Московский Государственный Университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики

Разработка функциональных основ системы визуализации информационных графов алгоритмов

Дипломная работа студентки группы 423 Гадиевой Тамары Романовны
Научный руководитель: Антонов Александр Сергеевич
Москва, 2023

Цель работы и постановка задачи

Основная цель данной работы - создание описания модели информационного графа для возможности дальнейшей его визуализации в рамках общей системы визуализации информационных графов алгоритмов.

Для достижения этой цели нужно решить следующие задачи:

1. Изучить понятие информационного графа алгоритма, его свойства и способы применения и анализа; изучить синтаксис языка Algolang, предназначенного для описания графа алгоритма; изучить необходимые программные библиотеки для работы с XML-файлами и для вычисления регулярных выражений, используемых при описании графов
2. Разработать способы получения и обработки нужной информации о графе алгоритма из описания на языке Algolang
3. Разработать формат итоговой модели описания информационного графа
4. Реализовать алгоритм, который на основе информации, полученной из описания графа на языке Algolang, строит описание модели многомерного графа

Понятие информационного графа

Информационный граф алгоритма — ациклический граф, вершины которого соответствуют операциям алгоритма, а дуги - связям по данным между этими операциями.

Две вершины связываются дугой, если вторая использует данные, вычисленные в первой.

Анализ информационного графа позволяет, например, определить множества независимых друг от друга операций, найти подходящее распределение операций по процессорам вычислительной системы, обнаружить узкие места

Язык AlgoLang и правила написания на языке

```
<algo>
        <params>
                <param name="n" type="int" value="5"></param>
                <param name="m" type="int" value="4"></param>
        </params>

        <block dims="2">
                <arg name="i" val="1..m"></arg>
                <arg name="j" val="1..n+1"></arg>
                <vertex condition="j>1" type="2">
                        <in src="i,j-1"></in>
                </vertex>
        </block>
</algo>
```

рис. 1 Описание информационной структуры алгоритма умножения матрицы на вектор на языке AlgoLang

```
int n = 5;
int m = 4;

for(int i = 1; i <= m; i++)
  for(int j = 1; j <= n + 1; j++)
    vec_out[i] += matrix[i][j] * vec _in[j];
```

рис. 2 Алгоритм умножения матрицы на вектор на языке C

Схема работы программы

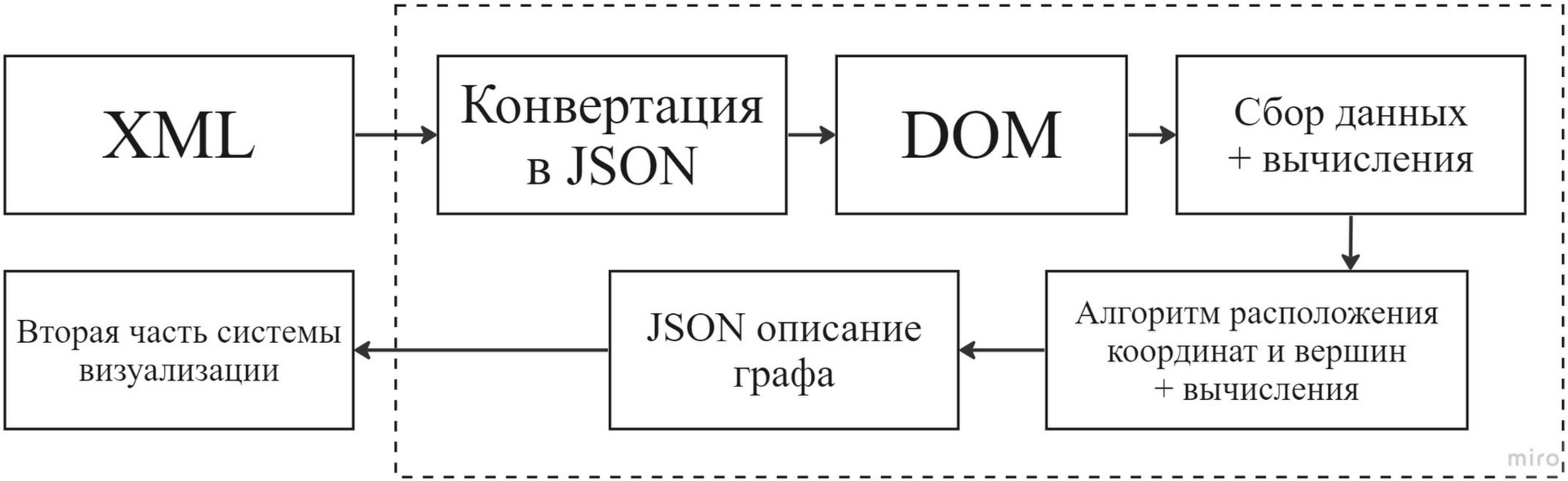


рис. 4 Блок-схема работы реализованной программы

Разбор XML файла

xml2json - библиотека для конвертирования файла формата XML в формат JSON.

RapidJSON - библиотека для парсинга файла формата JSON.
Поддерживает прикладные интерфейсы типов DOM и SAX.
В работе данной системы используется формат DOM, предназначенный для работы с небольшими файлами.

RapidJSON работает на основе библиотеки RapidXML, предназначенной для парсинга файлов формата XML, но имеет преимущества в виде более контролируемого прохода по DOM представлению данных и расширенного функционала, в том числе обработки ошибок.

