Описание алгоритма/метода

Лекция 5

План описания алгоритма/метода

- История
- Математическое описание
- Алгоритм/метод
  - Схема алгоритма
  - Псевдокод
- Временн*а*я сложность
- Примеры
- Применение

История

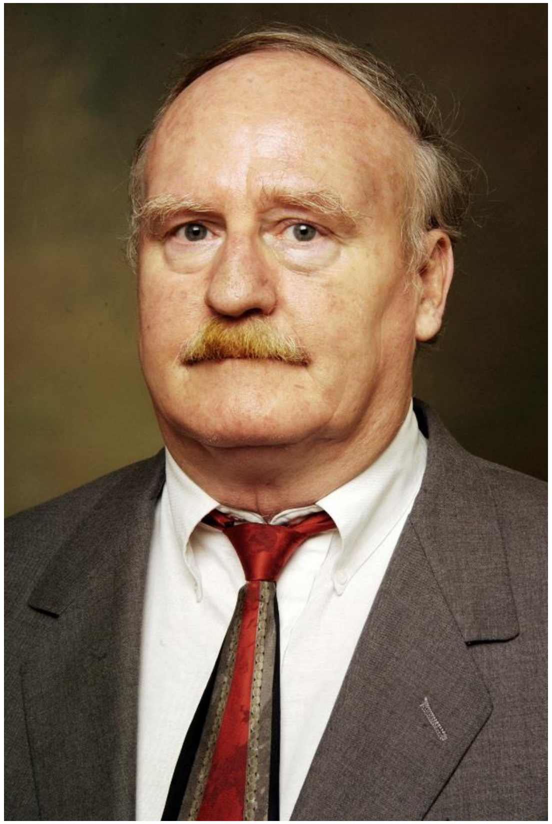
- Алгоритм Бойера-Мура – метод поиска подстроки в строке

История

- Алгоритм Бойера-Мура
- Кто авторы метода?

История

Robert Stephen Boyer



(год рожд. ?)

J Strother Moore

(год рожд. ?)

История

- Когда впервые был опубликован?

История

- Boyer R. S., Moore J. S. A Fast String Searching Algorithm // Communications of the ACM. 1977. Vol. 20. P. 762-772.

Programming Techniques

G. Manacher, S.L. Graham Editors

A Fast String Searching Algorithm

Robert S. Boyer
Stanford Research Institute

J Strother Moore
Xerox Palo Alto Research Center

An algorithm is presented that searches for the location, "*i*," of the first occurrence of a character string, "*pat*," in another string, "*string*." During the search operation, the characters of *pat* are matched starting with the last character of *pat*. The information gained by starting the match at the end of the pattern often allows the algorithm to proceed in large jumps through the text being searched. Thus the algorithm has the unusual property that, in most cases, not all of the first *i* characters of *string* are inspected. The number of characters actually inspected (on the average) decreases as a function of the length of *pat*. For a random English pattern of length 5, the algorithm will typically inspect *i*/4 characters of *string* before finding a match at *i*. Furthermore, the algorithm has been implemented so that (on the average) fewer than *i* + *patlen* machine instructions are executed. These conclusions are supported with empirical evidence and a theoretical analysis of the average behavior of the algorithm. The worst case behavior of the algorithm is linear in *i* + *patlen*, assuming the availability of array space for tables linear in *patlen* plus the size of the alphabet.

Key Words and Phrases: bibliographic search, computational complexity, information retrieval, linear time bound, pattern matching, text editing

CR Categories: 3.74, 4.40, 5.25

1. Introduction

Suppose that *pat* is a string of length *patlen* and we wish to find the position *i* of the leftmost character in the first occurrence of *pat* in some string *string*:

```
pat:        AT-THAT
string: ... WHICH-FINALLY-HALTS.--AT-THAT-POINT ...
i:                                ↑
```

The obvious search algorithm considers each character position of *string* and determines whether the successive *patlen* characters of *string* starting at that position match the successive *patlen* characters of *pat*. Knuth, Morris, and Pratt [4] have observed that this algorithm is quadratic. That is, in the worst case, the number of comparisons is on the order of $i * patlen$.[1]

Knuth, Morris, and Pratt have described a linear search algorithm which preprocesses *pat* in time linear in *patlen* and then searches *string* in time linear in $i + patlen$. In particular, their algorithm inspects each of the first $i + patlen - 1$ characters of *string* precisely once.

