# Алгоритм поиска диссонансов временного ряда

## Определения

Временной ряд (time series) T представляет собой хронологически упорядоченную последовательность вещественных значений  $t_1, t_2, ..., t_N$ , ассоциированных с отметками времени, где N – длина последовательности.

 $Paccmosnue\ (distance)\$ между подпоследовательностями C и M представляет собой функцию, которая в качестве аргументов принимает C и M, и возвращает неотрицательное число R. Для подпоследовательностей функция расстояния является симметричной, т.е. Dist(C,M)=Dist(M,C).

Пусть имеется временной ряд T, подпоследовательность C длины n, начинающаяся с позиции p, и подпоследовательность M длины n, начинающаяся с позиции q. Подпоследовательности C и M называются несамопересекающимися (non-self match), если  $|p-q| \ge n$ .

### Определения

Подпоследовательность D длины n, начинающаяся с позиции l называется  $\partial uc-$  сонансом временного ряда T, если D находится на наибольшем расстоянии от ближайшей несамопересекающейся с D подпоследовательности, чем любые другие подпоследовательсти временного ряда, т.е.  $\forall C \in T$ , несамопересекающихся с D  $M_D$  и с C  $M_C: min(Dist(D, M_D)) > min(Dist(C, M_C))$ .

Подпоследовательность D длины n, начинающаяся с позиции p называется K-M ducconancom временного <math>pяda, если D имеет K-е по размеру расстояние от ближайшей несамопересекающейся подпоследовательности, при этом не имея пересекающихся частей с i-ми диссонансами, начинающимися на позициях  $p_i$ , для всех  $1 \le i \le K$ , т.е.  $|p-p_i| \ge n$ .

### Подходы к нахождению диссонансов:

- 2a составить матрицу расстояний для всех подпоследовательностей (для self-match будут фиктивные расстояния). Затем найти максимум из минимумов расстояний.
- 2б на основе последовательного алгоритма HOTSAX Кеога. На основе полученной эвристики выбирается оптимальный порядок перебора подпоследовательностей при поиске расстояния до ближайшего соседа. При этом находить расстояние до ближайшего соседа придется только для нескольких первых подпоследовательностей. Для оставшихся будет срабатывать условие «раннего выхода» из цикла.

## Основной алгоритм

### Входные данные

- T временной ряд  $(t_i, 1 \le i \le m)$
- *m* длина ряда
- n длина подпоследовательности
- С множество подпоследовательностей

## Результат работы

- $bsf\_loc$  позиция начала диссонанса временного ряда
- $bsf\_dist$  расстояние до ближайшего соседа подпоследовательности-диссонанса

## Алгоритм

- 1. Подготовка: составить вектор  ${\it C}$  из всех подпоследовательностей временного ряда .
- 2. Вычислить матрицу расстояний D каждой подпоследовательности временного ряда с каждой. Расстояния между подпоследовательностями  $C_i$  и  $C_j$  вычисляются по формуле:

$$D_{i,j} = \sum_{k=1}^{n} (C_i^k - C_j^k)^2$$

- 3. В каждой і-й строке матрицы D ( $1 \le i \le m-n+1$ ), соответствующей подпоследовательности  $C_i$  заменить расстояния до пересекающихся с  $C_i$  подпоследовательностей (self-match) на  $\infty$
- 4. В каждой і-й строке матрицы D находим минимальный элемент  $D_{i,j \ min}$ , формируем вектор  $V_{min}$  из элементов  $D_{i,j \ min}$ .
- 5. Находим максимальный элемент в векторе  $V_{min}$  (позиция этого элемента будет cootветствовать bsf\_pos, а значение bsf\_dist.

## Хранение подпоследовательностей временного ряда

Временной ряд Т:

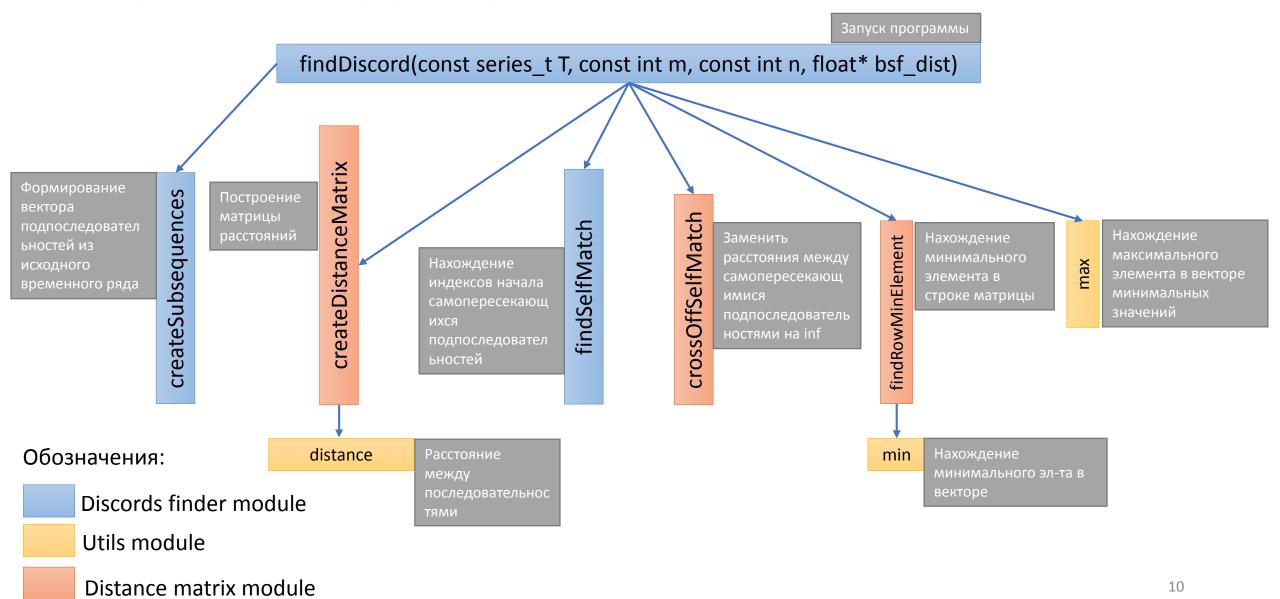


# Поиск диссонансов: построение матрицы расстояний и нахождение самого далекого из ближайших соседей для каждой из подпоследовательностей

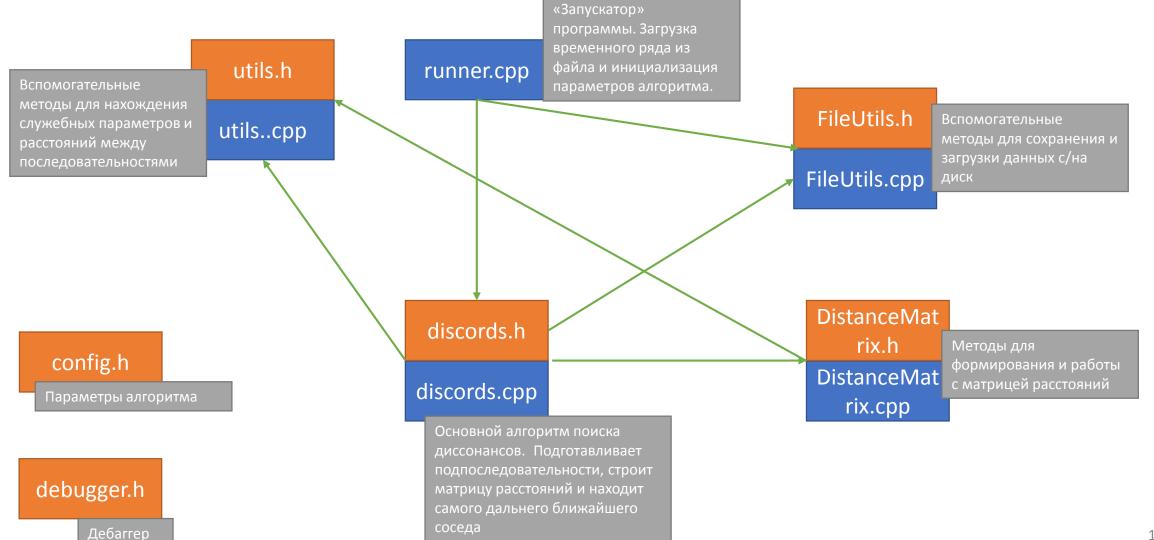


$$D_{C_i,C_j} = \text{Euclid\_dist}(C_i, C_j) = \sum_{k=1}^{n} (C_i^k - C_j^k)^2, 1 \le j \le m - n + 1$$

 $\infty$  — фиктивное расстояние (между подпоследовательностью и самой собой и между подпоследовательностью и self-match подпоследовательностями)



