Министерство Образования и Науки РФ

Новосибирский Государственный Технический Университет

Кафедра Вычислительной математики

Курсовой проект

по дисциплине «Программирование»

на тему «База данных произвольных типов в двоичном файле»

Факультет: АВТ

Группа: АП-619

Студент: Тараканов ский А.Н.

Преподаватель: Романов Е.Л.

Новосибирск

2008 г.

Содержание

Задание……3

Структурное описание разработки……4

Функциональное описание……8

Описание работы программы……16

Приложение……23

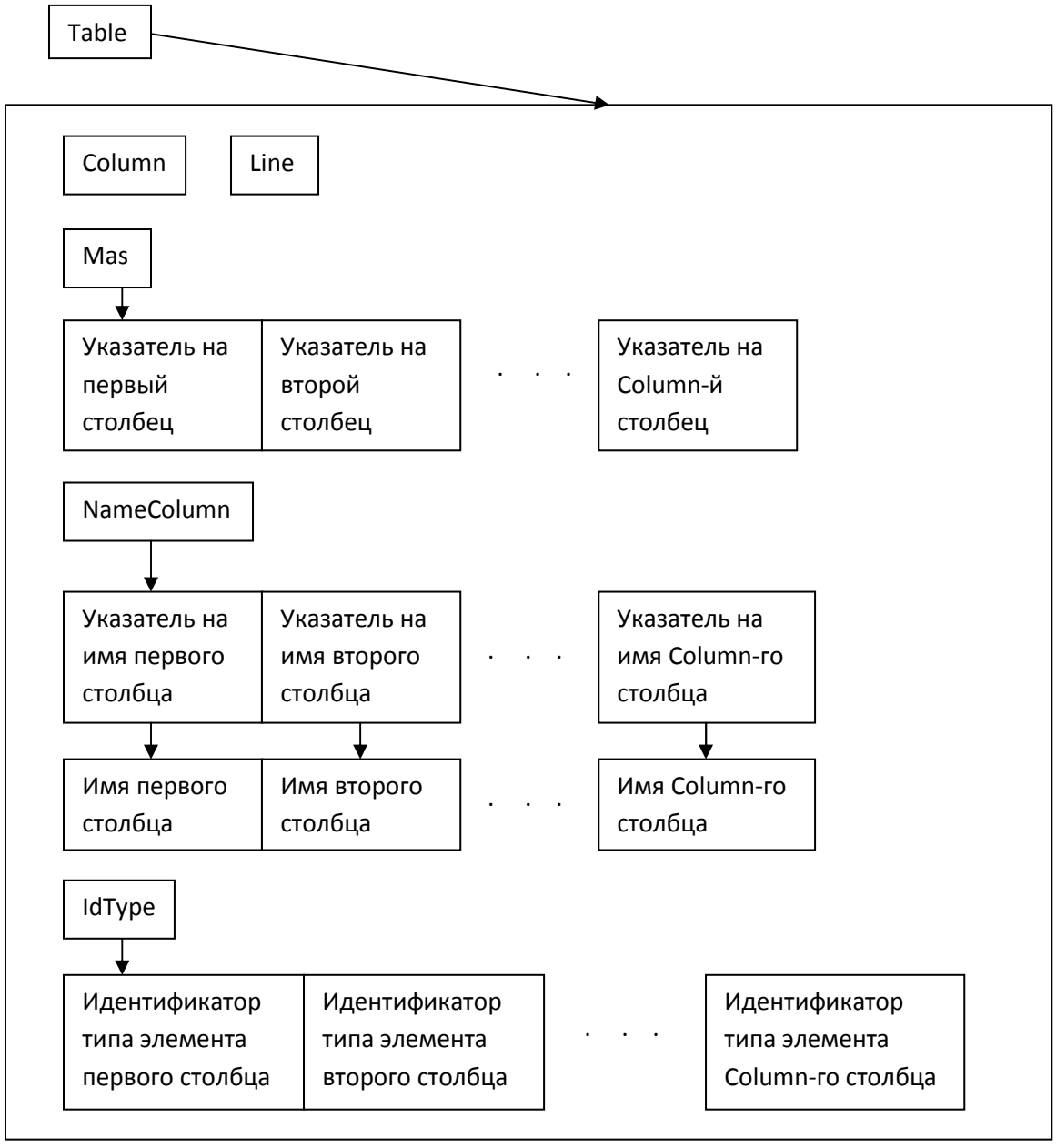
Список литературы……68

Задание

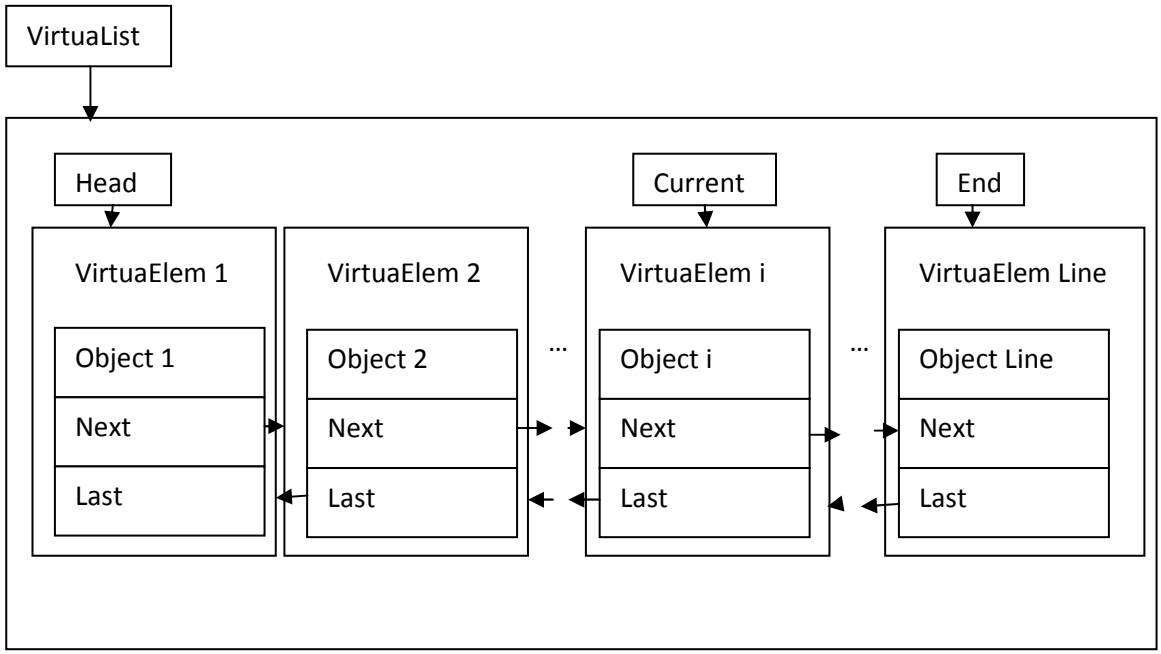
Необходимо разработать интерфейс для объединения в структуру данных множества объектов различных классов - абстрактный базовый класс объектов Object, для которого предусмотреть виртуальные методы: загрузки объекта из текстовой строки, выгрузки объекта в текстовую строку в динамической памяти, добавления объекта в последовательный двоичный файл, чтения объекта из последовательного двоичного файла, возврата уникального идентификатора класса, возврата указателя на строку с именем класса, сравнения двух объектов, “сложения “(объединения) двух объектов, создание динамической копии объекта. Сделать классы хранимых объектов производными от абстрактного базового класса Object. Предусмотреть создание заголовка таблицы со столбцами объектов выбранных типов, добавление, удаление, редактирование строк, сортировку по любому столбцу. Структура данных в файле: описание заголовка таблицы (массив дескрипторов столбцов), размерность и смещение массива указателей на записи.

