DOI 10.37882/2223–2966.2023.02–2.19

ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЙ ЯЗЫК ДЛЯ РАБОТЫ С ГРАФАМИ

PROBLEM-ORIENTED LANGUAGE FOR WORKING WITH GRAPHS

S. Koryagin
D. Popova

Summary. An overview of existing tools for working with graphs. The implementation of a problem-oriented programming language is proposed, which in the future may become a Python library. The possibility of flexible expansion of functionality is provided.

Keywords: Problem-oriented language, formal description, graphs, graph algorithms, Python library.

Корягин Сергей Викторович
К.т.н., Российский Технологический Университет МИРЭА
dongenealog2003@mail.ru
Попова Дарья Леонидовна
Российский Технологический Университет МИРЭА
pdl13@yandex.ru

Аннотация. Обзор существующих инструментов для работы с графами. Предложена реализация проблемно-ориентированного языка программирования, который в перспективе может стать библиотекой Python. Предусмотрена возможность гибкого расширения функционала.

Ключевые слова: Проблемно-ориентированный язык, формальное описание, графы, алгоритмы на графах, библиотека Python.

Часто в работе для решения задачи необходимо использование базовых алгоритмов на графах. Чтобы быстрее и проще было разобраться в том или ином алгоритме лучше всего его визуализировать. Посмотреть, как он работает для различных наборов данных, при небольших изменениях структуры. Для визуализации графов существует достаточно большое количество инструментов.

Наиболее известными являются Graph Online, Graph Editor, Progr@m4you. На рисунке 1 представлены интерфейсы вышеназванных инструментов.

Благодаря таким инструментам становится проще понять, как работал алгоритм. Чаще всего встречаются базовые алгоритмы для работы с графами, такие как: поиск в глубину (DFS), поиск в ширину (BFS), поиск кратчайшего пути, обнаружение циклов, минимальное остовное дерево, раскраска графов, максимальный поток, топологическая сортировка.

В процессе решения задачи граф может видоизменяться — удаление/добавление вершин/рёбер. Такие изменения затруднительно вносить в графических редакторах, особенно при работе с большим набором элементов графа.

Для понимания как будет работать алгоритм при небольшом изменении графа требуется автоматизация изменения его структуры, вместе с возможностью ещё раз запустить необходимый алгоритм. Для решения этой задачи можно разработать проблемно-ориентированный язык.

Проблемно-ориентированный язык

Проблемно-ориентированные (или предметно-ориентированные) языки программирования (англ. DSL — Domain Specific Language) подразумевают, что в языке наряду с универсальными управляющими конструкциями и типами данных присутствуют встроенные средства для описания понятий, характерных для конкретной предметной области, для решения задач которой предназначается данный язык. Предметно-ориентированные языки программирования используются в различных сферах — атомной энергетике, космических исследованиях, радиотехнике и пр. [1, 2, 3]. В данном случае рассмотрим проблемно-ориентированный язык, разработанный для работы в области теории графов [4] .

Первым этапом является описание синтаксиса будущего языка, для этого воспользуемся формой Бэкуса–Наура (далее БНФ).

Формальное описание проблемно-ориентированного языка будет иметь вид:

1. *Язык = «begin» исх.граф </действие.действие/> вычисление вывод «end»*
2. *Исх.граф = «{[« вершина.вершина «] » </ребро.ребро/> «}»*
3. *Действие = [addV! delV] « («вершина»)»! [addE! delE] ребро*
4. *Вычисление = «path» ребро*
5. *Вывод = «print (« [«adjacency»! «list»] «)»*
6. *Ребро = «(«вершина «,» вершина «)»*
7. *Вершина = целое*