## Файловая структура



## Файловая структура

```
FileUtils.h
                                                          DistanceMatrix.h
                                                                                                                      Debugger.h
                                                          matrix: matrix t
char* BASE DIR = "/data/";
char* TIME SERIES FILE NAME = "timeSeries.bin";
                                                          createDistanceMatrix (timeSeriesSubsequences:
char* RESULT FILE NAME = "result.bin";
                                                          float**): matrix t
char* DISTANCE MATRIX FILE NAME =
                                                                                                                      Utils.h
                                                          findRowMinElement (rowIndex : long) : float
"timeSeries.bin":
                                                          crossOffSelfMatch (rowld : long, startIndexes :
                                                                                                                      POS INF 999999999.0f
series t readTimeSeries(ifstream reader);
                                                          long*): void
                                                                                                                      NEG INF -9999999999.0f
matrix t readDistanceMatrix(ifstream reader);
bool writeResult(ofstream writer, long bsfPos, float
                                                                                                                      distance(p1:item t, p2:item t):item t
bsfDist);
                                                                                                                      distance2 (p1: item t, p2: item t): item t
bool writeDistanceMatrix(ofstream writer, matrix t
                                                                                                                      distance2(series1: series t, series2: series t, long length): item t
distMatrix);
                                                                                                                      distance(series1: series t, series2: series t, long length): item t
                                                                                                                      min (series : series t) : item t
                                                                                                                      max (series: series t): item t
Discords.h
                                                                                                                      Config.h
timeSeries: series t
                                                                                                                      int n;
best_so_far dist: float
                                                                                                                      int m;
long best so far pos: long
                                                                                                                      typedef float item_t;
                                                                                                                      typedef item_t* series_t;
start(): void
                                                                                                                      typedefitem t** matrix t;
findDistances(): void
findBsfParams(): void
findSelfMatch(timeSeries : series t, startIndex : long) : int*
```

## Ограничения алгоритма

Размер временного ряда ~ 50 000 элементов типа float при объеме памяти ускорителя 8gb

Длина ряда	Размер элемента	Длина подпослед овательнос ти	К-во подпоследова тельностей	Эл-тов в м-це расстояний	Эл-тов в м-це подпоследовате льностей	Размер памяти, Гб
1 000 000	4	128	999 873	999 746 016 129	127 983 744	3 999,50
500 000	4	1024	498 977	248 978 046 529	510 952 448	997,96
45 000	4	1024	43 977	1 933 976 529	45 032 448	7,92
45 000	4	128	44 873	2 013 586 129	5 743 744	8,08
43 200	4	7	43 194	1 865 721 636	302 358	7,46
175 000	4	128	174 873	30 580 566 129	22 383 744	122,41
175 000	4	1024	173 977	30 267 996 529	178 152 448	121,78

## Тестовые данные

BruteForce							
DataSet	Кол-во элементов	Время работы					
HotSAX							
DataSet	Кол-во элементов	Время работы					
Последовательный с матрицей расстояний							
DataSet	Кол-во элементов	Время работы					

Old: последовательный алгоритм на основании алгоритма HOTSAX Keora (Keogh)

### Входные данные

- T временной ряд  $(t_i, 1 \le i \le m)$
- *m* длина ряда
- n длина подпоследовательности
- С множество подпоследовательностей
- w длина слова (в SAX аппроксимации подпоследовательностей),  $1 \le w \le n, n \ mod \ w = 0$
- Lookup table (LT) для точек разделения в SAX
- A мощность алфавита для SAX представления подпоследовательностей

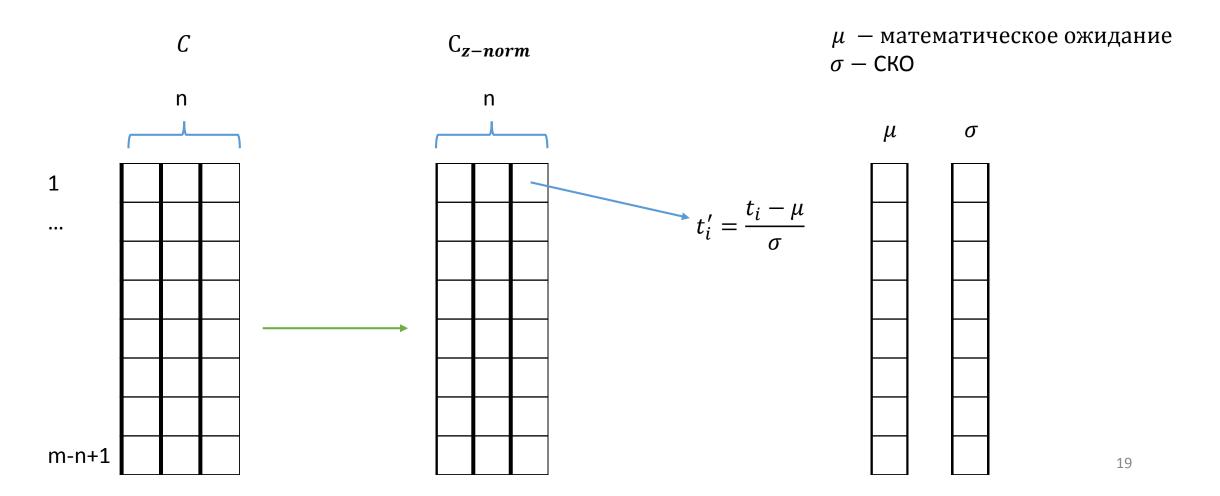
## б.) Подбор эвристики для цикла с ранним выходом из итераций. Алгоритм:

- 1. Подготовка (выбор эвристики) подбор порядка подачи подпоследовательностей, при котором возможно быстро отбрасывать неподходящие подпоследовательности.
- 2. Поиск диссонансов перебор упорядоченных подпоследовательностей временного ряда с поиском наибольшего расстояния до ближайшего соседа

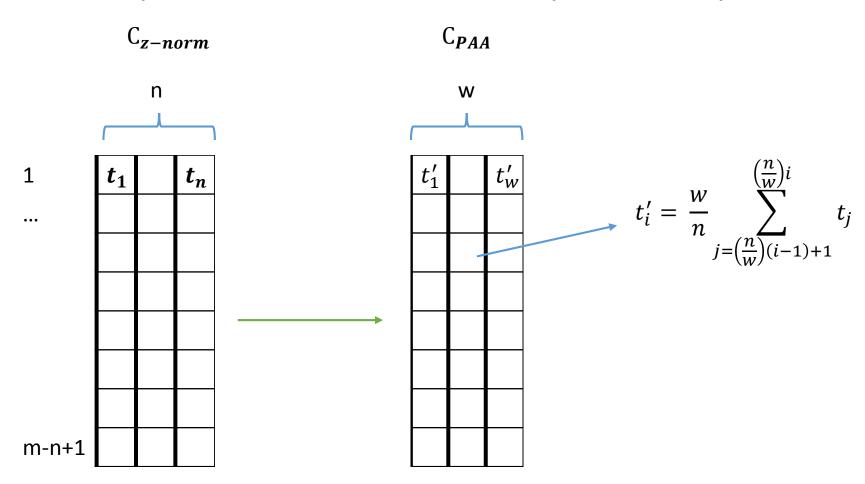
## Алгоритм:

- 1. Подготовка (выбор эвристики) подбор порядка подачи подпоследовательностей, при котором возможно быстро отбрасывать неподходящие подпоследовательности.
  - 1. Z-нормализация подпоследовательностей  $C_{i}$  временного ряда
  - 2. Кусочная аппроксимация (РАА-представление)
  - 3. Кодирование с помощью lookup table
  - 4. Подсчет частот, нахождение мин. значения
- 2. Поиск диссонансов перебор упорядоченных подпоследовательностей временного ряда с поиском наибольшего расстояния до ближайшего соседа, с постепенным обновлением best\_so\_far\_dist и «быстрым» выходом из итераций цикла.

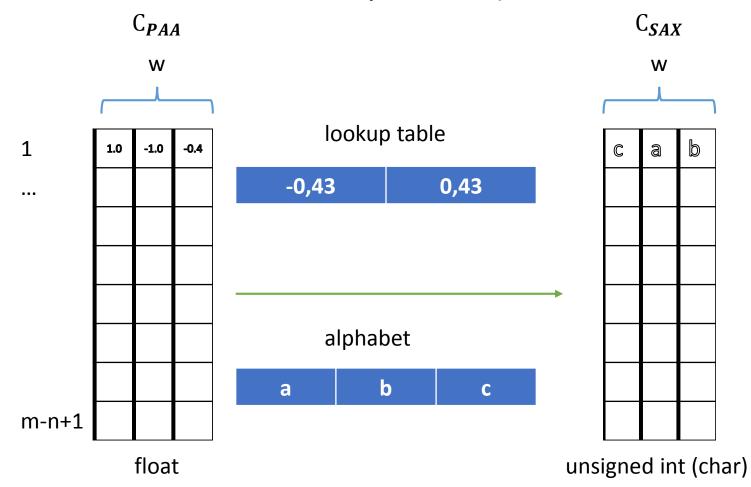
#### 1.1 z-нормализация подпоследовательностей $\mathsf{C}_i$ временного ряда



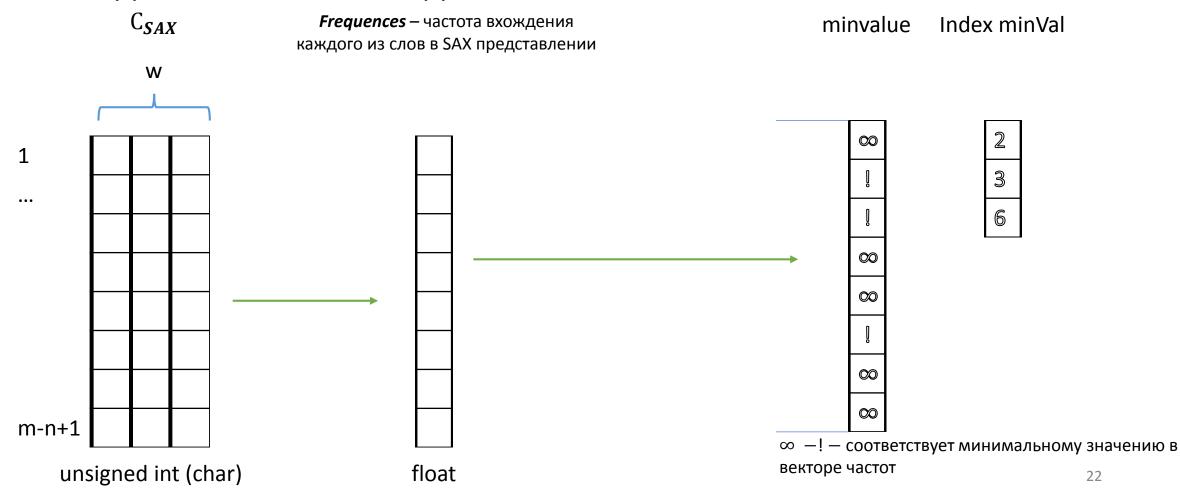
#### 1.2 Аппроксимация с помощью кусочной агрегации (РАА)



