Министерство образования Республики Беларусь
Учреждение образования
Белорусский государственный университет
информатики и радиоэлектроники

УДК 004.048; 004.021; 424.44; 424.272.2

СИНЦОВ
Сергей Викторович

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМОВ ОПЕРАЦИЙ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ЗНАНИЙ

ДИССЕРТАЦИЯ
на соискание степени магистра технических наук
по специальности 05.13.17 – Теоретические основы информатики

Научный руководитель
Ивашенко Валерьян Петрович
кандидат технических наук

Минск 2017

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

B^A – множество всех функциональных отображений из A в B

clz – функция количества нулевых разрядов перед самым старшим единичным разрядом

[*m*,..*n*] – $\left\{x \middle| \left(\left(x \in \mathbb{N}\right) \wedge \left(m \le x\right) \wedge \left(x \le n\right) \right) \right\}$

ОЗУ – оперативное запоминающее устройство

ОКМД – одиночный поток команд, множественный поток данных

ОС – операционная система

ЦПУ – центральное процессорное устройство

CPU – central processing unit (ЦПУ)

CUDA – compute unified device architecture

DSP – digital signal processor (цифровой процессор обработки сигналов)

GPU – graphics processing unit (графическое процессорное устройство)

MMX – multimedia extensions (мультимедийные расширения ОКМД инструкций)

OpenCL – open computing language (открытый язык вычислений)

OWL – web ontology language (язык описания онтологий для семантической паутины)

OSTIS – open semantic technologies for intelligent systems (открытые семантические технологии проектирования интеллектуальных систем)

RDF – resource description framework (средства описания ресурсов для семантической паутины)

RDFS – resource description framework schema («схема» RDF)

SC-код – semantic computer code (семантический компьютерный код)

SCg – Semantic Code graphical (графический SC-код)

SCP – semantic code programming (SC-код программирования)

SCs – semantic code string (строковый SC-код)

SIMD – single instruction, multiple data (ОКМД)

SPARQL – SPARQL Protocol and RDF Query Language

SSE – streaming SIMD extensions (потоковое SIMD-расширение процессора)

VLIW – very long instruction word (вычислительная архитектура с «очень длинными машинными командами»)

ВВЕДЕНИЕ

Наличие в информационных системах данных и знаний большого объёма требует эффективных механизмов их обработки. Последние, в сочетании с тенденциями автоматизации задач анализа данных, извлечения и интеграции знаний в единую систему знаний, а также задач информационного поиска, прогнозирования, поддержки принятия решений и управления [2], всё чаще ориентируются на применение методов, моделей и систем искусственного интеллекта, поддерживающих обработку семантически структурированных данных, что в свою очередь позволяет сократить семантический разрыв между пользователем и распространёнными архитектурами электронных вычислительных машин и снизить затраты на решение перечисленных задач и интерпретацию результатов решения пользователем.

Однако интеллектуальные системы [2, 33], основанные на знаниях и управляемые знаниями [5] имеют ряд свойств, в основе которых лежат свойства самих знаний как данных, обладающих внутренней интерпретируемостью, сложноструктурированностью, связностью, активностью и семантической метрикой [15]. При этом от данных систем требуется открытость, обучаемость, работа в режимах реального времени и направленность на автоматизацию решения одновременно широкого класса задач в промышленности и других отраслях народного хозяйства [2]. Характерной чертой развития современных информационных технологий является ориентация на нечисловую обработку больших объёмов сложноструктурированных данных и переход к нелинейным формам представления и организации хранения информации, а также ассоциативным моделям доступа к обрабатываемой информации.

При практическом инженерном анализе и разработке интеллектуальных систем необходимо обеспечить для них определённые характеристики производительности на каждом из информационных уровней их физической и логической архитектуры, что включает в себя количественные оценки затрат времени, памяти, энергии и других ресурсов [10, 38], необходимые для прогнозирования и планирования интеллектуальной системой процесса решения задач, а также предупреждения нештатных и аварийных ситуаций. Другими словами, интеллектуальная система должна знать о том, какие задачи она может решить в настоящий или будущий моменты времени и какие ресурсы, в каком количестве, ей для этого понадобятся.

Для развития интеллектуальных систем, учитывая особенности решаемых такими системами задач, недостаточную производительность распространённых вычислительных архитектур, а также физические ограничения на рост производительности вычислительных систем и развитие

семантических моделей представления и обработки знаний, можно выделить следующие подходы к разработке интеллектуальных систем:

1. использование для представления и обработки знаний средств и моделей, основанных на семантических сетях;
2. использование гетерогенных параллельных вычислительных архитектур в качестве аппаратной основы интеллектуальных систем.

Первый подход, отличающийся своей универсальностью, позволяет реализовать систему, способную обрабатывать не просто сложноструктурированные данные, но и знания, в условиях их нестационарности, неточности, противоречивости, неактуальности и неполноты. Кроме этого данный подход оказывается эффективным при решении задач интеграции одних баз знаний в другие [11].

Второй подход позволяет частично решить вопрос производительности систем обработки семантических сетей, возникающий с ростом сети. При этом значительное увеличение скорости обработки семантических сетей может быть получено, если задействовать все возможности распространённых параллельных вычислительных систем, разработав такие эффективные алгоритмы операций параллельной обработки семантических сетей, реализация которых может быть осуществлена на широком классе различных параллельных вычислительных систем (суперскалярных, векторных, многоядерных, многопроцессорных, многомашинных [27, 41, 75, 78]).

Диссертационная работа посвящена исследованию, разработке и реализации алгоритмов операций параллельной обработки знаний.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Тема диссертации соответствует приоритетным направлениям «Промышленные и строительные технологии и производство: робототехника, интеллектуальные системы управления» согласно пункту 3 перечня приоритетных направлений научных-технической деятельности в Республике Беларусь на 2016–2020 гг. (указ Президента Республики Беларусь от 22 апреля 2015 г. № 166), а также приоритетным направлениям «Информатика и космические исследования» согласно пункту 5 перечня приоритетных направлений научных исследований Республики Беларусь на 2016–2020 гг. (постановление Совета Министров Республики Беларусь от 12 марта 2015 г. № 190) [25, 26].

Целью диссертационной работы является разработка и реализация программных средств операций параллельной обработки знаний, представленных моделями с теоретико-множественной семантикой.

В соответствии с поставленной целью в диссертационной работе решаются следующие **задачи**:

1. Провести аналитический обзор существующих программных средств и алгоритмов операций параллельной обработки знаний.
2. Разработать модель и алгоритмы стратегий перераспределения участков линейно адресуемой битовой памяти.
3. Разработать онтологическую модель теоретико-множественных операций модели обработки знаний в конечной памяти и алгоритмы теоретико-множественных операций параллельной обработки знаний.
4. Реализовать алгоритмы стратегий перераспределения участков линейно адресуемой битовой памяти и алгоритмы теоретико-множественных операций параллельной обработки знаний.

Объектом исследования являются модели представления и обработки знаний с теоретико-множественной семантикой.

Предметом исследования являются алгоритмы теоретико-множественных операций параллельной обработки знаний.

Для решения поставленных задач применялись методы теории множеств, теории графов, теории алгоритмов, теории параллельных вычислений, методы и средства инженерии знаний. Предложенная в работе онтологическая модель теоретико-множественных операций модели обработки знаний разработана на основе такой модели представления знаний, как семантические сети, и записана на языке унифицированного семантического кодирования информации.

Новизна полученных результатов:

1. Разработана онтологическая модель теоретико-множественных операций модели обработки знаний, основанная на семантических сетях, и отличающаяся использованием семейства языков унифицированного семантического кодирования информации.

2. Разработаны модель и алгоритмы стратегий перераспределения участков линейно адресуемой битовой памяти на уровнях управления устройствами и данными системы обработки знаний. Разработанные алгоритмы стратегий управления памятью отличаются тем, что сохраняют стратегию выбора первого подходящего свободного участка памяти при перераспределении участков памяти, дают оценку перерасхода памяти по причине внешней фрагментации, не превышающую $O\left(U * \log_2\left(U\right)\right)$ (где U – объём занятой памяти), и позволяют получить приближённое решение задачи оптимального дискретного управления памятью, имея постоянную среднюю амортизационную оценку временной сложности.

3. Разработаны алгоритмы теоретико-множественных операций параллельной обработки знаний, отличающиеся поддержкой мультимножеств и ориентацией на использование параллелизма потока данных и реализацию на мелко-и среднезернистой параллельной вычислительной архитектуре с произвольным доступом к общей памяти.

Положения, выносимые на защиту:

1. Онтологическая модель теоретико-множественных операций модели обработки знаний, основанная на семантических сетях, и отличающаяся использованием семейства языков унифицированного семантического кодирования информации. В основу онтологической модели положено понятие онтологии как явной спецификации концептуализации (согласно Т. Груберу), а сама спецификация строится с использованием средств модели спецификации знаний.

2. Модель и алгоритмы стратегий перераспределения участков линейно адресуемой битовой памяти на уровнях управления устройствами и данными системы обработки знаний. Разработанные алгоритмы стратегий позволяют получить приближённое решение задачи оптимального дискретного управления памятью и отличаются тем, что сохраняют стратегию выбора первого подходящего свободного участка памяти при перераспределении участков памяти, дают оценку перерасхода памяти по причине внешней фрагментации, не превышающую $O\left(U * \log_2\left(U\right)\right)$ (где U – объём занятой памяти), имея при этом постоянную среднюю амортизационную оценку времени работы, если время доступа к ячейке памяти в зависимости от размера памяти постоянно, а количество записей данных в ячейки памяти зависит как

минимум прямо пропорционально от произведения времени одного перевыделения на количество перевыделений.

3. Алгоритмы теоретико-множественных операций параллельной обработки знаний, отличающиеся поддержкой мультимножеств и ориентацией на использование параллелизма потока данных и реализацию на мелко-и среднезернистой параллельной вычислительной архитектуре с произвольным доступом к общей памяти. Теоретическая оценка временной сложности данных алгоритмов при использовании разработанной модели стратегий перераспределения участков линейно адресуемой битовой памяти на уровнях управления устройствами и данными системы обработки знаний, не превышает $O(\log(n)*n/p)$, где n – суммарная мощность множеств, p – количество процессоров.

Личный вклад соискателя учёной степени состоит в непосредственном участии во всех этапах диссертационного исследования, анализе отечественной и зарубежной научной литературы, разработке и реализации моделей и алгоритмов, проведении экспериментов и анализе полученных результатов, написании и оформлении рукописи диссертации и основных публикаций по выполненной работе, аппробации результатов исследования. Соавтором совместных публикаций является научный руководитель, к-т техн. наук, доцент В.П. Ивашенко, принимавший непосредственное участие во всех этапах исследования: выборе направления исследований, постановке задач, обсуждении теоретических и практических результатов. В публикациях с соавтором вклад соискателя определяется рамками представленных в диссертации результатов.

Апробация результатов диссертации состоит в следующем: материалы работы докладывались и обсуждались на следующих научных конференциях и семинарах: 2-ая научно-практическая конференция «BIGDATA and Advanced Analytics» (Минск, 2016); Международный конгресс по информатике «Информационные системы и технологии» CSIST’16 (Минск, 2016); Международная научная конференция «Информационные технологии и системы» IST’2016 (Минск, 2016).

Основные результаты диссертации опубликованы в 5 научных работах, из них 1 статья (объёмом 0,5 авторского листа) в научном журнале, включённом в Перечень научных изданий, утверждённый Высшей аттестационной комиссией, 4 статьи в сборниках материалов научных конференций.

ГЛАВА 1
АНАЛИТИЧЕСКИЙ ОБЗОР ПРОГРАММНЫХ СРЕДСТВ И АЛГОРИТМОВ ОПЕРАЦИЙ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ЗНАНИЙ

Рассмотрим архитектуру интеллектуальных систем, основанную на базе знаний, машине обработки знаний и пользовательском интерфейсе [2, 4]. В такой архитектуре можно выделить уровень управления устройствами, уровень управления данными и уровень управления знаниями [14]. На уровне управления устройствами реализуется управление памятью, вычислительными и другими устройствами, что необходимо для реализации языка определения и управления данными [3]. Уровень управления данными необходим для реализации языка представления знаний, который используется на уровне управления знаниями для реализации систем, основанных на знаниях.

На каждом из уровней используются свои информационные структуры с соответствующими им структурами данных (списковыми, древовидными, многосвязными [15, 18, 19]) и формальные языки [13]. Выразительная мощность языков [36] определяется поддержкой различных видов семантики: проективной [7], дескриптивной [7], рефлексивной [7], денотационной [31], операционной [31], аксиоматической [77], модельно-теоретической [65] и др. [23].

Спецификация процессов обработки структур данных, описывающих состояние объектов управления на разных архитектурных уровнях управления, строится на основе соответствующих формальных моделей обработки информации [31, 38], которые задаются формальным языком, множеством начальных информационных структур и абстрактной машиной обработки информации. В свою очередь абстрактная машина обработки информации задаётся памятью и множеством операций над хранящимися в памяти конструкциями и определяет операционную семантику формального языка [31].

При этом выделяются специализированные формальные языки, использующиеся для программирования процесса решения задач, т.е. интерпретации модели обработки информации [31]. Каждой операции абстрактной машины обработки информации ставится в соответствие некоторая программа (т.е. текст состоящий из некоторого упорядоченного подмножества операторов языка программирования), реализуемая на конкретной программной, аппаратной или программно-аппаратной платформе [22]. Так же среди формальных языков выделяются специализированные языки спецификаций программ, которые обеспечивают формальное определение класса задач, решаемых с помощью каждой программы [2, 31].

Среди прочих выделяются модели обработки информации, основанные на семантических сетях [2, 4, 15, 31, 59]. Далее рассмотрим языки программирования, ориентированные на обработку семантических сетей.

1.1 Языки обработки знаний

При реализации интеллектуальных систем важным является выбор языка программирования, ориентированного на обработку знаний. В связи с этим было разработано множество универсальных или специализированных языков программирования, использующихся для реализации систем искусственного интеллекта: РЕФАЛ [2], Datalog [3], GP [86], Gremlin [91], KQML [66], LISP [2], OPS5 [2], OQL [44], Planner [2], PROGRES [105], Prolog [2], RuleML [55], SCP [32], Smalltalk [2], SNOBOL [2], SPARQL [95], Cypher [85], Wave language [93], Wolfram Language [103] и др. Выделяются процедурные и декларативные языки программирования [28], преимущественно описывающие соответствующие формы и виды знаний. Так же выделяются языки с линейным (символьные языки) и нелинейным синтаксисом (графовые языки) [31].

Рассмотрим языки программирования, ориентированные на обработку графовых информационных структур – сетей, в частности, семантических сетей [4, 32, 59]. Процесс решения задач в модели обработки информации, основанной на графовом представлении данных и знаний, заключается в использовании различных операторов поиска фрагментов сети с заданными свойствами и изменения конфигурации сети (трансформации графа) согласно заданным условиям [32, 92].

С этой точки зрения на сегодняшний день, среди прочих языков [70, 93, 104], выделяются языки: GP, Gremlin, PROGRES, SCP, SPARQL, Cypher. Проведём анализ данных языков, отталкиваясь от их выразительных возможностей и некоторых особенностей реализации их абстрактных машин.

В таблице А.1 перечислены некоторые свойства, относящиеся непосредственно к рассматриваемым языкам, ориентированным на обработку графовых структур. В силу перечисленных выше особенностей систем, основанных на знаниях, стоит отметить важность поддержки языком рефлексивной и модельной семантик, а также явного (т.е. управляемого на уровне синтаксиса) параллельного и асинхронного исполнения программ. Рефлексивная семантика является частным случаем проективной семантики и позволяет изучать интерпретацию формализованного языка в категориях этого же языка и, таким образом, решать задачи анализа моделей [7]. Модельно-теоретическая семантика [7, 17] даёт возможность строить теоретико-множественные модели предметных областей и интерпретировать знакосочетания формального языка в терминах модели предметной области. Поддержка параллельной асинхронной обработки графовых структур

необходима для обеспечения необходимой производительности интеллектуальных систем.

Язык программирования **SCP** (**S**emantic **C**ode **P**rogramming) является подъязыком абстрактного машинно-ориентированного языка SC-код (Semantic Computer Code) [31, 32], имеющего формальную семантику и использующегося для унифицированного семантического кодирования всевозможных знаний в виде однородных семантических сетей с базовой теоретико-множественной интерпретацией. SC-код поддерживает, в частности, модельно-теоретическую и рефлексивную семантики и позволяет описывать структуру любой информационной конструкции, не принадлежащей SC-коду, будучи хорошо приспособленным к использованию в условиях нестационарности, неточности, противоречивости, неактуальности и неполноты знаний. Язык SCP принадлежит классу процедурных языков программирования с поддержкой параллелизма и ориентированных на переработку нечисловой информации. Знакосочетания языка SCP и обрабатываемые им данные являются графовыми конструкциями, хранящимися в графовой структурно перестраиваемой ассоциативной памяти. Также язык SCP обладает Тьюринг-полнотой и, кроме этого, позволяет интегрировать процедуры над информационными конструкциями, представленными в SC-коде, с процедурами, записанными на традиционных языках программирования (таких, например, как C/C++ [16]). Программы языка SCP могут быть записаны как в текстовом (SCn, SCs, M4SCP), так и в графическом (SCg) форматах.

Язык программирования **PROGRES** (**PRO**gramming with **G**raph **RE**writing **S**ystems) является мультипарадигменным языком с формально заданной семантикой, сильной статической типизацией и поддержкой процедурного и логического стилей программирования, программы которого могут быть записаны как в текстовом формате, так и в формате графических диаграмм. Язык PROGRES построен на основе графовых грамматик [Rozenberg_1999] и может быть использован для представления и получения данных в объектно-ориентированных базах данных, спецификации структур данных и средств обработки графовых структур, синтаксического анализа диаграмм, определения семантики визуальных языков запросов к базам данных. Язык PROGRES обладает Тьюринг-полнотой, т.е. на нём можно запрограммировать любую вычислимую функцию [17]. Кроме того, язык принадлежит классу недетерминированных языков программирования, поддерживая возможность возврата по потоку управления (backtracking of control flow) и возможность возврата по структурам данных (backtracking on data structures).

Язык программирования **GP** (**G**raph **P**rograms) принадлежит классу декларативных языков с поддержкой логического стиля программирования и является недетерминированным языком, основанным (так же, как и язык PROGRES) на графовой грамматике. Язык GP является Тьюринг-полным и

ориентирован на программирование решений задач на графах в виде программ, записанных в текстовом и/или графическом форматах. Главной целью языка является формальная верификация программ и формальный логический вывод, что обусловливает наличие у данного языка формальной аксиоматической семантики.

Язык программирования **SPARQL** (**S**PARQL **P**rotocol and **R**DF **Q**uery **L**anguage) составляет часть технологии Semantic Web [99] (Семантическая Паутина), наряду с форматом представления данных RDF, языком описания отношений для модели представления данных RDF Schema, языком описания онтологий OWL и др. Язык SPARQL схож с языком SQL [3], обладает линейным синтаксисом и является декларативным языком запросов к данным, хранящимся в формате RDF [95, 99].

Язык программирования **Gremlin** принадлежит классам процедурных и декларативных языков программирования, позволяя производить обходы графа и сопоставлять графовые конструкции с заданным образцом. Язык Gremlin является символьным языком с формально заданной операционной семантикой. Язык Gremlin поддерживает определение предметно-ориентированных подъязыков, позволяет производить оптимизацию программ во время компиляции, поддерживает явный параллелизм и распределённую обработку информации в многомашинных системах с распределённой памятью [27] и, кроме этого, обладает Тьюринг-полнотой.

Язык программирования **Cypher Query Language** (язык запросов Cypher) принадлежит классу декларативных языков с линейным SPARQL подобным синтаксисом и является языком запросов к графовой базе данных Neo4J. При этом машина интерпретации языка Cypher реализована на платформе виртуальной машины JVM, на языке Java [100] как часть самой базы данных.

Множество операторов языков программирования, ориентированных на обработку графовых структур, можно классифицировать по их дескриптивной и операционной семантике следующим образом [3, 32, 85, 95].

Операторы не изменяющие конфигурацию сети:

– операторы поиска фиксированной графовой конструкции, (не)удовлетворяющей заданным условиям;

– операторы поиска по образцу (поиска графовой конструкции, изоморфной заданной графовой конструкции);

– операторы поиска путей в графе;

– операторы обхода графа (в ширину, в глубину);

– операторы условных и безусловных переходов;

– операторы порождения и управления параллельными процессами исполнения программы;

– операторы проверки условий (проверки типа операнда, проверки (не)строгого (не)равенства операндов);

– операторы арифметических операций;

– операторы агрегирования;

– операторы теоретико-множественных операций (объединения, пересечения, разности и др.) над операндами (множества вершин, дуг, рёбер) и другие операторы.

Операторы изменяющие конфигурацию сети:

– операторы копирования графовых конструкций по образцу (генерации графовой конструкции, изоморфной заданной графовой конструкции);

– операторы генерации элементов графа (вершин, дуг, рёбер) и графовых конструкций;

– операторы удаления элементов графа и графовых конструкций, удовлетворяющих заданным условиям;

– операторы изменения свойств элементов графа и другие операторы.

Множество операций абстрактных машин, определяющих операционную семантику данных языков программирования, может взаимно-однозначно соответствовать множеству операторов языка либо отличаться от него. В общем случае дескриптивная семантика разных языков определяется разным подмножеством приведённого множества операторов, которые записываются в общем случае отличными друг от друга синтаксическими конструкциями. При этом рассматриваемые языки практически одинаково хорошо приспособлены для обработки графовых структур. Исключение составляют языки, не обладающие вычислительной полнотой (языки запросов Cypher и SPARQL), а также языки, не поддерживающие на уровне операторов параллельное и асинхронное исполнение программ (PROGRES, GP, языки запросов Cypher и SPARQL).

Однако рассматриваемые языки программирования, значительно различаются по своим выразительным возможностям и, следовательно, лучшей или худшей приспособленностью для обработки знаний, закодированных в виде семантических сетей. С этой точки зрения язык SCP обладает наибольшей семантической мощностью, так как является подъязыком абстрактного графового языка SC-код, обладающего, в частности, рефлексивной и модельно-теоретической семантикой. Отсутствие декларативности у языка SCP компенсируется возможностями создания на базе SC-кода целого семейства совместимых между собой подъязыков предметных областей. Язык SPARQL, хотя и входит в семейство стандартов технологии Semantic Web (RDF, RDFS, OWL и др. [99]), направленных на представление и обработку знаний, однако, кроме своей семантической ограниченности, также синтаксически и семантически не совместим с другими языками технологии Semantic Web, что затрудняет переход от представления знаний (в том числе процедурных) к их обработке. Аналогичное заключение можно сделать в отношении оставшихся языков программирования.

Во множестве операций машин обработки информации, соответствующих рассматриваемым языкам программирования, можно выделить класс операций, обеспечивающих решение базовых задач по преобразованию текущего состояния памяти машины, к которым относятся арифметические, логические, строковые, теоретико-множественные и информационно-поисковые операции. В процессе решения задач с использованием рассматриваемых языков программирования, а также в процессе интерпретации программ данных языков, приходится решать вспомогательные задачи на графовых структурах. К числу таких задач можно отнести топологическую сортировку, поиск в ширину, поиск в глубину, поиск путей, поиск компонент связности, поиск изоморфного подграфа, раскраску графа, каноническую разметку графа и др. [9, 35, 40, 50, 74]. В большинстве перечисленных задач, а также в задачах формального логического вывода, интеллектуального анализа данных (в частности, в большинстве алгоритмов метода анализа формальных понятий [79]) и др., ключевыми являются базовые теоретико-множественные операции, а именно принадлежность элемента множеству, пересечение, объединение и разность множеств, которые обеспечивают поддержку модельно-теоретической семантики языка обработки семантических сетей.

1.2 Программные средства обработки знаний

Реализация интеллектуальной системы подразумевает, в частности, реализацию соответствующего языка программирования, использующегося для обработки данных и знаний, а значит – реализацию абстрактной машины обработки информации, определяющей операционную семантику языка, на некоторой платформе (программной, аппаратной, программно-аппаратной), характеризующейся определёнными физическими параметрами, такими, как затраты памяти, временны́е затраты, энергоинформационные затраты (потери), и другие [14, 22]. При этом говорят о реализуемой платформе (в нашем случае – платформе языка программирования) и реализующей платформе [14, 21], которая в свою очередь может быть реализована на некоторой реализующей платформе и т.д.

Проанализируем реализацию платформ выбранных языков программирования со следующих позиций: с позиции использования в реализации абстрактных машин обработки информации неявного (т.е. поддерживаемого на уровне исполнителя) параллелизма и с позиции наличия формальных спецификаций операций и программ реализуемых машин. Результаты анализа собраны в таблицах А.1–А.4.

Говоря о параллелизме, в зависимости от отношения объёма вычислений, выполненных в параллельной задаче, к объёму коммуникаций для обмена

сообщениями, выделяют такие его виды (формы), как мелкозернистый (скалярный параллелизм), среднезернистый и крупнозернистый параллелизм (в форме векторного параллелизма и параллелизма независимых ветвей) [27, 41]. Можно дать следующие условные характеристики перечисленным видам параллелизма. Так, мелкозернистый параллелизм обеспечивают аппаратная платформа (процессор) и, частично, компилятор языка, который может явно использовать специальные параллельные команды вычислительных архитектур, например, типа SIMD (VLIW, SSE, MMX и др.) согласно систематике Флинна [27, 41]. Среднезернистный параллелизм формы независимых ветвей обеспечивается при разработке многопоточных программ, возможно, исполняющихся на разных вычислительных устройствах гетерогенной параллельной вычислительной архитектуры (многоядерными CPU, GPU, DSP и др.) в пределах одной машины (вычислительного узла) [75, 78]. При этом среднезернистый векторный параллелизм может быть обеспечен при использовании устройств GPU, обладающих архитектурой типа MIMD [41, 75, 78]. Крупнозернистый параллелизм обеспечивается на уровне операционной системы (ОС) и распределённых вычислений над распределённой разделяемой (неразделяемой) памятью [20]. Мелкозернистый параллелизм поддерживается большинством современных компиляторов языков программирования C/C++ и Java [16, 100].

Стоит отметить, что в настоящее время существует две технологии, использующиеся в качестве инструментов для написания программ, ориентированных на исполнение в гетерогенных вычислительных архитектурах, в частности на вычислительных устройствах GPU, а именно технологии CUDA [29] и OpenCL [75]. Причём, в отличие от технологии CUDA, технология OpenCL основана на открытом стандарте, позволяет работать с различными типами устройств и позволяет исполнять один и тот же код на различных типах устройств.

Согласно проведённому анализу, язык программирования **SCP** реализован в рамках проекта OSTIS [83] в виде интерпретатора с открытым исходным кодом, написанным на языке программирования C и исполняющимся на аппаратных платформах семейства x86 (x86-64). Реализованная платформа языка не специфицирована и поддерживает мелкозернистый параллелизм и параллелизм независимых ветвей в виде процессов и потоков ОС, исполняющихся на многоядерных устройствах типа CPU.

Для исходных кодов языка программирования **PROGRES** написан транслятор [87], переводящий их в исходные коды языков программирования C и Modula-2 [102], исполняющиеся на аппаратных платформах x86 (x86-64). Исходные коды системы PROGRES не находятся в открытом доступе, платформа языка не специфицирована и поддерживает только мелкозернистый параллелизм.

Язык программирования **Cypher** реализован как часть системы Neo4j [81] на языке программирования Java, на базе платформы виртуальной машины JVM [100]. Реализация языка Cypher открыта, не специфицирована и поддерживает мелкозернистый параллелизм и параллелизм независимых ветвей в виде процессов и потоков ОС, исполняющихся на многоядерных устройствах типа CPU.

Программы языка программирования **GP** компилируются в байт-код виртуальной машины YAM [68], написанной на языке программирования C. Исходные коды компилятора GP и машины YAM не находятся в открытом доступе, платформа языка не специфицирована и поддерживает только мелкозернистый параллелизм.

Из всех рассматриваемых языков программирования для языков **SPARQL** и **Gremlin** насчитывается больше всего реализаций, выполненных различными независимыми коллективами в рамках различных, конкурирующих между собой систем. Сравнительное описание большинства данных реализаций приведено в таблицах А.3 и А.4 соответственно для SPARQL (среди них: [46, 53, 57, 82, 88, 96, 97] и др.) и Gremlin (среди них: [49, 53, 60, 81, 82, 97, 70, 101] и др.), при этом реализации выполнены на таких языках программирования, как C/C++, Java, Lisp, JavaScript, C# [2, 16, 34, 39, 100] и др. В основе систем, реализующих язык Gremlin, лежит виртуальная машина Apache TinkerPop [49]. Кроме этого системы, реализующие язык Gremlin, способны исполнять и язык SPARQL, так как его исходные коды могут быть переведены в коды виртуальной машины Gremlin.

Как видно из приведённых в таблицах А.3 и А.4 данных все реализации рассматриваемых языков программирования поддерживают мелкозернистый параллелизм (в силу возможностей современных компиляторов и вычислительных архитектур), большинство реализаций также поддерживает многопоточное исполнение на многоядерных процессорах (CPU), некоторые реализации поддерживают крупнозернистый параллелизм, являясь распределёнными системами, работающими с неразделяемой памятью, и только одна (Blazegraph) – поддерживает среднезернистый параллелизм, используя возможности вычислительных устройств GPU средства технологии CUDA. Однако ни одна из реализованных платформ рассматриваемых языков в явном виде формально не специфицирована, из чего следует, что качественные характеристики языковых платформ необходимо извлекать из естественно-языковых документаций, или из исходных кодов, или при устном общении с разработчиками данных платформ, при условии, что указанные источники доступны для анализа.

Также отметим, что существуют попытки параллельной реализации некоторых операторов поиска языка SPARQL с использованием гетерогенной вычислительной архитектуры (CPU, GPU) [43, 58, 63], однако авторы не

затрагивали в своих исследованиях другие операции, в частности теоретико-множественные.

1.3 Средства формальной спецификации знаний

Для разработки программных систем в соответствии с современными стандартами качества программного обеспечения [10, 98] (например, ISO/IEC 25000:2014), определяющими количественные меры оценки качества программного обеспечения, работающего в тех или иных условиях, необходимы средства описания предъявляемых к системе требований, среди которых, согласно SquaRE (Systems and software Quality Requirements and Evaluation) [98], функциональная пригодность, уровень производительности, надёжность, защищённость и др. Результат описания требований называется спецификацией.

Существуют различные средства формальной записи спецификаций, с помощью которых разрабатываемую систему можно описать с различным уровнем детализации. Кроме этого такие спецификации могут напрямую или опосредованно использоваться далее самой системой в качестве метазнаний о ней самой и хранящихся в ней знаниях. В частности, погружение спецификации операций модели обработки знаний в базу знаний системы, основанной на знаниях, может позволить производить оценку затрат ресурсов и планирование процесса решения задач в данной системе.

К таким средствам относятся алгебраические языки и модели Larch, OBJ, Lotos, CASL, Z-нотация, VDM, сети Петри и др. [94], временны́е логики [37, 47], языки определения данных (например, SQL) [3], языки описания онтологий (OWL, SC-языки) [6, 8, 12, 99] и др.

Среди перечисленных средств спецификации знаний выделяется семейство совместимых sc-языков [6, 8, 12], которые задают модель унифицированного семантического представления знаний, являющейся частным случаем такой модели представления знаний, как семантические сети, и которые используют унифицированный способ семантического кодирования Semantic Computer code (SC-код) [31]. Особенностями SC-кода являются: простой алфавит, содержащий узлы и дуги, простой синтаксис, базовая теоретико-множественная интерпретация. В отличие от других средств спецификации знаний Semantic Code ориентирован на универсальность, т.е. на представление любых видов знаний. Semantic Code является средством унификации представляемых знаний.

1.4 Алгоритмы теоретико-множественных параллельных операций

Согласно архитектурному устройству системы, основанной на знаниях, задача обработки множеств, решаемая на уровнях управления данными и знаниями, нуждается в решении задачи оптимального управления памятью на уровне управления устройствами, так как все рассмотренные в разделе 1.3 динамические структуры данных, использующиеся для представления множеств [18, 19] в конечной памяти, требуют эффективных операций динамического перераспределения памяти [18], от временно́й и пространственной сложностей которых зависят временна́я и пространственная сложности теоретико-множественных операций.

В настоящее время известны системы динамического перераспределения памяти такие, как tlsf [42], ptmalloc2, tcmalloc [67], jemalloc [64], hoard [76], lockless [80] и др. Также известно доказательство оценки худшего случая внешней фрагментации [90]. Система jemalloc основана на системе двойников, однако не даёт строгих характеристик внешней фрагментации для размеров участка памяти больших, чем 2 килобайта. Системы tlsf и tcmalloc используют стратегию лучшего подходящего, которая имеет тенденцию к увеличению количества свободных участков малого размера [18] и может иметь квадратичную оценку пространственной сложности из-за сильно фрагментированной памяти [90]. Оптимальная (по Робсону [90]) оценка производительности приводится для системы hoard, в которой оценка перерасхода памяти по причине внешней фрагментации не превосходит $O\left(\log_2\left(B/b\right)*U\right)$, где U – объём занятой памяти, B, b – максимальный и минимальный размеры занятых участков соответственно. Для большинства других систем не приводится математических оценок ни для времени исполнения операций выделения, высвобождения, перевыделения участков памяти, ни для степени внешней фрагментации. Кроме того, рассмотренные системы не обеспечивают сохранение стратегии выбора подходящего участка для операции перевыделения.

