Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

**Институт информационных технологий математики и механики**

Отчет по лабораторной работе

**Сравнение алгоритмов для поиска компонент связности в графе.**

**Выполнили:** студенты группы 0836-2

Доронин Р.О. и Федорова Д.А

**Проверил:**

Грибанов Д.В.

Нижний Новгород

2017

Оглавление

[Постановка задачи 3](#_Toc451168331)

[Алгоритм поиска компонент связности 4](#_Toc451168332)

[Разделенные множества 5](#_Toc451168333)

[Поиск компонент связности на основе разделенных множеств 6](#_Toc451168334)

[Подтверждение корректности 7](#_Toc451168335)

[Сравние алгоритмов по времени работы 1](#_Toc451168336)0

[Вывод 11](#_Toc451168335)

# ****Постановка задачи****

* Реализовать алгоритмы для поиска компонент связности в графе:

1. поиск в глубину
2. поиск в ширину
3. алгоритм с использованием разделенных множеств

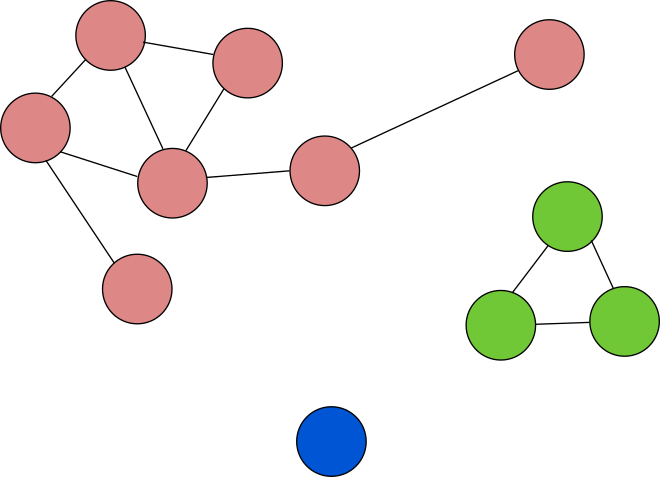
* Рассмотреть онлайн вариант с динамическим добавлением вершин и ребер.
* Использовать различные реализации структуры данных разделенное множество:

1. на массивах
2. на древовидных структурах
3. на древовидных структурах с рангами
4. сжатие путей.

* Сравнить вышеперечисленные алгоритмы и провести анализ работы получившихся реализаций с наборами данных различной длинны, сделать вывод.

# Алгоритм поиска компонент связности

**Компонента связности графа** — некоторое множество вершин [графа](http://dic.academic.ru/dic.nsf/ruwiki/8789) такое, что для любых двух вершин из этого множества существует путь из одной в другую, и не существует пути из вершины этого множества в вершину не из этого множества.



*Рис1. Граф с тремя компонентами связанности*

Для поиска компонент связности можно использовать [поиск в ширину](http://dic.academic.ru/dic.nsf/ruwiki/130131) или [поиск в глубину](http://dic.academic.ru/dic.nsf/ruwiki/96862). При этом затраченное **время** будет **линейным** (относительно количества вершин и ребер).

Фактически, мы будем производить серию обходов: сначала запустим обход из первой вершины, и все вершины, которые он при этом обошёл — образуют первую компоненту связности. Затем найдём первую из оставшихся вершин, которые ещё не были посещены, и запустим обход из неё, найдя тем самым вторую компоненту связности. И так далее, пока все вершины не станут помеченными.

Итоговая асимптотика составит : в самом деле, такой алгоритм не будет запускаться от одной и той же вершины дважды, а, значит, каждое ребро будет просмотрено ровно два раза (с одного конца и с другого конца).

# ****Разделенные множества****

*Разделенные множества* — это абстрактный *тип данных*, предназначенный для представления коллекции, состоящей из некоторого числа  попарно непересекающихся подмножеств  заданного *множества* .

Как правило, в таких задачах вычисления начинаются с пустой коллекции подмножеств (  ). Затем по мере вычислений формируются новые подмножества, включаемые в коллекцию. Формирование новых подмножеств происходит либо путем создания одноэлементного подмножества, либо путем объединения уже существующих в коллекции подмножеств. Для осуществления таких действий используются имена включенных в коллекцию подмножеств. В качестве имени подмножества будем использовать один из его элементов (главный элемент), выбираемый по определенному правилу. Поскольку в коллекции всегда будут находиться попарно непересекающиеся подмножества *множества* , такое имя будет однозначно определять требуемое *подмножество*.