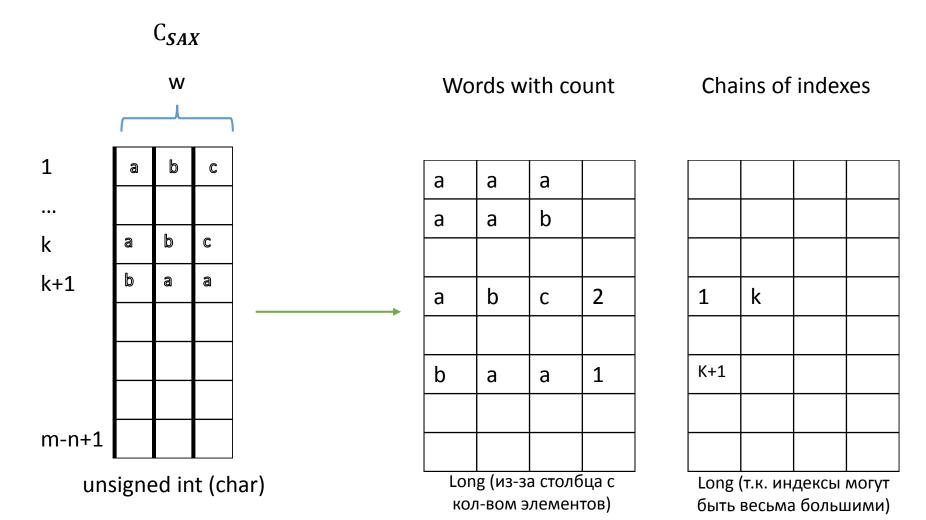
1.3 Кодирование с помощью lookup table (аппроксимация с помощью символьной агрегации)



#### 1.4 Подсчет частот, нахождение мин. значения



1.5 Построение цепочек с индексами начала подпоследовательностей, имеющих одинаковое SAX представление



```
For(int p = 0; p < m-n+1; p++) {
            min = ∞;
            For(int j = 0; j < m-n+1; j++) {
                         if(is_self_match(C_p, C_i) {
                                      continue;
                         dist_i = \text{Euclid\_dist}(C_p, C_i)
                         if (dist<sub>i</sub> < best_so_far_dist) {</pre>
                                      -> C_n не подходит
                                      break;
                         if (dist_i < min) {
                                      min = dist_i;
            If(min > best_so_far_dist) {
                         best_so_far_dist = min;
                         best so far pos = p;
```

Последовательная версия алгоритма
Output: диссонансом является подпоследовательность начинающаяся с best\_so\_far\_pos позиции в исходном временном ряде.

```
For(int p in IndexMinVal) {
                                         Не распараллеливаем, т.к.
  min = \infty;
                                        предположительно мало
  For(int j in ChainsOfIndexes[sax(C_n)]){
                                        итераций
    if(is self match(C_n, C_i) {
      continue;
    dist_i = Euclid\_dist(C_p, C_i)
   if (dist_i < best\_so\_far\_dist) { -> C_p} не подходит Ранний выход
      break:
    if (dist_i < min) {
      min = dist_i;
  #pragma omp parallel for scheduled(guided)
 For(int j int j = 0; j < m-n+1 && not in ChainsOfIndexes[sax(C_n)]; j++){
    if(is_self_match(C_n, C_i) {
      continue;
    dist_i = Euclid dist(C_n, C_i)
    if (dist_i < best\_so\_far\_dist) { -> C_p не подходит Ранний выход
      break:
    if (dist_i < min) {
      min = dist_i;
 If(min > best so far dist) {
    best_so_far_dist = min;
    best so far pos = p;
```