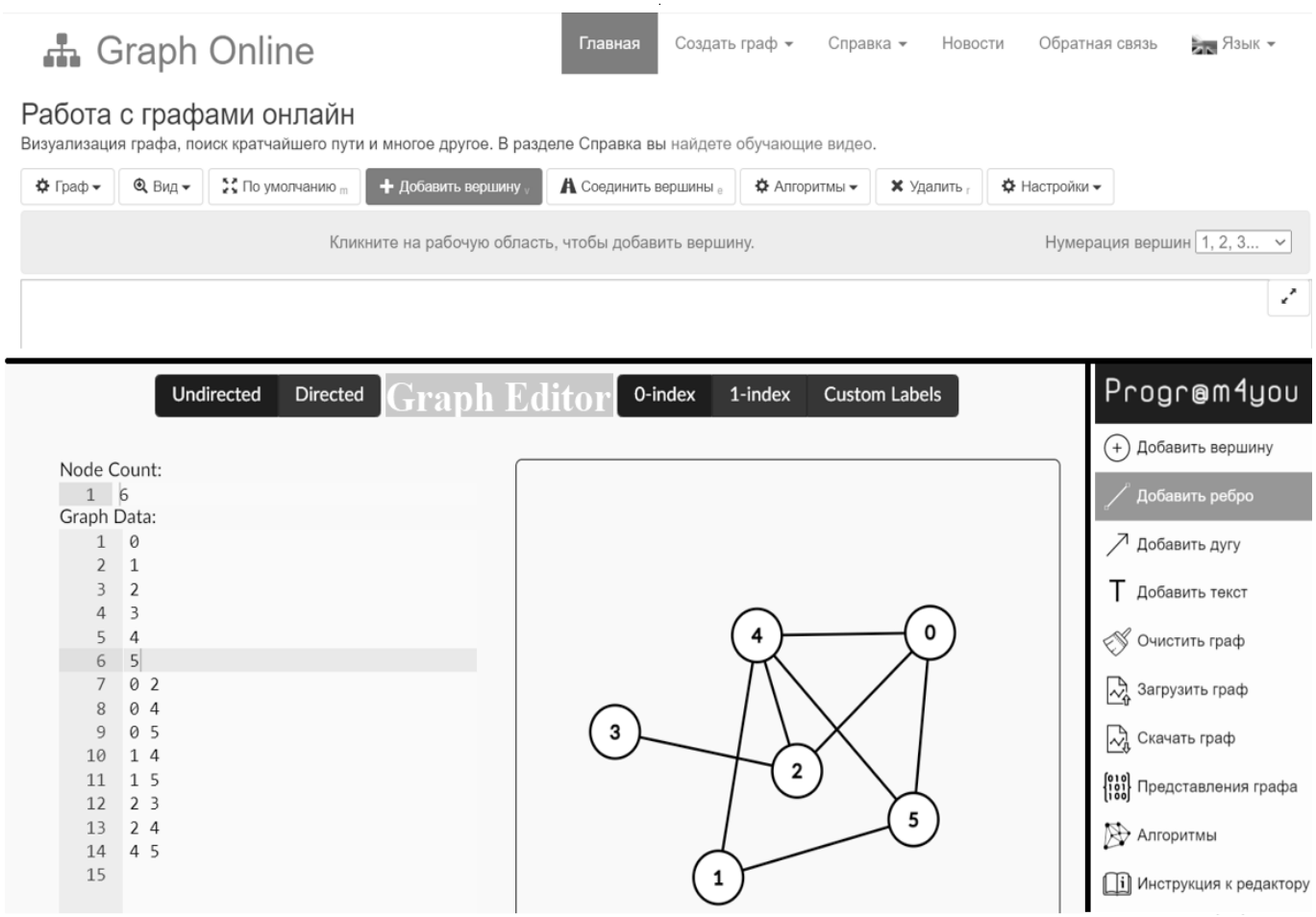

Рис. 1. Интерфейсы инструментов для работы с графами

8. *Целое = циф.циф*
9. *Цифра = “0”! “1”! “2”!...! “9”*

Для корректной работы проблемно-ориентированного языка было разработано транслирующее средство [5–7], которое использует диагностический аппарат для поиска синтаксических (неправильно написанные терминалы языка) и логических (удаление несуществующего элемента, добавление уже существующего элемента) ошибок.

Описание разработанного программного обеспечения

Для программной реализации проблемно-ориентированного языка [8] использовался высокоуровневый язык программирования Python.

Рассмотрим, как выглядит интерфейс, он представлен на рисунке 2.

