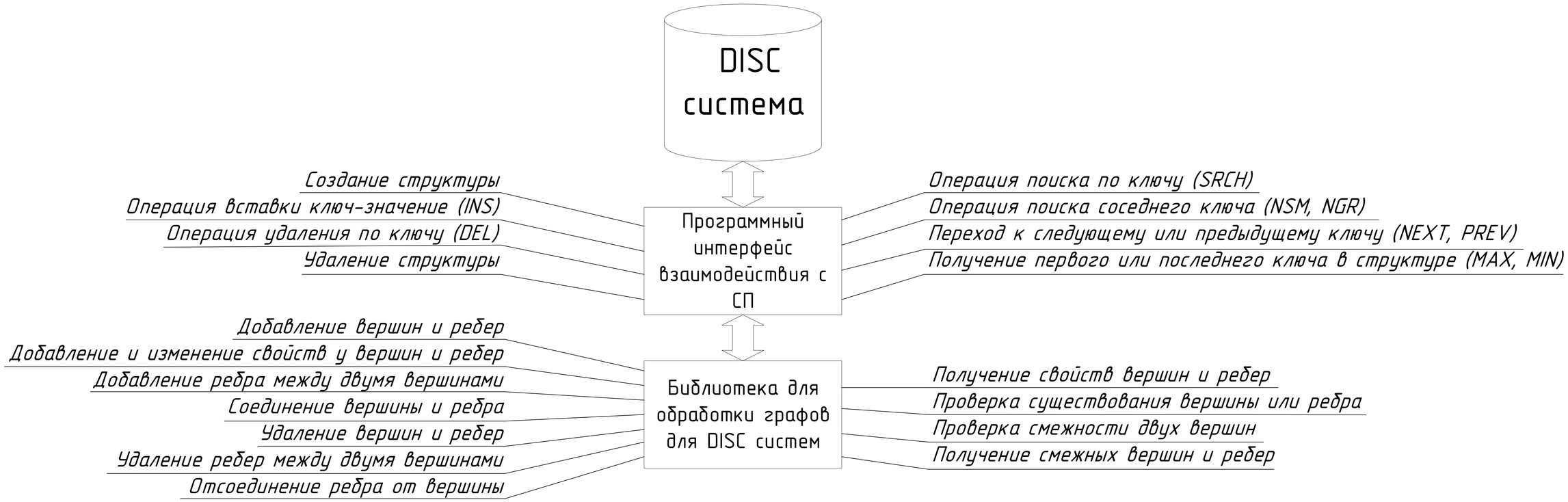
Библиотека элементов программного интерфейса для обработки графов



ЦЕЛИ

Создание библиотеки элементов программного интерфейса для обработки графов для системы с дискретным набором команд.

ЗАДАЧИ

- Анализ и исследование программной модели процессора с набором команд дискретной математики
- Анализ программного интерфейса процессора обработки структур
- Анализ технологий хранения и обработки графов
- Исследование принципов обработки графовых моделей систем с дискретным набором команд
- Разработка графовых моделей хранения графов для таких систем
- Разработка программной части компонентов системы
- Разработка симулятора для программного интерфейса процессора обработки структур
- Разработка библиотеки для обработки ультраграфов для систем с дискретным набором команд
- Обеспечение качества и надёжности

Выпускная квалификационная работа магистра				Библиотека элементов программного интерфейса для обработки графов			
Изм. Лист	№ докум.	Подпись	Дата	Цели и задачи	Лит	Масса	Масштаб
Разраб.	Кирьяненко А.В.		25.05.2025				
Провер.	Попов А.Ю.		25.05.2025		Лист 1	Листов 10	
Т.контроль	Еремин О.Ю.						
Реценз.					МГТУ им. Н.Э. Баумана группа ИУ6-41М		
Н.контроль							
Утв.							

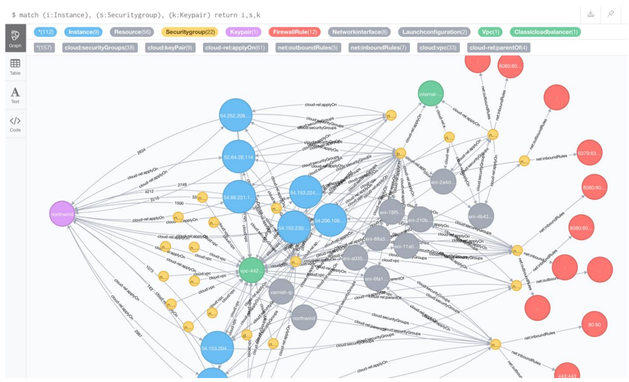
Анализ технологий хранения и обработки графов

Boost – собрание библиотек классов, использующих функциональность языка C++ и предоставляющих удобный кроссплатформенный высокоуровневый интерфейс для решения различных задач программирования (работа с данными, алгоритмами, файлами, потоками и т. п.).
Свободно распространяются по лицензии Boost Software License вместе с исходным кодом.

Boost Graph Library предоставляет гибкую и эффективную реализацию концепций графов.

Можно выбрать представление графа:
- список смежности (adjacency_list);
- матрица смежности (adjacency_matrix).

В библиотеке имеется большая база алгоритмов, среди которых:
- поиск в ширину;
- поиск в глубину;
- алгоритм Беллмана-Форда;
- алгоритм Дейкстры;
- алгоритм Прима;
- алгоритм Краскала;
- нахождение компонент связности графа;
- задача о максимальном потоке;
- обратный алгоритм Катхилла-Макки;
- алгоритм топологической сортировки.



Neo4j – графовая система управления базами данных с открытым исходным кодом, реализованная на Java. По состоянию на 2019 год считается самой распространённой графовой СУБД.

Язык запросов – Cypher

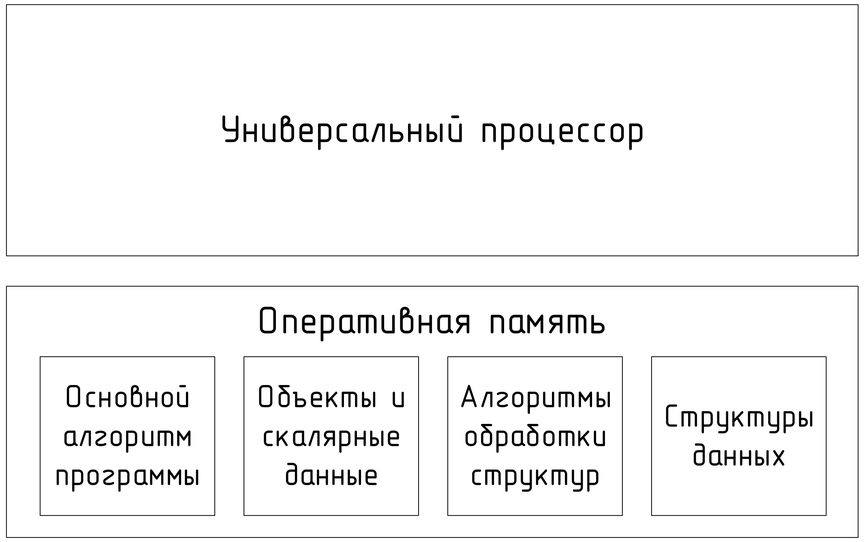
Интерфейс программирования приложений для СУБД реализован для многих языков программирования, включая Java, Python, Clojure, Ruby, PHP, также реализовано API в стиле REST.

LEDA – это библиотека классов C++ от AlgoSol для эффективной обработки типов данных и алгоритмов. LEDA предоставляет алгоритмическую базу в области графовых и сетевых задач, геометрических вычислений, комбинаторной оптимизации и других.

Также LEDA включает интерфейс для ввода и вывода графов для различных платформ, что позволяет делать визуализацию графов, и анимацию работы графовых алгоритмов.

Выпускная квалификационная работа магистра					Библиотека элементов программного интерфейса для обработки графов		
Изм.	Лист	№ докум.	Подпись	Дата	Анализ технологий хранения и обработки графов	Лит	Масса / Масштаб
Разраб.		Кирьяненко А.В.		25.05.2020			
Провер.		Попов А.Ю.		25.05.2020		Лист 2	Листов 10
Т.контроль		Еремин О.Ю.				МГТУ им. Н.Э. Баумана группа ИУ6-41М	
Реценз.							
Н.контроль							
Утв.							

