МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики Кафедра математического обеспечения и суперкомпьютерных технологий**

**ОТЧЕТ**

по учебной практике на тему:

**«Способы организации таблиц»**

**Выполнил:** студент группы 381708-2

Сазанов Д. Е.

Подпись

**Проверил:** к.т.н. доцент каф. Программной инженерии ИИТММ

Штанюк А.А.

Подпись

Нижний Новгород 2019.

# Содержание

[Формулировка задания 3](#_Toc10491111)

[Анализ задания 4](#_Toc10491112)

[Программная реализация 7](#_Toc10491113)

[Классы, используемые в работе: 7](#_Toc10491114)

[Спецификации классов 9](#_Toc10491115)

[Результаты тестирования 16](#_Toc10491116)

[Выводы 18](#_Toc10491117)

[Список источников 19](#_Toc10491118)

[Приложение 20](#_Toc10491119)

# Формулировка задания

Работа направлена на изучение способов организации таблиц и начальное знакомство с принципами проектирования структур хранения, используемых в методах решения прикладных задач. На примере таблиц изучаются возможность выбора разных вариантов структур хранения, анализ их эффективности и определения областей приложений, в которых выбираемые структуры хранения являются наиболее эффективными.

В качестве практической задачи, на примере которой будут продемонстрированы возможные способы организации таблиц, рассматривается проблема статистической обработки результатов экзаменационной успеваемости студентов (выполнение таких, например, вычислений как определение среднего балла по предмету и/или по группе при назначении студентов на стипендию или при распределении студентов по кафедрам и т.п.).

# Анализ задания

Представление данных во многих задачах из разных областей человеческой деятельности может быть организовано при помощи таблиц. Таблицы представляют собой последовательности строк (записей), структура строк может быть различной, но обязательным является поле, задающее имя (ключ) записи.

Таблицы применяются в бухгалтерском учете (ведомости заработной платы), в торговле (прайс-листы), в образовательных учреждениях (экзаменационные ведомости) и являются одними из наиболее распространенных структур данных, используемых при создании системного и прикладного математического обеспечения. Таблицы широко применяются в трансляторах (таблицы идентификаторов) и операционных системах, могут рассматриваться как программная реализация ассоциативной памяти и т.п.

Существование отношения «иметь имя» является обязательным в большинстве разрабатываемых программистами структур данных; доступ по имени в этих структурах служит для получения соответствия между адресным принципом указания элементов памяти ЭВМ и общепринятым (более удобным для человека) способом указания объектов по их именам.

Основные понятия и определения

**Таблица** (от лат. *tabula* – доска) – динамическая структура данных, базисным множеством которой является семейство линейных структур из записей (базисное отношение включения определяется операциями вставки и удаления записей).

**Запись** – кортеж, каждый элемент которого обычно именуется полем.

**Имя записи (ключ)** – одно из полей записи, по которому обычно осуществляется поиск записей в таблице; остальные поля образуют тело записи.

Двоичное дерево поиска – это представление данных в виде дерева, для которого выполняются условия:

* для любого узла (вершины) дерева существует не более двух потомков (двоичное дерево);
* для любого узла значения во всех узлах левого поддерева меньше значения в узле;
* для любого узла значения во всех узлах правого поддерева больше значения в узле.

**Хеш-функция** – функция, ставящая в соответствие ключу номер записи в таблице (используется при организации таблиц с вычислимым входом).

В рамках данной работы ставится задача создания программных средств, поддерживающих табличные динамические структуры данных (таблицы) и базовые операции над ними:

* поиск записи;
* вставка записи (без дублирования);
* удаление записи.

Выполнение операций над таблицами может осуществляться с различной степенью эффективности в зависимости от способа организации таблицы.

В рамках работы, показатель эффективности — это количество операций, необходимых для выполнения операции поиска записи в таблице. Величина этого показателя должна определяться как аналитически (при использовании тех или иных упрощающих предположений), так и экспериментально на основе проведения вычислительных экспериментов.