We now present a search algorithm which is usually "sublinear": It may not inspect each of the first $i + patlen - 1$ characters of *string*. By "usually sublinear" we mean that the expected value of the number of inspected characters in *string* is $c * (i + patlen)$, where $c < 1$ and gets smaller as *patlen* increases. There are patterns and strings for which worse behavior is exhibited. However, Knuth, in [5], has shown that the algorithm is linear even in the worst case.

The actual number of characters inspected depends on statistical properties of the characters in *pat* and *string*. However, since the number of characters inspected on the average decreases as *patlen* increases, our algorithm actually speeds up on longer patterns.

Furthermore, the algorithm is sublinear in another sense: It has been implemented so that on the average it requires the execution of fewer than $i + patlen$ machine instructions per search.

The organization of this paper is as follows: In the next two sections we give an informal description of the algorithm and show an example of how it works. We then define the algorithm precisely and discuss its efficient implementation. After this discussion we present the results of a thorough test of a particular

Математическое описание

- Ввести все обозначения
- Строгость и корректность
- Нумерация формул

Алгоритмы

- Схема алгоритма
- Псевдокод

Алгоритм – схема алгоритма

- ГОСТ 19.701-90

ГОСУДАРСТВЕННЫЙ СТАНДАРТ
СОЮЗА ССР

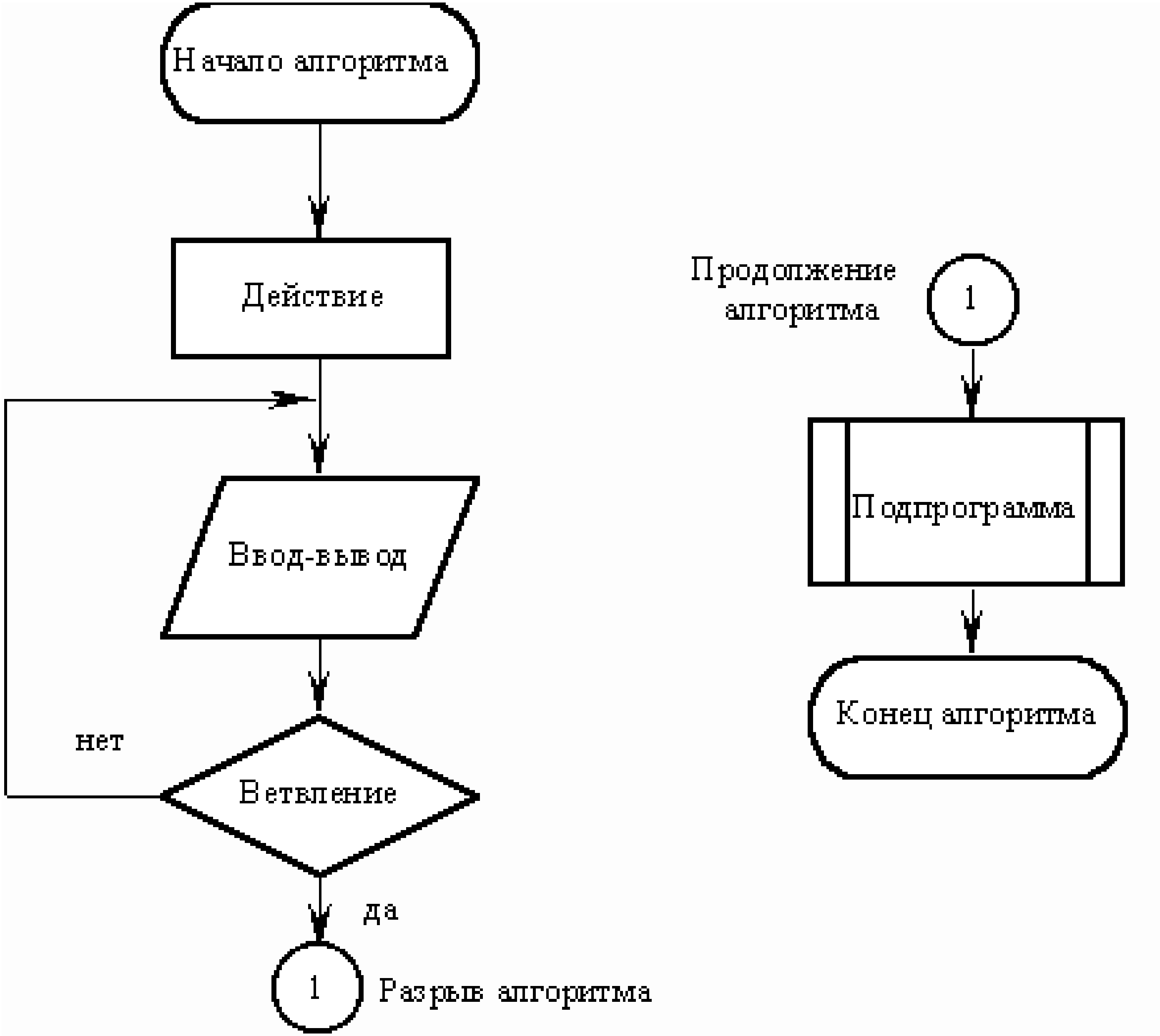
ЕДИНАЯ СИСТЕМА ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