Структурное описание разработки

Данная программа представляет собой интерфейс для работы с базой данных произвольных типов в двоичном файле. При работе с базой данных вся информация хранится в оперативной памяти компьютера. Также существует возможность сохранить информацию о текущей базе данных на жесткий диск в виде двоичного файла с заданным именем (имя вводится пользователем с клавиатуры), с последующей загрузкой из этого файла обратно в оперативную память, если понадобится. Информация о базе данных хранится в виде таблицы (Table), представляющей собой набор структур данных: двух целых чисел – количества столбцов (Column) и строк (Line) таблицы, динамического массива указателей на столбцы таблицы (Mas), динамического массива указателей на имена столбцов таблицы (NameColumn), а также динамического массива идентификаторов типов элементов, хранимых в столбцах таблицы (IdType).



Столбцы таблицы представляют собой структуру данных – двусвязный список (VirtuaList). Элементы списка также представлены в виде структур данных (VirtuaElem), которые содержат указатель на следующий элемент списка (Next), предыдущий элемент списка (Last), а также указатель на элемент абстрактного типа данных (Object). Также для увеличения скорости доступа к элементам списка в структуре данных хранятся указатели на начало первый элемент (Head), на последний элемент (End) и на текущий элемент (Current).



Для сохранения базы данных в двоичный файл используется следующий формат хранения данных. Первыми заносятся данные о размере таблицы: количество столбцов таблицы, а затем количество строк. Затем записываются данные о столбцах: идентификаторы типов столбцов, имена столбцов. В последнюю очередь записываются столбцы. Для каждого столбца сохраняется количество элементов столбца, затем сохраняются элементы столбца. Процесс загрузки происходит в том же порядке, одновременно строится таблица.

Для работы с базой данных предусмотрены следующие операции: создание пустой базы данных, создание базы данных определенного размера, добавление столбца, добавление строки, удаление столбца, удаление строки, редактирование любого элемента базы данных, упорядочение элементов любого столбца базы данных,

генерирование случайных значений для элементов базы данных, сохранение базы данных в двоичный файл, загрузка базы данных из двоичного файла.

Для упорядочения элементов столбца по возрастанию используется три вида сортировки: сортировка выбором, быстрая сортировка и сортировка слиянием. Рассмотрим алгоритмы данных сортировок в общем виде. Для того, чтобы объяснить алгоритм для простоты будем проводить сортировку массива элементов A размера N - A[N].

Сортировка выбором (Сложность $O(N^2)$)

1. Найти минимальный элемент массива A[N]
2. Поместить минимальный элемент в конец массива A
3. Уменьшить N на 1
4. Если N<2, то перейти на шаг 5, иначе перейти на шаг 1
5. Поместить элемент с номером N в конец массива A
6. Конец алгоритма

Поиск минимального элемента (Сложность O(N))

1. Установить i=2, X присвоить значение первого элемента A
2. если i<N перейти на шаг 3, иначе перейти на шаг 6
3. Если i-й элемент массива A меньше X, то перейти на шаг 4, иначе перейти на шаг 5
4. X присвоить значение i-го элемента массива A
5. Увеличить i на 1, перейти на шаг 2
6. Минимальный элемент находится в X

Быстрая сортировка (Сложность O(N*log(N)))

1. Присвоить L=1, R=N
2. Присвоить i=L, j=R
3. Если L<R перейти на шаг 3, иначе перейти на шаг 14
4. Присвоить X значение элемента с номером ((L+R)/2 – целая часть)
5. Если j-й элемент больше X, то перейти на шаг 6, иначе перейти на шаг 7
6. Уменьшить j на 1, перейти на шаг 5

7. Если i-й элемент меньше X, то перейти на шаг 8, иначе перейти на шаг 9
8. Увеличить i на 1, перейти на шаг 7
9. Если i < j, то перейти на шаг 10, иначе перейти на шаг 12
10. Поменять местами элементы с номерами i и j
11. Увеличить i на 1, уменьшить j на 1, перейти на шаг 5
12. Выполнить алгоритм с шага 3 для L=L и R=j
13. Выполнить алгоритм с шага 3 для L=j+1 и R=R
14. Конец алгоритма

Сортировка слиянием (Сложность O(N*log(N)))

1. Присвоить L=1, R=N
2. Если R<=L, то перейти на шаг
3. Присвоить m целую часть (L+R)/2
4. Выполнить алгоритм с пункта 2 для L=L и R=m
5. Выполнить алгоритм с пункта 2 для L=m+1 и R=R
6. i присвоить L, j присвоить m+1, d=L
7. Если i<m+1, то перейти на шаг 8, иначе перейти на шаг 13
8. Если j<R+1, то перейти на шаг 9, иначе перейти на шаг 13
9. Если i-й элемент меньше j-го, то перейти на шаг 10, иначе перейти на шаг 11
10. i увеличить на 1, d увеличить на 1, перейти на шаг 7
11. Переместить j-й элемент в позицию d
12. d увеличить на 1, i увеличить на 1, перейти на шаг 7
13. Конец алгоритма

Таблица может хранить данные любого типа, но в данной работе организовано хранение только двух видов элементов: числовая строка (строка цифр начинающаяся с 0 или 1 – знак числа) - NumString и целое число Cel. Для каждого из этих типов данных определены все типичные операции для целых чисел, а именно арифметические операции и операции сравнения.

Функциональное описание.