Универсальный микропроцессор



- *Обрабатывает числа.*

- *Аппаратно реализует арифметическую и логическую обработку.*

- *Обработка сложных моделей данных (деревьев. графов) выполняется на конвейере микропроцессора последовательно.*

- *Использует ОЗУ для хранения программ, данных структур данных, множеств и т.д.*

- *Распределение памяти для хранения структур данных осуществляется программно.*

Микропроцессор Leonhard

- *Обрабатывает множества.*

- *Аппаратно реализует математический аппарат дискретной математики.*

- *Микроархитектура Leonhard обеспечивает параллельную обработку множеств, структур данных, графов.*

- *Использует независимую память для хранения структур данных, множеств, графов.*

- *Распределение памяти осуществляется аппаратными механизмами микропроцессора Leonhard.*

Выпускная квалификационная работа магистра				Библиотека элементов программного интерфейса для обработки графов		
Изм.	Лист	№ докум.	Подпись	Дата	Схема структурная системы с аппаратной поддержкой операций дискретной матиматики	Лит.	Масса	Масштаб
Разраб.		Кирьяненко А.В.		25.05.2022				
Провер.		Попов А.Ю.		25.05.2022		Лист 3	Листов 10	
Т.контроль		Еремин О.Ю.						
Реценз.						МГТУ им. Н.Э. Баумана группа ИУ6-41М		
Н.контроль								
Утв.								

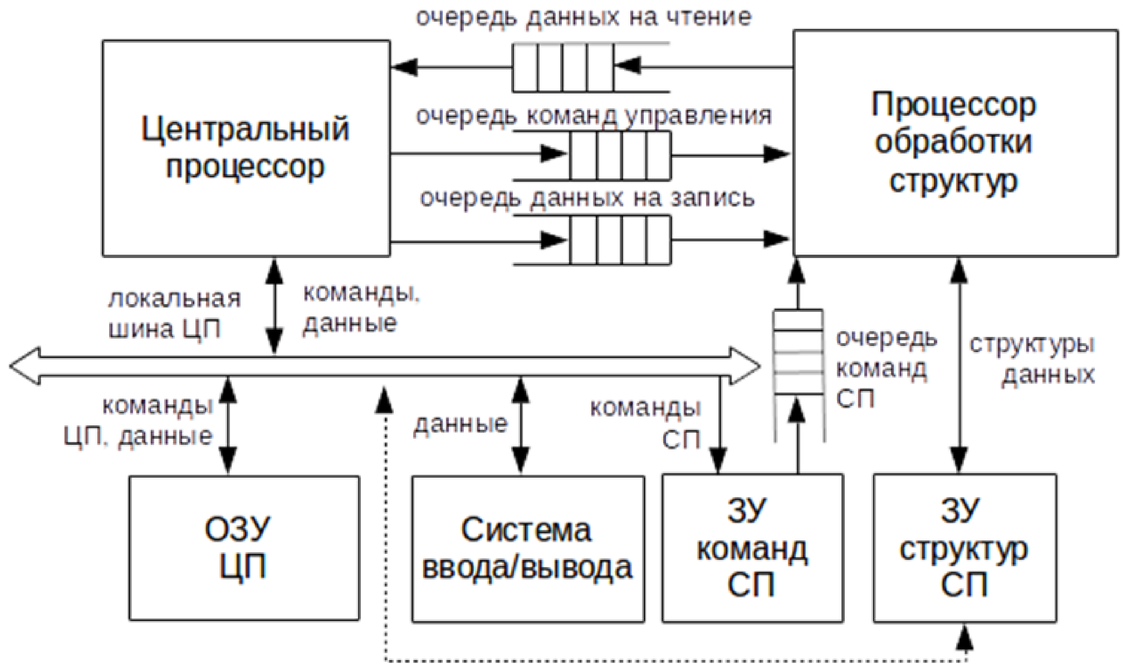
Программная модель процессора с набором команд дискретной математики

Формат данных Leonhard x64

Структура	Ключ	Значение
3 бита	64 бита	64 бита

Команды дискретной математики:

- **Search (SRCH)** - поиск значения по ключу.
- **Insert (INS)** - вставляет пару ключ-значение в структуру. SPU обновляет значение, если указанный ключ уже находится в структуре.
- **Delete (DEL)** - удаляет указанный ключ из структуры данных.
- **Neighbors (NSM, NGR)** - поиск соседнего ключа, который меньше (или больше) заданного и возвращает его значение.
- **Maximum /minimum (MAX, MIN)** - поиск первого или последнего ключа в структуре данных.
- **Cardinality (CNT)** - определяет количество ключей, хранящихся в структуре.
- **AND, OR, NOT** - выполняют объединения, пересечения и дополнения в двух структурах данных.
- **Срезы (LS, GR, LSEQ, GREQ)** - извлекают подмножество одной структуры данных в другую.
- **Переход к следующему или предыдущему (NEXT, PREV)** - находят соседний (следующий или предыдущий) ключ в структуре данных относительно переданного ключа (исходный ключ должен обязательно присутствовать в структуре).
- **Удаление структуры (DELS)** - очищает все ресурсы, используемые заданной структурой.
- **Squeeze (SQ)** - дефрагментирует блоки памяти DSM, используемые структурой.
- **Jump (JT)** - указывает SPU код ветвления, который должен быть синхронизирован с CPU.



Параметры Leonhard x64

Параметр	Значение
Максимальный размер команды в локальной памяти команд СП (LCM)	144 бит
Максимальный размер команды из ЦП	160 бит
Максимальный размер результата из СП в ЦП	64 бит
Количество разрядов поля ключа	64 бит
Количество разрядов поля значения	64 бит
Расположение байт в памяти	Младший байт по младшему адресу
Размер внешней памяти структур	4 ГБайта
Максимальное количество ключей в структуре	100 663 296
Кратность вершины B+ дерева	8
Количество ключей на нижнем уровне дерева	6
Максимальное количество хранимых структур	7

Выпускная квалификационная работа магистра					Библиотека элементов программного интерфейса для обработки графов	
Изм.	Лист	№ докум.	Подпись	Дата	Программная модель процессора с набором команд дискретной математики	Лит. / Масса / Масштаб
Разраб.		Кирьяненко А.В.		25.05.2021		
Провер.		Попов А.Ю.		25.05.2021		Лист 4 / Листов 10
Т.контроль		Еремин О.Ю.				
Реценз.						МГТУ им. Н.Э. Баумана группа ИУ6-41М
Н.контроль						
Утв.						

Модели хранения графов для систем с дискретным набором команд

Модель для ориентированного остовного графа

Ключ		**Значение**
id вершины	0..0	Данные вершины
id вершины	id смежной вершины	

Модель для взвешенного ориентированного остовного графа

Ключ			**Значение**
id вершины	0..0		Данные вершины
id вершины	0..0	id смежной вершины	Вес ребра
id вершины	Вес	id смежной вершины	

Модель для ориентированного мультиграфа

Ключ			**Значение**
0..0	id ребра	id смежной вершины 1	id смежной вершины 2
id вершины	0..0		Данные вершины
id вершины	id ребра	id смежной вершины	

Модель для взвешенного ориентированного мультиграфа

Ключ				**Значение**
0..0	id ребра	id смежной вершины 1	id смежной вершины 2	
id вершин	0..0			Данные вершины
id вершины	Вес	id ребра	id смежной вершины	

Модель для гиперграфа

Ключ		**Значение**
id вершины	0..0	Данные вершины
id вершины	id ребра	

Ключ		**Значение**
id ребра	0..0	Вес ребра
id ребра	id вершины	

Модель для ультраграфа

Ключ			**Значение**
0	id вершины	0..0	Данные вершины
Бит инцидентности	id вершины	id ребра	

Ключ			**Значение**
0	id ребра	0..0	Вес ребра
Бит инцидентности	id ребра	id вершины	

Модель для взвешенного ультраграфа

Ключ			**Значение**
0	0..0		Общее кол-во вершин
0	id вершины	0..0	Данные вершины
0	id вершины	1..1	Кол-во входящих ребер
Бит инцидентности	id вершины	Вес ребра	id ребра
1	id вершины	1..1	Кол-во исходящих ребер

