Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Мегафакультет компьютерных технологий и управления

Кафедра информатики и прикладной математики

Алгоритмы и структуры данных

Лабораторная работа №5

Группа: P3218

Студент: Петкевич Константин

Преподаватель: Зинчик А. А.

2017г

Задание

Есть N карточек. На каждой из них черными чернилами написан ее уникальный номер - число от 1 до N. Также на каждой карточке красными чернилами написано еще одно целое число, лежащее в промежутке от 1 до N (некоторыми одинаковыми "красными" числами могут помечаться несколько карточек).

Необходимо выбрать из данных N карточек максимальное число карточек таким образом, чтобы множества "красных" и "черных" чисел на них совпадали.
Для примера выше это будут карточки с "черными" номерами 2, 3, 4 (множество красных номеров, как и требуется в задаче, то же - {2,3,4}).

Идея: представим карты в виде ориентированного графа с вершинами от 1 до N. Каждая вершина может иметь одно выходящее ребро и неограниченное количество входящих. Так, карта (3, 4) является вершиной с номером 3, которая имеет ребро к вершине 4.

Также заметим, что все вершины указывающие сами в себя должны попасть в ответ – т.к. по умолчанию подходят под условие. Также заметим, что в выходное множество можно добавлять все вершины, которые образуют между собой цикл (количество таких циклов не ограничено). Таким образом алгоритм можно решить либо поиском всех циклов, либо, более простым способом –

1. Удалить их графа все вершины указывающие в себя и добавить их в ответ
2. Удалить из графа все вершины, которые либо не имеют выходного ребра, либо не имеют входного
3. Повторить второй шаг до тех пор, пока удалять будет нечего
4. Оставшиеся вершины представляют собой все циклы – просто добавить их в ответ

Листинг кода

```
public class Main {

   private static ArrayList<Vertex> generate(int N) {
      ArrayList<Vertex> arr = new ArrayList<Vertex>();

      for (int i = 0; i < N; ++i) {
         arr.add(new Vertex(i));
      }

      for (int i = 0; i < N; ++i) {
         Vertex source = arr.get(i);
         Vertex to = arr.get((int)Math.floor(Math.random()*arr.size()));
         source.to = to;
         to.in.add(source);
      }
      return arr;
   };

   public static void main(String[] args) {
      ArrayList<Vertex> input = generate(100000);
      System.out.println(input);
      ArrayList<Vertex> result = solve(input);
      for (Vertex v : result) {
         System.out.print(v.id + " ");
      }

   }
```

```
    private static ArrayList<Vertex> solve(ArrayList<Vertex> input) {
        ArrayList<Vertex> loops = findLoops(input);
        // Find all nodes pointing on themselfes and add them to answer
        for (Vertex v : loops) {
            input.remove(v);
        }

        // Find a node which has nothing pointing on it or pointing to nowhere - delete it
        for (int i = 0; i < input.size(); ++i) {
            Vertex cur = input.get(i);

            if (cur.in.size() == 0) {
                input.remove(cur);
                if (cur.to != null) {
                    cur.to.in.remove(cur);
                }
                i = -1;
            } else if (cur.to == null) {
                input.remove(cur);
                i = -1;
            }
        }

        //Everything what is left after clearing is the answer!
        loops.addAll(input);

        return loops;
};

    private static ArrayList<Vertex> findLoops(ArrayList<Vertex> input) {
        ArrayList<Vertex> loops = new ArrayList<Vertex>();

        for (int i = 0; i < input.size(); ++i) {
            Vertex v = input.get(i);
            if (v.to == v) {
                loops.add(v);
            }
        }
        return loops;
    }
}
```

Класс Vertex

```
public class Vertex {
  public ArrayList<Vertex> in;
  public Vertex to;
  public Integer id;

  public Vertex(Integer id) {
    this.in = new ArrayList<Vertex>();
    this.id = id;
  }

  @Override
  public String toString() {
    return String.format("%s %s", id, to.id);
  }
}
```