В отличие от перечисленных систем динамического перераспределения памяти временна́я и пространственная сложности для операции выделения/высвобождения в системе [11] оцениваются соответственно как $O\left(\ln^2\left(m\right)*f\left(m\right)\right)$ и $O\left(\ln^3\left(m\right)\right)$, а перерасход памяти по причине внешней фрагментации для этой системы оценивается как $O\left(U*\log_2\left(U\right)\right)$. Однако в данной системе отсутствует операция перевыделения участков памяти.

Основными способами представления конечных множеств в конечной линейно адресуемой памяти относятся следующие структуры данных:

– линейные списки [18];

– линейные связанные списки (одно-и двусвязные, развёрнутые) [18, 24];

– битовые шкалы [24];

– деревья [19];

– хэш-таблицы [19]. Далее по тексту термин «массив» употребляется в значении непрерывного участка памяти, состоящего из последовательных ячеек памяти [11].

Пусть заданы универсум U мощностью w, конечное множество $\mathfrak{A} \subseteq U$ мощностью m и конечное множество $\mathfrak{B} \subseteq U$, мощностью n, где $m \le n \le w$ и взаимно-однозначное соответствие $I \in \left[1,..\left|\left(\mathfrak{A} \cup \mathfrak{B}\right)\right|\right]^{(\mathfrak{A} \cup \mathfrak{B})}$.

Для представления множеств $\mathfrak{A}$ и $\mathfrak{B}$ в виде линейных списков в памяти создаются массивы A и B размерами m и n соответственно такие, что

$$R(l,r,x) = \left((l \le x) \wedge (x \le r)\right),$$
$$P(\alpha,\lambda,s,i) = \exists\gamma\left(\left((\gamma \in \alpha) \wedge (\gamma \ne \lambda)\right) \to \left(s[i] = I(\gamma)\right)\right),$$
$$\forall a \exists i\left((a \in \mathfrak{A}) \to \left(R(1,m,i) \wedge \left(A[i] = I(a)\right) \wedge \left(\neg\left(P(\mathfrak{A},a,A,i)\right)\right)\right)\right),$$
$$\forall b \exists i\left((b \in \mathfrak{B}) \to \left(R(1,n,i) \wedge \left(B[i] = I(b)\right) \wedge \left(\neg\left(P(\mathfrak{B},b,B,i)\right)\right)\right)\right).$$

При таком представлении временна́я сложность операций пересечения и объединения составит $O(m*n)$. В случае, если массивы A и B упорядочить по значению их элементов, то указанные операции можно реализовать путём слияния упорядоченных массивов [24] за время, не превышающее $O(m+n)$. Алгоритмы над данным представлением множеств, ориентированные на последовательную реализацию, рассматривались также в [51, 61] и др., где авторы предлагают различные оптимизации классического алгоритма слиянием, направленные на уменьшение числа сравнений элементов массивов.

К достоинствам представления множеств в виде линейных списков можно отнести простоту реализации соответствующей структуры данных и связанных с ней теоретико-множественных операций, произвольный доступ к элементам массива по индексу за постоянное время, её хорошую приспособленность для выполнения над ней параллельных вычислений. К недостаткам – её линейность, тогда как большинство объектов нелинейны по своей природе, а также необходимость поддерживать упорядоченность элементов массива для обеспечения постоянной или логарифмической временно́й сложности для операции принадлежности элемента множеству и менее чем квадратичной временно́й сложности для других теоретико-множественных операций.

В случае, когда все элементы множества (их уникальные числовые идентификаторы) не хранятся в одном массиве, а хранятся в разных, связанных между собой в цепочку массивах, то говорят о представлении множеств в виде

связанных списков [18]. Алгоритмы теоретико-множественных операций такой структуры данных схожи с алгоритмами операций для линейных списков, однако сама структура данных лишена достоинств линейных списков, за исключением большей гибкости при размещении её в конечной памяти.

Для представления множеств $\mathfrak{A}$ и $\mathfrak{B}$ в виде битовых шкал, когда универсум конечен, в памяти создаются массивы A и B размерами w такие, что

$$\forall a\left(\left((a\in\mathfrak{A})\to\left(A\left[I(a)\right]=1\right)\right)\vee\left(\left(\neg(a\in\mathfrak{A})\right)\to\left(A\left[I(a)\right]=0\right)\right)\right),$$
$$\forall b\left(\left((b\in\mathfrak{B})\to\left(B\left[I(b)\right]=1\right)\right)\vee\left(\left(\neg(b\in\mathfrak{B})\right)\to\left(B\left[I(b)\right]=0\right)\right)\right).$$

При таком представлении пересечение (объединение) множеств $\mathfrak{A}$ и $\mathfrak{B}$ есть множество $\mathfrak{C}$ ($\mathfrak{D}$), которому соответствует массив C (D) такой, что

$$\forall i\left(R(1,w,i)\to\left(C[i]=\left(A[i]\,\&\,B[i]\right)\right)\right),$$
$$\forall i\left(R(1,w,i)\to\left(D[i]=\left(A[i]\,|\,B[i]\right)\right)\right).$$

Другие операции определяются аналогичным образом. Если w не превышает разрядность компьютера ω (размер машинного слова) и в машине присутствуют соответствущие поразрядные операции, то операции над множествами $\mathfrak{A}$ и $\mathfrak{B}$ выполняются за постоянное время $O(1)$. Подобное представление, совмещённое с линейными списками, рассматривалось, например, в работе [56], где w может превышать ω, а временна́я сложность операции пересечения k множеств оценивается как

$$O\left(\sum_{i=1}^{k}\left|\mathfrak{L}_i\right|\Big/\sqrt{\omega}+k*\left|\bigcap_{i=1}^{k}\mathfrak{L}_i\right|\right),$$

где $\mathfrak{L}_i\subseteq U$,

k – количество пересекаемых множеств.

Представление множеств в виде битовых шкал является хорошо приспособленным для выполнения над ним параллельных вычислений, однако оно эффективно, если мощности множеств $\mathfrak{L}_i$ сравнимы с мощностью универсума U, в противном случае данное представление неэффективно с точки зрения потребляемой памяти.

С точки зрения структур данных дерево – есть ациклический связный граф со множеством узлов $\mathfrak{T}$, обладающий следующими свойствами: а) существует один выделенный узел $r\in\mathfrak{T}$, называемый корнем, б) существует

разбиение $\mathfrak{T}$ на $\{r\}$ и множества $\mathfrak{T}_1,...,\mathfrak{T}_k$, в) каждое множество $\mathfrak{T}_i$ $(0 < i \le k)$ является деревом и называется поддеревом дерева $\mathfrak{T}$, г) узлы без поддеревьев называются листовыми. Элементы множества $\mathfrak{A}$ ($\mathfrak{B}$) хранятся в массивах, соответствующих узлам $\mathfrak{T}$.

Известны различные виды деревьев: красно-чёрные, АВЛ, дерамиды (treaps) и др. [18, 54]. В работе [54] рассматриваются алгоритмы операций пересечения и объединения множеств, последовательная реализация которых исполняется за время $\Theta\left(m * \log\left(n/m+1\right)\right)$, а параллельная – за время $O\left(\log\left(m\right) * \log\left(n\right)\right)$ (длина критического пути ярусно-параллельной формы графа алгоритма [41]), когда число процессоров не меньше *m*.

Достоинством представления множеств в виде древовидных структур данных является возможность реализации большинства операций за логарифмическое или линейно-логарифмическое время, зависящее от высоты дерева, а также возможность эффективной параллельной реализации данных операций (согласно [54]). Однако программная реализация деревьев требует более сложной организации работы с памятью и поддержания условия их сбалансированности.

Для представления множества $\mathfrak{A}$ в виде хэш-таблицы (одного из вариантов реализации хэш-таблицы [19]) в памяти создаётся массив *A* некоторого размера *v* и задаётся функция $h \in \left[1,..v\right]^{\mathfrak{A}}$ (хэш-функция) такие, что

$$\forall a\left(\left(a \in \mathfrak{A}\right) \to \left(I\left(a\right) \in A\left[h\left(a\right)\right]\right)\right).$$

Хэш-таблицы представляют компромисс между представлением в виде линейных списков и деревьями (связными списками) и имеют существенный недостаток, заключающийся в том, что при мощности $\mathfrak{A}$ большей значения *v* по принципу Дирихле возникают коллизии, что может требовать пересоздания хэш-таблицы и определения новой *h*.

Фактически существуют системы Blazegraph [53], Kinetica [69], SqreamDB [69], BlazingDB [69], Polymatica [69], Gunrock [73], IrGL [84], ориентированные на параллельные мелко-и среднезернистые вычислительные архитектуры (GPU) и реализующие обработку данных, представленных на основе сетевых и реляционных моделей с использованием средств технологии CUDA. Однако большинство из них (Kinetica, SqreamDB, BlazingDB, Polymatic) являются закрытыми коммерческими системами, не распространяющими информацию об алгоритмах теоретико-множественных операций и деталях их реализации. Система Gunrock в явном виде не содержит базовых теоретико-множественных операций (для неё известна операция соединения – операция реляционной алгебры), а подобная информация для системы IrGL отсутствует.

Известны работы [52, 54, 62, 71, 72], в которых рассматриваются алгоритмы базовых теоретико-множественных операций, ориентированные на параллельную реализацию на гетерогенных вычислительных архитектурах (CPU, GPU).

Выводы

1. Распространённые языки программирования, ориентированные на обработку графовых структур практически одинаково хорошо приспособлены для обработки данных, представленных графовыми конструкциями. Исключение составляют языки, не обладающие вычислительной Тьюринг-полнотой (языки запросов Cypher и SPARQL), а также языки, не поддерживающие на уровне операторов параллельное и асинхронное исполнение программ (GP, PROGRES, языки запросов Cypher и SPARQL).
2. Рассмотренные языки программирования являются языками низкого уровня, тогда как прикладные системы, работающие со знаниями, должны позволять пользователям работать на языке, приближенном к семантике предметной области, оперирующем ее объектами и отношениями. С этой точки зрения рассматриваемые языки программирования, значительно различаются по своим выразительным возможностям и, следовательно, лучшей или худшей приспособленностью для обработки знаний. Наиболее «семантически мощным» [36] языком является язык SCP, так как это подъязык абстрактного графового языка SC-код, обладающего, в частности, рефлексивной и модельно-теоретической семантикой. Отсутствие декларативности у языка SCP компенсируется возможностями создания на базе SC-кода целого семейства совместимых между собой подъязыков. Язык SPARQL, хотя и входит в семейство стандартов технологии Semantic Web (RDF, RDFS, OWL и др. [99]), направленных на представление и обработку знаний, однако, помимо своей семантической ограниченности, также синтаксически и семантически не совместим с другими языками технологии Semantic Web, что затрудняет переход от представления знаний (в том числе процедурных) к их обработке. Аналогичное заключение можно сделать в отношении оставшихся языков программирования GP, Gremlin, PROGRES и язык запросов Cypher, последний из которых не обладает формальной семантикой.
3. Несмотря на то, что все реализации рассматриваемых языков программирования поддерживают мелкозернистый параллелизм (в силу возможностей современных компиляторов и вычислительных архитектур), и на то, что большинство реализаций поддерживает многопоточное исполнение на многоядерных процессорах (CPU) и крупнозернистый параллелизм, являясь распределёнными системами, работающими с неразделяемой памятью, только

одна реализация (Blazegraph) – поддерживает среднезернистый параллелизм потока данных, используя возможности вычислительных устройств GPU.

4. Ни одна из реализованных платформ рассматриваемых языков в явном виде формально не специфицирована (ни на одном из архитектурных уровней управления системы обработки знаний), из чего следует, что качественные характеристики языковых платформ необходимо извлекать из естественно-языковых документаций, или из исходных кодов, или при устном общении с разработчиками данных платформ, при условии, что указанные источники доступны.

ГЛАВА 2
ФОРМАЛЬНАЯ СПЕЦИФИКАЦИЯ ОПЕРАЦИЙ МОДЕЛИ ОБРАБОТКИ ЗНАНИЙ

2.1 Онтологическая модель теоретико-множественных операций модели обработки знаний в конечной памяти

Рассмотрим линейную битовую память *MEMORY*, где «бит» понимается, как эталон хранения информации, способный находиться ровно в двух различных состояниях. Модель обработки знаний состоит из множества состояний STATES линейной битовой памяти *MEMORY* размера *m* и множества унарных операций:

$intersection \in STATES^{STATES}$	–	операция пересечения множеств;
$union \in STATES^{STATES}$	–	операция объединения множеств;
$complement \in STATES^{STATES}$	–	операция разности множеств.

Данные операции не являются деструктивными, т.е. они не изменяют обрабатываемые ими исходные множества. Кроме этого выделяются операции управления памятью:

$allocate \in STATES^{STATES}$	–	операция выделения участка памяти;
$deallocate \in STATES^{STATES}$	–	операция высвобождения участка памяти;
$reallocate \in STATES^{STATES}$	–	операция перевыделения участка памяти;
$read\_cell \in STATES^{STATES}$	–	операция чтения ячейки памяти;
$write\_cell \in STATES^{STATES}$	–	операция чтения ячейки памяти.

Онтологическая модель записана средствами унифицированного семантического кодирования знаний (SCg-кода) и использует модель спецификации знаний в конечных предметных областях, описанную в [12], в которой понятие онтологии рассматривается как тройка $\langle G, R, O \rangle$ [2],

где *G* – непустое конечное множество обозначений спецификации фрагмента,

R – ориентированное конечное множество отношений на обозначениях фрагмента,

O – ориентированное конечное множество функций интерпретации обозначений.

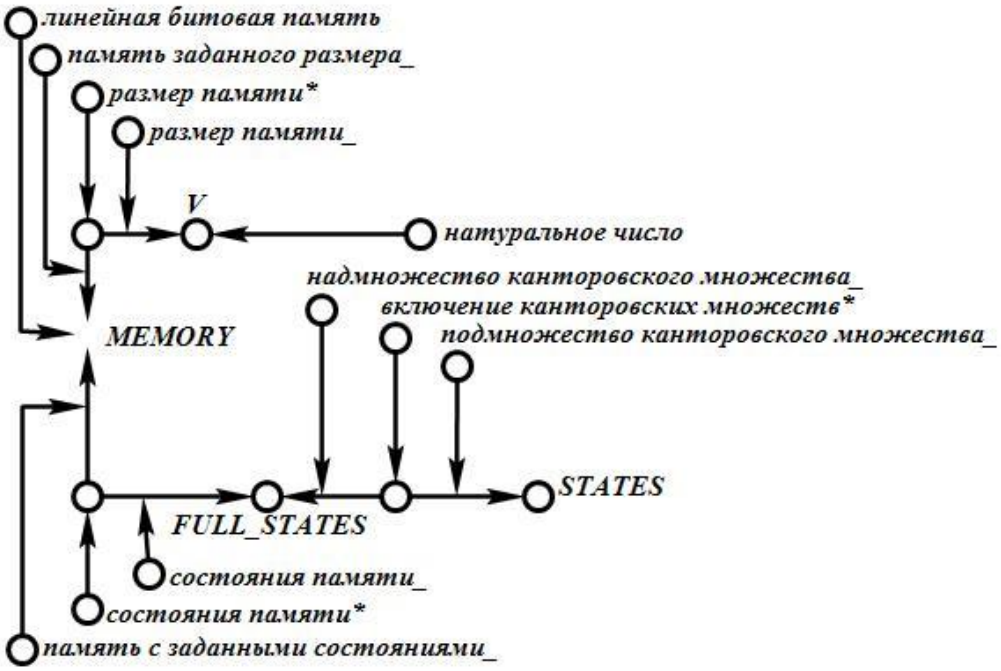


Рисунок 2.1 – SCg запись факта, что память *MEMORY* размера *V* имеет множество состояний *STATES*

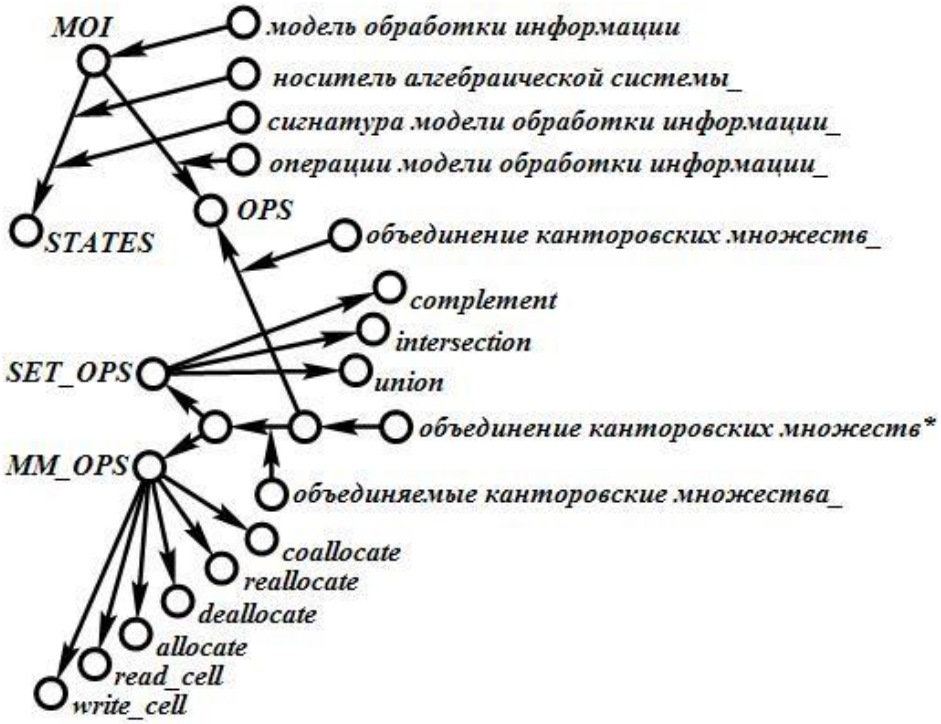


Рисунок 2.2 – SCg запись факта, что существует модель обработки информации *MOI* с заданным носителем и множеством операций (сигнатурой *OPS*)

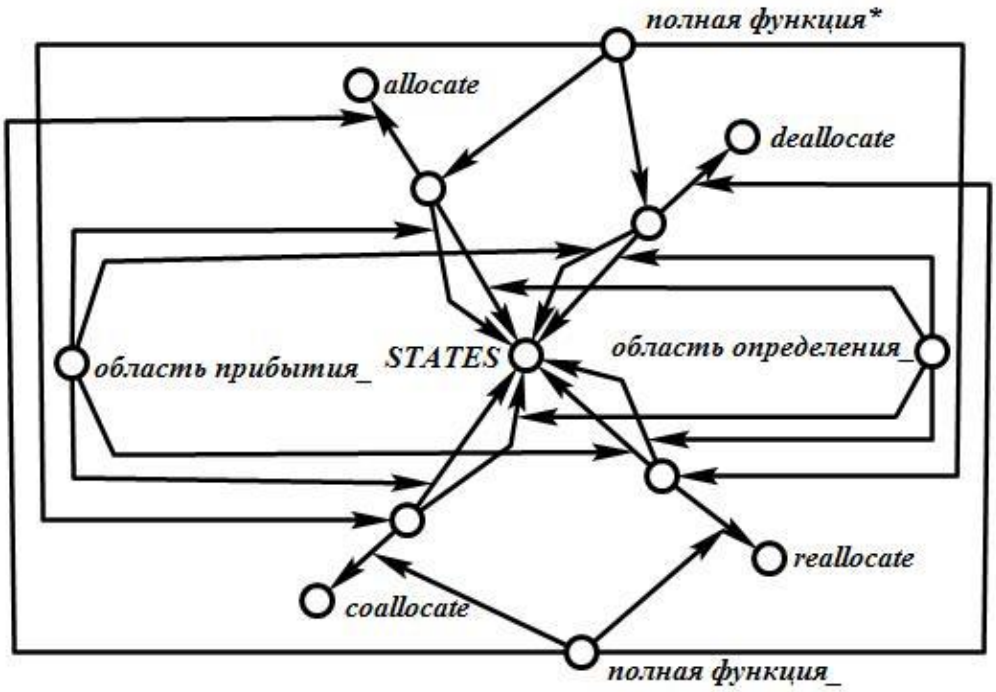


Рисунок 2.3 – SCg запись факта, что операции *allocate*, *reallocate*, *deallocate*, *coallocate* являются полными функциями с областями прибытия и определения *STATES*

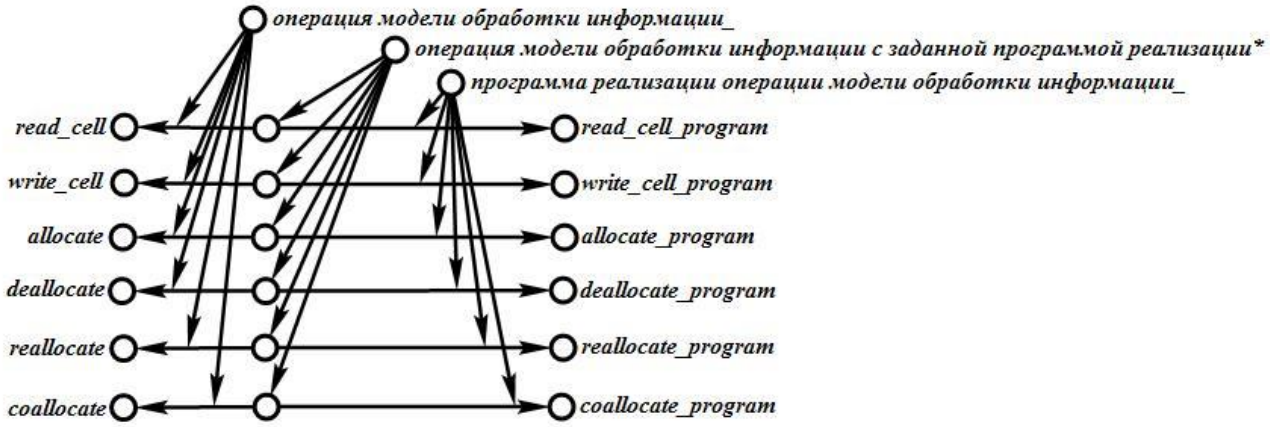


Рисунок 2.4 – SCg запись факта, что операции управления памятью модели обработки информации находятся в отношении «операция модели обработки информации с заданной программой реализации» с конкретной программой

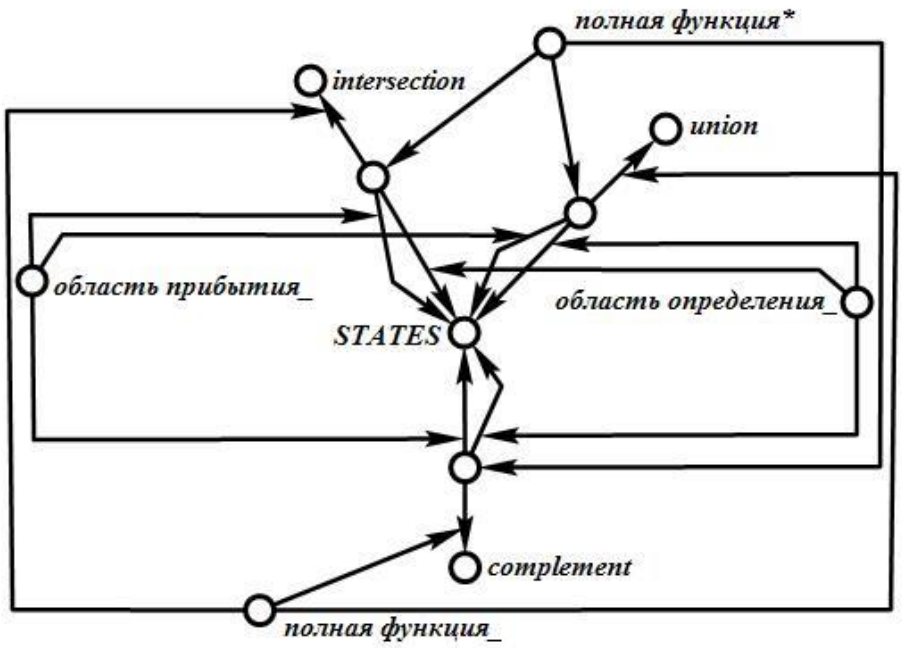


Рисунок 2.5 – SCg запись факта, что операции *intersection*, *union*, *complement* являются полными функциями с областями прибытия и определения *STATES*

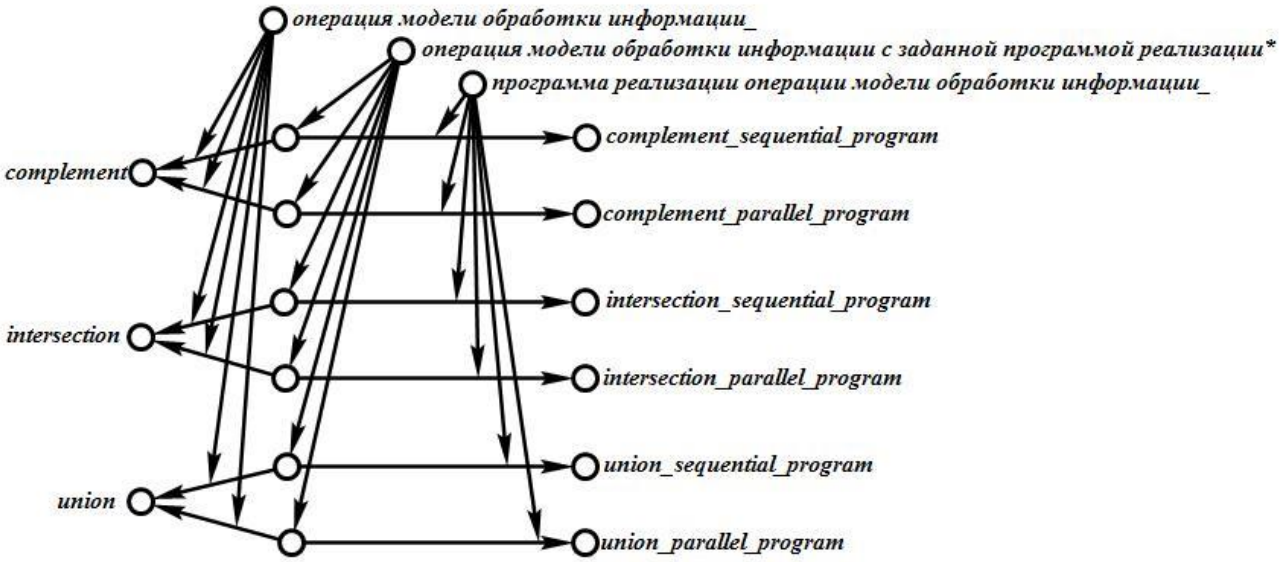


Рисунок 2.6 – SCg запись факта, что теоретико-множественные операции модели обработки информации находятся в отношении «операция модели обработки информации с заданной программой реализации» с конкретной программой

2.2 Модель и алгоритмы стратегий перераспределения участков линейно адресуемой битовой памяти

Рассмотрим конечную линейно адресуемую битовую память *MEMORY* размера *m* (обычно $m = 2^p$) ячеек. Пусть время доступа (чтения/записи) к одной ячейке памяти есть значение функции *f*(*m*).

Память *MEMORY* имеет конечное множество состояний *STATES*. Для линейной битовой памяти размера *m*:

$$STATES \subseteq \{0,1\}^m.$$

Каждая ячейка памяти разрядности $w \le m$ однозначно идентифицируется адресом из множества адресов:

$$INDEX = \{0,1\}^w.$$

Выделяется множество адресов:

$$WELL\_INDEX = INDEX \setminus \{NULL\_INDEX\},$$

где $NULL\_INDEX \in INDEX$.

Определим следующие операции:

$allocate \in STATES^{STATES}$ – операция выделения участка памяти;
$deallocate \in STATES^{STATES}$ – операция высвобождения участка памяти;
$reallocate \in STATES^{STATES}$ – операция перевыделения участка памяти;
$processing \in STATES^{STATES}$ – операция обработки данных.

$$processing = processing_+ \cup processing_- \cup processing_=;$$

$processing_+ \in STATES^{STATES}$ – операция обработки данных, добавляющая новые данные в несвободный участок памяти;
$processing_- \in STATES^{STATES}$ – операция обработки данных, удаляющая старые данные из несвободного участка памяти;
$processing_= \in STATES^{STATES}$ – операция обработки данных, не изменяющая количество занятых ячеек несвободного участка памяти.

Две базовые операции – выделение и высвобождение – необходимы для управления свободными и несвободными участками памяти, где не происходит доступ к данным [11]. Операция перевыделения, исполняется как на уровне управления устройствами, так и на уровне управления данными и используется для управления участками памяти, в которых содержатся данные соответствующей нестационарной структуры данных. При перевыделении участка памяти может происходить копирование массива данных из одного участка памяти в другой, что соответствует исполнению операции обработки данных. Операция обработки данных исполняется на уровне управления данными и работает с занятыми и незанятыми ячейками несвободных участков памяти.

Определим функцию множества адресов всех несвободных участков памяти, находящейся в одном из состояний:

$$used \in \left(2^{INDEX}\right)^{STATES}.$$

Определим функцию множества адресов всех занятых участков памяти, находящейся в одном из состояний:

$$busy \in \left(2^{INDEX}\right)^{STATES}.$$

Пусть задан профиль операций обработки данных длины n:

$$processings \in processing^{[1,..n]}.$$

Пусть задано соответствие, определяющее множество решений о переходах между состояниями (множество управляющих воздействий):

$$decision \in STATES^{[1,..n]}.$$

Сформулируем задачу оптимального управления обработкой данных, хранящихся в линейно адресуемой памяти со множеством состояний *STATES*. Пусть уравнение состояния задаётся формулой (2.1):

$$\begin{aligned} &reallocate\_strategy = reallocate\_strategy \cap reallocate, \\ &decision(i+1) = processing(i)\left(reallocate\_strategy\left(decision(i)\right)\right). \end{aligned} \quad (2.1)$$

Пусть критерий качества задаётся формулой (2.2):

$$S\left(decision\right)=\sum_{i=1}^{n}\frac{T\left(reallocate,decision\left(i\right)\right)+T\left(processings\left(i\right),decision\left(i\right)\right)}{\sum_{t=1}^{n}\left|busy\left(decision\left(t\right)\right)\setminus busy\left(decision\left(t-1\right)\right)\right|},\quad(2.2)$$

где T – функция времени исполнения операции над множеством состояний памяти.

Необходимо определить такое соответствие *decision*, которые бы минимизировало функционал *S*(*decision*):

$$S\left(decision\right)=\min\left(S\left(d\right),d\right).$$

Далее приведём опишем подход, позволяющий получить приближённое решение поставленной задачи для случая, когда профиль операций обработки данных заранее неизвестен.

Пусть размер участка памяти монотонно возрастает с 1 до $n \le m$ за n операций перевыделения. Если цепочка состоит из последовательных операций выделения нового участка памяти, копирования данных в новый участок памяти и высвобождения старого участка памяти, то временная сложность n операций перевыделения в худшем случае оценивается как $O\left(n^2 * f\left(m\right)\right)$, а сложность одной операции перевыделения – $O\left(c * f\left(m\right)\right)$, где $c = 2,..n$.

Временну́ю сложность можно понизить, выделяя участок бóльшего, чем требуется, размера. Это позволит избежать копирования данных для некоторого числа последующих операций перевыделения для выделенного участка памяти (такое решение используется, например, в [67] и других системах динамического перераспределения памяти, а также рассматривается в работах [1, 18, 89]).

Для этого фактический размер g(*w*)(*v*) выделяемого участка памяти поставим в соответствие g(*w*) (2.3) каждому затребованному размеру *v* ($v \le$ g(*w*)(*v*) $\le m$) согласно некоторому правилу *r*, где *w* – старый размер выделенного участка памяти:

$$g\left(w\right)\left(v\right)=\begin{cases} r_3 \left| \left(r_1 \le r_2\right) \wedge \left(v \ge r_1\right) \wedge \left(v \le r_2\right) \wedge \left(v > w\right) \\ r_4 \left| \left(r_1 \le r_2\right) \wedge \left(v \ge r_1\right) \wedge \left(v \le r_2\right) \wedge \left(v < w\right) \\ r_5 \left| \left(r_1 > r_2\right) \wedge \left(v > r_2\right) \wedge \left(v \le r_1\right) \wedge \left(w \ge r_3\right) \wedge \left(w \le r_4\right) \right. \end{cases}.\quad(2.3)$$

Правило *r* задаётся пятёркой $\left\langle r_1, r_2, r_3, r_4, r_5 \right\rangle$,
где при $r_1 \le r_2$

r_1 – минимальное значение затребованного размера выделяемого участка,

r_2 – максимальное значение затребованного размера выделяемого участка,

r_3 – фактический размер участка для выделения,

r_4 – фактический размер участка для выделения при перевыделении участка меньшего размера,

r_5 – флаг наличия гистерезиса;

а при $r_1 > r_2$

r_1 – минимальное значение затребованного размера выделяемого участка,

r_2 – максимальное значение затребованного размера выделяемого участка,

r_3 – минимальное значение старого размера выделенного участка,

r_4 – максимальное значение старого размера выделенного участка,

r_5 – фактический размер участка для выделения или выделения при перевыделении участка.

Стратегия распределения участков памяти есть множество S правил r.

Выделяя участок бóльшего, чем требуется размера для случая монотонно возрастающего размера участка можно получить среднюю амортизационную стоимость одной операции перевыделения равную $O(1)$ (к примеру, для $g(w)(v)=\min(2^{\lceil \log_2 v \rceil}, m)$). При этом придётся затратить дополнительный объём памяти.