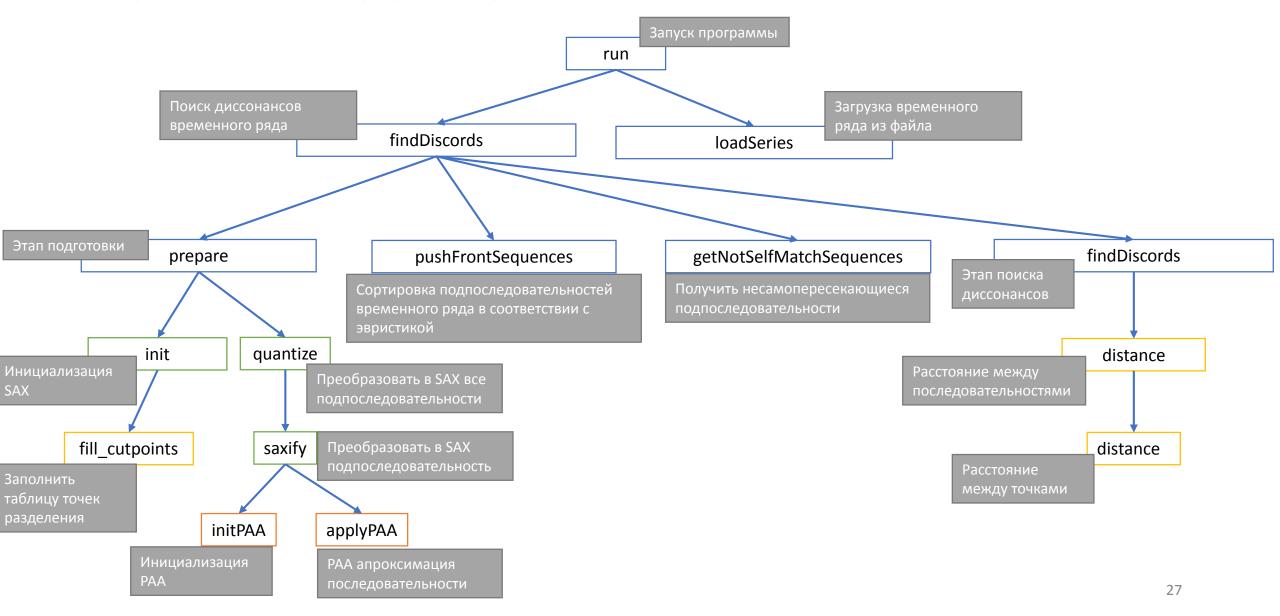
Для получения наилучшего (наиболее близкого к диссонансу) значения best\_so\_far\_dist выполняется внешний цикл по всем подпоследовательностям с наиболее редко встречающимися SAX представлениями И два внутренних цикла: по предположительным ближайшим соседям подпоследовательностей из внешнего цикла и второй по всем оставшимся подпоследовательностям.

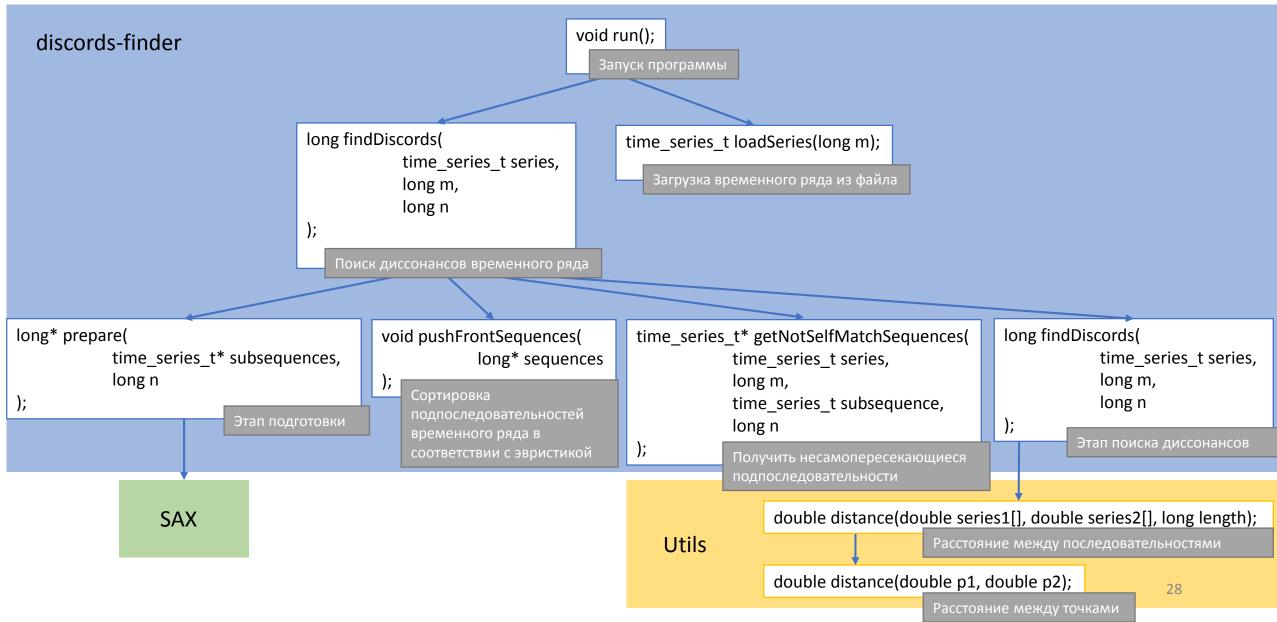
Output: диссонансом является подпоследовательность начинающаяся с best\_so\_far\_pos позиции в исходном временном ряде.

