Алгоритм поиска диссонансов временного ряда

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

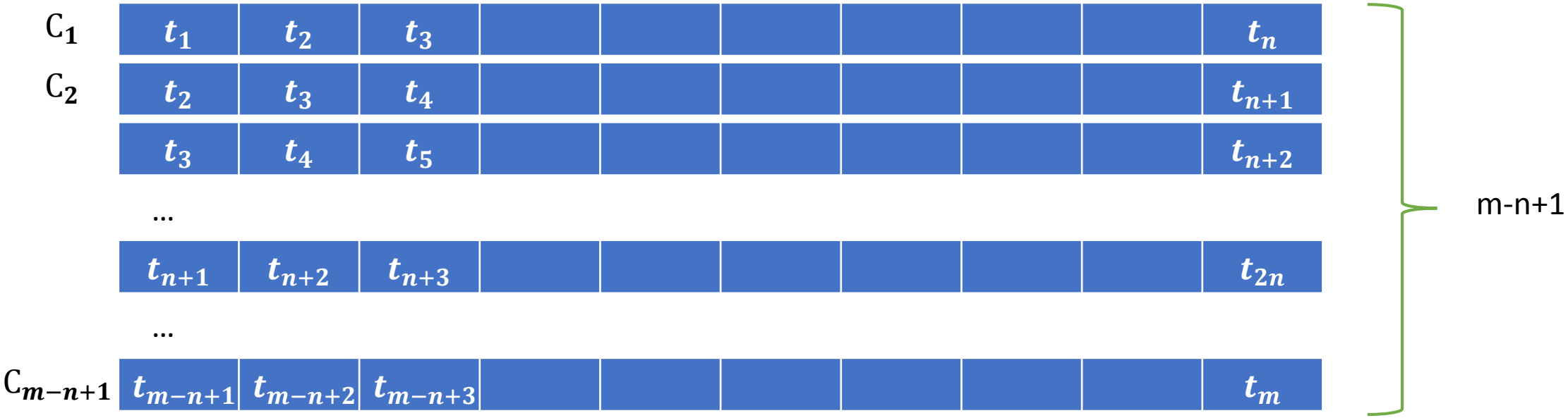
Входные данные

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

Хранение подпоследовательностей временного ряда

Временной ряд T:

Подпоследовательности C:

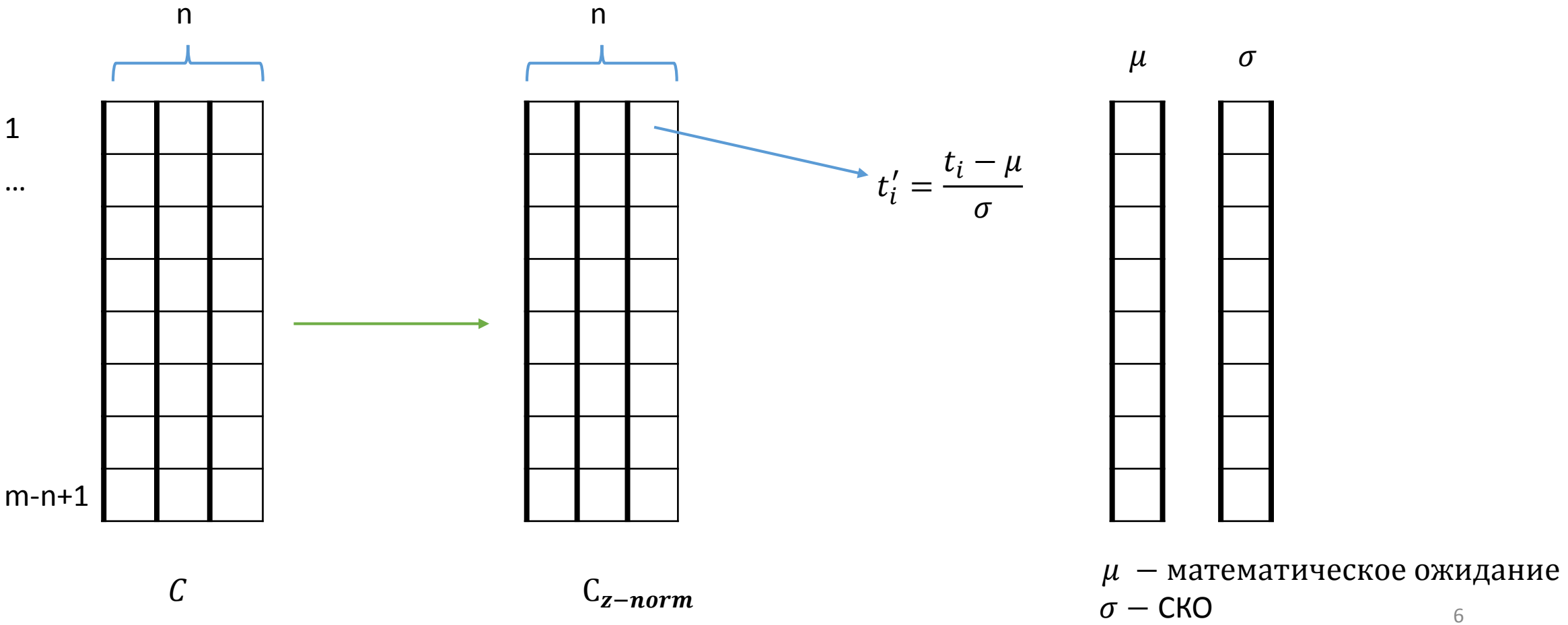


Алгоритм:

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 (PAA-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

