УДК 62 **Технические науки**

Путнин Валерий Игоревич, *старший программист, X5RetailGroup*

АВТОМАТИЗАЦИЯ ПРОЦЕССА ПОСТРОЕНИЯ РАСПИСАНИЯ С ПРИМЕНЕНИЕМ ТОПОЛОГИЧЕСКОЙ СОРТИРОВКИ НА ГРАФЕ

Аннотация: В данной статье будет рассмотрена проблема построения расписания из наборов зависимых курсов. Для упрощения понимания задачи обозначим каждый уникальный предмет уникальной цифрой. Так же обозначим зависимость курсов через размещение их в списке по 2 элемента. Зависимостью в данной статье подразумевается необходимость прохождения одного курса для прохождения второго. Реализация решения данной задачи будет выполнена на языке программирования Java. Пример отображения зависимости: [x, y], где для того что бы приступить к курсу x необходимо пройти курс y. Результатом выполнения будет список курсов в порядке от курса без зависимости и курса с максимальным количеством зависимостей.

Ключевые слова: алгоритмы, java, топологическая сортировка, поиск в глубину, поиск на графах автоматизация процессов.

Abstract: In this article, we will consider the problem of constructing a schedule from sets of dependent courses. To simplify the understanding of the problem, we will designate each unique item with a unique number. We also denote the dependence of the courses by placing them in a list of 2 elements. The dependency in this article implies the need to complete one course in order to complete the second. The implementation of the solution to this problem will be performed in the Java programming language. Example of displaying a dependency: [x,y], where in order to start the course x, you need to pass the course y. The result of the execution will be a list of courses in order from the course with no dependency and the course with the maximum number of dependencies.

Keywords: algorithms, java, topological sorting, depth-first search, graph search process automation.

Для усложнения задачи и приближения её к реальным условиям добавим больше курсов и отобразим набор курсов в виде списка: [[m, b], [b,f], [f,c], [c,e], [m,a], [a,g], [g, x], [x,z]]. Для наибольшей наглядности отобразим зависимости в виде графа.

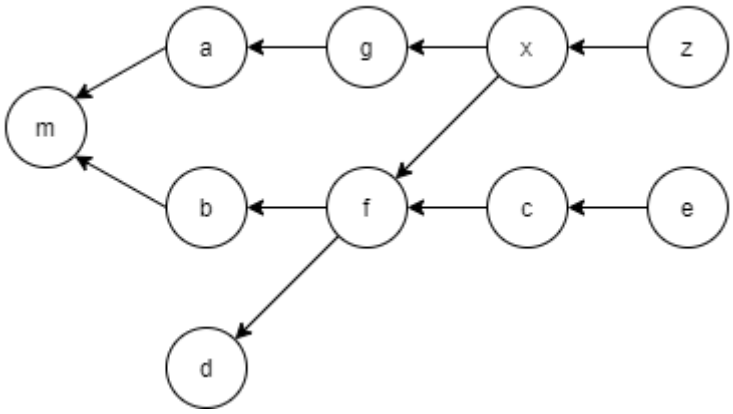

Рис.1 Зависимость к курсов в виде направленного графа

Цель задачи: Обойти граф в заданном направлении и отобразить курсы в последовательном порядке. Пример обхода графа в последовательном направлении выглядит следующим образом: [z, x, e, c, d, f, g, a, b, m] или альтернативным решением будет список вида [z, e, x, c, d, f, g, b, a, m].

Для решения задач подобного вида используется алгоритм топологической сортировки без привязки к конкретной реализации. Для реализации данного алгоритма, будет произведен поиск элементов через поиск в глубину.

Так как список курсов представлен в виде списка списков, и такая структура данных не подходит для решения задачи с помощью топологической сортировки нам необходимо преобразовать набор курсов в такую структуру данных, которая будет представлять граф. Для отображения графа на любом языке программирования используется так называемый «Смежный список» [1]. Создадим смежный список из набора курсов следующего вида:

{m:[a,b]},{a:[g]},{b:[f]},{g:[x]},{x:[z]},{d:[f]},{f:[c]},{c:[e]} где конструкция с фигурными обозначает вершину(курс) и список направленных в эту вершину зависимых вершин(курсов).