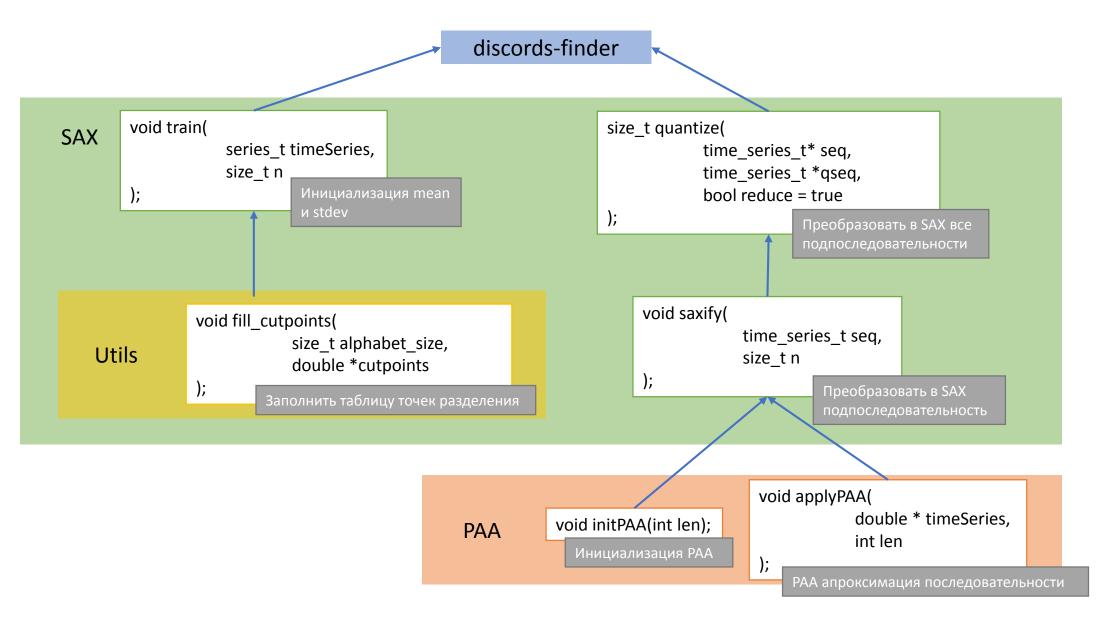
```
#pragma omp parallel for scheduled(guided)
For(int p = 1; p not in IndexMinVal && p < m - n + 1; p++) {
  min = ∞;
  For(int j in ChainsOfIndexes[sax(C_n)]){
    if(is_self_match(C_p, C_i) {
      continue;
    dist_i = Euclid\_dist(C_n, C_i)
    if (dist_j < 	ext{best\_so\_far\_dist}) { -> C_p не подходит
                                                      Ранний выход
      break;
    if (dist_i < min) {
      min = dist_i;
 For(int j int j = 0; j < m-n+1 && not in ChainsOfIndexes[sax(C_n)]; j++){
    if(is_self_match(C_n, C_i) {
      continue:
    dist_i = Euclid dist(C_n, C_i)
    if (dist_i < best\_so\_far\_dist) { -> C_p не подходит Ранний выход
      break:
    if (dist_i < min) {
      min = dist_i;
 If(min > best so far dist) {
    best_so_far_dist = min;
    best so far pos = p;
```

Далее внешний цикл по всем подпоследовательностям, которые не имеют наиболее редко встречающееся SAX представление. И два внутренних цикла: по предположительным ближайшим соседям подпоследовательностей из внешнего цикла и второй по всем оставшимся подпоследовательностям.

Output: диссонансом является подпоследовательность начинающаяся с best\_so\_far\_pos позиции в исходном временном ряде.





