Ермолина Виктория, Пми-3,4

Задача

Вар 3. Ориентированный граф задан списком ребер. Каждое ребро представляет собой тройку (вершина, вершина, длина). Все длины являются натуральными числами. Найти расстояние между двумя заданными вершинами в графе. Если пути между указанными вершинами нет, вывести –1.

# Описание алгоритма на Haskell

Алг updateMarks

Если расстояние между текущей вершиной и вершиной из списка меток, обрабатываемой в данный момент времени = -1 то оставить для и вершиной, обрабатываемой в данный момент времени старую метку, и добавить кортеж (обрабатываемая вершина, ее неизмененная метка) к результату выполнения Алг updateMarks для текущей вершины, текущего расстояния, оставшихся элементов списка меток и списка ребер.

Иначе если метка обрабатываемой вершины равна -1 то добавить кортеж (обрабатываемая вершина, текущее расстояние + вес ребра между обрабатываемой вершиной и текущей) к результату выполнения Алг updateMarks для текущей вершины, текущего расстояния, оставшихся элементов списка меток и списка ребер.

Иначе если метка обрабатываемой вершины больше суммы текущего расстояния и веса ребра между обрабатываемой вершиной и текущей, то добавить кортеж (обрабатываемая вершина, текущее расстояние + вес ребра между обрабатываемой вершиной и текущей) к результату выполнения Алг updateMarks для текущей вершины, текущего расстояния, оставшихся элементов списка меток и списка ребер.

Иначе добавить кортеж (обрабатываемая вершина, ее неизмененная метка) к результату выполнения Алг updateMarks для текущей вершины, текущего расстояния, оставшихся элементов списка меток и списка ребер.

Кон алг updateMarks

Изначальные параметры процедуры Dijkstra – стартовая вершина как текущая вершина, 0 как текущее расстояние, искомая вершина, преобразованный в виде списка вершин с метками (для всех вершин метка равна -1, стартовая в списке отсутствует) список ребер, список ребер

Алг Dijkstra

Если текущая вершина равна искомой то вернуть текущее расстояние

Иначе если расстояние первого минимума не равно -1 то выполнить Алг Dijkstra для вершины первого минимума, расстояния первого минимума, обновленного списка меток, из которого удалена вершина первого минимума, и списка ребер

Иначе Если расстояние второго минимума не равно -1 то выполнить Алг Dijkstra для вершины второго минимума, расстояния второго минимума, обновленного списка меток, из которого удалена вершина второго минимума, и списка ребер

Иначе вернуть -1, при этом

Обновленный список меток = переменная, куда записывается результат выполнения Алг updateMarks для текущей вершины, текущего расстояния, списка меток и списка ребер

Первый минимум = записывается вершина с минимальной меткой из обновленного списка

Первый минимум = записывается вершина со второй минимальной меткой из обновленного списка

Кон алг Dijkstra

# Описание алгоритма на C#

В программе реализован класс «ребро» (Edge) вида (вершина, вершина, расстояние), а также класс «граф» (Graph), содержащий список ребер и вершин, задающих некоторый граф.

Алг GetAdjacentNodesInOrientedGraph

Определить список ребер как переменную результата.

Для каждого ребра в списке ребер графа

Нц

Если исходящая вершина ребра совпадает с переданной параметром то добавить ребро в результат

Кц

Вернуть результат

Кон алг GetAdjacentNodesInOrientedGraph

Алг AllNodesVisited

Для каждой вершины в списке вершин графа

Нц

Если текущая вершина не посещена (значение visitedNodes по этой вершине = ложь), вернуть как результат функции «Ложь»

Кц

Вернуть как результат функции «Истина»

Кон алг AllNodesVisited

Процедуре Dijkstra на вход подается два параметра строкового типа: начальная вершина (startNode) и искомая вершина (desiredNode)

Алг Dijkstra

Нач

Задать переменную меток (marks) как словарь, где ключами является строка, а значением – целочисленное значение;