Класс Table

Реализует таблицу данных хранящихся в динамической памяти с возможностью сохранения данных в двоичный файл и последующей загрузки данных из двоичного файла. Содержит следующий набор функций:

Функция очистки таблицы / `void Clean (void)`

Выполняет уничтожение данных таблицы и выделенную под них память, устанавливает размеры таблицы в ноль и обнуляет указатели на массив столбцов, на имена столбцов, на идентификаторы типа элементов столбцов в значение *NULL*.

Функция добавления строки / `int Addl (int Num=0)`

Выполняет процедуру добавления пустой строки в таблицу по номеру *Num*, если номер не задан, то по умолчанию *Num*=0. Если *Num*=0, то выполняется добавление строки в конец таблицы. Процесс добавления выполняется следующим образом: для каждого столбца создается объект соответствующего типа и добавляется в список с помощью функции *AddElem* класса *VirtuaList* либо по логическому номеру, либо в конец списка, *Line* увеличивается на 1.

Функция добавления столбца / `int Addc (int, char *, int Num=0)`

Выполняет процедуру добавления пустого столбца в таблицу по номеру *Num*, если номер не задан, то по умолчанию *Num*=0. Если *Num*=0, то выполняется добавление столбца в конец таблицы. Также в функцию передаются идентификатор типа столбца и имя столбца. Процесс добавления выполняется следующим образом: перераспределяется память с помощью функции *realloc* затем сдвигаются вправо на одну позицию все элементы массивов *Mas*, *NameColumn*, *IdType*, начиная с позиции добавления, если *Num*=0, то сдвиг не осуществляется, затем формируется список из *Line* элементов и заносится в данную позицию массива *Mas*, а также заносятся в соответствующие позиции имя столбца и идентификатор типа его элементов, *Column* увеличивается на 1.

Функция удаления строки / `int Dell (int)`

Выполняет процедуру удаления строки с заданным номером. Процесс удаления происходит следующим образом: для каждого столбца вызывается функция *DelElem* класса *VirtuaList*, *Line* уменьшается на 1.

Функция удаления столбца / `int Delc (int)`

Выполняет процедуру удаления столбца с заданным номером. Процесс удаления происходит следующим образом: уничтожается список с заданным номером, затем для каждого элемента, лежащего правее заданной позиции, массивов *Mas*, *NameColumn*, *IdType* выполняется сдвиг влево но одну позицию, затем память перераспределяется с помощью функции *realloc*, урезая размеры всех массивов, *Column* уменьшается на 1.

Функция вывода на экран основных параметров таблицы / `void OutputP (void)`

Выполняет вывод в стандартный выходной поток размеров таблицы *Column* и *Line*.

Ф ункция вывода на экран таблицы / `void Output (void)`

Выполняет вывод на экран таблицы в следующем виде: вывод происходит по столбцам, то есть первыми выводятся атрибуты столбца (порядковый номер, имя, идентификатор типа), затем выводятся элементы данного столбца. Вывод происходит с задержками, то есть после вывода каждых 4-х столбцов происходит останов с запросом нажатия любой клавиши клавиатуры для продолжения вывода.

Функция сортировки элементов таблицы / `int Sort (int Num=0, int S=2)`

Выполняет сортировку столбца таблицы с номером *Num*, если номер не задан, то по умолчанию *Num*=0. Если *Num*=0, то выполняется сортировка каждого столбца таблицы. Также есть возможность выбрать алгоритм, с помощью которого выполняется сортировка, что обеспечивает параметр *S*, который по умолчанию установлен для быстрой сортировки. Процесс сортировки выполняется с помощью вызова функции *Sort* класса *VirtuaList* для каждого столбца или для какого-то одного столбца.

Функция редактирования элемента таблицы / `int Edit (int, int)`

Выполняет редактирование заданного элемента таблицы, путем считывания нового значения с клавиатуры для соответствующего объекта. Процесс получения заданного объект осуществляется с помощью функции *Get* класса *Table*.

Функция сохранения таблицы в двоичный файл / `int Save (char *)`

Выполняет процедуру сохранения данных таблицы в двоичный файл с заданным именем. Для выполнения данного действия используется объект класса *BinFile*, а в частности его функция *Append*, с помощью которой происходит создание записей переменной длины в двоичном файле. Для осуществления согласованности действий используется файловый указатель *pos* на текущую позицию в двоичном файле.

Функция загрузки таблицы из двоичного файла / `int Load (char *)`

Выполняет процедуру загрузки данных таблицы из двоичного файла с заданным именем. Для выполнения данного действия используется объект класса *BinFile*, а в частности его функция *Load*, с помощью которой производится чтение записей переменной длины из двоичного файла. Для осуществления согласованности действий используется файловый указатель *pos* на текущую позицию в двоичном файле.

Функция генерации таблицы / `int Gen (int Num=0)`

Выполняет процедуру заполнения столбца таблицы или всей таблицы (зависит от параметра *Num*) случайными значениями. Если *Num* не задан, то по умолчанию он равен 0. Если *Num*=0, то выполняется генерация значений для всей таблицы. Процесс генерации обеспечивается функцией *Random* класса *Table*.

Функция генерации столбца / `void Random (int)`

Выполняет процедуру заполнения элементов списка, столбца таблицы с заданным номером, случайными значениями генерируемыми функцией *rand*. Для доступа к элементам списка используется функция *Get* класса *VirtuaList*, значение сгенерированной строки вносится в объект абстрактного типа данных с помощью виртуальной функции *Input* класса *Object*.

Класс VirtuaList

Реализует структуру данных типа «двусвязный список». С возможностью хранения элементов абстрактного типа данных Object, а также с возможностью сохранения данных в двоичный файл и загрузки данных из двоичного файла. Содержит следующий набор функций:

Функция очистки списка / `void Clean (void)`

Выполняет процедуру очистки содержимого списка. То есть происходит уничтожение элементов списка и установка параметров списка в нулевые значения.

Функция добавления по логическому номеру / `int AddElem (Object *, int)`

Выполняет процедуру добавления объекта абстрактного типа данных *Object* в список по логическому номеру. Если подразумевается добавление в конец списка, то происходит вызов альтернативной функции *AddElem* класса *VirtuaList* предназначенной для быстрой вставки объекта в конец списка.

Функция добавления в конец списка / `int AddElem (Object *)`

Выполняет процедуру вставки объекта абстрактного типа данных *Object* в конец списка. Быстрый доступ к концу списка осуществляется с помощью переменной класса End, в которой хранится адрес последнего элемента списка.

Функция удаления по логическому номеру / `void DelElem (int)`

Выполняет процедуру удаления элемента из списка по логическому номеру. Высвобождается память, используемая удаляемым элементом, и происходит перестройка указателей соседних объектов. Если нужно меняются значения указателей на начало списка, конец списка.