Ключ			**Значение**
0	0..0		Общее кол-во ребер
0	id ребра	0..0	Вес ребра
0	id ребра	1..1	Кол-во вершин, в кот. входит ребро
Бит инцидентности	id ребра	id вершины	
1	id ребра	1..1	Кол-во вершин, из кот. выходит ребро

Модель ультраграфа с атрибутами для ребер

Ключ					**Значение**
0	0..0				Общее кол-во вершин
0	id вершины	0..0			Данные вершины
0	id вершины	1..1			Кол-во входящих ребер
Бит инцидентности	id вершины	id атрибута	Значение атрибута	id ребра	
1	id вершины	1..1			Кол-во исходящих ребер

Ключ				**Значение**
0	0..0			Общее кол-во ребер
0	id ребра	0..0	id атрибута	Значение атрибута
0	id ребра	1..1		Кол-во вершин, в кот. входит ребро
Бит инцидентности	id ребра	id вершины	0..0	
1	id ребра	1..1		Кол-во вершин, из кот. выходит ребро

Реализованная модель ультраграфа

Ключ				**Значение**
id графа	**Бит инцидентности**	**id вершины**	**id ребра**	
id графа	0	0 ... 0		Общее кол-во вершин
id графа	0	id вершины	0 ... 0	Данные вершины
id графа	0	id вершины	1 ... 1	Кол-во исходящих ребер
id графа	1	id вершины	1 ... 1	Кол-во входящих ребер

Ключ				**Значение**
id графа	**Бит инцидентности**	**id ребра**	**id вершины**	
id графа	0	0 ... 0		Общее кол-во ребер
id графа	0	id ребра	0 ... 0	Данные ребра
id графа	0	id ребра	1 ... 1	Кол-во вершин, из кот. выходит ребро
id графа	1	id ребра	1 ... 1	Кол-во вершин, в кот. входит ребро

Выпускная квалификационная работа магистра					Библиотека элементов программного интерфейса для обработки графов	
Изм.	Лист	№ докум.	Подпись	Дата	Модели хранения графов для систем с дискретным набором команд	Лит. / Масса / Масштаб
Разраб.		Кирьяненко А.В.		25.05.2023		
Провер.		Попов А.Ю.		25.05.2023		Лист 5 / Листов 10
Т.контроль		Еремин О.Ю.				
Реценз.						МГТУ им. Н.Э. Баумана группа ИУ6-41М
Н.контроль						
Утв.						

Диаграмма классов программного интерфейса процессора обработки структур

<<Интерфейс>>
AbstractStructure

```
# gsid_t gsid;
+ u32 get_power() = 0;
+ virtual status_t insert(key_t key, value_t value, flags_t flags = NO_FLAGS) = 0;
+ virtual status_t del(key_t key, flags_t flags = NO_FLAGS) = 0;
+ virtual pair_t search(key_t key, flags_t flags = P_FLAG) = 0;
+ virtual pair_t min(flags_t flags = P_FLAG) = 0;
+ virtual pair_t max(flags_t flags = P_FLAG) = 0;
+ virtual pair_t next(key_t key, flags_t flags = P_FLAG) = 0;
+ virtual pair_t prev(key_t key, flags_t flags = P_FLAG) = 0;
+ virtual pair_t nsm(key_t key, flags_t flags = P_FLAG) = 0;
+ virtual pair_t ngr(key_t key, flags_t flags = P_FLAG) = 0;
# adds_rslt_t adds();
# dels_rslt_t dels();
```

Structure<KeyNameT, ValueNameT>

```
- AbstractStructure *_base;
+ Structure(KeyLength key_length, bool is_val_constr = false);
+ Structure(ValueLength value_length);
+ Structure(KeyLength key_length, ValueLength value_length);
+ status_t insert(key_t   key, value_t   value, flags_t flags = NO_FLAGS);
+ status_t insert(KeyData key, value_t   value, flags_t flags = NO_FLAGS);
+ status_t insert(key_t   key, ValueData value, flags_t flags = NO_FLAGS);
+ status_t insert(KeyData key, ValueData value, flags_t flags = NO_FLAGS);
+ status_t del(key_t   key, flags_t flags = NO_FLAGS);
+ status_t del(KeyData key, flags_t flags = NO_FLAGS);
+ pair_t search(key_t   key, flags_t flags = P_FLAG);
+ pair_t search(KeyData key, flags_t flags = P_FLAG);
+ pair_t min(flags_t flags = P_FLAG);
+ pair_t max(flags_t flags = P_FLAG);
+ pair_t next(key_t   key, flags_t flags = P_FLAG);
+ pair_t next(KeyData key, flags_t flags = P_FLAG);
+ pair_t prev(key_t   key, flags_t flags = P_FLAG);
+ pair_t prev(KeyData key, flags_t flags = P_FLAG);
+ pair_t nsm(key_t   key, flags_t flags = P_FLAG);
+ pair_t nsm(KeyData key, flags_t flags = P_FLAG);
+ pair_t ngr(key_t   key, flags_t flags = P_FLAG);
+ pair_t ngr(KeyData key, flags_t flags = P_FLAG);
+ u32 get_power();
```

BaseExternValue

```
- value_t _id;
+ BaseExternValue();
+ explicit BaseExternValue(value_t id);
+ value_t get_id();
+ void set_id(value_t id);
+ operator data_t();
+ static void spu_insert(gsid_t gsid, key_t key, BaseExternValue &value);
+ static void spu_remove(gsid_t gsid, key_t key);
# value_t get_next_id();
```

ExternValue<T>

```
+ ExternValue() : BaseExternValue() {}
+ ExternValue(value_t id) : BaseExternValue(id) {}
+ virtual T & get() = 0;
+ virtual void set(T value) = 0;
+ operator T();
+ ExternValue& operator<< (value_t id);
+ ExternValue& operator<< (pair_t pair);
+ ExternValue& operator<< (T value);
+ T& operator>> (T value);
```

HashMapExternValue<T>

```
- static std::unordered_map<value_t, T> _data;
+ HashMapExternValue() : ExternValue<T>() {}
+ HashMapExternValue(value_t id) : ExternValue<T>(id) {}
+ HashMapExternValue(pair_t pair) : ExternValue<T>(pair.value) {}
+ HashMapExternValue(T value) : ExternValue<T>();
+ T & get() override;
+ void set(T value) override;
```

BaseStructure

```
- Fileops fops;
+ u32 get_power();
+ status_t insert(key_t key, value_t value, flags_t flag);
+ status_t del(key_t key, flags_t flags = NO_FLAGS);
+ pair_t search(key_t key, flags_t flags = P_FLAG);
+ pair_t min(flags_t flags = P_FLAG);
+ pair_t max(flags_t flags = P_FLAG);
+ pair_t next(key_t key, flags_t flags = P_FLAG);
+ pair_t prev(key_t key, flags_t flags = P_FLAG);
+ pair_t nsm(key_t key, flags_t flags = P_FLAG);
+ pair_t ngr(key_t key, flags_t flags = P_FLAG);
# adds_rslt_t adds();
# dels_rslt_t dels();
```

Simulator

```
- std::map<key_t, value_t> *_data;
+ u32 get_power();
+ status_t insert(key_t key, value_t value, flags_t flags);
+ status_t del(key_t key, flags_t flags = NO_FLAGS);
+ pair_t search(key_t key, flags_t flags = P_FLAG);
+ pair_t min(flags_t flags = P_FLAG);
+ pair_t max(flags_t flags = P_FLAG);
+ pair_t next(key_t key, flags_t flags = P_FLAG);
+ pair_t prev(key_t key, flags_t flags = P_FLAG);
+ pair_t nsm(key_t key, flags_t flags = P_FLAG);
+ pair_t ngr(key_t key, flags_t flags = P_FLAG);
# adds_rslt_t adds();
# dels_rslt_t dels();
```

FieldsContainer<NameT, ContentT>

```
- ContentVector cont_vec;
- std::string ClName;
+ FieldsContainer(std::string ClassName) : cont_vec({0}), ClName(ClassName) {}
+ FieldsContainer(ContentVector content_vector, std::string ClassName);
+ FieldsContainer(std::initializer_list<ContentStruct> content_initializer_list, std::string ClassName);
+ friend auto begin(FieldsContainer<NameT, ContentT>& container);
+ friend auto end(FieldsContainer<NameT, ContentT>& container);
# inline ContentT& find_data_by_name(NameT name);
# inline const ContentT& find_data_by_name(NameT name) const;
# inline void push(ContentStruct addict);
# inline void clear();
```