Интерфейс содержит 5 основных полей:

1. Поле «БНФ» недоступно для редактирования пользователю, в нём описана структура БНФ, которая была рассмотрена выше, в соответствии с этой структурой пишется программа на проблемно-ориентированном языке.
2. Поле «Ваш код» доступно для редактирования пользователю, в нём пишется код.
3. Поле «Представление графа» недоступно для редактирования пользователю, необходимо для визуального отображения графа
4. Поле «Вывод программы» недоступно для редактирования пользователю, необходимо для вывода результата работы программы
5. Поле «Текущее состояние графа» недоступно для редактирования пользователю, необходимо для контроля состояния графа, особенно полезно при возникновении логических ошибок

Так же есть две кнопки:

1. Кнопка «Удалить лишние пробелы» очищает поле от лишних пробелов, заменяя два и более пробела одним

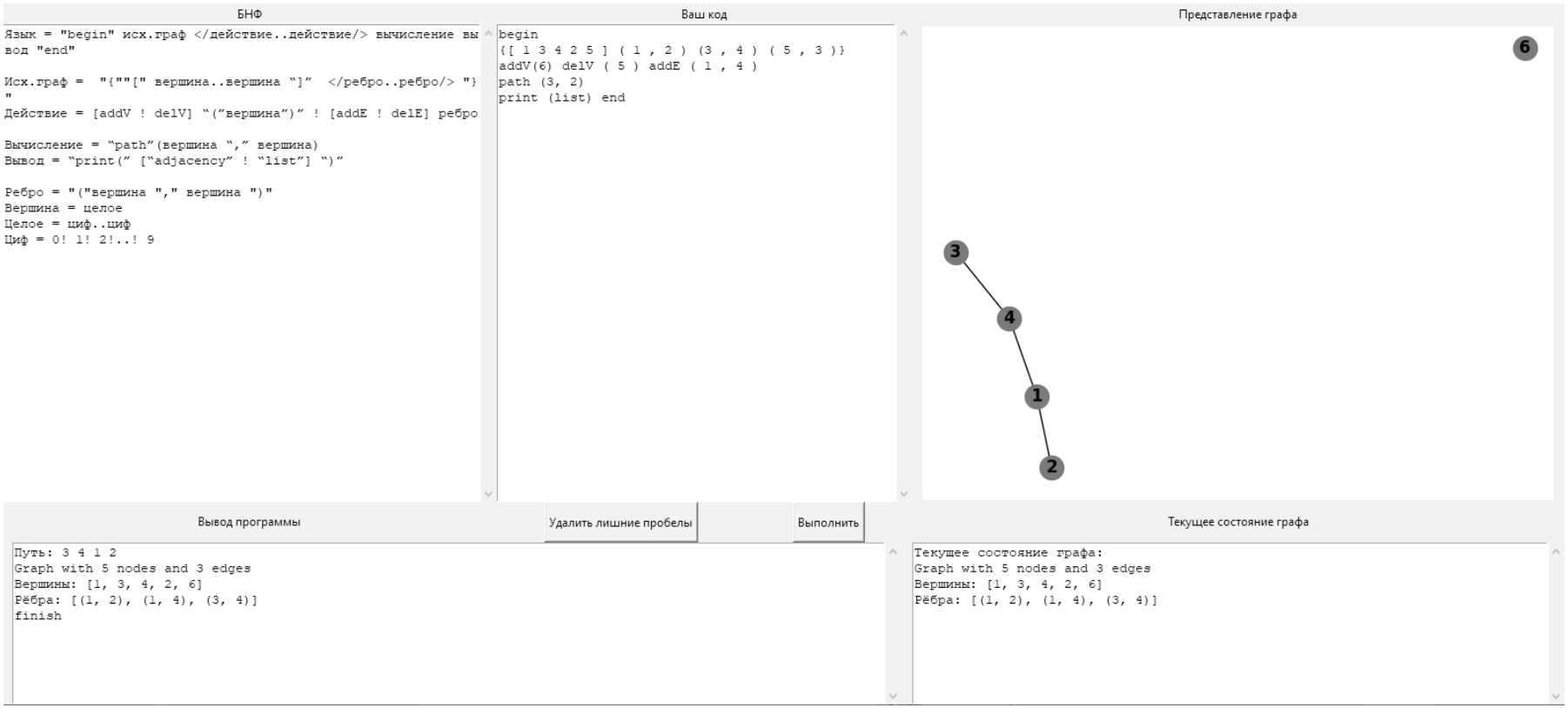

Рис. 2. Вывод программы после выполнения кода

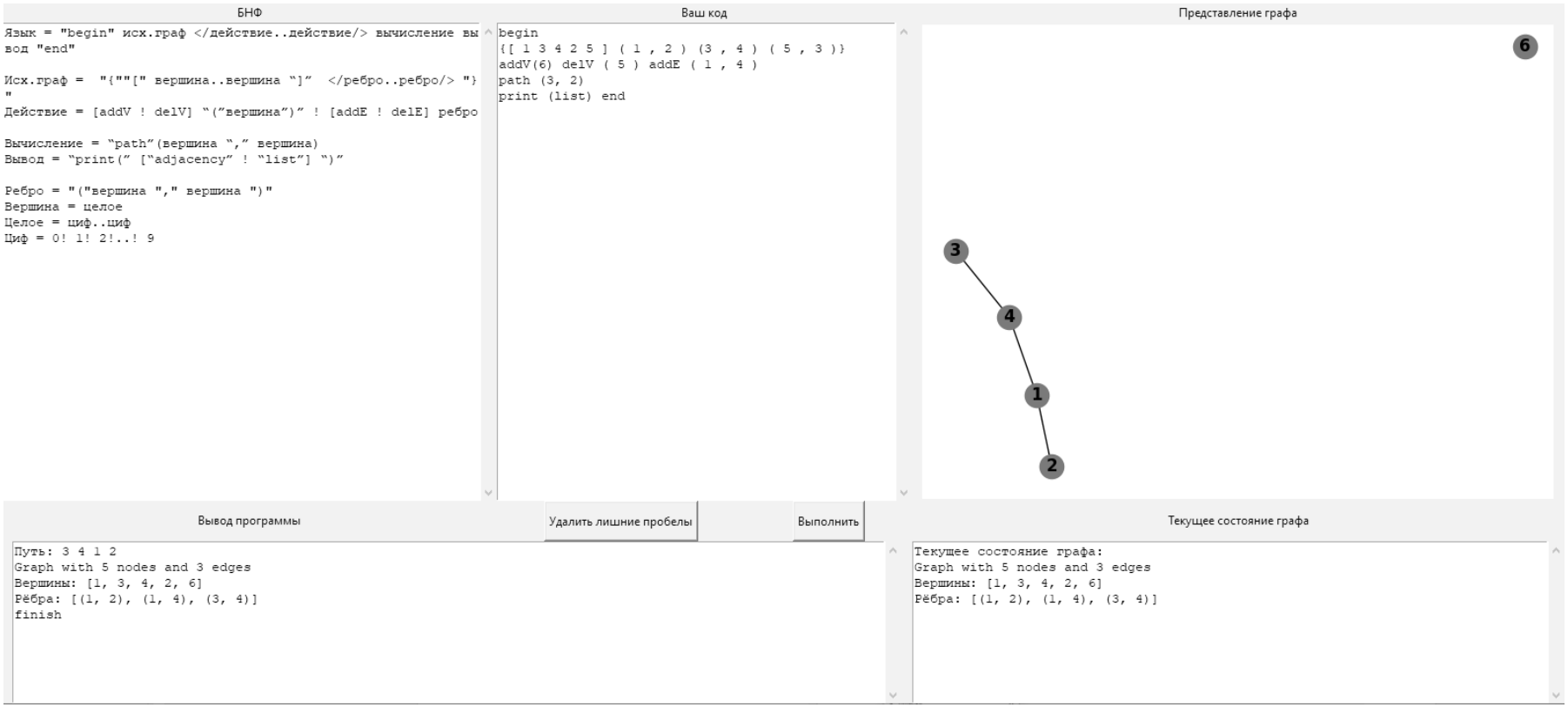

Рис. 3. Вывод программы после выполнения кода

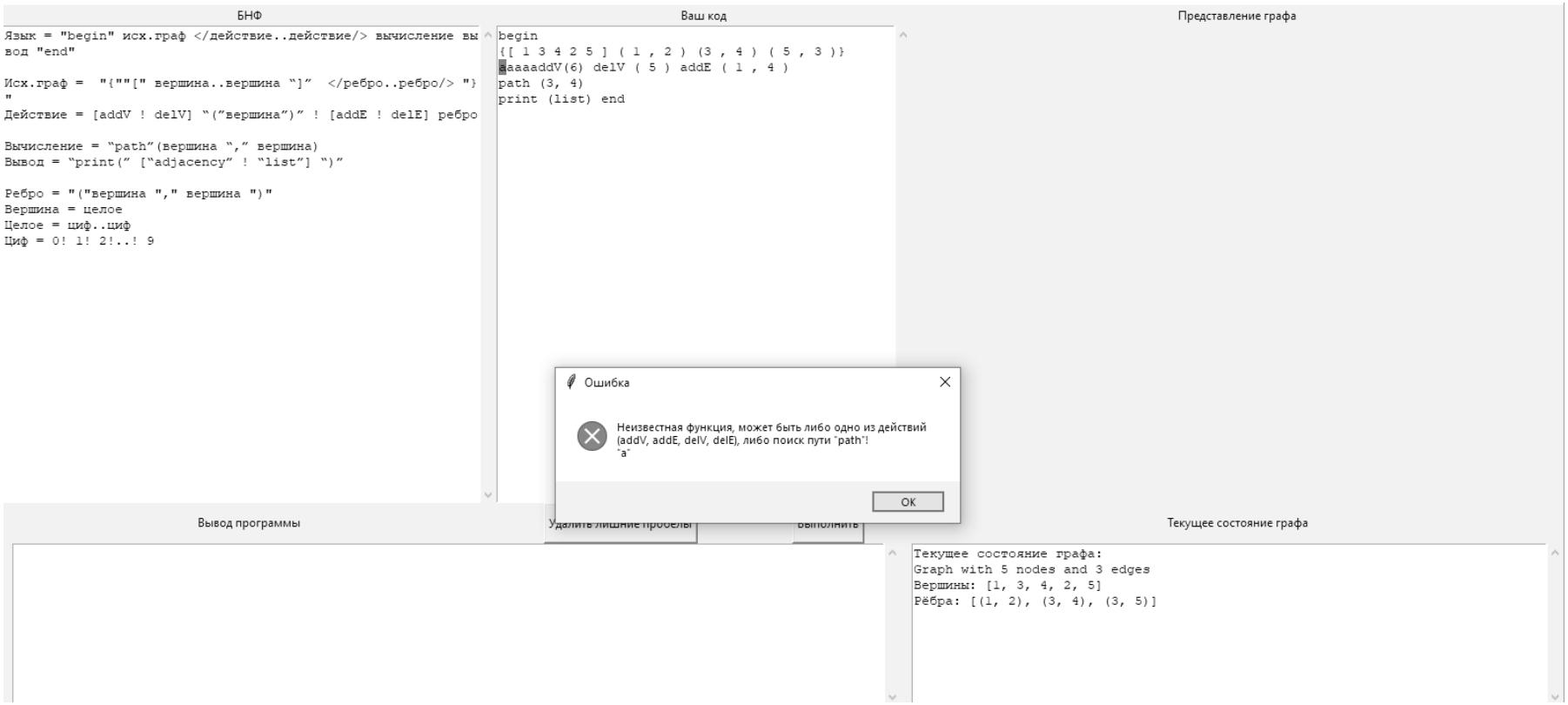

Рис. 4. Вывод программы при синтаксической ошибке

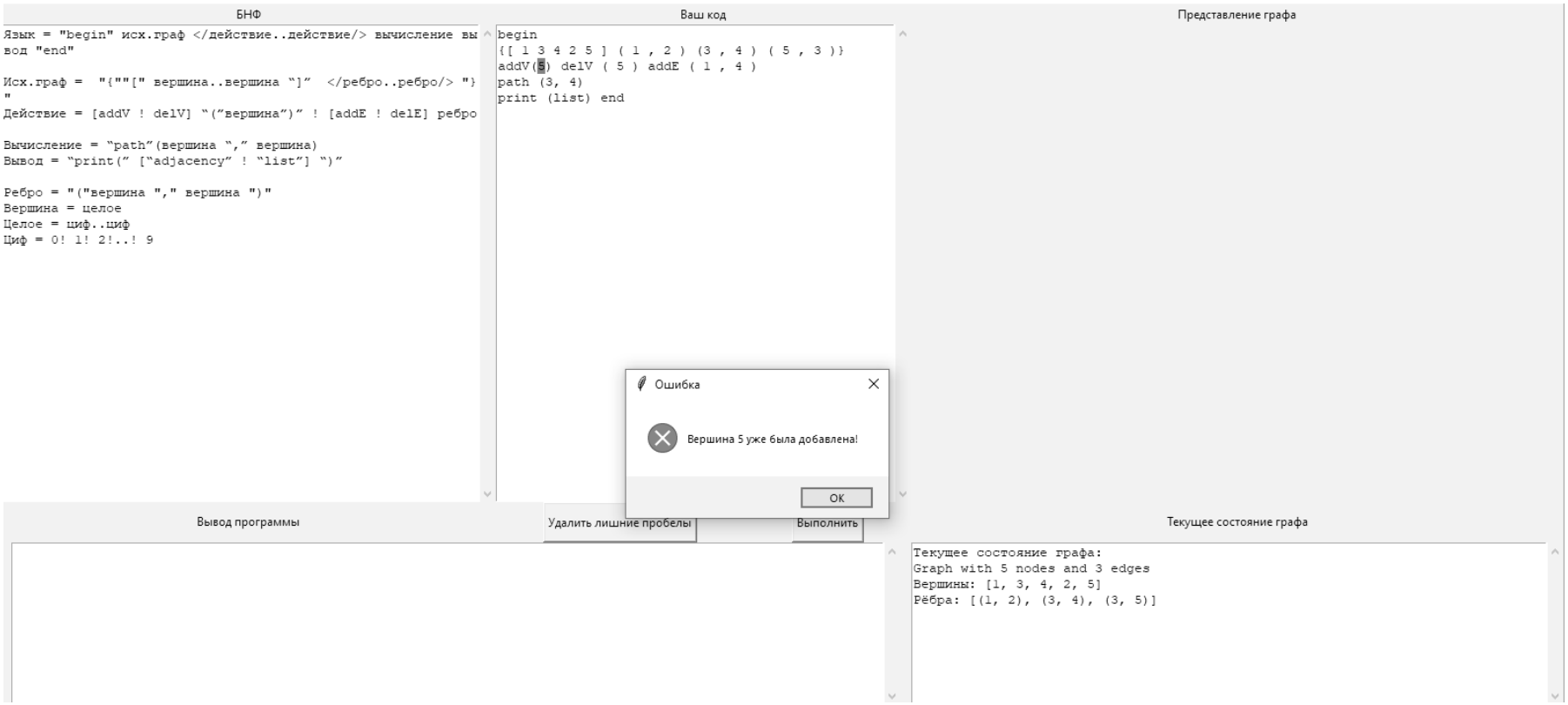

Рис. 5. Вывод программы при логической ошибке

2. Кнопка «Выполнить» начинает процесс выполнения кода

В первой версии поддерживаются следующие действия:
1. задание параметров исходного состояния графа: список вершин и список рёбер;
2. изменение структуры графа: добавление вершин, удаление вершин, добавление рёбер, удаление рёбер;
