МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

01.04.02 Прикладная математика и информатика
(код и наименование направления подготовки)

Математическое моделирование
(направленность (профиль))

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему **Исследование методов реализации квантовых алгоритмов для решения задачи поиска данных**

Студент Г.Ю. Березовский _____
(И.О. Фамилия) (личная подпись)

Научный руководитель О.М. Гущина _____
(И.О. Фамилия) (личная подпись)

Руководитель программы _д.ф.-м.н. доцент, С.В. Талалов_ _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский_____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Тольятти 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 4
ГЛАВА 1 АНАЛИЗ АЛГОРИТМОВ ПОИСКА ДАННЫХ И ПОСТАНОВКА ЗАДАЧИ ИССЛЕДОВАНИЯ 9
1.1 Классические алгоритмы поиска данных 9
1.1.1 Последовательный поиск 9
1.1.2 Бинарный поиск 11
1.1.2 Интерполяционный поиск 12
1.1.4 Поиск на основе Хеша 13
1.2 Квантовые алгоритмы поиска данных 16
1.2.1 Алгоритм Гровера 17
1.2.2 Алгоритмы квантового поиска 18
1.3 Сравнительный анализ классических и квантовых алгоритмов 20
ГЛАВА 2 ОПИСАНИЕ МАТЕМАТИЧЕСКОЙ МОДЕЛИ АЛГОРИТМА ГРОВЕРА 23
2.1Классические вычисления 23
2.2 Квантовые вычисления 25
2.2.1 Квантовые состояния и кубиты 26
2.2.2 Несколько кубит 33
2.2.3 Гейты и квантовые схемы 36
2.2.4 Квантовая схемотехника 42
2.2.5 Принципы квантовых вычислений 45
2.2.6 Общая архитектура квантового компьютера 49
2.3 Математическая модель алгоритма Гровера 51
ГЛАВА 3 РЕАЛИЗАЦИЯ КВАНТОВОГО АЛГОРИТМА 57
3.1 Язык программирования Q# 57
3.2 Язык программирования Haskell 63
3.2.1 Строгая типизация Haskell 64
3.2.2 Программное обеспечение Haskell 65

3.3 Язык программирования Quipper 66
3.4 Описание функциональности компьютерной модели алгоритма Гровера. 74
ГЛАВА 4 РЕЗУЛЬТАТЫ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ 80
4.1 Результаты расчета компьютерной модели алгоритма Гровера 80
4.2 Сравнительный анализ компьютерной модели на Q# и классической компьютерной модели на языке C++ 82
ЗАКЛЮЧЕНИЕ 84
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 86
ПРИЛОЖЕНИЕ А 92
ПРИЛОЖЕНИЕ Б 95

ВВЕДЕНИЕ

В последние годы было проведено значительное количество исследований по квантовым компьютерам – машинам, которые используют квантово-механические явления для решения математических задач, которые трудны или трудноразрешимы для обычных компьютеров. Такой вывод можно сделать на основании:

1. В 2016 году продажу поступает квантовый компьютер D-Wave2000Q с процессором на 2000 кубит по цене 15 000 000 долларов [0].

2. В 2013 году компания Google купила квантовый компьютер у компании D-Wave [59].

3. Компания D-Wave, которая занимается постройкой квантовых вычислительных систем, получила заказ на установку своего нового устройства D-Wave 2X, которое содержит более 1 тысячи кубитов, в лаборатории квантового искусственного интеллекта, совместной лаборатории НАСА и Google, говорится в пресс-релизе компании. Новая установка будет использоваться для исследований в области машинного обучения, искусственного интеллекта и в других областях [59].

4. Группа исследователей компании Google под руководством Джона Мартиниса уже близка к созданию первого универсального квантового компьютера, который, возможно, сможет превзойти классические суперкомпьютеры. Как пишет MIT Technology Review, ученые, знакомые с ходом работы лаборатории Мартиниса, уверены, что уже в 2017 году он может представить квантовый компьютер с чипом на 50 кубитов [50].

5. Ученые Российского квантового центра, МФТИ (Лаборатория искусственных квантовых систем при участии Технологического центра), создали первый в России сверхпроводящий кубит - основной элемент будущих квантовых компьютеров, которые смогут превзойти самые мощные современные суперкомпьютеры. Это важный шаг, необходимый для создания

квантовых вычислительных устройств, которые в будущем произведут революцию в области вычислительной техники [39].

6. Правительство Нидерландов приняло решение вложить 135 миллионов евро в разработку технологий квантовых вычислений, говорится в сообщении на сайте голландского кабинета министров [48].

7. Компания D-Wave нуждается в том, чтобы привлечь больше программистов к созданию приложений для своих машин. Для этого специалисты разработали новый программный инструмент Qbsolv, который позволит разрабатывать программы для машин D-Wave, не вникая в квантово-механические тонкости. Этот инструмент позволит создать сообщество «квантовых» программистов, чтобы расширить потенциальную базу для этого сообщества, код программы выложен в открытый доступ, и все желающие смогут модифицировать его «под себя» [49].

8. IBM позволяет заинтересованным людям получить доступ к 5-кубитному квантовому компьютеру под названием IBM Quantum Experience. Оборудование размещено в исследовательской лаборатории IBM в штате Нью-Йорк. Компания предоставляет программный интерфейс и возможность запуска экспериментальной программы на реальном квантовом компьютере [50].

9. IBM сообщила о создании рабочего прототипа 50-кубитного квантового процессора. Это большой шаг вперед по сравнению с предыдущим достижением компании — 17-кубитным квантовым компьютером [43].

10. Компания Google построила квантовый процессор, в котором 72 сверхпроводниковых кубита объединены в двумерный массив. Этот процессор использует ту же технологию, что и предыдущий 9-кубитный квантовый компьютер, построенный компанией и имеющий низкий процент ошибок при вычислениях. Новую разработку компания представила на ежегодной встрече Американского физического сообщества в Лос-Анджелесе, кратко о ней сообщается в блоге компании [54].

На основе выше сказанного можно сделать вывод о том, что крупные IT компании и государственные организации вкладывают в квантовые вычисления большие ресурсы и данная тема исследования является актуальной.

Научная проблема заключается в том, что в большинство квантовых алгоритмов не реализованы в виде компьютерной модели.

Цель исследования: анализ и математическое моделирование квантовых алгоритмов, и программная реализация квантовых алгоритмов на языке Q#.

Объект исследования: задача поиска данных.

Предмет исследования: алгоритмы решения задачи поиска данных.

Для решения цели исследования сформулированы и решены следующие задачи:

- исследование классических алгоритмов поиска данных;
- исследование квантового Алгоритма Гровера;
- исследование алгоритмов квантового поиска;
- математическое моделирование рассматриваемых квантовых алгоритмов;
- сравнение квантовых алгоритмов с классическими;
- исследование языков программирования для реализации квантовых вычислений;
- разработка программы, реализующий квантовый алгоритм с помощью языка программирования Q#;
- отладка и параметрическое исследование предложенной программы.

Апробация работы была представлена на следующих конференциях:

- III Международная научно-практическая конференция (школе-семинаре) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук», Тольятти, 2017 г.

- IV Международная научно-практическая конференция (школе-семинаре) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук», Тольятти, 2018 г.

Научная новизна состоит в том, что квантовые алгоритмы реализованы на языке Q#.

Теоретические основы исследования включают использование трудов отечественных и зарубежных авторов по исследованию классических алгоритмов поиска данных и квантовых алгоритмов поиска данных.

Теоретическая значимость заключается в описании квантовых алгоритмов поиска.

Практическая значимость состоит в том, что были подобраны наилучшие алгоритмы, решающие задачу поиска данных, реализованы на языке программирования, что позволит при разработке квантового компьютера запустить на нем данную программу.

Достоверность и обоснованность научных положений и выводов, сделанных в магистерской диссертации, следует из адекватности математических моделей, используемых в работе, подтверждается сравнением результатов данной работы с известными экспериментальными итеоретическими данными, полученными ранее другими авторами.

Основные положения, выносимые на защиту:

1. Превосходство квантовых алгоритмов над классическими алгоритмами.
2. Компьютерная модель квантового алгоритма Гровера на языке Q#.

Диссертация состоит из введения, четырех глав, заключения, списка литературы и приложения. Объем диссертации составляет 90 страниц и содержит 22 рисунка, список литературы включает 59 наименование источников отечественных и зарубежных авторов.

В первой главе диссертации описываются классические и квантовые алгоритмы поиска сортированных и несортированных данных, проводится

сравнительный анализ алгоритмов и выполняется постановка задачи исследования.

Во второй главе проводится описание квантовых вычислений и описывается построение математической модели.

В третьей главе описывается язык программирования, используемый для квантовых вычислений, программное обеспечение которое нужно для использования данного языка и описываются функции компьютерной модели. Программный продукт состоит из тела программы и функций, реализующих алгоритмы: математической модели, функцию черного ящика, а также унитарные операции. Проводится результат компьютерного моделирования трехкубитного квантового компьютера при корректных и некорректных данных.

В четвертой главе описываются результаты компьютерной модели разработанной на языке программирования Q#. Проводится сравнительный анализ с классической компьютерной моделью на языке C++.

В заключении описаны выводы результатов работы программного продукта.

В приложение А находится листинг кода программного продукта.

ГЛАВА 1 АНАЛИЗ АЛГОРИТМОВ ПОИСКА ДАННЫХ И ПОСТАНОВКА ЗАДАЧИ ИССЛЕДОВАНИЯ

В данной главе речь идёт об анализе предметной области и производится постановка задачи исследования. Описываются классические алгоритмы поиска данных. Рассматриваются применение квантовых алгоритмов для задачи поиска данных, производится сравнительный анализ квантовых алгоритмов с классическими алгоритмами.

При написании главы использовались материалы из [7,8,9,11,15,16,18 и др.].

1.1 Классические алгоритмы поиска данных

Алгоритмы поиска занимают очень важное место среди прикладных алгоритмов, и это утверждение не нуждается в доказательстве. Все алгоритмы поиска разбиваются на две большие группы в зависимости от того, упорядочен или нет массив данных, в котором проводится поиск. Рассмотрим простые алгоритмы поиска заданного элемента в одномерном массиве данных [10].

1.1.1 Последовательный поиск

Наиболее примитивный, а значит, наименее эффективный способ поиска — это обычный последовательный просмотр массива.

Пусть требуется найти элемент X в массиве из n элементов. Значение элемента X вводится с клавиатуры.

В данном случае известно только значение разыскиваемого элемента, никакой дополнительной информации о нем или о массиве, в котором его надо искать, нет. Поэтому для решения задачи разумно применить очевидный метод — последовательный перебор элементов массива и сравнение значения очередного элемента с заданным образцом. Пусть $Flag = 1$, если значение, равное X, в массиве найдено, в противном случае $Flag = 0$. Обозначим k — индекс найденного элемента. Элементы массива — целого типа.

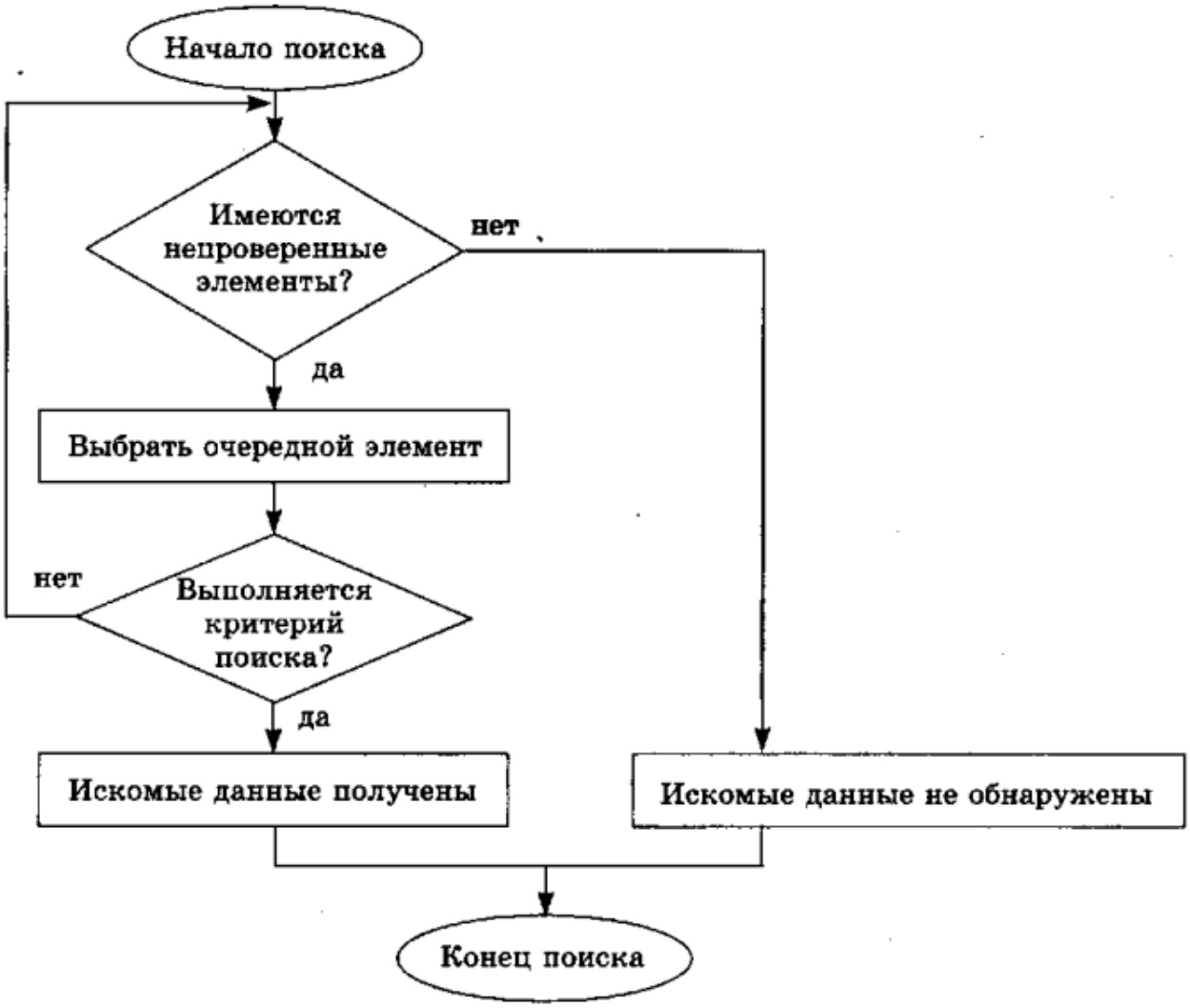

Рисунок 1.1 – Блок схема алгоритма последовательного поиска

Присвоим начальные значения переменным *Flag* и *k* и возьмем первый элемент массива (блок 1).

Пока не просмотрены все элементы и пока элемент, равный X, не найден (выход «Да» блока 2), выполняем цикл.

Сравниваем очередной элемент массива со значением X, и если они равны (выход «Да» блока 3), то запоминаем его индекс и поднимаем Flag = 1. Если элемент еще не найден (выход «Нет» блока 2), продолжаем просмотр элементов массива.

Выход из цикла (блок 2) возможен в двух случаях:

- элемент найден, тогда $Flag = 1$, а в переменной k сохранен индекс найденного элемента;
- элемент не найден, тогда $Flag = 0$, а k также равен 0.

Исследуя данные в этой главе можно сказать, что временная сложность алгоритма последовательного поиска есть $O(n)$. Временная сложность алгоритма на прямую связана с количеством элементов данных.

1.1.2 Бинарный поиск

Если известно, что массив упорядочен, то можно использовать бинарный поиск.

Пусть даны целое число X и массив размера n, отсортированный в порядке не убывания, т. е. для любого k $(1 <= k < n)$выполняется условие $a[k - 1] <= a[k]$. Требуется найти такоеi, что $a[i] = X$, или сообщить, что элемента X нет в массиве. Идея бинарного поиска состоит в том, чтобы проверить, является ли X средним по положению элементом массива. Если да, то ответ получен. Если нет, то возможны два случая:

1. X меньше среднего по положению элемента, следовательно, в силу упорядоченности массива можно исключить из рассмотрения все элементы массива, расположенные правее этого элемента, так как они заведомо больше его, который, в свою очередь, больше X, и применить этот метод к левой половине массива [12].
2. X больше среднего по положению элемента, следовательно, рассуждая аналогично, можно исключить из рассмотрения левую половину массива и применить этот метод к его правой части [12].

Средний по положению элемент и в том, и в другом случае в дальнейшем не рассматривается. Таким образом, на каждом шаге отсекается та часть массива, где заведомо не может быть обнаружен элемент X.

Приведем блок схему алгоритма:

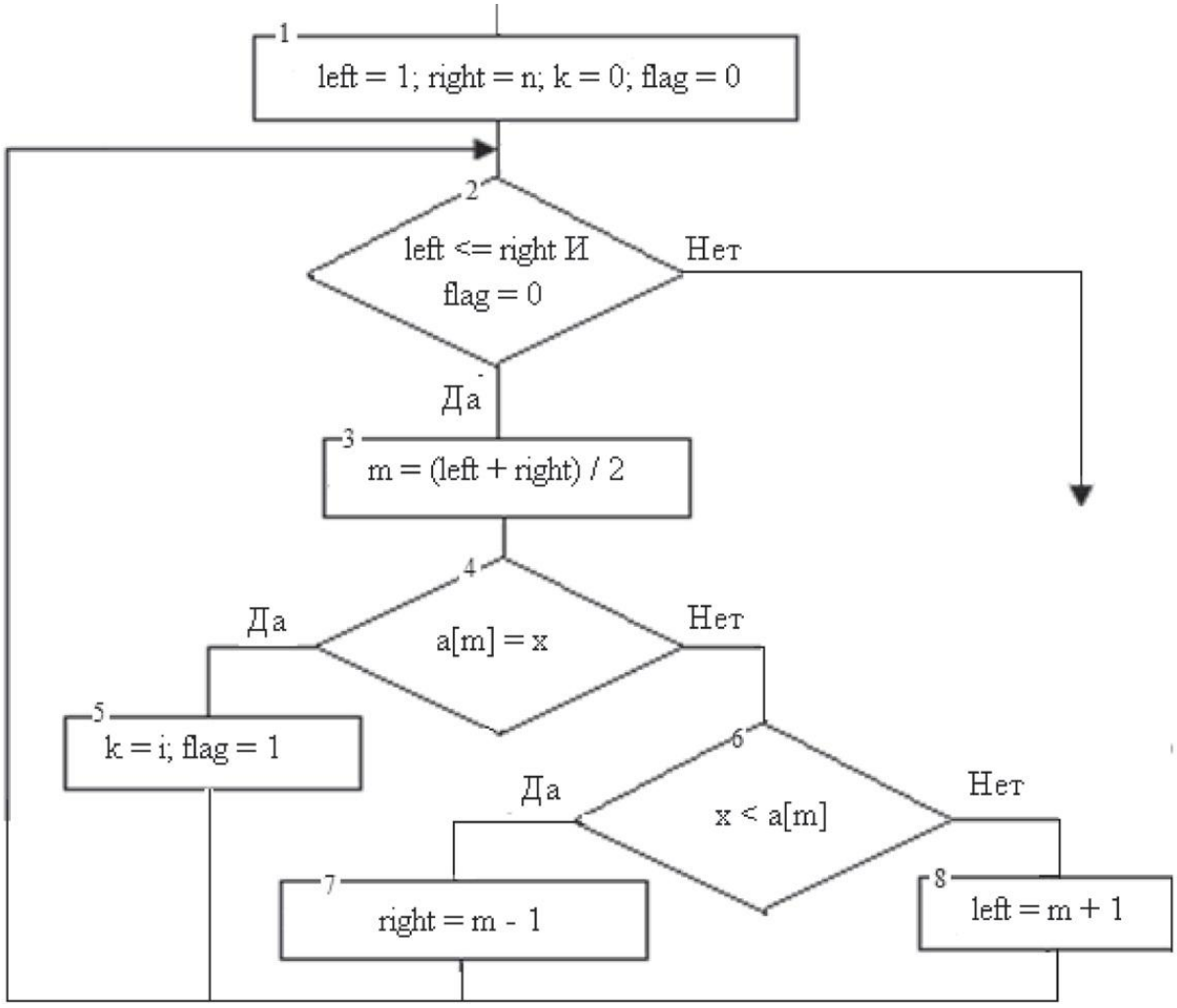

Рисунок 1.2 – Блок схема алгоритма бинарного поиска

Бинарный поиск требует $\log_2 N$запросов для решения задачи, то есть данный алгоритм имеет логарифмическую сложность $O(\log N)$

Исследовав литературу [10,12,13] , можно сделать вывод что алгоритм бинарного поиска является одним из самых быстрых классических алгоритмов поиска сортированных данных.