В работе предлагается реализовать следующие типы таблиц:

* просмотровые (неупорядоченные);
* упорядоченные (сортированные);
* таблицы со структурами хранения на основе деревьев поиска;
* хеш-таблицы или перемешанные (с вычисляемыми адресами).

Необходимо разработать интерфейс доступа к операциям поиска, вставки и удаления, не зависящий от способа организации таблицы.

Демонстрацию работоспособности разрабатываемых при выполнении работы программ следует проводить на примере таблиц, содержащих данные о результатах экзаменационной успеваемости студентов. Для данного примера следует реализовать следующие операции:

1. Для таблицы с результатами успеваемости отдельной студенческой группы:

* получение сведений об успеваемости отдельного студента (оценка по конкретному предмету, средняя оценка по всем предметам);
* определение средней оценки по группе по отдельному предмету или по всем предметам;
* подсчет количества студентов-отличников и т.п.

1. Для нескольких таблиц с результатами успеваемости для всех студенческих групп курса:

* получение средних оценок по всем студенческим группам по отдельному предмету или по всем предметам;
* определение студенческой группы с лучшей успеваемостью по конкретному предмету (или по всем предметам экзаменационной сессии);
* подсчет количества студентов-отличников для всего курса и т.п.

Задача обработки результатов успеваемости может быть расширена возможностью хранения данных для нескольких экзаменационных сессий (например, за весь период обучения студентов).

# Программная реализация

## Классы, используемые в работе:

* **TTabRecord.h**, **TTabRecord.cpp** – модуль с классом объектов-значений для записей таблицы;
* **TTable.h** – абстрактный базовый класс, содержит спецификации методов таблицы;
* **TArrayTable.h**, **TArrayTable.cpp** – абстрактный базовый класс для таблиц с непрерывной памятью;
* **TScanTable.h**, **TScanTable.cpp** – модуль с классом, обеспечивающим реализацию просматриваемых таблиц;
* **TSortTable.h**, **TSortTable.cpp** – модуль с классом, обеспечивающим реализацию упорядоченных таблиц;
* **TTreeNode.h**, **TTreeNode.cpp** – модуль с абстрактным базовым классом объектов-значений для деревьев;
* **TTreeTable.h**, **TTreeTable.cpp** – модуль с классом, реализующим таблицы в виде деревьев поиска;
* **TBalanceNode.h**, **TBalanceNode.cpp** – модуль с базовым классом объектов-значений для сбалансированных деревьев;
* **TBalanceTree.h**, **TBalanceTree.cpp** – модуль с классом, реализующим таблицы в виде сбалансированных деревьев поиска;
* **THashTable.h**, **THashTable.cpp** – модуль с базовым классом, обеспечивающим реализацию таблиц с вычислимым входом;
* **TArrayHash.h**, **TArrayHash.cpp** – модуль с классом, обеспечивающим реализацию хеш-таблиц (разрешение коллизий на основе открытого перемешивания);
* **TListHash.h**, **TListHash.cpp** – модуль с классом, обеспечивающим реализацию хеш-таблиц (разрешение коллизий на основе метода цепочек);
* **TTableTestkit.cpp** – модуль программы тестирования.

## Спецификации классов

Класс объектов-значений для записей таблицы (файл TTabRecord.h):

typedef string TKey // тип ключа записи

// Класс объектов-значений для записей таблицы

class TTabRecord : public TDatValue

{

protected: // поля

TKey Key; // ключ записи

PTDatValue pValue; // указатель на значение

public: // методы

TTabRecord (TKey k=””, PTDatValue pVal = NULL)// конструктор

void SetKey(TKey k);// установить значение ключа

TKey GetKey(void); // получить значение ключа

void SetValuePtr(PTDatValue p);// установить указатель на данные

PTDatValue GetValuePTR (void); // получить указатель на данные

virtual TDatValue \* GetCopy(); // изготовить копию

TTabRecord & operator = (TTabRecord &tr);// присваивание

virtual int operator == (const TTabRecord &tr); // сравнение =

virtual int operator < (const TTabRecord &tr); // сравнение «<»

virtual int operator > (const TTabRecord &tr); // сравнение «>»