### Операции над разделенными множествами

**СОЗДАТЬ** ( ). Эта операция предназначена для введения в коллекцию нового подмножества, состоящего из одного элемента , при этом предполагается, что  не входит ни в одно из подмножеств коллекции, созданной к моменту выполнения этой *операции*. Элемент  указывается в качестве параметра. Именем созданного подмножества будет считаться сам элемент.

**ОБЪЕДИНИТЬ** **(** ). С помощью этой *операции* можно объединить два подмножества коллекции, имеющие, соответственно, имена   и  , в одно новое *подмножество*, при этом оба объединяемые подмножества удаляются из коллекции, а вновь построенное *подмножество* получает некоторое имя. Во всех рассматриваемых нами случаях именем нового полученного в результате этой *операции* подмножества будет одно из  имен   или . Имена объединяемых подмножеств указываются в качестве параметров.

**НАЙТИ** (  ). Эта операция позволяет определить имя  того подмножества коллекции, которому принадлежит элемент . Если элемент  до выполнения *операции* не входил ни в одно из подмножеств коллекции, то в качестве  берется 0.

Рассмотрим несколько способов представления коллекции разделенных множеств в памяти компьютера:

* с помощью массива;
* с помощью древовидной структуры;
* с помощью древовидной структуры с использованием *рангов вершин*;
* с помощью древовидной структуры с использованием *рангов вершин* и сжатия путей.

Последний из перечисленных способов является наиболее эффективным по времени выполнения.

# Поиска компонент связности на основе разделенных множеств

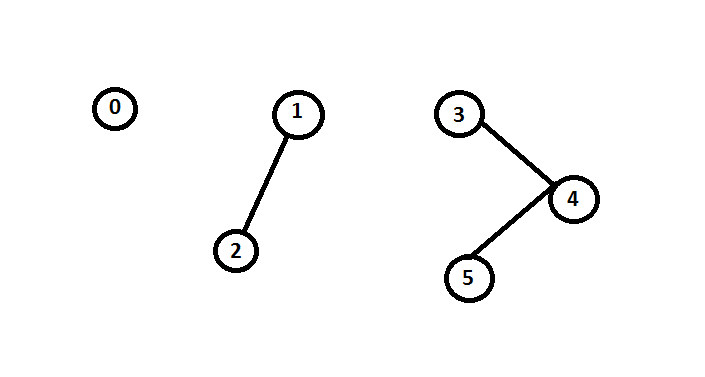
Рассмотрим задачу выделения *компонент* связности неориентированного графа. Напомним, что компонентой связности называется максимальное по включению *подмножество* вершин графа такое, что любые две его вершины связаны цепью. Полагаем, что *вершины графа* пронумерованы числами  и каждое *ребро* представлено парой **(** ) номеров вершин. Предполагаем также, что множество ребер не пусто.

