на единицу. Для определения второй границы окна поиска выполняется поиск справа на лево по **SearchWindow**. Первый кадр, который подходит по условию и является второй границей **SearchWindow**.

Далее представлена схема алгоритма работы с окном поиска.

Данная функция выполняет корректировку окна (если это необходимо), а также меняет индекс опорного кадра.

* **streamSig** – сигнатура из видеопотока,
* **leftIndex** - левый край окна в массиве сигнатур ролика,
* **rightIndex** - правый край окна в массиве сигнатур ролика,
* **currentSignIndex** - индекс ожидаемого кадра в массиве сигнатур ролика.

Алгоритм поиска похожих частей по сигнатуре входящего видео с порогом C\_3, может возвращать несколько таких частей, так как каждую из них необходимо обработать, так же продолжать работать на нескольких итерациях сравнения, возникает потребность работы с несколькими **VideoPart** одновременно. Структура **SearchInfo** созданная решить эту проблему, объединяя в себе всю необходимую информацию для работы.

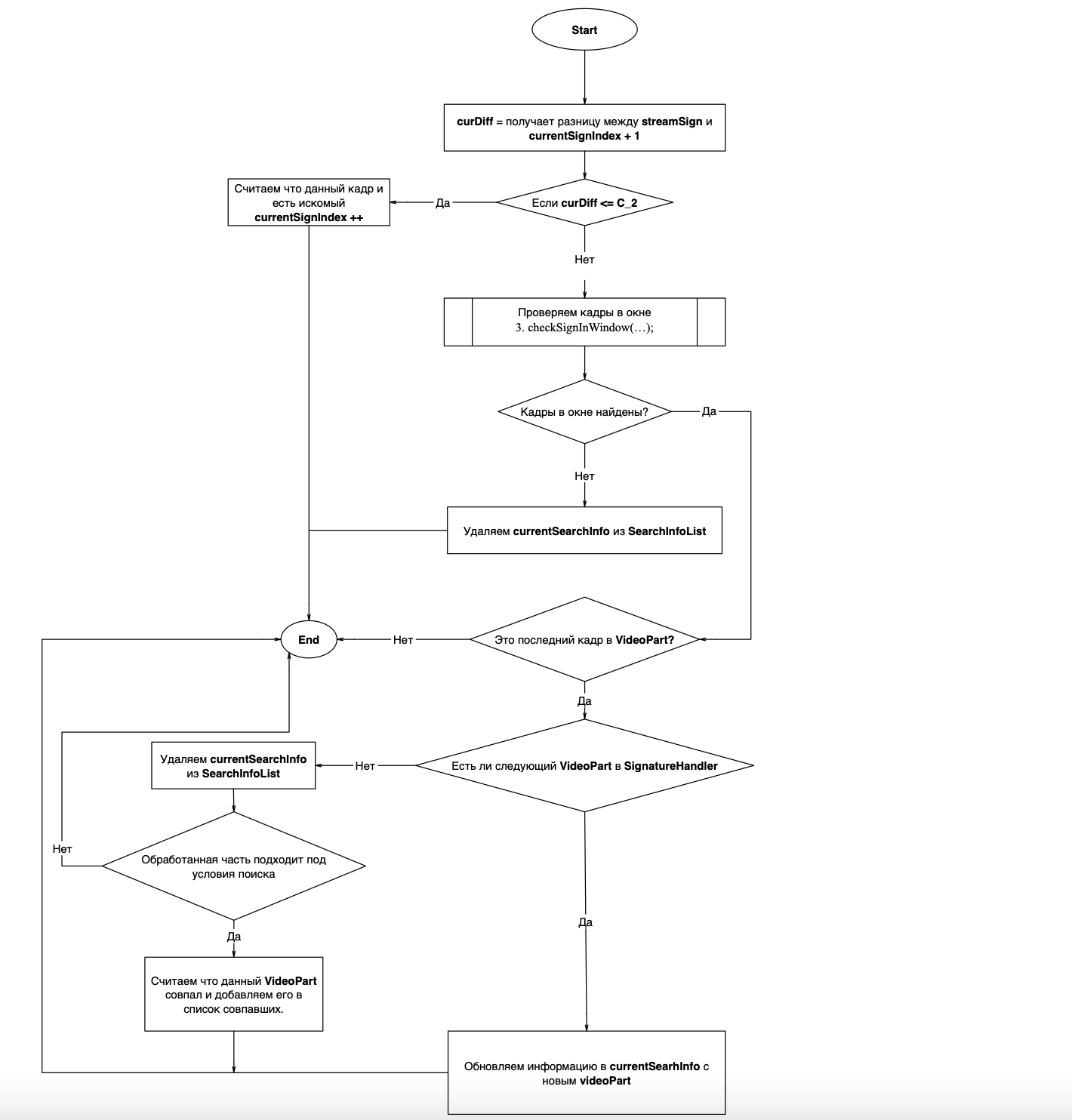
## Работа с **VideoPart** в рамках **SearchInfo**.

В алгоритме, представленном в данной дипломной работе, производится работа с каждым **VideoPart** отдельно. Для начала выполняется сравнение сигнатуры кадра входящего видеопотока с сигнатурой кадра, которого мы ожидаем следующим в рамках **VideoPart**. Если результат сравнения < C\_2, считается что данный кадр и есть искомый, иначе производится работа с сигнатурами внутри окна. Если следующий кадр выходит за рамки текущего **VideoPart**, работа с данным **VideoPart** закончена. Если сегмент подходит под условия поиска, он добавляется в список совпавших частей.