//дружественные классы для различных типов таблиц, см. далее

friend class TArrayTable;

friend class TScanTable;

friend class TSortTable;

friend class TTreeNode;

friend class TTreeTable;

friend class TArrayHash;

friend class TListHash;

};

Тип TKey описывает тип значений ключей записи. В данном примере для типа ключей использован класс string из стандартной библиотеки шаблонов (STL).

Абстрактный базовый класс содержит спецификации методов таблицы (TTable.h).

class TTable : public TDataCom {

protected:

int DataCount; // количество записей в таблице

int Efficiency; // показатель эффективности выполнения операции

public:

TTable(){ DataCount=0; Efficiency=0;} // конструктор

virtual ~TTble( ) {}; // деструктор

// информационные методы

int GetDataCount ( ) const {return DatCount;} // к-во записей

int GetEfficiency ( ) const {return Efficiency;} // эффективность

int IsEmpty ( ) const {return DataCount == 0;} //пуста?

virtual int IsFull ( ) const =0; // заполнена?

// доступ

virtual TKey GetKey (void) const=0;

virtual PTDatValue GetValuePTR (void) const =0;

// основные методы

virtual PTDatValue FindRecord (TKey k) =0; // найти запись

virtual void InsRecord (TKey k, PTDatValue pVal ) =0; // вставить

virtual void DelRecord (TKey k) =0; // удалить запись

// навигация

virtual int Reset (void) =0; // установить на первую запись

virtual int IsTabEnded (void) const=0; // таблица завершена?

virtual int GoNext (void) =0; // переход к следующей записи

// (=1 после применения для последней записи таблицы)

};

Все методы данного класса разделены на четыре группы.

1. Информационные методы

* GetDataCount – получить количество записей,
* GetEfficiency – получить эффективность последней операции,
* IsEmpty – проверить, не является ли таблица пустой,
* IsFull – проверить, не является ли таблица полной.

1. Методы доступа к записям

* GetKey – получить значение ключа текущей записи ,
* GetValuePTR – получить указатель на значение текущей записи.

1. Методы навигации по таблице (итератор)

* Reset – установить текущую позицию на первую запись,
* IsTabEnded – проверка завершения таблицы (под ситуацией завершения таблицы понимается состояние после применения метода GoNext для текущей позиции, установленной на последней записи таблицы),
* GoNext – переместить текущую позицию на следующую запись.

1. Методы обработки таблицы

* FindRecord – поиск записи по значению ключа,
* InsRecord – вставка записи в таблицу,
* DelRecord – удаление записи.

Абстрактный класс для таблиц с непрерывной памятью служит для управления структурой хранения (TArrayTable)

#define TabMaxSize 25

enum TDataPos {FIRST\_POS, CURRENT\_POS, LAST\_POS};

class TArrayTable : public TTable {

protected:

PTTabRecord \*pRecs; // память для записей таблицы

int TabSize; // макс. возм.количество записей в таблице

int CurrPos; // номер текущей записи (нумерация с 0)

public:

TArrayTable(int Size=TabMaxSize); // конструктор

~TArrayTble( ) {}; // деструктор

// информационные методы

virtual int IsFull ( ) const ; // заполнена?

int GetTabSize( ) const ; // к-во записей

// доступ

virtual TKey GetKey (void) const;

virtual PTDatValue GetValuePTR (void) const;

virtual TKey GetKey (TDataPos mode) const;

virtual PTDatValue GetValuePTR (TDataPos mode) const;