**Алгоритм выделения компонент связности неориентированного графа**

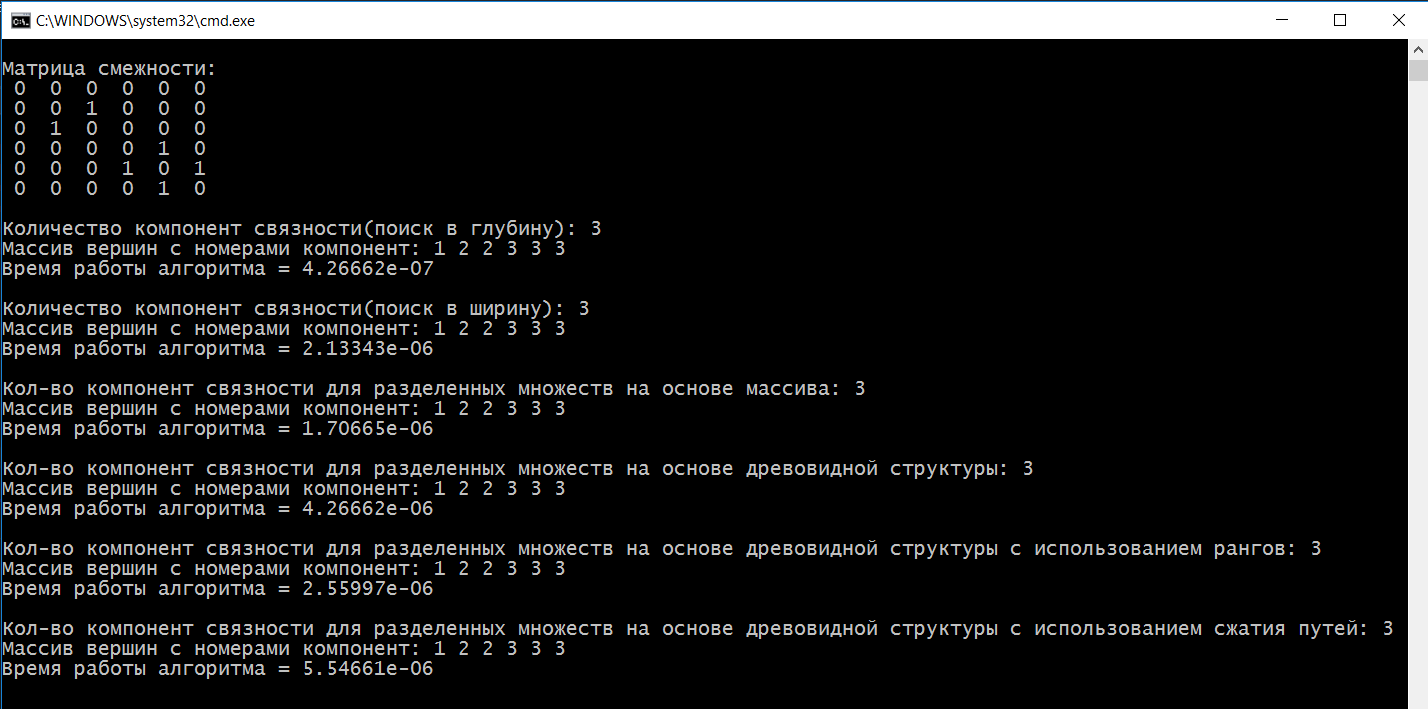
1. Создать коллекцию из синглетонов множества ;
2. Прочитать очередное ребро ;
3. Найти имя подмножества коллекции, содержащего элемент ;
4. Найти имя подмножества коллекции, содержащего элемент ;
5. Если ,то объединить подмножества с именами и;
6. Если есть еще непрочитанные ребра, перейти к п.2, в противном случае закончить вычисления.

# ****Подтверждение корректности****

1. Для подтверждения корректности работы алгоритмов был выбран граф c заранее заданной матрицей смежности:

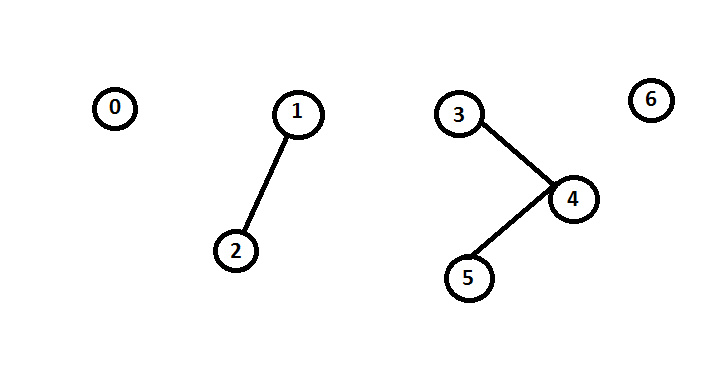


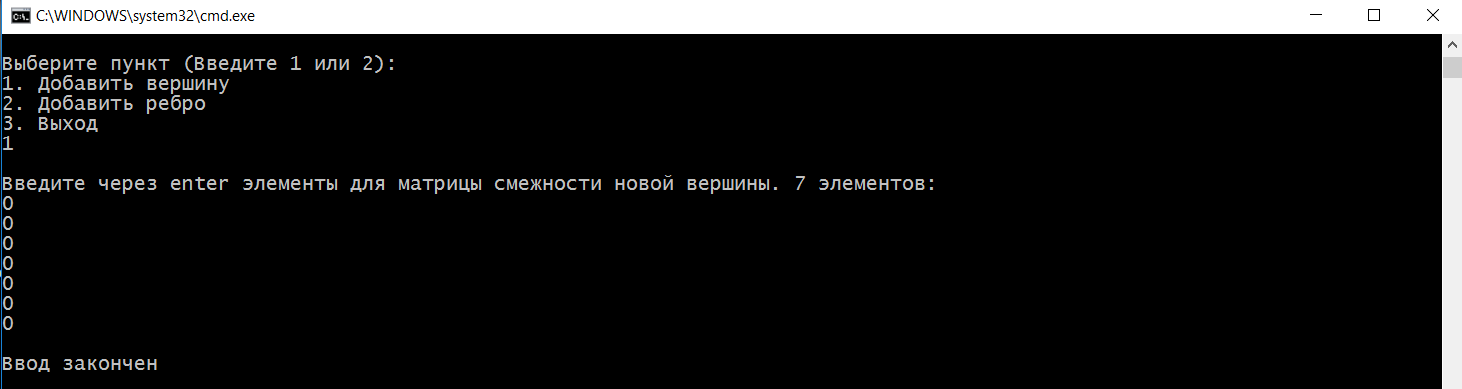
Из представленных ниже результатов видно, что количество компонент связности в каждом алгоритме совпадает и равно 3.



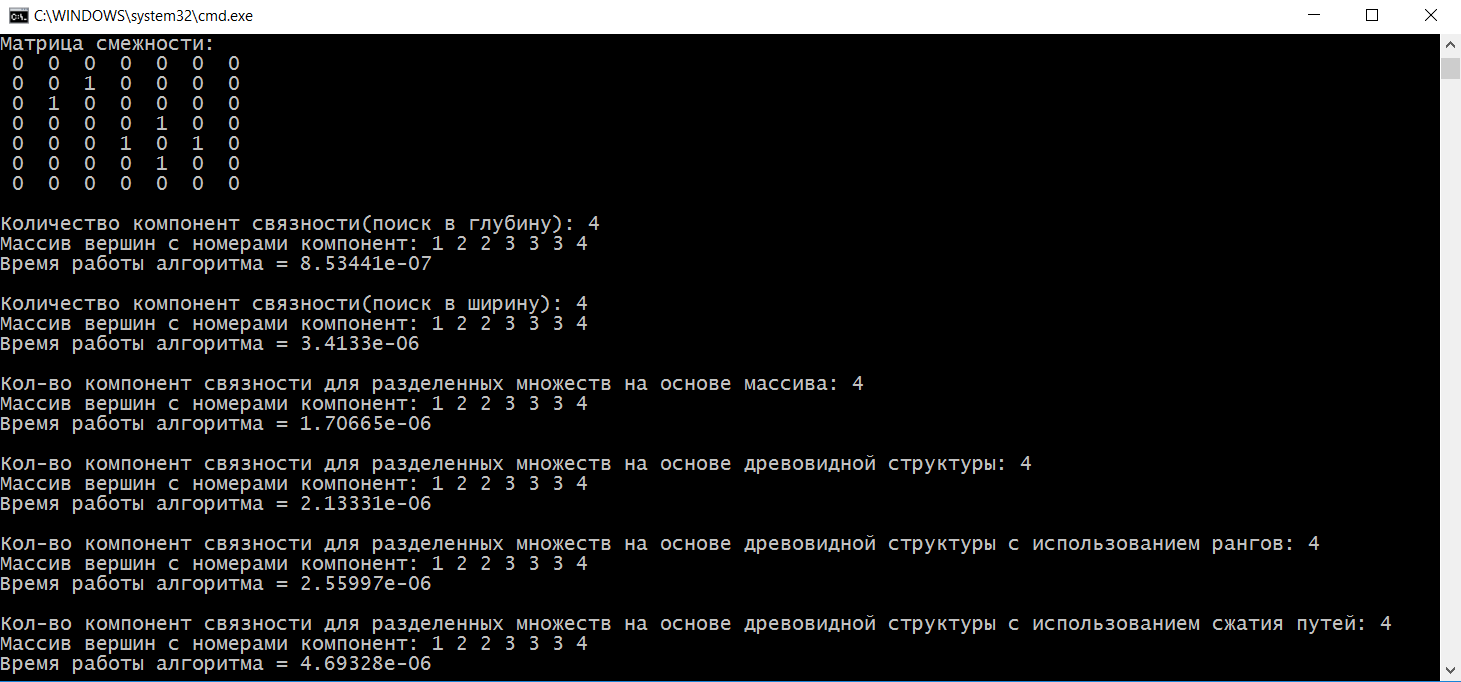
1. Продемонстрируем онлайн вариант с динамическим добавлением вершин и ребер.