Fileops

```
- int descriptor = 0;
+ Fileops(const char* filename);
+ ~Fileops();
+ template<typename CmdFrmt, typename RsltFrmt>
  RsltFrmt execute(CmdFrmt &cmd);
```

<<Интерфейс>>
AbstractFields<NameT>

```
+ virtual ~AbstractFields() {};
+ virtual operator data_t() = 0;
+ virtual AbstractFields& operator= (data_t fields_data) = 0;
+ virtual AbstractFields& operator= (Data fields_data) = 0;
+ virtual data_t& operator[](NameT name) = 0;
```

FieldsLength<NameT>

```
+ FieldsLength() : Parent("FieldsLength") {}
+ FieldsLength(LengthVector length_vector);
+ FieldsLength(std::initializer_list<LengthStruct> length_initializer_list);
+ inline static data_t mask(u8 length);
+ data_t fieldMask(NameT name) const;
+ const u8& operator[](NameT name) const;
+ u8& operator[](NameT name);
```

FieldsData<NameT>

```
+ FieldsData() : Parent("FieldsData") {}
+ FieldsData(DataVector data_vector) : Parent(data_vector, "FieldsData") {}
+ FieldsData(std::initializer_list<DataStruct> data_initializer_list);
+ void push(DataStruct addict);
+ void clear();
+ const data_t& operator[](NameT name) const;
+ data_t& operator[](NameT name);
```

data_container

```
+ u32 cont[SPU_WEIGHT];
+ data_container();
+ template <typename T> data_container(T data);
+ template <typename T> operator T();
+ template <typename T1, T2> set(T1 &to, T2 &from);
+ u32& operator[](u8 idx);
+ const u32& operator[](u8 idx);
```

pair_containter

```
+ key_t key;
+ value_t value;
+ status_t status;
```

FieldsPointer<NameT>

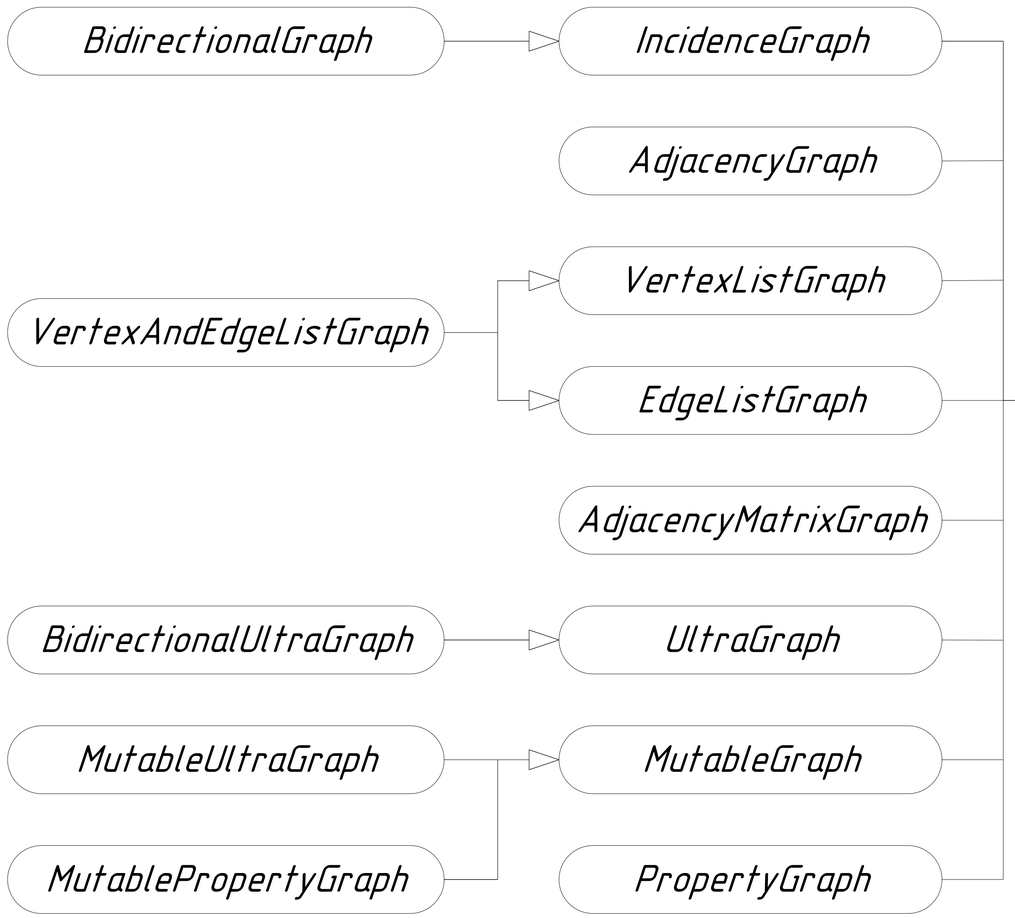
```
- Fields<NameT>* _ptr;
- bool del_ptr = true;
+ FieldsPointer(Fields<NameT>* fields_ptr);
+ FieldsPointer(const FieldsPointer& other);
+ ~FieldsPointer();
+ inline operator data_t() override;
+ FieldsPointer& operator= (data_t fields_data) override;
+ FieldsPointer& operator= (Data fields_data) override;
+ inline data_t& operator[](NameT name) override;
+ friend auto begin(FieldsPointer<NameT>& fields_ptr);
+ friend auto end(FieldsPointer<NameT>&   fields_ptr);
```

Fields<NameT>

```
- Length length;
- Data data;
+ Fields(Length fields_length);
+ operator data_t() override;
+ Fields& operator= (data_t fields_data) override;
+ Fields& operator= (Data fields_data) override;
+ data_t& operator[](NameT name) override;
+ friend auto begin(Fields<NameT>& fields);
+ friend auto end(Fields<NameT>&   fields);
```

Выпускная квалификационная работа магистра					Библиотека элементов программного интерфейса для обработки графов
Изм.	Лист	№ докум.	Подпись	Дата	Диаграмма классов программного интерфейса процессора обработки структур
Разраб.		Кирьяненко А.В.		25.05.2021	Лит. / Масса / Масштаб
Провер.		Попов А.Ю.		25.05.2021	Лист 6 / Листов 10
Т.контроль		Еремин О.Ю.			МГТУ им. Н.Э. Баумана группа ИУ6-41М
Реценз.					
Н.контроль					
Утв.					

Концепции представления графов



Концепция графа списка ребер (EdgeListGraph)

Выражение	Описание
`boost::graph_traits<G>::edge_iterator`	Итератор по всем ребрам.
`boost::graph_traits<G>::edges_size_type`	Целочисленный тип для обозначения количества ребер в графе.
`std::pair<edge_iterator, edge_iterator> edges(g)`	Возвращает диапазон итераторов, обеспечивающий доступ ко всем ребрам.
`edges_size_type num_edges(g)`	Количество ребер в графе *g*.
`vertex_descriptor source(e, g)`	Возвращает источник для ребра *e*.
`vertex_descriptor target(e, g)`	Возвращает сток для ребра *e*.

Концепция графа списка вершин (VertexListGraph)

Выражение	Описание
`boost::graph_traits<G>::vertex_iterator`	Итератор по всем вершинам.
`boost::graph_traits<G>::vertices_size_type`	Целочисленный тип для обозначения количества вершин в графе.
`std::pair<vertex_iterator, vertex_iterator> vertices(g)`	Возвращает диапазон итераторов, обеспечивающий доступ ко всем вершинам в графе *g*.
`vertices_size_type num_vertices(g)`	Количество вершин в графе *g*.

Концепция изменяемого графа (MutableGraph)

Выражение	Описание
`vertex_descriptor add_vertex(g)`	Добавляет вершину в граф *g*.
`void clear_vertex(v, g)`	Удаляет все ребра смежные с вершиной.
`void remove_vertex(v, g)`	Удаляет вершину *v* из графа *g*.
`std::pair<edge_descriptor, bool> add_edge(u, v, g)`	Вставляет ребро в граф *g* между вершинами *u* и *v*. Если граф запрещает параллельные ребра, то флаг устанавливается в значение *false*.
`void remove_edge(u, v, g)`	Удаляет все ребра между вершинами.
`void remove_edge(e, g)`	Удаляет ребро *e* из графа *g*.