Функция поиска номера максимального элемента / `int Max (void)`

Выполняет поиск элемента списка с наибольшим значением. Так как поиск использует функцию сравнения элементов *Cmp* класса *Object*, то для проверки однотипности объектов списка используется функция *bCorrect* класса *VirtuaList*.

Функция поиска номера минимального элемента / `int Min (void)`

Выполняет поиск элемента списка с наименьшим значением. Для осуществления поиска используется альтернативная функция *Min* класса *VirtuaList* для поиска номера минимального элемента в подсписке длиной меньшей либо равной *N* (размер списка). Также осуществляется проверка на однотипность элементов списка с помощью функции *bCorrect* класса *VirtuaList*.

Функция сортировки элементов списка / `int Sort (int Num=1)`

Выполняет процедуру упорядочения элементов списка по возрастанию их значений одним из трех способов, которые организуются с помощью функций *Qsort*, *SelectSort* и *Merge* класса *VirtuaList*. Выбор осуществляется после анализа значения параметра *Num*, который по умолчанию равен 1. Если *Num*=1, то выполняется сортировка выбором, если 2, то быстрая сортировка, если 3, то сортировка слиянием.

Функция получения элемента с заданным номером / `Object *Get (int)`

Выполняет поиск элемента в списке по заданному логическому номеру.

Функция получения текущего элемента / `Object *Get (void)`

Возвращает текущий элемент списка, данные о адресе которого находятся в переменной *Current*.

Функция сохранения в двоичный файл / `int Save (char *, FPTR &)`

Выполняет процедуру сохранения данных списка в двоичный файл с заданным именем, начиная с заданной позиции. Для сохранения используется объект класса *BinFile*,

а в частности его функция *Append*, с помощью которой производится создание записей переменной длины в двоичном файле. Для сохранения элементов используется виртуальная функция *Save* класса *Object*.

Функция загрузки из двоичного файла / `int Load (char *, FPTR &)`

Выполняет процедуру чтения данных списка из двоичного файла и конструирования списка из считанных элементов. Для работы с двоичным файлом используется объект класса *BinFile*, а в частности его функция *Load*, с помощью которой производится считывание записей переменной длины из двоичного файла. Для загрузки элементов используется функция *Load* класса *Object*.

Функция проверки однотипности элементов / `int bCorrect (void)`

Выполняет анализ содержимого списка и сообщает о сравнимости элементов списка. Для установки сравнимости элементов используется виртуальная функция Id класса Object.

Функция поиска номера минимального элемента подсписка / `int Min (int)`

Выполняет процедуру поиска номера минимального элемента в подсписке основного списка до заданного элемента. Функция введена в основном для работы с обменной сортировкой. Реализован оригинальный алгоритм, представленный в пункте «Структурное описание разработки». Процедура поиска осуществляется с использованием виртуальной функции сравнения *Cmp* класса *Object*.

Функция обменной сортировки / `void SelectSort (void)`

Выполняет упорядочение элементов списка по возрастанию значений. Для осуществления этой процедуры используется функция поиска номера минимального элемента подсписка *Min* класса *VirtuaList*. В функции реализован оригинальный алгоритм, представленный в пункте «Структурное описание разработки».

Функция быстрой сортировки / `void Qsort (int, int)`

Выполняет упорядочение элементов списка по возрастанию значений. Функция является рекурсивной, в качестве параметров которой используются номера элементов списка, являющихся левой и правой границами подсписка, сортируемого на данном шаге рекурсии. Для сравнения элементов используется виртуальная функция *Cmp* класса *Object*. В функции реализован оригинальный алгоритм, представленный в пункте «Структурное описание разработки».

Функция сортировки слиянием / `void Merge (int, int)`

Выполняет упорядочение элементов списка по возрастанию значений. Функция является рекурсивной, в качестве параметров которой используются номера элементов списка, являющихся левой и правой границами подсписка, сортируемого на данном шаге рекурсии. Если размер подсписка равен 2, то, если нужно, происходит простой обмен элементов методом «трех стаканов» без изменения структуры списка. Для сравнения элементов используется виртуальная функция *Cmp* класса *Object*. В функции реализован оригинальный алгоритм, представленный в пункте «Структурное описание разработки».

Класс Object

Реализует интерфейс работы с объектом абстрактного типа. Содержит следующие функции:

Функция получения идентификатора типа / `virtual int Id (void)=0`

Возвращает уникальный идентификатор типа объекта. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция получения имени типа / `virtual char *Name (void)=0`

Возвращает имя типа объекта. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция инициализации объекта / `virtual int Input (char *)=0`

Выполняет загрузку нового значения объекта из переданной строки. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция вывода объекта в строку / `virtual char *Output (void)=0`

Возвращает строку со значением объекта. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция сравнения / `virtual int Cmp (Object &)=0`

Выполняет процедуру сравнения текущего объекта с переданным объектом. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция сложения / `virtual void Add (Object &)=0`

Выполняет процедуру добавления к текущему объекту значения переданного объекта. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция копии / `virtual Object *Copy (void)=0`

Возвращает копию текущего объекта. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция сохранения / `virtual FPTR Save (char *, FPTR)=0`

Выполняет процедуру сохранения текущего объекта в двоичный файл с заданным именем в заданную позицию. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция загрузки / `virtual FPTR Load (char *, FPTR)=0`

Выполняет процедуру загрузки значения объекта из двоичного файла с заданным именем из заданной позиции. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция вывода / `virtual ostream &out (ostream &) =0`

Выводит данные объекта в заданный поток. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Функция ввода / `virtual istream &in (istream &) =0;`

Выполняет загрузку данных объекта из заданного потока. Конкретная реализация находится в классе, описывающем тип данного объекта с одноименным названием.

Класс BinFile

Реализует интерфейс работы с двоичным файлом последовательного доступа, содержащего записи переменной длины. Содержит следующий набор функций:

Функция создания двоичного файла / `BOOL Create (char *)`

Выполняет создание на жестком диске файла с заданным именем. Функция создана на базе функции *open* класса *fstream*. В первую позицию созданного файла заносится адрес последней записи, который хранится в переменной *LastRec*.

Функция открытия двоичного файла / `BOOL Open (char *)`

Выполняет открытие файла с заданным именем. Функция создана на базе функции *open* класса *fstream*. При открытии происходит считывание из первой позиции адреса последней записи и сохранение его в переменной *LastRec*.

Функция получения размера двоичного файла / `FPTR Size (void)`

Возвращает адрес последней позиции в двоичном файле. Функция создана на основе функций *seekg* и *tellg* класса *fstream*.