// основные методы

virtual PTDatValue FindRecord (TKey k) =0; // найти запись

virtual void InsRecord (TKey k, PTDatValue pVal ) =0; // вставить

virtual void DelRecord (TKey k) =0; // удалить запись

//навигация

virtual int Reset (void); // установить на первую запись

virtual int IsTabEnded (void) const; // таблица завершена?

virtual int GoNext (void) ; // переход к следующей записи

//(=1 после применения для последней записи таблицы)

virtual int SetCurrentPos (int pos);// установить текущую запись

int GetCurrentPos (void) const; //получить номер текущей записи

friend TSortTable;

};

Данный класс обеспечивает управление памятью (выделение и освобождение памяти). Методы доступа применимы к первой, текущей и последней записям таблицы, желаемый вариант доступа задается через параметр метода доступа. Методы SetCurrentPos (установить текущую позицию на запись с заданным номером) и GetCurrentPos (получить номер текущей записи) вводят операции прямого доступа к записям таблицы. В классе реализованы методы навигации.

Класс, обеспечивающий реализацию просматриваемых таблиц (TScanTable).

class TScanTable: public TArrayTable {

public:

TScanTable(int Size=TabMaxSize): TArrayTable(Size){};//конструктор

// основные методы

virtual PTDatValue FindRecord (TKey k) ;//найти запись

virtual void InsRecord (TKey k, PTDatValue pVal ) ;//вставить

virtual void DelRecord (TKey k) ;//удалить запись

};

В данном классе реализованы методы обработки таблицы.

Упорядоченные таблицы (TSortTable).

enum TSortMethod {INSERT\_SORT, MERGE\_SORT, QUIQ\_SORT};

class TSortTable: public TScanTable {

protected:

TSortMethod SortMethod; // метод сортировки

void SortData (void); // сортировка данных

void InsertSort (PTTabRecord \*pMem, int DataCount); // метод вставок

void MergeSort (PTTabRecord \*pMem, int DataCount); // метод слияния

void MergeSorter (PTTabRecord \* &pData,PTTabRecord \* &pBuff,int Size);

void MergeData (PTTabRecord \*&pData,PTTabRecord \*&pBuff,int n1,int n2);

void QuiqSort (PTTabRecord \*pMem, int DataCount); // быстрая сортировка

void QuiqSplit (PTTabRecord \*pData, int Size, int &Pivot);

public:

TSortTable(int Size=TabMaxSize): TScanTable(Size){};// конструктор

TSortTable(const TScanTable &tab); // из просматриваемой таблицы

TSortTable & operator=(const TScanTable &tab); // присваивание

TSortMethod GetSortMethod(void); // получить метод сортировки

void SetSortMethod (TSortMethod sm);// установить метод сортировки

// основные методы

virtual PTDatValue FindRecord (TKey k) ; // найти запись

virtual void InsRecord (TKey k, PTDatValue pVal ) ; // вставить

virtual void DelRecord (TKey k) ; // удалить запись

};

В данном классе приводятся три метода сортировки, другие методы сортировки могут рассматриваться как задания для самостоятельного выполнения. Метод поиска записи FindRecord основан на алгоритме бинарного поиска, в методах вставки InsRecord и удаления DelRecord происходит обращение к методу поиска и производится перепаковка таблицы.

Абстрактный базовый класс объектов-значений для деревьев (TTreeNode).

class TTreeNode;

typedef TTreeNode \*PTTreeNode;

class TTreeNode: public TTabRecord {

protected:

PTTreeNode pLeft, pRight; // указатели на поддеревья

public:

TTreeNode(TKey k=””, PTDatValue pVal=NULL, PTTreeNode pL=NULL,

PTTreeNode pR=NULL ): TTabRecord(k,pVal), pLeft(pL), pRight(pR) {};