Однако вообразим ситуацию немонотонного изменения размера участка памяти, в которой за каждым увеличением следует уменьшение размера участка памяти, и наоборот, а для каждой операции перевыделения происходит копирование данных. Избежать такой худший случай можно, используя стратегию с гистерезисом S_h. Этот тип стратегий, при уменьшении размера участка памяти с y до v, даёт значение $g(y)(v) > g(w)(v)$, где $g(w)(v)$ – фактический размер участка памяти при увеличении размера участка памяти с w до v. Значение $g(y)(v)$ определяется в соответствии с шагом гистерезиса h. Единичный шаг означает, что $g(y)(v)$ выбирается минимальным из бóльших фактических размеров участков.

Пусть правило r для v находится в стратегии S некоторой функцией поиска. Предлагается два способа реализации функции поиска: на основе хранимых правил и на основе метаправил. Первый способ сводится к бинарному поиску хранящихся в памяти правил. При втором способе искомое правило r формируется путём вычисления некоторых его компонент по формуле.

Рассмотрим правила $r_1 \le r_2$. Для них применима формула (2.4), которая в общем виде задаёт зависимость между исходным фактическим размером x_i участка памяти и j-ым бóльшим фактическим размером x_{i+j} перевыделяемого участка:

$$x_{i+j} = \left(k^j * \sqrt[q]{x_i} + b * \sum_{l=0}^{j-1} k^l \right)^q, \tag{2.4}$$

где $k \in \mathbb{R}^+$,
$b \in \mathbb{N}$,
$q \in \mathbb{R}^+$ – постоянные параметры.

Таким образом, формула (2.4) позволяет вычислять некоторые последовательности натуральных чисел. Каждый i-ый член последовательности определяет i-ое правило как $\left\langle x_{i-1}+1, x_i, x_i, x_{i+h}, \operatorname{sgn}(h) \right\rangle$.

Из (2.4) можно получить оценки для затрат времени и дополнительного объёма памяти. В стратегиях с гистерезисом в один шаг удельное время добавления одного элемента данных (и последующего его удаления) в связный участок бóльшего размера выражается формулой (2.5):

$$\frac{\tau_a + \tau_d + f(m) * \left(x_i + x_{i+1} \right)}{x_{i+1} - x_i} = \frac{\tau_a + \tau_d + f(m) * \left(x_i + \left(k * \sqrt[q]{x_i} + b \right)^q \right)}{\left(k * \sqrt[q]{x_i} + b \right)^q - x_i}, \tag{2.5}$$

где τ_a – время расширения участка памяти,
τ_d – время сокращения участка памяти.

При $k > 1$ (геометрическая прогрессия), в худшем случае удельное время оценивается как (2.6):

$$\lim_{i \to \infty} \frac{\tau_a + \tau_d + f(m) * \left(x_i + \left(k * \sqrt[q]{x_i} + b \right)^q \right)}{\left(k * \sqrt[q]{x_i} + b \right)^q - x_i} = \\ = f(m) * \left(\frac{k^q + 1}{k^q - 1} \right) + \frac{1}{k^q - 1} * \lim_{i \to \infty} \left(\frac{\tau_a + \tau_d}{x_i} \right) \tag{2.6}$$

Тогда для добавления n элементов данных с учётом возможных удалений как уже имеющихся, так и добавленных данных, справедлива временна́я сложность $O\left(f(m) * n + d * \tau \right)$, где d – количество перевыделений, τ – время

одного перевыделения. Если время доступа к ячейке памяти $O(1)$, то временна́я сложность зависит линейно от n при $n \geq d * \tau$. Таким образом, среднее амортизационное время добавления одного экземпляра данных в участок памяти постоянно, если время доступа к ячейке памяти в зависимости от размера памяти постоянно, а количество записей данных в ячейки памяти зависит от произведения времени одного перевыделения на количество перевыделений как минимум прямо пропорционально.

Отношение размеров исходного участка и наименьшего из бо́льших размеров перевыделяемых участков выражается формулой (2.7):

$$\frac{x_{i+1}}{x_i} = \frac{\left(k * \sqrt[q]{x_i} + b\right)^q}{x_i}, \alpha = \lim_{i \to \infty} \frac{x_{i+1}}{x_i} = k^q. \quad (2.7)$$

Из (2.7) можно оценить дополнительные затраты запасной памяти как α. Для добавления одного элемента данных в участок размером x_i может потребоваться в k^q раз больше памяти, чем было изначально. Постоянные k, b, q могут быть подобраны таким образом, что при сокращении затрат памяти временна́я сложность не будет превосходить линейно-логарифмическую оценку.

Преимущество использования хранимых правил в гибкости, так как они позволяют определять стратегии, которые нельзя определить с помощью формирования правил по формуле (2.4). Однако для хранения правил необходим дополнительный объём памяти. При этом следует брать во внимание зависимость бинарного поиска от числа правил. Метаправила предпочтительны для сохранения объёма памяти, а также в случае их аппаратной реализации.

На рисунках 2.1–2.3 изображены блок-схемы алгоритмов операций выделения, высвобождения и перевыделения участков памяти, основанные на использовании модели стратегий перераспределения участков памяти. Блок-схемы соответствующих предопределённых подпрограмм изображены на рисунках Б.1–Б.6.

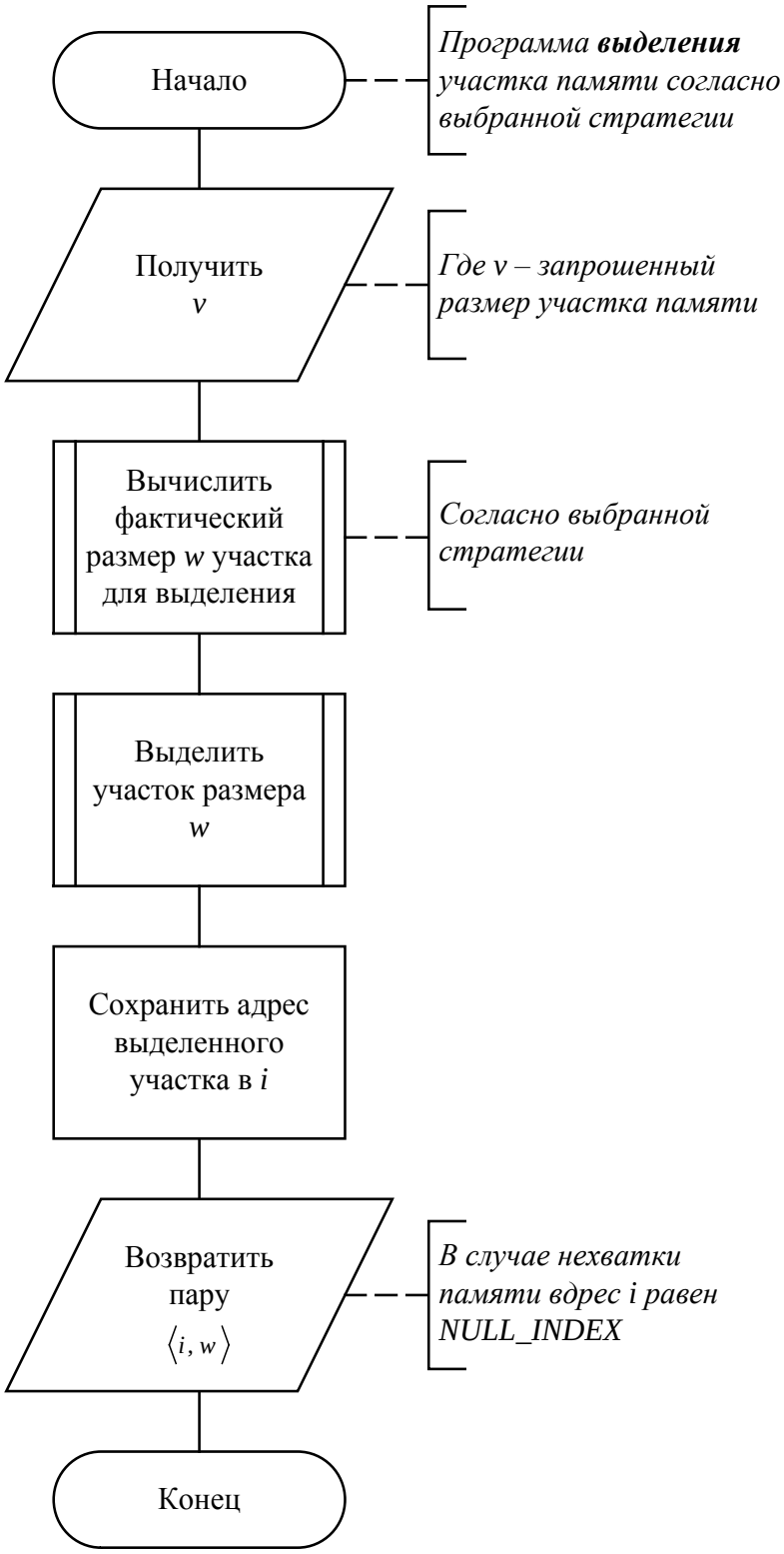


Рисунок 2.1 – Блок-схема алгоритма операции выделения участка памяти согласно выбранной стратегии

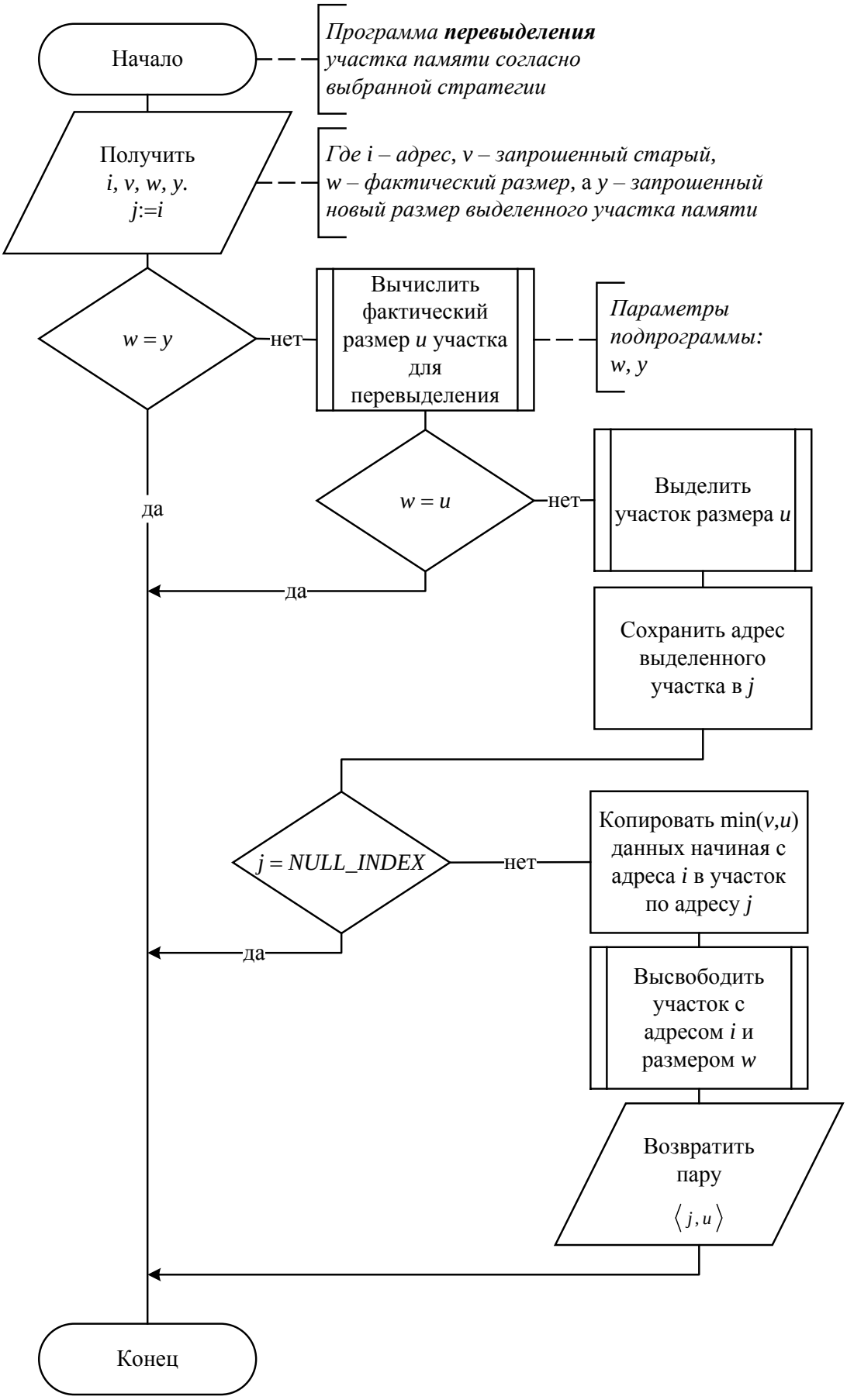


Рисунок 2.2 – Блок-схема алгоритма операции перевыделения участка памяти согласно

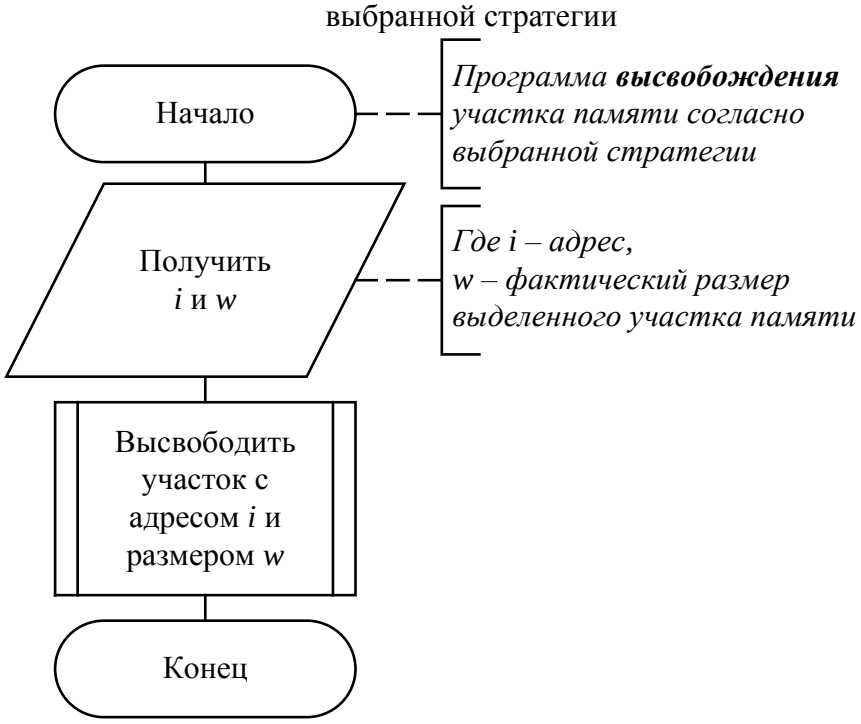


Рисунок 2.3 – Блок-схема алгоритма операции высвобождения участка памяти согласно выбранной стратегии

2.3 Алгоритмы теоретико-множественных операций параллельной обработки знаний

Пусть U – конечный универсум мощностью w, а $I \in [1,..w]^U$ – взаимно-однозначное функциональное соответствие.

Пусть $\hat{\mathfrak{A}} = \langle \mathfrak{A}, \pi_1 \rangle$ – конечное мультимножество мощностью m, где $\mathfrak{A} \subseteq U$, $\pi_1 \in [1,..m]^{\mathfrak{A}}$.

Пусть $\hat{\mathfrak{B}} = \langle \mathfrak{B}, \pi_2 \rangle$ – конечное мультимножество мощностью n, где $\mathfrak{B} \subseteq U$, $\pi_2 \in [1,..n]^{\mathfrak{B}}$, $m \le n$.

Мощность мультимножества:

$$\left\| \hat{\mathfrak{A}} \right\| = \sum_a^{a \in \mathfrak{A}} \pi_1(a) = m.$$

Определим операцию пересечения мультимножеств как

$$\hat{\mathfrak{A}} \hat{\cap} \hat{\mathfrak{B}} = \left\langle (\mathfrak{A} \cap \mathfrak{B}), \pi_{\hat{\cap}} \in [1,..m]^{(\mathfrak{A} \cap \mathfrak{B})} \right\rangle,$$

где соответствие $\pi_{\hat{\cap}}$ обладает свойством

$$\forall c\Big(\big(c\in(\mathfrak{A}\cap\mathfrak{B})\big)\to\big(\pi_{\hat{\cap}}(c)=\min\big(\pi_1(c),\pi_2(c)\big)\big)\Big).$$

Определим операцию объединения мультимножеств как

$$\hat{\mathfrak{A}}\,\hat{\cup}\,\hat{\mathfrak{B}}=\Big\langle(\mathfrak{A}\cup\mathfrak{B}),\pi_{\hat{\cup}}\in\big[1,..(m+n)\big]^{(\mathfrak{A}\cup\mathfrak{B})}\Big\rangle,$$

где соответствие $\pi_{\hat{\cup}}$ обладает свойством

$$\forall c\Big(\big(c\in(\mathfrak{A}\cup\mathfrak{B})\big)\to\big(\pi_{\hat{\cup}}(c)=\max\big(\pi_1(c),\pi_2(c)\big)\big)\Big).$$

Пусть в памяти хранятся массивы *A* и *B* размеров *m* и *n* такие, что

$$R(l,r,x)=\big((l\le x)\wedge(x\le r)\big),$$
$$P(\alpha,\lambda,s,i)=\exists\gamma\Big(\big((\gamma\in\alpha)\wedge(\gamma\ne\lambda)\big)\to\big(s[i]=I(\gamma)\big)\Big),$$
$$\forall a\exists i\forall j\Big(\big(a\in\mathfrak{A}\big)\to\big(R(1,m,i)\wedge R\big(i,i+\pi_1(a),j\big)\wedge\big(A[j]=I(a)\big)\wedge\big(\neg\big(P(\mathfrak{A},a,A,j)\big)\big)\big)\Big),$$
$$\forall b\exists i\forall j\Big(\big(b\in\mathfrak{B}\big)\to\big(R(1,n,i)\wedge R\big(i,i+\pi_2(b),j\big)\wedge\big(B[j]=I(b)\big)\wedge\big(\neg\big(P(\mathfrak{B},b,B,j)\big)\big)\big)\Big).$$

Массивы *A* и *B* упорядочены

$$\forall i\forall j\Big(\big(R(1,m,i)\wedge R(1,m,j)\wedge(i\le j)\big)\to\big(A[i]\le A[j]\big)\Big),$$
$$\forall i\forall j\Big(\big(R(1,n,i)\wedge R(1,n,j)\wedge(i\le j)\big)\to\big(B[i]\le B[j]\big)\Big).$$

Алгоритм операции пересечения мультимножеств:

1. Выполнить поиск минимального и максимального индексов вхождения каждого элемента массива *A* в массив *B*:

⟨*L*, *U*⟩ ← *RangeBinarySearch*(⟨*A*, *B*⟩).

2. Отметить в массиве *C* минимальные индексы равных элементов массива *A*:

$$C[i]\leftarrow\begin{cases}i\,\Big|\,A\big[\max\big(\{0\}\cup\{i-1\}\big)\big]<A[i]\\0\,\Big|\,A\big[\max\big(\{0\}\cup\{i-1\}\big)\big]=A[i]\end{cases}.$$

3. Вычислить максимумы $D \leftarrow MaxScatter(C)$:

$k \leftarrow ((\sim 0) >> (\text{clz}(Length(A)) + 1)) + 1$
пока $(k > 0)$:
    если $(i \geq k)$, *то* $D[i] \leftarrow \max(\{D[i]\} \cup \{D[i-k]\})$
    $k = k >> 1$

4. Вычислить

$$E[i] \leftarrow \begin{cases} 1 \mid i - D[i] < U[i] - L[i] \\ 0 \mid i - D[i] \geq U[i] - L[i] \end{cases}.$$

5. Вычислить массив F префиксных сумм массива E:

$F[i] \leftarrow E[i] + PrefixSum(E)[i]$.

6. Вычислить:

если $(E[i]=1)$, *то* $R[F[i] - 1] \leftarrow A[i]$.

7. Возвратить $\langle R, F[Length(A) - 1] \rangle$.

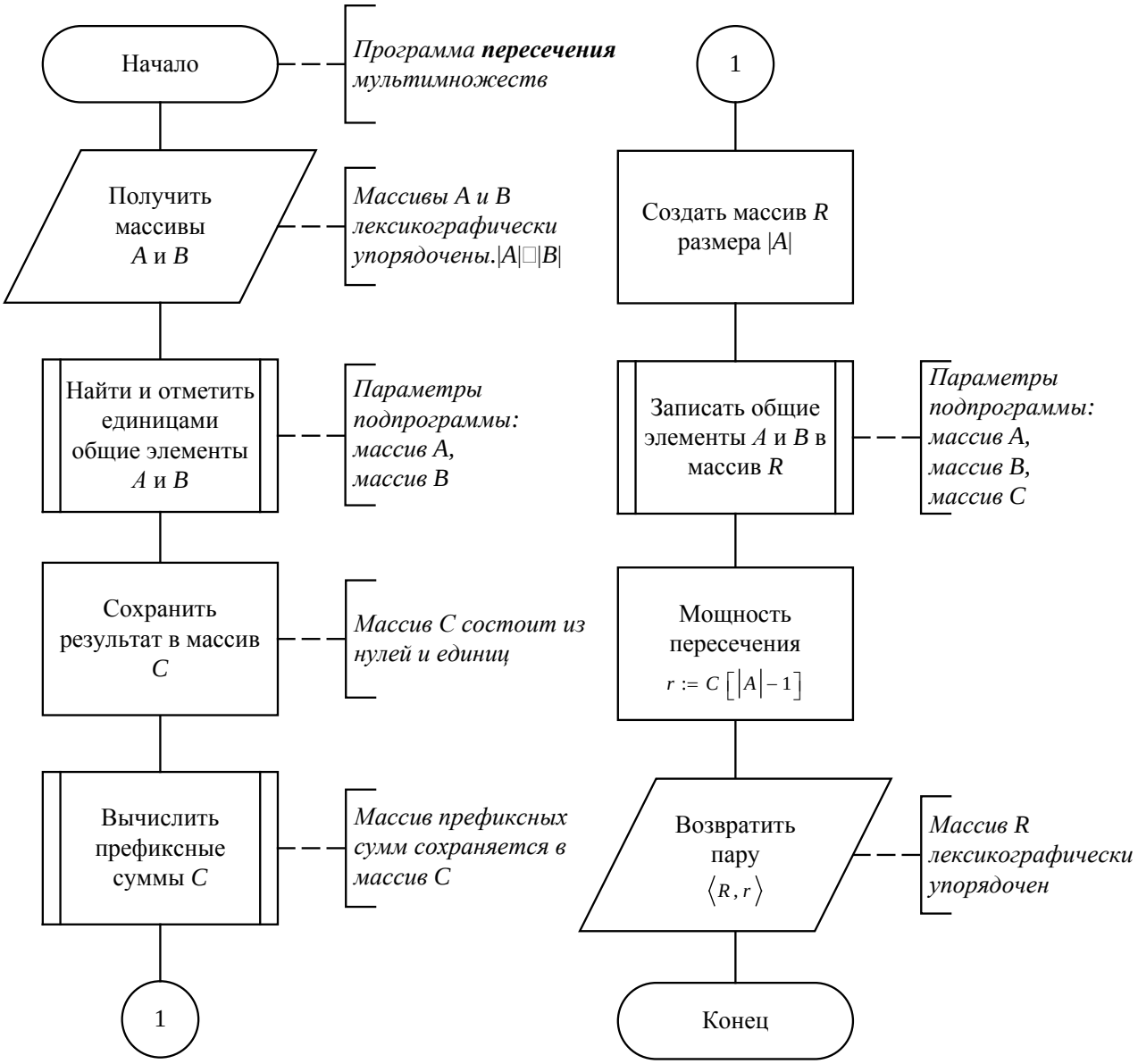


Рисунок 2.4 – Блок-схема алгоритма операции пересечения мультимножеств

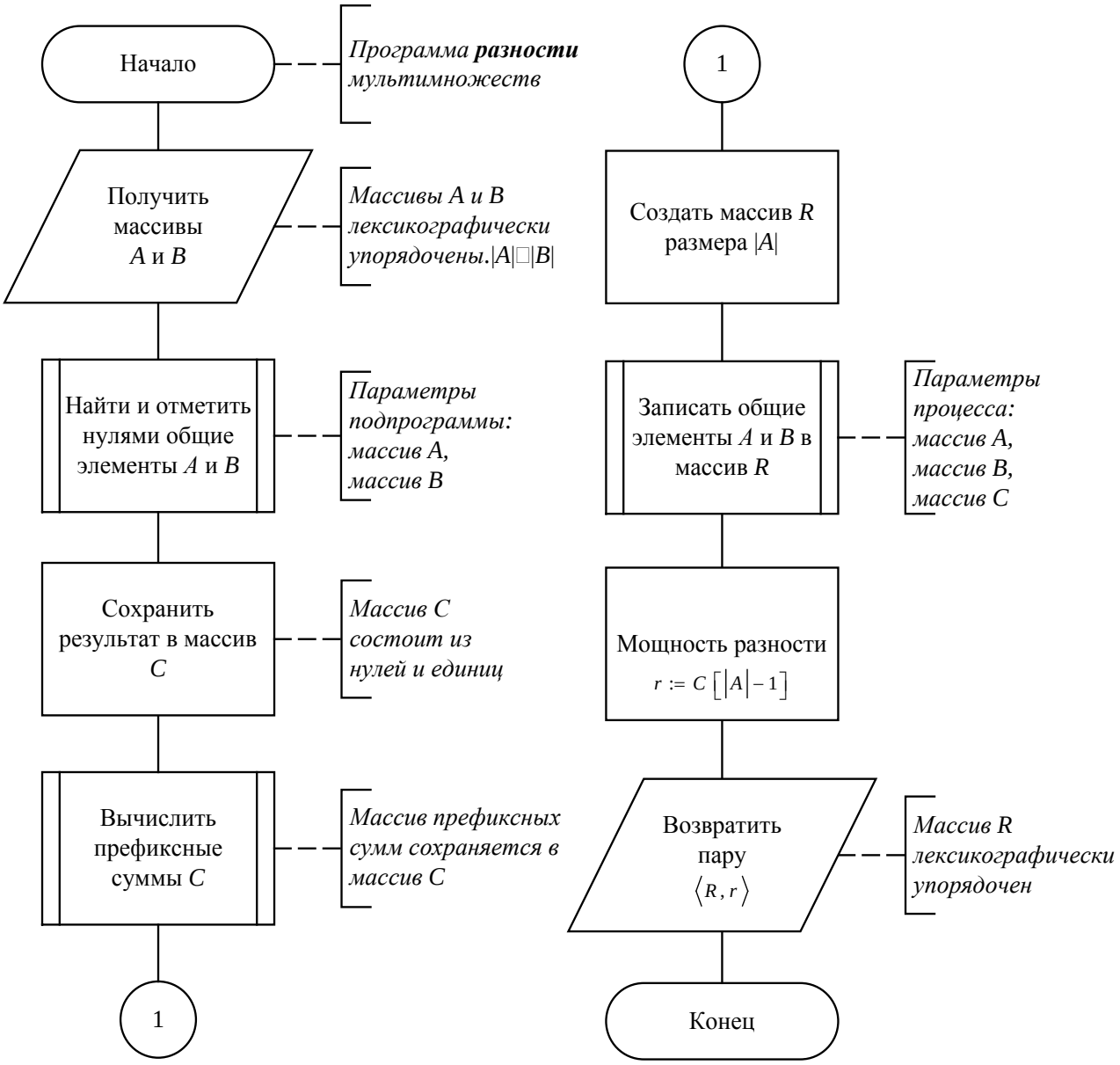


Рисунок 2.5 – Блок-схема алгоритма операции разности мультимножеств

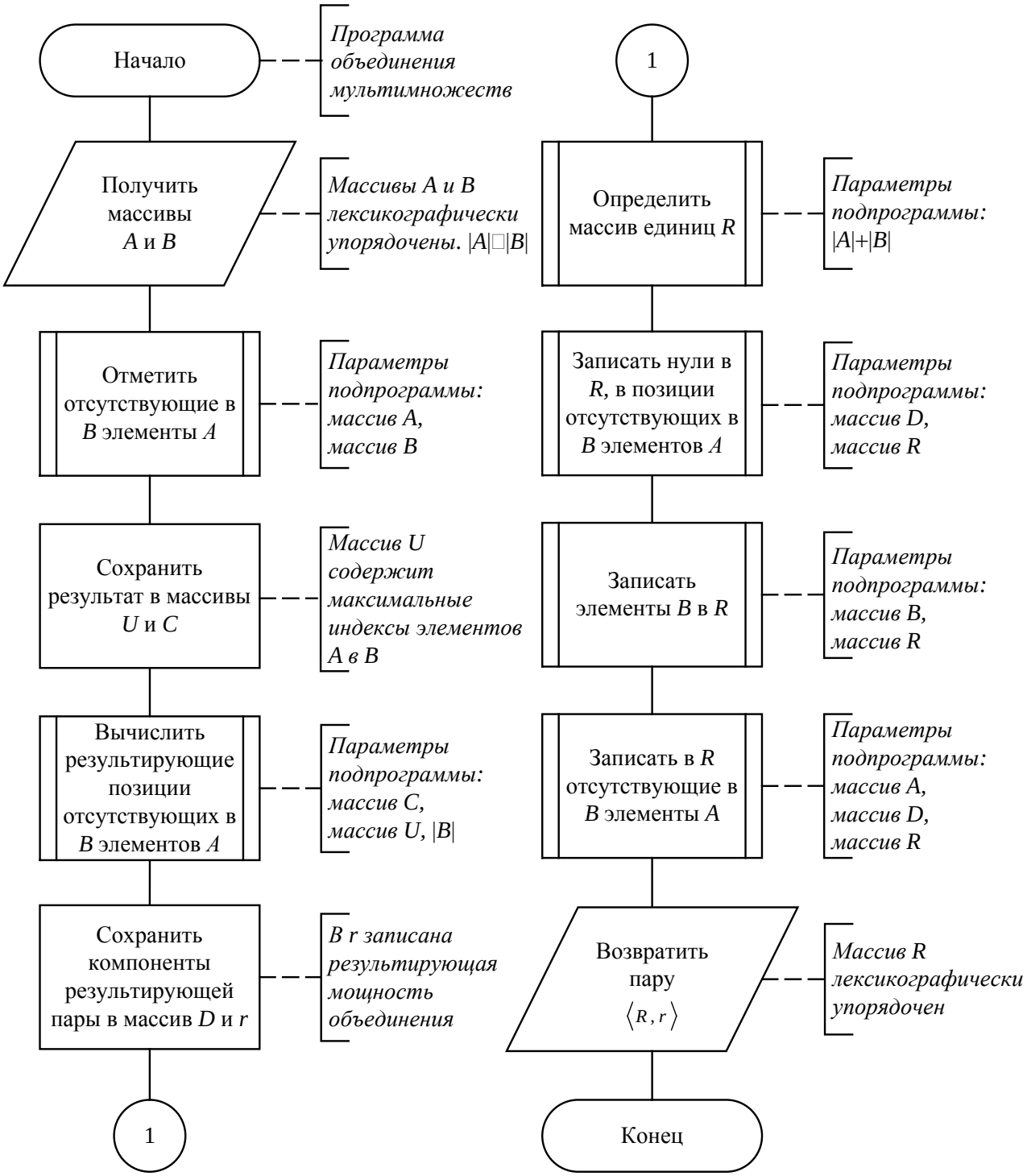


Рисунок 2.6 – Блок-схема алгоритма операции объединения мультимножеств

Алгоритм операции объединения мультимножеств:

1. Выполнить поиск минимального и максимального индексов вхождения каждого элемента массива A в массив B:

$\langle L, U\rangle \leftarrow RangeBinarySearch(\langle A, B\rangle)$.

2. Отметить в массиве C минимальные индексы равных элементов массива A:

$$C[i] \leftarrow \begin{cases} i \,\middle|\, A\left[\max\left(\{0\} \cup \{i-1\}\right)\right] < A[i] \\ 0 \,\middle|\, A\left[\max\left(\{0\} \cup \{i-1\}\right)\right] = A[i] \end{cases}.$$

3. Вычислить максимумы $D \leftarrow MaxScatter(C)$:
$k \leftarrow ((\sim 0) >> (\text{clz}(Length(A))+1))+1$
пока $(k > 0)$:
  если $(i \geq k)$, *то* $D[i] \leftarrow \max(\{D[i]\} \cup \{D[i-k]\})$
  $k >>= 1$
4. Вычислить

$$G[i] \leftarrow \begin{cases} 0 \,\middle|\, i - D[i] < U[i] - L[i] \\ 1 \,\middle|\, i - D[i] \geq U[i] - L[i] \end{cases}.$$

5. Вычислить массив H префиксных сумм массива G:
$H \leftarrow PrefixSum(G)$.
6. Выполнить для каждого элемента массива H:
если $(((i=0) \wedge (H[i]=1)) \vee ((i>0) \wedge (H[i]=H[i-1]+1)))$, *то* $E[i] \leftarrow H[i]+U[i]$
иначе $E[i] \leftarrow 0$.
7. Вычислить
для i от 0 до $Length(A) + Length(B) - 1$: $R[i] \leftarrow 1$.
8. Выполнить для каждого элемента массива E:
если $(E[i]>0)$, *то* $R[E[i]-1] \leftarrow 0$
9. Вычислить массив T префиксных сумм массива R:
  $T[i] \leftarrow R[i]+PrefixSum(R)[i]$; $T[-1] \leftarrow 0$.
10. Выполнить для каждого элемента $(i \geq 0)$ массива T:
если $((T[i] \leq Length(B)) \wedge (T[i-1] <> T[i]))$, *то* $R[i] \leftarrow B[T[i]-1]$.
После этого массив R содержит все элементы массива B.
11. Выполнить для каждого элемента массива A:
если $((i<Length(A)) \wedge (E[i]>0))$, *то* $R[E[i]-1] \leftarrow A[i]$.
После этого массив R содержит и элементы массива A.
12. Возвратить
$\langle R, H[Length(A)-1] + Length(B) \rangle$.