Функция добавления записи / `FPTR Append (void *, int)`

Выполняет процедуру добавления новой записи в конец двоичного файла. Процесс записи заключается в записи размера переданных данных и записи самих данных. Функция использует функции *write*, *tellg*, *seekp* класса *fstream*, а также функцию *Size* класса *BinFile* для проверки корректности записи.

Функция чтения записи / `void *Load (int &)`

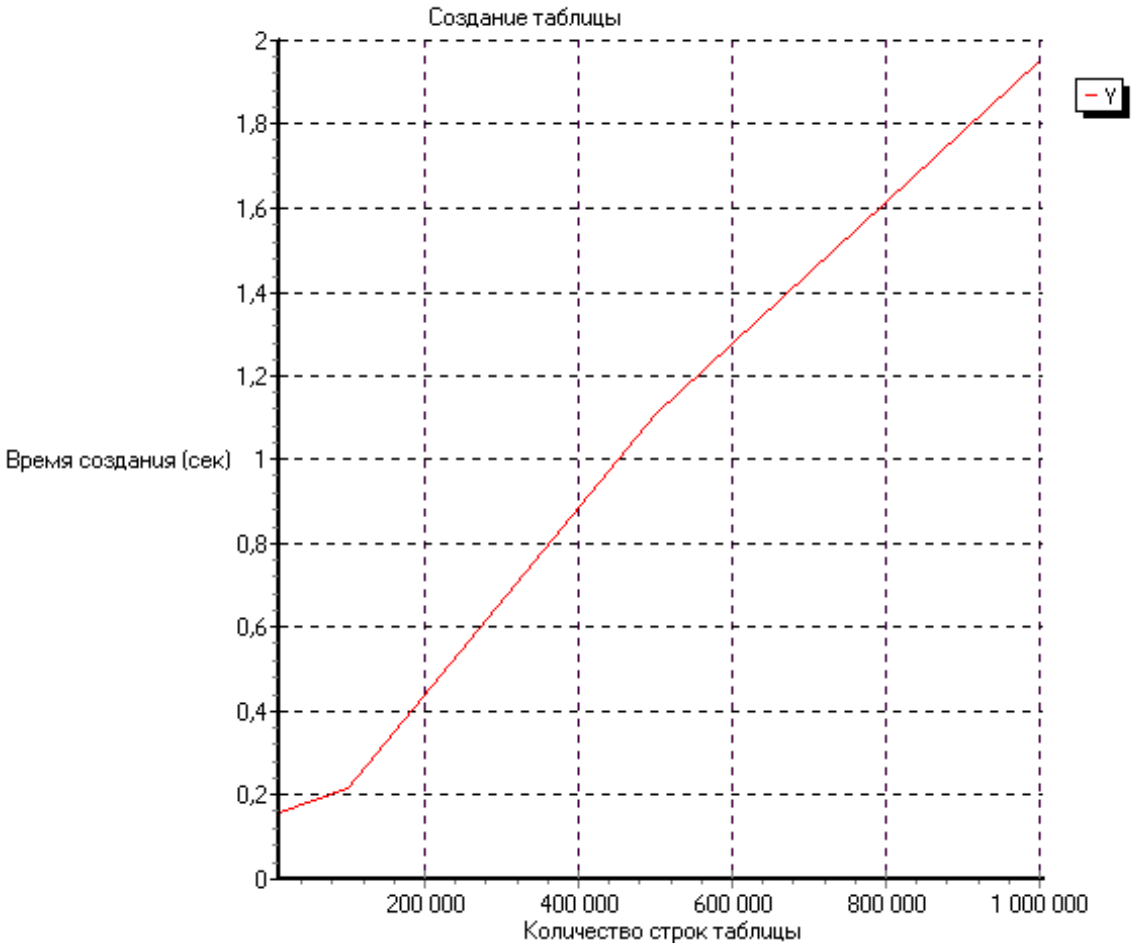
Выполняет процедуру считывания записи из двоичного файла из текущей позиции. Порядок считывания тот же, что и записи в функции *Append* класса *BinFile*. Для считывания используется функция *read* класса *fstream*.

Описание работы программы.

Для описания работы программы протестируем ее основные методы работы с таблицей, затратные по времени, а точнее: создание таблицы большого размера, сортировка таблицы большого размера, сохранение таблицы в файл, загрузка таблицы из файла, генерация случайных значений таблицы. Для упрощения задачи будем использовать таблицы с двумя столбцами и переменным количеством строк.

1. Создание таблицы

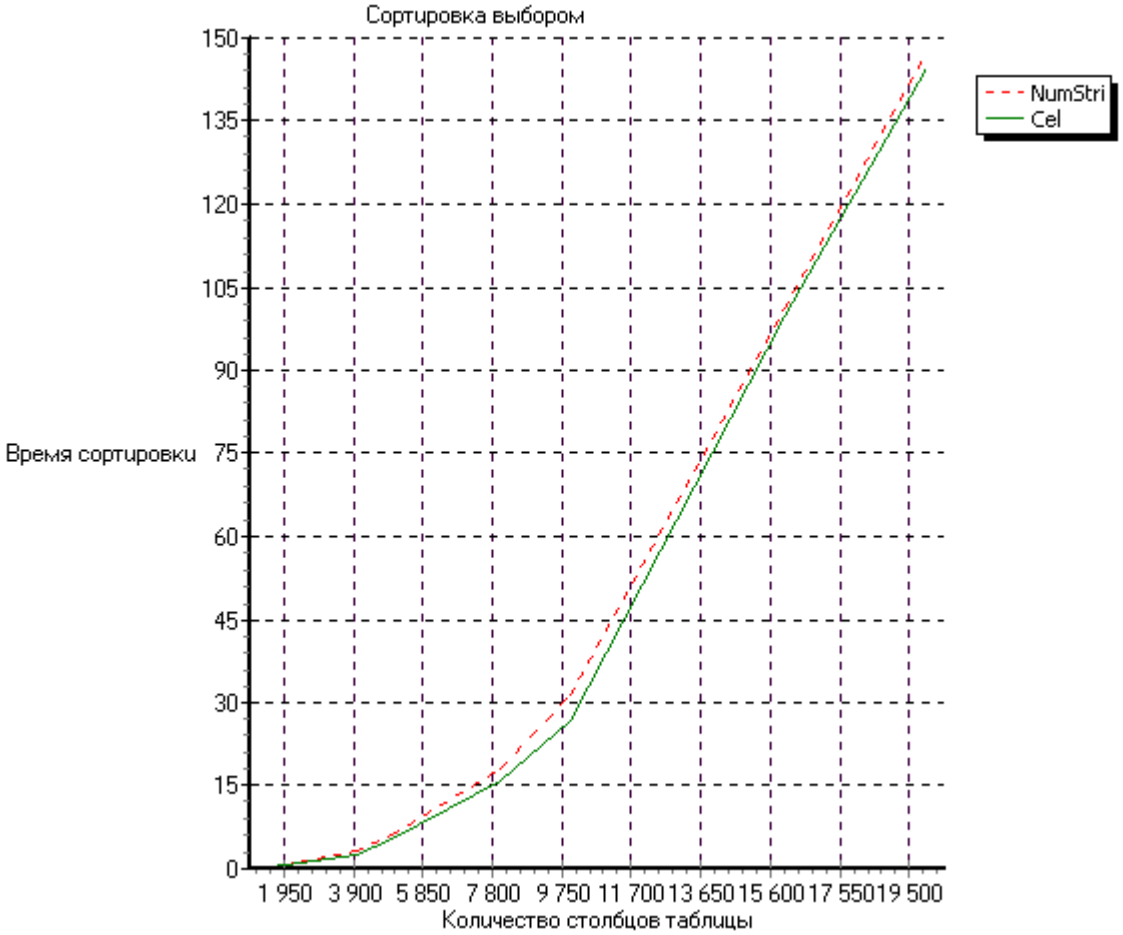
Количество строк таблицы	Время создания таблицы (сек)
1000	0,016
10000	0,016
100000	0,219
500000	1,109
1000000	1,953