Далее представлена схема работы алгоритма с очередным **VideoPart**

Данная функция выполняет работу с **SearchInfo**. Берет сигнатуру из **VideoPart** и считает разницу сигнатур. В зависимости, от которой выполняет те или иные действия.

* **currentSignIndex** - индекс текущего кадра в **VideoPart**.
* **currentSearchInfo** – класс **SearchInfo** с которым ведется работа.

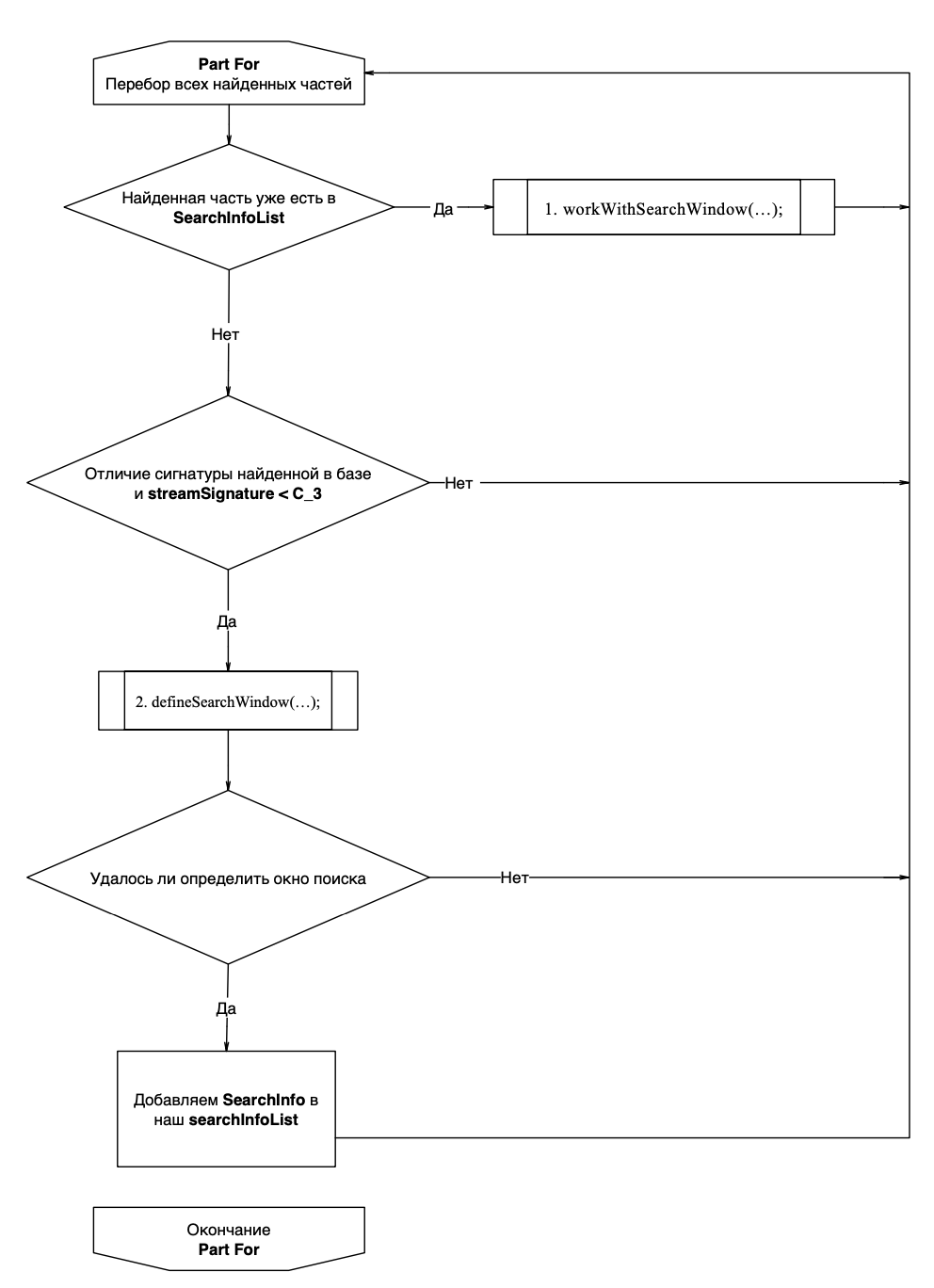


## Работа с найденными **VideoPart**

Как было описано выше, алгоритм поиска по текущему кадру из видеопотока внутри структуры хранения сегментов ролика может возвращать как несколько **VideoPart**, так и не одного. Все они сохраняются в рамках **SearchInfo** и производится обработка. Выше были описаны вспомогательные алгоритмы, которые находятся внутри основного. Основной алгоритм работает как конечный автомат. Т.е. берет кадр из входящего потока и выполняет обработку. В ходе работы этого алгоритма выполняется обработка как вновь найденного набора **VideoPart**, так и набора **VideoPart** найденных на предыдущем шаге конечного автомата. Для хранения **VideoPart** выбрана структура хранения – лист. Хоть она и не отличается высокой скоростью поиска, но в этом и нет необходимости, так как работа с **VideoPart** будет выполняться последовательно, а операции добавления и удаление производятся часто.

Далее представленная общая схема работы алгоритма при том условии, что в результате работы алгоритма поиска похожих сегментов с порогом C\_3, сегменты были найдены.

* **streamSignature** - сигнатура текущего кадра из видеопотока,
* **searchInfoList** – лист из экземпляров класса **SearchInfo**.



После окончания работы с найденными частями, выполняется повторный проход по **SearchInfoList** и обработка тех **SearchInfo**, с которыми работа не производилась, функцией **workWithSearchWindow**.