**СХЕМЫ АЛГОРИТМОВ, ПРОГРАММ,
ДАННЫХ И СИСТЕМ**

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ И ПРАВИЛА ВЫПОЛНЕНИЯ

**ГОСТ 19.701—90
(ИСО 5807—85)**

Издание официальное


Начало алгоритма
Действие
Ввод-вывод
нет
Ветвление
да
1
Разрыв алгоритма
Продолжение алгоритма
1
Подпрограмма
Конец алгоритма

Алгоритм – псевдокод

- Стандартов нет

[Ахо, Хопкрофт, Ульман, 2003]

```
procedure INSERT (x: elementtype; p: position; var L: LIST );
    { INSERT вставляет элемент x в позицию p в списке L }
    var
        q: position;
    begin
        if L.last >= maxlength then
            error('Список полон')
        else if (p > L.last + 1) or (p < 1) then
            error('Такой позиции не существует')
        else begin
            for q:= L.last downto p do
                { перемещение элементов из позиций p, p+1, ... на
                    одну позицию к концу списка }
                L.elements[q+1]:= L.elements[q];
            L.last:= L.last + 1;
            L.elements[p]:= x
        end
    end; { INSERT }
```

[Кормен, Лейзерсон, Ривест, Штайн, 2013]

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Вставка A[j] в отсортированную
        последовательность A[1 .. j − 1].
    i = j − 1
    while i > 0 и A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

[Dasgupta, Papadimitriou, Vazirani, 2006]

Figure 4.3 Breadth-first search.

```
procedure bfs(G, s)
Input:    Graph G = (V, E), directed or undirected; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞

dist(s) = 0
Q = [s] (queue containing just s)
while Q is not empty:
   u = eject(Q)
   for all edges (u, v) ∈ E:
      if dist(v) = ∞:
         inject(Q, v)
         dist(v) = dist(u) + 1
```

[Дасгупта, Пападимитриу, Вазирани, 2014]

Рис. 4.3. Поиск в ширину.