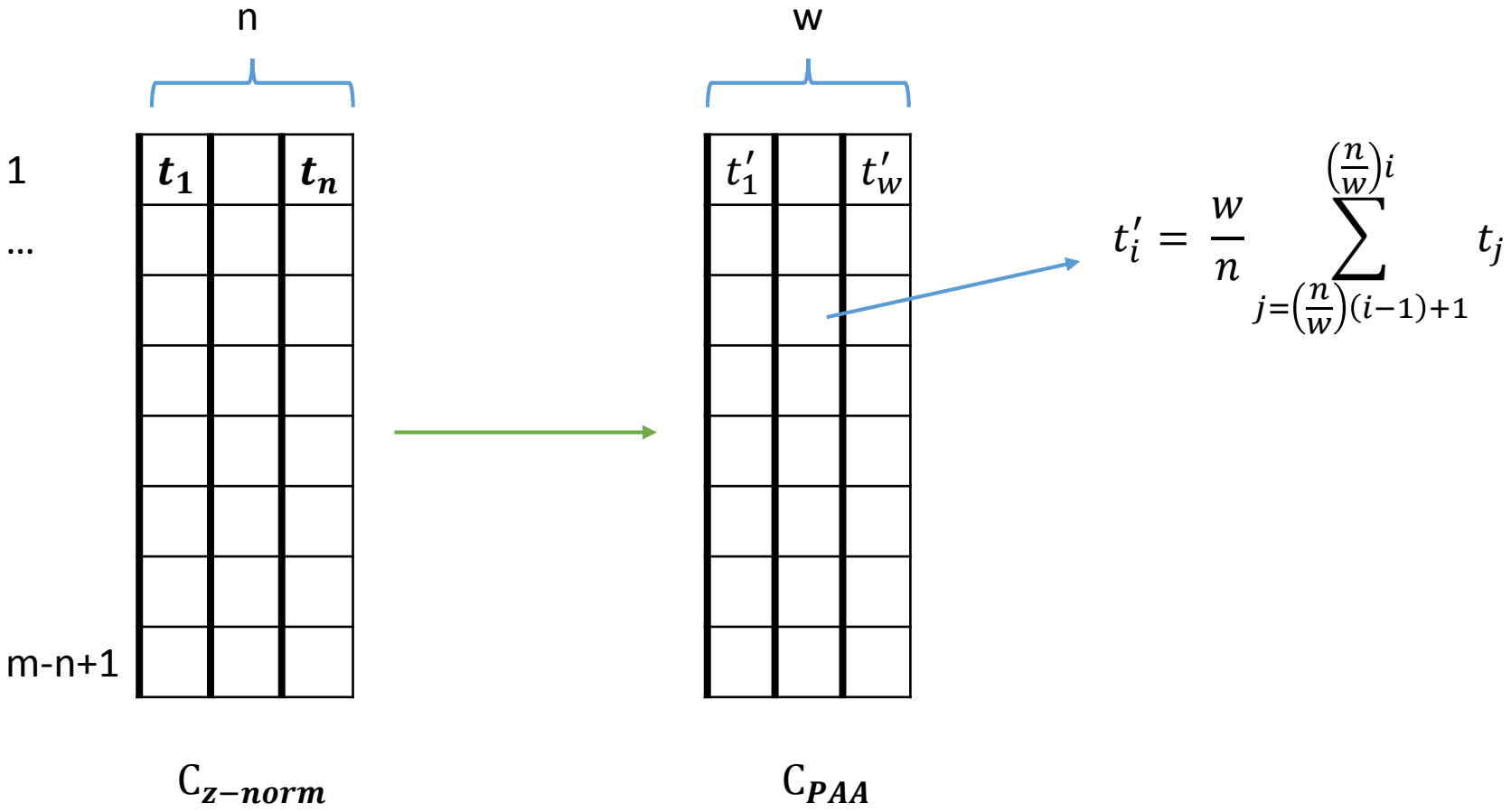
1. Подбор эвристики

1.1 z-нормализация подпоследовательностей C_i временного ряда