Теоретическая оценка временно́й сложности приведённых алгоритмов с учётом временны́х затрат на управление памятью не превосходит

$$O\left(\log(n) * n/p + \ln^2(m) * f(m)\right),$$

где $n = Length(A)+Length(B)$ – суммарная длина входных массивов,
p – количество процессоров.

Выводы

В соответствии с проведённым анализом различных формальных моделей обработки информации разработана формальная спецификация теоретико-множественных операций модели обработки знаний в конечной линейно адресуемой памяти. Формальная спецификация состоит из трёх частей, согласующихся с общим принципом архитектурного устройства машины обработки знаний:

1. Сформулирована обобщённая задача оптимального управления памятью с конечным числом состояний на уровне управления устройствами и данными системы обработки знаний. Решение данной задачи заключается в минимизации функционала обратного пропускной способности операции перевыделения участков памяти. Для приближённого решения поставленной задачи в условиях, когда заранее неизвестен профиль операций обработки данных, управляющие воздействия (решения о фактическом перевыделении нового участка памяти) производятся в соответствии с разработанной моделью стратегий перераспределения участков линейно адресуемой памяти. Носителем модели является множество состояний памяти и множество правил переходов между состояниями; сигнатурой – отношения между состояниями памяти и правилами переходов. Предложены алгоритмы реализации модели стратегий. Среднее амортизационное время работы алгоритма операции перевыделения участков памяти (добавления или удаления данных в выделенный участок памяти) является постоянной величиной, если время доступа к ячейке памяти в зависимости от размера памяти постоянно, а количество записей данных в ячейки памяти зависит как минимум прямо пропорционально от произведения времени одного перевыделения на количество перевыделений.

2. Основываясь на теоретических оценках временны́х и пространственных характеристик уровня управления устройствами и данными, обеспечиваемых моделью и алгоритмами стратегий перераспределения участков линейно адресуемой памяти, на уровне управления данными разработаны алгоритмы теоретико-множественных операций параллельной обработки знаний. Данные алгоритмы отличаются поддержкой мультимножеств и ориентацией на использование параллелизма потока данных и реализацию на мелко-и среднезернистой параллельной вычислительной архитектуре с произвольным доступом к общей памяти. Теоретическая оценка временной сложности данных алгоритмов при использовании средств модели стратегий перераспределения участков линейно адресуемой битовой памяти не превышает $O\left(\log\left(n\right)*n/p+\ln^2\left(m\right)*f\left(m\right)\right)$,

где n – суммарная мощность множеств,
p – количество процессоров,
m – размер памяти,

$f(m)$ – значение функции времени доступа (чтения/записи) к одной ячейке памяти.

3. Разработана онтологическая модель теоретико-множественных операций модели обработки знаний, которая специфицирует машину обработки информации формальной теоретико-множественной модели [31] в конечной памяти, т.е. памяти с конечным числом состояний. Построенная онтологическая модель является подмоделью модели спецификации знаний, описанной в [12], и записана средствами унифицированного семантического кодирования знаний (SC-кода). Полученная спецификация описывает одну конкретную модель обработки информации, однако может быть расширена на целый класс моделей при использовании языка логики первого порядка. Кроме этого данная спецификация может напрямую или опосредованно использоваться системой обработки знаний в качестве метазнаний о её собственных характеристиках. В частности, погружение данной спецификации операций в базу знаний системы, основанной на знаниях, может позволить производить оценку затрат ресурсов и планирование процесса решения задач в данной системе.

ГЛАВА 3
РЕАЛИЗАЦИЯ И ЭКСПЕРИМЕНТАЛЬНАЯ ПРОВЕРКА ОПЕРАЦИЙ МОДЕЛИ ОБРАБОТКИ ЗНАНИЙ

3.1 Операция перевыделения участков линейно адресуемой памяти

Модель и соответствующие алгоритмы стратегий перераспределения участков памяти были реализованы на языке C++ с использованием программной реализации системы динамического управления памятью [11] (назовём её em).

Экспериментальная проверка осуществлялась для четырёх стратегий (таблица 3.1). Компоненты *i*-го правила каждой из стратегий определялись согласно формуле (2.2).

Таблица 3.1 – Описание стратегий для тестирования

Наименование стратегии	Вид зависимости	Значения констант
геометрическая	$x_i = 2^i$	k=2, b=0, q=1
степенная	$x_i = (1 + 580 * i)^2$	k=1, b=580, q=2
линейная (а)	$x_i = 17 * i + 1$	k=1, b=17, q=1
линейная (б)	$x_i = 4133222 * i + 1$	k=1, b=4133222, q=1

Объём памяти $m = 9 * 2^{24}$ ячеек. Размер ячейки памяти 32 бита. Пространство признаков для тестирования включает:

– старый размер l выделенного участка памяти;

– новый размер u выделяемого участка памяти;

– время работы t операции перевыделения участка с размера l на u;

– шаг гистерезиса $h \in \{0,1\}$;

– способ реализации функции поиска.

Пара значений $\langle l, u \rangle \in \mathfrak{V} \times \mathfrak{V}$, где

$$\mathfrak{V} = \{2097151 * j + 1 \mid j = 0,..,16\}, |\mathfrak{V} \times \mathfrak{V}| = 17 * 17 = 289.$$

Использовалась компьютерная конфигурация с ЦПУ Intel i5-3570, ОЗУ 1x8GB 1333MHz, под управлением ОС x64 Windows 8. На рисунке Д.1 для каждой из тестируемых стратегий кружками изображены требуемые размеры v_j, а крестиками – фактические размеры x_j, образующие множество

$$\mathfrak{X} = \left\{ x_j \middle| \left(\left(x_j = g(1)(v_j) \right) \wedge \left(v_j \in \mathfrak{V} \right) \right) \right\}.$$

Так как система em основана на стратегии выбора первого подходящего незанятого участка, то для приближения эксперимента к наихудшим условиям память фрагментировалась так, как изображено на рисунке 3.1, где области чёрного цвета означают занятые участки памяти. Области *P*, *Q*, *R* равны *p*, фиксированы и вычисляются по формуле (3.1):

$$p = \left\lfloor \varphi * \frac{m - 2 * \max(\mathfrak{V}) - \max(\mathfrak{X}) - 2 - \rho}{3 * (\delta + 1)} \right\rfloor, \tag{3.1}$$

где $\varphi = 1$ – коэффициент степени фрагментации,

$\rho = 8$ – число служебных ячеек,

$\delta = 1$ – размер участков, которыми области *P*, *Q*, *R* фрагментировались так, что незанятый участок размера δ ячеек чередовался с занятым участком размера 1 ячейка.

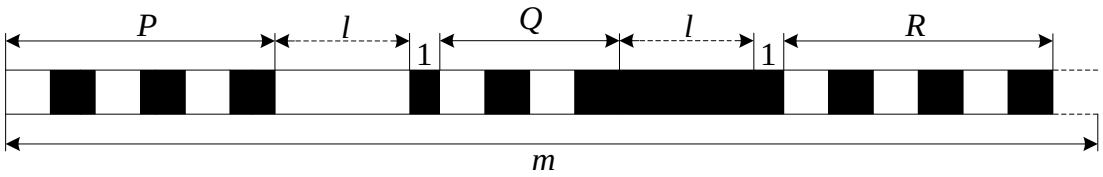


Рисунок 3.1 – Фрагментация памяти

При этом участки размером <*l* могут быть помещены в область размером *l* между *P*, *Q*, а участки размером >*l* помещаются в незанятую область памяти правее *R*. Формула (3.1) для φ=1 даёт близкое к максимальному допустимому значение p=7781178 ячеек, общее для всех тестируемых стратегий. Результаты тестирования стратегий приведены для φ=1 и изображены на рисунках Г.2–Г.8.

Графики на рисунках Г.2 и Г.4 показывают зависимость времени исполнения (в нс) операции перевыделения от размера выделяемых участков для стратегий без гистерезиса и с гистерезисом соответственно при использовании функции поиска на основе метаправил. Подписи max, mean и min указывают на максимальное, среднее и минимальное значения соответственно, полученные на множестве из 17∗17 точек.

Графики на рисунке Г.3 показывают, во сколько раз время исполнения (в нс) t_1 операции перевыделения при использовании стратегий без гистерезиса с функцией поиска на основе метаправил отличается от времени исполнения t_2 операции перевыделения при использовании тех же стратегий, но с функцией поиска на основе хранимых правил. Графики на рисунке Г.4 показывают во сколько раз время исполнения (в нс) t_1 операции перевыделения при использовании стратегий с гистерезисом с функцией поиска на основе

метаправил отличается от времени исполнения t_2 операции перевыделения при использовании тех же стратегий, но с функцией поиска на основе хранимых правил. Тёмные области на графиках, изображённых на рисунках Г.3 и Г.4 указывают на области, где отношение времён меньше либо равно 1.

График на рисунке Г.6 показывает зависимость натурального логарифма пропускной способности (бит/с) от размера, на который изменяется выделенный участок для геометрической стратегии с гистерезисом. Тёмная плоскость на графике отмечает нулевую пропускную способность. Значения *c* на главной диагонали стремятся к минус бесконечности, так как *l*=*u*.

Сравнительное тестирование различных систем динамического управления памятью дано, например, в [80]. Для сравнения с предлагаемой системой em была взята система tcmalloc, как самая производительная, исходя из приведённых в [80] данных.

Алгоритм операции перевыделения в системе tcmalloc основан на:

1. выделении нового участка памяти с фактическим размером, бóльшим на 25%, чем исходный, при его увеличении;
2. сокращении фактического размера исходного участка памяти до размера затребованного при уменьшении исходного более, чем на 50%.

Аналогичная стратегия была реализована для системы em. Графики на рисунке Г.7 показывают зависимость двоичного логарифма отношения времён исполнения (в нс) операций перевыделения в системах em (t_{em}) и tcmalloc (t_{tc}) для максимальных размеров перевыделяемого участка (слева направо) 512 ячеек, 4114 ячеек и 2^{25} ячеек в случае, когда память нефрагментирована.

Графики на рисунке Г.8 показывают зависимость двоичного логарифма отношения времён исполнения (в нс) операций перевыделения в системах em (t_{em}) и tcmalloc (t_{tc}) для максимальных размеров перевыделяемого участка (слева направо) 512 ячеек, 4114 ячеек и 2^{25} ячеек в случае, когда память фрагментирована (δ=5, p=1024 для размеров 512 и 4114 ячеек и p=1 для 2^{25} ячеек).

В системе tcmalloc память фрагментировалась последовательным выделением участков с размерами 4 байта и $4*\delta$ байт (т.к. размер ячейки в системе em равен 32 бита) для областей *P*, *Q*, *R* и других участков, согласно схеме фрагментации, изображённой на рисунке 3.1. После этого высвобождался каждый нечётный участок с минимальным адресом (всего высвобождалось $3*p$ участков).

3.2 Теоретико-множественные операции параллельной обработки знаний

Соответствующие алгоритмы теоретико-множественных операций параллельной обработки знаний были реализованы на языке C++ с

использованием средств платформы параллельного программирования OpenCL. Для тестирования использовались вычислительные архитектуры: CPU (Intel Core i7 3520M CPU) и GPU (Intel HD Graphics 4000).

Исходными данными служили массивы A и B, где $|A| \le |B|$, кодирующие соответствующие мультимножества $\hat{\mathfrak{A}}$ и $\hat{\mathfrak{B}}$. Было проведено шесть тестов, в каждом из которых использовались различные мультимножества, характеристика которых приведена в таблице 3.2. В каждом из тестов мощность мультимножеств равнялась 2^k, где $k = 1,..22$.

Таблица 3.2 – Описание входных данных для тестирования

Наименование теста	Характеристика входных массивов A и B, кодирующих мульмножества $\hat{\mathfrak{A}}$ и $\hat{\mathfrak{B}}$
тест 1	Массивы A и B изначально лексикографически неупорядочены, а каждый $A[i]$ и $B[j]$ есть случайная величина, равномерно распределённая на отрезке $[0, \max(\|A\|, \|B\|)/2]$
тест 2	A={1,3,5,...}, B={0,2,4,...}
тест 3	A и B не содержат кратных элементов и любой $A[i]$ больше любого $B[j]$
тест 4	A={0,1,4,5,8,9,...}, B={2,3,6,7,10,11,...}
тест 5	A={1,2,3,...}, B={1,2,3,...}
тест 6	A={0,0,0,...}, B={0,0,0,...}

Гранулы (kernels) OpenCL, соответствующие параллельной реализации теоретико-множественных операций, исполнялись только устройством GPU, одной рабочей группой и максимально возможным количеством рабочих элементов. При этом параллельная реализация операций пересечения (разности) и объединения потребовала объём дополнительной памяти равный $|A|$ и $3*|A|+2*|B|$ соответственно.

Для оценки выигрыша (проигрыша) в производительности при использовании параллельной реализации алгоритмов операций пересечения, объединения и разности мультимножеств были реализованы соответствующие последовательные алгоритмы, в основе которых лежит слияние упорядоченных списков [24], имеющие пространственную сложность $\Theta(|A|)$ и $\Theta(|A|+|B|)$ и выполненные с определёнными оптимизациями так, что оценка временно́й сложности для операции пересечения равна $O(|A|+|B|)$ в худшем случае и $O(\log_2(|B|))$ в лучшем случае, а для операции объединения – $O(|A|+|B|)$.

Последовательные реализации алгоритмов операций пересечения (объединения, разности) исполнялись:

1. в виде OpenCL гранулы устройством GPU, одной рабочей группой и одним рабочим элементом;

2. устройством CPU в виде однопоточной программы, написанной на языке C++.

Результаты сравнительного тестирования параллельных и последовательных реализаций операций пересечения и объединения мультимножеств при использовании устройства GPU для их исполнения изображены на рисунках 3.2 и 3.3. На рисунках Д.1 и Д.2 изображены того же тестирования, только при использовании устройства CPU для исполнения последовательных реализаций операций пересечения и объединения.

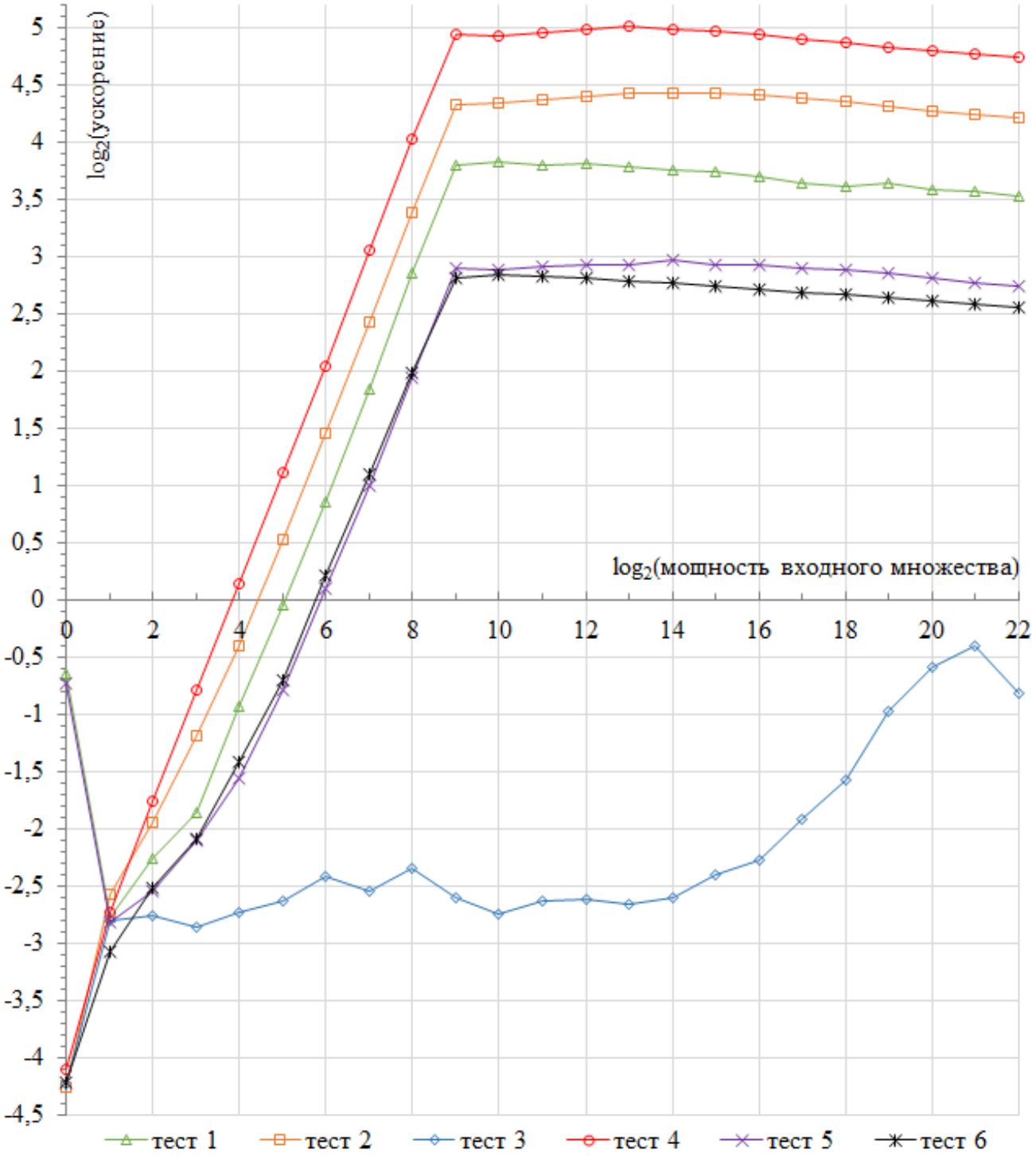


Рисунок 3.2 – График зависимости логарифма коэффициента ускорения по основанию 2 параллельной реализации операции пересечения по сравнению с последовательной реализацией (на устройстве GPU) от логарифма мощности пересекаемых множеств по основанию 2

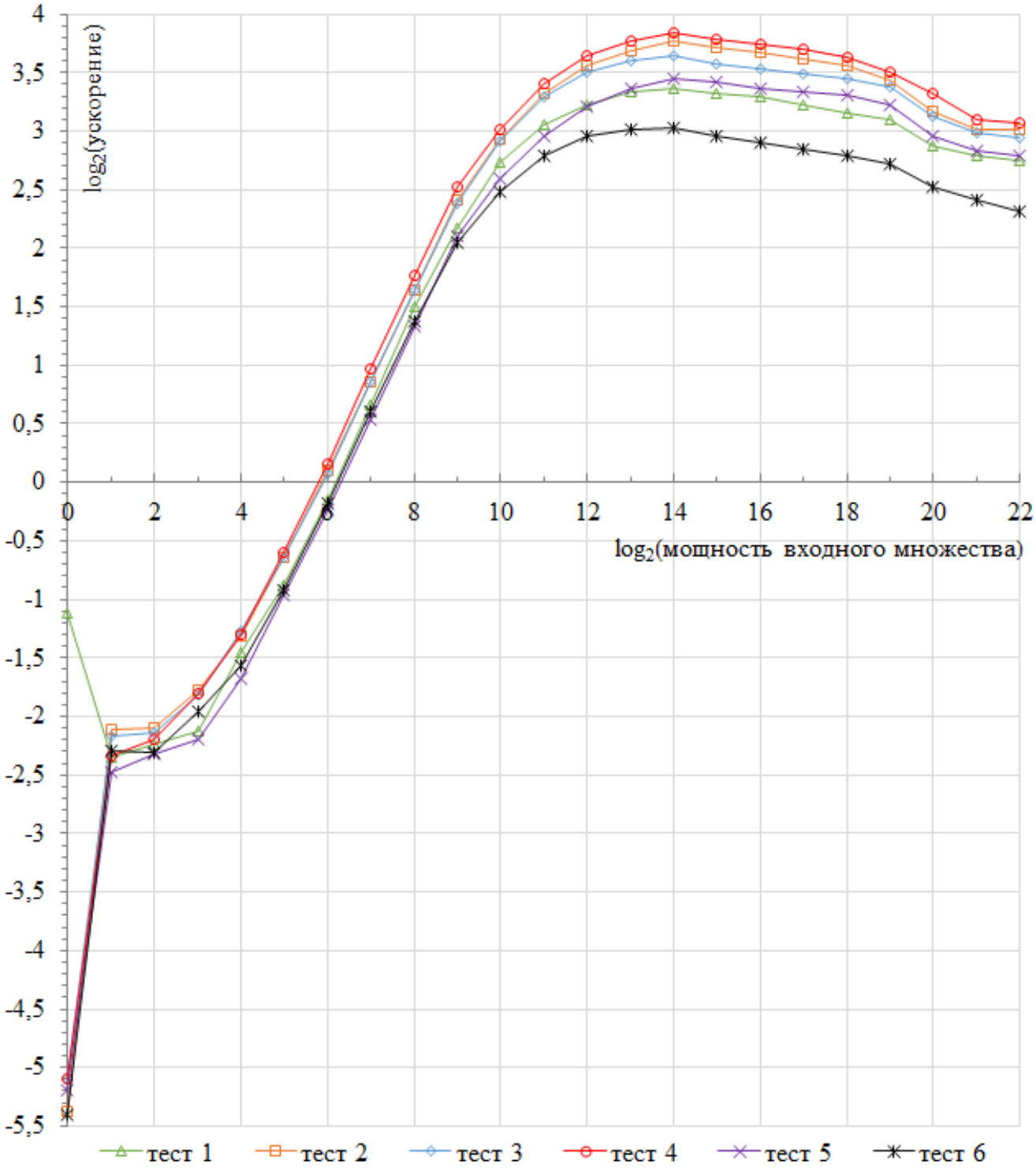


Рисунок 3.3 – График зависимости логарифма коэффициента ускорения по основанию 2 параллельной реализации операции объединения по сравнению с последовательной реализацией (на устройстве GPU) от логарифма мощности пересекаемых множеств по основанию 2

Выводы

Реализованы разработанные модели и алгоритмы стратегий перераспределения участков линейно адресуемой памяти, а также алгоритмы теоретико-множественных операций параллельной обработки знаний, ориентированные на использование параллелизма потока данных и графических процессорных устройств. В результате экспериментальной проверки реализованных программных средств установлено следующее:

1. Степенная и геометрическая стратегии с гистерезисом перераспределения участков памяти позволяют значительно снизить суммарное время, затраченное на все операции перевыделения как в случае использования функции выбора на основе хранимых правил, так и в случае использования функции выбора на основе метаправил, что согласуется с теоретической оценкой среднего амортизационного времени работы операции перевыделения. Время, затраченное на исполнение операций перевыделения при использовании стратегий с функцией поиска на основе метаправил, отличается от времени исполнения операции перевыделения при использовании тех же стратегий, но с функцией поиска на основе хранимых правил не более, чем в 1.6–2.5 раза, однако позволяет сохранить дополнительный объём памяти, пропорциональный количеству правил в стратегии. Пропускная способность операции перевыделения в лучшем случае (при использовании геометрической стратегии с гистерезисом) приближается к значению $9*10^{12}$ бит в секунду. Проведённое сравнение реализации предложенных алгоритмов, основанной на системе динамического управления памятью [11], с аналогичной системой tcmalloc [67] показало, что, в случае нефрагментированной памяти, для максимального размера перевыделяемого участка 512 ячеек суммарное время работы всех операций перевыделения в системе tcmalloc в 1.3 раза меньше. Однако для максимальных размеров перевыделяемого участка 4114 и 2^{25} ячеек суммарное время, наоборот, в 1.46 и 7030 раз выше. В случае фрагментированной памяти для максимальных размеров перевыделяемого участка 512 и 4114 ячеек суммарное время работы всех операций перевыделения в системе tcmalloc в 4.7 и 1.96 раза меньше, однако для максимального размера перевыделяемого участка 2^{25} ячеек суммарное время, наоборот, в 2.38 раза выше.

2. Параллельная реализация алгоритма операции пересечения (объединения, разности) мультимножеств, допускает параллельное исполнение на различных устройствах гетерогенной параллельной вычислительной архитектуры, не ограничиваясь только лишь графическими процессорными устройствами. Пропускная способность параллельной реализации операции объединения мультимножеств позволяет получить ускорение от 5 до 15 раз по сравнению с последовательной реализацией для достаточно больши́х размеров

объединяемых множеств (более 64 элементов). Пропускная способность параллельной реализации операции пересечения (разности) сильно зависит от характера входных мультимножеств. В лучшем случае ускорение составляет от 5 до 32 раз, однако в худшем случае, например, когда операция пересечения не совершает полезной работы (результат пересечения много меньше размеров входных мультимножеств), но на её исполнение затрачивается значительное процессорное время, параллельная реализация проигрывает последовательной в 1.3–7 раз. Несмотря на то, что параллельная реализация операций пересечения, разности и объединения мультимножеств не достигает идеальной теоретической временной оценки в силу небольшого, по сравнению с размерами обрабатываемых мультимножеств, числа процессоров и принципов организации коллективной работы процессоров с памятью, разработанные алгоритмы масштабируются с ростом количества процессоров.

ЗАКЛЮЧЕНИЕ

1. Исходя из свойств знаний и систем обработки знаний, а также из того, что наиболее универсальными формальными моделями представления и обработки информации являются модели с теоретико-множественной семантикой, проведён анализ формальных моделей, основанных на семантических сетях. Рассмотрены языки и операции распространённых графовых моделей обработки информации. Показано, что для всех данных моделей теоретико-множественные операции являются ключевыми. Рассмотрены различные способы кодирования конечных множеств и алгоритмы теоретико-множественных операций над ними. Большинство существующих алгоритмов теоретико-множественных операций параллельной обработки знаний ориентировано на использование параллелизма задач.

2. Проведён анализ средств формальной спецификации знаний и, на основе унифицированных семантических сетей [4] и семейства языков унифицированного семантического кодирования информации, разработана онтологическая модель теоретико-множественных операций модели обработки знаний. В основу онтологической модели положено понятие онтологии как явной спецификации концептуализации (согласно Т. Груберу), а сама спецификация строится с использованием средств модели спецификации знаний [12].

3. Получено приближённое решение обобщённой задачи дискретного управления памятью с конечным числом состояний на уровне управления устройствами и данными системы обработки знаний. Решение заключается в использовании разработанной модели и алгоритмов стратегий перераспределения участков линейно адресуемой битовой памяти на уровнях управления устройствами и данными системы обработки знаний. Разработанные алгоритмы основаны на системе динамического управления памятью, описанной в [11], и сохраняют стратегию выбора первого подходящего свободного участка памяти при перераспределении участков памяти, в связи с чем оценка перерасхода памяти по причине внешней фрагментации не превышает $O\left(U * \log_2\left(U\right)\right)$, где U – объём занятой памяти. В общем случае временна́я сложность добавления n элементов данных с учётом возможных удалений как уже имеющихся, так и добавленных данных, в выделенный участок памяти для геометрических стратегий оценивается как $O\left(f\left(m\right) * n + d * \tau\right)$, где m – объём памяти, d – количество перевыделений, $\tau = \ln^2\left(m\right)$ – время одного перевыделения в системе [11], $f(m)$ – время доступа (чтения/записи) к одной ячейке памяти. Исходя из этого, среднее амортизационное время добавления одного экземпляра данных в участок

памяти является постоянным, если время доступа к ячейке памяти в зависимости от размера памяти также постоянно, а количество записей данных в ячейки памяти зависит как минимум прямо пропорционально от произведения времени одного перевыделения на количество перевыделений.

4. На основе модели и алгоритмов стратегий перераспределения участков памяти разработаны алгоритмы теоретико-множественных операций параллельной обработки знаний, отличающиеся поддержкой мультимножеств и ориентацией на использование параллелизма потока данных и реализацию на мелко-и среднезернистой параллельной вычислительной архитектуре с произвольным доступом к общей памяти. Теоретическая оценка временно́й сложности данных алгоритмов при использовании разработанной модели стратегий перераспределения участков линейно адресуемой битовой памяти на уровнях управления устройствами и данными системы, основанной на знаниях, не превышает $O\left(\log_2(n)*n/p+\ln^2(m)*f(m)\right)$, где n – суммарная мощность множеств, p – количество процессоров, тогда как известные в настоящее время подобные алгоритмы дают оценку не менее, чем $O\left(n+p*\log_2(n)\right)$, без учёта временны́х затрат на управление памятью.

5. Разработанные модель и алгоритмы стратегий перераспределения участков линейно адресуемой памяти, а также алгоритмы теоретико-множественных операций параллельной обработки знаний, реализованы в виде программ на языке C++ с использованием средств платформы OpenCL. Осуществлено тестирование разработанных алгоритмов. В результате вычислительных экспериментов, проведённых над полученными программными средствами модели и алгоритмов стратегий перераспределения участков памяти, и сравнения их с другими реализациями установлено, что разработанные на основе системы [11] динамического управления памятью алгоритмы в одних случаях дают преимущество во времени исполнения, но в других случаях до нескольких раз проигрывают им, также получено подтверждение их теоретической средней амортизационной оценки временно́й сложности. Вычислительные эксперименты над разработанными алгоритмами теоретико-множественных операций показали, что их параллельные реализации масштабируются с ростом количества процессоров и, в большинстве случаев, обеспечивают меньшее время исполнения, чем их последовательные аналоги, при использовании современных распространённых графических процессорных устройств. Идеальная теоретическая оценка их временно́й сложности не подтверждена, что связано с небольшим, по сравнению с мощностями обрабатываемых множеств, количеством процессоров и принципов организации коллективной работы процессоров с памятью.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Алгоритмы: построение и анализ : пер. с англ. / Т. Кормен [и др.]. – М. : Вильямс, 2013. – 1328 с.
2. Гаврилова, Т. А. Базы знаний интеллектуальных систем : учеб. пособие / Т. А. Гаврилова, В. Ф. Хорошевский. – СПб. [и др.] : Питер, 2000. – 382 с.
3. Гарсиа-Молина, Г. Системы баз данных : полн. курс : пер. с англ. / Г. Гарсиа-Молина, Дж. Ульман, Дж. Уидом. – М. : Вильямс, 2003. – 1083 с.
4. Голенков, В. В. Графодинамические модели параллельной обработки знаний: принципы построения, реализации и проектирования / В. В. Голенков, Н. А. Гулякина // Открытые семантические технологии проектирования интеллектуальных систем: OSTIS-2012 : материалы II Междунар. науч.-техн. конф., Минск, 16–18 февр. 2012 г. / Белорус. гос. ун-т информатики и радиоэлектроники, Администрация Парка высоких технологий ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2012. – С. 23–52.
5. Голенков, В. В. Семантическая технология компонентного проектирования систем, управляемых знаниями / В. В. Голенков, Н. А. Гулякина // Открытые семантические технологии проектирования интеллектуальных систем: OSTIS-2015 : материалы V Междунар. науч.-техн. конф., Минск, 19–21 февр. 2015 г. / Белорус. гос. ун-т информатики и радиоэлектроники, Администрация Парка высоких технологий ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2015. – С. 57–78.
6. Голенков, В. В. Структуризация смыслового пространства / В. В. Голенков, Н. А. Гулякина // Открытые семантические технологии проектирования интеллектуальных систем: OSTIS-2014 : материалы V Междунар. науч.-техн. конф., Минск, 20–22 февр. 2014 г. / Белорус. гос. ун-т информатики и радиоэлектроники, Администрация Парка высоких технологий ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2014. – С. 65–78.
7. Горбатов, В. А. Фундаментальные основы дискретной математики: информационная математика : учебник / В. А. Горбатов. – М. : Наука, 2000. – 544 с.
8. Давыденко, И.Т. Средства структуризации семантических моделей баз знаний / И.Т. Давыденко, Н.В. Гракова, Е.С. Сергиенко, А.В. Федотова // OSTIS. 2016. – С. 93–106. ISBN 978-985-543-034-7
9. Евстигнеев, В. А. Применение теории графов в программировании / В. А. Евстигнеев ; под ред. А. П. Ершова. – М. : Наука, 1985. – 352 с.

10. Жарко, Е. Ф. Оценка качества программного обеспечения АСУ ТП АЭС: теоретические основы, основные тенденции и проблемы / Е. Ф. Жарко // Труды X Международной конференции «Идентификация систем и задачи управления», SICPRO'15 : Москва, 26–29 янв. 2015 г. / Ин-т проблем упр. Рос. акад. наук. – М., 2015. – С. 1129–1143.
11. Ивашенко, В. П. Алгоритмы полилогарифмической временной и логарифмической пространственной сложности для системы динамического распределения линейно адресуемой памяти с однородным доступом к данным / В. П. Ивашенко // Карповские научные чтения : сб. науч. ст. / Белорус. гос. ун-т. – Минск, 2012. – Вып. 6, ч. 1. – С. 196–201.
12. Ивашенко, В. П. Модели и алгоритмы интеграции знаний на основе однородных семантических сетей / В. П. Ивашенко // Открытые семантические технологии проектирования интеллектуальных систем: OSTIS-2015 : материалы V Междунар. науч.-техн. конф., Минск, 19–21 февр. 2015 г. / Белорус. гос. ун-т информатики и радиоэлектроники, Администрация Парка высоких технологий ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2015. – С. 111–132.
13. Ивашенко, В. П. Модели обработки информации в интеллектуальных системах, основанных на семантических технологиях / В. П. Ивашенко, А. С. Бельчиков, А. П. Еремеев // Информационные технологии и системы 2016 (ИТС 2016) : материалы междунар. науч. конф., Минск, 26 окт. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники. – Минск, 2016. – С. 106–107.
14. Ивашенко, В. П. Принципы платформенной независимости и платформенной реализации OSTIS / В. П. Ивашенко, М. М. Татур // Открытые семантические технологии проектирования интеллектуальных систем: OSTIS-2016 : материалы VI Междунар. науч.-техн. конф., Минск, 18–20 февр. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники, Администрация Парка высоких технологий ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2016. – С. 145–150.
15. Искусственный интеллект : справочник : в 3 кн. / под ред. Э. В. Попова. – М. : Радио и связь, 1990. – Кн. 2 : Модели и методы / Д. А. Поспелов [и. др.]. – 304 с.
16. Керниган, Б. Язык программирования С / Б. Керниган, Д. Ритчи ; пер. с англ. и ред. В. Л. Бродового. – 2-е изд., перераб. и доп. – М. [и др.] : Вильямс, 2013. – 289 с.
17. Клини, С. К. Математическая логика = Mathematical logic : пер. с англ. / С. К. Клини ; пер. с англ. Ю. А. Гастева ; под ред. Г. Е. Минца ; предисл. Ю. А. Гастева, Г. Е. Минца. – Изд. 4-е. – М. : ЛКИ, 2008. – 480 с.