2. Сортировка таблицы

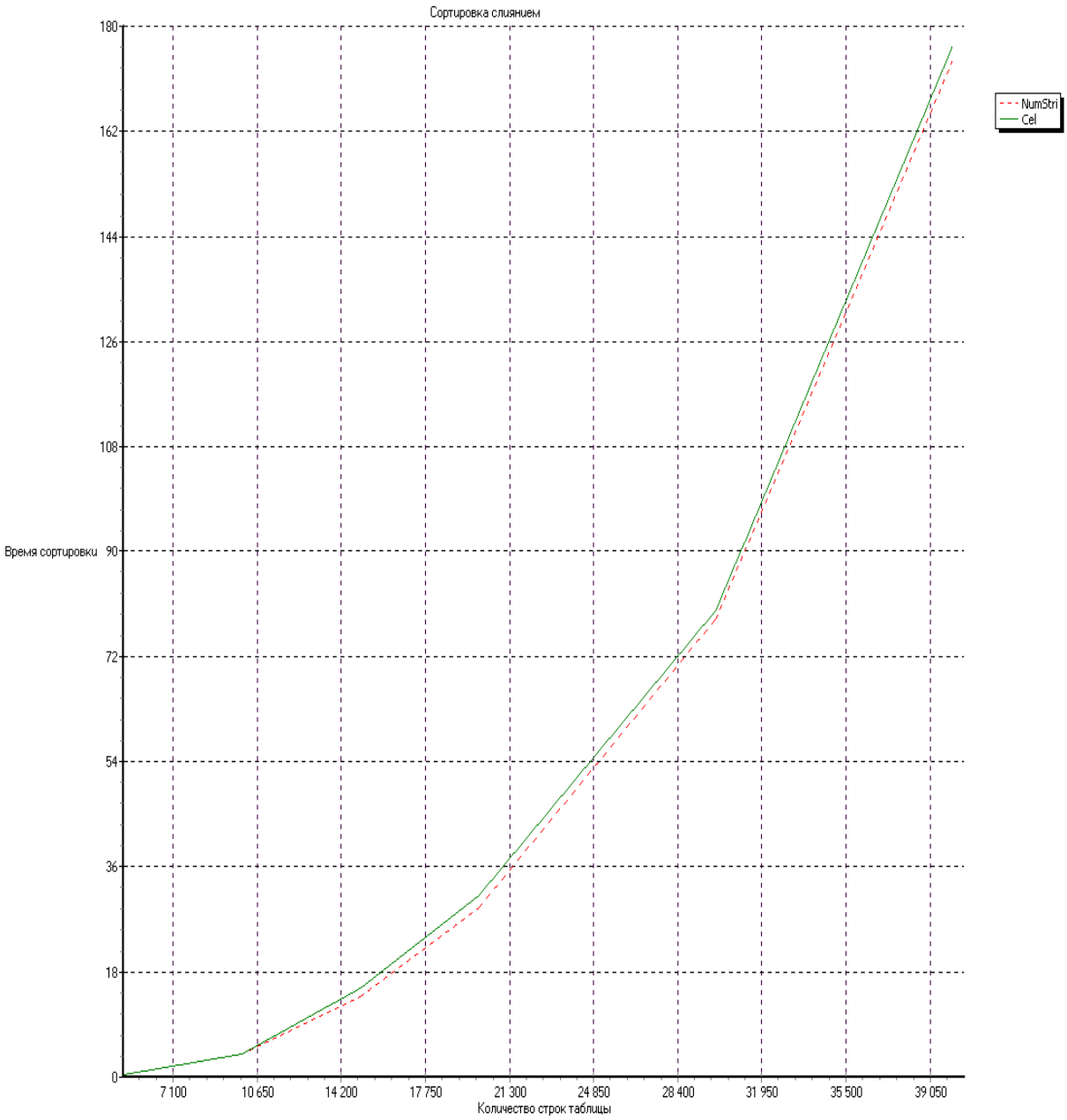
Сортировка выбором

Количество строк таблицы	Время сортировки (Тип элементов NumString) (сек)	Время сортировки (Тип элементов Cel) (сек)
1000	0.125	0.125
2000	0.610	0.562
4000	3.281	2.547
5000	6.063	5.266
8000	18.25	16.032
10000	31.266	26.625
16000	101.156	99.907
20000	147.36	144.734



