КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Семестровая работа №1

по алгоритмам и структурам данных

Тема: «Линейные структуры»

Выполнил:

студент группы 11-906

Камиль Аксанов

Проверил:

старший преподаватель кафедры

программной инженерии, к.ф.-м.н.

Хадиев К.Р.

Казань 2020

**Содержание**

1. Задача
2. Алгоритм
3. Оценка сложности

**Задача**

**ВАРИАНТ 8**

Класс **GraphCode**, хранящий список ребер для графа. Элемент – ребро, содержащее два номера своих вершин.

Методы:

* **конструктор GraphCode(int [][] mi)**: построение графа по матрице инцидентности
* **int [][] getMI()**: построение матрицы инцидентности
* **void insert(int i, int j):** вставка некоторого ребра (i,j).
* **void delete(int i, int j):** удаление ребра (i,j) из списка;
* **GraphCode getEdgesWithNode(int i):** возврат списка ребер, инцидентных вершине i
* **void modify(int i):** модифицировать список в связи с удалением вершины i из графа (удалить все ребра, инцидентные удаленной вершине)
* **ArrayList<Integer> show(int m):** возвратить список вершин, степень инцидентности которых более m.

**Алгоритм**

Требуется создать класс, хранящий список ребер для графа. В этом списке каждый элемент – ребро, содержащее два номера своих вершин.

Мы реализуем класс List и ListElement для хранения ребер. Далее, мы создаем класс Ribs, который будет хранить в себе два номера вершин ребра. Осталось разобраться с GraphCode’ом и его методами. В классе GraphCode у нас будут два атрибута: peaks(кол-во вершин графа) и наш реализованный NewList.

1. Конструктор GraphCode(int[][] mi): проходимся по данной нам матрице, находим вершины, инцидентные текущему ребру, и прибавляем их в наш List, перед этим создавая новый объект класса Ribs и закидывая вершины туда.
2. getMI() : проходимся по List’у, и ставим единицу напротив вершин элемента листа. Таким образом заполняем новую матрицу инцидентности и возвращаем её.
3. Insert(int i, int j): Проходимся по листу, если в нём еще нет тех вершин, которые мы передали в качестве аргумента к методу, то увеличиваем peaks. Далее просто добавляем в конец списка новое ребро.
4. Delete(int i, int j): Проходим по списку и ждём совпадения i и j вершин и тех, что хранятся в очередном элементе листа, при совпадении цикл завершается и удаляется это ребро.
5. getEdgesWithNode(int i): создаем новый GraphCode, далее, проходим по листу, если хоть один номер вершины совпадает с i => мы с помощью insert добавляем это ребро в наш новый GraphCode.
6. Modify(int i): проходимся по листу, если есть хоть одно совпадение номера вершины и i => удаляем это ребро с помощью delet’a. В конце уменьшаем кол-во peaks на единицу.
7. Show(int m): проходимся по нашей матрице инцидентности, если какая-то вершина инцидентна более m ребрам => закидываем её в наш новый arraylist.

**Оценка сложности**

**По времени:**

1. Конструктор GraphCode(int[][] mi): О(peaks \* edges) – вложенный цикл
2. getMI(): О(peaks \* edges) – вложенный цикл
3. Insert(int i, int j): O(n) – односвязный список
4. Delete(int i, int j): O(n) – односвязный список
5. Modify(int i): O(n^2) – проход по списку, а в нём delete
6. getEdgesWithNode(int i): O(n^2) – проход по списку, а в нём insert
7. show(int m): O(peaks \* list.size()) – вложенный цикл

**По памяти:**

1. О(n) - новая память выделяется только на односвязный список
2. O(peaks \* edges) – новый двумерный массив
3. O(n) – вставка с поиском
4. О(n) – удаление с поиском
5. O(n) – односвязный список, поиск
6. O(n) – односвязный список, поиск
7. O(peaks \* list.size()) – создание нового двумерного массива