Концепция изменяемого графа свойств (MutablePropertyGraph)

Выражение	Описание
`vertex_descriptor add_vertex(vp, g)`	Добавляет вершину со свойствами *vp* в граф *g*.
`std::pair<edge_descriptor, bool> add_edge(u, v, ep, g)`	Добавляет ребро со свойствами ep между вершинами *u* и *v* в граф *g*.

Концепция графа свойств (PropertyGraph)

Выражение	Описание
`boost::property_map<G, Property>::type`	Тип изменяемой карты свойств.
`boost::property_map<G, Property>::const_type`	Тип неизменяемой карты свойств.
`get(property, g)`	Возвращает объект карты свойства для графа *g*.
`get(property, g, x)`	Получить значение свойства для вершины или ребра *x*.
`put(property, g, x, v)`	Установить значение свойства для вершины или ребра *x*.

Концепция абстрактного графа (Graph)

Выражение	Описание
`boost::graph_traits<G>::vertex_descriptor`	Тип дескриптора вершины.
`boost::graph_traits<G>::edge_descriptor`	Тип дескриптора ребра.
`boost::graph_traits<G>::directed_category`	Указывается является ли граф ориентированным.
`boost::graph_traits<G>::edge_parallel_category`	Здесь описывается, допускает ли класс графа вставку параллельных ребер (ребер с одинаковым источником и стоком).
`boost::graph_traits<G>::traversal_category`	Здесь описываются способы посещения вершин и ребер графа.

Концепция матрицы смежности (AdjacencyMatrix)

Выражение	Описание
`std::pair<edge_descriptor, bool> edge(u, v, g)`	Возвращает пару, состоящую из флага, указывающего, существует ли ребро между u и v в графе g, и дескриптора ребра.

Концепция графа смежности (AdjacencyGraph)

Выражение	Описание
`boost::graph_traits<G>::adjacency_iterator`	Итератор по смежным вершинам.
`std::pair<adjacency_iterator, adjacency_iterator> adjacent_vertices(v, g)`	Возвращает диапазон итераторов, обеспечивающий доступ к смежным вершинам для вершины *v* в графе *g*.

Концепция графа инцидентности (IncidenceGraph)

Выражение	Описание
`boost::graph_traits<G>::out_edge_iterator`	Итератор по исходящим ребрам.
`boost::graph_traits<G>::degree_size_type`	Целочисленный тип степени вершины.
`std::pair<out_edge_iterator, out_edge_iterator> out_edges(v, g)`	Возвращает диапазон итераторов, обеспечивающий доступ к исходящим ребрам вершины *u* в графе *g*.
`vertex_descriptor source(e, g)`	Возвращает дескриптор источника для ребра *e*.
`vertex_descriptor target(e, g)`	Возвращает дескриптор стока для ребра *e*.

Концепция двунаправленного графа (BidirectionalGraph)

Выражение	Описание
`boost::graph_traits<G>::in_edge_iterator`	Итератор по входящим ребрам.
`std::pair<in_edge_iterator, in_edge_iterator> in_edges(v, g)`	Возвращает диапазон итераторов, обеспечивающий доступ к входящим ребрам для вершины *v* в графе *g*.
`degree_size_type in_degree(v, g)`	Возвращает количество входящих ребер.
`degree_size_type degree(e, g)`	Возвращает степень вершины.

Концепция ультраграфа (UltraGraph)

Выражение	Описание
`graph_traits<G>::target_iterator`	Итератор по вершинам «стокам».
`std::pair<target_iterator, target_iterator> targets(e, g)`	Возвращает диапазон итераторов, обеспечивающий доступ к стокам для ребра *e* в графе *g*.
`degree_size_type targets_cnt(e, g)`	Возвращает количество вершин «стоков» для ребра *e*.

Концепция двунаправленного ультраграфа (BidirectionalUltraGraph)

Выражение	Описание
`graph_traits<G>::source_iterator`	Итератор по вершинам «источников».
`std::pair<source_iterator, source_iterator> sources(e, g)`	Возвращает диапазон итераторов, обеспечивающий доступ к источникам для ребра *e* в графе *g*.
`degree_size_type sources_cnt(e, g)`	Возвращает количество вершин «источников» для ребра *e*.

Концепция изменяемого ультраграфа (MutableUltraGraph)

Выражение	Описание
`void connect_source(e, v, g)`	Присоединяет как «источник» вершину *v* к ребру *e*.
`void connect_target(e, v, g)`	Присоединяет как «сток» вершину *v* к ребру *e*.
`void disconnect_source(e, v, g)`	Отсоединяет вершину «источник» *v* от ребра *e*.
`void disconnect_target(e, v, g)`	Отсоединяет вершину «сток» *v* от ребра *e*.

Выпускная квалификационная работа магистра					Библиотека элементов программного интерфейса для обработки графов		
Изм.	Лист	№ докум.	Подпись	Дата	Концепции представления графов	Лит.	Масса	Масштаб
Разраб.		Кирьяненко А.В.		25.05.2020				
Провер.		Попов А.Ю.		25.05.2020		Лист 7	Листов 10	
Т.контроль		Еремин О.Ю.						
Реценз.						МГТУ им. Н.Э. Баумана группа ИУ6-41М		
Н.контроль								
Утв.								

Диаграмма классов библиотеки элементов программного интерфейса для обработки графов

SpuUltraGraph

- id_t _graph_id = 0;
- static id_t global_id_sequence;
- id_t _global_id;
- SpuUltraGraphTraits _graph_traits;
- FieldsLength<SPU_STRUCTURE_ATTRS> _edge_id_fields_len;
- FieldsLength<SPU_STRUCTURE_ATTRS> _vertex_fields_len;
- FieldsLength<SPU_STRUCTURE_ATTRS> _edge_fields_len;
- GraphStructure _vertex_struct;
- GraphStructure _edge_struct

+ explicit SpuUltraGraph(id_t graph_id, const SpuUltraGraphTraits& spu_graph_traits);
+ id_t get_id() const;
+ id_t get_global_id() const;
+ vertex_descriptor add_vertex();
+ vertex_descriptor add_vertex(vertex_descriptor id, value_t value);
+ void put_vertex(vertex_descriptor id, value_t value);
+ void remove_vertex(vertex_descriptor v);
+ bool has_vertex(vertex_descriptor id) const;
+ vertices_size_type num_vertices() const;
+ vertices_size_type degree(vertex_descriptor v) const;
+ vertices_size_type out_degree(vertex_descriptor v) const;
+ vertices_size_type in_degree(vertex_descriptor v) const;
+ void clear_vertex(vertex_descriptor v, bool remove_edges=true);
+ void disconnect_source(edge_descriptor e, vertex_descriptor v);
+ void disconnect_target(edge_descriptor e, vertex_descriptor v);
+ edge_descriptor add_edge();
+ edge_descriptor add_edge(edge_descriptor id, value_t value);
+ edge_descriptor add_edge(vertex_descriptor from, vertex_descriptor to);
+ edge_descriptor add_edge(edge_descriptor id, vertex_descriptor from, vertex_descriptor to, value_t value);
+ void put_edge(edge_descriptor id, value_t value);
+ void connect_target(edge_descriptor edge, vertex_descriptor vertex);
+ void connect_source(edge_descriptor edge, vertex_descriptor vertex);
+ bool has_edge(edge_descriptor id) const;
+ edge_descriptor get_free_edge_descriptor(weight_t weight) const;
+ edge_descriptor get_edge_descriptor(id_t edge_id, weight_t weight) const;
+ weight_t get_weight(edge_descriptor edge) const;
+ edges_size_type num_edges() const;
+ edges_size_type source_cnt(edge_descriptor e);
+ edges_size_type target_cnt(edge_descriptor e);
+ vertex_descriptor source(edge_descriptor e) const;
+ vertex_descriptor target(edge_descriptor e) const;
+ void remove_edge(edge_descriptor edge);
+ void remove_edge(vertex_descriptor from, vertex_descriptor to);
+ value_t get_vertex_value(vertex_descriptor v) const;
+ value_t get_edge_value(edge_descriptor e) const;
+ edge_descriptor edge(vertex_descriptor from, vertex_descriptor to) const;
+ Vertices vertices() const;
+ AdjacentVertices adjacent_vertices(vertex_descriptor v) const;
+ Edges edges() const;
+ ParallelEdges parallel_edges(vertex_descriptor from, vertex_descriptor to) const;
+ OutEdges out_edges(vertex_descriptor v) const;
+ InEdges in_edges(vertex_descriptor v) const;

spu_ug_readable_property_map<PROPERTY, VALUE_T>

- const SpuUltraGraph *_g;