Структура, используемая библиотекой RapidJSON

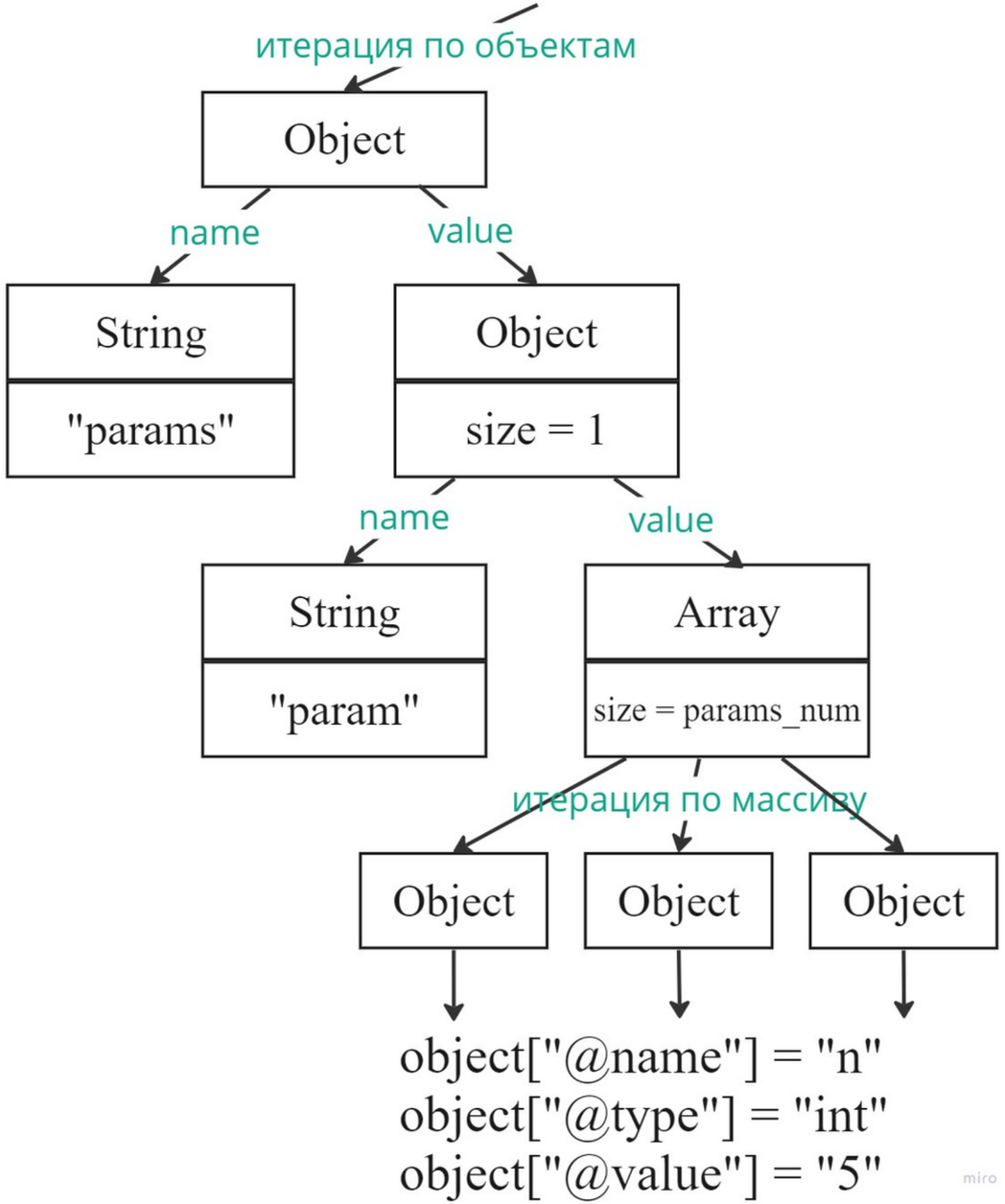


рис. 5 DOM структура описания параметров

```
<params>
        <param name="n" type="int" value="5"></param>
        <param name="m" type="int" value="4"></param>
</params>
```

рис. 6 Фрагмент программы на языке Algolang: описание параметров

Вычисления регулярных выражений

ExprTk - библиотека для вычисления регулярных выражений.

- Легко интегрируется в проекты, поскольку представляет собой один заголовочный файл
- Быстро производит вычисления
- Поддерживает большое количество различных операций