3. вычисление кратчайшего пути для пары вершин;
4. вывод графа: виде матрицы смежности, списка рёбер.

Пример работы программы

Рассмотрим три основных случая:
1. Для вернонаписанного кода программа выведет результат работы.
2. При неверном написании ключевых слов (синтаксическая ошибка) будет выведено сообщение об ошибке, так же подсветится место, на которое необходимо обратить внимание.
3. При синтаксически вернонаписанной программе могут возникать логические ошибки, информация о них так же выводится в всплывающем окне. В случае логической ошибки полезным будет использование информации из поля «Текущее состояние графа», оно поможет быстрее исправить ошибку.

Пример работы вернонаписанной программы представлен на рисунке 3.

Примеры срабатывания диагностического аппарата представлен на рисунках 4–5.

Возможности расширения функционала

В данной реализации граф неориентированный и невзвешенный, для добавления ориентации и веса графа необходимо поменять описание ребра, а также добавить определение новых нетерминалов — «вес» и «ориентация»:

Ребро = «(«вершина ориентация вершина </»,» вес/>»)»

Ориентация = «,»! «<-»! «->»! «<->»

Вес = </»-»/> целое

Теперь к разделителю «,», который означает что ребро неориентированно, в описании ребра добавились новые значения:

«<-» — однонаправленное ребро от правой вершины к левой вершине

«->» — однонаправленное ребро от левой вершины к правой вершине

«<->» — двунаправленное ребра

В описании веса перед целым значением может быть «-», так как вес бывает отрицательным.