Задать переменную посещенных ребер (visitedNodes) как словарь, где ключами является строка, а значением – логическое значение;

Для каждой вершины в списке вершин графа

Нц

Присвоить переменной посещенных ребер по текущей вершине значение «Ложь»

Кц

Задать строковую переменную текущей вершины (currentNode), присваивая ей значение начальной переданной в процедуру вершины (startNode)

Пока currentNode – не искомая вершина и значение функции AllNodesVisited не истина

Нц

Задаем переменную флага на необходимость выхода из цикла noReachableNodes со значением «Истина»

Задать переменную минимума со значением максимально возможного значения целочисленного типа.

Для каждой вершины в списке вершин графа

Нц

Если метка вершины не равна -1 и значение метки меньше минимума и вершина еще не посещена

То поменять значение минимума на значение метки, выбрать вершину как текущую вершину, выставить флаг в «Ложь»

Кц

Если Значение флага «Истина» то прерываем работу цикла.

В посещенных вершинах для текущей вершины выставить «Истина»

Получить список соседних вершин текущей вершины (adjacentNodes) с помощью метода GetAdjacentNodesInOrientedGraph, передавая как параметр текущую вершину

Отсортировать список соседних вершин по их расстоянию от текущей вершины

Для каждой вершины в списке соседних вершин текущей вершины

Нц

Если метка вершины равна -1 или значение метки больше суммы метки текущей вершины и расстояния до вершины

То поменять значение метки вершины на сумму метки текущей вершины и расстояния до вершины

Кц

Кц

Если список меток пуст или не содержит метку искомой вершины то вернуть -1

Вернуть метку искомой вершины

Кон алг Dijkstra

# Трудоемкость разработки

Для описания алгоритма в декларативном стиле тяжело было представить реализацию итеративного алгоритма с меняющимися на каждом шаге данными (учитывая особенности функциональной парадигмы, исключающей изменение значений переменных). Описания более абстрактные: иногда тяжело сформулировать последовательность действий без явных шагов, как в императивном стиле, но при этом сами функции более абстрактные и позволяют выполнять некоторые действия, не задумываясь над деталями реализации (Например, фильтр списка по определенному предикату).

В то же время императивный стиль более привычный, но даже с ним кратковременно возникли проблемы после дня разработки программы на Haskell и полного погружения в функциональную парадигму. Был выбран язык C#, поэтому было достаточно легко определить свои классы, моделирующие объект графа, что способствовало удобству и интуитивности работы с ним.

По продолжительности примерно восемь часов реализации алгоритма Дейкстры на Хаскелле против примерно четырех часов реализации на С#.

# Трудоемкость тестирования и отладки

В силу отсутствия какого-либо серьезного IDE для хаскелла, не было возможности отладки программными средствами, что вызвало необходимость выполнять всю трассировку и проверку корректности работы функций вручную: прослеживать путь от передачи параметров в функцию через всю цепочку преобразований над данными, фиксируя промежуточные результаты самостоятельно в отдельном месте, чтобы не «терять нить» происходящего.

С другой стороны, Visual Studio позволяет ставить брейкпоинты на любом операторе, смотреть значения переменных, объектов и структур в процессе отладки прямо в среде разработки, что, разумеется, сильно упрощает тестирование и отладку: в любой момент возможно посмотреть, правильно ли занеслись значения в структуры и какие значения получились после определенных преобразований.

# Объем полученного кода

Если учитывать только код, относящийся к реализации алгоритма Дейкстры, реализация на Хаскелле занимает 65 строк, на С# - 104 строки.

# Быстродействие, скорость работы

На небольших графах обе программы отрабатывают моментально.

На тесте с полным графом c 99 вершинами программа на C# отработала моментально, на хаскелле – 6 сек

На тесте с полным графом c 249 вершинами программа на C# отработала моментально, на хаскелле – 8 мин 14 сек

На тесте с полным графом c 1000 вершинами программа на C# отработала за 10 сек, на хаскелле – 10 минут