|  |  |
| --- | --- |
| Министерство науки и высшего образования Российской Федерации | |
| федеральное государственное бюджетное образовательное учреждение  высшего образования  «Иркутский государственный университет»  (ФГБОУ ВО «ИГУ») | |
| Институт математики и информационных технологий | |
| Кафедра вычислительной математики и оптимизации | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **КУРСОВАЯ РАБОТА** | |
| по направлению «Прикладная математика и информатика» | |
| профиль «Математические методы и информационные технологии» | |
|  |  |
|  |  |
| Решение задач с графами на языке Haskell | |
|  | |
|  | |
|  | |
|  | |
|  | Студента курса очного отделения |
|  | группы 02321-ДБ |
|  | Подрядчикова Владимира Валерьевича |
|  |  |
|  |  |
|  | Руководитель: |
|  | к. ф.-м. н., доцент |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Спиваков А. П. |
|  |  |
|  |  |
|  | Допущена к защите |
|  | Зав. кафедрой, д. ф.-м. н., профессор |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Высотин А. Д. |
|  |  |
|  |  |
|  |  |

Иркутск - 2022

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc122835830)

[1 НЕМНОГО О ГРАФАХ И ИХ ЗАДАНИИ В HASKELL 4](#_Toc122835831)

[2 ЗАДАЧИ С ГРАФАМИ. ПОСТАНОВКА И РЕАЛИЗАЦИЯ РЕШЕНИЯ 6](#_Toc122835832)

[2.1 Задача 1. 6](#_Toc122835833)

[2.2 Задача 2. 7](#_Toc122835834)

[ЗАКЛЮЧЕНИЕ 9](#_Toc122835835)

[СПИСОК ЛИТЕРАТУРЫ 10](#_Toc122835836)

# ВВЕДЕНИЕ

Haskell – функциональный язык программирования, сильно отличающийся от императивных и смешанных языков разработки. Важной особенность языка является поддержка ленивых вычислений, что позволяет ускорить работу программы. В Haskell аргументы функции вычисляются только тогда, когда действительно требуются для вычисления. Причем ленивые вычисления выполняются без вмешательства самого программиста.

Haskell – актуален. Язык применяется в финансовом секторе – для разработки собственных инструментов в крупных банках и других компаниях. Также Haskell часто используется для обработки текстов, синтаксического анализа, а также веб-разработки.

Целью данной курсовой работы является ознакомление c Haskell’ем посредством решения задач, связанных с графами, укрепление и развитие навыков программирования.

Теория графов в настоящее время является интенсивно развивающимся разделом математики. Это объясняется тем, что в виде графовых моделей описываются многие объекты и ситуации, что очень важно для нормального функционирования общественной жизни. Именно этот фактор определяет актуальность их более подробного изучения. В качестве примеров графов могут выступать чертежи многоугольников, электросхемы, схематичное изображение авиалиний, метро, дорог и т.п. Генеалогическое дерево также является графом, где вершинами служат члены рода, а родственные связи выступают в качестве ребер графа. Теория информации широко использует свойства двоичных деревьев. Процесс размножения бактерий – это одна из разновидностей ветвящихся процессов, встречающихся в биологической теории. Интернет – всемирная система объединенных компьютерных сетей для хранения и передачи информации. Сеть интернет можно представить в виде графа, где вершины графа – это интернет сайты, а ребра – это ссылки (гиперссылки), идущие с одних сайтов на другие.

# НЕМНОГО О ГРАФАХ И ИХ ЗАДАНИИ В HASKELL

Граф – это топологическая модель, состоящая из множества вершин и множества соединяющих их рёбер.

Вершина – точка графа, объект, имеющий некоторое значение.

Ребро – пара двух вершин, связанных друг с другом. Рёбрам может быть присвоен вес и направление.

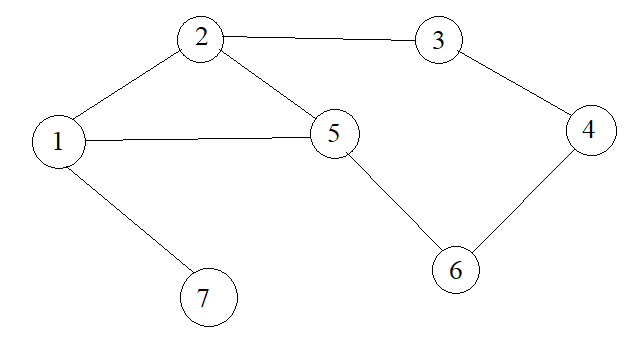


Рисунок 1. – Пример обычного графа

Задать граф в Haskell можно различными способами. Один из методов заключается в представлении каждого ребра отдельно как одного условия.