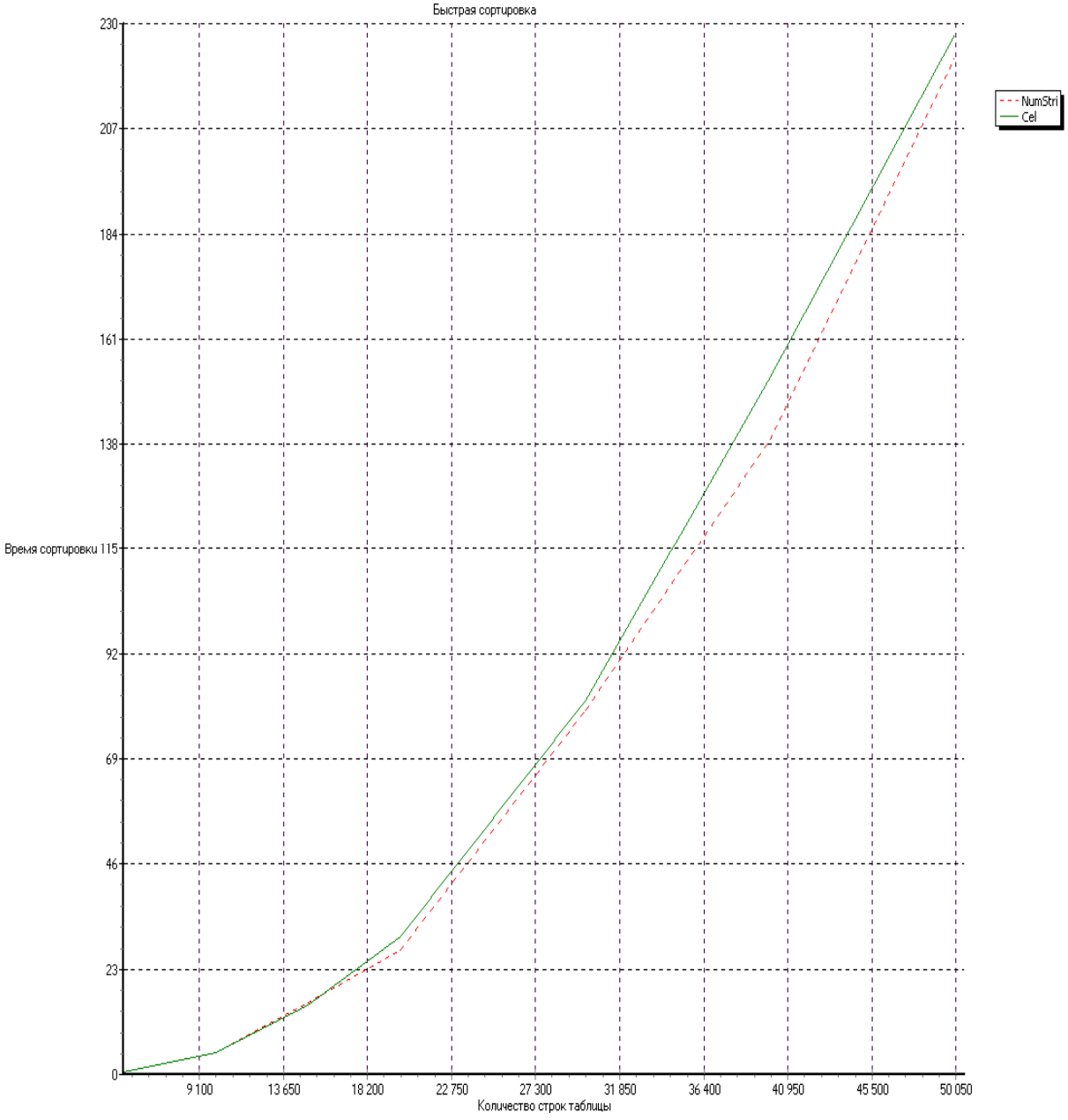
Сортировка слиянием

Количество строк таблицы	Время сортировки (Тип элементов NumString) (сек)	Время сортировки (Тип элементов Cel) (сек)
5000	0.375	0.344
10000	3.953	4.015
15000	13.797	15.156
20000	29.094	31.125
30000	78.406	79.859
40000	174.234	176.609



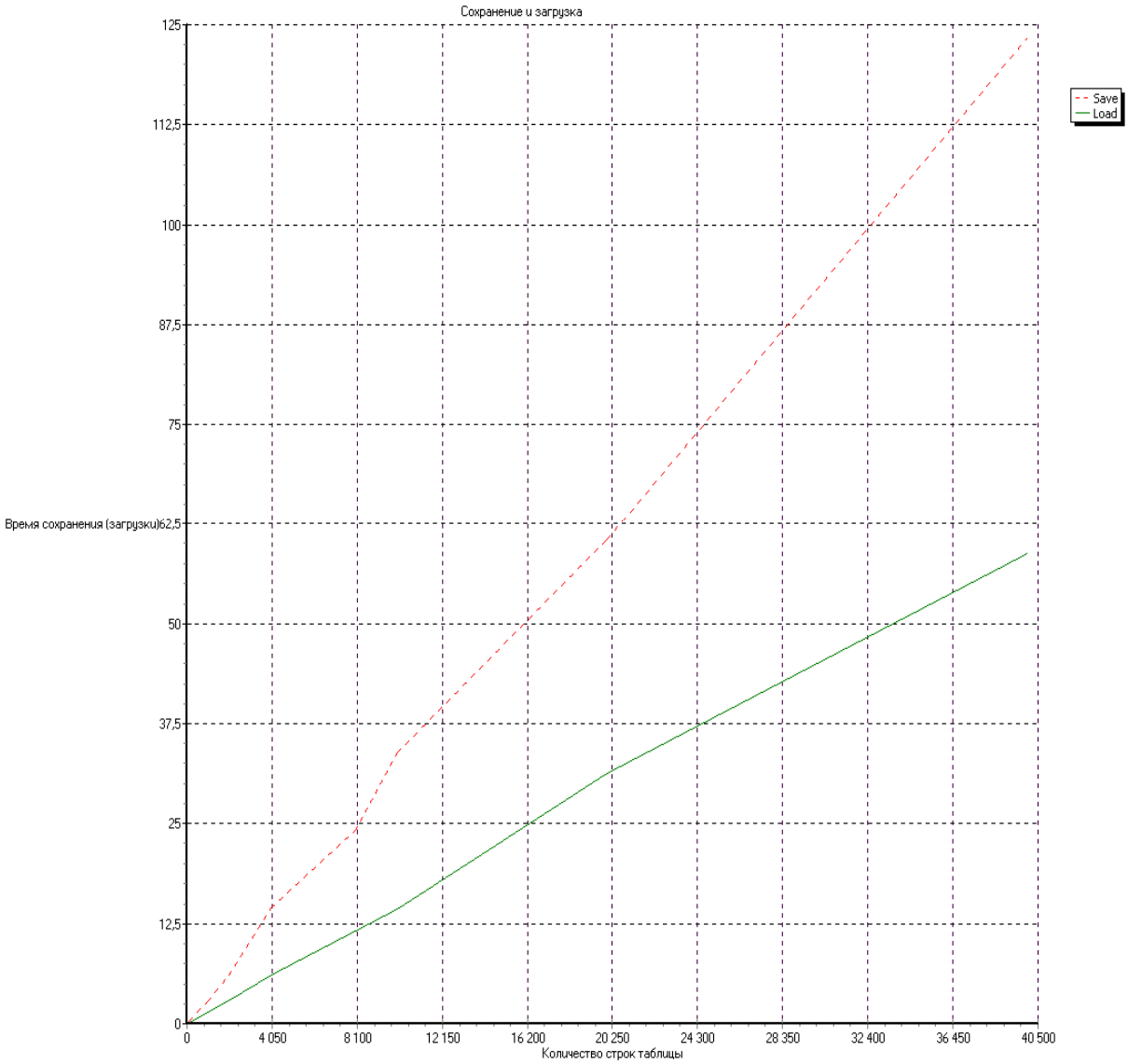
Быстрая сортировка

Количество строк таблицы	Время сортировки (Тип элементов NumString) (сек)	Время сортировки (Тип элементов Cel) (сек)
5000	0.625	0.61
10000	4.766	4.781
15000	15.687	15.203
20000	27.109	30.046
30000	79.813	81.813
40000	139.110	152.766
50000	222.578	227.641