1.1.2 Интерполяционный поиск

Бинарный поиск каждый раз ищет целевой элемент массива в середине раздела, интерполяционный пытается ускорить процесс — он старается угадать расположение целевого элемента в массиве по его значению. Предположим, в массиве содержится 1000 элементов со значениями от 1 до 100. Если наша цель

— найти число 30, то его нужно искать в районе первой трети массива, где-то рядом с индексом 300. Общее распределение чисел не всегда позволяет получить результат с прицельной точностью, но он может оказаться довольно близким[21].

Формула, определяющая алгоритм интерполяционного поиска, выглядит следующим образом:

$$mid = left + \frac{(key - A\ left\ ) * (right - left)}{A\ right\ - A[left]}$$

где mid — номер элемента, с которым сравнивает значение ключа,

key — искомый ключ,

A — массив упорядоченных элементов,

$left$ и $right$— номера крайних элементов области поиска.

Если данные распределены очень неравномерно, и вы ищете наихудшее целевое значение, данный алгоритм обладает производительностью O(N). Если распределение относительно равномерное, ожидаемая производительность составит $O\ log(\log N)$[21].

Исследовав литературу [21,14] можно сказать что данный алгоритм используется лишь на очень больших таблицах, причем делается несколько шагов интерполяционного поиска, а затем на малом подмассиве используется бинарный или последовательный варианты.

1.1.4 Поиск на основе Хеша

В предыдущих разделах по поиску описаны алгоритмы, пригодные при небольшом количестве элементов (последовательный поиск) или в упорядоченной коллекции (бинарный поиск). Но нам нужны более мощные методы поиска в больших коллекциях, которые не обязательно упорядочены. Одним из наиболее распространенных подходов к решению этой задачи является использование хеш-функции для преобразования одной или нескольких характеристик искомого элемента в индекс в хеш-таблице. Поиск

на основе хеша (Hash-BasedSearch) имеет в среднем случае производительность, которая лучше производительности любого другого алгоритма поиска, описанного в этой главе. Во многих книгах об алгоритмах рассматривается поиск на основе хеша в теме, посвященной хеш-таблицам. Вы также можете найти эту тему в книгах о структурах данных, в которых рассматриваются хеш-таблицы[12].

В поиске на основе хеша *n* элементов коллекции *C* сначала загружаются в хеш-таблицу H с b ячейками (“корзинами”), структурированными в виде массива. Этот шаг предварительной обработки имеет производительность *O(n)*, но улучшает производительность будущих поисков с использованием концепции хеш-функции.

Хеш-функция представляет собой детерминированную функцию, которая отображает каждый элемент *C* на целочисленное значение h. На минуту предположим, что $0 < h < b$. При загрузке элементов в хеш-таблицу элемент *C* вставляется в ячейку *H[h]*. После того как все элементы будут вставлены, поиск элемента *t* становится поиском *t* в *H[hash(t)]* [12].

Хеш-функция гарантирует только, что если два элемента, C_i и C_j равны, то$hash(C_i) = hash(C_j)$. Может случиться так, что несколько элементов в *C* имеют одинаковые значения хеша. Такая ситуация называется коллизией, и хеш-таблице необходима стратегия для урегулирования подобных ситуаций. Наиболее распространенным решением является хранение в каждой ячейке связанного списка (несмотря на то, что многие из этих связанных списков будут содержать только один элемент). При этом в хеш-таблице могут храниться все элементы, вызывающие коллизии. Поиск в связанных списках должен выполняться линейно, но он будет быстрым, потому что каждый из них может хранить максимум несколько элементов. Приведенный далее псевдокод описывает решение коллизий с использованием связанного списка. Общая схема поиска на основе хеша показана на рис. 1.3 (здесь рассматривается небольшой пример ее применения). Ее компонентами являются:

- множество U которое определяет множество возможных хеш-значений. Каждый элемент $e \in C$ отображается на хеш-значение $h \in U$;
- хеш-таблица H, содержащая b ячеек, в которых хранятся n элементов из исходной коллекции C;
- хеш-функция $hash$, которая вычисляет целочисленное значение h для каждого элемента e, где $0 < h < b$.

Эта информация хранится в памяти с использованием массивов и связанных списков.

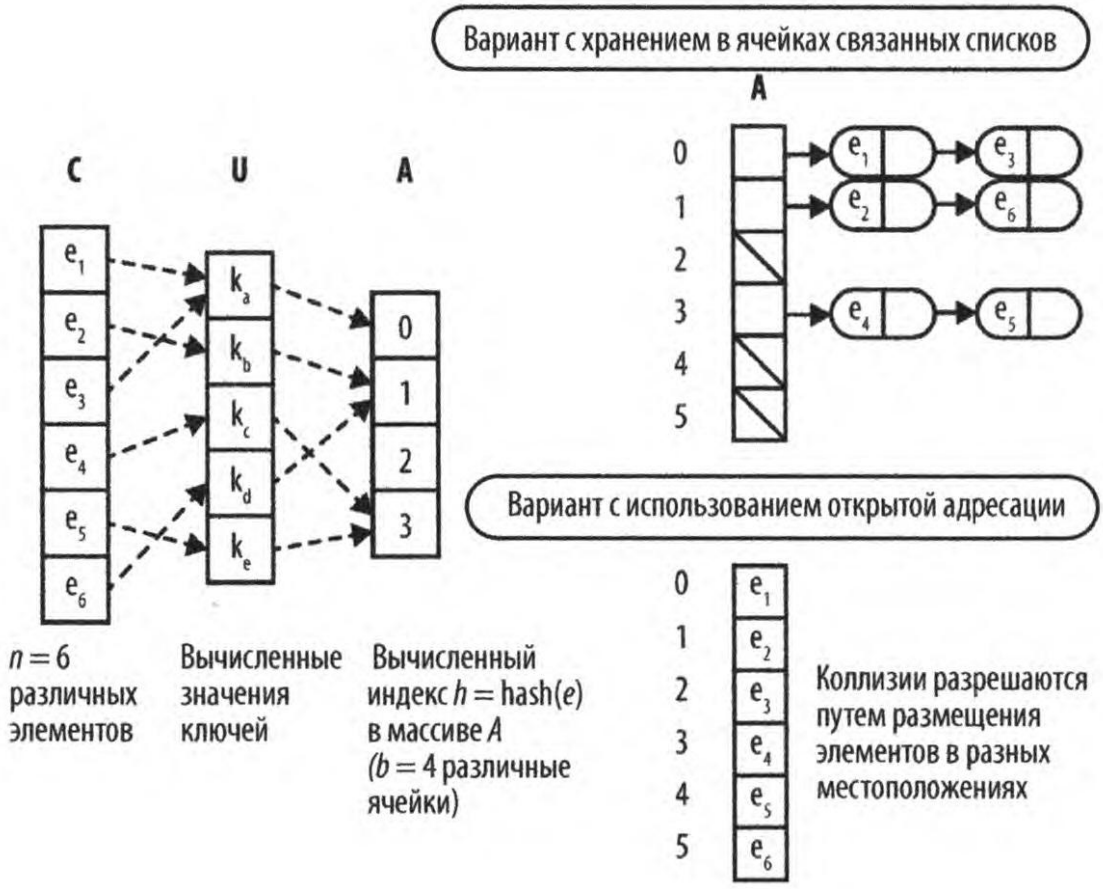

Рисунок 1.3 – Общая схема хеширования

У поиска на основе хеша имеются две основные проблемы: разработка хеш-функции и обработка коллизий. Плохо выбранная хеш-функция может привести к плохому распределению ключей в первичной памяти с двумя последствиями:

- многие ячейки в хеш-таблице могут остаться неиспользованными и зря тратить память;
- будет иметься много коллизий, когда много ключей попадают в одну и ту же ячейку, что существенно ухудшает производительность поиска [12].

Приведем блок схему алгоритма в основе которого лежит хеш-функция:

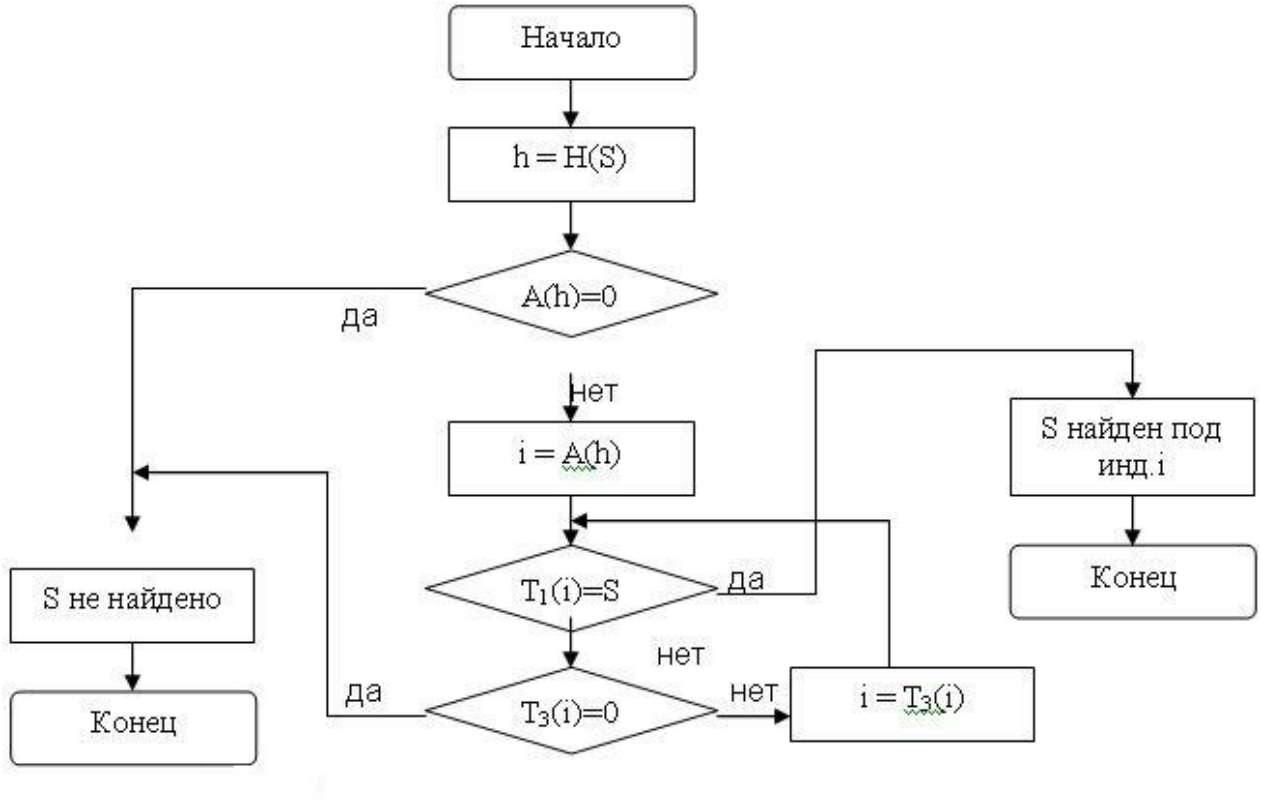

Рисунок 1.4 – Блок схема алгоритма в основе которого лежит хеш-функция

Исследовав литературу [11,18] можно сделать вывод что алгоритмы на основе хеша имеют хорошую временную сложность (такую же как бинарный поиск), но при больших данных появляются проблемы с хеш-таблицой.

1.2 Квантовые алгоритмы поиска данных

В наше время можно наблюдать связь между информатикой и физикой. Оказалось, что эффективность решения задач напрямую зависит от законов

физики. К примеру, для вычислительных устройств, основанных на квантовых законах (квантовых компьютерах) существуют алгоритмы которые решают задачи гораздо эффективнее чем все известные алгоритмы для компьютеров. В подпунктах данной главы рассмотрим квантовые алгоритмы поиска данных.

1.2.1 Алгоритм Гровера

Алгоритм Гровера решает задачу неструктурированного поиска. Если есть неупорядоченный набор данных и требуется найти в нём какой-то один элемент, удовлетворяющий специфическому требованию. Этот алгоритм использует свойство квантовой интерференции для того, чтобы найти значения некоторого параметра, на котором заданная функция выдаёт определённый результат [8].

Пусть дана функция — булева функция. Цель: найти хотя бы один корень уравнения $f(x) = 1$. На классическом компьютере, если f — произвольна, нам понадобиться $O(N)$операций,то есть, полный перебор. Если f в конъюнктивой нормальной форме — то данная задача является NP-полной.

Алгоритм Гровера состоит из следующих шагов:

1. *Инициализация начального состояния*. Необходимо подготовить равновероятностную суперпозицию состояний всех входных кубитов. Это делается при помощи применения соответствующего гейта Адамара, который равен тензорному произведению n унарных гейтов Адамара друг на друга [8].

2. *Применение итерации Гровера*. Данная итерация состоит из последовательного применения двух гейтов — оракула и так называемого гейта диффузии Гровера (будут детально рассмотрены ниже). Эта итерация осуществляется $\overline{2^n}$раз[8].

3. *Измерение.* После применения итерации Гровера достаточное количество раз необходимо измерить входной регистр кубитов. С очень высокой вероятностью измеренное значение даст указание на искомый параметр. Если необходимо увеличить достоверность ответа, то алгоритм

прогоняется несколько раз и вычисляется совокупная вероятность правильного ответа.

Схема алгоритма Гровера представлена на рисунке 1.4:

Рисунок 1.5 - Общая диаграмма квантовой схемы алгоритма Гровера

Алгоритм Гровера позволяет достичь квадратичное ускорение в сравнении с классическими алгоритмами перебора – за$O(\ \overline{n})$[8].

На основе выше сказанного можно сделать вывод что алгоритм Гровера дает квадратичное ускорение по сравнению с классическими алгоритмами.

1.2.2 Алгоритмы квантового поиска

Одним из важных аспектов в структурированной информации является поиск. Задача поиска очень актуальна. Один из самых эффективных классических алгоритмов решающий эту задачу является Бинарный поиск.

Бинарный поиск - классический алгоритм поиска элемента в отсортированном массиве, использующий дробление массива на половину. Бинарный поиск требует $\log_2 N$запросов для решения задачи, то есть данный алгоритм имеет логарифмическую сложность $O(\log N)$.

Существуют квантовые алгоритмы, которые решают данную задачу:

1. Алгоритм Фархи и его коллег, для решения задачи использует $0.53\log_2 N$запросов [51].
2. Инвариантные квантовые алгоритмы, для решения задачи используется $0.433\log_2 N$запросов[51].
3. Вероятностный квантовый алгоритм, ожидаемая эффективность которого

находится на уровне$\frac{1}{3}\log_2 N$запросов [57].

Доказано, что нижней границей эффективности квантового алгоритма является значение$\frac{\pi}{4}\log_2 N$ [51].

Результаты вычисления алгоритма Фархии его коллег.

Алгоритм Фархи находит решение уравнения

$$f\ x\ = 1 \tag{1.1}$$

где *f* есть булевая функция от *n* переменных.

При дополнительном условии

$$g\ x_1\ = 1 \tag{1.2}$$

где $x\ = x_1 x_2$ разбиение строки *x*на две строки одинаковой длины.

Задача: пусть дан список из *N* чисел, упорядоченных от меньшего к большему. Для заданного числа *x* необходимо определить его позицию в списке. Наилучший классический алгоритм требует $\log_2 N$ запросов, к примеру алгоритм бинарного поиска. Э. Фархи со своими коллегами показал, что на квантовом компьютере требуется всего лишь $0.53\log_2 N$запросов. В источнике [21]описываются теоретические результаты из исследования Э.Фархи и его коллег.

Таблица 1.1 - Результаты вычисления квантового и классического алгоритма

N	**k=1**	**2**	**3**	**4**	**5**	**6**	**Классический алгоритм**
64	0.2036	0.6495	0.9615	0.9997	1.000	1.000	Дает точный результат при k =5
256	0.0788	0.3886	0.8221	0.9907	0.9999	1.000	7
1024	0.0282	0.2000	0.5981	0.9324	0.9983	1.000	10
2048	0.0165	0.1374	0.4818	0.8690	0.9939	0.9997	11
4096	0.0096	0.0924	0.3755	0.7834	0.9819	0.9992	12

где N-количество упорядоченных элементов;

k – количество вхождений в функцию.

Как видно в таблице, квантовый алгоритм дает правильный ответ с определенной вероятностью, для повышения точности увеличивают количество вхождений.

Исследовав литературу [50,54] можно сделать вывод, что квантовые алгоритмы в задачи поиска элемента в упорядоченном списке дают константный выигрыш по сравнению с классическими.

1.3 Сравнительный анализ классических и квантовых алгоритмов

Произведем сравнительный анализ алгоритмов.