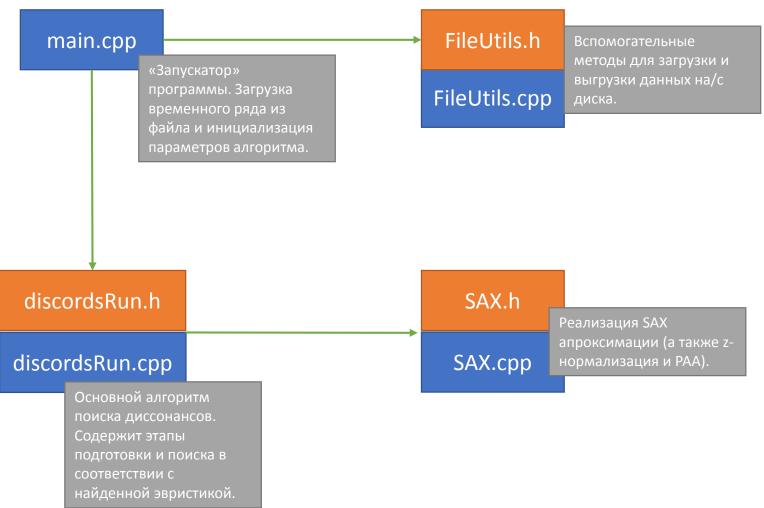


## Файловая структура

config.h
Параметры алгоритма

debugger.h

Дебаггер



## Файловая структура

#### double \* paaRepresentation; int length; int\* counts; void initPAA(int len); void applyPAA(double \* timeSeries, int len);

# size\_t m\_window\_size; size\_t m\_string\_size; size\_t m\_alphabet\_size; timeseries\_properties\_t timeseriesWithProperties;

```
void init(const vector<double> *timeSeries,
size_t window_size, size_t string_size, size_t
alphabet_size);
void saxify(vector<double> * seq,
vector<char> *syms);
size_t quantize(const vector<double> * seq,
vector<int> *qseq, bool reduce = true);
```

#### Discords.h

```
typedef double * time_series_t;
#define POSITIVE_INF
double best_so_far_dist;
long best_so_far_pos;

time_series_t* getNotSelfMatchSequences(time_series_t series, long m, time_series_t subsequence, long n);
void pushFrontSequences(long* sequences);
long* prepare(time_series_t* subsequences, long n);
long findDiscords(time_series_t series, long m, long n);
void run();
```

#### Utils.h

```
double m_baseline_stdev; // stdev of series
bool m_trained; // mean and stdev was found
};

void fill_cutpoints(size_t alphabet_size, vector<double> *cutpoints);
double distance(double p1, double p2);
double distance2(double p1, double p2);
double distance2(double series1[], double series2[], long length);
double distance(double series1[], double series2[], long length);
timeseries properties t findTimeSeriesProperties(vector<double> *timeSeries);
```

double \* zNormalization(const double \*seq, double baseline mean, double

#### Params.h

baseline stdev);

```
time_series_t originalSeries;
int n;
int m;
int w;
int A;
```

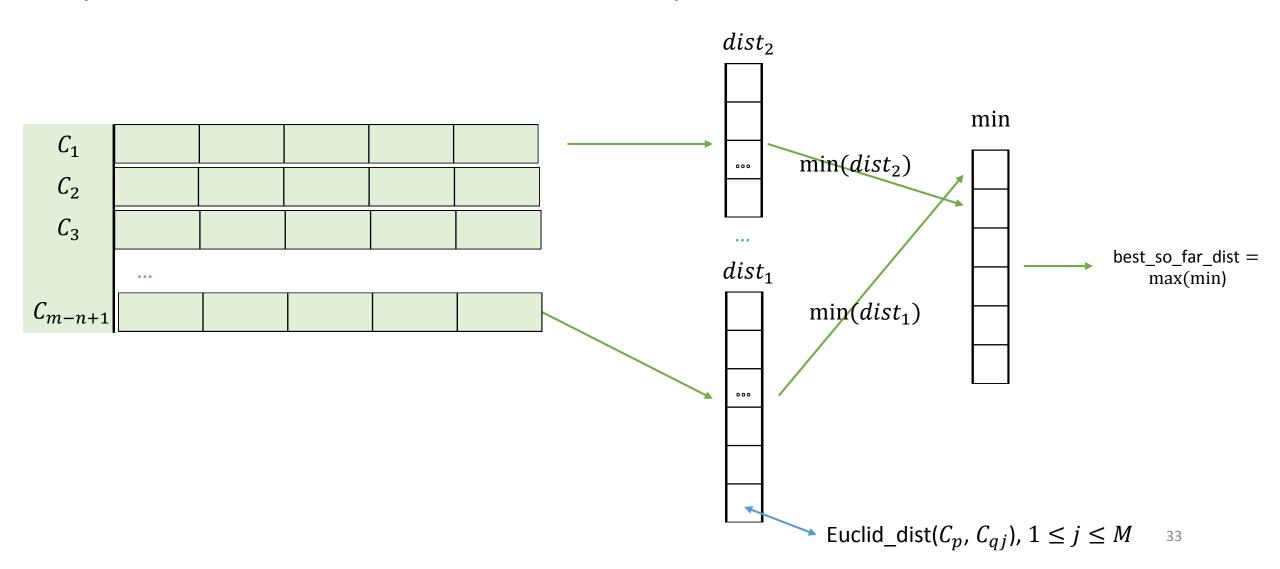
typedef struct timeseries properties t {

vector<double> \*timeSeries;// time series

double m baseline mean; // mean of series

#### Debugger.h

## Доп. Слайды (архив)



## Упорядочивание множества подпоследовательностей

Index minVal