18. Кнут, Д. Искусство программирования : пер. с англ. : в 3 т. / Д. Кнут ; под общ. ред. Ю. В. Козаченко. – М. : Вильямс, 2012. – Т. 1 : Основные алгоритмы. – 712 с.
19. Кнут, Д. Искусство программирования : пер. с англ. : в 3 т. / Д. Кнут ; под общ. ред. Ю. В. Козаченко. – М. : Вильямс, 2012. – Т. 3 : Сортировка и поиск. – 822 с.
20. Косяков, М. С. Введение в распределенные вычисления : учеб. пособие / М. С. Косяков. – СПб. : С.-Петерб. нац. исслед. ун-т информ. технологий, механики и оптики, 2014. – 155 с.
21. Кузьмицкий, В. М. Принципы построения графодинамического параллельного компьютера, ориентированного на решение задач искусственного интеллекта : дис. … канд. техн. наук : 05.13.11 / В. М. Кузьмицкий. – Минск, 2000. – 236 л.
22. Минский, М. Вычисления и автоматы / М. Минский ; пер. с англ. Б. Л. Овсиевича, Л. Я. Розенблюма. – М. : Мир, 1971. – 366 с.
23. Непейвода, Н. Н. Семантика алгоритмических языков / Н. Н. Непейвода // Итоги науки и техники. Сер.: Теория вероятностей. Мат. статистика. Теорет. кибернетика. – 1983. – Т. 20. – С. 95–166.
24. Новиков, Ф. А. Дискретная математика для программистов / Ф. А. Новиков. – СПб. : Питер Мейл, 2016. – 496 с.
25. О приоритетных направлениях научно-технической деятельности в Республике Беларусь на 2016–2020 годы [Электронный ресурс] : Указ Президента Респ. Беларусь, 22 апр. 2015 г., № 166 // КонсультантПлюс. Беларусь / ООО «ЮрСпектр», Нац. центр правовой информ. Респ. Беларусь. – Минск, 2016.
26. О приоритетных направлениях научных исследований Республики Беларусь на 2016–2020 годы [Электронный ресурс] : постановление Совета Министров Респ. Беларусь, 12 марта 2015 г., № 190 // КонсультантПлюс. Беларусь / ООО «ЮрСпектр», Нац. центр правовой информ. Респ. Беларусь. – Минск, 2016.
27. Орлов, С. А. Организация ЭВМ и систем : учебник / С. А. Орлов, Б. Я. Цилькер. – 3-е изд. – СПб. : Питер, 2014. – 688 с.
28. Осуга, С. Обработка знаний / С. Осуга ; пер. с яп. В. И. Этова. – М. : Мир, 1989. – 293 с.
29. Параллельные вычисления на GPU. Архитектура и программная модель CUDA : учеб. пособие / А. В. Боресков [и др.] ; предисл. В. А. Садовничий. – М. : Изд-во Моск. гос. ун-та, 2012. – 336 с.
30. Плесневич, Г. С. Спецификация потока задач в расширенной интервальной логике Аллена / Г. С. Плесневич, Нгуен Тхи Минь Ву // Открытые семантические технологии проектирования интеллектуальных систем: OSTIS-2016 : материалы VI Междунар. науч.-техн. конф., Минск,

18–20 февр. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники, Администрация Парка высоких технологий ; редкол.: В. В. Голенков (отв. ред.) [и др.]. – Минск, 2016. – С. 295–300.
31. Представление и обработка знаний в графодинамических ассоциативных машинах / В. В. Голенков [и др.] ; под ред. В. В. Голенкова. – Минск : Белорус. гос. ун-т информатики и радиоэлектроники, 2001. – 410 с.
32. Программирование в ассоциативных машинах = Programming in associative machines / В. В. Голенков [и др.]. – Минск : Белорус. гос. ун-т информатики и радиоэлектроники, 2001. – 274 с.
33. Рассел, С. Искусственный интеллект. Современный подход : пер. с англ. / С. Рассел, П. Норвиг. – М. [и др.] : Вильямс, 2006. – 1407 с.
34. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# : пер. с англ. / Д. Рихтер. – СПб. [и др.] : Питер, 2016. – 896 с.
35. Седжвик, Р. Фундаментальные алгоритмы на C++ : [в 5 ч.] : пер. с англ. / Р. Седжвик – СПб. [и др.] : ДиаСофтЮП, 2001–2002. – Ч. 5 : Алгоритмы на графах. – 2002. – 484 с.
36. Семантическая модель сложноструктурированных баз данных и баз знаний : учеб. пособие / В. В. Голенков [и др.]. – Минск : Белорус. гос. ун-т информатики и радиоэлектроники, 2004. – 262 с.
37. Тейз, А. Логический подход к искусственному интеллекту: От модальной логики к логике баз данных: Пер. с франц./Тейз А., Грибомон П., Юлен Г. и др. - М.: Мир, 1998.-494 с, ил.
38. Ульянов, М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ : учеб. пособие / М. В. Ульянов. – М. : Физматлит, 2008. – 304 с.
39. Флэнаган, Д. JavaScript : подроб. рук. : пер. с англ. / Д. Флэнаган. – 6-е изд. – СПб. ; М. : Символ-Плюс, 2013. – 1080 с.
40. Харари, Ф. Теория графов = Graph theory / Ф. Харари ; пер. с англ. и предисл. В. П. Козырева ; под ред. Г. П. Гаврилова. – Изд. 2-е. – М. : Едиториафл УРСС, 2003. – 296 с.
41. Шпаковский, Г. И. Реализация параллельных вычислений: кластеры, многоядерные процессоры, грид, квантовые компьютеры / Г. И. Шпаковский. – Минск : Белорус. гос. ун-т, 2010. – 155 с.
42. A constant-time dynamic storage allocator for real-time systems / M. Masmano [et al.] // Real-Time Systems. – 2008. – Vol. 40, № 2. – P. 149–179.
43. Accelerating keyword search for big RDF WebData on many-core systems / C. Choksuchat [et al.] // Intelligent software methodologies, tools and techniques : 14th intern. conf., Naples, 15–17 Sept. 2015 : proceedings / ed.: H. Fujita, G. Guizzi. – Cham, 2015. – P. 190–202.

44. Alashqur, A. M. OQL: a query language for manipulating object-oriented databases / A. M. Alashqur, S. Y. W. Su, H. Lam // VLDB'89 : proc. of the 15th intern. conf. on Very large data bases, Amsterdam, 22–25 Aug. 1989 / Very Large Data Base Endowment ; ed.: P. M. G. Apers, G. Wiederhold. – Palo Alto, 1989. – P. 433–442.
45. Alglave, J. Towards a formalization of the HSA memory model in the cat language [Electronic resource] / J. Alglave, L. Maranget // HSA Foundation. – Mode of access: http://www.hsafoundation.com/?ddownload=5381. – Date of access: 15.11.2016.
46. AllegroGraph [Electronic resource]. – Mode of access: http://allegrograph.com. – Date of access: 22.11.2016.
47. Allen, J.A. Maintaining knowledge about temporal intervals// Communications of the ACM, 20(11), 1983. – P.832-843.
48. Al-Sibahi, A. S. On the computational expressiveness of model transformation languages / A. S. Al-Sibahi. – Copenhagen : IT Univ., 2015. – 18 p. – (IT University technical report series ; TR-2015-184).
49. Apache TinkerPop [Electronic resource]. – Mode of access: https://tinkerpop.apache.org/index.html. – Date of access: 22.11.2016.
50. Babai, L. Graph isomorphism in quasipolynomial time / L. Babai // STOC'16 : proc. of the 48th annu. Symp. on theory of computing, Cambridge, 19–21 June 2016 / Assoc. for Computing Machinery Spec. Interest Group for Algorithms a. Computation Theory. – New York, 2016. – P. 684–697.
51. Baeza-Yates, R. Experimental analysis of a fast intersection algorithm for sorted sequences / R. Baeza-Yates, A. Salinger // String processing and information retrieval : 12th intern. conf., SPIRE 2005, Buenos Aires, 2–4 Nov. 2005 : proceedings / ed.: M. Consens, G. Navarro. – Berlin ; New York, 2005. – P. 13–24.
52. Baxter, S. Multisets [Electronic resource] / S. Baxter // NVIDIA Research Projects. – Mode of access: https://nvlabs.github.io/moderngpu/sets.html. – Date of access: 11.09.2016.
53. Blazegraph [Electronic resource]. – Mode of access: https://www.blazegraph.com. – Date of access: 22.11.2016.
54. Blelloch, G. E. Just join for parallel ordered sets / G. E. Blelloch, D. Ferizovic, Y. Sun // SPAA'16 : proc. of the 28th Symp. on parallelism in algorithms a. architectures, Pacific Grove, 11–13 July 2016 / Assoc. for Computing Machinery Spec. Interest Group for Algorithms a. Computation Theory. – New York, 2016. – P. 253–264.
55. Boley, H. RuleML 1.0: the overarching specification of web rules / H. Boley, A. Paschke, O. Shafiq // RuleML'10 : proc. of the intern. symp. on Semantic web rules, Washington, 21–23 Oct. 2010 / ed.: M. Dean [et al.]. – Berlin, 2010. – P. 162–178.

56. Boling, D. Fast set intersection in memory / D. Boling, A. C. König // Proc. of the VLDB Endowment. – 2011. – Vol. 4, № 4. – P. 255–266.
57. BrightstarDB [Electronic resource] : a native RDF database for the .NET platform. – Mode of access: http://brightstardb.com. – Date of access: 22.11.2016.
58. Choksuchat, C. On the HDT with the tree representation for large RDFs on GPU / C. Choksuchat, C. Chantrapornchai // International Conference on Parallel and Distributed Systems (ICPADS 2013) : Seoul, 15–18 Dec. 2013 : proceedings / Inst. of Electrical a. Electronics Engineers, Computer Soc. – P. 651–656.
59. Conceptual graphs / J. F. Sowa [et al.] // Handbook of knowledge representation / F. van Harmelen [et al.]. – Amsterdam ; Boston, 2008. – P. 213–237.
60. DataStax Enterprise Graph [Electronic resource] : the only scalable real-time graph database // DataStax. – Mode of access: http://www.datastax.com/products/datastax-enterprise-graph. – Date of access: 22.11.2016.
61. Demaine, E. Adaptive set intersections, unions, and differences / E. Demaine, A. López-Ortiz, J. Munro // Proceedings of the eleventh annual ACM-SIAM Symposium on Discrete Algorithms / Assoc. for Computing Machinery, Soc. for Industr. a. Appl. Mathematics. – New York ; Philadelphia, 2000. – P. 743–752.
62. Efficient lists intersection by CPU-GPU cooperative computing [Electronic resource] / Di Wu [et al.] // IPDPS 2010 : proc. of the 2010 IEEE Intern. symp. on parallel a. distributed processing, Atlanta, 19–23 Apr. 2010 / Inst. of Electrical a. Electronics Engineers. – Piscataway, 2010. – Mode of access: http://www.ipdps.org/ipdps2010/ipdps2010-slides/LSPP/IPDPS_LSPP.pdf. – Date of access: 23.11.2016.
63. Efficient RDF representation and parallel join processing Algorithm on general purpose many-core / M. Bae [et al.] // International Symposium on Computer, Consumer and Control (IS3C 2016) : Xi'an, 4–6 July 2016 / Nat. Chin-Yi Univ. of Technology, Xi'an Univ. of Science a. Technology. – Xi'an, 2016. – P. 378–382.
64. Evans, J. A scalable concurrent malloc(3) implementation for FreeBSD [Electronic resource] / J. Evans // BSDCam 2006 : the techn. BSD conf., Ottawa, 12–13 May 2006 / Univ. of Ottawa. – Ottawa, 2006. – Mode of access: https://www.bsdcan.org/2006/papers/jemalloc.pdf. – Date of access: 23.11.2016.
65. Farrugia, J. Model theoretic semantics for the web / J. Farrugia // WWW'03 : proc. of the 12th intern. World wide web conf., Budapest, 20–24 May 2003 /

Assoc. for Computing Machinery Digital Libr. ; ed. G. Hencsey. – Budapest, 2003. – P. 29–38.

66. Finin, T. KQML as an agent communication language / T. Finin // CIKM'94 : proc. of the 3rd intern. Conf. on inform. a. knowledge management, Gaithersburg, 29 Nov. – 2 Dec. 1994 / Nat. Inst. of Standards a. Technology. – Gaithersburg, 1994. – P. 456–463.
67. Ghemawat, S. TCmalloc: thread-caching malloc [Electronic resource] / S. Ghemawat, P. Menage // Google Performance Tools. – Mode of access: http://goog-perftools.sourceforge.net/doc/tcmalloc.html. – Date of access: 16.11.2016.
68. GP (Graph Programs) – the programming languages and Systems Research Group [Electronic resource] // Department of Computer Science. – Mode of access: https://www.cs.york.ac.uk/plasma/wiki/index.php?title=GP_(Graph_Programs) . – Date of access: 20.11.2016.
69. GPU-accelerated applications [Electronic resource] // NVIDIA. – Mode of access: http://www.nvidia.com/content/gpu-applications/PDF/gpu-applications-catalog.pdf. Date of access: 09.12.2016.
70. GRAKN.AI [Electronic resource]. – Mode of access: https://grakn.ai. – Date of access: 22.11.2016.
71. Green, O. GPU merge path: a GPU merging algorithm / O. Green, R. McColl, D. A. Bader // ICS'12 : proc. of the 26th Intern. conf. on supercomputing, Venice, 25–29 June 2012 / Assoc. for Computing Machinery Spec. Interest Group on Computer Architecture, Assoc. for Computing Machinery Digital Libr. ; ed. U. Banerjee. – New York, 2012. – P. 331–340.
72. Guan, X. Parallel methods for solving fundamental file rearrangement problems / X. Guan, M. A. Langston // J. of Parallel a. Distributed Computing. – 1992. – Vol. 14, № 4. – P. 436–439.
73. Gunrock: a high-performance graph processing library on the GPU / Y. Wang [et al.] // ACM SIGPLAN Not. – 2015. – Vol. 50, № 8. – P. 265–266.
74. Hartke, S. G. McKay's canonical graph labeling algorithm / S. G. Hartke, A. J. Radclife // Communicating Mathematics. – 2009. – Vol. 479. – P. 99–111.
75. Heterogeneous computing with OpenCL 2.0 / D. Kaeli [et al.]. – Waltham : M. Kaufmann, 2015. – 330 p.
76. Hoard: a scalable memory allocator for multithreaded applications / E. D. Berger [et al.] // ASPLOS-IX : proc. of the 9th intern. conf. on Architectural support for programming lang. a. operating systems, Cambridge, 12–15 Nov. 2000 / Assoc. for Computing Machinery. – New York, 2000. – P. 117–128.

77. Hoare, C. A. R. An axiomatic basis for computer programming / C. A. R. Hoare // Communications of the ACM. – 1969. – Vol. 12, № 10. – P. 576–580.
78. HSA Multi-vendor specification [Electronic resource] // HAS Foundation. – Mode of access: http://www.hsafoundation.com/?ddownload=5319. – Date of access: 20.11.2016.
79. Kuznetsov, S. Comparing performance of algorithms for generating concept lattices / S. Kuznetsov, S. Obiedkov // J. of Experimental a. Theoretical Artificial Intelligence. – 2002. – Vol. 14, № 2/3. – P. 189–216.
80. Lockless: benchmarks of the lockless memory allocator [Electronic resource] // Lockless. – Mode of access: http://locklessinc.com/benchmarks_allocator.shtml. – Date of access: 16.11.2016.
81. Neo4j [Electronic resource] : the world's leading graph database. – Mode of access: https://neo4j.com. – Date of access: 20.11.2016.
82. OrientDB [Electronic resource]. – Mode of access: http://orientdb.com/orientdb. – Date of access: 22.11.2016.
83. OSTIS [Электронный ресурс]. – Режим доступа: http://ims.ostis.net. – Дата доступа: 20.11.2016.
84. Pai, S. A compiler for throughput optimization of graph algorithms on GPUs / S. Pai, K. Pingali // ACM SIGPLAN Not. – 2016. – Vol. 51, № 10. – P. 1–19.
85. Panzarino, O. Learning Cypher: write powerful and efficient queries for Neo4j with Cypher, its official query language / O. Panzarino. – Birmingham : Packt Publ., 2014. – 162 p.
86. Plump, D. The graph programming language GP / D. Plump // CAI'2009. Algebraic informatics : proc. of the 3rd intern. conf., Thessaloniki, 19–22 May 2009 / ed.: S. Bozapalidēs, G. Rachōnēs. – Berlin ; Heidelberg ; New York, 2009. – P. 99–122.
87. PROGRES – A Graph Grammar Programming Environment [Электронный ресурс] – Режим доступа: http://www-i3.informatik.rwth-aachen.de/tikiwiki/tiki-index.php%3Fpage_ref_id=213.html. Дата доступа: 20.11.2016.
88. RDF application development for IBM data servers [Electronic resource] // IBM. – Mode of access: http://www.ibm.com/support/knowledgecenter/SSEPGG_11.1.0/com.ibm.swg.im.dbclient.rdf.doc/doc/c0059661.html. – Date of access: 22.11.2016.
89. Resizable arrays in optimal time and space / A. Brodnik [et al.] // WADS'99 : proc. of the 6th intern. Workshop on algoritms a. data structures, Vancouver, 11–14 Aug. 1999 / ed.: F. Dehne [et al.]. – Berlin ; New York, 1999. – P. 37–48.

90. Robson, J. M. Worst case fragmentation of first fit and best fit storage allocation strategies / J. M. Robson // Computer. J. – 1977. – Vol. 20, № 3. – P. 242–244.
91. Rodriguez, M. A. The Gremlin graph traversal machine and language / M. A. Rodriguez // DBPL'2015 : proc. of the 15th symp. on Database programming lang., Pittsburg, 27 Oct. 2015 / Assoc. for Computing Machinery Spec. Interest Group on Programming Lang., Assoc. for Computing Machinery ; ed.: J. Cheney, T. Neumann. – New York, 2015. – P. 1–10.
92. Rozenberg, G. Handbook of graph grammars and computing by graph transformations : in 3 vol. / G. Rozenberg. – Singapore ; New Jersey : World Sci., 1997–1999. – 3 vol.
93. Sapaty, P.S. The WAVE-1: a new ideology and language for distributed processing on graphs and networks / P.S. Sapaty // Computers and artificial intelligence. – 1987. – V. 6, № 5. – P. 421–436.
94. Sommerville, I. Chapter 27. Formal Specification. Software engineering [Электронный ресурс] - Режим доступа: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/ExtraChaps/FormalSpec.pdf. Дата доступа: 14.12.2016
95. SPARQL 1.1 query language [Electronic resource] : W3C recommendation, 21 March 2013 / ed.: S. H. Garlik, A. Seaborne // World Wide Web Consortium (W3C). – Mode of access: https://www.w3.org/TR/sparql11-query. – Date of access: 23.11.2016.
96. SparqlImplementations [Electronic resource] // W3C Wiki. – Mode of access: https://www.w3.org/wiki/SparqlImplementations. – Date of access: 22.11.2016.
97. Stardog [Electronic resource]. – Mode of access: http://stardog.com. – Date of access: 22.11.2016.
98. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) [Electronic resource] : ISO/IEC 25000:2014(en) // ISO. – Mode of access: https://www.iso.org/obp/ui/#iso:std:iso-iec:25000:ed-2:v1:en. – Date of access: 22.11.2016.
99. Szeredi, P. The semantic web explained: the technology and mathematics behind Web 3.0 / P. Szeredi, G. Lukcsy, T. Benk. – New York : Cambridge Univ. Press, 2014. – 478 p.
100. The Java virtual machine specification [Electronic resource] : JavaSE 8 / ed.: T. Lindholm [et al.]. – [S. l. : s. n.], 2015. – Mode of access: https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf. – Date of access: 15.11.2016.
101. TITAN Distributed Graph Database [Electronic resource]. – Mode of access: http://titan.thinkaurelius.com. – Date of access: 22.11.2016.

102. Wirth, N. Programming in Modula-2 / N. Wirth. – 4th ed. – Berlin ; New York : Springer, 1988. – 182 p.
103. Wolfram, S. An elementary introduction to the Wolfram language / S. Wolfram. – Champaign : Wolfram Media, 2015. – XV, 324 p.
104. Wood, P. T. Query languages for graph databases / P. T. Wood // ACM SIGMOD Rec. – 2012. – Vol. 41, № 1. – P. 50–60.
105. Zündorf, A. Graph pattern matching in PROGRES / A. Zündorf // Graph grammars and their application to computer science : sel. papers of the 5th intern. workshop, Williamsburg, 13–18 Nov. 1994 / ed. J. Cuny. – Berlin ; New York, 1996. – P. 454–468.

Список публикаций соискателя

1–А. Ивашенко, В. П. Операции управления массивами данных в линейно адресуемой памяти / В. П. Ивашенко, С. В. Синцов // Докл. БГУИР. – 2016. – № 6. – С. 86–93.
2–А. Ивашенко, В. П. Алгоритмы параллельной реализации анализа формальных понятий для приближенных множеств в однородных семантических сетях / В. П. Ивашенко, С. В. Синцов // BIGDATA and Advanced Analytics : материалы 2-ой науч.-практ. конф., Минск, 15–17 июня 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники. – Минск, 2016. – С. 110–118.
3–А. Ивашенко, В. П. Параллельная реализация операций над приближенными множествами для анализа формальных понятий в однородных семантических сетях / В. П. Ивашенко, С. В. Синцов // Пятнадцатая национальная конференция по искусственному интеллекту с международным участием КИИ-2016 : Смоленск, 3–7 окт. 2016 г. : тр. конф. : в 3 т. / Рос. ассоц. искусств. интеллекта, Федер. исслед. центр «Информатика и управление» Рос. акад. наук. – Смоленск, 2016. – Т. 1. – С. 343–351.
4–А. Синцов, С. В. Параллельная реализация операции пересечения множеств для решения задачи анализа формальных понятий / С. В. Синцов // Международный конгресс по информатике: информационные системы и технологии (CSIST'16) : материалы междунар. науч. конгр., Минск, 24–27 окт. 2016 г. / Евраз. ассоц. ун-тов [и др.]. – Минск, 2016. – С. 913–918.
5–А. Синцов, С. В. Параллельная реализация операции пересечения множеств для решения задачи анализа формальных понятий / С. В. Синцов // Информационные технологии и системы 2016 (ИТС 2016) : материалы междунар. науч. конф., Минск, 26 окт. 2016 г. / Белорус. гос. ун-т информатики и радиоэлектроники. – Минск, 2016. – С. 132–133.

ПРИЛОЖЕНИЕ А

Таблица А.1 – Исследуемые свойства языков программирования, ориентированных на обработку графовых информационных структур

Название языка	Год появления и последнего обновления специфика-ции	Парадигма		Синтаксис		Поддержка модельно-теоретической семантики	Поддержка рефлексивной семантики	Поддержка явного параллелизма и асинхронности	Тьюринг-полнота
		процедурная	декларативная	линейный (символьный)	нелинейный (графовый)				
SCP	1993, 2016	+		+	+	+	+	+	+
PROGRES	1994, 1995	+	+	+	+				+
GP (версия 2)	2007, 2012		+	+	+				+
SPARQL (версия 1.1)	2008, 2013		+	+					
Gremlin	2009, 2015	+	+	+				+	+
Cypher	2013, 2016		+	+					

Таблица А.2 – Исследуемые свойства реализации абстрактных машин языков программирования, ориентированных на обработку графовых структур

Название языка	Реализующая платформа	Поддерживаемые виды параллелизма				Открытый исходный код	Спецификация реализации
		мелкозернистый	Средне-и крупнозернистый				
			потоки и процессы ОС	MIMD (GPU)	распределённость памяти		
SCP	Интерпретатор, реализованный на языке C	+	+			+	
PROGRES	Транслятор, переводящий исходные коды на языки C и Modula-2	+					
GP (версия 2)	Компилятор, переводящий исходные коды в байт-код виртуальной машины YAM, реализованной на языке C	+					
SPARQL (версия 1.1)	Интерпретаторы, реализованные на языках C/C++, Java, Lisp, JavaScript, C# и др.	+	+	+	+	+	
Gremlin	Интерпретаторы, реализованные на Java и др.	+	+	+	+	+	
Cypher	Интерпретатор, реализованный Java	+	+			+	

Таблица А.3 – Платформы, реализующие язык программирования SPARQL версии 1.1

Реализующая платформа	Язык реализации	Год последнего обновления	Вид параллелизма				Открытый исходный код	Коммерческое использование	Спецификация реализации	Представление в виде семантической сети
			мелкозернистый	средне-и крупнозернистый						
				потоки и процессы ОС	MIMD (GPU)	распределённость памяти				
Open Anzo	Java	2009	+	+			+			+
Hercules	JavaScript	2009	+				+			
StrixDB	C	2011	+	+						+
Mulgara	Java	2012	+	+			+			+
Apache Jena	Java	2014	+				+			+
Apache Marmotta	Java	2014	+	+			+			+
Redland (Rasqal RDF Query Library)	C	2014	+				+			+
AllegroGraph	Lisp	2015	+	+		+		+[1]		+
Blazegraph	Java	2016	+	+	+	+	+[2]	+[1]		+
BrightstarDB	C#	2016	+	+			+			+
Conceptual Resource Search Engine (Corese)	Java	2016	+	+		+	+			+
IBM DB2	C/C++	2016	+	+				+		+
Ontotext Graph DB	Java	2016	+	+				+[1]		+
Oracle Spatial and Graph	C/C++	2016	+	+		+		+		+
OrientDB	Java	2016	+	+		+	+[2]	+[1]		
RDF4J (Sesame)	Java	2016	+				+			+
Stardog	Java	2016	+	+				+[1]		+
Virtuoso	C	2016	+	+		+	+[2]	+[1]		+

Примечание. Знак «+[1]» означает наличие двух реализаций: коммерческой и бесплатной с ограниченной функциональностью; знак «+[2]» означает частичную открытость исходного кода.

Таблица А.4 – Платформы, реализующие язык программирования Gremlin

Реализующая платформа	Язык реализации	Год последнего обновления	Вид параллелизма				Открытый исходный код	Коммерческое использование	Спецификация реализации	Представление в виде семантической сети
			мелкозернистый	средне-и крупнозернистый						
				потоки и процессы ОС	MIMD (GPU)	распределённость памяти				
Blazegraph	Java	2016	+	+	+	+	+[2]	+[1]		+
DSEGraph (Titan)	Java	2016	+	+		+		+[1]		
GRAKN.AI	Java	2016	+	+		+	+			+
Neo4j	Java	2016	+	+			+[2]	+[1]		
OrientDB	Java	2016	+	+		+	+[2]	+[1]		
Stardog	Java	2016	+	+				+[1]		+
TinkerGraph	Java	2016	+				+			
Titan	Java	2016	+	+		+	+			

Примечание. Знак «+[1]» означает наличие двух реализаций: коммерческой и бесплатной с ограниченной функциональностью; знак «+[2]» означает частичную открытость исходного кода.

ПРИЛОЖЕНИЕ Б

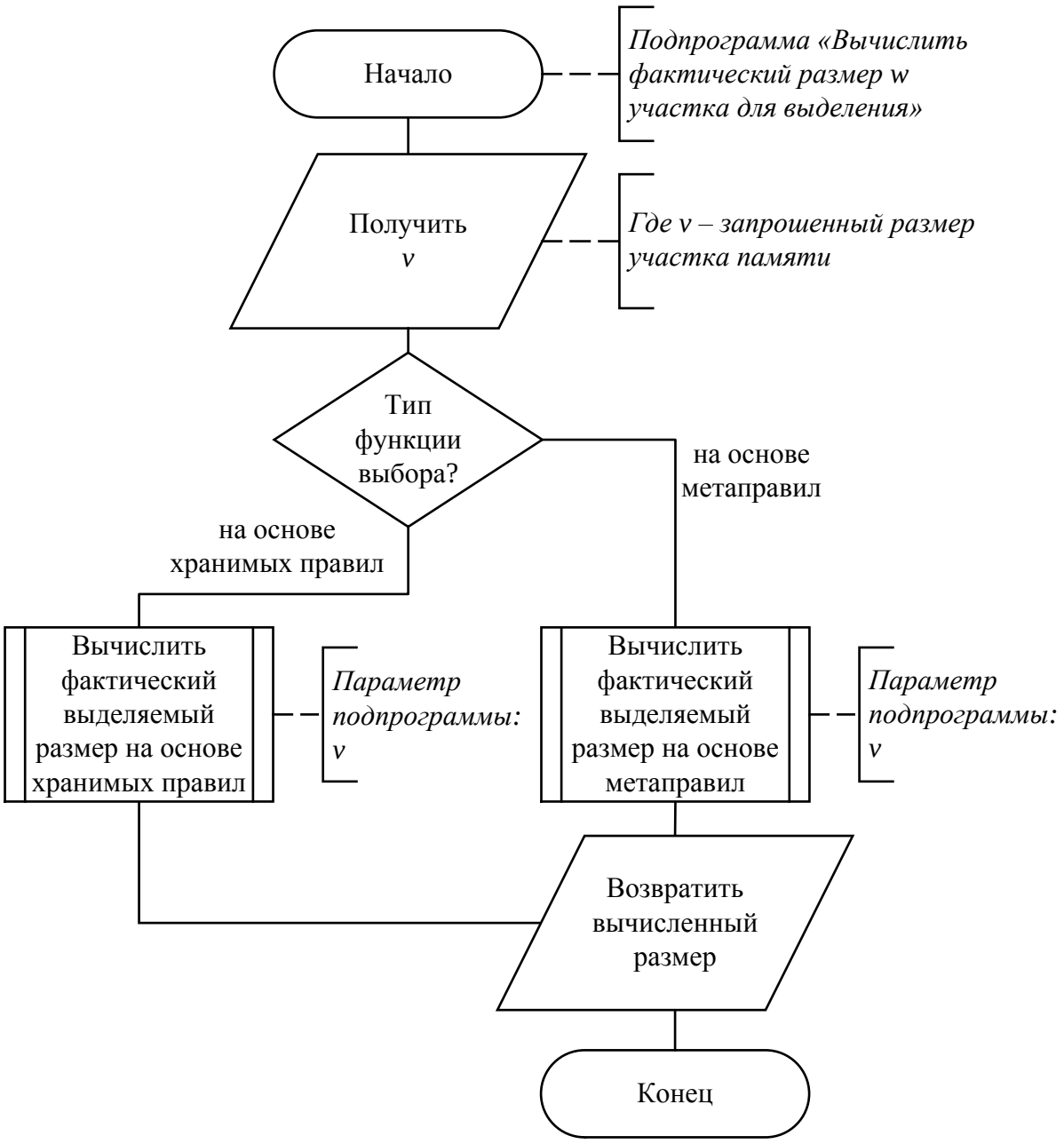


Рисунок Б.1 – Блок-схема подпрограммы «Вычислить фактический размер w участка для выделения»