1. Подбор эвристики

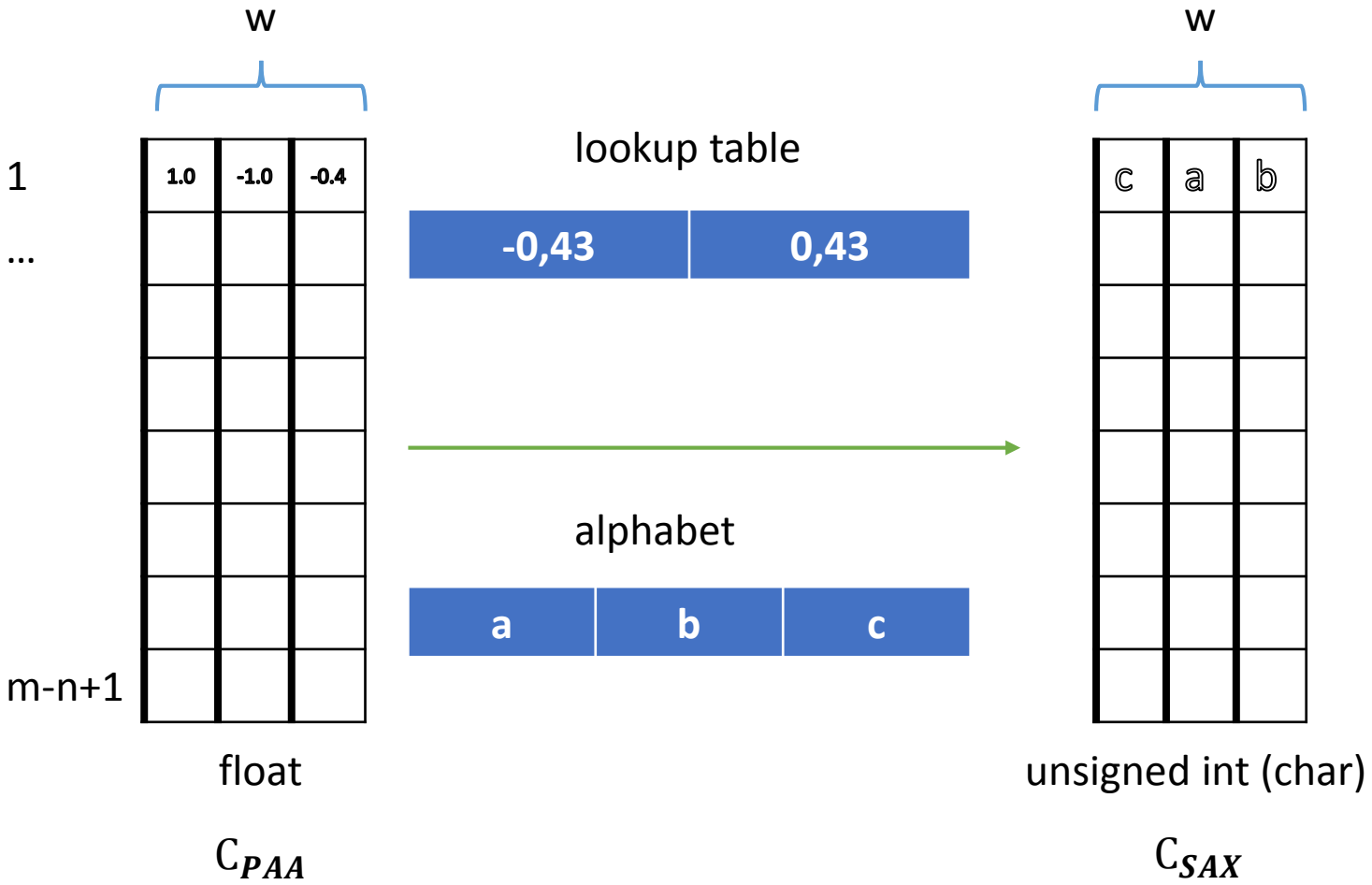
1.2 Кусочная аппроксимация (PAA-представление)