Алгоритм построения модели графа

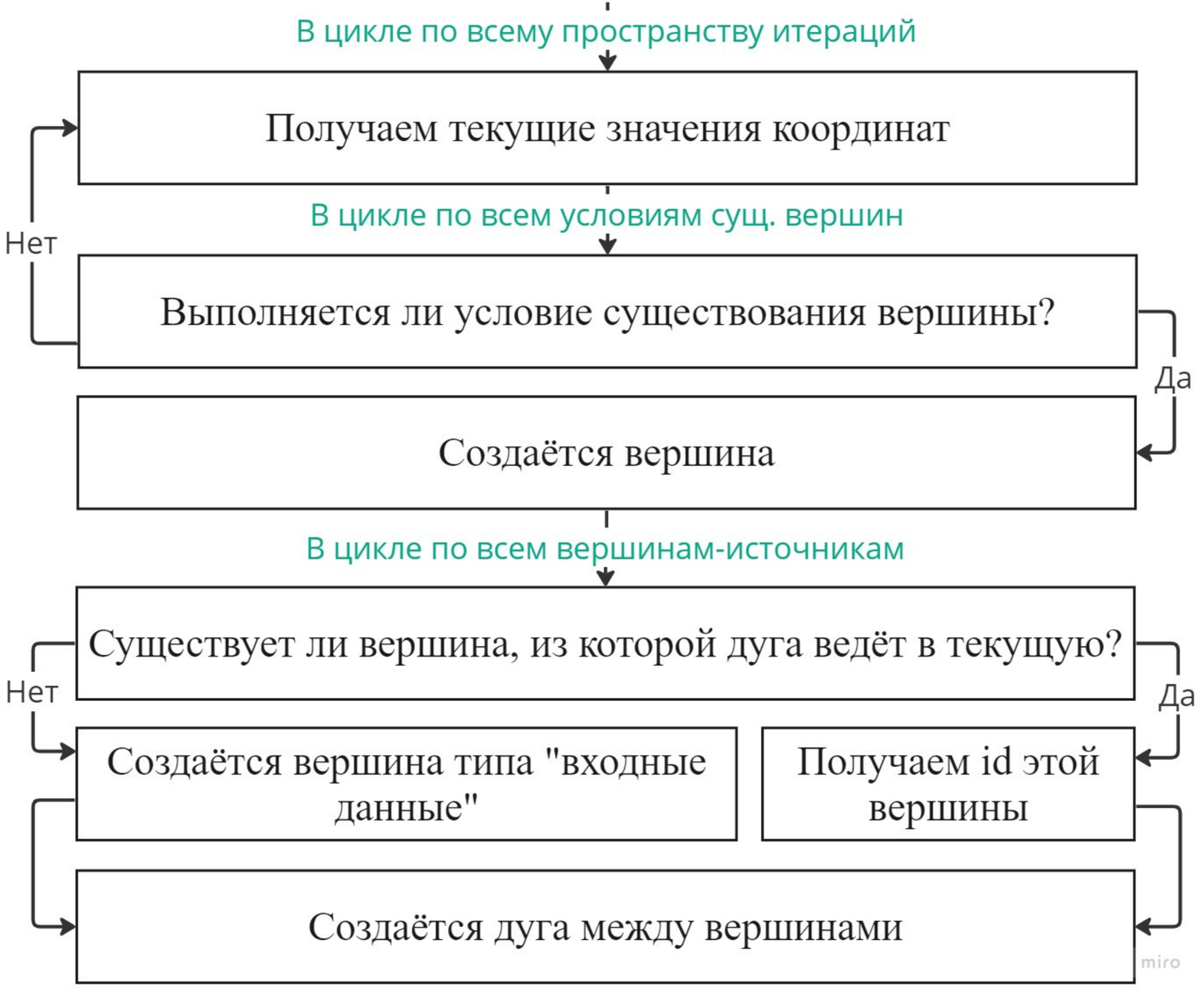


рис. 7 Блок-схема работы алгоритма построения описания модели графа

Принцип расположения вершин и дуг графа

```
<arg name="i" val="1..3"></arg>
<arg name="j" val="1..4"></arg>
<arg name="k" val="1..5"></arg>
```

рис. 8 Фрагмент описания графа на языке Algolang: пространство итераций

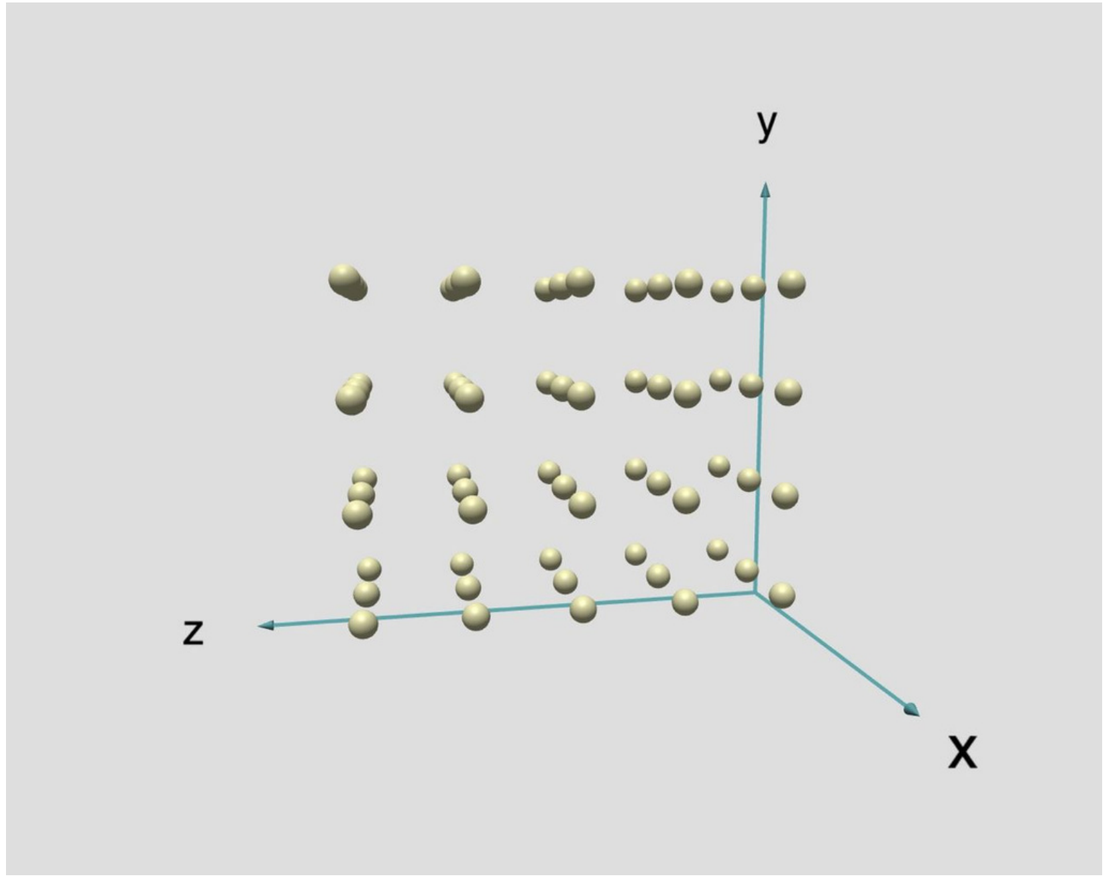


рис. 9 Сетка пространства итераций в трёхмерной декартовой системе кординат

Выходные данные


```
{
        "vertices": [
                { "id": 0, "coordinates": [0, 0, 0], "type": "0" },
                { "id": 1, "coordinates": [1, 0, 0], "type": "1" },
                { "id": 2, "coordinates": [2, 0, 0], "type": "1" }
        ],
        "edges": [
                { "id": 0, "sourceVertexId": 0, "targetVertexId": 1, "type": "0" },
                { "id": 1, "sourceVertexId": 1, "targetVertexId": 2, "type": "0" }
        ]
}
```