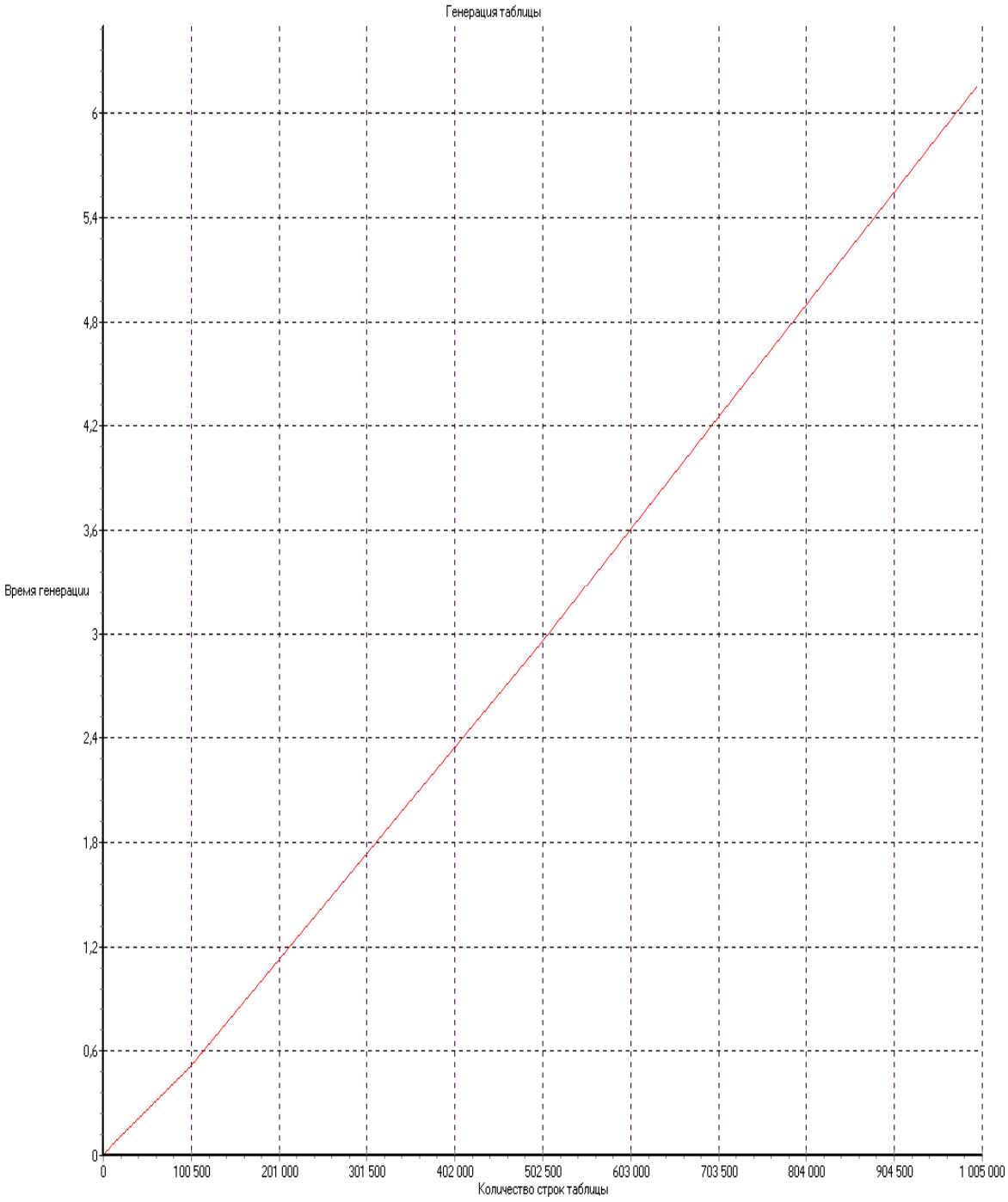
3. Сохранение в файл и загрузка таблицы из файла

Количество строк таблицы	Время сохранения в файл (сек)	Время загрузки из файла (сек)
100	0.281	0.151
500	1.422	0.701
1000	2.824	1.442
2000	6.069	2.925
4000	14.501	6.028
8000	24.125	11.627
10000	33.889	14.33
20000	60.503	31.275
40000	123.468	58.791



4. Генерация случайных значений таблицы

Количество строк таблицы	Время генерации (сек)
1000	0.01
10000	0.06
100000	0.511
500000	2.943
1000000	6.159



Из данных временных характеристик можно сделать несколько выводов:

Временные расходы на создание таблицы достаточно малы, также как и расходы на генерацию новых значений таблицы.

Операции сохранения и загрузки достаточно весомы, и их лучше не применять для таблиц с количеством строк больше 50000. Операция загрузки таблицы выполняется в два раза быстрее, чем операция сохранения.

Сортировка выбором значительно отстает по времени от сортировки слиянием и быстрой сортировки.

Сортировка слиянием дает лучшие результаты на небольших наборах данных, но быстрая сортировка выигрывает на наборах данных большого размера.

Не имеет смысла запускать сортировку для таблиц с количеством строк более 50000, так как в этом случае сортировка выполняется довольно большой промежуток времени.

Как видно из результатов исследования мы получили довольно «слабые» временные характеристики программы. Скорее всего, причиной этого послужило то, что мы использовали дополнительную структуру данных – список для реализации столбцов таблицы. Вследствие этого мы имели проблемы с непосредственным доступом к нужному значению таблицы и на поиск этого значения уходили дополнительные затраты процессорного времени и памяти. Одним из предполагаемых решений является введение дополнительной арности массива указателей для хранения столбцов элементов. В результате чего, мы получим структуру данных с «естественным» доступом к любому элементу таблицы и сможем улучшить показатели нашей программы.