Начнем с отсортированных данных, изображенных на рисунке 1.6:

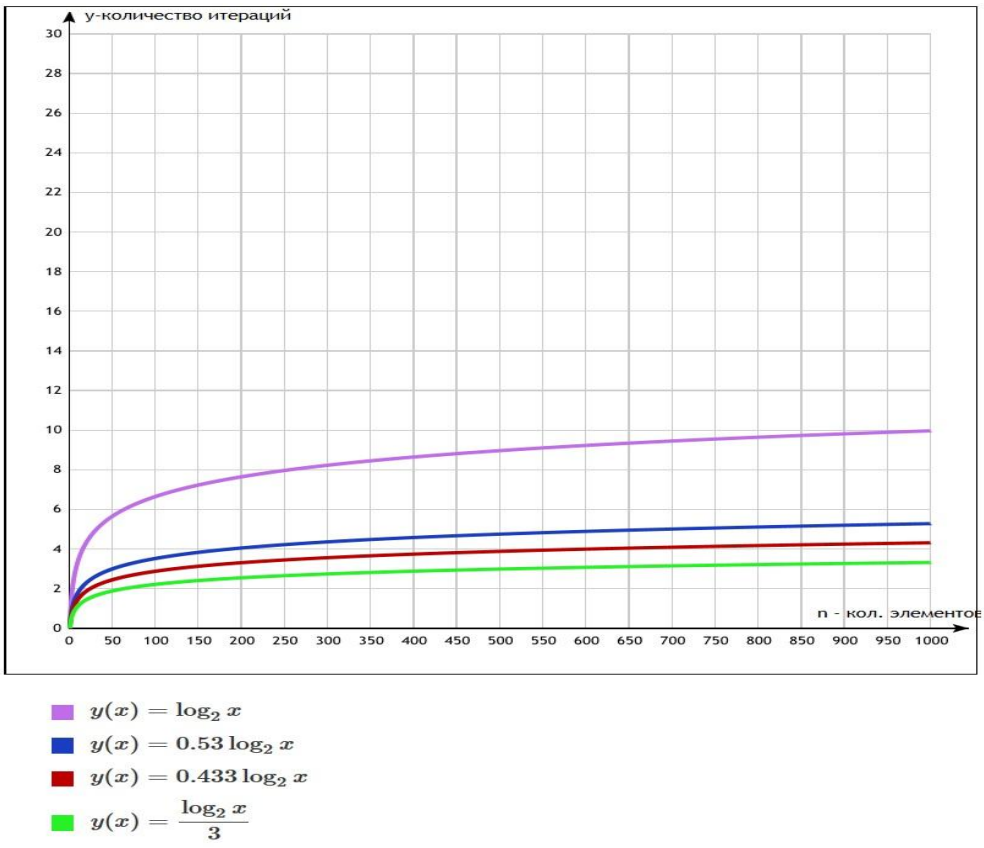

Рисунок 1.6 - Сложность алгоритмов в отсортированном массиве

На рисунке 1.6 изображена зависимость количество итераций алгоритмов от количества элементов. Данный рисунок показывает за сколько итераций

алгоритм найдет элемент в отсортированном массиве данных. На рисунке изображены такие сложностиO алгоритмов:

- $y(x) = log_2 x$ сложность бинарного алгоритма поиска;
- $y(x) = 0.53log_2 x$ сложность алгоритма Фархи и его коллег;
- $y(x) = 0.433log_2 x$ сложность инвариантных квантовых алгоритмов
- $y(x) = 0.53log_2 x$ сложность вероятностного квантового алгоритма

поиска.

Квантовые алгоритмы в задачи поиска элемента в упорядоченном списке дают константный выигрыш по сравнению с классическими.

Рассмотрим данные в не сортированном массиве, изображенные на рисунке 1.7:

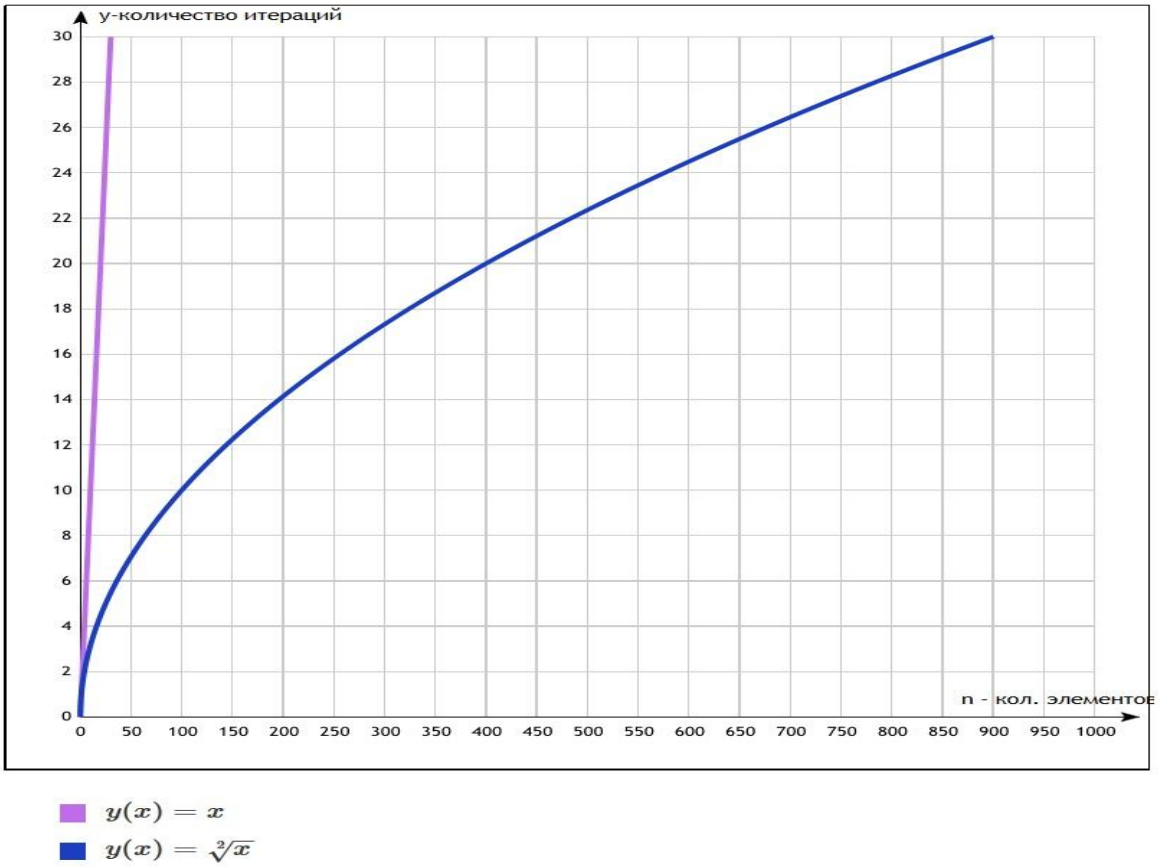

Рисунок 1.7 - Сложность алгоритмов в неотсортированном массиве

На рисунке 1.7 изображены:

- $y(x) = x$ – сложность алгоритма последовательного поиска;
- $y\ x\ =\ \overline{x}$ – сложность алгоритма Гровера.

Алгоритм Гровера позволяет достичь квадратичное ускорение в сравнении с классическими алгоритмами перебора.

Исследовав данную главу можно сделать вывод, что квантовые алгоритмы в задачи поиска элемента в сортированном списке при больших данных дают константный выигрыш по сравнению с классическими, а в несортированном списке квадратичное ускорение.

ГЛАВА 2 ОПИСАНИЕ МАТЕМАТИЧЕСКОЙ МОДЕЛИ АЛГОРИТМА ГРОВЕРА

В данной главе производится постановка задачи математической модели квантового алгоритма Гровера. Описываются модель квантовых вычислений, а также архитектура квантового компьютера.

При написании главы использовались материалы из [8,54,9,59,36 и др.].

2.1 Классические вычисления

Начнем со слова вычисление. Существует довольно много дополняющих друг друга определений этого понятия. Для наших целей подойдет самое общее определение, выделяющее важные для нас характеристики вычислений[9].

Вычисление - это конечный по времени физический процесс с фиксированным (не обязательно конечным) набором состояний, каждое из которых может быть описано в некоторой кодировке [9].

Выбор набора состояний и определение способа их описания зависят целиком от пользователя процесса. На одном и том же физическом явлении вычисления можно организовать по-разному.

Каждое состояние несет некоторую информацию (в заданной кодировке). Начальное состояние определяет входные данные процесса, конечное выходные. Это позволит нам определить информацию, как описание состояния некоторой физической системы.

Информация - это описание в некоторой кодировке состояния физической системы (не обязательно выполняющей вычисления)[9].

Подобный материалистический подход позволит нам пойти еще дальше и определить такие понятия как количество информации и количество вычислений.

Количество вычислений - количество смен состояний вычислительного процесса[9].

Для определения количества информации мы воспользуемся формулой Шеннона, предложенной американским криптоаналитиком Клодом Шенноном в 1948 году. Формула Шеннона определяет информационную емкость системы, имеющей *n* состояний, для каждого из которых определена его вероятность P_i.

$$I = - \sum_{i=1}^{n} P_i \log_b(P_i) \quad (2.1)$$

Если мы возьмем логарифм по основанию 2 ($b = 2$), то информационная емкость по формуле будет рассчитываться в битах. Бит минимальная частичка информации, еще одно понятие, подаренное миру Клодом Шенноном. Для системы, имеющей *n* равновероятных состояний, количество информации в битах по формуле Шеннона выглядит проще:

$$I = \sum_{i=1}^{n} \frac{1}{n} lg \frac{1}{n} = \lg(n) \quad (2.2)$$

Например, для простейшего переключателя (рис 2.1), имеющего два равновероятных состояния, мы получаем по формуле ($I = lg2 = 1$) информационную емкость 1 бит. Получается, что такой переключатель может хранить (содержать) один бит информации.

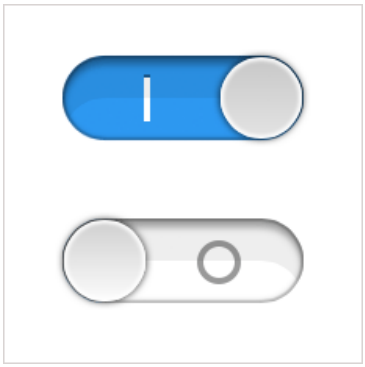

Рисунок 2.1 – Переключатель с двумя состояниями

Исследовав литературу [38,37,36] можно сделать вывод что классические вычисления не используют всех возможностей , предоставляемой природой , и на помощь приходят квантовые вычисления.

2.2 Квантовые вычисления

В области квантовых вычислений и квантовой информации изучаются задачи по обработке информации, которые могут быть выполнены с использованием квантовомеханических систем. Чтобы понять, почему это произошло, мы должны вернуться в прошлое и последовательно рассмотреть каждую из областей, внесших фундаментальный вклад в квантовые вычисления и квантовую информацию, квантовую механику, информатику, теорию информации и криптографию.

В 1982 году реализовали вычислительный процесс на базе квантовой системы. Таким образом, мы смогли бы с пользой эксплуатировать "вредные" квантовые эффекты, и получить вычислители, мощность которых растет экспоненциально с ростом количества базовых элементов [37].

Уже через три года (1985 г.) англичанин Дэвид Дойч предложил математическую модель квантовых вычислений и рассмотрел простую задачу (известную как "задача Дойча"), для которой он продемонстрировал преимущества квантовых вычислений над классическими [37].

За этой работой последовало несколько довольно экзотических результатов и, наконец, в 1994-м году был опубликован алгоритм Питера Шора, показавший, что некоторые пока не разрешимые в практическом смысле задачи могут иметь эффективное решение на квантовом компьютере. В 1996-м году Лов Гровер предложил квантовый алгоритм, решающий любую задачу из класса NP в общем виде с квадратичным по сравнению с классической моделью ускорением.

Преимущества квантовых вычислений над классическими стали очевидны сразу по двум направлениям с точки зрения миниатюризации базового элемента вычислительного процесса, и с точки зрения теоретических преимуществ модели [37].

Квантовые вычисления и квантовая информация научила нас думать о вычислениях физически, данных подход открывает много новых возможностей

в области связи и обработки информации. Было выяснено что любая физическая теория, не только квантовая механика, может служить базисом для теории обработки информации. Специалисты по информатике и теории информации получили новую плодотворную парадигму для исследований. В результате таких исследовании создаются устройства обработки информации, намного превосходящие системы [37].

Исследовав литературу [37] можно сделать вывод, что квантовые вычисления и квантовая информация предоставляет новые инструменты, позволяющие перебрасывать «мост» от простого к сложному: в сфере вычислений и алгоритмов.

2.2.1 Квантовые состояния и кубиты

Прежде всего, для понимания модели квантовых вычислений необходимо овладеть понятийным аппаратом, который используется для описания и работы. Вся терминология пришла прямиком из квантовой механики.

Перед рассмотрением основного понятия в модели квантовых вычислений — кубита, необходимо изучить понятие квантового состояния. Квантовым состоянием будем называть совокупность из некоторого символа (наименования квантового состояния) и приписанного к нему коэффициента, причём этот коэффициент является комплексным числом. Квантовое состояние будет записываться как:

$$\mathbf{a}|\mathbf{s} \tag{2.3}$$

где a– комплексный коэффициент,

s -наименование квантового состояния.

Последнее обычно состоит из одного символа, например: «0», «1», «+», «-». Так что, квантовыми состояниями, например, являются такие объекты, как $|0\rangle$ и $|1\rangle$.

Кубитом, в этом случае, называется просто список квантовых состояний. Кубит состоит из списка квантовых состояний, при этом есть одно ограничение

— сумма квадратов модулей всех комплексночисленных коэффициентов обязательно должна равняться 1.

Для понимания квантовых вычислений нужно знать понятия «базис». Базисом называется набор кубитов, которые взаимно ортогональны друг другу.

Для простоты понимания и соответствия традиционному пониманию бита базисом назван набор кубитов |0⟩ и |1⟩, после чего было введено векторное представление кубита. Для этих двух базисных кубитов векторное представление следующее:

$$|0 = \begin{matrix}1\\0\end{matrix}$$
$$|1 = \begin{matrix}0\\1\end{matrix} \quad (2.4)$$

Ортогональность в таком случае определяется как равенство нулю скалярного произведения двух векторных представлений кубитов.

0 и 1 в векторах это комплексно-численные коэффициенты α при базисных кубитах, то есть так и получается, что $|0\rangle = 1|0\rangle + 0|1\rangle$, а $|1\rangle = 0|0\rangle + 1|1\rangle$. Таким образом, произвольный кубит можно разложить в базисе, и такое разложение записывается как $|\varphi\rangle=\alpha|0\rangle+\beta|1\rangle$, и оно называется линейной суперпозицией базисных состояний, и, соответственно, в виде вектора такой кубит представляется как:

$$\begin{matrix}\alpha\\\beta\end{matrix} \quad (2.5)$$

где α и β — некоторые комплексные числа такие, что сумма квадратов их модулей равна строго 1.

Комплексный коэффициент при базисном кубите, или, что то же, при базисном квантовом состоянии, — это так называемая амплитуда вероятности. Это понятие из квантовой механики, и оно обозначает тот простой факт, что при измерении вероятность обнаружения кубита в этом квантовом состоянии равна квадрату модуля его амплитуды. Именно поэтому сумма квадратов

модулей должна равняться строго единице, поскольку при измерении кубит будет обнаружен либо в том, либо в другом базисном квантовом состоянии.

Необходимо отметить, что, как следует из положений квантовой механики, после измерения кубит переходит в какое-либо состояние из числа входящих в базис, в рамках которого производится измерение, при этом вероятность перехода кубита в это состояние равна как раз квадрату модуля амплитуды при этом состоянии в базисе.

Пусть коэффициенты перед квантовыми состояниями будут не комплексными, а действительными. В этом случае всё очень просто: каждый кубит представляет собой единичный вектор, а его разложение в базисе — это проекции данного вектора на произвольно выбранные ортогональные «оси»:

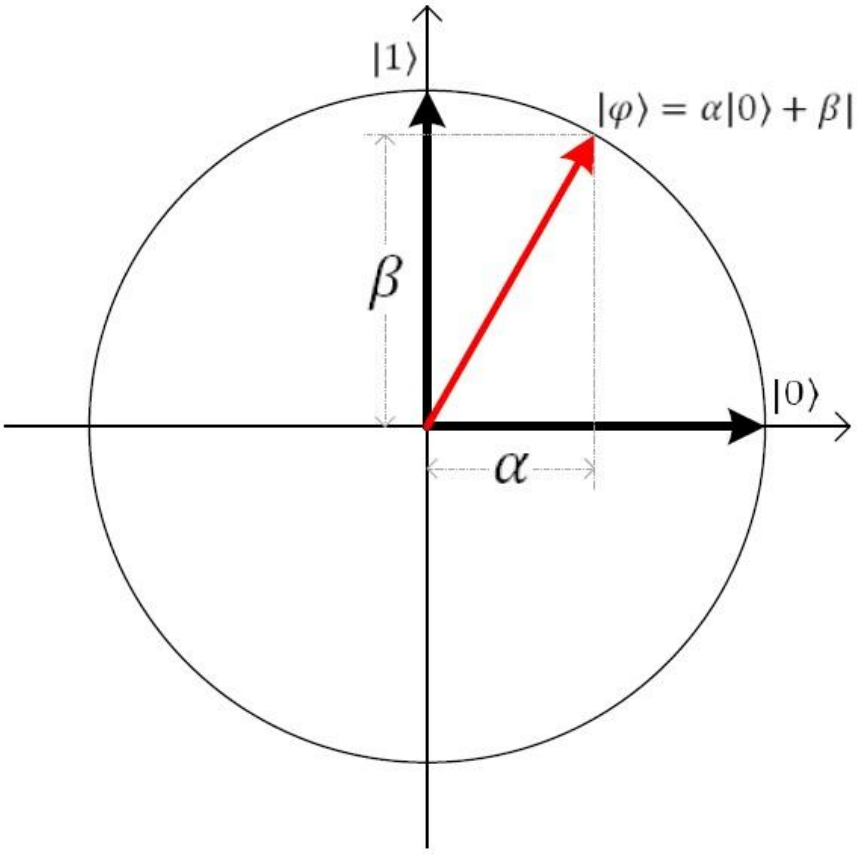

Рисунок 2.2 - Графическое представление разложения кубита в базисе в случае действительных коэффициентов

Тут так же необходимо отметить, что «оси», то есть базисные квантовые состояния выбраны абсолютно произвольно — единственное условие, которому они должны удовлетворять, — это ортогональность. Просто так уж сложилось по традиции, что выбирают горизонтальный вектор $|0\rangle$ и вертикальный вектор $|1\rangle$. Но, само собой разумеется, это не единственный базис. Базисов может существовать бесконечное количество — любая пара ортогональных единичных векторов может служить базисом.