рис. 10 Пример выходных данных

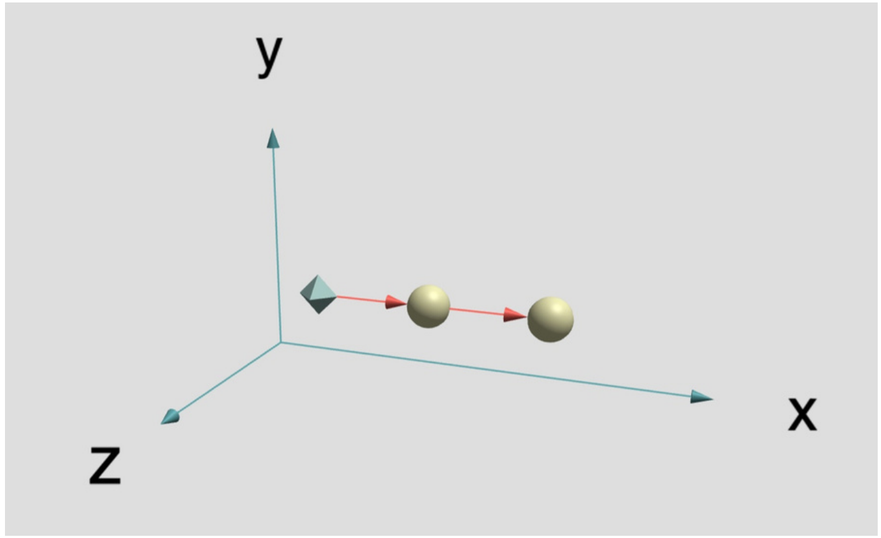
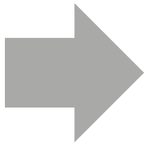


рис. 10 Визуализация примера выходных данных

Примеры результата работы системы визуализации

```
int n = 5;
int m = 4;

for(int i = 1; i <= m; i++)
  for(int j = 1; j <= n + 1; j++)
    vec_out[i] += matrix[i][j] * vec _in[j];
```



```
<algo>
        <params>
                <param name="n" type="int" value="5"></param>
                <param name="m" type="int" value="4"></param>
        </params>

        <block dims="2">
                <arg name="i" val="1..m"></arg>
                <arg name="j" val="1..n+1"></arg>
                <vertex condition="j>1" type="2">
                        <in src="i,j-1"></in>
                </vertex>
        </block>
</algo>
```

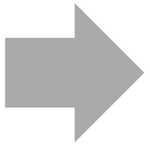


рис. 11 Цепочка преобразований алгоритма умножения матрицы на вектор
(алгоритм на языке С -> визуализация графа алгоритма)

Примеры результата работы системы визуализации


```
{
    "vertices": [
        { "id": 0, "coordinates": [1, 0, 0], "type": "1" },
        { "id": 1, "coordinates": [0, 0, 0], "type": "0" },
        { "id": 2, "coordinates": [0, 1, 0], "type": "0" },
        { "id": 3, "coordinates": [1, 1, 0], "type": "1" },
        { "id": 4, "coordinates": [0, 2, 0], "type": "0" },
        { "id": 5, "coordinates": [0, 3, 0], "type": "0" },
        { "id": 6, "coordinates": [1, 2, 0], "type": "1" },
        { "id": 7, "coordinates": [0, 4, 0], "type": "0" },
        { "id": 8, "coordinates": [0, 5, 0], "type": "0" },
        { "id": 9, "coordinates": [1, 3, 0], "type": "1" },
        { "id": 10, "coordinates": [0, 6, 0], "type": "0" },
        { "id": 11, "coordinates": [0, 7, 0], "type": "0" },
        { "id": 12, "coordinates": [2, 0, 0], "type": "1" },
        { "id": 13, "coordinates": [2, 1, 0], "type": "1" },
        { "id": 14, "coordinates": [3, 0, 0], "type": "1" }
    ],
    "edges": [
        { "id": 0, "sourceVertexId": 1, "targetVertexId": 0, "type": "0" },
        { "id": 1, "sourceVertexId": 2, "targetVertexId": 0, "type": "0" },
        { "id": 2, "sourceVertexId": 4, "targetVertexId": 3, "type": "0" },
        { "id": 3, "sourceVertexId": 5, "targetVertexId": 3, "type": "0" },
        { "id": 4, "sourceVertexId": 7, "targetVertexId": 6, "type": "0" },
        { "id": 5, "sourceVertexId": 8, "targetVertexId": 6, "type": "0" },
        { "id": 6, "sourceVertexId": 10, "targetVertexId": 9, "type": "0" },
        { "id": 7, "sourceVertexId": 11, "targetVertexId": 9, "type": "0" },
        { "id": 8, "sourceVertexId": 0, "targetVertexId": 12, "type": "0" },
        { "id": 9, "sourceVertexId": 3, "targetVertexId": 12, "type": "0" },
        { "id": 10, "sourceVertexId": 6, "targetVertexId": 13, "type": "0" },
        { "id": 11, "sourceVertexId": 9, "targetVertexId": 13, "type": "0" },
        { "id": 12, "sourceVertexId": 12, "targetVertexId": 14, "type": "0" },
        { "id": 13, "sourceVertexId": 13, "targetVertexId": 14, "type": "0" }
    ]
}
```