Добавим новую вершину, несмежную ни с какими другими вершинами графа:

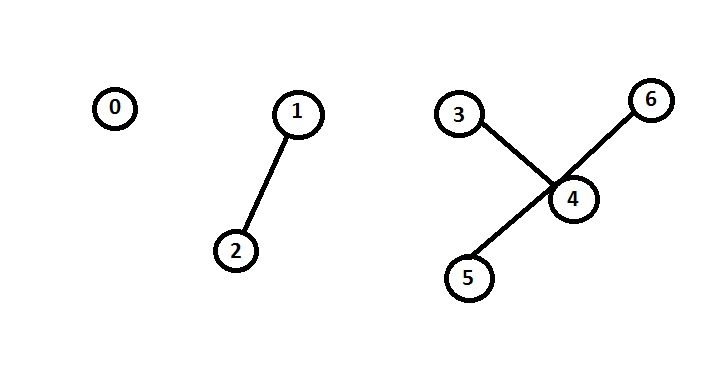


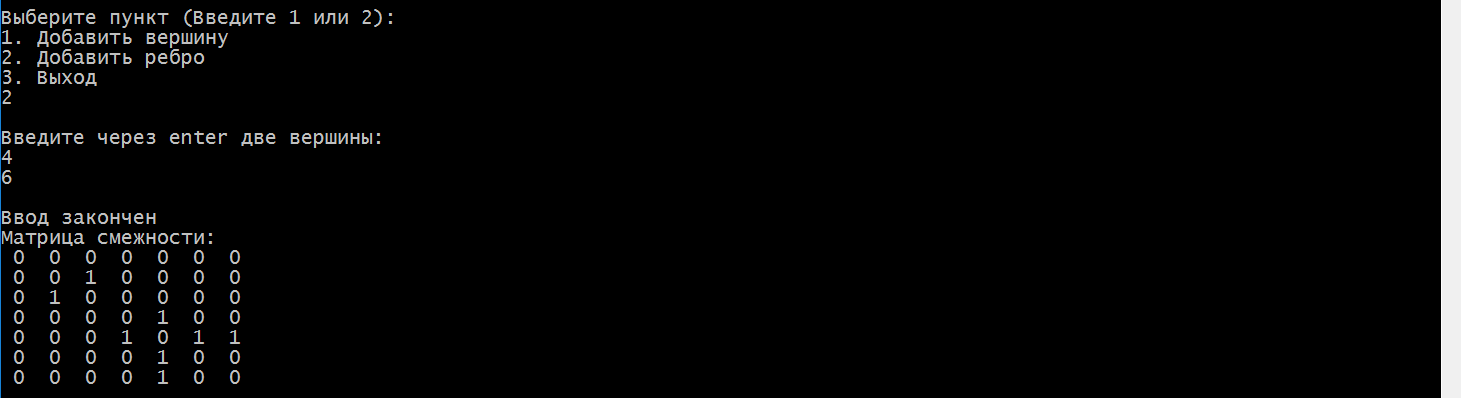


В новом получившимся графе образовалось 4 компоненты связности.