Например, другим часто используемым базисом является базис $\{|+\rangle, |-\rangle\}$:

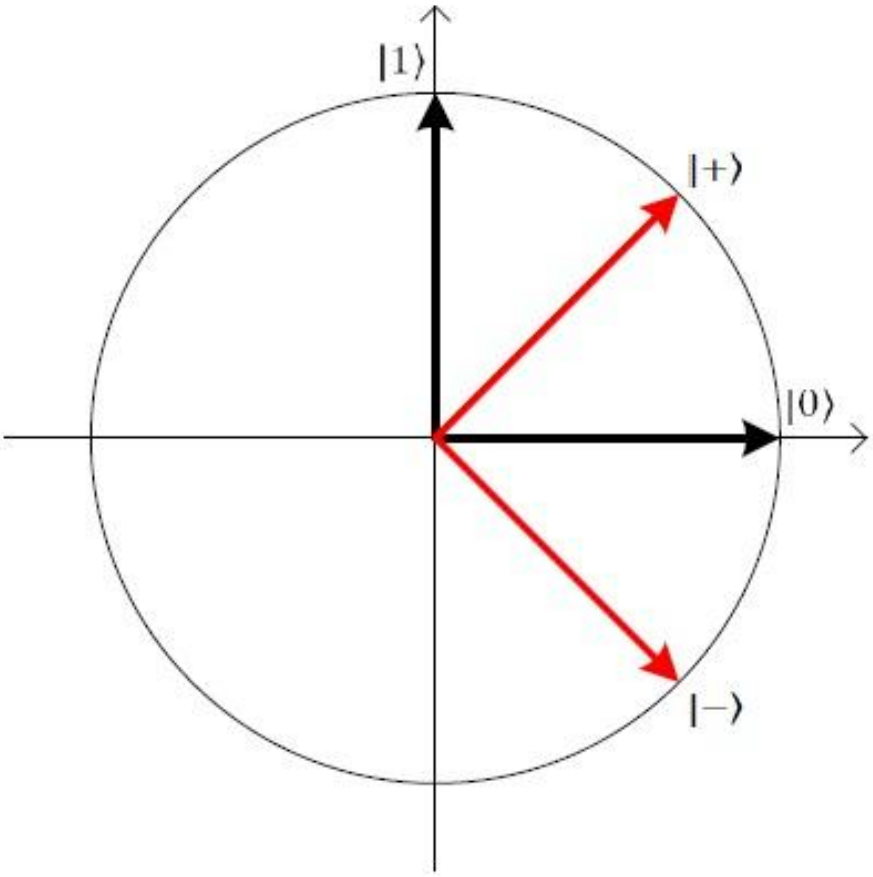

Рисунок 2.3 - Базис $\{|+\rangle, |-\rangle\}$ и его расположение относительно стандартного базиса

Естественно, что оба квантовых состояния этого базиса можно выразить через основной базис $\{|0\rangle, |1\rangle\}$:

$$+ = \frac{1}{\sqrt{2}}|0 + \frac{1}{\sqrt{2}}|1 \qquad (2.6)$$

$$- = \frac{1}{\sqrt{2}}|0 - \frac{1}{\sqrt{2}}|1 \qquad (2.7)$$

Этот базис также часто используется в модели квантовых вычислений.

Если кубит $|\varphi\rangle$ разлагается в стандартном базисе как $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$, то для перевода его в базис $\{|+\rangle, |-\rangle\}$ можно воспользоваться следующей формулой:

$$|\varphi\rangle_{+|-} = \langle +|\varphi\rangle|+\rangle + \langle -|\varphi\rangle|-\rangle = \frac{\alpha+\beta}{\sqrt{2}}|+\rangle + \frac{\alpha-\beta}{\sqrt{2}}|-\rangle \qquad (2.8)$$

формула поясняется при помощи диаграммы на следующем рисунке:

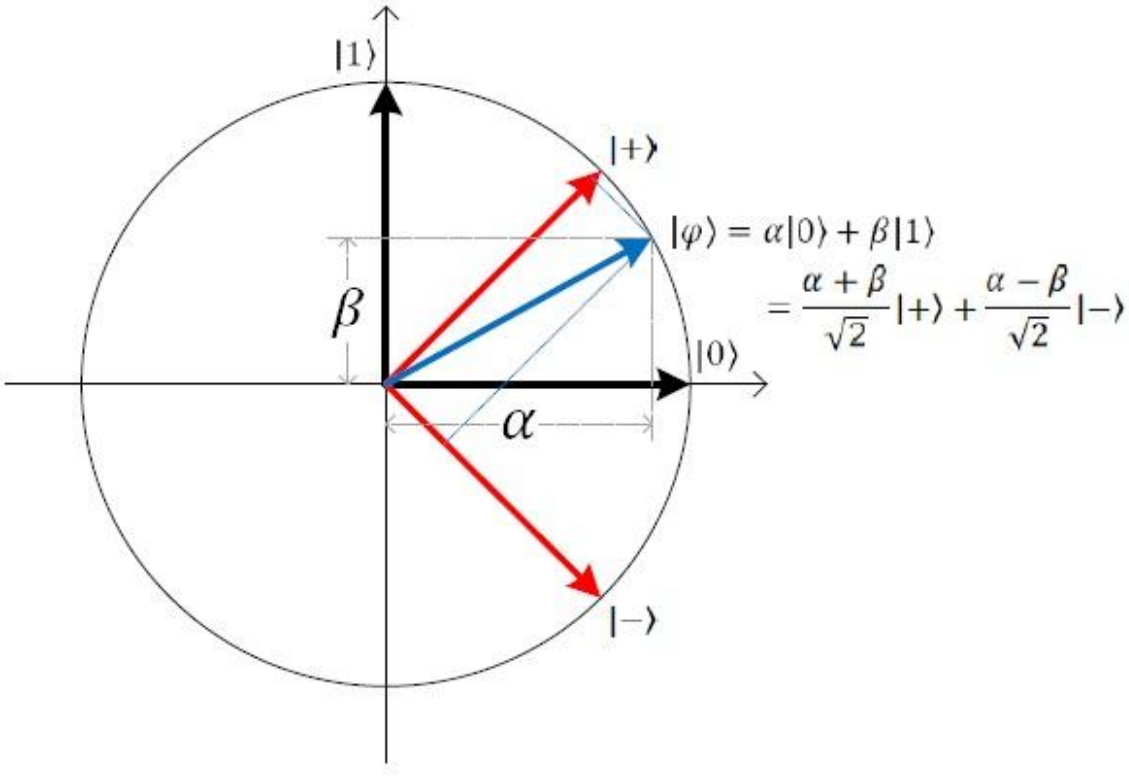

Рисунок 2.4 - Перевод кубита в базис $\{|+\rangle, |-\rangle\}$

Матричные операции дают возможность измерении кубита и помогают узнать амплитуды вероятности нахождения данного кубита в квантовых состояниях, определяемых некоторым базисом. Для этого надо перемножать не векторы, а сразу умножить сопряжённую матрицу, представляющую собой

базис, на векторное представление кубита. В результате получится вектор, представляющий собой именно амплитуды вероятностей. Матрица стандартного базиса выглядит так:

$$\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \tag{2.9}$$

и её комплексно-сопряжённая матрица выглядит точно так же. Если эту матрицу умножить на представление кубита в стандартном базисе, то в результате, получатся коэффициенты α и β, то есть амплитуды вероятности нахождения кубита в базисных состояниях $|0\rangle$ и $|1\rangle$. А вот матрица базиса $\{|+\rangle,|-\rangle\}$ выглядит так:

$$\begin{matrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{matrix} \tag{2.10}$$

Если действительные коэффициенты при двух базисных квантовых состояниях приводили к появлению одномерного пространства состояний (окружность), то резонно предположить, что использование комплексных коэффициентов приведёт к появлению двумерного пространства — сферы. Такая визуализация называется сферой Блоха.

Поскольку каждый кубит нормирован, его длина всегда равна единице (то есть, напомним, $|\alpha|^2 + |\beta|^2 = 1$), то разложение в стандартном базисе можно записать как:

$$|\varphi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\omega}\sin\frac{\theta}{2}|1\rangle\right) \tag{2.11}$$

Поскольку коэффициенты α и βявляются комплексными числами, то ничто не мешает записать их как систему уравнений:

$$\alpha = e^{i\gamma}\cos\frac{\theta}{2} = \cos\gamma\cos\frac{\theta}{2} + i\sin\gamma\cos\frac{\theta}{2} \tag{2.12}$$

$$\beta = e^{i\gamma}e^{i\omega}\sin\frac{\theta}{2} = ei\ \gamma + \omega\ \sin\frac{\theta}{2} = \cos\ \gamma + \omega\ \sin\frac{\theta}{2} + i\sin(\gamma + \omega)\sin\frac{\theta}{2} \tag{2.13}$$

В квантовом мире множитель $e^{i\gamma}$ можно опустить, поскольку он не приводит к каким-либо наблюдаемым эффектам — это просто некоторый фазовый коэффициент, который при проведении измерения никак не влияет на

результаты. Это значит, что в данном случае приведённые выше уравнения можно переписать в виде:

$$\alpha = \cos\frac{\theta}{2} + i\sin\frac{\theta}{2} \tag{2.14}$$

$$\beta = \cos\omega\sin\frac{\theta}{2} + i\sin\omega\sin\frac{\theta}{2} \tag{2.15}$$

Кубит описывается двумя угловыми параметрами — θ и ω, а они приводят к чёткой геометрической интерпретации кубита с комплексными коэффициентами в виде точки на сфере:

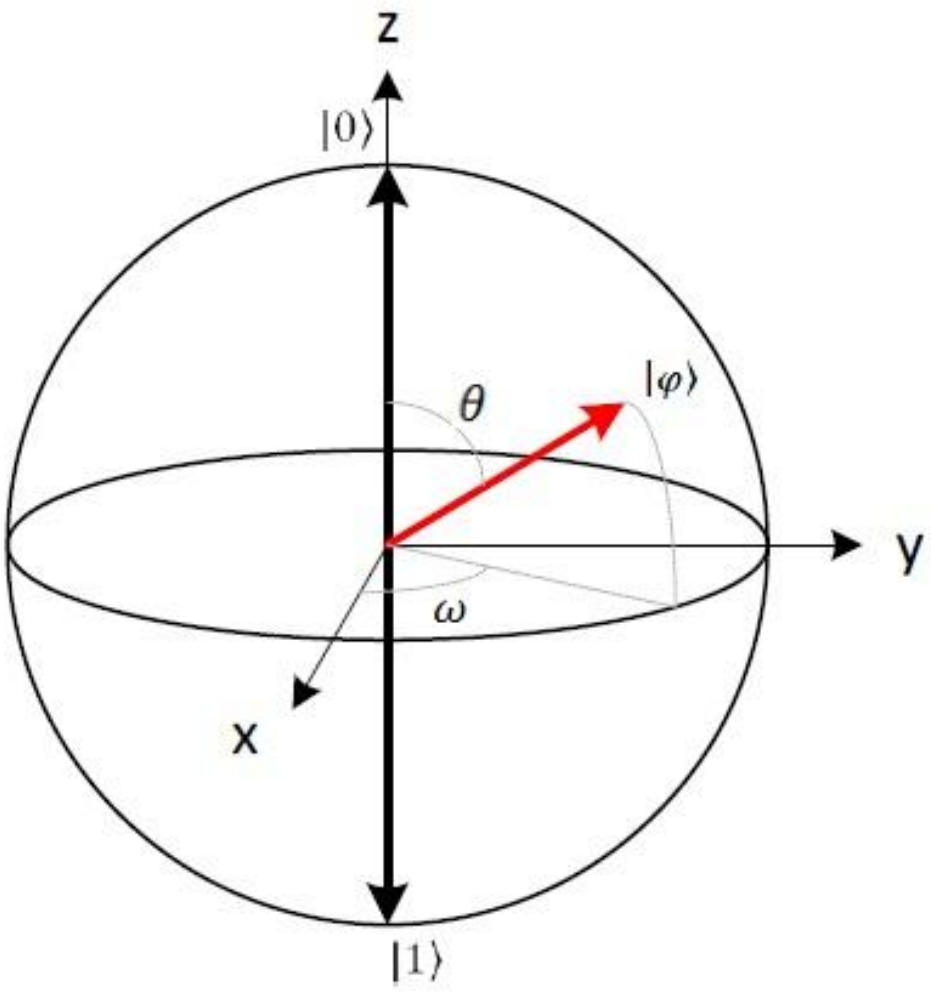

Рисунок 2.5- Кубит как точка на сферах Блоха

Следует описать важную особенность операции измерения в модели квантовых вычислений, которая, если можно так выразиться, контр-интуитивна. Измерение — это дорога в один конец. Другими словами, если мы

произвели измерение кубита и получили какой-то конкретный результат (с определённой вероятностью), то сам кубит принял значение, полученное в результате измерения, и вся суперпозиция базисных состояний была потеряна. Восстановить её невозможно.

Таким образом, после измерения вся квантовая информация, которая хранилась посредством суперпозиции базисных состояний с комплексными коэффициентами, теряется, а остаётся только одна из двух альтернатив. Именно поэтому кубит и называется «битом» — несмотря на то, что потенциально в нём в «скрытом виде» хранится бесконечное количество информации (произвольная суперпозиция двух ортонормированных базисных состояний), при измерении из него можно получить только один бит информации — либо одно, либо другое состояние в базисе измерения.

2.2.2 Несколько кубит

В квантовой механике модели квантовых вычислений считается, что любые два кубита всегда находятся в разных пространствах состояний, поэтому даже если обозначения их базисов одинаковые, то всё равно считается, что базисы разные (и для этого иногда используют индексы, которые тут же начинают опускать). Однако это — усложнение рассмотрения, поэтому далее будет считаться, что все кубиты находятся в одном и том же пространстве состояний и раскладываются на суперпозицию в одинаковых базисах.

Перечислим основополагающие принципы, на которых строится модель квантовых вычислений.

Есть два кубита $|\varphi\rangle$ и $|\psi\rangle$. Они представляют собой две суперпозиции базисных состояний, скажем, в стандартном базисе: $|\varphi\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$ и $|\psi\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$ — это просто два обычных кубита. Соединив их в двухкубитную систему, получится:

$$|\varphi\rangle|\psi\rangle = \alpha_1\alpha_2|0\rangle|0\rangle + \alpha_1\beta_2|0\rangle|1\rangle + \beta_1\alpha_2|1\rangle|0\rangle + \beta_1\beta_2|1\rangle|1\rangle \quad (2.16)$$

Для упрощения обозначений запись $|\varphi\rangle|\psi\rangle$ сокращают до $|\varphi\psi\rangle$, и это так называемое тензорное произведение кубитов или, ещё шире, тензорное произведение квантовых систем. Внезапно система из двух кубитов представляет собой суперпозицию четырёх базисных состояний. Добавление третьего кубита в квантовую систему приводит к появлению суперпозиции восьми состояний. Так что система из $\mathbf{n}$ кубитов представляет собой суперпозицию из 2^{n} квантовых состояний.

Квантовая модель вычислений говорит, что вычисления с кубитом производятся одновременно со всеми базисными состояниями, на которые кубит разложен. Поскольку для одного кубита имеет место два базисных состояния, при работе с одним кубитом вычисления одновременно производятся с двумя квантовыми состояниями. Таким образом, при работе с многокубитовой системой вычисления одновременно производятся с экспоненциальным числом квантовых состояний.

Опишем на примере тензорное произведение:

$$00 = 0\ 0 = \begin{matrix}1\\0\end{matrix}\ \begin{matrix}1\\0\end{matrix} = \begin{matrix}1 & 1\\0 & 0\\0 & 1\\0 & 0\end{matrix} = \begin{matrix}1\\0\\0\\0\end{matrix} \tag{2.17}$$

Точно таким же образом выходит следующее представление других квантовых состояний двухкубитовой системы:

$$01 = \begin{matrix}0\\1\\0\\0\end{matrix} \tag{2.18}$$

$$10 = \begin{matrix}0\\0\\1\\0\end{matrix} \tag{2.19}$$

$$11 = \begin{matrix}0\\0\\0\\1\end{matrix} \tag{2.20}$$

Можно убедиться, что эти четыре вектора образуют ортонормированный базис в четырёхмерном комплекснозначном пространстве. Матрица этого базиса выглядит следующим образом:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \tag{2.21}$$

Системы из n-кубитов стандартным базисом является единичная матрица размера $2^n \times 2^n$. Но, само собой разумеется, что базисов может быть бесконечное число: единственное ограничение — это ортонормированность всех векторов.

Существует такая суперпозиция квантовых состояний:

$$|\varphi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \tag{2.22}$$

Данная суперпозиция отвечает главному свойству — её коэффициенты, то есть амплитуды вероятности, соответствуют условию равенства единице суммы квадратов модулей. Так что такое разложение системы из двух кубитов вполне может существовать. Достаточно рассмотреть систему уравнений:

$$\begin{aligned} \alpha_1\alpha_2 &= \frac{1}{\sqrt{2}} \\ \alpha_1\beta_2 &= 0 \\ \beta_1\alpha_2 &= 0 \\ \beta_1\beta_2 &= \frac{1}{\sqrt{2}} \end{aligned} \tag{2.23}$$

чтобы понять, что у нее нет решения. Такая квантовая суперпозиция называется запутанное состояние, и у таких состояний есть множество применений в рамках модели квантовых вычислений. В практических целях выделяют четыре подобных состояния, которые называются состояниями Белла по имени одного из исследователей. Джон Стюарт Белл доказал при помощи эксперимента, что в квантовой механике нет так называемых «скрытых параметров», и что сама

природа мира оперирует неопределённостью, которая «схлопывается» в момент измерения.

Есть четыре состояния Белла, которые, образуют ортонормированный базис:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \quad (2.24)$$

$$\Phi^- = \frac{1}{\sqrt{2}}\ 00\ - \frac{1}{\sqrt{2}}|11\rangle \quad (2.25)$$

$$|\varphi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle \quad (2.26)$$

$$|\varphi^-\rangle = \frac{1}{\sqrt{2}}|01 - \frac{1}{\sqrt{2}}|10\rangle \quad (2.27)$$

Многие квантовые алгоритмы используют данные состояния, именно наличие таких запутанных состояний позволяет осуществлять некоторые достоинства квантовых вычислений, недоступные в классической вычислительной модели, поскольку в ней попросту нет аналога этому явлению.

2.2.3 Гейты и квантовые схемы

В модели квантовых вычислений каждое вычисление осуществляется при помощи так называемых гейтов. Гейт в терминах квантовой механики — это «унитарное преобразование», то есть некоторая квадратная матрица U, размерность которой соответствует количеству базовых квантовых состояний, подчиняющаяся простому свойству:

$$U^\dagger U = UU^\dagger = I \quad (2.28)$$

где $U^\dagger$ - есть эрмитово-сопряжённая матрица к U.

Эрмитовое сопряжение является операцией над квадратными матрицами с комплексными числами. Для того чтобы получить эрмитово-сопряжённую матрицу, необходимо транспонировать матрицу и заменить все числа в ней на их комплексно-сопряжённые.