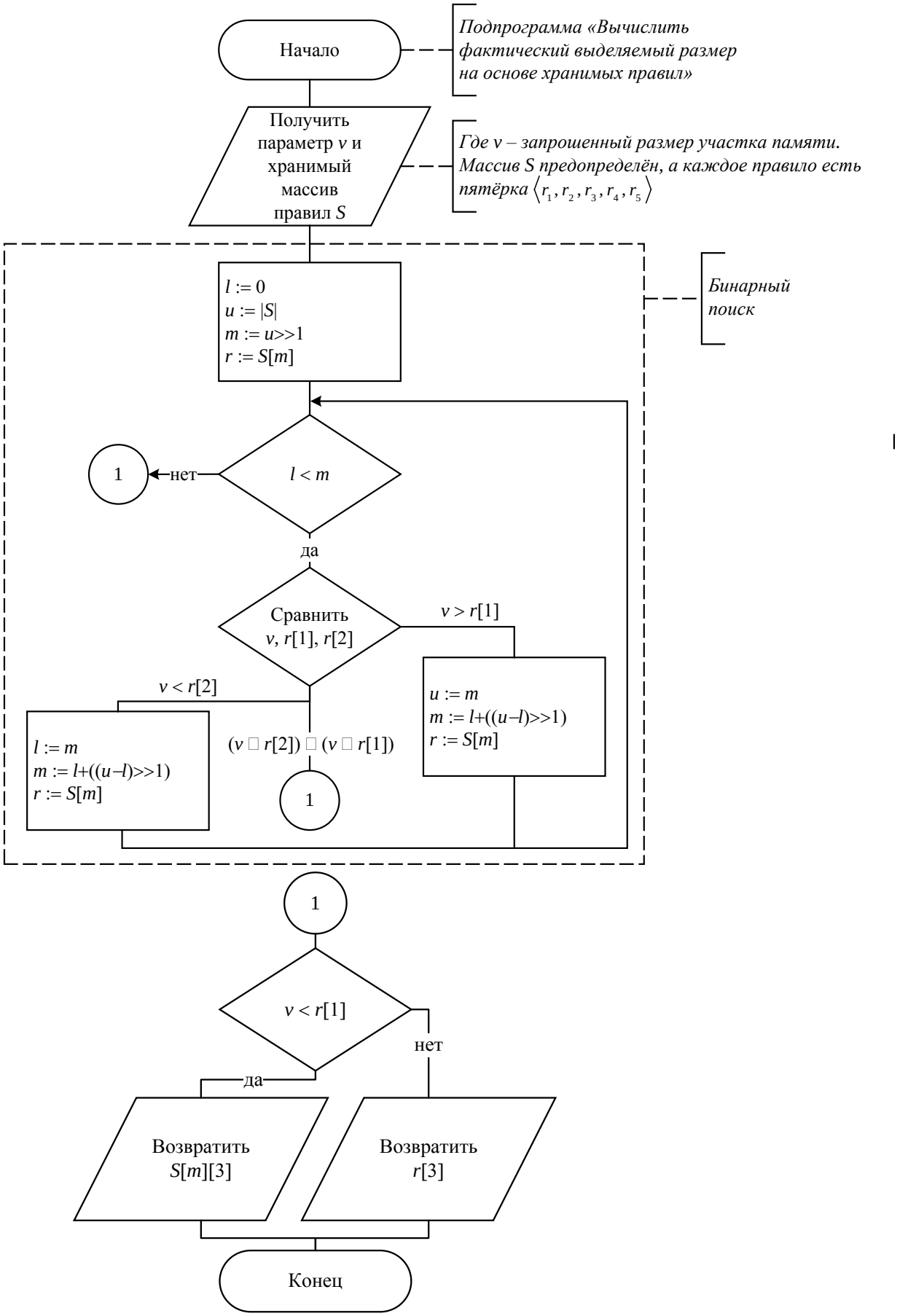
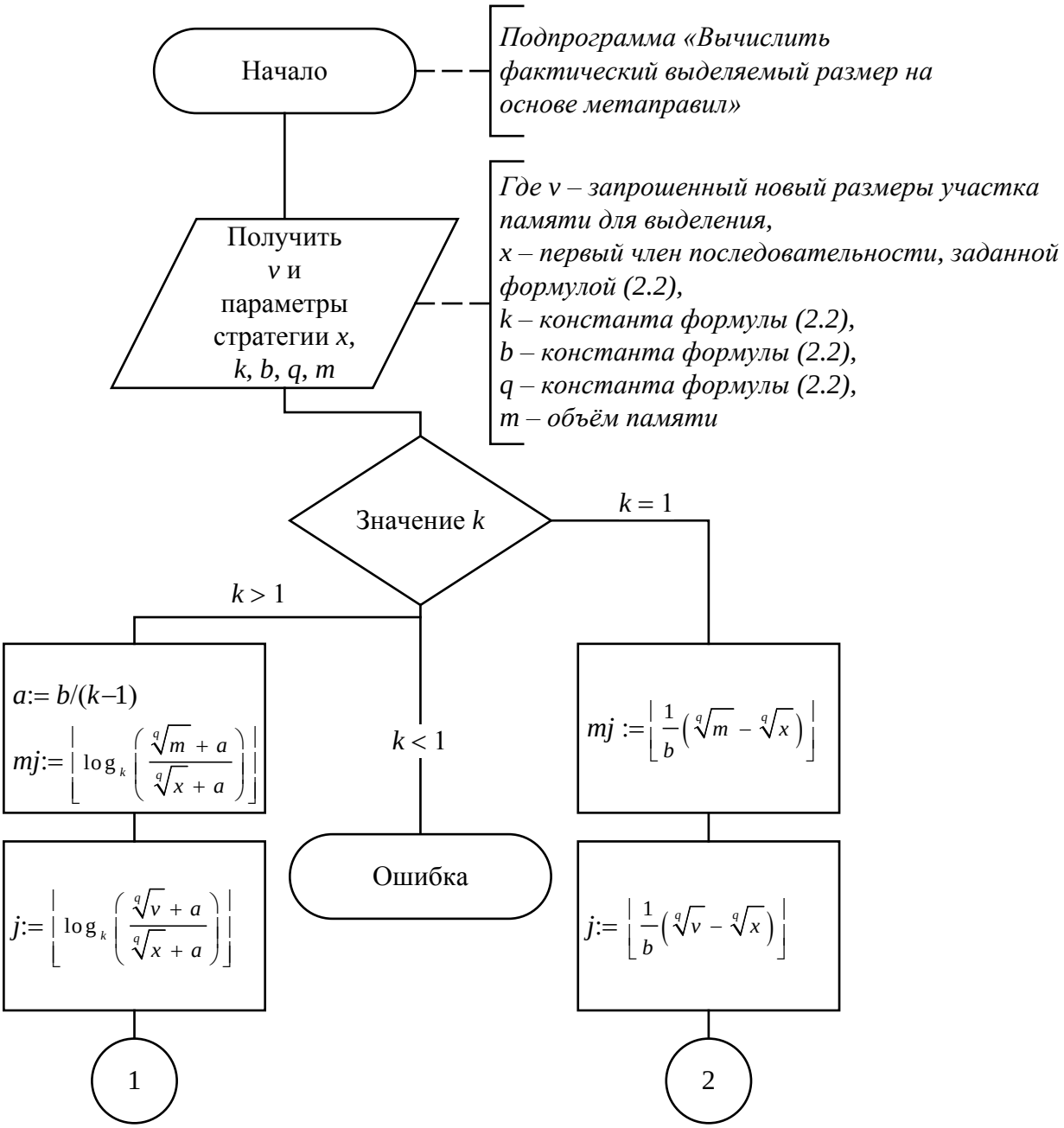
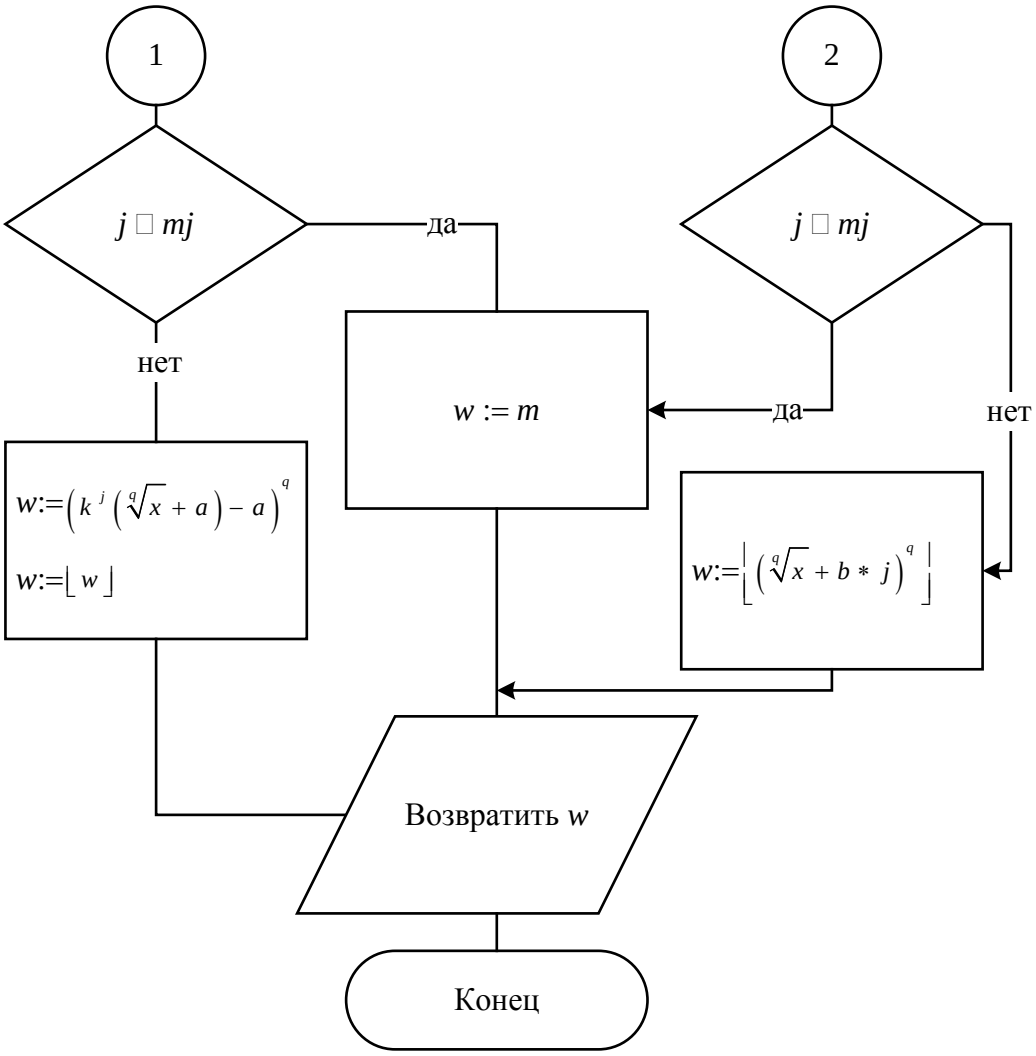


Рисунок Б.2 – Блок-схема подпрограммы «Вычислить фактический выделяемый размер на основе хранимых правил»


Начало
Подпрограмма «Вычислить фактический выделяемый размер на основе метаправил»
Получить v и параметры стратегии x, k, b, q, m
Где v – запрошенный новый размеры участка памяти для выделения,
x – первый член последовательности, заданной формулой (2.2),
k – константа формулы (2.2),
b – константа формулы (2.2),
q – константа формулы (2.2),
m – объём памяти
Значение k
k = 1
k > 1
k < 1
a:= b/(k–1)
mj:= ⌊ log_k ((√[q]m + a) / (√[q]x + a)) ⌋
mj := ⌊ 1/b (√[q]m − √[q]x) ⌋
Ошибка
j:= ⌊ log_k ((√[q]v + a) / (√[q]x + a)) ⌋
j:= ⌊ 1/b (√[q]v − √[q]x) ⌋
1
2

а)



б)

Рисунок Б.3 – Блок-схемы (*а*, *б*) подпрограммы «Вычислить фактический выделяемый размер на основе метаправил»

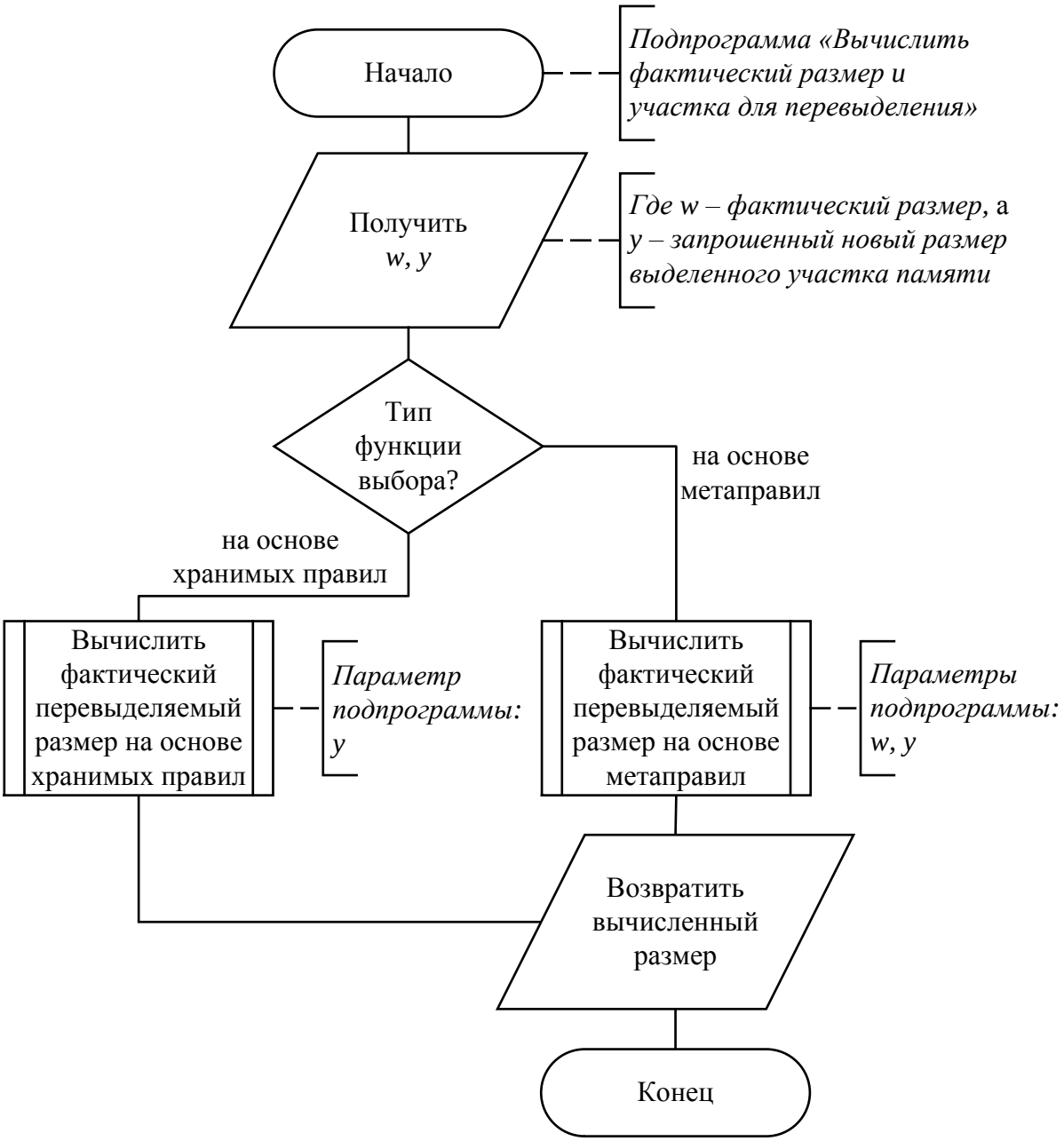
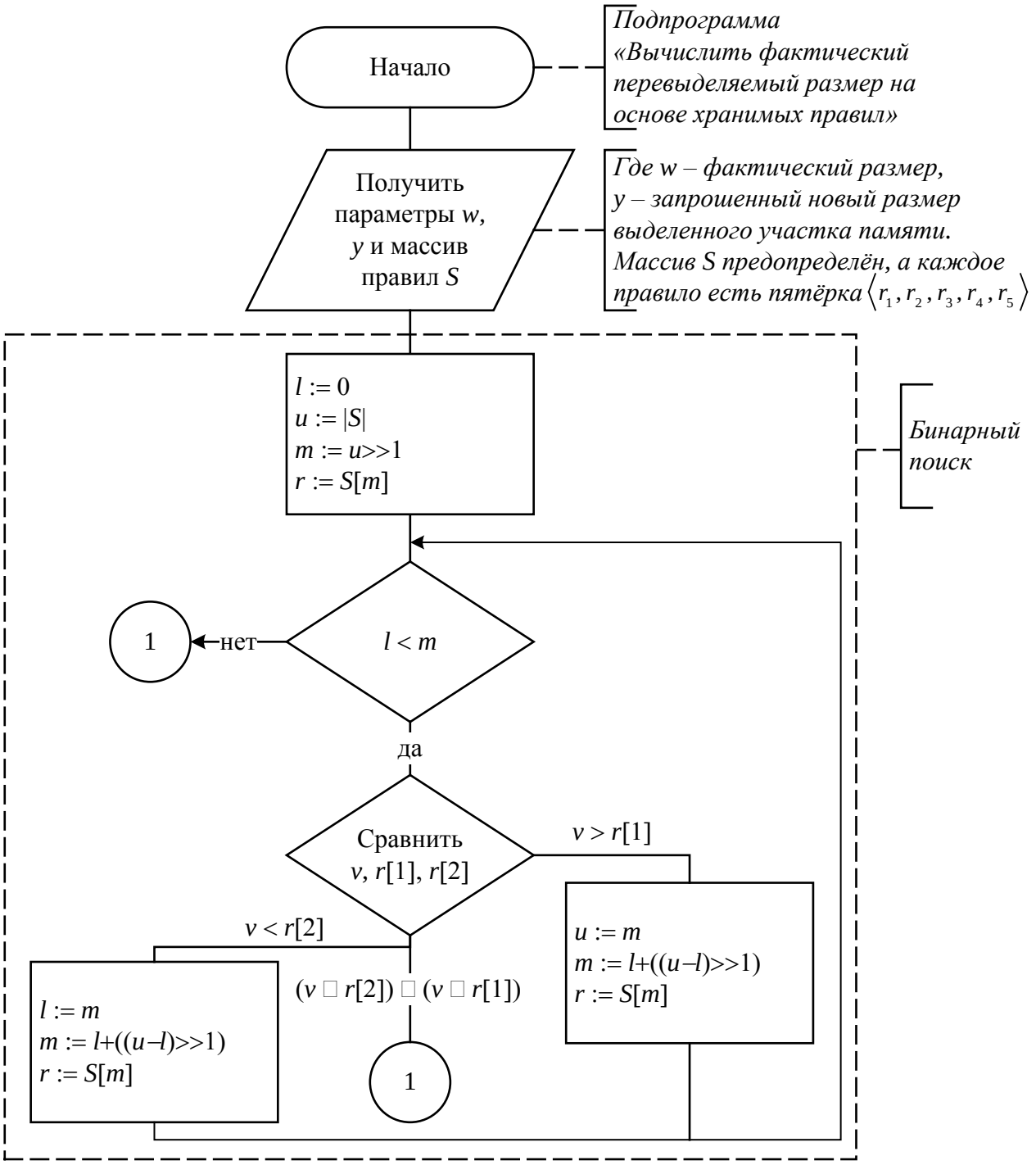
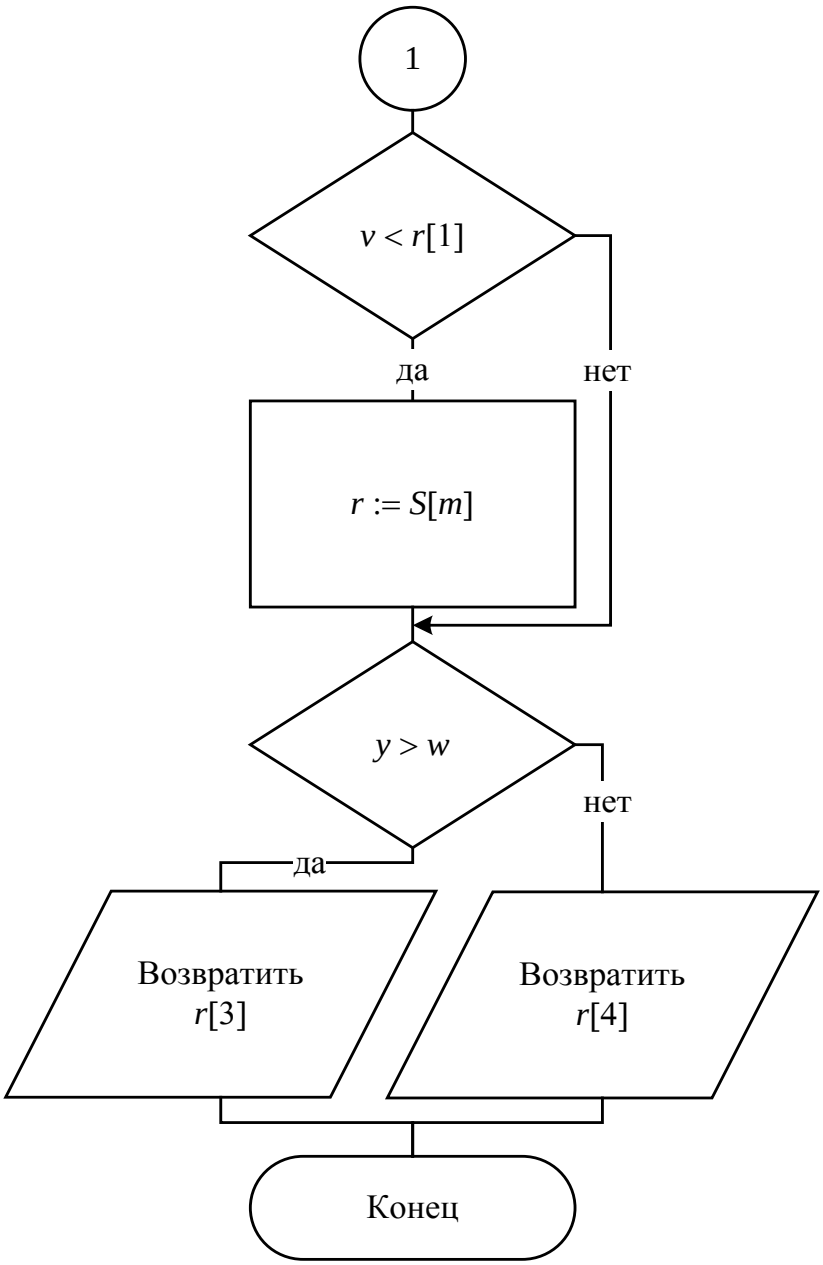


Рисунок Б.4 – Блок-схема подпрограммы «Вычислить фактический размер u участка для перевыделения»

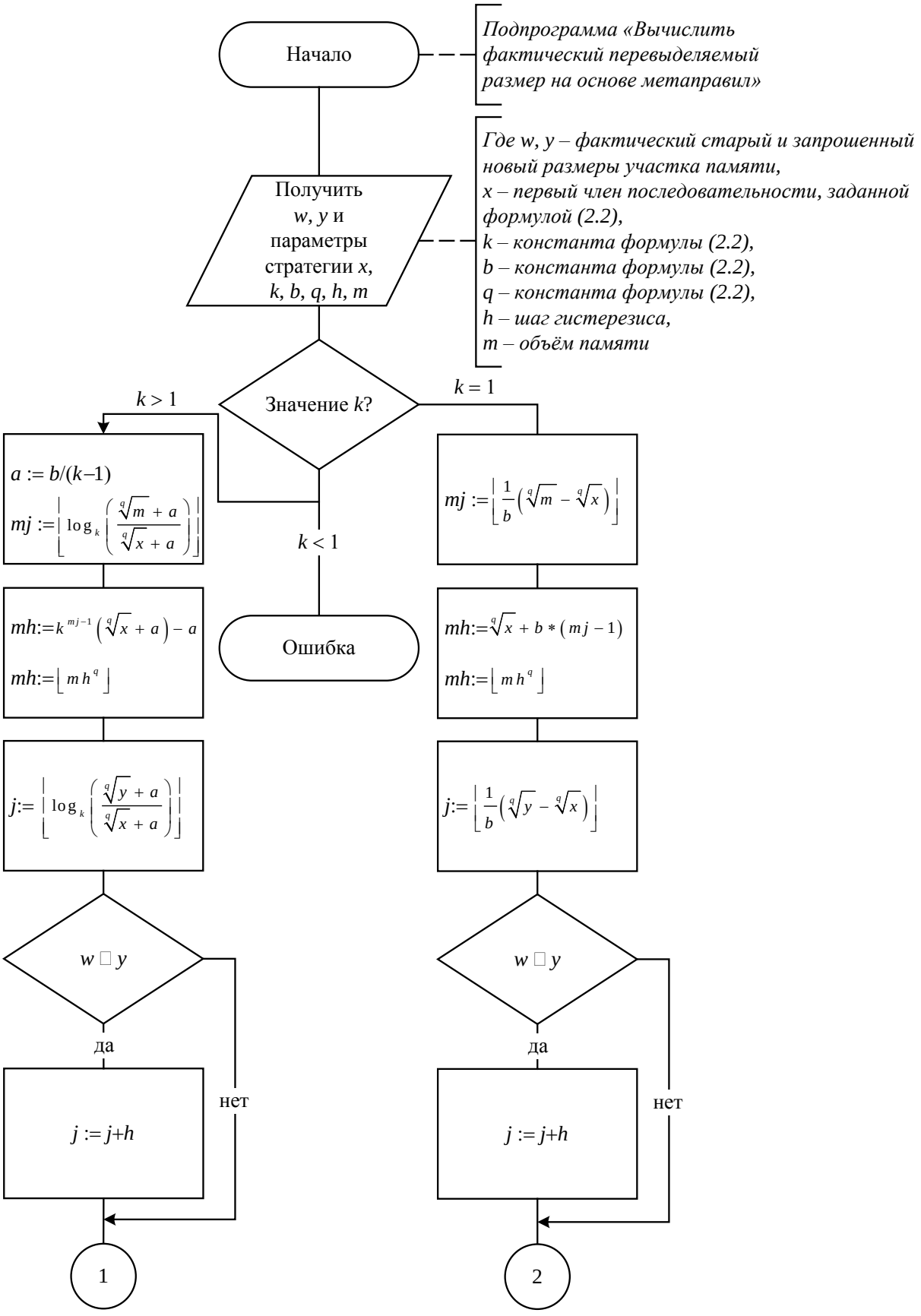

Начало
Подпрограмма «Вычислить фактический перевыделяемый размер на основе хранимых правил»
Получить параметры w, y и массив правил S
Где w – фактический размер, y – запрошенный новый размер выделенного участка памяти. Массив S предопределён, а каждое правило есть пятёрка ⟨r1, r2, r3, r4, r5⟩
l := 0
u := |S|
m := u>>1
r := S[m]
Бинарный поиск
l < m
нет
1
да
Сравнить v, r[1], r[2]
v > r[1]
u := m
m := l+((u–l)>>1)
r := S[m]
v < r[2]
l := m
m := l+((u–l)>>1)
r := S[m]
(v □ r[2]) □ (v □ r[1])
1

а)

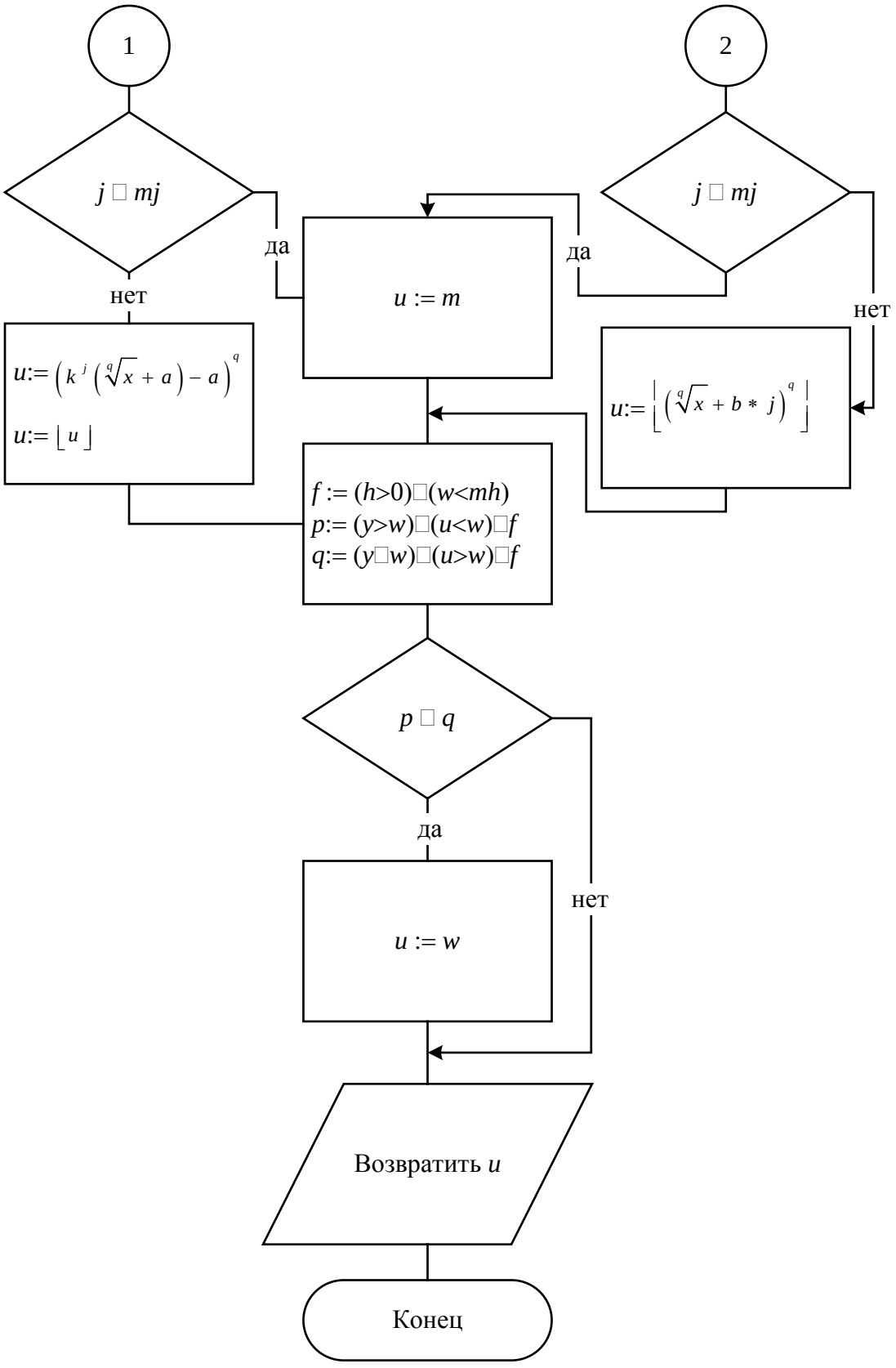


б)

Рисунок Б.5 – Блок-схемы (*а*, *б*) подпрограммы «Вычислить фактический перевыделяемый размер на основе хранимых правил»


Начало
Подпрограмма «Вычислить фактический перевыделяемый размер на основе метаправил»
Получить w, y и параметры стратегии x, k, b, q, h, m
Где w, y – фактический старый и запрошенный новый размеры участка памяти,
x – первый член последовательности, заданной формулой (2.2),
k – константа формулы (2.2),
b – константа формулы (2.2),
q – константа формулы (2.2),
h – шаг гистерезиса,
m – объём памяти
Значение k?
k > 1
k = 1
k < 1
a := b/(k–1)
mj := ⌊ log_k ((√[q](m) + a) / (√[q](x) + a)) ⌋
mh := k^(mj−1) (√[q](x) + a) − a
mh := ⌊ mh^q ⌋
j := ⌊ log_k ((√[q](y) + a) / (√[q](x) + a)) ⌋
mj := ⌊ (1/b) (√[q](m) − √[q](x)) ⌋
mh := √[q](x) + b * (mj − 1)
mh := ⌊ mh^q ⌋
j := ⌊ (1/b) (√[q](y) − √[q](x)) ⌋
w □ y
да
нет
j := j+h
Ошибка
1
2

а)



б)

Рисунок Б.6 – Блок-схема подпрограммы «Вычислить фактический перевыделяемый размер на основе метаправил»

ПРИЛОЖЕНИЕ В

В1 Блок-схемы алгоритма пересечения мультимножеств

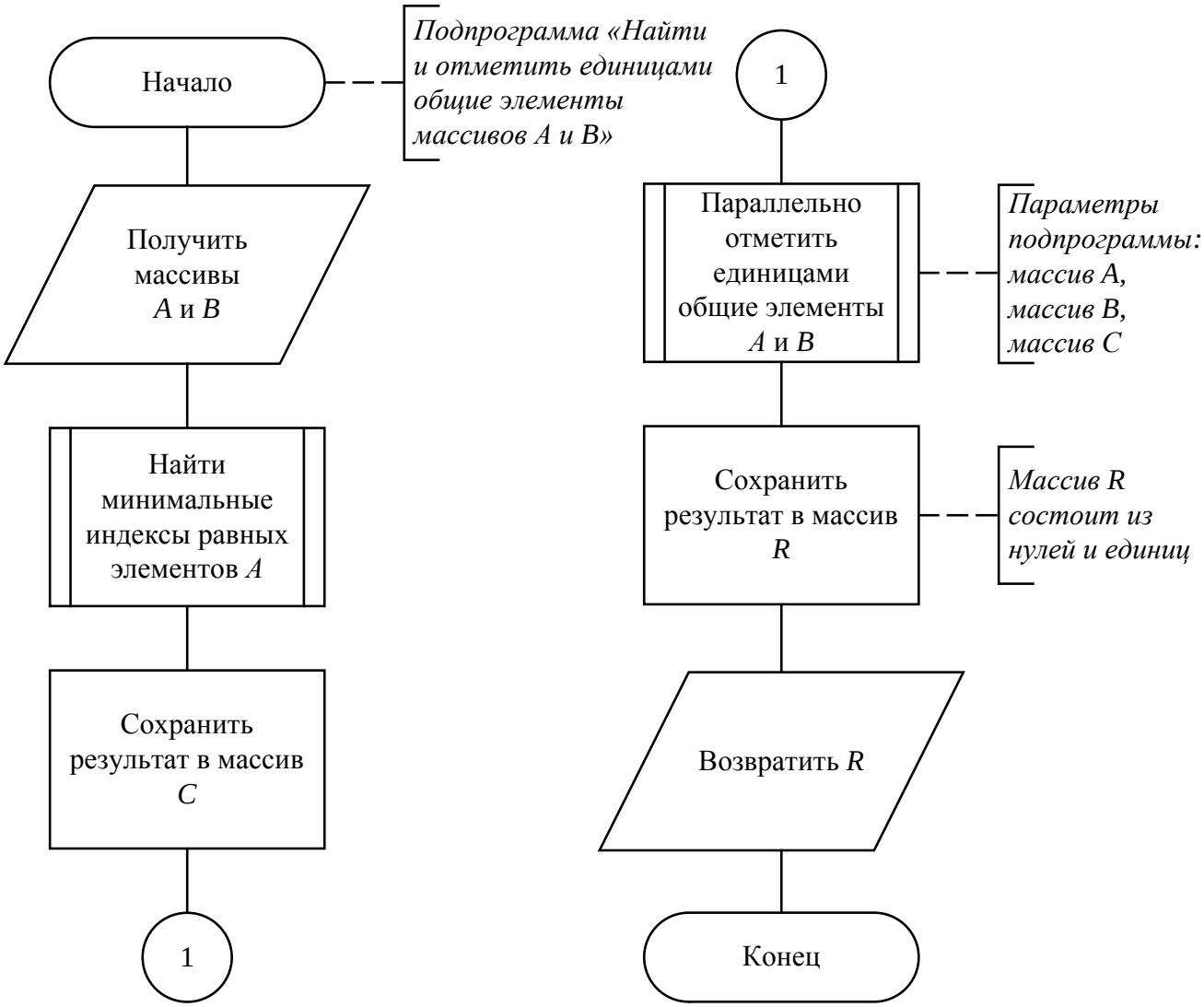


Рисунок В1.1 – Блок-схема подпрограммы «Найти и отметить единицами общие элементы массивов A и B»