+ VALUE_T get(SPU_GRAPH::id_t id);

spu_ug_property_map <PROPERTY, VALUE_T>

- SpuUltraGraph *_g;

+ VALUE_T get(SPU_GRAPH::id_t id);
+ void put(SPU_GRAPH::id_t id, VALUE_T value);

GraphStructure

- std::shared_ptr<AbstractStructure> _struct;

+ GraphStructure() : _struct(new Structure<>) {}
+ GraphStructure(std::shared_ptr<AbstractStructure> structure);
+ void set(std::shared_ptr<Structure<>> structure);
+ GraphStructure& operator=(std::shared_ptr<AbstractStructure> structure);
+ status_t insert(key_t key, value_t value, flags_t flags);
+ status_t del(key_t key, flags_t flags = NO_FLAGS);
+ pair_t search(key_t key, flags_t flags = P_FLAG);
+ pair_t min(flags_t flags = P_FLAG);
+ pair_t max(flags_t flags = P_FLAG);
+ pair_t next(key_t key, flags_t flags = P_FLAG);
+ pair_t prev(key_t key, flags_t flags = P_FLAG);
+ pair_t nsm(key_t key, flags_t flags = P_FLAG);
+ pair_t ngr(key_t key, flags_t flags = P_FLAG);
- static inline pair_t check_spu_resp(pair_t resp);
- static inline status_t check_spu_resp_status(status_t resp);

AdjacentEdgesIterator<int I>

- const SpuUltraGraph *_g;
- vertex_descriptor _v;
- edge_descriptor _edge;

+ AdjacentEdgesIterator() : _g(nullptr), _v(0), _edge(0) {}
+ AdjacentEdgesIterator(const SpuUltraGraph *g, vertex_descriptor v, edge_descriptor edge);
+ edge_descriptor dereference() const;
+ bool equal(const AdjacentEdgesIterator& other) const;
+ void increment();
+ void decrement();

AdjacentEdges<int I>

- const GraphStructure *_s;
- vertex_descriptor _v;

+ AdjacentEdges(const SpuUltraGraph *g) : _g(g) {}
+ AdjacentEdgesIterator<I> begin();
+ AdjacentEdgesIterator<I> end();
+ AdjacentEdgesIterator<I> rbegin();
+ AdjacentEdgesIterator<I> rend();

EdgeVerticesIterator<int I>

- const SpuUltraGraph *_g;
- edge_descriptor _e;
- vertex_descriptor _v;

+ EdgeVerticesIterator(const SpuUltraGraph *g=nullptr, edge_descriptor e=0, vertex_descriptor v=0)
+ edge_descriptor dereference() const;
+ bool equal(const EdgeVerticesIterator<I>& other) const;
+ void increment();
+ void decrement();

EdgeVertices<int I>

- const SpuUltraGraph *_g;
- edge_descriptor _e;

+ EdgeVertices(const SpuUltraGraph *g, SpuUltraGraph::edge_descriptor e);
+ EdgeVerticesIterator<I> begin();
+ EdgeVerticesIterator<I> end();
+ EdgeVerticesIterator<I> rbegin();
+ EdgeVerticesIterator<I> rend();

StructureIterator

- const GraphStructure *_s;
- SPU::pair_t _pair;
- SPU::key_t _end;

+ StructureIterator(const GraphStructure *structure, SPU::key_t key, SPU::key_t end=0, SPU::status_t status=INIT_STATUS);
+ SPU::pair_t dereference() const;
+ bool equal(const StructureIterator &other) const;
+ void increment();
+ void decrement();

StructureRange

- const GraphStructure *_s;
- SPU::key_t _start, _end;

+ StructureRange(const GraphStructure *structure, key_t start, key_t end);
+ StructureIterator begin();
+ StructureIterator end()

VertexIterator

- const GraphStructure *_s;
- vertex_descriptor _v;

+ VertexIterator() : _g(nullptr) {}
+ VertexIterator(const SpuUltraGraph *g, vertex_descriptor v=0);
+ vertex_descriptor dereference() const;
+ bool equal(const VertexIterator& other) const;
+ void increment();
+ void decrement();

Vertices

- const GraphStructure *_s;

+ Vertices(const SpuUltraGraph *g) : _g(g) {}
+ VertexIterator begin();
+ VertexIterator end()

EdgeIterator

- const GraphStructure *_s;
- edge_descriptor _edge;

+ EdgeIterator() : _g(nullptr) {}
+ EdgeIterator(const SpuUltraGraph *g, edge_descriptor edge=0);
+ edge_descriptor dereference() const;
+ bool equal(const EdgeIterator& other) const
+ void increment();
+ void decrement();

Edges

- const GraphStructure *_s;

+ Vertices(const SpuUltraGraph *g) : _g(g) {}
+ EdgeIterator begin();
+ EdgeIterator end()

ParallelEdgesIterator

- const SpuUltraGraph *_graph;
- vertex_descriptor _from, _to;
- id_t _edge;

+ ParallelEdgesIterator(const SpuUltraGraph *g, vertex_descriptor from, vertex_descriptor to, id_t edge=0);
+ edge_descriptor dereference() const;
+ bool equal(const ParallelEdgesIterator& other) const;
+ void increment();
+ void decrement();

ParallelEdges

- const GraphStructure *_s;
- vertex_descriptor _from, _to;

+ ParallelEdges(const SpuUltraGraph *g, SpuUltraGraph::vertex_descriptor from, SpuUltraGraph::vertex_descriptor v)
+ EdgeIterator begin();
+ EdgeIterator end()

AdjacentVerticesIterator

- const GraphStructure *_s;
- edge_descriptor _edge;

+ AdjacentVerticesIterator(const SpuUltraGraph *g=nullptr, vertex_descriptor v=0, edge_descriptor e=0, vertex_descriptor u=0);
+ vertex_descriptor dereference() const;
+ bool equal(const AdjacentVerticesIterator& other) const;
+ void increment();
+ void decrement();

AdjacentVertices

- const GraphStructure *_s;
- vertex_descriptor _v;

+ AdjacentVertices(const SpuUltraGraph *g, vertex_descriptor v) : _g(g), _v(v) {}
+ AdjacentVerticesIterator begin();
+ AdjacentVerticesIterator end();
+ AdjacentVerticesIterator rbegin();
+ AdjacentVerticesIterator rend();