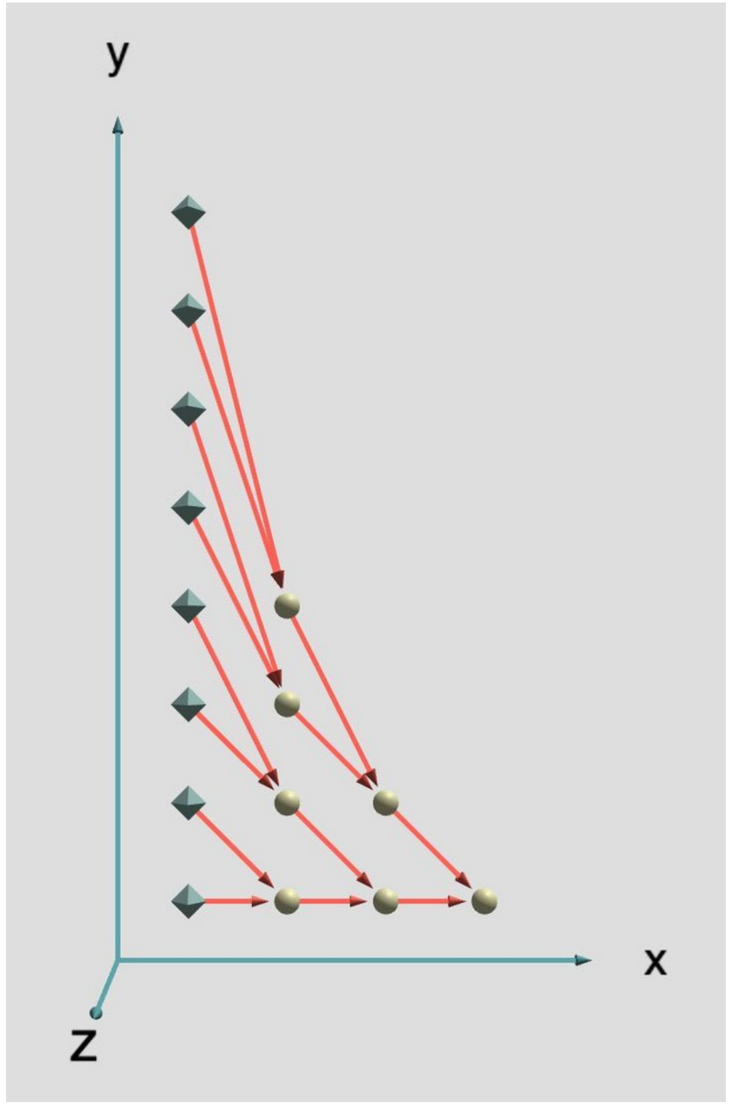



рис. 11 Цепочка преобразований алгоритма умножения матрицы на вектор
(алгоритм на языке C -> визуализация графа алгоритма)