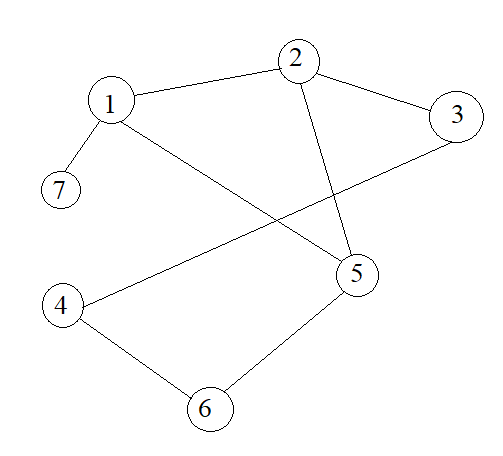


Рисунок 2 – Графическое представление графа, описанного формой с закрытыми вершинами

edge(1,2).

edge(2,3).

edge(4,6).

edge(4,3).

edge(5,6).

edge(5,2).

edge(5,1).

edge(1,7).

Так называемая форма с закрытыми вершинами. Однако с такой формой сложно работать.

Другой метод заключается в представлении всего графа в виде одного объекта данных. Согласно определению графа как пары из двух множеств (узлов и ребер), мы можем использовать следующий терм для представления примера графа:

graph ([b, c, d, f, g, h, k],[e(b, c),e(b, f), e(c, f),e(f, k),e(g, h)])

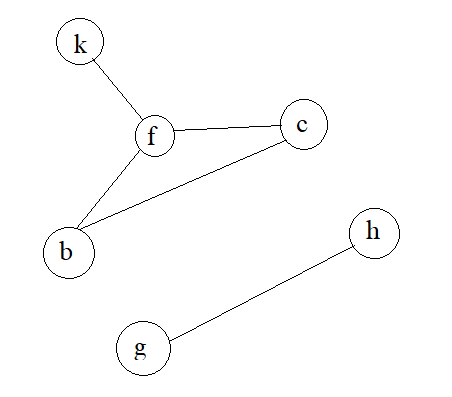


Рисунок 3. – Графическое представление графа

Эта форма графа называется терм-формой.

Третий способ представления заключается в том, чтобы связать с каждым узлом набор узлов, которые находятся рядом с этим узлом. Такой представление называется списком смежности.

[n(b,[c, f]), n(c, [b, f]), n(d,[]), n(f,[b, c, k]), ...] (Рис. 3)

Четвертый способ – простой и интуитивно понятный. Задаются пары узлов, обозначающие рёбра между ними.

[(b, c), (f, c), (g, h), (f, b), (k, f), (h, g)] (Рис. 3)

Список не обязательно сортировать, он может даже содержать одно и то же ребро несколько раз.

# ЗАДАЧИ С ГРАФАМИ. ПОСТАНОВКА И РЕАЛИЗАЦИЯ РЕШЕНИЯ

## Задача 1.

**Постановка задачи: Найти все пути из узла a в узел b. Программа должна принимать значения узлов a,b и граф. На выход будет получен массив из найденных путей.**

**Реализация:**

|  |
| --- |
| module Main (main) where  main::IO()  main = do print $ show (path 1 4 n1)    n1 = [(1,2),(1,3),(1,4),(2,4)]  path :: Eq a => a -> a -> [(a, a)] -> [[a]]  path a b n  | a == b = [[b]]  | otherwise = [ [a] ++ output | pair <- n, (fst pair) == a, output  <- path (snd pair) b [x | x <- n, x /= pair]] |

Для функции задан класс Eq определяющий равенство (==) и неравенство (/=)., то есть определяет отношения рефлексивности, симметрии, транзитивности и отрицания. Функция принимает значения типа **а** и массив пар (так задан наш граф), возвращая массив путей (тоже массивов). Также использованы встроенные функции **fst** и **snd**, возвращаемые первый и второй элемент пары соответственно. Ниже приведена их реализация.

Для решения задачи применен несложный рекурсивный алгоритм с использованием генераторов списков. На первой итерации берутся все пары с одинаковым первым значением

(fst pair) == a

После, если второй элемент равен b, составляется список [a,b], если же – не равен b, то применяется рекурсивный вызов функции path, где за первый аргумент берется второй из рассматриваемой пары,

(snd pair)

а за второй – b, массив же даётся тот же, но удаляется рассматриваемая пара.