$$t'_i = \frac{w}{n} \sum_{j=\left(\frac{n}{w}\right)(i-1)+1}^{\left(\frac{n}{w}\right)i} t_j$$

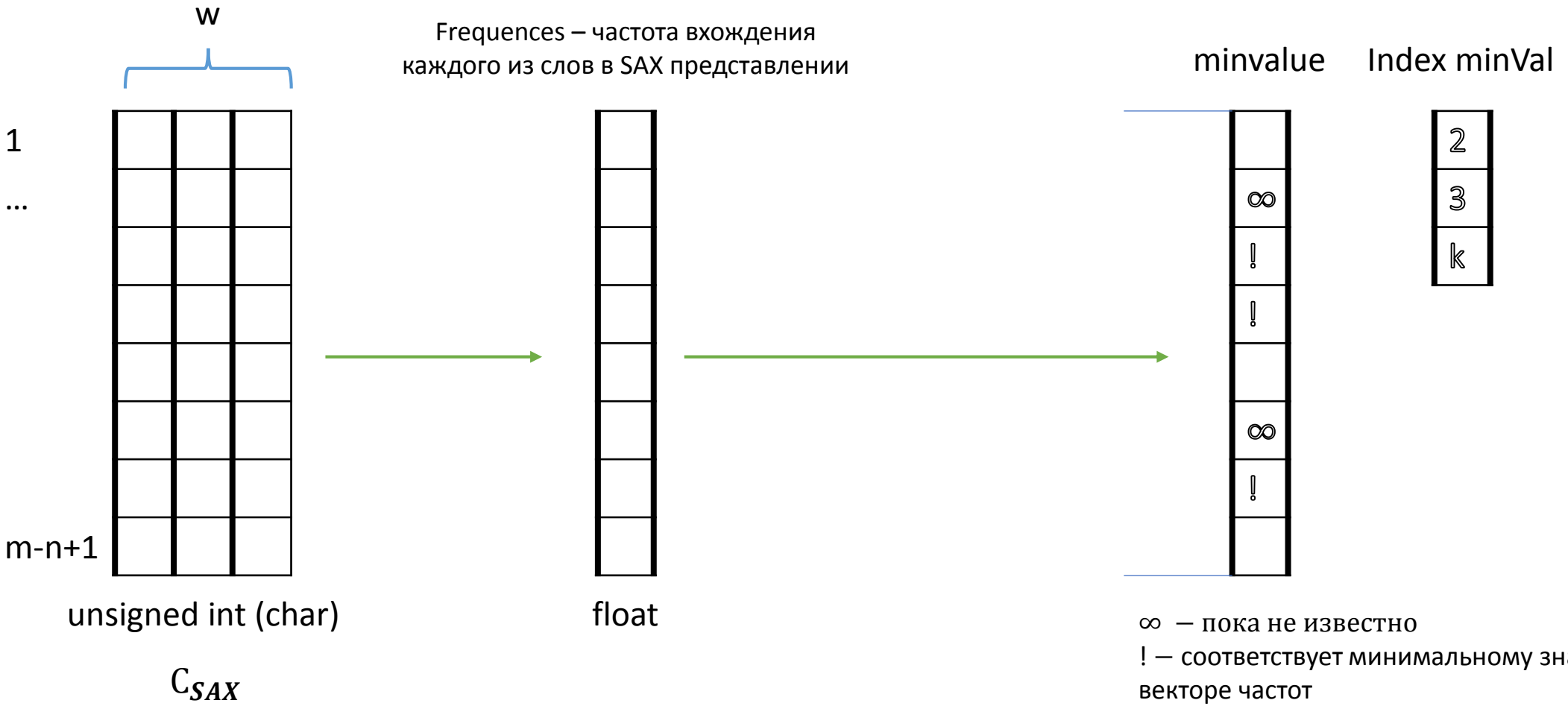
1. Подбор эвристики

1.3 Кодирование с помощью lookup table



1. Подбор эвристики

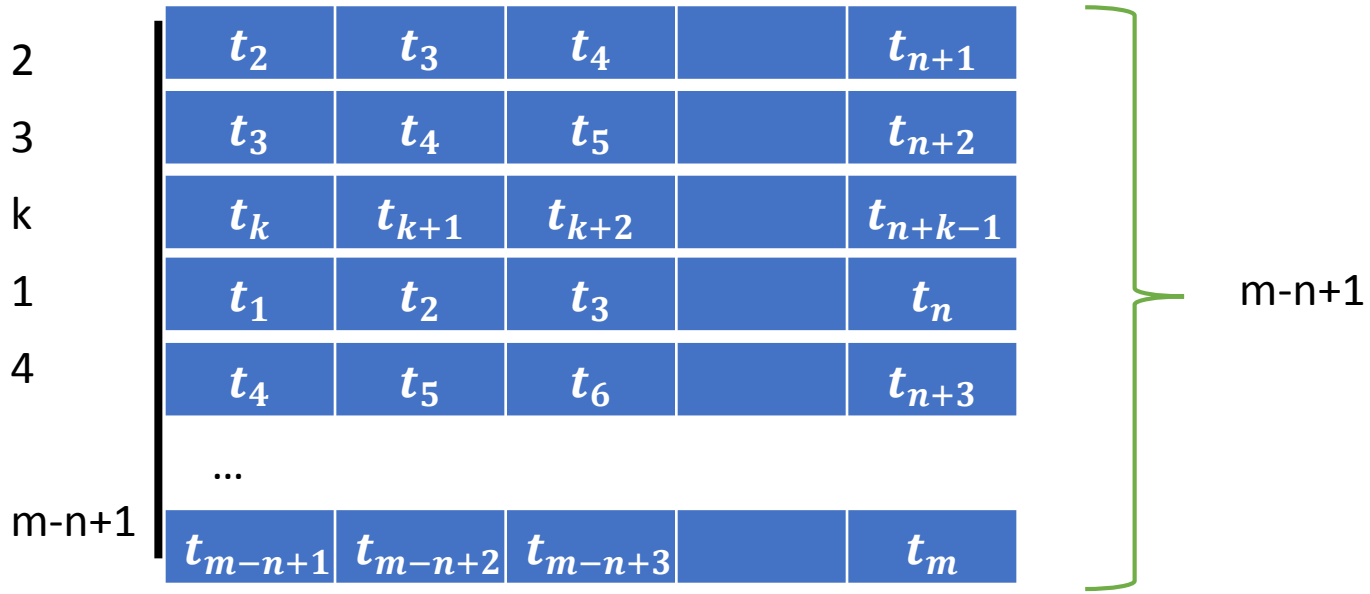
1.4 Подсчет частот, нахождение мин. значения