Такие преобразования являются обратимыми в модели квантовых вычислений, поскольку если мы применим к какому-либо кубиту некоторое

унитарное преобразование, то для получения первоначального состояния кубита мы можем применить эрмитовое сопряжение этого унитарного преобразования, в результате чего будет получен первоначальный кубит:

$$U^{\dagger}(U|\varphi\rangle) = U^{\dagger}U|\varphi\rangle = (U \dagger U)|\varphi\rangle = I|\varphi\rangle = |\varphi\rangle \quad (2.29)$$

Тривиальным оператором является единичная матрица, поскольку эрмитово-сопряжённой к ней является она сама же, а произведение двух единичных матриц есть единичная матрица.

Изучим действие матрицы X:

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle \quad (2.29)$$

$$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \quad (2.30)$$

$$X|\varphi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \quad (2.31)$$

Этот унитарный оператор можно считать квантовым аналогом логического элемента НЕ, поскольку он переводит кубит $|0\rangle$ в $|1\rangle$ и наоборот. У суперпозиции базовых вычислительных состояний этот оператор просто меняет местами коэффициенты при базисных состояниях. Данный оператор считают квантовой операцией отрицания.

Рассмотрим действие оператора Y:

$$X|0\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{matrix} 0 \\ i \end{matrix} \quad (2.32)$$

$$X|1\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -i \\ 0 \end{pmatrix} \quad (2.33)$$

$$X|\varphi\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} -\beta i \\ \alpha i \end{pmatrix} \quad (2.34)$$

Данные оператор называется «сдвиг фазы», нет аналогов в традиционной логике. А оператор Z просто меняет знак у коэффициента при базовом состоянии $|1\rangle$. Четыре матрицы Паули (I, X, Y и Z) образуют полный базис унитарных операторов в двумерном гильбертовом пространстве.

Опишем гейт Адамара, поскольку он очень примечателен тем, что переводит кубиты из стандартного вычислительного базиса в базис $\{|+\rangle,|-\rangle\}$ и обратно. Этот гейт обозначается как H и имеет следующую матрицу:

$$H = \begin{matrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{matrix} \quad (2.36)$$

Этот унитарный оператор часто используется в квантовых алгоритмах.

Гейты бывают не только однокубитовые, унитарное преобразование может воздействовать на систему кубитов произвольного размера. В этом случае размерность матрицы унитарного преобразования равно размерности базиса, то есть 2^k для k кубитов. Наиболее важный двухкубитовый гейт, который называется CNOT («контролируемое НЕ»):

$$CNOT = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \quad (2.35)$$

Данный гейта понять просто — он применяет ко второму кубиту гейт X тогда и только тогда, когда первый кубит равен $|1\rangle$. То есть имеет место система равенств:

$$CNOT\ 00 = 00$$
$$CNOT\ 01 = 01$$
$$CNOT\ 10 = 11$$
$$CNOT|11\rangle = |10\rangle$$

Этот гейт осуществляет квантовую условную операцию — условием является значение первого кубита, и если оно равно $|1\rangle$, ко второму кубиту применяется гейтX.

Рассмотрим то, как гейты обозначаются. Большинство гейтов обозначаются прямоугольником с входами и выходами, причём число выходов

в точности равно числу входов (вычисления обратимы). В самом прямоугольнике указывается наименование гейта:

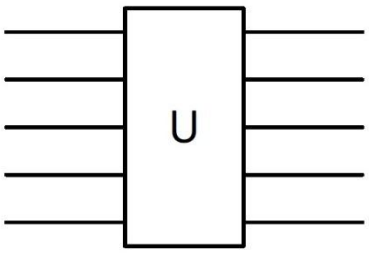

Рисунок 2.6 - Условное обозначение произвольного гейта U на квантовых схемах.

Если необходимо указать, какие кубиты подаются на тот или иной вход, то эти кубиты указываются непосредственно около входа. Обычно для запуска процесса квантовых вычислений все входные кубиты инициализируются в значение $|0\rangle$,и если около входа не указан кубит (как на рисунке вверху), то считается, что на этот вход подаётся именно значение $|0\rangle$.

Особое обозначение имеется у гейта CNOT, как у «квантового условного оператора». Он обозначается так, как указано на следующем рисунке:

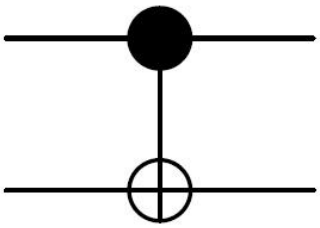

Рисунок 2.7 – Условное обозначение гейтаCNOT

Чёрный кружок обозначает контролирующий кубит, то есть в зависимости от значения которого к контролируемому кубиту применяется квантовая операция отрицания. Соответственно, контролируемый кубит обозначается кружком с крестом, и это легко запомнить, поскольку в логике символом $\oplus$ обозначают логическую операцию «Исключающее ИЛИ» (XOR), а

именно так можно определить результат контролируемого кубита: $|b'\rangle = |a\rangle \oplus |b\rangle$, где a — контролирующий кубит, а b — контролируемый. При этом необязательно, что контролирующий кубит находится сверху, на некоторых схемах его удобнее рисовать снизу.

Контролируемый гейт CU обозначается схожим образом:

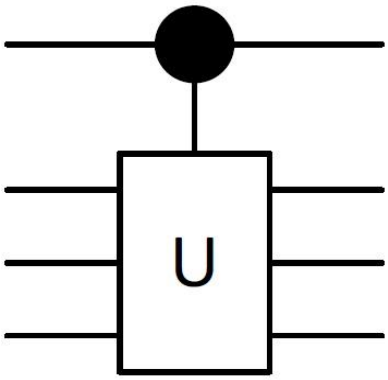

Рисунок 2.8 - Условное обозначение гейта CU

Часто в квантовых алгоритмах используются так называемые оракулы, которые являются квантовыми аналогами чёрных ящиков. Если есть некоторая функция на двоичных строках $f: \{0,1\}^n \rightarrow \{0,1\}^m$, то соответствующиморакулом для неё будет являться некоторое унитарное преобразование $Uf : \{0,1\}^{n+m} \rightarrow \{0,1\}^{n+m}$, которое считается заданными вычисляющим эту функцию f. Считается, что есть некоторый физический процесс, который обратимым образом вычисляет эту функцию, осуществляя квантово-механическое унитарное преобразование. Обозначаться такой оракул будет следующим образом:

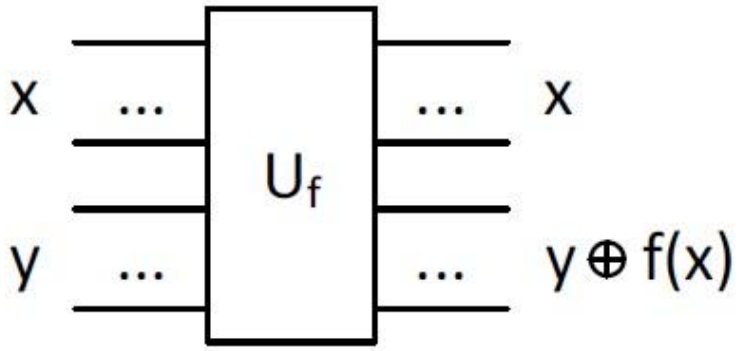

Рисунок 2.9 - Условное обозначение оракула U_f

Здесь регистр x является входными данными для функции fи этот параметр занимает n кубитов. Регистр y является вспомогательным (и обычно, но не обязательно, проинициализированным в строку нулей) и занимает m кубитов. На выходе результат вычисления функции складывается по модулю 2 с этим регистром.

Процесс измерения кубита обозначается стилизованным изображением измерительного прибора:

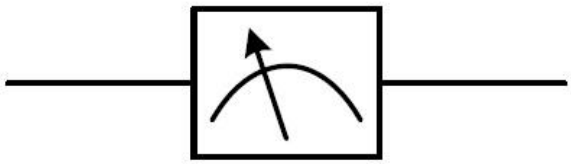

Рисунок 2.10 - Условное обозначение операции измерения

Данные обозначения гейтов являются строительными блоками при изображении квантовых схем, то есть диаграмм для визуализации квантовых алгоритмов. Вот пример одной квантовой схемы:

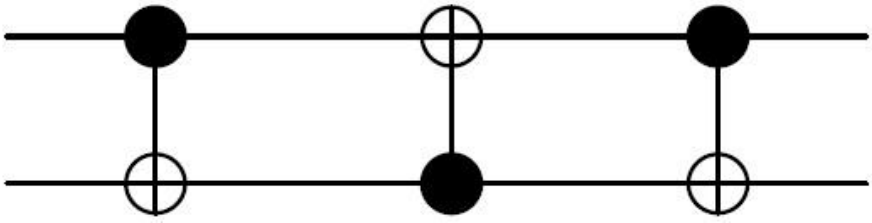

Рисунок 2.11 - Пример квантовой схемы

Эта квантовая схема примечательна тем, что меняет местами квантовые состояния двух кубитов, поданных на её вход. Это можно легко проверить для базисных состояний двухкубитовой квантовой системы $\{|00\rangle,|01\rangle,|10\rangle,|11\rangle\}$ при помощиследующей рисунка:

Входные кубиты	Результат первого гейта	Результат второго гейта	Результат третьего гейта
$\|00\rangle$	$\|00\rangle$	$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$	$\|11\rangle$	$\|10\rangle$
$\|10\rangle$	$\|11\rangle$	$\|01\rangle$	$\|01\rangle$
$\|11\rangle$	$\|10\rangle$	$\|10\rangle$	$\|11\rangle$

Рисунок 2.12 - Результат работы квантовой схемы с рис 2.11

Матричное представление этой квантовой схемы выглядит так:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Данный раздел позволяет понять построение квантовых схем.

2.2.4 Квантовая схемотехника

Квантовая схемотехника — это методология анализа и синтеза квантовых схем, реализующих те или иные алгоритмы (в общем понимании, не только квантовые). Обобщённо любой вычислительный процесс представляется в виде тройки (вход, процесс преобразования, выход). Задачами квантовой схемотехники можно назвать:

1. Прямой анализ. При наличии схемы входа и описания вычислительного процесса определить схему выхода.

2. Обратный анализ. При наличии описания вычислительного процесса и схемы выхода определить схему входа.

3. Синтез. При наличии схем входа и выхода построить описание вычислительного процесса.

Прямой анализ. С математической точки зрения эта задача решается достаточно просто. Входная схема — это описание множества возможных значений, которые могут быть поданы на вход квантовому вычислительному процессу. Поскольку квантовый регистр, то есть набор кубитов, может быть представлен в виде вектора, а каждый гейт в квантовой схеме представляет собой унитарную матрицу, то задача прямого анализа сводится к последовательному умножению матриц на вектор. А можно сначала перемножить все матрицы в правильном порядке, а затем итоговую матрицу умножить на входной вектор. Проведя эту процедуру на всех возможных значениях входа (либо применив иные техники анализа), можно получить все выходные значения, после чего объединить их в схему. Данный вид анализа затрудняется тем, что при увеличении количества кубитов размерность векторов и матриц увеличивается экспоненциально. И если для квантовых регистров, состоящих из 1 — 3 кубитов такую процедуру ещё можно провести, то ручной анализ для четырёх кубитов уже затруднителен, а для десяти кубитов и больше уже сложно проводить и перемножение матриц на компьютере [8].

Обратный анализ. В рамках модели квантовых вычислений задача обратного анализа тривиально сводится к задаче прямого анализа. Поскольку вычисления являются обратимыми, а матрицы всех гейтов — унитарными, то для обратного анализа заданной квантовой схемы достаточно обратить её, то есть перекоммутировать гейты в обратном порядке, сделав выход входом и наоборот, а сами гейты преобразовать в эрмитово-сопряжённые. После этого проводится описанная ранее процедура прямого анализа. Само собой разумеется, что вышеприведённые рассуждения применимы только к квантовым схемам, в которых нет операции измерения, которая является

необратимой. С другой стороны, измерение обычно применяется в самом конце квантовых вычислений, когда необходимо получить классический результат, поэтому обращение схемы можно осуществить, отбросив операции измерения. И есть совсем немного квантовых алгоритмов, в которых измерение производится в середине процесса вычислений (например, квантовая телепортация), и к таким алгоритмам и их квантовым схемам этот метод обратного анализа не подходит, и надо использовать иные (если они вообще существуют — в общем случае обратная задача неразрешима).

Синтез. Задача построения квантовой схемы по заданным входу и выходу усиливается, по сравнению с классическим случаем, необходимостью обеспечить обратимость вычислений. В общем случае произвольный вычислительный процесс может быть описан как двоичная функция, принимающая на вход n битов и возвращающая m битов. Такая функция в классическом варианте может быть построена при помощи базисного (универсального) набора логических элементов.

Наличие классической схемы из универсального набора строительных блоков для заданной функции переводит задачу синтеза в задачу построения квантового оракула. Как уже было указано выше, квантовый оракул строится из классического выражения функции следующим образом:

Наличие классической схемы из универсального набора строительных блоков для заданной функции переводит задачу синтеза в задачу построения квантового оракула. Как уже было указано выше, квантовый оракул строится из классического выражения функции следующим образом:

Квантовый оракул будет иметь по $(n + m)$ входов и выходов. Первые n входов принимают входные параметры функции, а следующие m входов инициализируются в $|0\rangle$. Соответственно, первые n выходов возвращают входные данные, а следующие m выходов получают сумму по модулю 2 (операция Исключающее ИЛИ) значения функции на заданном входе с m входными данными [8].

Все элементы классической схемы преобразуются в соответствующие квантовые гейты (например, можно воспользоваться универсальным набором из гейтов H и CNOT, либо квантовым аналогом элемента Тоффоли). Это может привести к появлению огромного количества так называемых мусорных кубитов.

Построенная схема из универсальных квантовых элементов подвергается процессу оптимизации. Это довольно нетривиальный процесс, и в каждом случае применяются свои методы и техники. Цель этого процесса — свести количество мусорных кубитов к минимуму, желательно вообще их исключить.

Если от мусорных кубитов избавиться не удалось (пусть их осталось l), то все они добавляются к входным и выходным (стало быть, входов и выходов становится $(n + m + l)$). Но эти кубиты должны быть вначале инициализированы в $|0\rangle$, а по окончании вычислений они также должны быть установлены в $|0\rangle$. Это следствие законов квантовой механики [8].

Таким образом, будет построена квантовая схема классической функции. Однако, это не единственный способ. Другим вариантом является построение одной унитарной матрицы для представления полной классической функции. Эта задача может быть успешно разрешена при помощи решения системы уравнений, получаемой из произведения матрицы на вектор. Проблема лишь в том, что количество уравнений и неизвестных растёт экспоненциально от количества кубитов. И если количество входов и выходов равно $(n + m)$, то количество уравнений и неизвестных в них будет равно, как полагается, $2^{2(n+m)}$. Решать такую систему просто для небольших чисел, а вот для больших уже проблематично.

2.2.5 Принципы квантовых вычислений

Перечислим основополагающие принципы, на которых строится модель квантовых вычислений.

Обратимость вычислений — главный принцип, из которого, в общем-то, следуют все остальные. Квантовые вычисления обратимы во времени, а это значит, что по результату любого вычисления можно восстановить исходные данные. Более того, для любой квантовой схемы можно построить дуальную схему, которая будет получать на вход результат работы первоначальной схемы и возвращать исходные данные для неё. Для этого достаточно всего лишь перевернуть схему справа налево, а все унитарные преобразования заменить на эрмитово-сопряжённые [8].

Поскольку все унитарные преобразования являются обратимыми, все они имеют ровно столько же выходов, сколько и входов. Из этих рассуждений следует, что квантовые вычисления избыточны. Ведь, например, самые простые логически обратимые элементы Тоффоли и Фредкина имеют по три входа и три выхода. Если эти элементы используются для получения, скажем, элемента НЕ, то по два входа и выхода будут незадействованными. А для элементов И, ИЛИ и других не используемыми будет один вход и два выхода. Для элемента FANOUT наоборот два входа использоваться не будут, равно как и один выход. Это можно проиллюстрировать следующей диаграммой. На рис. 2.13 показано выражение элемента Тоффоли через элемент Фредкина. Из всего огромного множества входных и выходных битов используются только по три с каждой стороны:

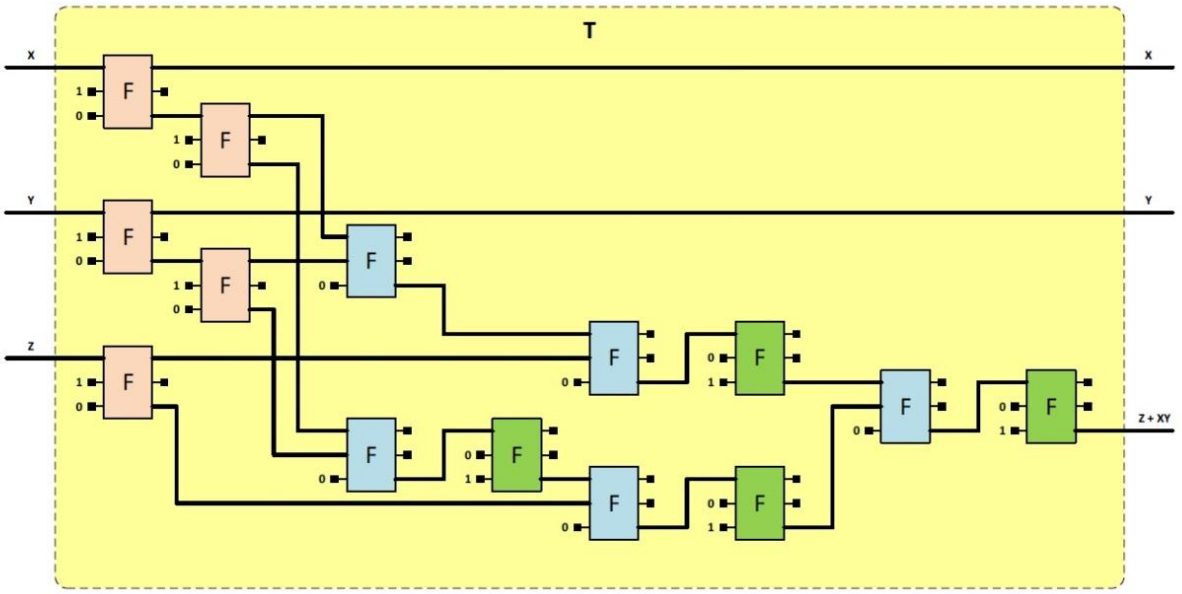

Рисунок 2.13 - Диаграмма, показывающая один из возможных вариантов выражения элемента Тоффоли через элемент Фредкина (красные элементы — FANOUT, синие — И, зелёные — НЕ)

Данный принцип влечёт, что в процессе работы квантовых алгоритмов будет накапливаться огромное количество «неиспользуемых» битов. Если их начать уничтожать (например, при помощи своеобразной «сборки мусора»), то начнёт выделяться огромное количество теплоты, а вычисления станут необратимыми [8].