Так как теперь есть веса и ориентация может понадобиться редактирование этих значений, введём новые действия:

Действие = [addV! delV] « («вершина»)»! [addE! delE] ребро! editV “(“вершина”)”! editE ребро

Что бы можно было в любом порядке совершать действия над структурой графа, делать вычисления и выводить результат необходимо ввести новый нетерминал и изменить описание нетерминала “Язык”:

Язык = “begin” исх.граф операция.операция “end”

Операция = действие! вычисление! вывод

Ввести дополнительные алгоритмы:

Вычисление = “path” ребро! “pathBFS” ребро! “pathDFS” ребро! “dijkstra” вершина! …

Вершины могут быть не только целочисленными, поэтому заменим определение нетерминала:

Вершина = буква </символ.символ/>! целое

Символ = буква! цифра

Буква = “A”! “B”! …! “Z”! “a”! “b”! …! “z”

После всех изменений структура языка будет выглядеть следующим образом:
1. *Язык = «begin» исх.граф операция.операция «end»*
2. *Операция = действие! вычисление! вывод*
3. *Исх.граф = «{[« вершина.вершина «] » </ребро.ребро/> «}»*
4. *Действие = [addV! delV] « («вершина»)»! [addE! delE] ребро! editV «(«вершина»)»! editE ребро*
5. *Вычисление = “path” ребро! «pathBFS» ребро! «pathDFS» ребро! «dijkstra» вершина! …*
6. *Вывод = «print (« [«adjacency»! «list»] «)»*

7. *Ребро = «(«вершина ориентация вершина </»,» вес/>»)»*
8. *Ориентация = «,»! «<-»! «->»! «<->»*
9. *Вес = </»-»/> целое*
10. *Вершина = буква </символ.символ/>! целое*
11. *Символ = буква! цифра*
12. *Буква = "A"! "B"! ...! "Z"! "a"! "b"! ...! «z»*
13. *Целое = циф.циф*
14. *Цифра = 0! 1! 2!.! 9*

Добавление новых возможностей языка выполняется достаточно просто, что позволяет быстро менять его под нужды пользователей. Разработанная структура БНФ универсальна и может быть реализована на любом языке программирования. Синтаксис разработанного языка схож с Python, что позволяет в перспективе создать библиотеку для языка Python.

Разработанное программное средство может использоваться индивидуально для того, чтобы было проще разобраться с алгоритмами. Подходит для использования в качестве методического материала для освоения дисциплин, в которых изучаются алгоритмы на графах, как наглядное пособие. В данное средство заложены необходимый функционал и возможности его дальнейшего расширения.

ЛИТЕРАТУРА

1. Ильин Д.Ю., Корягин С.В. Проблемно-ориентированный язык для описания модели поведения компьютерного оппонента (бота). Прикаспийский журнал: Управление и высокие технологии. 2015, № 1 (29), с. 193–207.
2. Князев К.А., Корягин С.В., Проблемно-ориентированный язык для описания параметров операционной системы компьютера. Cloud of Science, 2020, т. 6, № 4, с. 683–692.
3. Водчиц А.О., Корягин С.В. Проблемно-ориентированный язык описания музыкальных документов. Межвузовский сборник научных трудов "Задачи системного анализа, управления и обработки информации" Выпуск 6, 2020, с. 5–11.
4. Корягин С.В., Соловьев Р.В. Автоматизация перевода графического описания конечных автоматов в таблицу переходов. Межвузовский сборник научных трудов "Задачи системного анализа, управления и обработки информации" Выпуск 6, 2020, с. 98–102.
5. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технология и инструменты. пер. с англ. — М., 2011.
6. Свердлов С.З. Языки программирования и методы трансляции: учеб. пособие. — СПб.: Питер, 2007.
7. Корягин С.В. Перекодировщик языков непрерывного моделирования // Промышленные АСУ и контроллеры. 2013. № 10. С. 52–56.
8. Гойвертс Я., Левитан С. Регулярные выражения. Сборник рецептов. — Пер. с англ. — СПб.: Символ-Плюс, 2010.

Журнал «Современная наука: актуальные проблемы теории и практики»