PTTreeNode GetLeft(void) const; // указатель на левое поддерево

PTTreeNode GetRight(void) const; // указатель на правое поддерево

virtual TDatValue \* GetCopy(); // изготовить копию

friend class TTreeTable;

friend class TBalanceTree;

};

Таблицы со структурой хранения в виде деревьев поиска (TTreeTable).

class TTreeTable: public TTable {

protected:

PTTreeNode pRoot; // указатель на корень дерева

PTTreeNode \*ppRef;// адрес указателя на вершину-результата в FindRecord

PTTreeNode pCurrent;// указатель на текущую вершину

int CurrPos; // номер текущей вершины

stack < PTTreeNode> St;// стек для итератора

void DeleteTreeTab (PTTreeNode pNode); // удаление

public:

TTreeTable(): TTable(){CurrPos=0; PRoot=pCurrent=NULL; ppRef=NULL;}

~TTreeTable(){DeleteTreeTab (pRoot);} // деструктор

// информационные методы

virtual int IsFull ( ) const ; //заполнена?

//основные методы

virtual PTDatValue FindRecord (TKey k); // найти запись

virtual void InsRecord (TKey k, PTDatValue pVal ); // вставить

virtual void DelReco rd (TKey k); // удалить запись

// навигация

virtual TKey GetKey (void) const;

virtual PTDatValue GetValuePTR (void) const;

virtual int Reset (void); // установить на первую запись

virtual int IsTabEnded (void) const; // таблица завершена?

virtual int GoNext (void) ;// переход к следующей записи

//(=1 после применения для последней записи таблицы)

};

Для обхода дерева используется стек из библиотеки STL (в нем запоминаются указатели на пройденные узлы дерева).

Базовый класс объектов-значений для сбалансированных деревьев (TBalanceNode).

#define BalOk 0

#define BalLeft -1

#define BalRight 1

class TBalanceNode: public TTreeNode {

protected:

int Balance; // индекс балансировки вершины (-1,0,1)

public:

TBalanceNode (TKey k=””, PTDatValue pVal=NULL, PTTreeNode pL=NULL,

PTTreeNode pR=NULL, int bal=BalOk) ): TTreeNode(k,pVal,pL,pR),

Balance(bal) {}; // конструктор

virtual TDatValue \* GetCopy(); // изготовить копию

int GetBalance(void) const;

void SetBalance(int bal);

friend class TBalanceTree;

};

Класс для сбалансированных деревьев (TBalanceTree).

class TBalanceTree: public TTreeTable {

protected:

int InsBalanceTree(PTBalanceNode &pNode, TKey k, PTDatValue pVal);

int LeftTreeBalancing(PTBalanceNode &pNode); // баланс. левого поддерева

int RightTreeBalancing(PTBalanceNode &pNode);// баланс. правого поддерева

public:

TBalanceTree():TTreeTable(){} // конструктор

//основные методы

virtual void InsRecord (TKey k, PTDatValue pVal ); // вставить

virtual void DelRecord (TKey k); // удалить

};

Дерево является сбалансированным, если для каждого его узла высота левого и правого поддеревьев различаются не более чем на 1. Так как операции вставки и удаления могут нарушить сбалансированность дерева, то необходимо в таких случаях производить процедуру балансировки.

Базовый класс для таблиц с вычислимым входом (THashTable).

class THashTable : public TTable {

protected:

virtual unsigned long HashFunc(const Tkey key); // hash-функция

public:

THashTable() : TTable() {} //конструктор

};

В классе содержится хэш-функция, вычисляющая по ключу записи номер строки в структуре хранения. Класс для таблиц с вычислимым входом, использующий для разрешения коллизий открытое перемешивание (TArrayHash).