Собственно, на рисунке 2.13 видно, что для выражения одного элемента с тремя входами и тремя выходами по обратимой схеме пришлось задействовать 14 базовых элементов, и общее число входов и выходов на схеме стало равным по 26, из которых только по 3 входа и выхода используются, а остальные являются вспомогательными и, по сути, «неиспользуемые» [8].

Есть мнение, что за экспоненциальное ускорение решения некоторых задач, которое даёт модель квантовых вычислений, придётся заплатить экспоненциальным увеличением размера задействованной памяти, и ситуация с выражением элемента Тоффоли через элемент Фредкина это отчасти иллюстрирует.

Для решения этой проблемы был разработан метод, который позволяет уничтожать промежуточные «неиспользуемые» биты без потери обратимости вычислений.

Всё это обозначает, что в квантовых схемах нет и не может быть циклов и возвратов назад. Кубиты как бы двигаются по «проводам», проходя через гейты и преобразовываясь в соответствии с предназначением гейта. Поток управления квантовой программой движется от начала алгоритма только вперёд. Унитарное преобразование и контролируемое унитарное преобразование — вот единственные способы выполнения вычислений. Измерение — единственный способ получения результата, который, к тому же, «убирает» суперпозицию, в которой находятся кубиты.

Следующим принципом является квантовый параллелизм. Система кубитов в процессе прохождения через гейты унитарных преобразований параллельно решает одну и ту же задачу для огромного, экспоненциально большого количества данных из-за того, что кубиты находятся в суперпозиции базисных состояний. Поскольку с ростом числа кубитов размерность базиса растёт в степенной зависимости, квантовый параллелизм обладает огромной вычислительной мощью. Главное — уметь правильно пользоваться этим принципом [8].

С параллелизмом тесно связана интерференция. Некоторые алгоритмы используют интерференцию квантовых состояний для того, чтобы взаимно усилить требуемый результат и наоборот сгладить результат нежелательный. Повторяя несколько раз последовательность параллельной обработки кубитов с интерференцией их состояний, разработчик алгоритма может так усилить амплитуду искомого состояния, что дальнейшее измерение даст требуемый результат с высокой вероятностью (варьируя количество повторений шага с интерференцией, можно управлять уровнем вероятности, доводя его до любого заданного значения).

Квантовая запутанность — ещё один принцип, причём наименее изученный, равно как и наименее поддающийся рациональному осмыслению. Квантовая запутанность позволяет решить многие хитрые задачи, являясь ключевым фактором многих квантовых алгоритмов.

2.2.6 Общая архитектура квантового компьютера

Квантовые вычисления и их принципы позволяют изобразить следующую структуру архитектуру квантового компьютера, в рамках которого будут производиться вычисления [8].

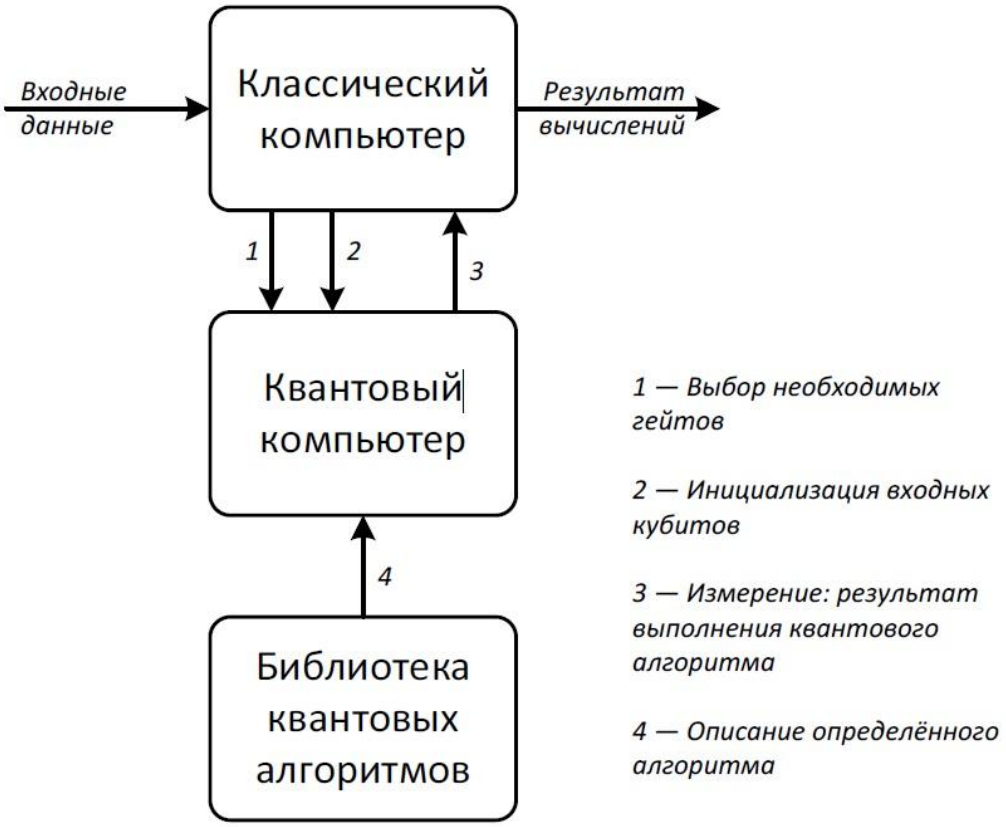

Рисунок 2.14 - Общая архитектура квантового компьютера

Программистов, занимающихся квантовыми вычислениями, можно разделить на два типа:

- системные квантовые программисты будут создавать новые изощрённые квантовые алгоритмы для размещения их в библиотеке квантовых

алгоритмов и последующего использования в работе всей системы;

- прикладные квантовые программисты будут использовать квантовые алгоритмы на практике, решая при помощи них прикладные задачи, но при этом они должны будут знать как номенклатуру алгоритмов в библиотеке, так и понимать смысл их работы, чтобы оптимальным образом производить их вызов и использование их результатов в прикладных программах для обычного компьютера.

Подведем краткие выводы.

Модель квантовых вычислений — это некоторая математическая система, которая позволяет производить произвольные вычисления, причём уже показано, что для некоторых задач эта модель позволяет реализовать намного более эффективные алгоритмы, чем традиционная вычислительная модель, основанная на машине Тьюринга или лямбда-исчислении. Реализация этой новой вычислительной модели в «железе» не имеет фундаментальных преград, так что дело только за технологиями [8].

Сама модель сводится к выполнению ряда унитарных преобразований векторов в 2^n-мерном гильбертовом пространстве, где n — количество кубитов, а каждое унитарное преобразование представляет собой матрицу специального вида. Таким образом, очень упрощённо, можно сказать, что по своей сути квантовые вычисления заключаются в перемножении матрицы размера $2^n \times 2^n$ (которая сама ещё должна быть получена как произведение целого набора матриц такого же или меньших размеров) на вектор размера 2^n. Эти вычисления, конечно же, можно выполнить и на обычных компьютерах, но сложность такого умножения будет расти экспоненциально в зависимости от количества моделируемых кубитов [8].

На основе выше сказанного можно сделать вывод, что реализация квантового компьютера позволит решить проблему экспоненциального роста сложности, при этом такой компьютер будет гибридным — скорее всего,

аналоговое квантовое вычислительное устройство будет управляться обычным цифровым компьютером.

2.3 Математическая модель алгоритма Гровера

Поскольку алгоритм Гровера представляет собой обобщенный, не зависящий от конкретной задачи поиск, функция f представляется в нем в виде черного ящика [8].

$$f: \{0,1\}^n \to \{0,1\}^n \tag{2.37}$$

В области определения fприсутствует одна особая точка, на которой fпринимает нужное нам значение:

$$\exists! \omega: f(\omega) = a \tag{2.38}$$

В приведенном примере с телефонным справочником ω - фамилия абонента, в то время как a - известный нам телефонный номер. Необходимо найти ω. Как видите, это обобщенный вариант любой задачи из NP. Зная ответ – число ω, мы можем легко его проверить, вызвав оракул и посмотрев, получим ли мы значение a.

Определим функцию f_ω следующим образом:

$$f_\omega(x) = \delta_{x=\omega} \tag{2.39}$$

f_ωпринимает значение «0» во всех точках, кромеω, от которойона равна «1». Подобную функцию легко реализовать, имея оракул f.

Оракул

Квантовый оракул действует на состояние вида $\frac{1}{\sqrt{2}}|x\rangle(|0\rangle - |1\rangle)$:

$$U_f \frac{1}{\sqrt{2}}|x\rangle(|0\rangle - |1\rangle) = (-1)^{f(x)} \frac{1}{\sqrt{2}}|x\rangle(|0\rangle - |1\rangle) \tag{2.40}$$

Мы можем рассмотреть проекцию U_fна подпространство аргумента $(|x\rangle)$:

$$U_\omega|x\rangle = (-1)^{f(x)}|x\rangle \tag{2.41}$$

Оператор U_ωдействует как тождественный оператор на любой базисный вектор, кромеω, поэтому, мы можем записать его следующий образом:

$$U_\omega|x = I - 2|\omega \ \omega| \tag{2.42}$$

Действия этого оператора на любой вектор $|x$ определяется следующим образом:

$$U_\omega|x = |x - 2|\omega \ \omega\, x \tag{2.43}$$

Для любого базисного вектора кромеω скалярное произведение $\omega\, x$ дает 0, и мы имеем тождественный оператор:

$$\omega\, x = 0 \tag{2.44}$$

$$U_\omega|x = x - 2\,\omega \ \omega\, x = |x \tag{2.45}$$

Для вектора ω мы имеем

$$U_\omega|\omega = \omega - 2\,\omega \ \omega\,\omega = -|\omega \tag{2.46}$$

Начальное состояние алгоритма Гровера, строится следующим образом:

$$|s = H|0\rangle^n = \frac{1}{2^{\frac{n}{2}}} \sum_{x=0}^{2^n-1}|x \tag{2.47}$$

где$|s$ –сумма всех базисных векторов подпространства аргумента.

Определим еще вспомогательный оператор U_S:

$$U_s = 2\ s \ \ s - I \tag{2.48}$$

Итерация Гровера.

Алгоритм Гровера итеративен. Каждая его итерация определяется, как применение операторов U_s и U_ω к текущему состоянию системы:

$$R = U_s U_\omega \tag{2.49}$$

Действия итерации Гровера на начальное состояние sизображено на рисунке 2.15.

На рисунке 2.15 вертикальная ось - это искомый вектор ω, а горизонтальная ось – это гиперпространство, образование всеми остальными базисными векторами. Вектор $|s$ - сумма всех базисных векторов – нависает над горизонтальной гиперплоскостью под углом θ [8].

$$\sin\theta = \ s\ \omega \ = \frac{1}{2^{\frac{n}{2}}} \tag{2.50}$$

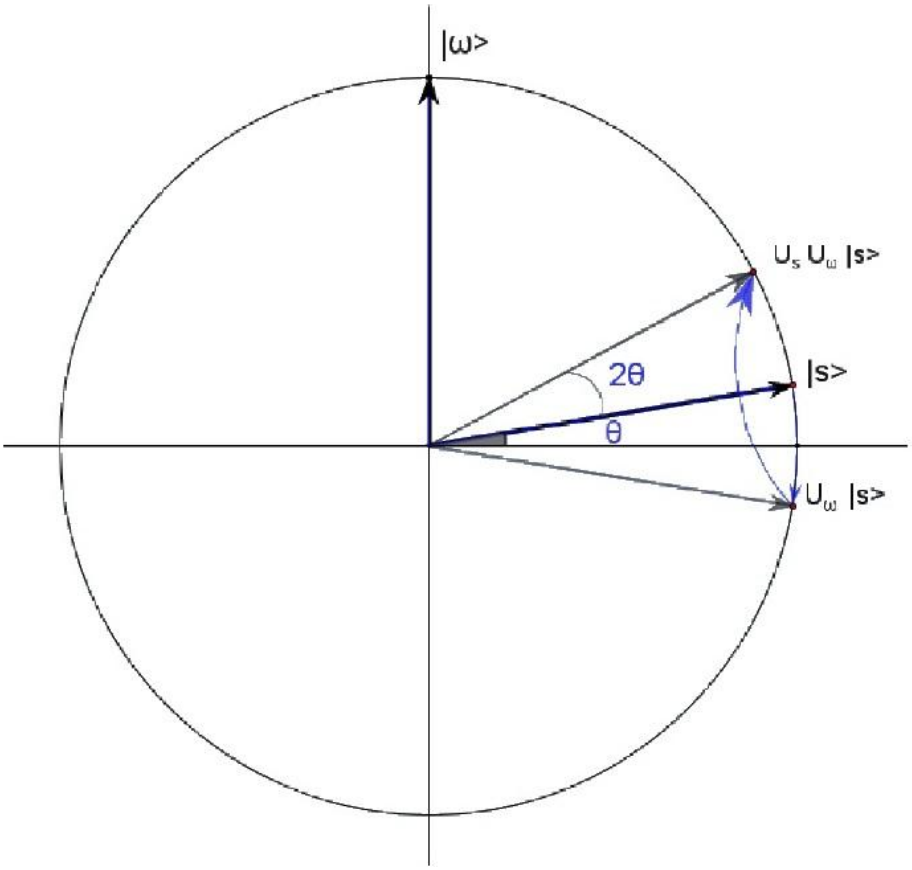

Рисунок 2.15 - Первая итерация Алгоритма Гровера

Чем больше размерность пространства, тем меньше угол θ, тем ниже находится вектор $|s\rangle$ [8].

Оператор U_ω является отражением вектора системы относительно гиперплоскости, ортогональной ω. Применив его, мы получаем вектор $U_\omega|s\rangle$, в котором все базисные составляющие остались неизменны, и только составляющая $|\omega\rangle$ заменилась на $-|\omega\rangle$. Оператор U_s это отражение вектора системы относительно вектора $|s\rangle$. После него мы получаем вектор $U_s U_\omega |s\rangle$.

Получается, что первая итерация алгоритма Гровера повернула исходный вектор к искомому вектору $|\omega\rangle$ на угол 2θ. Теперь угол вектора системы с горизонтальной гиперплоскостью составляет 3θ.

На Рисунке 2.16 Изображено действие второй итерации Гровера, которая также поворачивает вектор на угол 2θ в направлении $|\omega\rangle$.

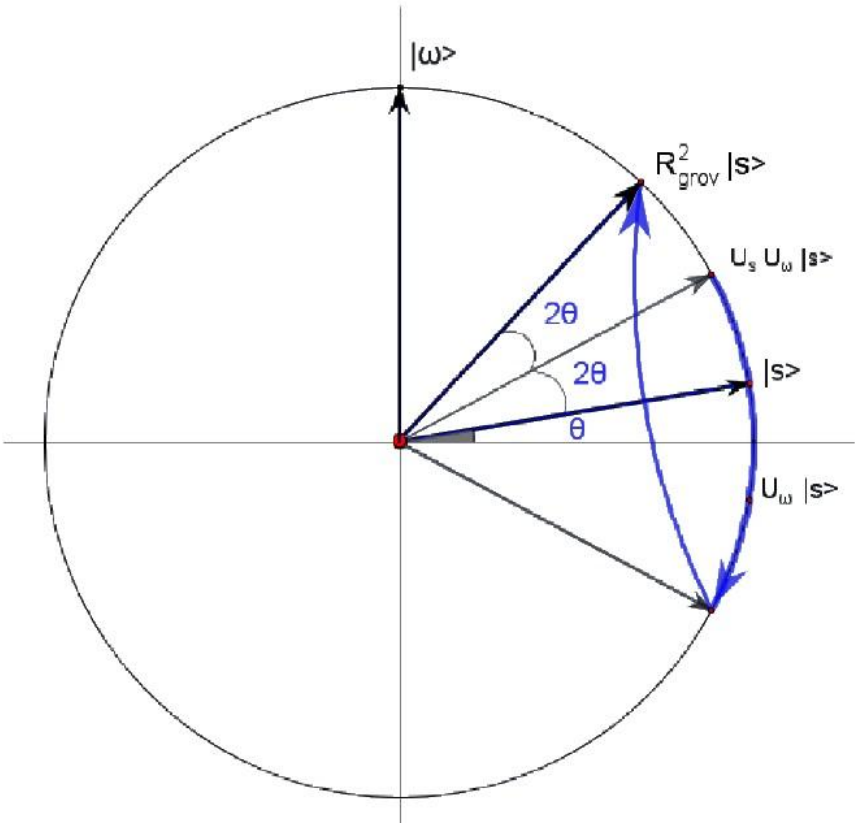

Рисунок 2.16 - Вторая итерация Алгоритма Гровера

Нетрудно убедиться, что каждая следующая итерация Гровера будет делать с вектором системы тоже самое – приближать его к$|\omega$ на угол 2θ. После Титераций угол между вектором системы и гиперплоскостью будет равен: $\theta + 2T\theta$ и нам нужно подобрать такое количество итераций T, чтобы он стал близок к$\pi/2$. Это позволит при измерении состояния получить вектор $|\omega$ свероятностью, близкой к единице[8].

$$\theta + 2T\theta = \frac{\pi}{2} \leftrightarrow T = \frac{\pi}{4\theta} - \frac{1}{2} \tag{2.51}$$

При больших значенияхnугол θ становится достаточно мал и может быть приближенно заменен своим сунусом (1):

$$T = \frac{\pi\ \sqrt{2^n}}{4} - \frac{1}{2} \approx \frac{\pi\ \sqrt{2^n}}{4} \tag{2.52}$$

Как видно вместо классических 2^nитераций квантовому компьютеру требуется только $2^{n/2}$обращений к оракулу. Ускорение является квадратичным

в данном случае что вполне может оказаться достаточно. Например, при n = 2 вместо 4-х обращений к оракулу нам необходимо только одно. А при n = 8 алгоритму Гровера потребуется около 12-ти итераций (вместо классических 256-ти).