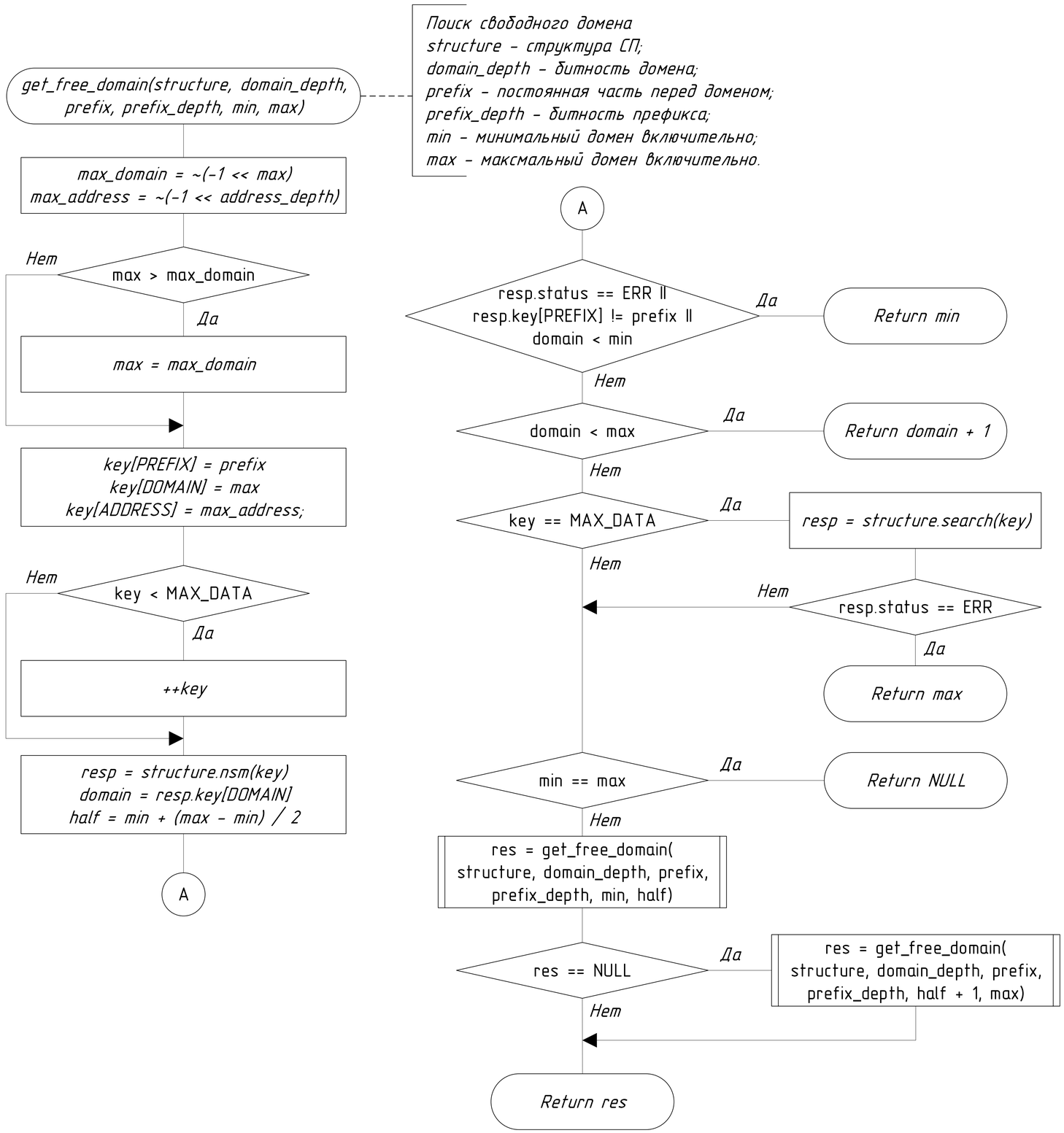
GraphPerformanceTest<G>

+ void (*test_func)(G&) = nullptr;
+ string results_file = "results.csv";
+ size_t avg_iterations_cnt = 10;
+ bool should_fill = true;
+ bool is_mutable_test = true;
+ pair<edge_t, bool> (*add_edge_func)(vertex_t, vertex_t, G&);
+ size_t start_vertices_cnt = 500;
+ size_t inc_vertices_value = 500;
+ size_t end_vertices_cnt = 100000;
+ size_t edges_per_vertex = 3;

+ GraphPerformanceTest(void (*test_func)(G&), string results_file);
+ void start();

Выпускная квалификационная работа магистра					Библиотека элементов программного интерфейса для обработки графов		
					Диаграмма классов библиотеки элементов программного интерфейса для обработки графов	Лит	Масса / Масштаб
Изм. / Лист	№ докум.	Подпись	Дата				
Разраб.	Кирьяненко А.В.		25.05.2025				
Провер.	Попов А.Ю.		25.05.2025				
Т.контроль	Еремин О.Ю.					Лист 8	Листов 10
Реценз.							
Н.контроль						МГТУ им. Н.Э. Баумана группа ИУ6-41М	
Утв.							

Алгоритм поиска свободного домена



Организация хранения данных в СП

Ключ (64 бит)			Значение (64 бит)
Префикс	Домен	Адрес	

Асимптотическая сложность:

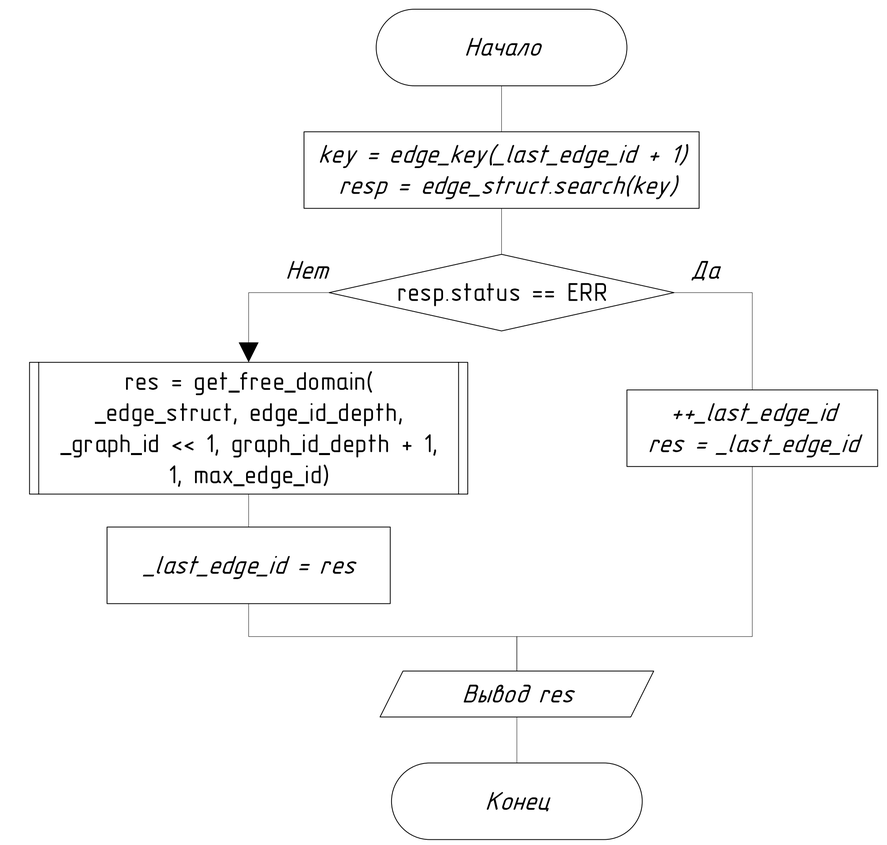
$$O(\log_2(2^{D_{domain}}) * \log_8(n)) = O(D_{domain} * \log_8(n))$$

D_{domain} – битность домена

n – количество записей в структуре СП

$\log_2(n)$ – асимптотическая сложность команд поиска СП

Алгоритм поиска свободного ID ребра



Выпускная квалификационная работа магистра					Библиотека элементов программного интерфейса для обработки графов			
Изм.	Лист	№ докум.	Подпись	Дата	Схемы алгоритмов	Лит.	Масса	Масштаб
Разраб.		Кирьяненко А.В.		25.05.2022				
Провер.		Попов А.Ю.		25.05.2022		Лист 9	Листов 10	
Т.контроль		Еремин О.Ю.						
Реценз.					МГТУ им. Н.Э. Баумана группа ИУ6-41М			
Н.контроль								
Утв.								