Примеры результата работы системы визуализации

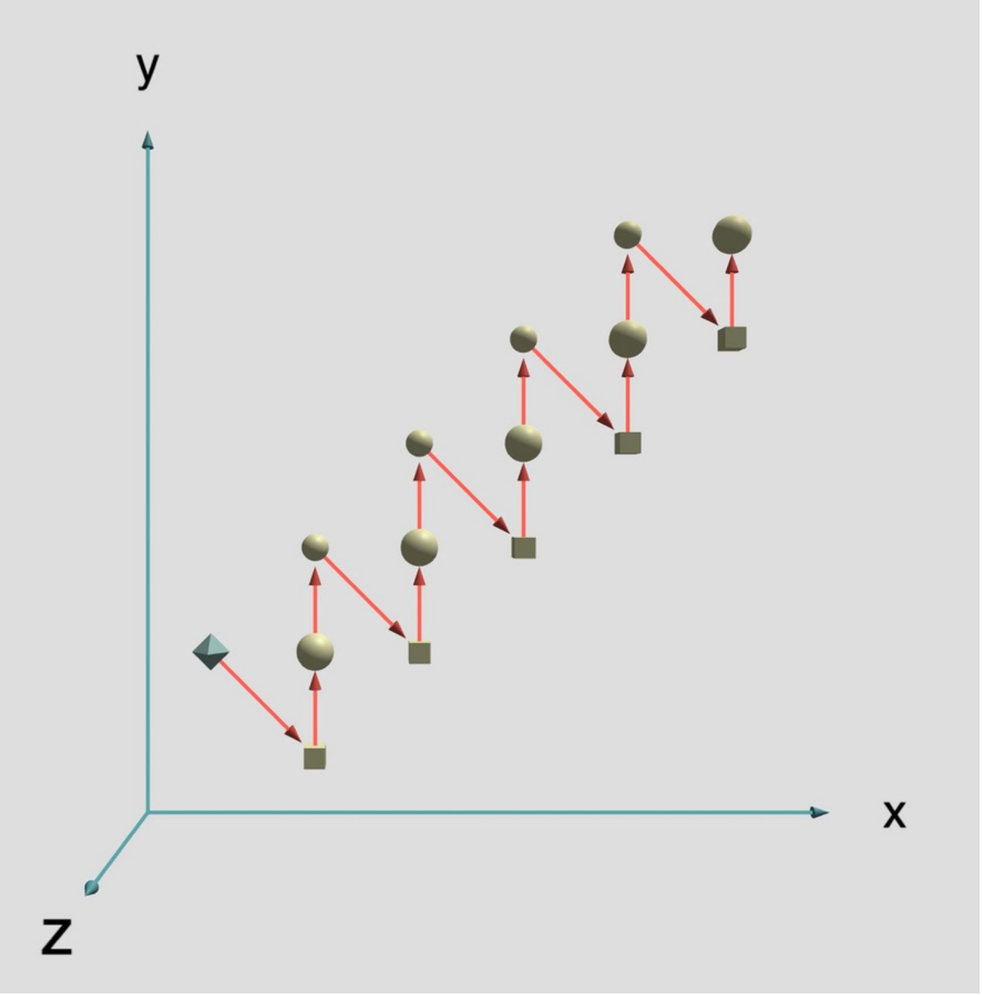


рис. 12 Компактная схема метода Гаусса для трёхдиагональной матрицы

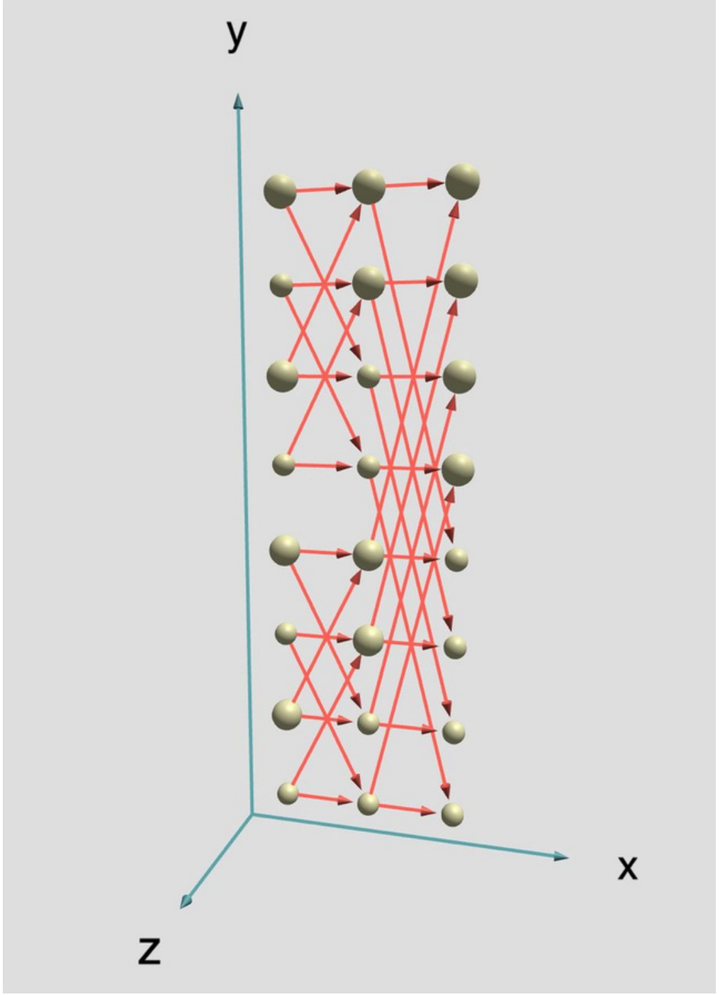
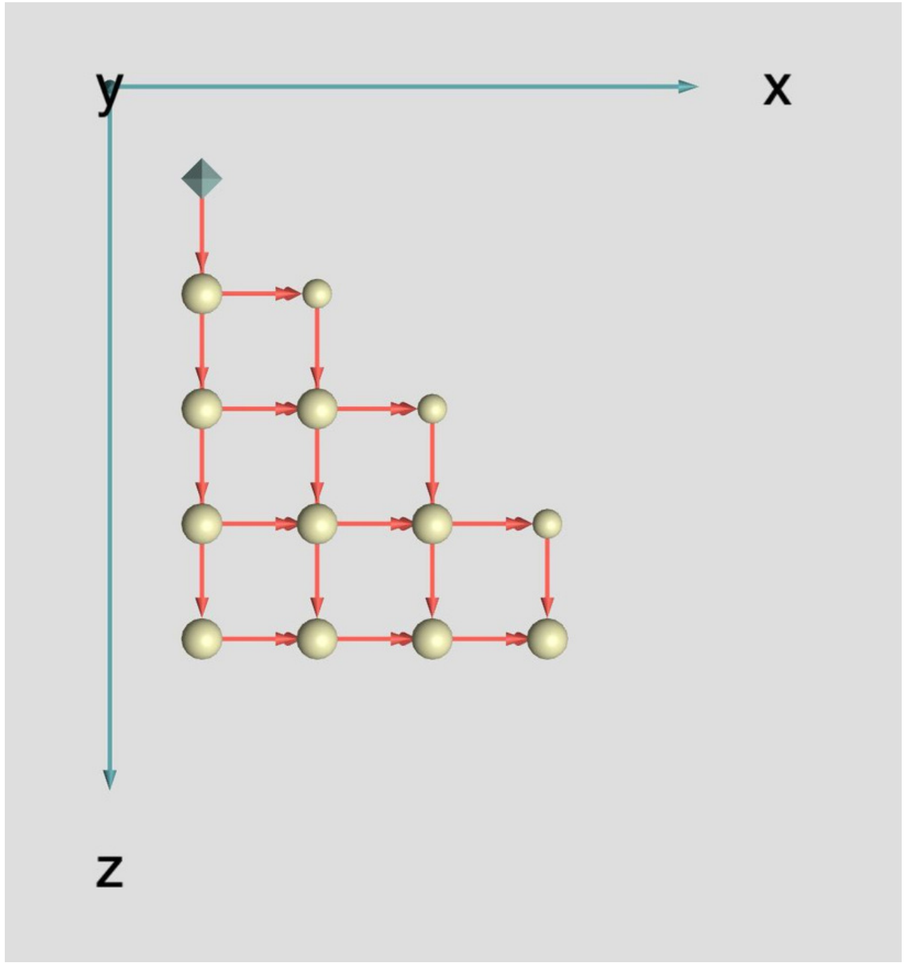


рис. 13 Простой алгоритм Кули-Тьюки быстрого преобразования Фурье для степеней двойки