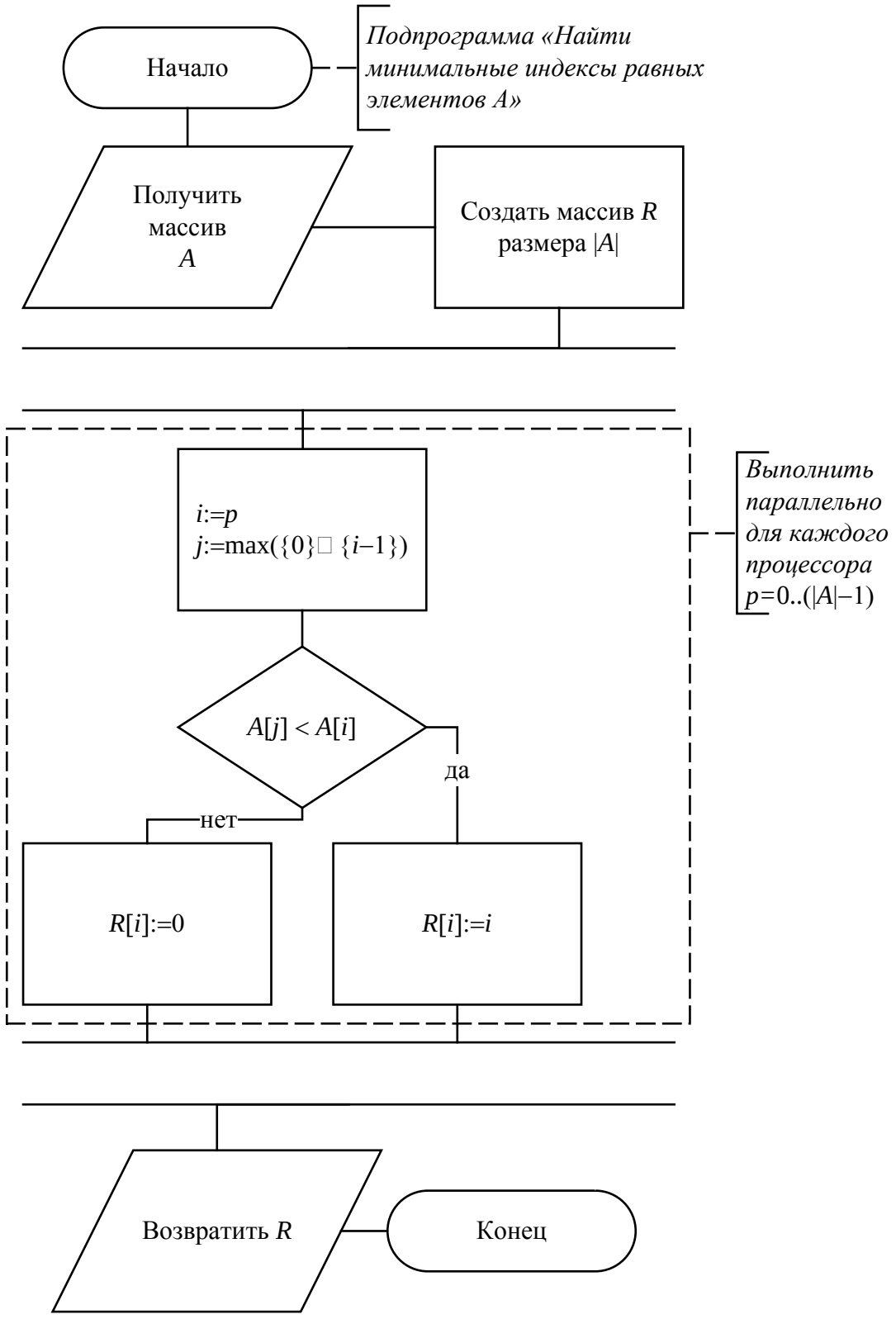


Рисунок В1.2 – Блок-схема подпрограммы «Найти минимальные индексы равных элементов А»

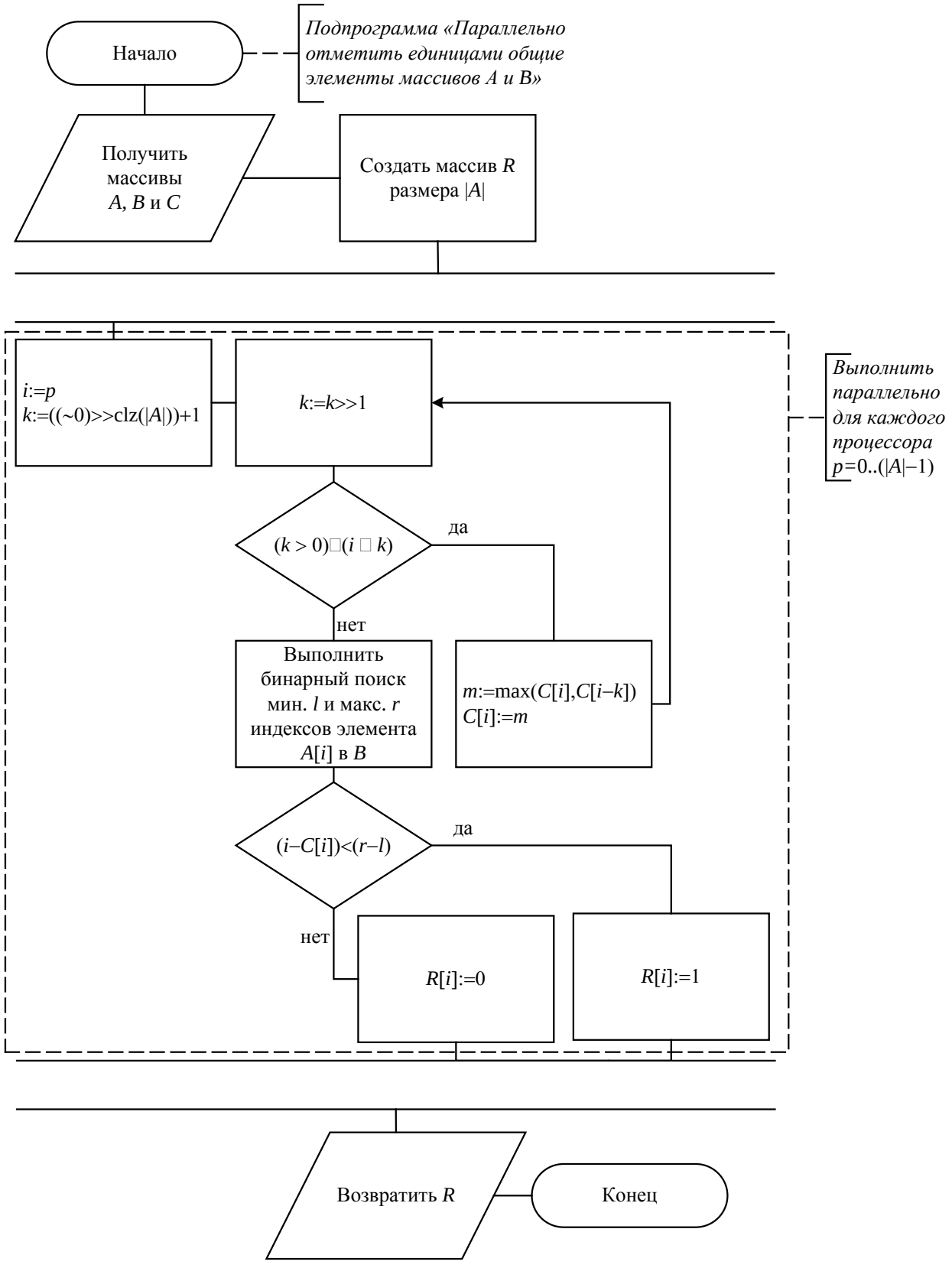


Рисунок В1.3 – Блок-схема подпрограммы «Параллельно отметить единицами общие элементы массивов A и B»

В2 Блок-схемы алгоритма разности мультимножеств

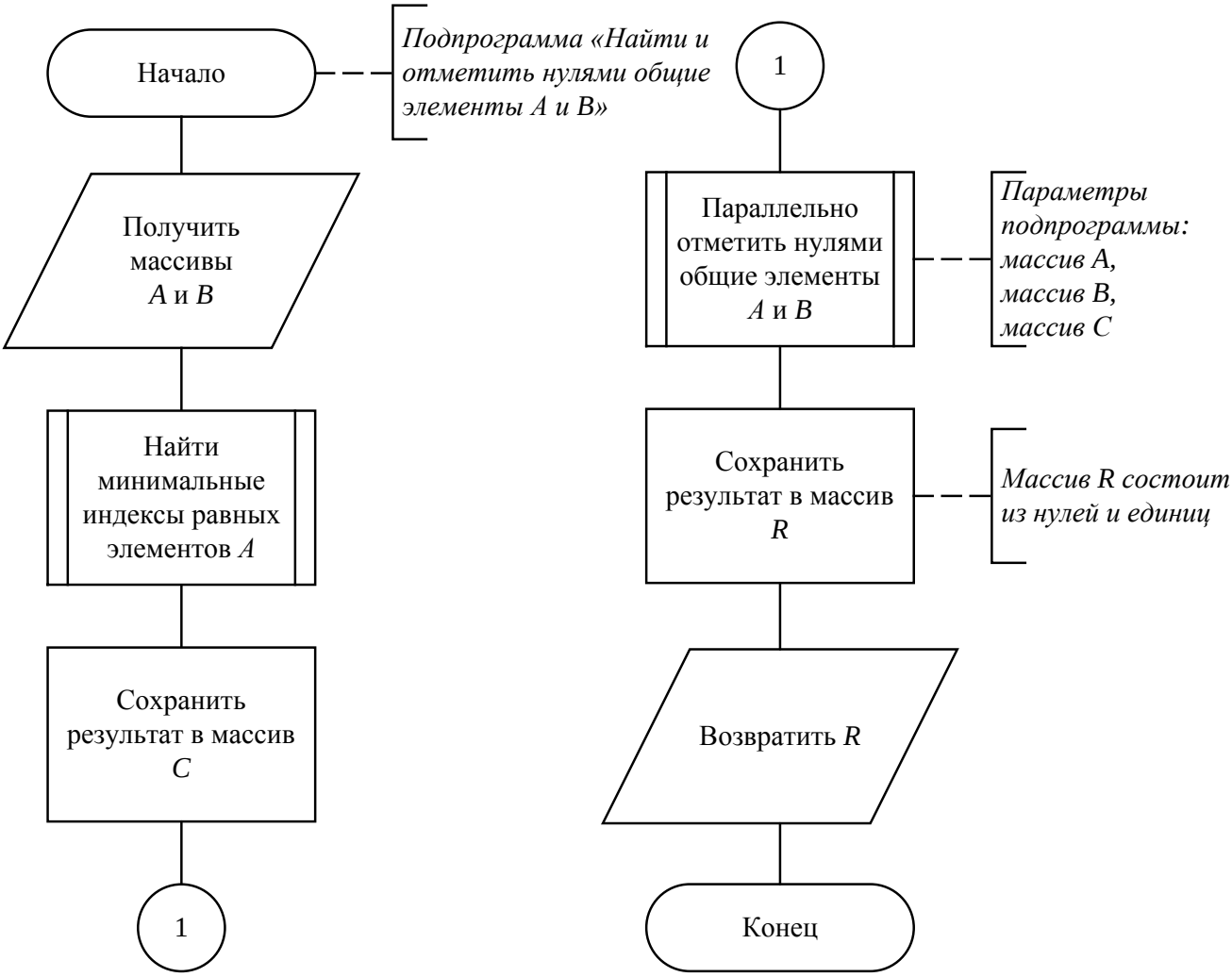


Рисунок В2.1 – Блок-схема подпрограммы
«Найти и отметить нулями общие элементы A и B»

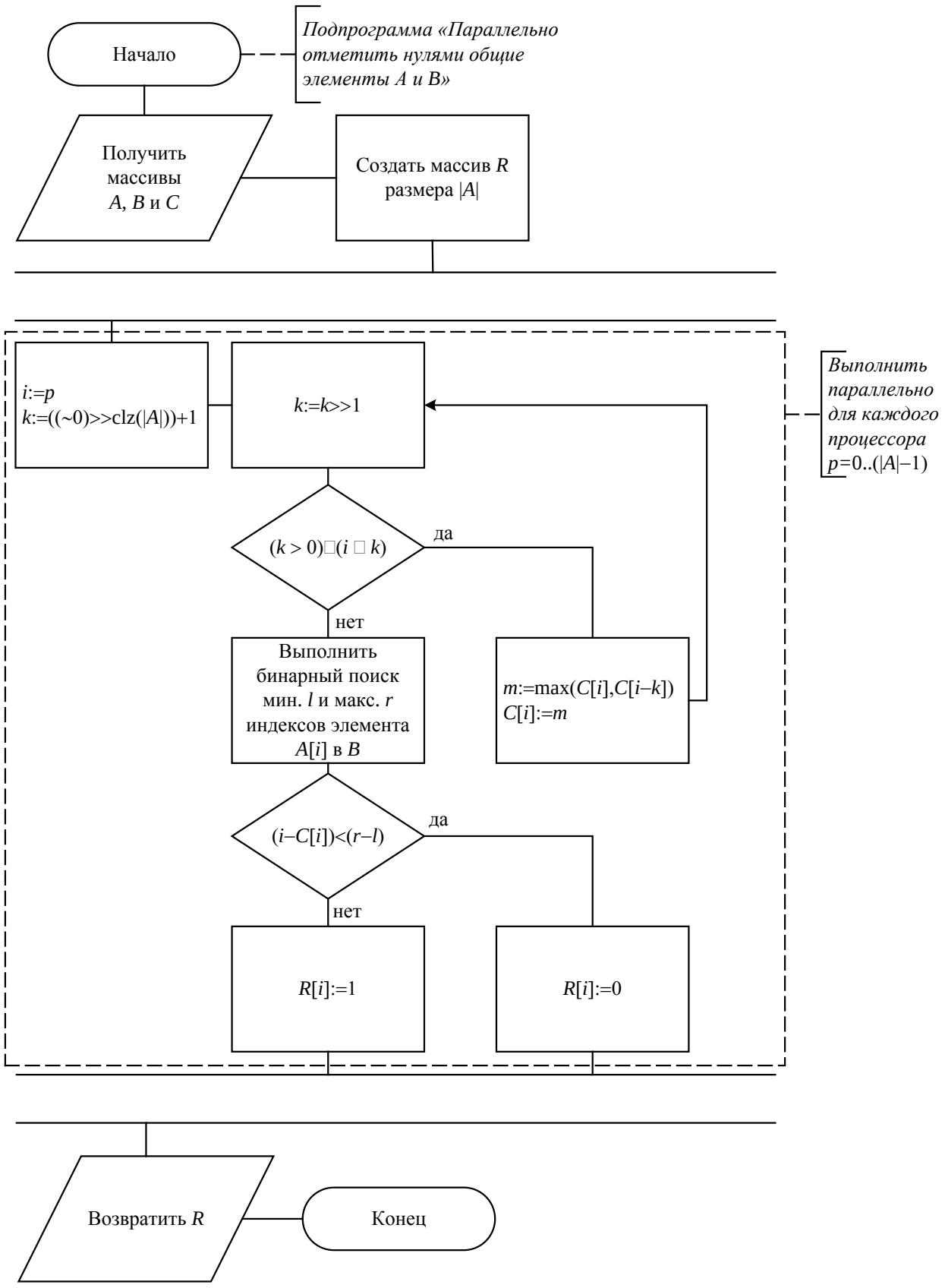


Рисунок В2.2 – Блок-схема подпрограммы «Параллельно отметить нулями общие элементы A и B»

В3 Блок-схемы алгоритма объединения мультимножеств

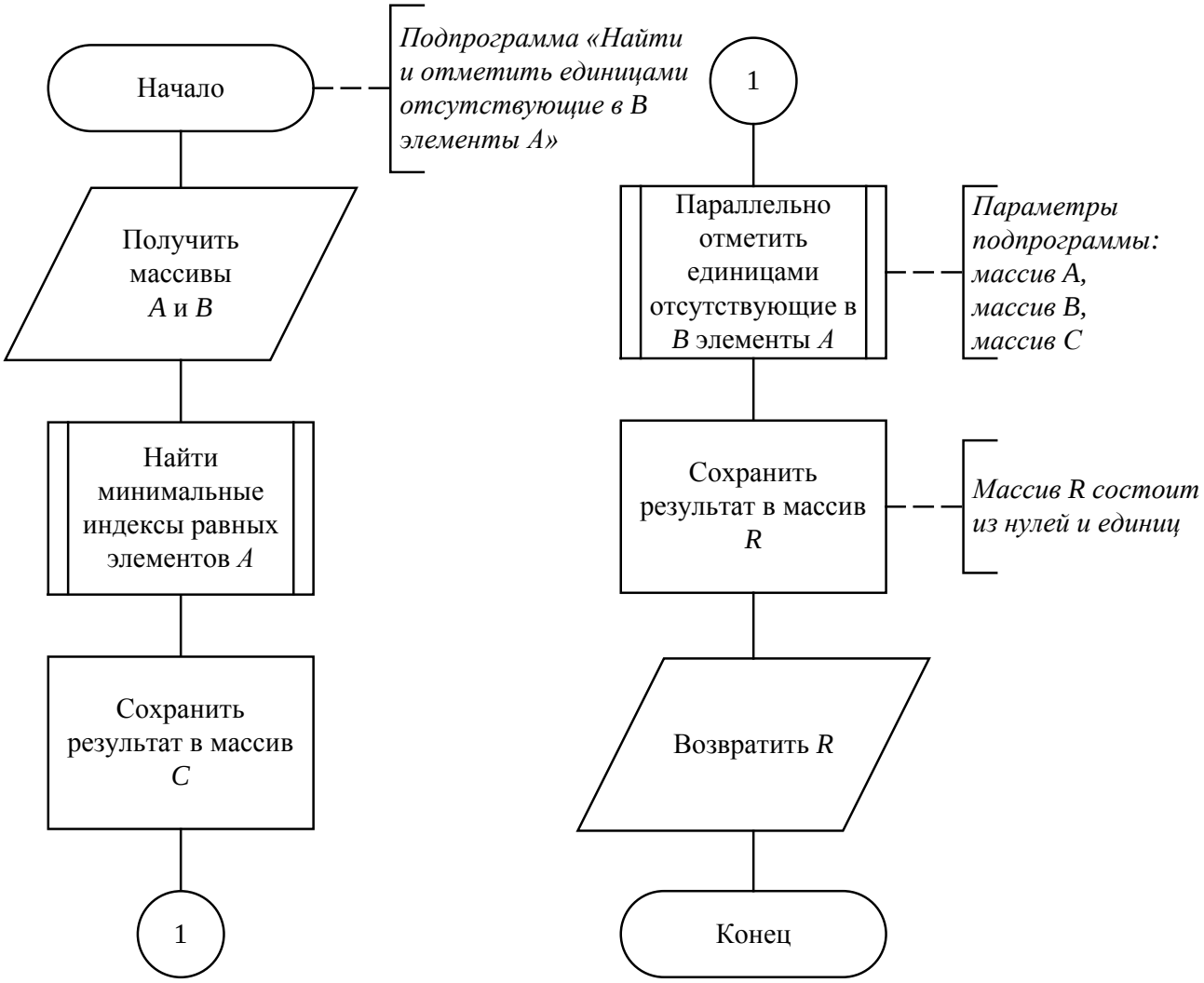


Рисунок В3.1 – Блок-схема подпрограммы «Найти и отметить единицами отсутствующие в B элементы A»

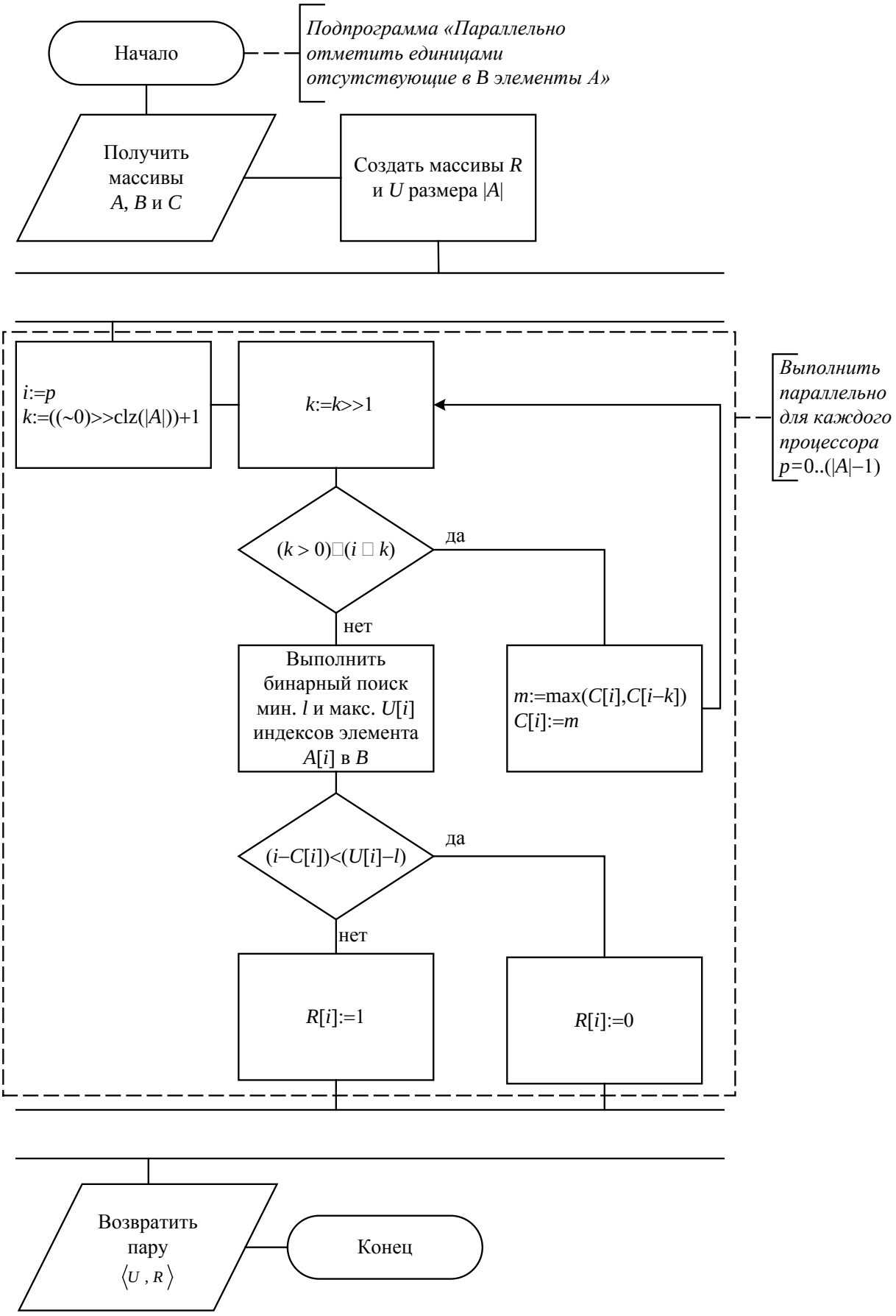


Рисунок В3.2 – Блок-схема подпрограммы «Параллельно отметить единицами отсутствующие в В элементы А»

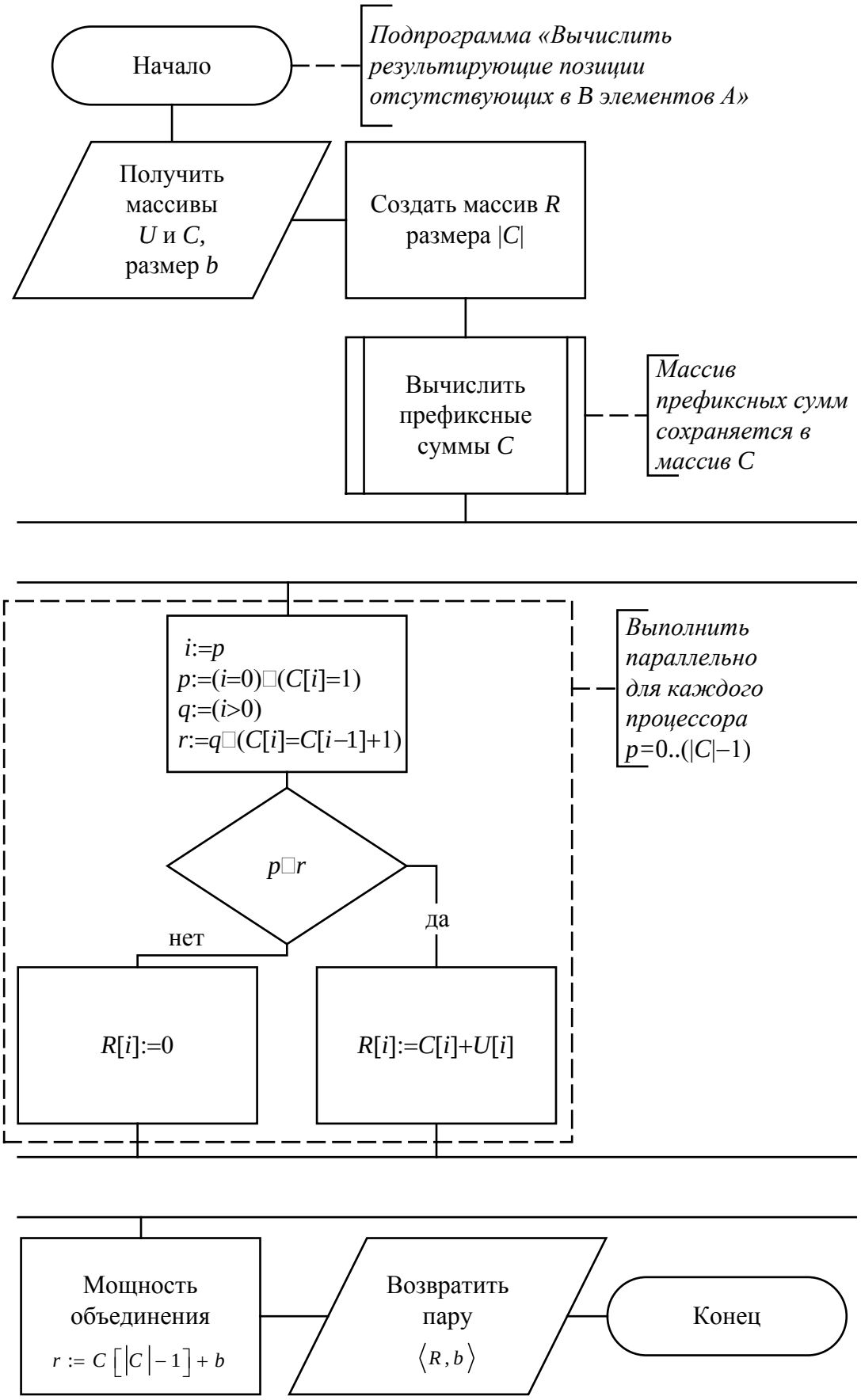


Рисунок В3.3 – Блок-схема подпрограммы «Вычислить результирующие позиции отсутствующих в B элементов A»

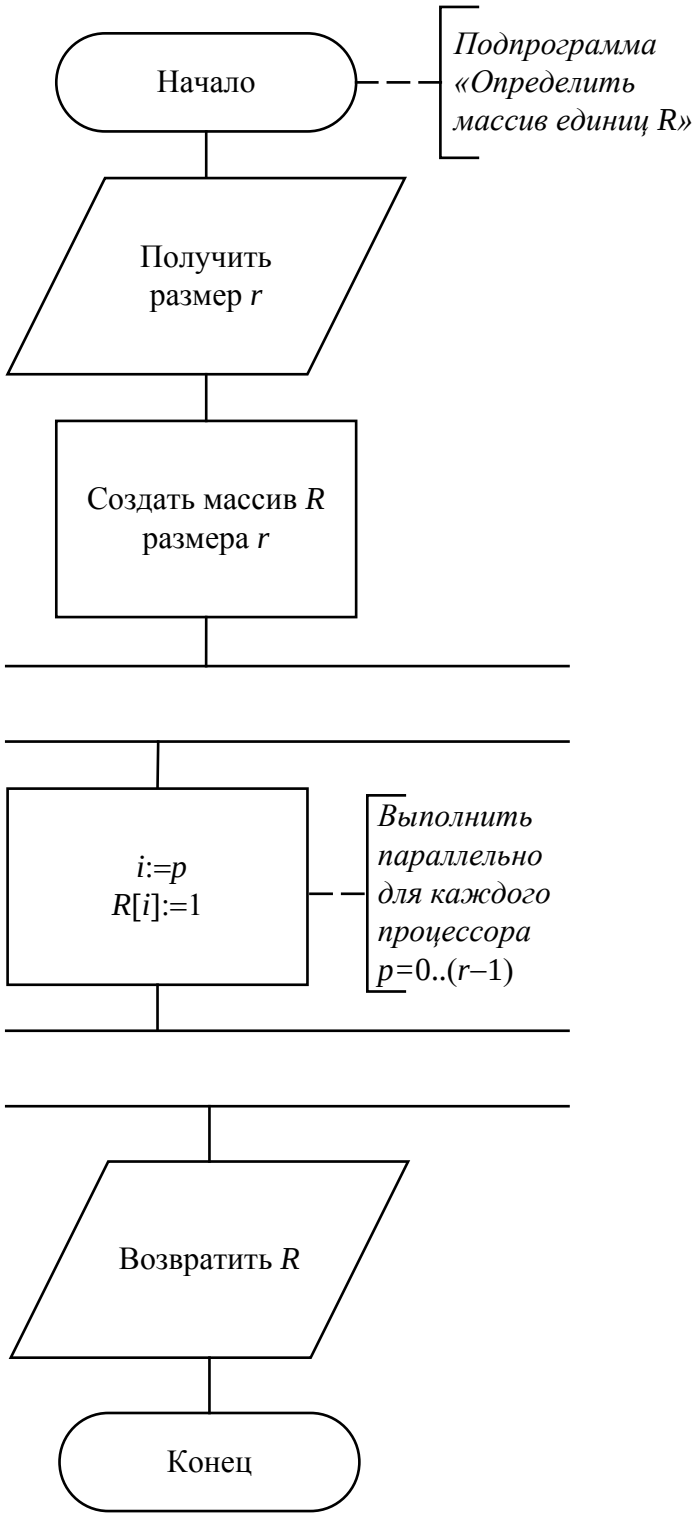


Рисунок В3.4 – Блок-схема подпрограммы «Определить массив единиц R»