#define TabMaxSize 25

#define TabHashStep 5

class TArrayHash : public THashTable {

protected:

PTTabRecord \*pRecs; // память для записей таблицы

int TabSize; // макс. возм. к-во записей

int HashStep; // шаг вторичного перемешивания

int FreePos; // первая свободная строка, обнаруженная при поиске

int CurrPos; // строка памяти при завершении поиска

PTTabRecord pMark; // маркер для индикации строк с удаленными записями

int GetNextPos(int pos){return (pos+HashStep)%TabSize;};// откр. перем.

public:

TArrayHash(int Size= TabMaxSize, int Step= TabHashStep); // конструктор

~TArrayHash();

// информационные методы

virtual int IsFull ( ) const ; // заполнена?

// доступ

virtual TKey GetKey (void) const;

virtual PTDatValue GetValuePTR (void) const;

// основные методы

virtual PTDatValue FindRecord (TKey k); // найти запись

virtual void InsRecord (TKey k, PTDatValue pVal ); // вставить

virtual void DelRecord (TKey k); // удалить запись

// навигация

virtual int Reset (void); // установить на первую запись

virtual int IsTabEnded (void) const; // таблица завершена?

virtual int GoNext (void) ; // переход к следующей записи

//(=1 после применения для последней записи таблицы)

};

При открытом перемешивании добавление новых записей возможно при наличии свободных строк в таблице. Разрешение коллизий происходит путем вычисления нового адреса для записи, пока не будет найдена свободная строка. Поэтому при удалении строка таблицы помечается как удаленная без фактического удаления и освобождения памяти, но сохраняется возможность записи в неё нового значения при необходимости.

Класс для таблиц с вычислимым входом, использующий для разрешения коллизий метод цепочек (TListHash).

#define TabMaxSize 25

class TListHash : public THashTable {

protected:

PTDatList \*pList; // память для массива указателей на списки записей

int TabSize; // размер массива указателей

int CurrList; // список, в котором выполнялся поиск

public:

TListHash(int Size= TabMaxSize); // конструктор

~TListHash();

// информационные методы

virtual int IsFull ( ) const ; // заполнена?

// доступ

virtual TKey GetKey (void) const;

virtual PTDatValue GetValuePTR (void) const;

// основные методы

virtual PTDatValue FindRecord (TKey k); // найти запись

virtual void InsRecord (TKey k, PTDatValue pVal ); // вставить

virtual void DelRecord (TKey k); // удалить запись

// навигация

virtual int Reset (void); // установить на первую запись

virtual int IsTabEnded (void) const; // таблица завершена?

virtual int GoNext (void) ; // переход к следующей записи

//(=1 после применения для последней записи таблицы)

};

В методе цепочек добавление новых записей в случае возникновения коллизии происходит путем добавления значения в список, в связи с чем необходимо использовать ранее разработанный класс для списков (TDatList).

# Результаты тестирования

Рисунок 1. График эффектиновсти операци вставки

Рисунок 2. График эффективности операции поиска

Рисунок 3. График эффективности операции удаления

# Выводы

В результате проделанной работы были реализованы таблицы с различными методами хранения данных. Каждая из них была протестирована на эффективность операций вставки, поиска и удаления записей, была собрана необходимая статистика и построены графики эффективности таблиц, определен наиболее эффективный тип таблиц – хеш-таблицы.

В работе активно использовались связный список и сбалансированное дерево – две структуры данных, позволяющие эффективно хранить информацию с возможностью быстрого ее поиска.

# Список источников

1. С/С++. Программирование на языке высокого уровня/ Т.А. Павловская, – СПБ.:Питер, 2003. –461 с: ил. ISBN 5-94723-568-4
2. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002.

— 896 с.: ил. — Парал. тит. англ. ISBN 5-8459-0262-2

1. Johnson M. Superscalar Microprocessor Design. — Englewood Cliff, New Jersey: Prentice Hall, 1991 ISBN-13 978-0138756345 ISBN-10 0138756341
2. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил. ISBN 5-94157-232-8
3. Stone H. High performance Computer Architecture. — Reading, MA: Addison-Wesley, 1993. ISBN-13 978-020151377