Примеры результата работы системы визуализации



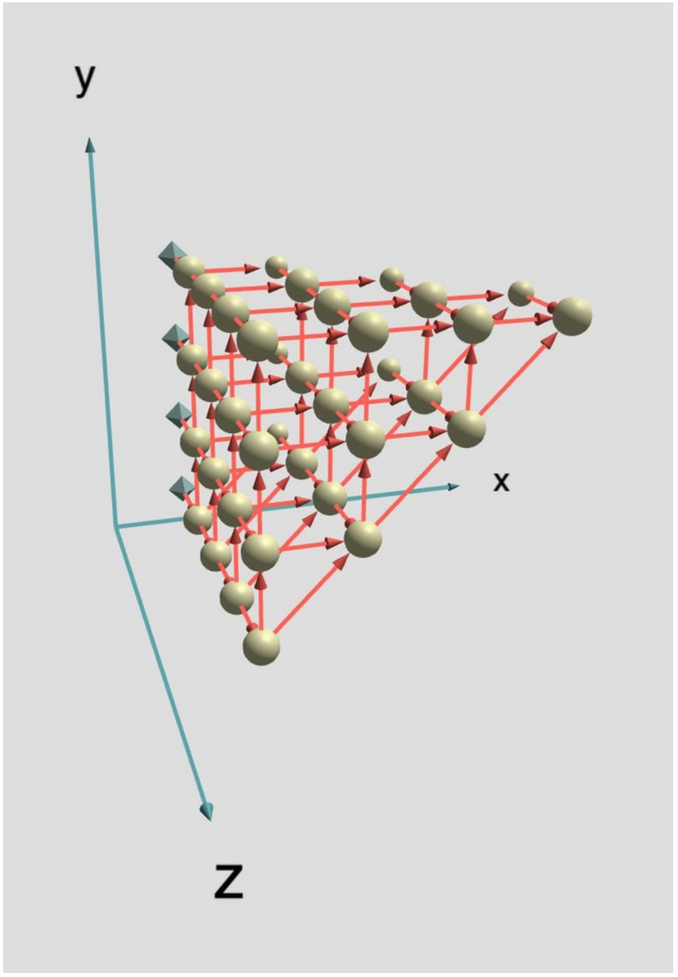
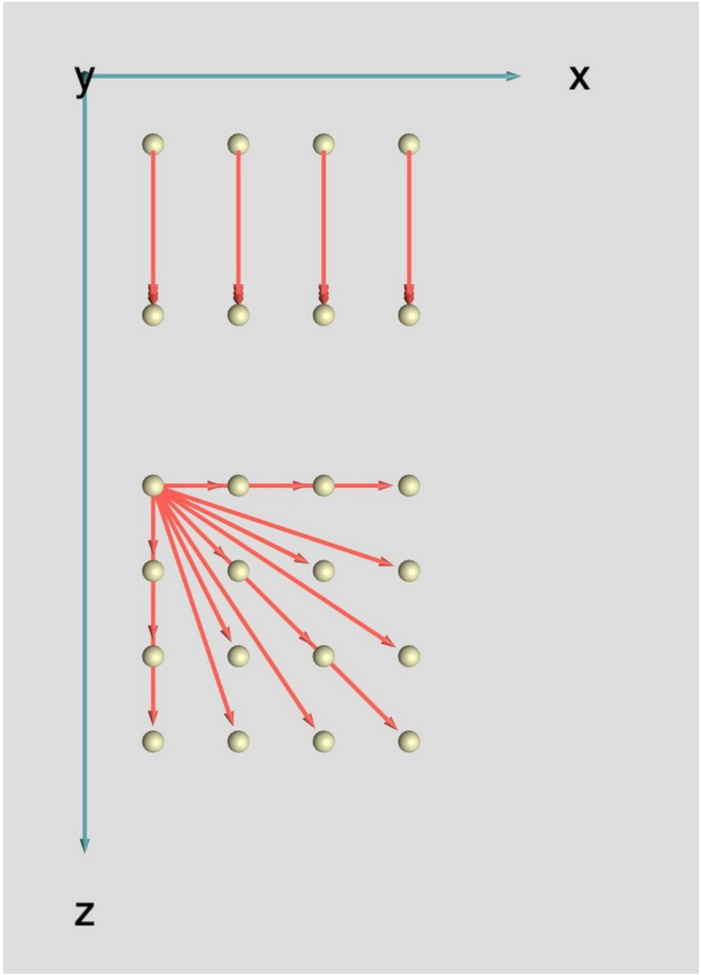