Упорядочивание множества подпоследовательностей

Index minVal

2
3
k

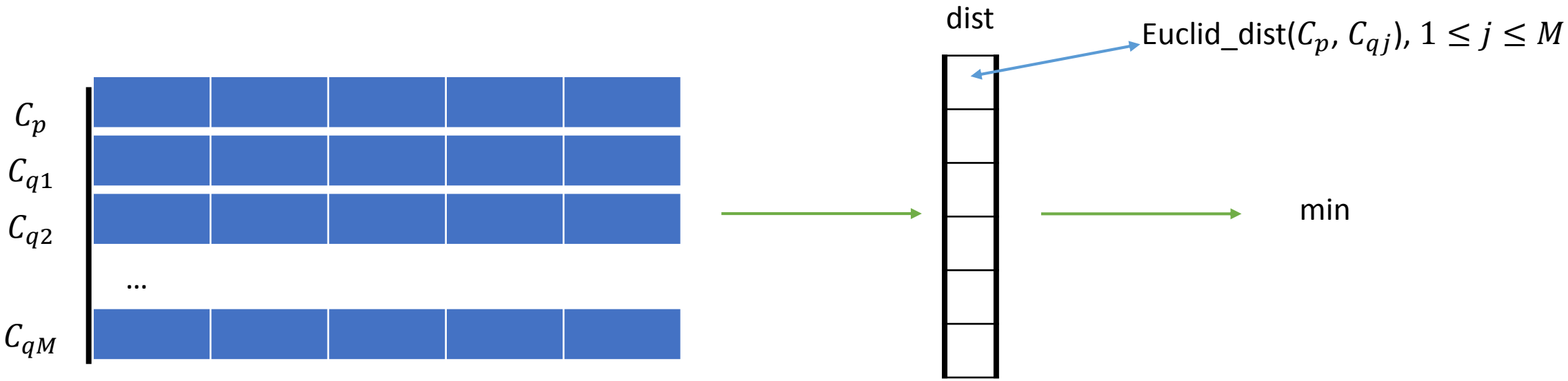


Алгоритм:

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
    1. Нахождение оптимальной best_so_far_dist (для подпоследовательностей - предположительных диссонансов)
    2. Нахождение расстояния до ближайшего соседа для оставшихся подпоследовательностей

2. Поиск диссонансов

2.1 Нахождение оптимальной best_so_far_dist (для подпоследовательностей - предположительных диссонансов)

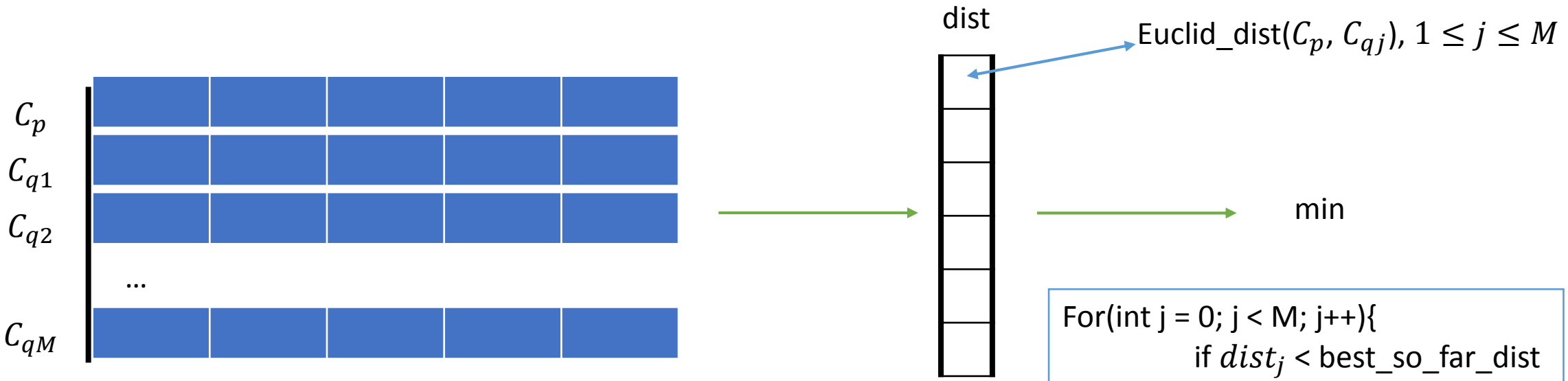


C_p – подпоследовательность, для которой нужно найти расстояние до ближайшего соседа

$\{C_{qi}\}$ – множество $not - selfmatch$ с C_p подпоследовательностей

2. Поиск диссонансов

2.2 Нахождение расстояния до ближайшего соседа для оставшихся подпоследовательностей




```
For(int j = 0; j < M; j++){
        if dist_j < best_so_far_dist
                -> не подходит
}
If(min > best_so_far_dist){
        best_so_far_dist = min;
        best_so_far_pos = p;
}
```

Output: диссонансом является подпоследовательность начинающаяся с best_so_far_pos позиции в исходном временном ряде.

Модульная структура

PAA

```
double * paaRepresentation;
int length;
int* counts;
```

```
void initPAA(int len);
void applyPAA(double * timeSeries, int len);
```

SAX

```
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);
```

Utils

```
typedef struct timeseries_properties_t {
 vector<double> *timeSeries;// time series
 double m_baseline_mean; // mean of series
 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);
vector<double> * zNormalization(const vector<double> *seq, double baseline_mean,
double baseline_stdev);
```

discords

```
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);
```

Params

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

Debugger

Файловая структура