[x | x **<-** n, x /= pair]]

Так происходит до тех пор, пока не будет выполнено условие выхода. В результате мы получаем все возможный пути из a в b.

В программе функция вызывается с аргументами a = 1, b = 4 и графом, графическое представление которого приведено ниже.

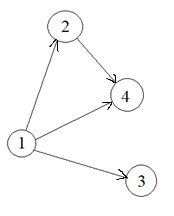
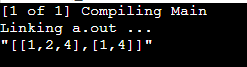


Рисунок 4. – Граф, задаваемый в программе

В результате выполнения программы мы получим вывод:



Несложно увидеть, что решение задачи является верным.

## Задача 2.

**Постановка задачи: Найти все циклы из заданного узла. Программа должна принимать значение узла a и граф. На выход будет получен массив найденных циклов.**

**Для решения данной задачи используется функция path из предыдущей задачи**

**Реализация:**

|  |
| --- |
| module Main (main) where  main::IO()  main = do print $ show (findcycle 1 n1)    n1 = [(1,2),(1,3),(1,4),(2,4),(4,1),(4,2)]    path :: Eq a => a -> a -> [(a, a)] -> [[a]]  path a b n  | a == b = [[b]]  | otherwise = [ [a] ++ output | pair <- n, (fst pair) == a, output  <- path (snd pair) b [x | x <- n, x /= pair]]  findcycle :: (Eq a) => a -> [(a, a)] -> [[a]]  findcycle a n = [a : output | pair <- n, (fst pair) == a, output <- path  (snd pair) a [x | x <- n, x /= pair]] ++  [a : output | pair <- n, (snd pair) == a, output <- path  (fst pair) a [x | x **<-** n, x /= pair]] |

Листинг 2

Работу выполняет функция findcycle. Она очень похожа на path. Мы берём первый искомый узел

(fst pair) == a

и находим узлы, с которыми он связан, а уже от них вызываем функцию path, которая из соединенных с **а** узлов ищет путь в **а**, что по сути и будет давать цикл. Первый генератор списков ищет цикл, отталкиваясь от первого элемента, а второй – от второго. В них просто поменяны местами функции fst и snd. Таким образом мы проходимся по нашему графу с разных направлений.

В программе функция вызывается с аргументами a = 1 и графом, графическое представление которого приведено ниже.

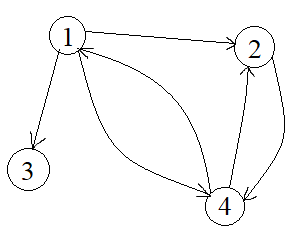


Рисунок 5. – Граф, задаваемый в программе

В результате выполнения программы мы получим вывод:



Несложно увидеть, что решение задачи является верным.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был рассмотрен функционал языка программирования Haskell для решения задач, связанных с графами. Haskell предоставляет удобные условия для реализации решения подобных задач.

Показана реализация решения проблем, где генераторы списков и рекурсивные вызовы функции выступают основными инструментами работы. Приведены графические примеры графов для наглядного представление и полного понимания темы. Таким образом, были закреплены навыки работы с рассматриваемым языком.

# СПИСОК ЛИТЕРАТУРЫ

1. Миран Липовача Изучай Haskell во имя добра! / Пер. с англ. Леушина Д., Синицына А., Арсанукаева Я.– М.: ДМК Пресс, 2012. – 490 с.: ил. ISBN 978-5-94074-749-9
2. Г.М. Сергиевский, Н.Г. Волченков. Функциональное и логическое программирование. – М.: Академия, 2010. – 320 с.: ил. ISBN: 978-5-7695-6433-8
3. Р.В. Душкин. 14 занимательных эссе о языке Haskell и функциональном программировании. – М.: ДМК Пресс, 2016. – 222 с.: ил. ISBN: 978-5-97060-360-4
4. С.В. Микони. Дискретная математика для бакалавра. Множества, отношения, функции, графы. – СПб.: Лань, 2013. – 192 с.: ил. ISBN: 978-5-8114-1386-7
5. Уилл Курт Программируй на Haskell / пер. с англ. Я. О. Касюлевича, А. А. Романовского и С. Д. Степаненко; под ред. В. Н. Брагилевского. – М.: ДМК Пресс, 2019. — 648 с.: ил. ISBN 978-5-97060-694-0