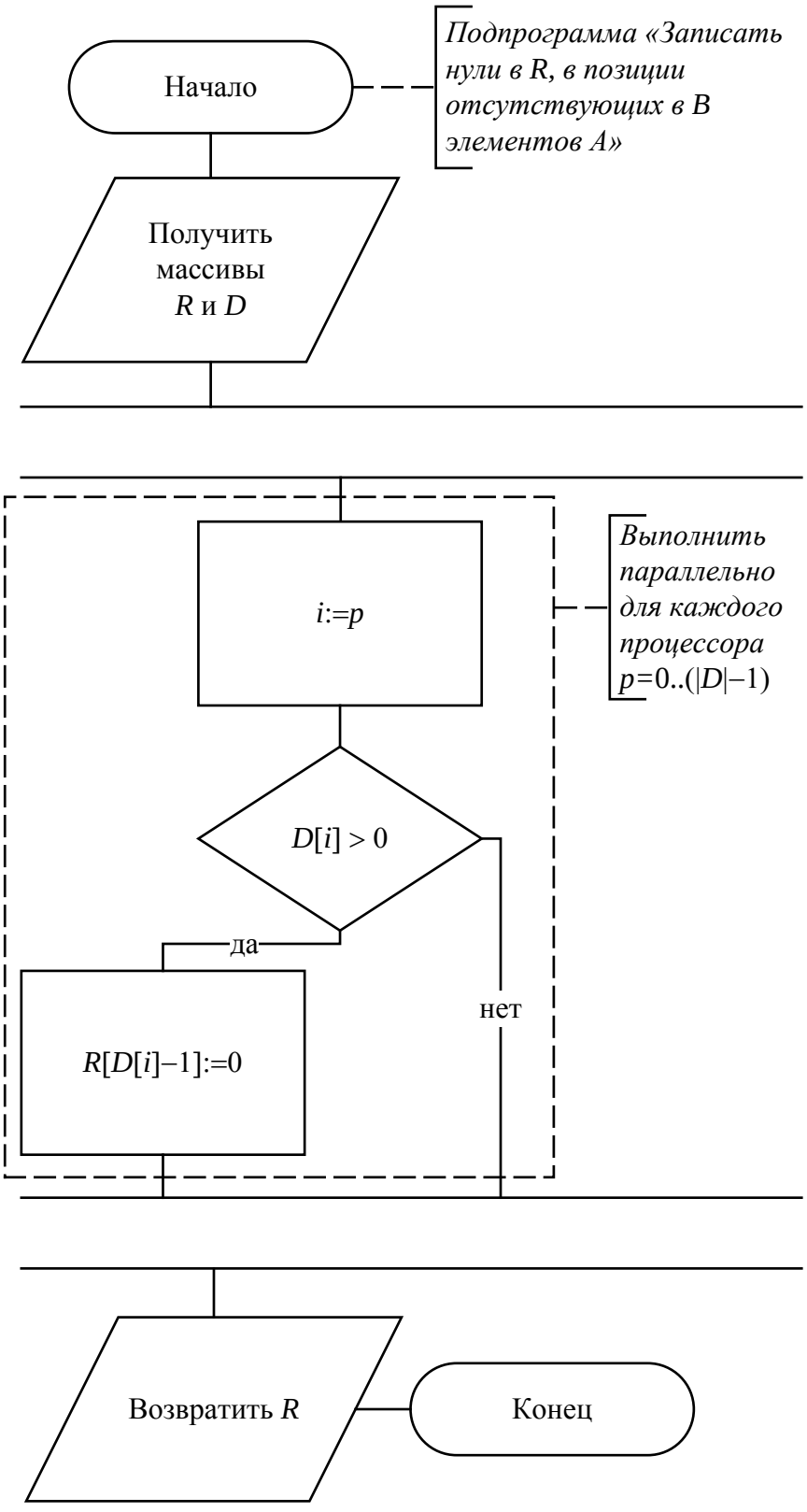


Рисунок В3.5 – Блок-схема подпрограммы «Записать нули в R, в позиции отсутствующих в B элементов A»

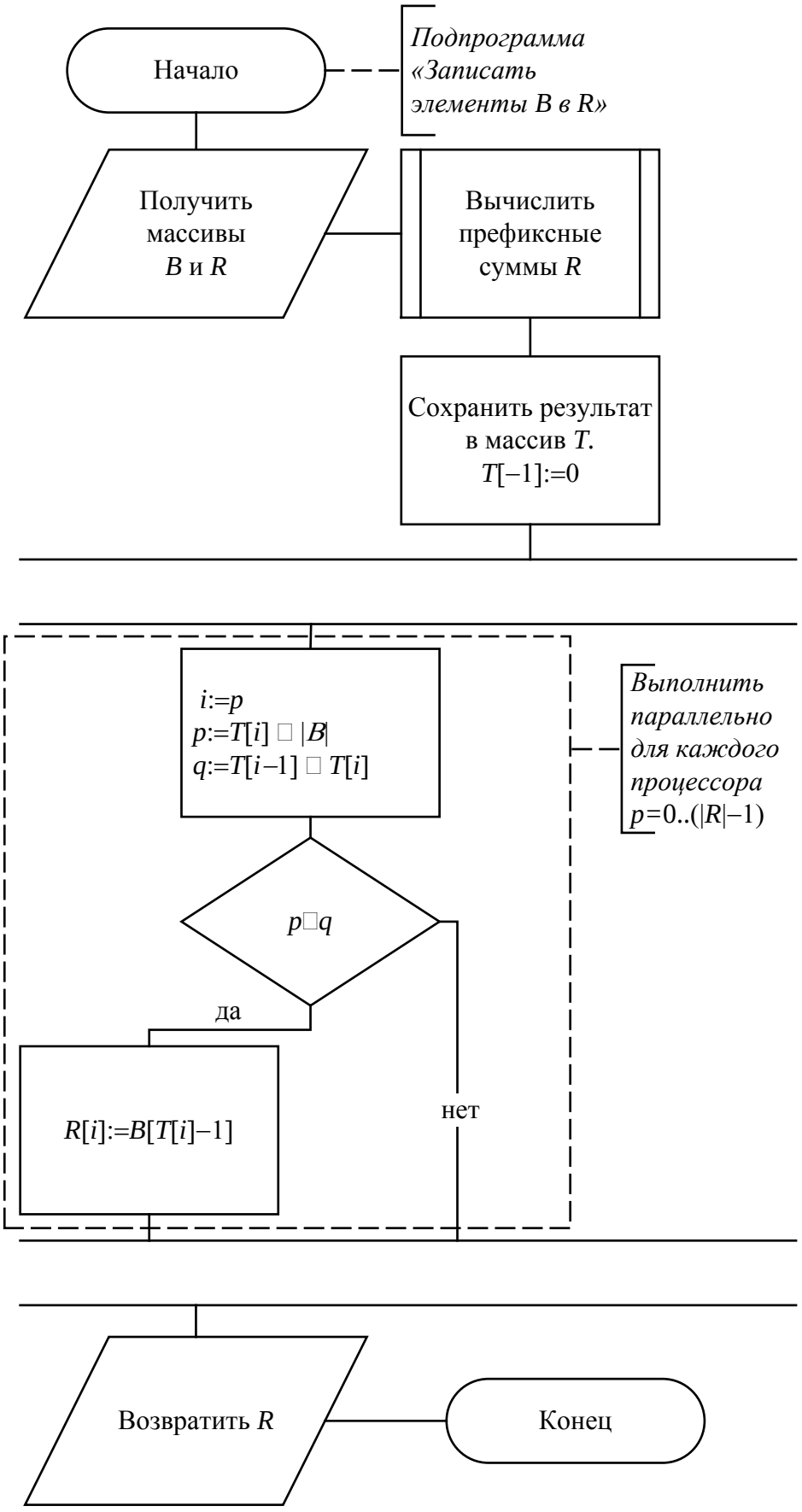


Рисунок В3.6 – Блок-схема подпрограммы «Записать элементы B в R»

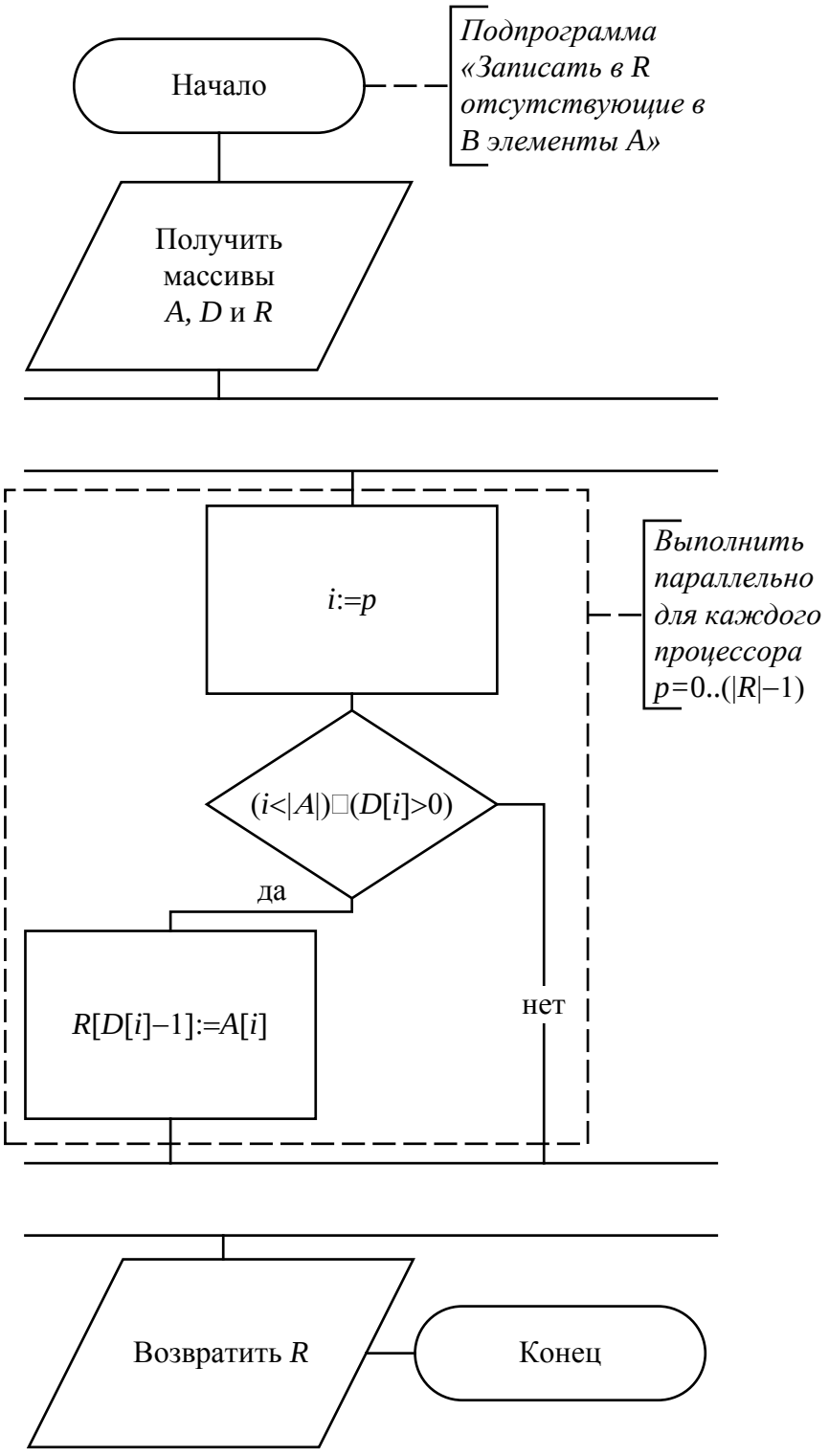


Рисунок В3.7 – Блок-схема подпрограммы «Записать в R отсутствующие в B элементы A»

ПРИЛОЖЕНИЕ Г

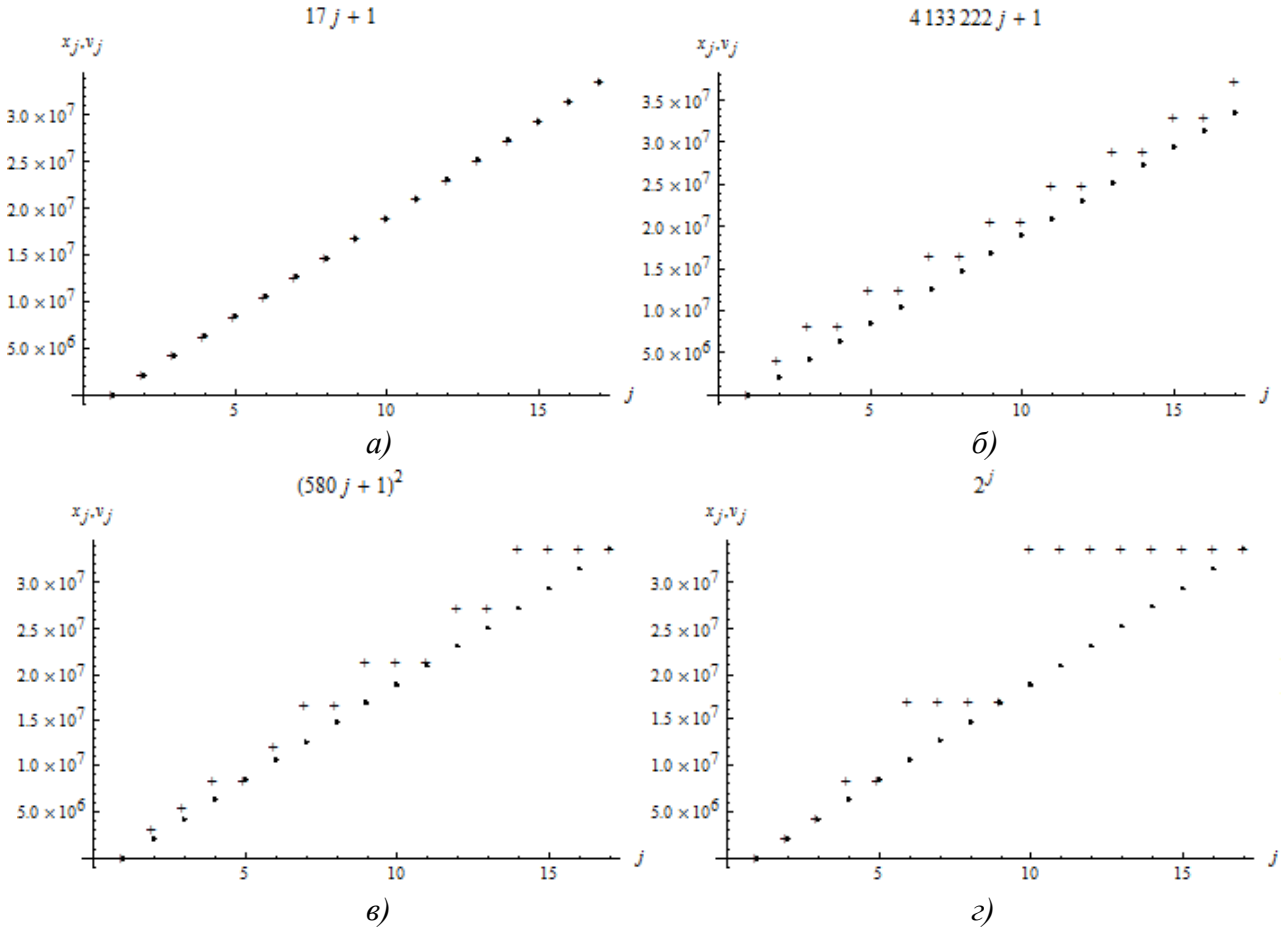


Рисунок Г.1 – Графики (*а*, *б*, *в*, *г*) зависимости фактического размера x_j выделяемого участка памяти и требуемого размера v_j участка памяти от номера правила *j* соответственно для линейной (а), линейной (б), степенной и геометрической стратегий

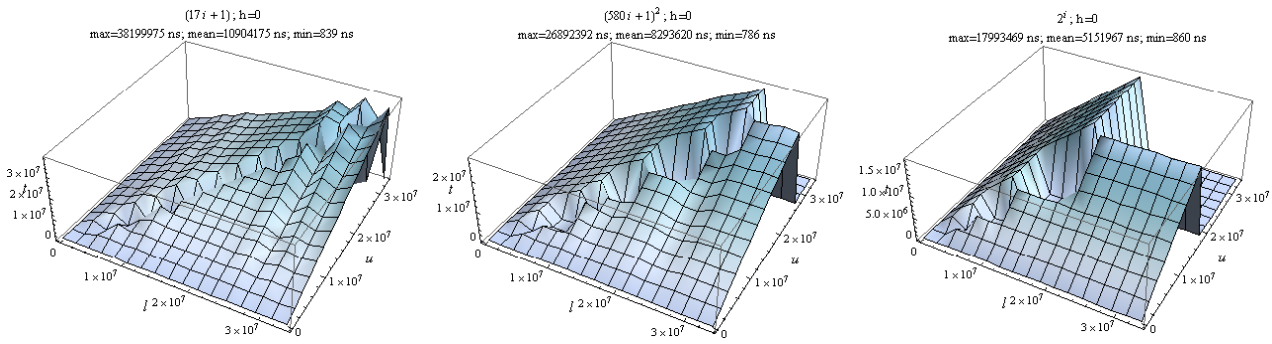


Рисунок Г.2 – Графики зависимости фактического времени исполнения операций перевыделения для стратегий без гистерезиса при использовании функции поиска на основе метаправил от начального и конечного размеров перевыделяемого участка памяти

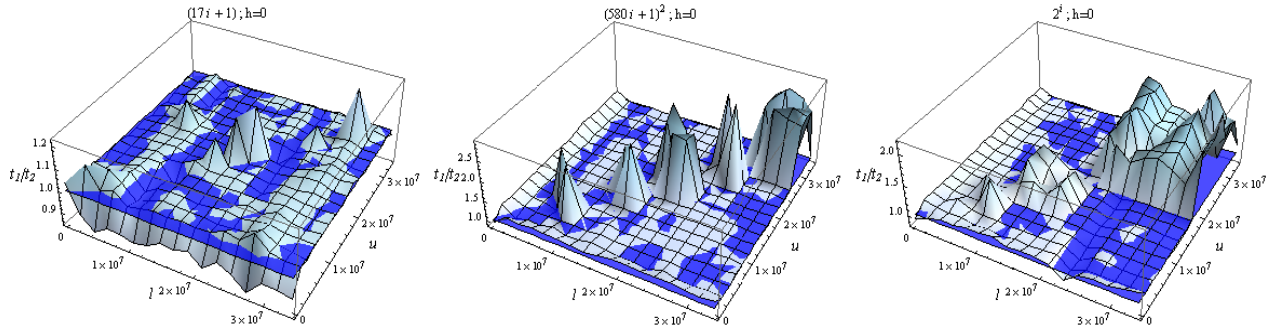


Рисунок Г.3 – Графики зависимости отношения времён исполнения операций перевыделения для стратегий без гистерезиса при использовании функции поиска на основе метаправил (t_1) и на основе хранимых правил (t_2) от начального и конечного размеров перевыделяемого участка памяти

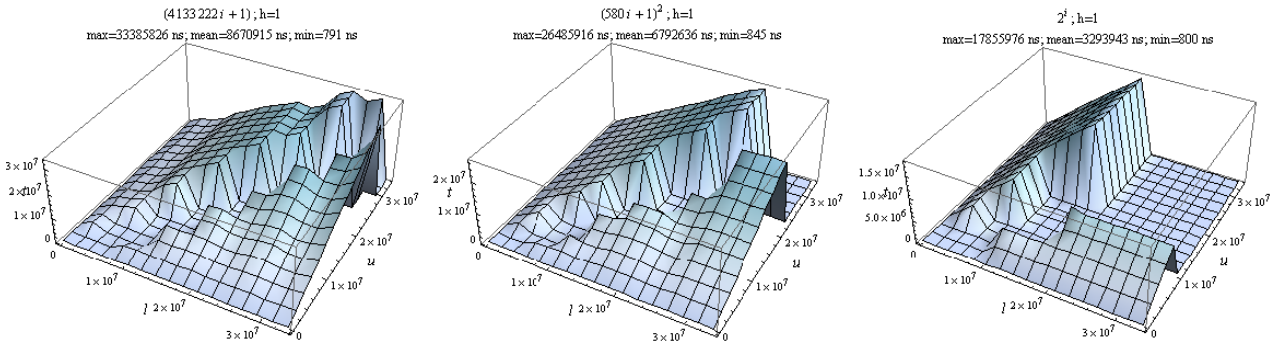


Рисунок Г.4 – Графики зависимости фактического времени исполнения операций перевыделения для стратегий без гистерезиса при использовании функции поиска на основе хранимых правил от начального и конечного размеров перевыделяемого участка памяти

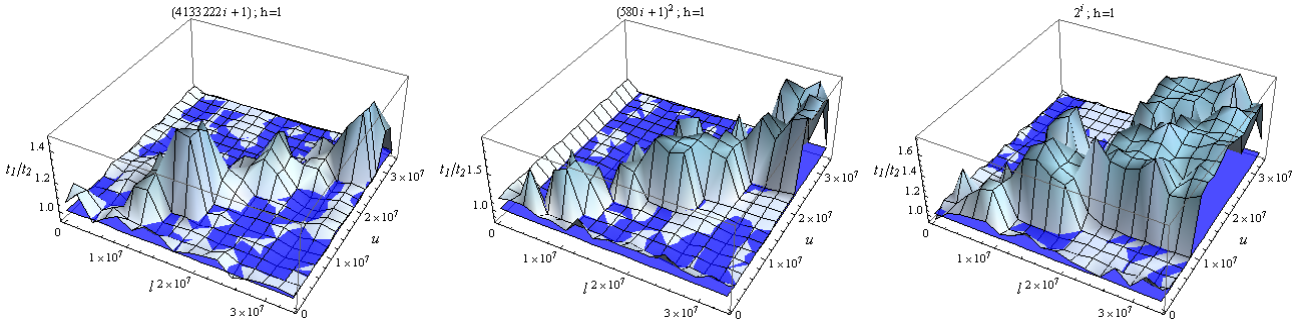


Рисунок Г.5 – Графики зависимости отношения времён исполнения операций перевыделения для стратегий с гистерезисом при использовании функции поиска на основе метаправил (t_1) и на основе хранимых правил (t_2) от начального и конечного размеров перевыделяемого участка памяти

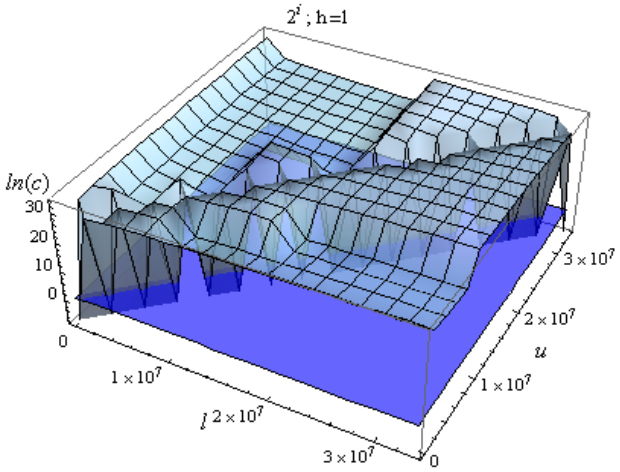


Рисунок Г.6 – График зависимости натурального логарифма пропускной способности операции перевыделения при использовании геометрической стратегии с гистерезисом от начального и конечного размеров перевыделяемого участка памяти

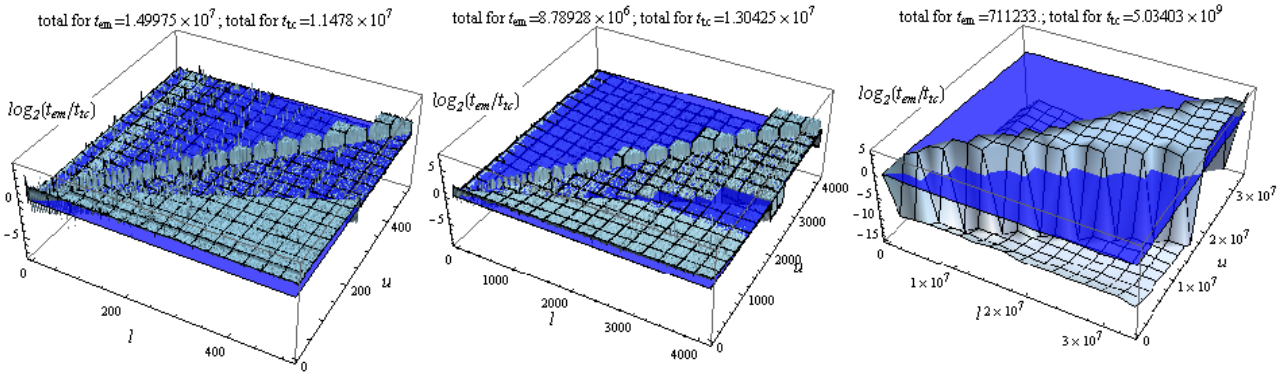


Рисунок Г.7 – Графики зависимости двоичного логарифма отношения времён исполнения (в нс) операций перевыделения в системах em (t_{em}) и tcmalloc (t_{tc}) для максимальных размеров перевыделяемого участка (слева направо) 512 ячеек, 4114 ячеек и 2^{25} ячеек в случае, когда память нефрагментирована

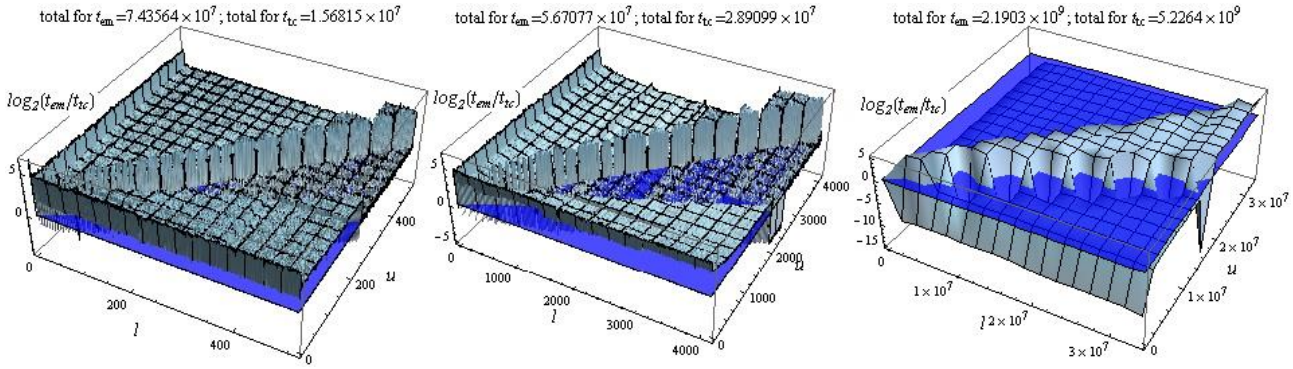


Рисунок Г.8 – Графики зависимости двоичного логарифма отношения времён исполнения (в нс) операций перевыделения в системах em (t_{em}) и tcmalloc (t_{tc}) для максимальных размеров перевыделяемого участка (слева направо) 512 ячеек, 4114 ячеек и 2^{25} ячеек в случае, когда память фрагментирована (δ=5, p=1024 для размеров 512 и 4114 ячеек и p=1 для 2^{25} ячеек)

ПРИЛОЖЕНИЕ Д

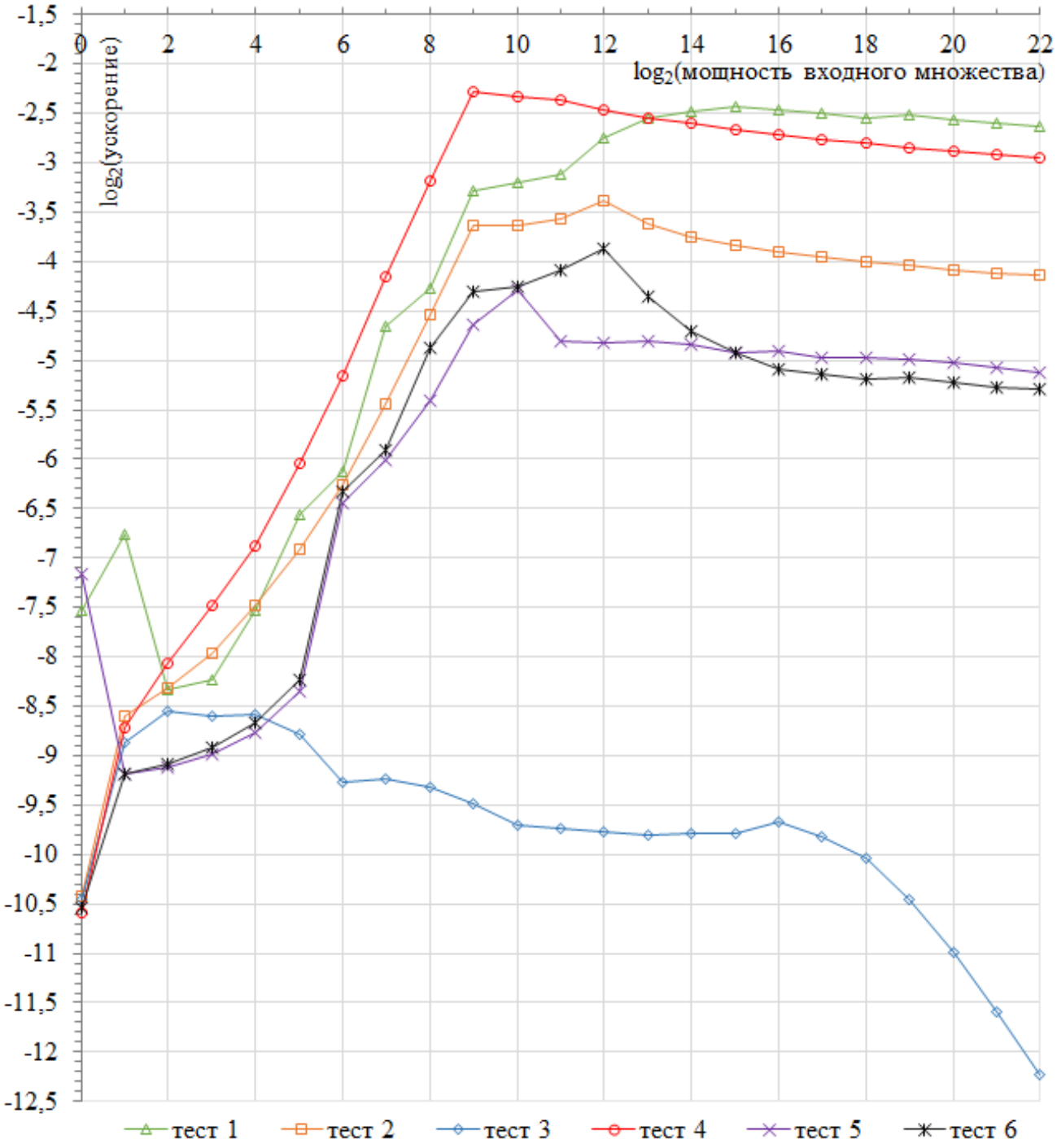


Рисунок Д.1 – Графики зависимости логарифма коэффициента ускорения по основанию 2 параллельной реализации операции пересечения по сравнению с последовательной реализацией (на устройстве CPU) от логарифма мощности пересекаемых множеств по основанию 2

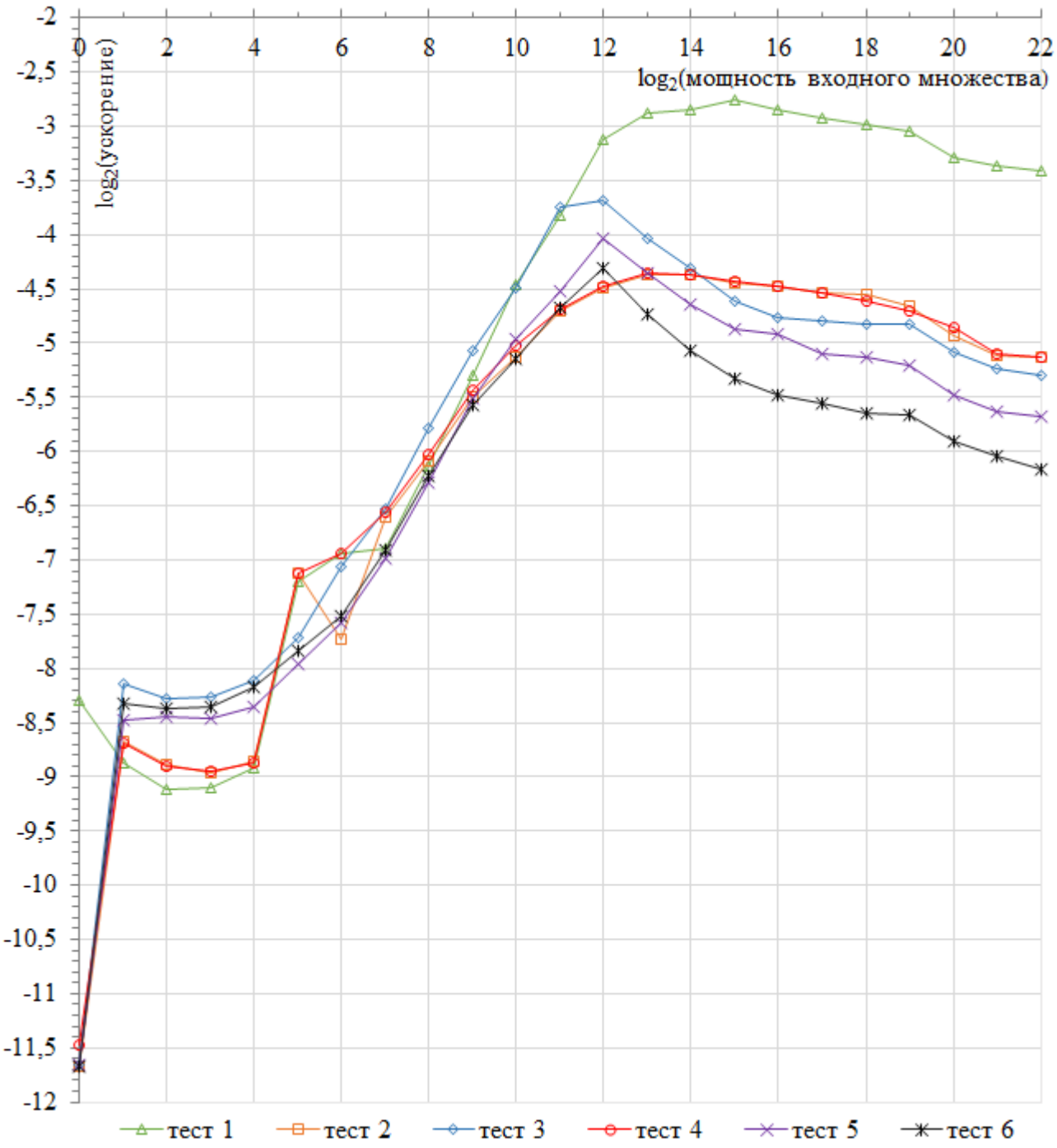


Рисунок Д.2 – Графики зависимости логарифма коэффициента ускорения по основанию 2 параллельной реализации операции объединения по сравнению с последовательной реализацией (на устройстве CPU) от логарифма мощности пересекаемых множеств по основанию 2