процедура BFS(G, s)
{Вход: граф $G(V, E)$, вершина $s \in V$.}
{Выход: для всех вершин u, достижимых из s,
  dist$[u]$ будет равно расстоянию от s до u.}
для всех вершин $u \in V$:
  dist$[u] = \infty$
dist$[s] = 0$
$Q = \{s\}$ {очередь из одного элемента}
пока Q не пусто:
  $u = \textsc{Eject}(Q)$
  для всех рёбер $(u, v) \in E$:
    если dist$[v] = \infty$:
      $\textsc{Inject}(Q, v)$
      dist$[v] =$ dist$[u] + 1$

[Flach, 2012]

Algorithm 11.3: Boosting($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

Input : data set D; ensemble size T; learning algorithm $\mathscr{A}$.

Output : weighted ensemble of models.

$w_{1i} \leftarrow 1/|D|$ for all $x_i \in D$; // start with uniform weights

for $t = 1$ to T **do**

  run $\mathscr{A}$ on D with weights w_{ti} to produce a model M_t;

  calculate weighted error ϵ_t;

  if $\epsilon_t \geq 1/2$ **then**

    set $T \leftarrow t - 1$ and break

  end

  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$; // confidence for this model

  $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$; // increase weight

  $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$; // decrease weight

end

return $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$

AdaBoost$(\mathbf{X}, \mathbf{Y}, \mathit{WeakLearn}, T)$

Входные данные:

$(x_1, y_1), \ \ldots, \ (x_n, y_n)$ – обучающие объекты;

$y_i \in \{1, \ldots, K\}$ – метки классов;

WeakLearn – слабый обучающий алгоритм;

T – количество итераций.

Инициализация весов: $w_1(i) = 1/n, \ \ i = 1 \ldots n$;

для t от 1 до T:

  построение классификатора F_t на основе обучающих объектов с весами w_t при помощи *WeakLearn*;

  вычисление ошибки для F_t: $\varepsilon_t = \sum\limits_{i: F_t(x_i) \neq y_i} w_t(i)$;

  если $\varepsilon_t \geq 1/2$ **тогда**: $T = t - 1$ и завершение цикла;

  вычисление уверенности классификатора: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$;

  обновление весов w_t:

    если $F_t(x_i) = y_i$ **тогда**: $w_{t+1}(i) = \frac{w_t(i)}{2(1 - \varepsilon_t)}$;

    если $F_t(x_i) \neq y_i$ **тогда**: $w_{t+1}(i) = \frac{w_t(i)}{2\varepsilon_t}$.

Результат работы:

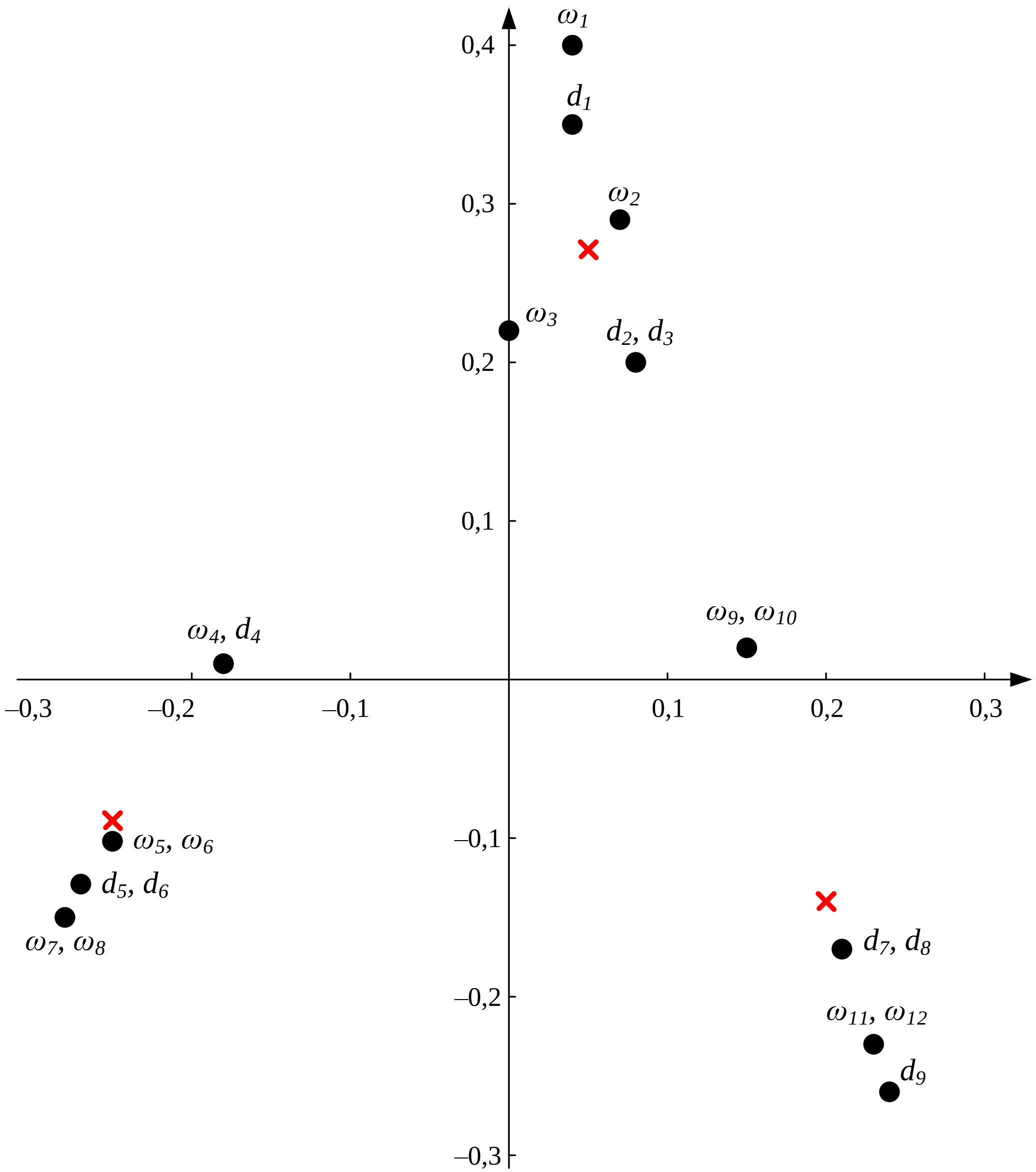
ансамбль классификаторов: $F(x) = \sum\limits_{t=1}^{T} \alpha_t F_t(x)$.