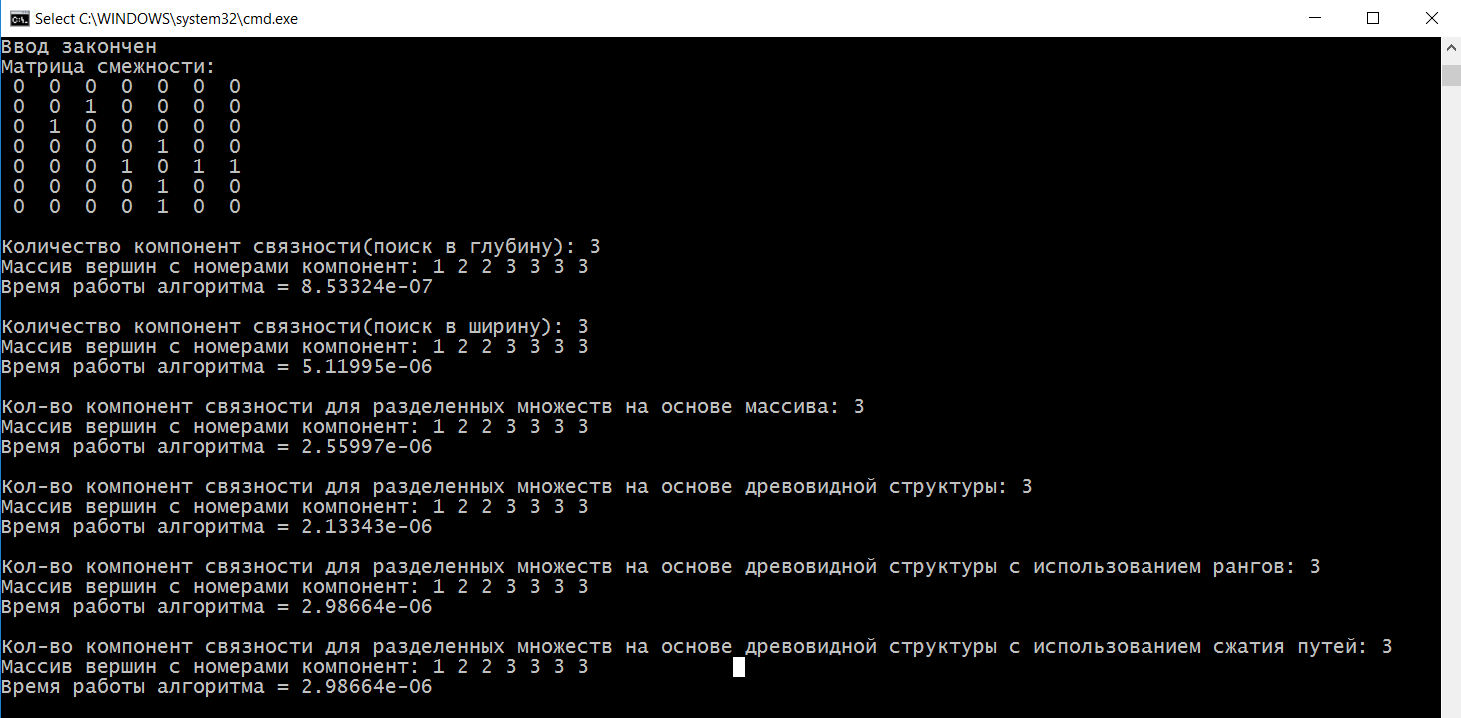


Добавим в граф ребро (4,6):





Получаем 3 компоненты связности.

****

**После выполнения каждого алгоритма представлен массив вершин с номерами компонент и время работы алгоритма.**

# ****Сравнение алгоритмов по времени работы****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ****Кол-во вершин графа**** | ****Поиск в глубину**** | ****Поиск в ширину**** | ****Разделенное множество на основе**** | | | |
| **массива** | **древовидной структуры** | **древовидной структуры с использованием рангов** | **древовидной структуры с использованием сжатия путей** |
| ****10**** | **1.27999e-06** | **2.98675e-06** | **4.26662e-06** | **3.84008e-06** | **9.38668e-06** | **8.10658e-06** |
| ****100**** | **1.32266e-05** | **4.18134e-05** | **6.4e-05** | **7.16799e-05** | **0.000105387** | **7.04001e-05** |
| ****1 000**** | **0.000969813** | **0.00170624** | **0.00253824** | **0.00186496** | **0.00189952** | **0.00189781** |
| ****10 000**** | **0.0921706** | **0.114186** | **0.230811** | **0.127084** | **0.118212** | **0.129741** |

**(Цветом выделено лучшее время работы алгоритма. Цветом выделено худшее время работы алгоритма)**

# ****Вывод****

Поиск в глубину практически во всех случаях работает быстрее остальных алгоритмов который выполняется за .

Что касается разделенных множеств, то среди них быстрее остальных работают РМ на основе древовидной структуры и РМ на основе древовидной структуры с использованием рангов.