Тестирование библиотеки элементов программного интерфейса для обработки графов

Параметры тестирования производительности

Параметр	Значение
Вид графа	Граф решетки
Начальное количество вершин	1000
Инкремент количества вершин	1000
Конечное количество вершин	100000
Начальное количество ребер	2000
Инкремент количества ребер	2000
Конечное количество ребер	200000
Количество тестов производительности	99
Количество повторных тестов для подсчета среднего	10
Общее количество тестов	990

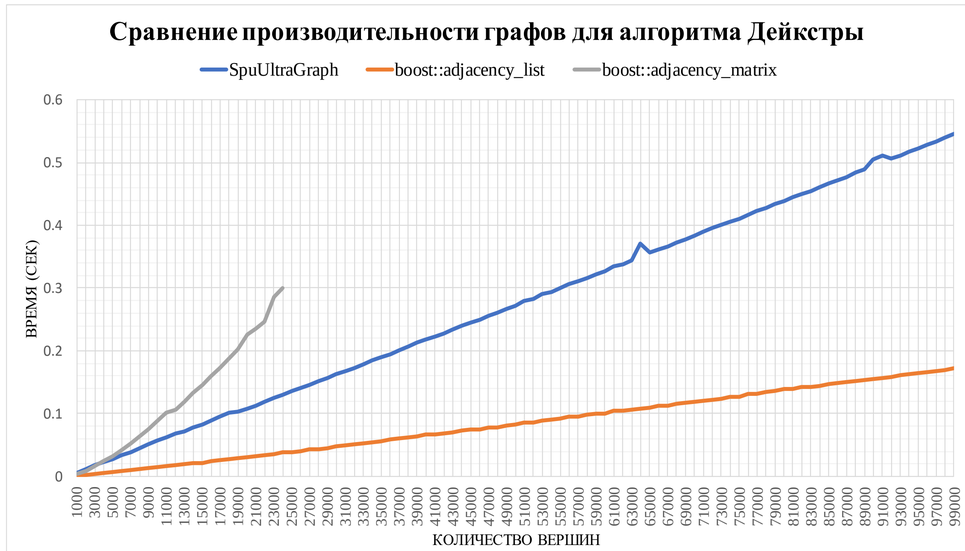
Для обеспечения качества и надежности в общем случае было написано 42 юнит-теста. Также для тестирования графовых концепций использовались тесты, предоставляемые библиотекой Boost Graph Library.

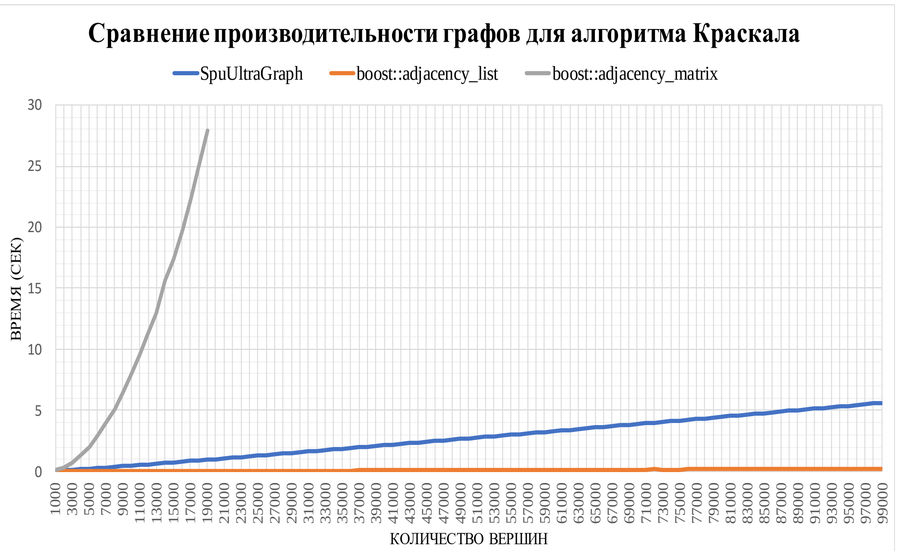
Была проведена оценка покрытия тестами, в ходе которой было выяснено, что тесты покрывают 82% исходного кода.

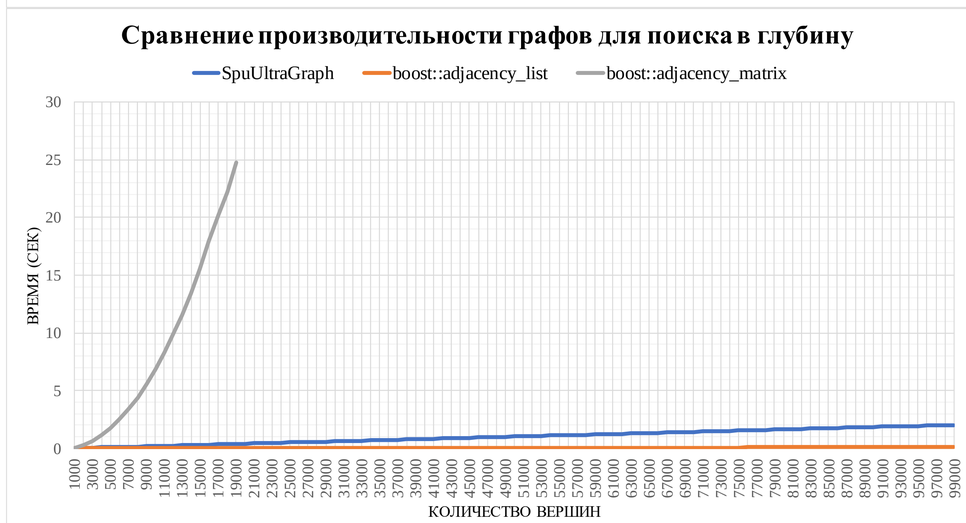
В результате проведенного тестирования было выяснено, что разработанная библиотека для обработки графов соответствует всем предъявляемым требованиям.

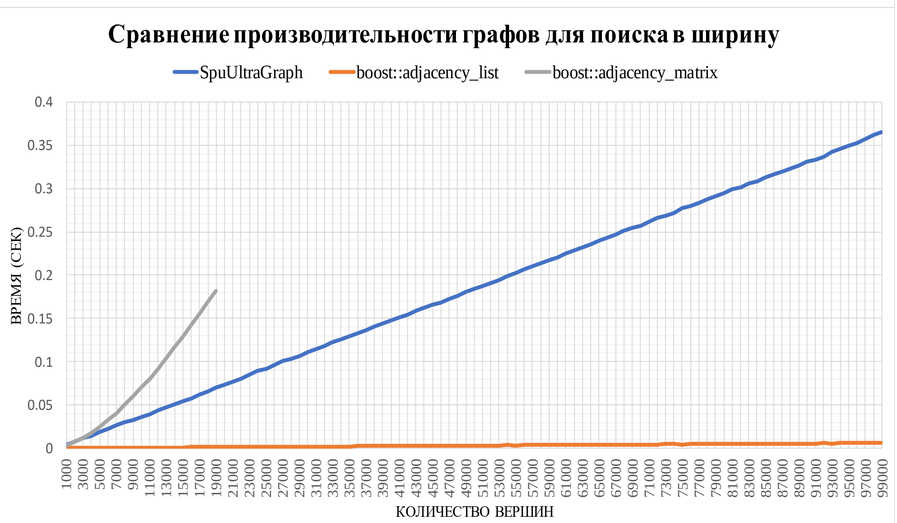
Технические характеристики ЭВМ

Параметр	Значение
Процессор	Intel Core i5-6600
Тактовая частота процессора	3,3 ГГц
Объем ОЗУ	8 Гб
Операционная система	Ubuntu 18.04

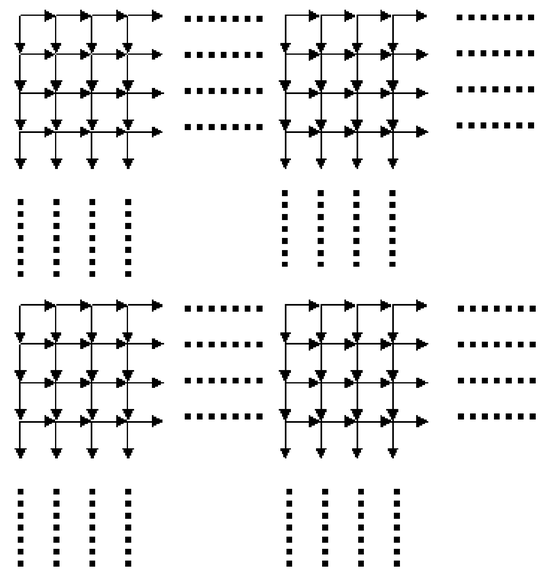








При тестировании производительности строился двумерный граф решетки



Выпускная квалификационная работа магистра				Библиотека элементов программного интерфейса для обработки графов	
Изм.	Лист	№ докум.	Подпись	Дата	
Разраб.		Кирьяненко А.В.		25.05.2020	Тестирование библиотеки элементов программного интерфейса для обработки графов
Провер.		Попов А.Ю.		25.05.2020	
Т.контроль		Еремин О.Ю.			Лит. / Масса / Масштаб
Реценз.					Лист 10 / Листов 10
Н.контроль					МГТУ им. Н.Э. Баумана группа ИУ6-41М
Утв.					