рис. 14 Метод Гивенса (вращений) QR-разложения квадратной матрицы

Примеры результата работы системы визуализации



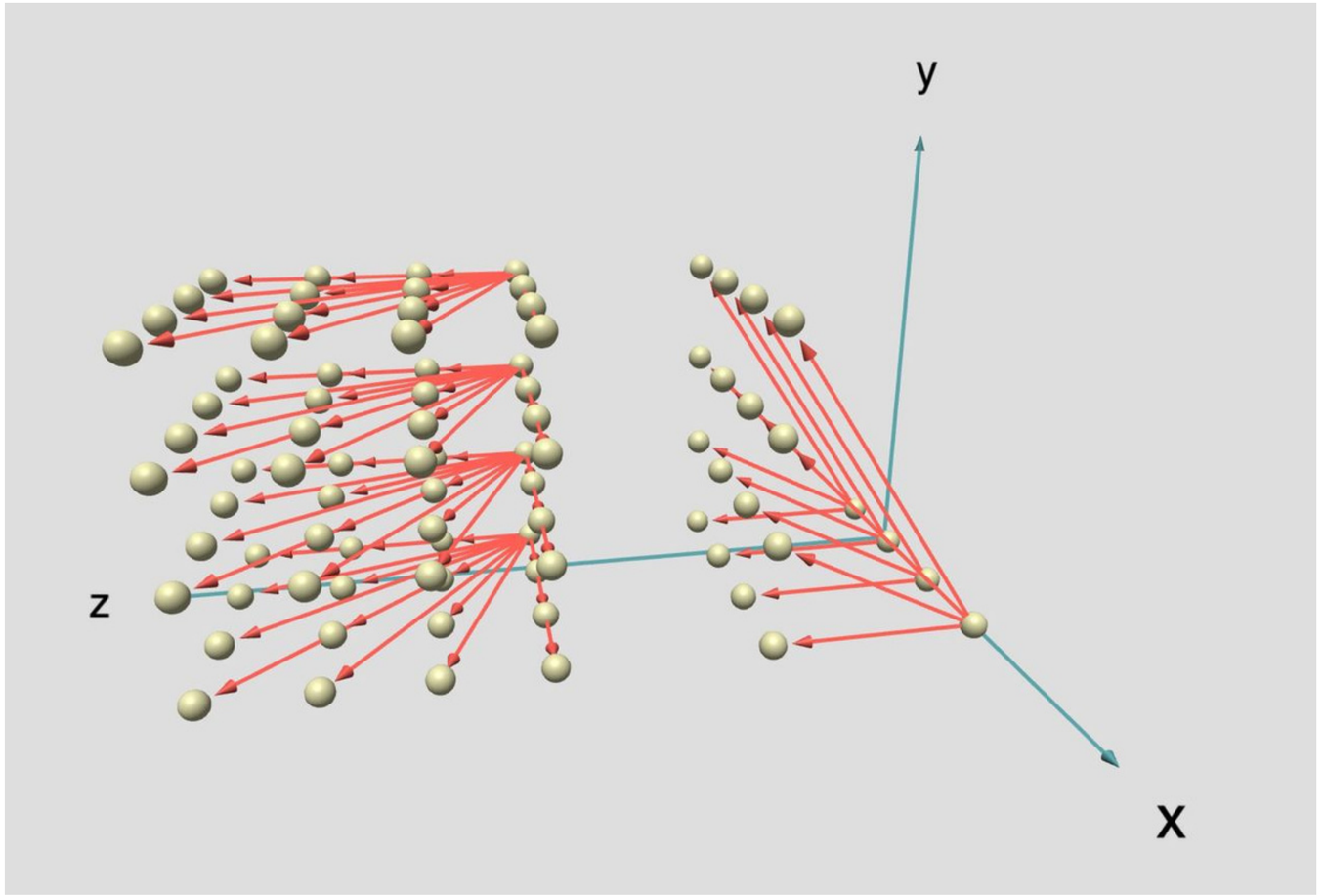


рис. 15 Пример визуализации нескольких блоков разной размерности
(в программе на языке C нескольких циклов разной вложенности)

Результаты работы

1. Реализован сбор информации о графе алгоритма по файлам, написанных на языке Algolang, с использованием изученных библиотек для работы с форматом XML
2. Реализована обработка информации о графе алгоритма с использованием изученных библиотек для работы с регулярными выражениями
3. Реализован алгоритм, который на основе собранных данных из описания графа на языке Algolang, строит описание многомерной модели графа, которое включает в себя расположение вершин и дуг определённым образом и описание их свойств. Данное описание модели может быть использовано второй частью общей системы визуализации информационных графов алгоритмов для создания итогового интерактивного изображения

Расширенный линейный класс задач

- в программе может использоваться любое число простых переменных и переменных с индексами;
- единственным типом исполнительного оператора может быть оператор присваивания, правая часть которого есть арифметическое выражение; допускается любое число таких операторов;
- все повторяющиеся операции описываются только с помощью циклов DO; структура вложенности циклов может быть произвольной; шаги изменения параметров циклов всегда равны +1; если у цикла нижняя граница больше верхней, то цикл не выполняется;
- допускается использование любого числа условных и безусловных операторов перехода, передающих управление "вниз" по тексту; не допускается использование побочных выходов из циклов;
- все индексные выражения переменных, границы изменения параметров циклов и условия передачи управления задаются, в общем случае, неоднородными формами, линейными как по параметрам циклов, так и по внешним переменным программы; все коэффициенты линейных форм являются целыми числами;
- внешние переменные программы всегда целочисленные, и вектора их значений принадлежат некоторым целочисленным многогранникам; конкретные значения внешних переменных известны только перед началом работы программы и *неизвестны* в момент ее исследования.

рис. 16 Определение линейного класса программ

Линейный класс может быть существенно расширен снятием ограничения линейности на перечисленные выражения

Язык AlgoLang и правила написания на языке

<Algorithm> ::== <*algo*> <Parameters> { <Block> } </*algo*>
<Parameters> ::== <*params*> { <Parameter> } </*params*>
<Parameter> ::== <*param name* = “<Name>” *type* = “<Type> [value = “<Number>”]></*param*>
<Type> ::== *int* | *float*
<Block> ::== <*block* [*id* = “<Number>“] *dims* = “<Number>”>{Argument}{Vertex}</*block*>
<Argument> ::== <*arg name* = “<Name>” *val* = “<RegExp>..<RegExp>”></*arg*>
<Vertex> ::== <*vertex condition* = “<RegExp>” *type* = “<Number>”>{Source}</*vertex*>
<Source> ::== <*in* [b*src* = “<Number>“] *src* = “<RegExp>{,<RegExp>}”></*in*>

рис. 17 Формальное описание синтаксиса языка Algolang в форме БНФ

Ярусно-параллельная форма графа

Ярусно-параллельная форма - это представление графа алгоритма, в котором:

- все вершины разбиты на перенумерованные подмножества ярусов;
- начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины;
- между вершинами, расположенными на одном ярусе, не может быть дуг.

Для ЯПФ графа алгоритма важным является тот факт, что операции, которым соответствуют вершины одного яруса, не зависят друг от друга (не находятся в отношении связи), и поэтому заведомо существует параллельная реализация алгоритма, в которой они могут быть выполнены параллельно на разных устройствах вычислительной системы.