Пример реализации алгоритма Гровера для двух кубитов на симуляторе квантового компьютера представлен на рисунке 2.17

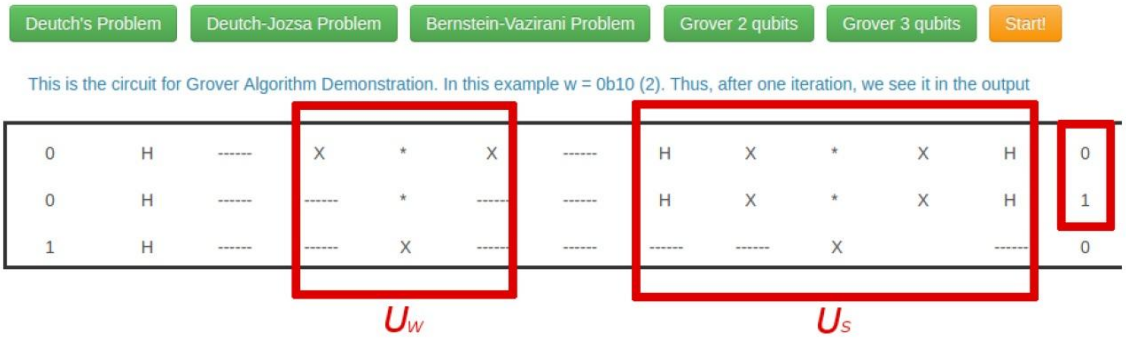

Рисунок 2.17 - Алгоритм Гровера на двух кубитах. $\omega = 0x10 = 2$

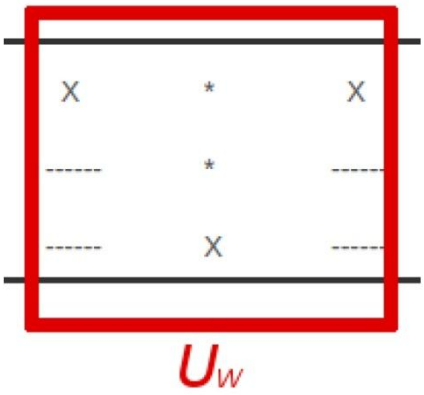

Рисунок 2.18 - Оператор U_ω. $\omega = 0x10 = 2$

Пусть дана функция — булева функция. Цель: найти хотя бы один корень уравнения f(x) = 1. На классическом компьютере, если f — произвольна, нам понадобиться O(N)операций, то есть, полный перебор. Если f в конъюнктивой нормальной форме — то данная задача является NP-полной. Алгоритм Гровера позволяет достичь квадратичное ускорение в сравнении с классическими алгоритмами перебора – за$O(\sqrt{n})$. Схема данного алгоритма представлена на рисунке:

Рисунок 2.19 - Общая диаграмма квантовой схемы алгоритма Гровера

В данной главе была проанализирована и описана математическая модель алгоритма Гровера, описана временная сложность алгоритма. На основе этих данных будет строиться компьютерная модель.

ГЛАВА 3 РЕАЛИЗАЦИЯ КВАНТОВОГО АЛГОРИТМА

В данной главе описываются квантовые языки программирования, выбирается язык для реализации алгоритма Гровера и описывается компьютерная модель алгоритма Гровера.

При написании главы использовались материалы из [8,20,21,22,23,24 и др.].

3.1 Язык программирования Q#

Компания Microsoft разработала новый язык программирования для квантовых вычислений под названием Q# в сентября 2017 года.

Данный язык должен объединить элементы программирования и квантовые вычисления. Вышедшая локальная версия поддерживает до 32 кубитов, используя 32 ГБ оперативной памяти. Также Microsoft предлагает другую версию этого симулятора, Azure, которая работает с 40 кубитами.

Q# представляет собой высокоуровневый язык программирования, предназначенный для написания скриптов, которые будут выполнять свои подпрограммы на квантовом процессоре, связанном склассическим хост-компьютером, который в конечном итоге будет получать результаты.

Разработчики, использующие язык, могут не иметь глубоких знаний о квантовой физике. Для заинтересованных, Microsoft предоставляет учебник по основным концепциям квантовых вычислений, охватывающий векторную и матричную математику, кубиты, обозначение Дирака, измерения Паули и квантовые схемы. Комплект разработки доступен бесплатно с подробными инструкциями по его установке и вводным учебным программам. Q# компилируется на квантовом симуляторе Visual Studio. Язык программирования Q# выглядит не так, как большинство других языков программирования, но, тем не менее, очень похож на C#. Самый первый пример, предоставленный Microsoft, включает создание сценария состояния Q# BellState – четырех запутанных состояний из двух кубитов. Конечный результат

приводит к наблюдению за запутыванием в двух измеренных битах на выходе программы. В более поздних учебных курсах пользователь прорабатывает скрипт для моделирования квантовой телепортации. Microsoft надеется, что введение такой новой концепции поможет заинтересовать потенциальных разработчиков и начать разработку в области квантовые вычисления [8]

Q# - это масштабируемый многопарадигмный язык программирования для квантовых вычислений. Q# - это язык квантового программирования, в котором его можно использовать для описания того, как выполняются инструкции на квантовых машинах. Машины, которые могут быть нацелены, включают в себя множество различных уровней абстракции, начиная от различных тренажеров и заканчивая фактическим квантовым оборудованием. Q# является многопарадигмой в том смысле, что он поддерживает функциональные и императивные стили программирования. Q# масштабируется тем, что позволяет писать программы на целевые машины различных размеров, начиная от небольших машин, всего лишь с нескольких сотен кубитов до больших машин с миллионами кубитов. Несмотря на то, что большие физические машины могут только принести свои плоды в будущее, Q# позволяет программисту программировать сложные квантовые алгоритмы уже сейчас. Более того, Q# позволяет масштабируемым образом выполнять различные задачи, такие как отладка, профилирование, оценка ресурсов и некоторые специализированные симуляции [55].

Естественной моделью квантового вычисления является рассмотрение квантового компьютера как сопроцессора, аналогичного тому, который используется для графических процессоров, GPUs, FPGAs и других вспомогательных процессоров. Основная логика управления управляет классическим кодом на классическом «хост-компьютере». Когда это необходимо и необходимо, хост-программа может вызывать подпрограмму, которая выполняется на дополнительном процессоре. Когда подпрограмма

завершается, хост-программа получает доступ к результатам подпрограммы [55].

На основе выше сказанного можно сделать вывод что язык программирования Q# является самые последним разработанным квантовым языком программирования, данный язык программирования является проблемно-ориентированным.

В квантовой модели три уровня вычислений:

- классическое вычисление, которое считывает входные данные, устанавливает квантовое вычисление, запускает квантовое вычисление, обрабатывает результаты вычисления и представляет результаты пользователю;
- квантовое вычисление, которое происходит непосредственно в квантовом устройстве и реализует квантовый алгоритм;
- классическое вычисление, требуемое квантовым алгоритмом во время его выполнения.

Не существует неотъемлемого требования, чтобы все эти три уровня были написаны на одном языке. Действительно, квантовые вычисления имеют несколько иные структуры управления и потребности в управлении ресурсами, чем классические вычисления, поэтому использование пользовательского языка программирования позволяет более естественным образом выражать общие закономерности в квантовых алгоритмах.

Сохранение классических вычислений означает, что язык квантового программирования может быть очень ограниченным. Эти ограничения могут обеспечить лучшую оптимизацию или более быстрое выполнение квантового алгоритма.

Q# (Q-sharp) - это язык программирования, специфичный для домена, используемый для выражения квантовых алгоритмов. Он должен использоваться для написания подпрограмм, выполняемых на дополнительном квантовом процессоре под управлением классической хост-программы и компьютера.

Q# предоставляет небольшой набор примитивных типов вместе с двумя способами (массивами и кортежами) для создания новых структурированных типов. Он поддерживает базовую процедурную модель для написания программ, с циклами и операциями if / then. Конструкциями верхнего уровня в Q# являются определяемые пользователем типы, операции и функции.

Обычно, когда мы думаем о компьютере, мы представляем одно устройство, использующее приложение, но современные вычислительные среды намного сложнее и сложнее. Приложение, с которым мы взаимодействуем, обычно опирается на несколько уровней программного обеспечения, которые обеспечивают выполнение приложения до уровня аппаратного обеспечения. Эти программные уровни необходимы для абстрагирования разработки прикладного решения от сложной сложности всей вычислительной системы. Если разработчику приходилось думать о шине, архитектурах кеша, протоколах связи и т. д. При написании простого приложения для смартфонов, задача стала бы намного сложнее. Концепция программного стека была разработана в классических вычислениях для решения этих проблем. Заимствуя классическую концепцию, стек программного обеспечения также является ключевой частью концепции квантовых вычислений с Q# [55].

Ключевой идеей программного стека является рекурсия. Он состоит из нескольких вложенных слоев интерфейсов, которые абстрагируют детали нижних уровней устройства от разработчика. Например, обычно используемый стек программного обеспечения включает в себя запуск ASP.NET (язык программирования), поверх SQL-сервера (системы управления реляционными базами данных), который работает поверх Internet Information Services (веб-сервера), который работает сверху сервера Windows (операционной системы), который управляет компьютерным оборудованием. Рассматривая программное обеспечение как иерархию, можно писать программное обеспечение в

ASP.NET, не требуя понимания низкоуровневых деталей всего программного обеспечения под ним [55].

Стек программного обеспечения в квантовых вычислениях принципиально не отличается и на практике работает на более низком уровне, чем традиционные стеки. Как выглядит квантовый стек? Квантовый компьютер не является заменой традиционных (часто называемых классических) компьютеров. Фактически, квантовые компьютеры почти наверняка будут работать в тандеме с классическими компьютерами для решения вычислительных задач. Частично это возникает из-за хрупкости квантовых данных. Квантовые данные настолько хрупкие, что, если вы даже смотрите на него, вы почти наверняка повредите наблюдаемую информацию. Таким образом, квантовые компьютеры должны быть спроектированы с учетом квантовой ошибки, так что блуждающие взаимодействия из его физической среды не наносят непреднамеренного повреждения информации и вычислений. По этой причине естественной целью Q# является скорректированный с ошибкой квантовый компьютер (часто называемый отказоустойчивым квантовым компьютером), который принимает список квантовых команд (называемых затворами или затворами) и применяет эти инструкции к сохраненным квантовым данным внутри. Заметим, что если количество операций кубитов и затворов в квантовом алгоритме или программе достаточно мало, исправление ошибок может быть абсолютно не необходимо. Однако по мере увеличения количества операций с кубитами и воротами это, безусловно, будет требовать, поэтому мы будем архитектовать наш стек программного обеспечения и Q#, чтобы эффективно и эффективно обрабатывать исправления ошибок и обеспечивать масштабируемые отказоустойчивые квантовые вычисления [55].

Коррекция ошибок требует, чтобы быстрый и надежный классический компьютер запускался совместно с квантовым компьютером для исправления ошибок, как они появляются в квантовом вычислении. На практике для

идентификации и исправления ошибок быстрее, чем они естественным образом накапливаются в квантовом компьютере, могут потребоваться такие компоненты, как программируемые пользователем вентильные матрицы (FPGA) или быстрые криогенные процессоры. В результате квантовый компьютер представляет собой гибридную машину, состоящую из нескольких различных вычислительных устройств, работающих в широком диапазоне температур. По этой причине гораздо более полезно подумать о программировании квантового компьютера через объектив программного стека, поскольку существует много уровней аппаратного и программного обеспечения (классического и квантового), необходимых для достижения в конечном итоге реализации квантового алгоритма на квантовый компьютер.

Ниже представлен концептуальный стек, который иллюстрирует разложение числа 8704143553785700723 на простые множители в квантовой вычислительной среде:

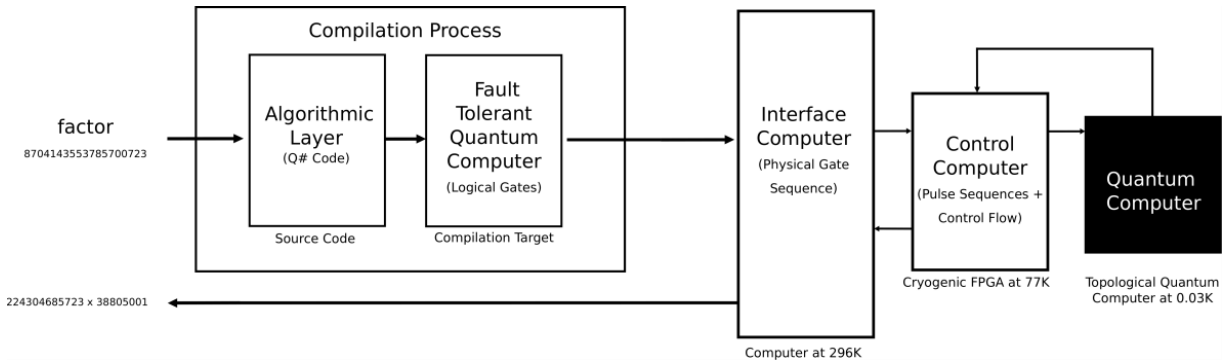

Рисунок 3.1 - Квантовый стек технологий

Существует несколько широких этапов программирования таких квантовых вычислений. Первая и, возможно, наиболее сложная фаза - это задание проблемы, которую вы хотите решить. В этом случае задача состоит в том, чтобы число 8704143553785700723 разложить на произведение двух простых чисел. Следующий шаг предполагает разработку алгоритма для решения этой вычислительной задачи. В этом случае используется алгоритм Шора. Этот алгоритм выражается в Q#, а затем выводится последовательность

квантовых операций, которая может быть запущена на идеализированном беспроблемном квантовом компьютере [55].

Цель Q# - предоставить простой язык, который позволяет разработчикам писать код, который нацелен на множество квантовых вычислительных платформ и интерфейс с промежуточными слоями программного обеспечения, которые стоят между пользователем и квантовым устройством. Язык облегчает это, охватывая понятие стека программного обеспечения и абстрагируя многие детали базового квантового компьютера, позволяя другим уровням стека, подвергаемым через язык, такой как C#, выполнять необходимые переводы из Q# кода в фундаментальный операции. Это позволяет разработчику сосредоточиться на том, что они делают лучше всего: разработке алгоритмов и решении проблем.

Учитывая все достоинства и легкость в установке данного пакета приложений для работы с языком программирования, алгоритм Гровера будет реализован на квантовом языке программирования Q#.

3.2 Язык программирования Haskell

Haskell — функциональный, типизированный язык программирования[38].

Haskell принадлежит к довольно обширному семейству функциональных языков, к которому относятся такие языки, как ML, Lisp, Scala и Clojure. Если у вас есть опыт работы с императивными или объектно-ориентированными языками, например с C, C++ или Java, то вы познакомитесь с новыми идеями, заложенными в семейство функциональных языков[38].

Используемый в этом языке подход к программированию можно свести к пяти пунктам:

- Haskellотносится к семейству функциональных языков.Haskell воплощает понятие чистоты, отделяя код с дополнительными эффектами от остального приложения[38]..

- Модель вычислений языка Haskell основана на концепции лени (laziness)[38]..
- Типы подвергаются статической проверке со стороны компилятора. Кроме того, особенностью Haskell является система типов, которая намного строже и выразительнее, чем обычно[38].
- Подходы к полиморфизму в Haskell основаны на параметричности (parametricity), напоминающей обобщения в Java и C#, и классах типов[38].

Более подробно с достоинствами языка Haskell можно ознакомиться в литературе [38].Рассмотрим более подробно некоторые аспекты языка программирования Haskell

3.2.1 Строгая типизация Haskell

В том или ином виде системы типов присутствуют практически во всех языках программирования. Система типов — это абстракция, в которой значения, способные появляться в процессе выполнения программы, распределяются по категориям и маркируются как типы. Эти типы обычно служат для ограничения возможного множества действий, применяемых к значению. Например, может быть разрешено объединение двух строк, но запрещено их деление.

В общем, эта маркировка может проверяться в двух определенных моментах: во время выполнения (динамическая типизация), как это обычно и бывает при использовании языков со слабой типизацией (loosertyping), допускающих, к примеру, неявные преобразования между целочисленными и строковыми значениями; или во время компиляции (статическая типизация), когда программы перед формированием на выходе целевого кода (обычно машинного кода, или байт-кода) и получения разрешения на запуск должны подвергаться проверке на наличие полностью определенных типов согласно понятиям языка. Haskell относится к этой второй категории: все программы перед выполнением должны пройти проверку на соответствие типов.

Некоторые из языков со статической типизацией, например Java или C#, нуждаются в период выполнения программы в дополнительной проверке типов. В отличие от этого после компиляции Haskell-программы никаких проверок типов больше не проводится, что существенно повышает производительность.

В Haskell очень строгая система типов. Под строгостью в данном случае понимается количество инвариантов, учитываемых в ходе компиляции до того, как будет выдана ошибка при выполнении приложения. Строгость повышает уверенность в коде, прошедшем проверку типов, и в кругах приверженцев языка Haskell нередко раздаются слова: «Раз код откомпилирован, то он будет работать». Строгая типизация порождает такой способ программирования, который называется программированием с ориентацией на типы (type-oriented programming). Обычно программист знает тип разрабатываемой им функции и у него есть общая идея насчет структуры кода. Затем он «заполняет пустые места» теми выражениями, которые вписываются в его представления [38].

3.2.2 Программное обеспечение Haskell

Однако преимущества Haskell не только в самом языке, но и в большом и постоянно расширяющемся множестве инструментов и библиотек, которые могут использоваться с языком.

Для Haskell доступно несколько компиляторов, которые обычно называются в честь того или иного города: на момент написания книги поддерживались компиляторы GHC (http://www.haskell.org/ghc/) — Глазго, UHC (http://www.cs.uu.nl/wiki/UHC) — Утрехта и JHC (http://repetae.net/computer/jhc/). Из них в качестве стандартного компилятора обычно используется GHC, обладающий самым широким набором свойств [38].

Как и у любого другого популярного языка программирования, у Haskell есть собственное онлайновое хранилище библиотек. Оно называется Hackage и доступно по адресу http://hackage.haskell.org/. Hackage отлично интегрируется с Cabal, инструментом для создания проектов на языке Haskell. В

Hackagesуществуют самые разнообразные библиотеки — от биоинформатики и до создания игровых программ, оконных диспетчеров и многого другого[38].

Кроме GHC и Cabal, в этой книге рассматриваются инструментальные средства, предназначенные для оказания помощи разработчикам в плане ускоренного создания высококачественного кода. И в первую очередь это GHC-профайлер. Он применяется для обнаружения утечек памяти и нерационального использования времени. Также мы уделим внимание средствам Hoogle и Haddock, применяемым для просмотра и создания документации [38].Для работы с данным языком нужно обзавестись рабочей копией Haskell на своей системе. Разработчики Haskell озаботились тем, чтобы процесс установки проходил легко и просто, создав дистрибутив HaskellPlatform, содержащий компилятор GHC, систему сборок и библиотек Cabal, а также полный набор библиотек. Чтобы получить HaskellPlatform, нужно перейти по адресу http://www.haskell.org/platform/, после чего выполнить все инструкции, соответствующие используемой операционной системе[38].

Установка под управлением операционной системы Windows осуществляется предельно просто: загружаемый файл является исполняемым, поэтому он сам обо всем позаботится.

На основе вышесказанного можно подвести вывод, что язык Haskell удобен в установке и использует функциональную парадигму программирования.

3.3 Язык программирования Quipper

Развитие понимания модели квантовых вычислений и получение экспериментальных результатов по обработке информации при помощи кубитов не могло не сказаться на том, что данный вопрос начал прорабатываться и с другой стороны.

Несколько коллективов исследователей задались целью спроектировать и реализовать язык программирования, который был бы удобен для описания

квантовых алгоритмов. Исторически первый таким языком стал язык QCL, или Quantum Computation Language (язык квантовых вычислений). Далее были попытки сделать ещё некоторые языки программирования для квантовых вычислений. А к сегодняшнему дню глубокое развитие при поддержке государственных институтов Канады и США получил язык программирования Quipper, который основан на языке Haskell, дополняя его всем необходимым для решения специфических задач модели квантовых вычислений.