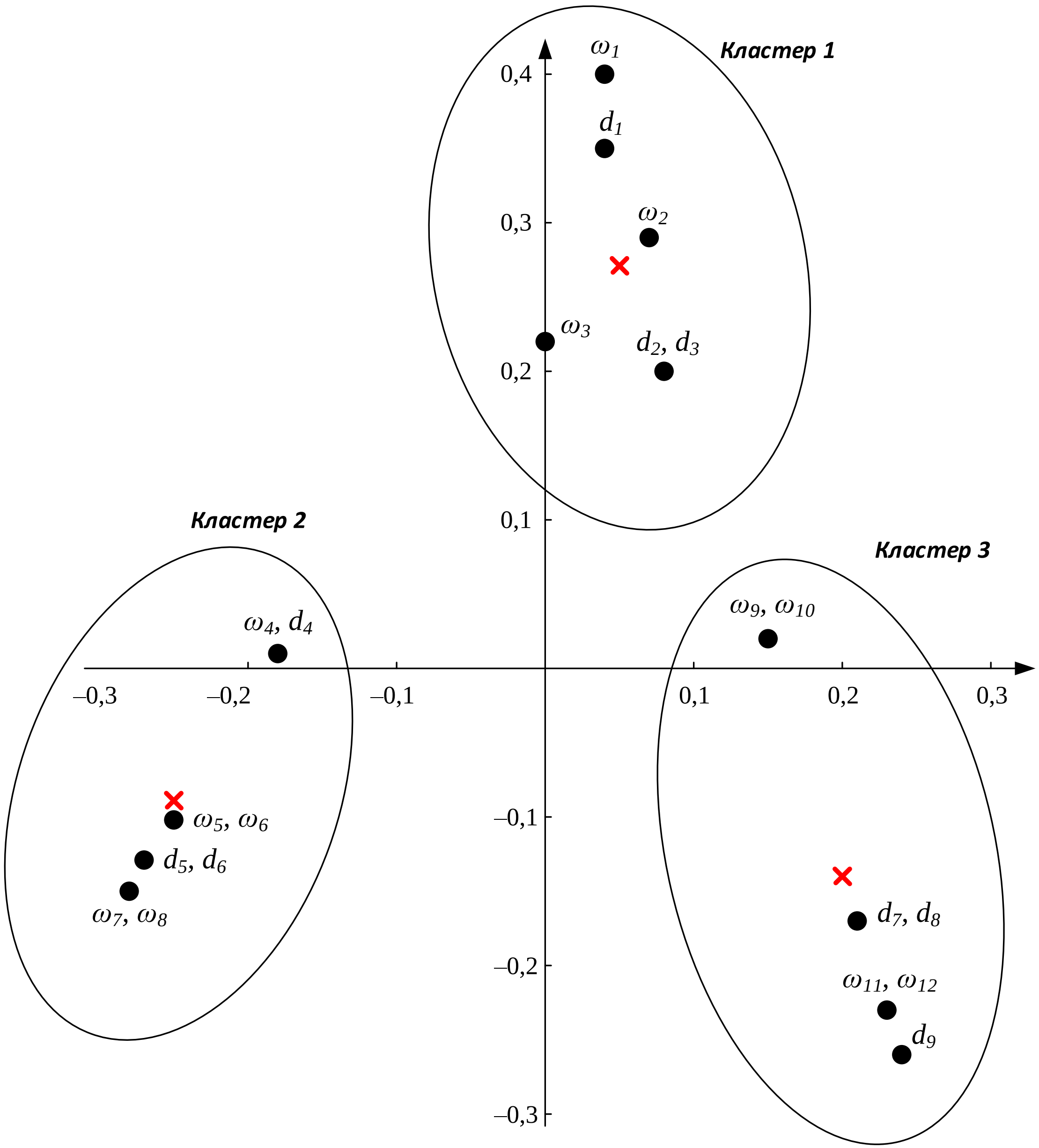
Временная сложность

- $O(C)$ – константная
- $O(\log n)$ – логарифмическая
- $O(n)$ – линейная
- $O(n^2)$ – квадратичная
- $O(n^p)$ – полиномиальная степени *p*
- $O(2^n)$ – экспоненциальная
- $\dots$

Примеры

- Простые
- С картинками
- В динамике


ω1
d1
ω2
ω3
d2, d3
0,4
0,3
0,2
0,1
–0,1
–0,2
–0,3
ω4, d4
ω9, ω10
–0,3
–0,2
–0,1
0,1
0,2
0,3
ω5, ω6
d5, d6
ω7, ω8
d7, d8
ω11, ω12
d9



Кластер 1
Кластер 2
Кластер 3
ω_1
d_1
ω_2
ω_3
d_2, d_3
ω_4, d_4
ω_5, ω_6
d_5, d_6
ω_7, ω_8
ω_9, ω_{10}
d_7, d_8
ω_{11}, ω_{12}
d_9
0,4
0,3
0,2
0,1
–0,1
–0,2
–0,3
–0,3
–0,2
–0,1
0,1
0,2
0,3

Применение

- Условия применения
- Области применения
- Ссылки

Домашнее задание

- Привести описание одного алгоритма/метода решения задачи по теме исследования в соответствии с планом описания:
  - История
  - Математическое описание
  - Алгоритм/метод
    - Схема алгоритма
    - Псевдокод
  - Временная сложность
  - Примеры
  - Применение