Далее опишем конвертацию списка зависимых курсов в «Смежный список» на языке Java:

```
Map<Integer, LinkedList<Integer>> graph = new HashMap<>();
for(int i = 0; i < prerequisites.length; i++) {
    if(graph.containsKey(prerequisites[i][0])) {
        graph.get(prerequisites[i][0]).add(prerequisites[i][1]);
    } else {
        graph.put(prerequisites[i][0], new LinkedList<>());
        graph.get(prerequisites[i][0]).add(prerequisites[i][1]);
    }
}
```

Рис.2 Конвертация списка предопределённого списка курсов в смежный список(граф)

После создания смежного списка мы сможем провести итерацию каждой записи в смежном списке, которая в свою очередь является ключом в карте. При итерации ключей карты будет необходимо производить поиск всех зависимых записей с помощью реализации алгоритма «Поиск в глубину», где поисковым значением будет запись без зависимостей, далее будем возвращаться рекурсивно по стеку вызова, который используется в алгоритме поиска в глубину и размещать полученные записи в результирующий список.

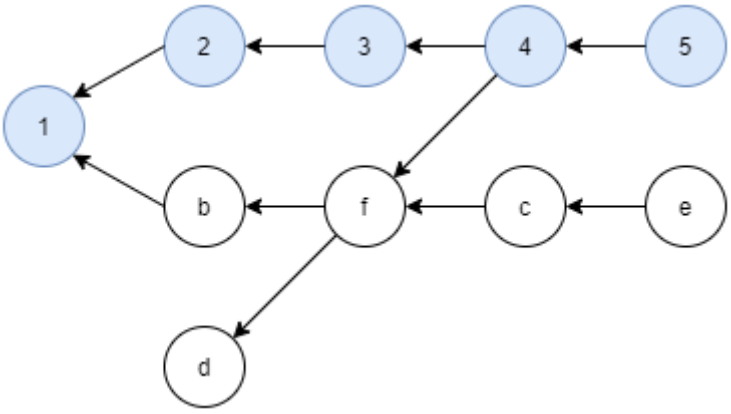

Рис.3 Визуализация обхода графа до конечной записи при первой итерации топологической сортировки

Для выполнения поиска, визуализация которого представлена на Рис.3, необходимо, что бы граф был направленным и не цикличным, так как при наличии цикла в графе, выполнения топологического сортировки будет не возможным и программа не будет иметь точки выхода. Для предотвращения зацикливания программы и создания точки выхода из приложения необходимо создать проверку на наличие цикла в графе [2]. Один из возможных вариантов решения этой проблемы, это создания состояния записи.

Запись будет обладать следующими состояниями: 1- запись не была прочитана, 2 - запись была прочитана и находится в стэке, 3 - запись была прочитана и занесена в результирующий список. Далее визуализируем состояние записей:

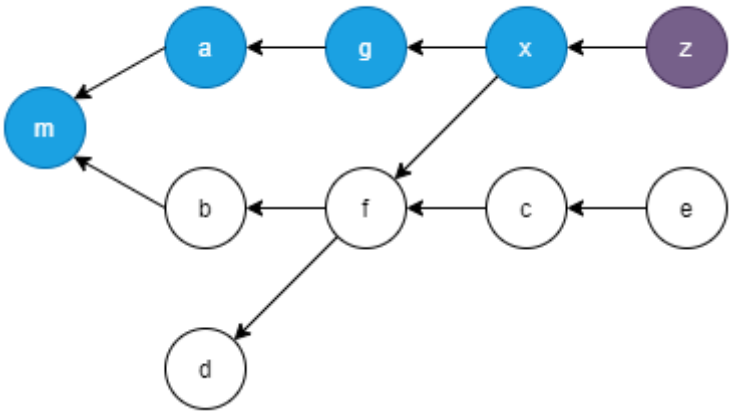

Рис.4 Визуализация состояний записей графа при первой итерации топологической сортировки

После итерации всех записей в графе, список в котором будет хранится результат будет следующего вида: [z, x, e, c, d, f, g, a, b, m], где в крайнем левом положении находятся записи с которых следует начинать приступать к курсам, далее размещаются курсы к которым можно преступить после прохождения предыдущих и крайнее правое положение занимают курсы, которые можно пройти только в последнюю очередь [3].

```
for(int i = 0; i < numCourses; i++) {
    if(!dfs(graph, i, res, statuses)) return new int[0];
}
```

Рис.5 Исполнение топологической сортировки

В данную реализацию топологической сортировки для поиска в глубину включен поиск циклов, при его наличии будет возвращаться пустой список [4].

В качестве итога можно заключить, что данная программа выполняет поставленные перед ней цели, а также ее реализация является наиболее эффективной по сравнению с альтернативными аналогами не построенных на графах.

Библиографический список:

1. Алгоритмы на Java. Автор Джитер Кевин Уэйн, Седжвик Роберт. 2012. 848 с.

2. Алгоритмы. Теория и практическое применение. Автор Стивенс Род. 2016. 546 с.

3. Алгоритмы. Справочник с примерами на C, C++, Java и Python | Хайнеман Джордж, Поллис Гари. 2017. 434 с.

4. Java. Полное руководство | Шилдт Герберт. 2015. 1377 с.