Однако не стоит думать, что к настоящему моменту разработано только два языка квантового программирования. Как это ни странно, всё их множество так же можно разделить на языки с императивной основой и на языки функциональные. Так к императивным языкам квантового программирования относятся: квантовый псевдокод (это вообще был первый формализованный язык для квантового программирования, который, однако, остался только на бумаге), уже упомянутый QCL, затем язык Q, и, наконец, язык qGCL (Quantum Guarded Command Language) [8].

С другой стороны, к функциональным языкам квантового программирования относятся следующие: QFC и QPL (оба разработаны П. Селинджером, различаются только синтаксисом — первый является языком квантовых схем), QML (тоже, как ни странно, основан на языке Haskell), квантовое лямбда-исчисление и, собственно, язык Quipper [8].

Следующая таблица представляет сводную информацию о существующих на сегодняшний день языках квантового программирования.

Таблица 3.1 – Квантовые языки программирования

Язык	**Разработчик**	**Парадигма**	**Основан на языке**	**Примечание**
Квантовый псевдокод	Knill E.	И	Псевдокод	Введена модель квантового вычислительного устройства QRAM.
QCL	Ömer B.	И	C	Первый язык, имеющий реализацию.
Q	Mlnařík H.	И	C++	Второй язык, имеющий реализацию
qGCL	Zuliani P.	И	GCL	
QFC	Selinger P.	Ф	—	Графический язык квантовых схем.
QPL	Selinger P.	Ф	—	Текстовый язык квантовых схем.
QML	Altenkirch T. Grattage J.	Ф	Haskell	
Квантовое лямбда-исчисление	Maymin P. Tonder A. Selinger P. Valiron B.	Ф	Лямбда-исчисление	Есть попытка реализации на языке Scheme.
Quipper	Green A.	Ф	Haskell	Язык имеет

Язык	Разработчик	Парадигма	Основан на языке	Примечание
	Lumsdaine P. Selinger P. Valiron B. Ross N.			реализацию компилятора и многих утилит для работы.
Q#		И	—	

Язык программирования Quipper — это самая современная разработка, основанная на всех последних достижениях и идеях относительно модели квантовых вычислений. Этот язык является проблемно-ориентированным языком, базовым для которого является язык Haskell. Это значит, что синтаксисом языка Quipper является синтаксис языка Haskell, а семантика определяется заложенными в проблемно-ориентированные программные сущности правилами [8].

Разработкой языка занимается международный коллектив, в который входят ведущие специалисты в области квантовых вычислений и функционального программирования. Именно функциональное программирование стоит ближе всего к модели квантовых вычислений.

Основная программная сущность языка Quipper — монада Circ, которая представляет квантовую схему. Именно в рамках этой монады производятся квантовые вычисления. Семантика языка Quipper предполагает три типа операций, которые могут быть применены к квантовому регистру (набору кубитов), это — инициализация, унитарное преобразование и измерение. Кубиты же могут быть основными (рабочими) и вспомогательными [8].

Несмотря на то, что язык Quipper реализован как встроенный язык программирования на базе языка Haskell, для него уже разработаны специальные утилиты и инструменты, которые облегчают программисту его труд (в частности, это, конечно же, симулятор квантового вычислительного устройства, но также и всевозможные утилиты для построения квантового

оракула из имеющейся классической функции, подсчёта количества гейтов в квантовой схеме, обращения квантовой схемы и т. д.). Кроме того, в язык внедрено несколько новых служебных слов, необходимых для учёта концепций модели квантовых вычислений [8].

Использование базового языка программирования (в данному случае — языка Haskell), по словам авторов языка Quipper, имеет множество преимуществ (например, развитая инфраструктура), однако несёт и некоторое количество потенциальных проблем. В частности, при разработке квантовых программ есть риск «свалиться» в написание программ на языке Haskell, не используя специально введённые идиомы языка Quipper, что сделает реализованные алгоритмы непереносимыми.

Кроме того, авторы языка Quipper отмечают, что в рамках языка Haskell им очень недостаёт двух важнейших идиом программирования, которые крайне требуются в модели квантовых вычислений — это линейные типы и зависимые типы (это странно, в языке Haskell есть зависимые типы, по крайней мере, в виде одного из расширений языка — прим. авт.). Это значит, что некоторые свойства квантовых программ придётся проверять на стадии исполнения, хотя они могли бы быть легко проверены на стадии компиляции [8].

Квантовые схемы, реализованные в языке Quipper, являются расширенными по сравнению со схемами, описываемыми в теоретической модели квантовых вычислений. Во-первых, в схемах языка Quipper разработчик может использовать как классические, так и квантовые типы данных, равно как и операции над ними. Классические данные могут управлять квантовыми операциями (но не наоборот, то есть квантовые данные не могут управлять классическими операциями). Как отмечалось выше, кубиты и их регистры могут быть измерены, и в результате измерения квантовые данные преобразуются в классические. Также в схемах могут быть использованы так называемые вспомогательные кубиты, то есть такие кубиты, область работы с

которыми явно определена, а потому их можно инициализировать и высвобождать в определённом интервале времени работы с квантовой схемой.

Однако, прежде чем рассматривать сами задачи, необходимо развернуть инструментарий для непосредственной работы. К этому инструментарию относятся следующие программные средства:

1. Haskell Platform, куда включены компилятор GHC и средство развёртывания приложений и библиотек Cabal [8].
2. Все дополнительные библиотеки, которые необходимы для корректной работы языка Quipper [8].
3. Для операционной среды Windows также потребуются специальные утилиты, которые позволят исполнять POSIX-команды [8].
4. Поставочный пакет самого языка Quipper.

Всё вышеперечисленное можно свободно скачать из сети Интернет. Далее описывается процесс развёртывания среды разработки языка Quipper для операционной системы Windows, при этом я полагаю, что те, кто использует иную операционную систему, вполне способны самостоятельно развернуть всё необходимое для работы [8].

На момент написания работы последней стабильной версией Haskell Platform является 2013.2.0.0, и её можно получить по адресу http://www.haskell.org/platform/. После получения дистрибутива его необходимо установить, следуя стандартному мастеру установки и отвечая на все его вопросы простым нажатием на кнопку «Далее». После установки будет доступен компилятор языка Haskell GHC последней версии, а также огромное множество дополнительных инструментов для разработки [8].

Поскольку Haskell Platform установлен, необходимо добавить все необходимые для языка Quipper библиотеки. Однако перед этим следует актуализировать версию утилиты Cabal, что делается при помощи последовательного запуска в командной строке терминала следующих команд:

```
C:\> cabal update
```

(эта команда актуализирует список доступных для установки пакетов языка Haskell, а также их последние версии), и

```
C:\> cabal install cabal-install
```

(и сама утилита Cabal наверняка рекомендует это сделать после актуализации пакетов). Выполнение этой команды займёт приличное время, и для её работы требуется подключение к сети Интернет, поскольку утилита Cabal самостоятельно скачивает всё необходимое для работы.

Теперь при помощи последовательного вызова в командной строке команды:

```
C:\> cabal install
```

где символ * обозначает следующие библиотеки (написание должно быть именно таким, с точностью до регистра символов):

- random
- mtl
- primes
- Lattices
- zlib
- easyrender
- fixedprec
- newsynth
- containers
- set-monad
- quickCheck

После установки всех необходимых библиотек также надо поставить среду MSYS, инсталляционный пакет для которой можно получить по адресу

http://downloads.sourceforge.net/mingw/MSYS-1.0.11.exe. После установки (рекомендуется использовать путь по умолчанию; по крайней мере, в пути к этому набору утилит не должно быть пробельных символов) необходимо ответить «Нет» на вопрос о наличии MinGW, а также прописать путь к подкаталогу binв переменную окружения PATH. После этого утилиты, необходимые для обеспечения совместимости со стандартом POSIX, будут доступны, и можно будет перейти к развёртыванию самого пакета для языка Quipper [8].

Инсталляционный пакет языка Quipper представляет собой простой заархивированный каталог, который содержит в себе всю поставку всех исходных файлов на языке Haskell, в которых определяется проблемно-ориентированный язык, приводятся многочисленные примеры, а также предлагаются для изучения семь полноценно реализованных квантовых алгоритмов. Скачать этот пакет можно с официальной страницы языка Quipper по адресу http://www.mscs.dal.ca/~selinger/quipper/[8].

Проще всего разархивировать инсталляционный архив в корень диска, в каталог «C:\quipper-0.6». После разархивирования необходимо прописать в переменную окружения PATH путь «C:\quipper-0.6\quipper\scripts» (или соответствующий, если разархивирование пакета было произведено в иной каталог). После этих манипуляций язык Quipper будет доступен для работы. Он поставляется, как и компилятор GHC языка Haskell, с компилятором (quipper) и интерпретатором (quipperi), хотя, конечно же, это просто обвязки в виде BAT-файлов над компилятором GHC и интерпретатором GHCi соответственно.

Чтобы запустить режим интерпретации quipperi необходимо зайти в терминал MSYS, в котором перейти в каталог, где хранятся исходники для работы (например, имя файла Example.hs), после чего выполнить команду[8]:

```
quipperi Example.hs
```

в результате которой будет запущен интерпретатор GHCi с подключением всех необходимых модулей языка Quipper. После этого можно начинать работать так, как можно работать непосредственно в GHCi [8].

Из-за того, что установочный пакет языка Quipper оформлен не через систему развёртывания Cabal, при работе с компилятором и интерпретатором GHC возникают определённые сложности. Компилятор GHC «не видит» библиотек языка Quipper, поэтому если есть потребность запустить именно GHC или GHCi, это необходимо делать с указанием пути к библиотеке языка Quipper. Например:

```
ghci -ic:/quipper-0.6
```

если язык Quipper установлен в каталоге «C:\quipper-0.6».

Надо отметить, что создатели языка программирования Quipper уже задумались над поставкой библиотек языка в стиле Cabal, однако, когда выйдет новый инсталляционный пакет — остаётся неизвестным [8].

Исследовав литературу [8] можно сделать вывод что квантовый язык программирования Quipper является одним из самых последних разработанных языков в данном направлении и является функциональным языком программирования, и требует при установке очень много утилит.

3.4 Описание функциональности компьютерной модели алгоритма Гровера

Рассмотрим такую задачу:

$$f\ x_1, x_2, x_3 = x_1 \& x_2 \& x_3 \qquad (3.1)$$

Эта функция принимает значение 1 только на одном из восьми вариантов входных значений, что и требуется для алгоритма Гровера. Для решения нам требуется построить оракул.

X_1	X_2	X_3	$f(\bar{X})$	$(-1)^{f(\bar{X})}$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	−1

Рисунок 3.2 - Таблица для построения оракула функции

Этой таблице соответствует следующая матрица 8×8.

Добавим некоторое количество полезных для конструирования оракулов, гейтов и служебных функций. В частности, в модуль Gate необходимо добавить три функции, которые будут использоваться очень часто.

Две из них позволяют создавать комплекснозначные матрицы на основе матриц из целых (перечислимых) чисел. Не секрет, что начальные состояния во многих алгоритмах задаются целыми числами, а оракулы часто состоят из значений 0 и 1, поэтому использование в коде комплексных чисел очень загромождает файл и препятствует пониманию. Поэтому две новых функции:

```
vectorToComplex :: Integral a
=> Vector a -> Vector (Complex Double)
vectorToComplex = map (\i ->fromIntegrali :+ 0.0)
matrixToComplex :: Integral a
=> Matrix a -> Matrix (Complex Double)
    matrixToComplex = mapvectorToComplex
```

Первая создаёт комплекснозначный вектор, а вторая — матрицу соответственно. А вот следующая функция используется для разбиения заданного списка на подсписки заданной длины. Раньше она была в составе функции для тензорного произведения матриц, а теперь её надо вынести в отдельное определение и засунуть в модуль Qubit из-за установленной иерархии модулей.

```
groups :: Int -> [a] -> [[a]]
groupsi s | null s = []
| otherwise = let (h, t) = splitAti s
    inh :groups i t
```

В соответствии с описанием алгоритма Гровера, нам потребуется оператор вычитания матриц друг из друга, а также функция для создания гейтов для нескольких кубитов на основе гейта для одного кубита. По крайней мере, эта функция потребуется для тензорного произведения гейтов I и H, чтобы иметь возможность применять их к квантовым регистрам, состоящим из нескольких кубитов.

Ну и для порядка определим такой же оператор для сложения матриц:

```
(<+>) :: Num a => Matrix a -> Matrix a -> Matrix a
    m1 <+> m2 = zipWith (zipWith (+)) m1 m2
```

Теперь перейдём к функциям для генерации гейтов. Пока в модуле Gate были реализованы функции для представления гейтов, обрабатывающих один или максимум два кубита. Однако при помощи оператора тензорного произведения и функции entangle можно создавать функции для представления гейтов, обрабатывающих произвольное количество кубитов. Например, вот так

выглядит обобщённая функция, преобразующая заданный однокубитовый гейт в тот же гейт, обрабатывающий заданное количество кубитов:

```
gateN :: Matrix (Complex Double)
->Int -> Matrix (Complex Double)
    gateN g n = foldl1 (<++>) $ replicate n g
```

При помощи стандартной функции replicate создаётся список из заданного количества однокубитовых гейтов. Далее этот список сворачивается посредством оператора тензорного произведения (<++>). А вот так эта функция используется:

```
gateIn :: Int -> Matrix (Complex Double)
gateIn = gateNgateI
gateHn :: Int -> Matrix (Complex Double)
gateHn = gateNgateH
```

Как видно, первая функция формирует тождественное преобразование для заданного количества кубитов, а вторая — преобразование Адамара опять же для заданного количества кубитов. Эти два многокубитовых гейта очень важны в модели квантовых вычислений и часто используются во всевозможных квантовых схемах.

Запишем оракул:

```
oracle :: Matrix (Complex Double)
oracle = matrixToComplex
[[1, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
```

```
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, -1]]
```

Реализуем функцию для представления оператора диффузии Гровера с учётом тех соображений относительно смены базиса (использовать кубит в квантовом состоянии $|+\rangle$ вместо $|0\rangle$):

```
diffusion :: Matrix (Complex Double)
diffusion = 2 <*> (qubitPlus3 |><| qubitPlus3)
<->gateIn 3
where
qubitPlus3 = toVector $
foldl1 entangle $
replicate 3 qubitPlus
```

Берём три кубита в состоянии $|+\rangle$ (qubitPlus), делаем из них список кубитов и сворачиваем его при помощи функции entangle. Далее переводим полученный квантовый регистр в векторное представление. Так получается локальное определение qubitPlus3.

Затем этот квантовый регистр qubitPlus3 векторно умножаем сам на себя, в результате чего получается матрица $|+\rangle\langle+|$, которая далее умножается на 2. Ну и из результата вычитается единичная матрица, подготовленная для трёх кубитов (то есть размера 8×8).

Реализуем сам алгоритм Гровера:

```
grover :: Matrix (Complex Double) -> IO String
grover f = initial |>gateHn 3
```

```
|> f |> diffusion
|> f |> diffusion
>>> (measure .fromVector 3)
where
initial = toVector $
foldrentanglequbitZero $
replicate 2 qubitZero
```

Поскольку алгоритм Гровера является вероятностным, он выдаёт правильный ответ только с какой-то очень высокой вероятностью. Это значит, что временами при запуске функции grover мы будем получать неправильный ответ, поэтому в данном случае предлагается оценить вероятность получения правильного ответа. Реализуем такую функцию:

```
main f n = do l <- replicateM n $ grover f
return $
    map (length &&& head) $ group $ sort l
```

Она применяет алгоритм Гровера для заданного оракула заданное количество раз, а результатом её работы является гистограмма результатов алгоритма.

В данной главе была рассмотрена компьютерная модель алгоритма Гровера на языке Q# и описаны все известные квантовые языки программирования.

ГЛАВА 4 РЕЗУЛЬТАТЫ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

В этой главе даётся оценка эффективности алгоритма разработанного программного продукта компьютерной модели, и приводятся результаты решения $f\ x_1, x_2, x_3 = x_1 \& x_2 \& x_3$ на квантовом языке программирования Q#.

4.1 Результаты расчета компьютерной модели алгоритма Гровера

В результате компьютерного моделирования алгоритма Гровера на языке программирования Q# была получена гистограмма результатов на рисунке 4.1

Рисунок 4.1 - Гистограмма частот получения результатов в алгоритме Гровера для трёх кубитов.

Правильный результат был получен примерно в 94.5 % случаев, а остальные результаты имеют частоту примерно в 0.78 %. Этого вполне достаточно для того, чтобы иметь возможность запустить алгоритм Гровера три раза и выбрать из этих трёх запусков результат, повторившийся по крайней мере дважды. Полный код программы можно посмотреть в **Приложение А.**

Если оракул будет возвращать фазу −1 на нескольких входных данных, в этом случае алгоритм Гровера тоже работает, однако для нахождения одного правильного ответа из множества требуется намного меньше итераций.

Пусть l — количество значений входных параметров, на которых функция принимает значение 1 (то есть оракул возвращает фазу −1). Тогда для нахождения одного правильного значения с большой вероятностью требуется $\frac{\sqrt{2n}}{l}$ итераций Гровера. Это можно продемонстрировать следующим кодом:

```
oracle' :: Matrix (ComplexDouble)
oracle' = matrixToComplex
[[1, 0, 0, 0, 0, 0, 0, 0],
[0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, -1]]
```

Если запустить функцию main с этим оракулом и дать возможность выполнить построение гистограммы на миллионе запусках, то будет явлена примерно следующая диаграмма.

Правильные ответы получили наименьшую частоту, поскольку сейчас функция grover, которая вызывается из функции main, выполняет две итерации Гровера, а для оракула с тремя правильными ответами необходима одна итерация. Как только выполнено больше итераций, чем требуется по алгоритму, ситуация переворачивается с ног на голову (поскольку у нас происходит переворот относительно среднего). Но в данном конкретном случае одной итерации так же недостаточно, поскольку частотные вероятности

правильных и неправильных ответов будут достаточно близки друг к другу (это резонно, поскольку была сделана только одна итерация).

Рисунок 4.2 - Гистограмма частот получения результатов в алгоритме Гровера для трех кубитов с тремя правильными ответами

В общем разработчик при использовании алгоритма Гровера всегда должен внимательно следить за количеством итераций и не допускать переворота ситуации в противоположную сторону.

4.2 Сравнительный анализ компьютерной модели на Q# и классической компьютерной модели на языке C++

Для сравнительного анализа был реализован алгоритм последовательного поиска(алгоритм нахождения заданного значения произвольной функции на некотором отрезке) на языке программирования C++. Данный алгоритм описан в Приложение Б.

Рассмотрим компьютерную модель на языке программирования Q# и языке программирования C++ на рисунке 4.3:

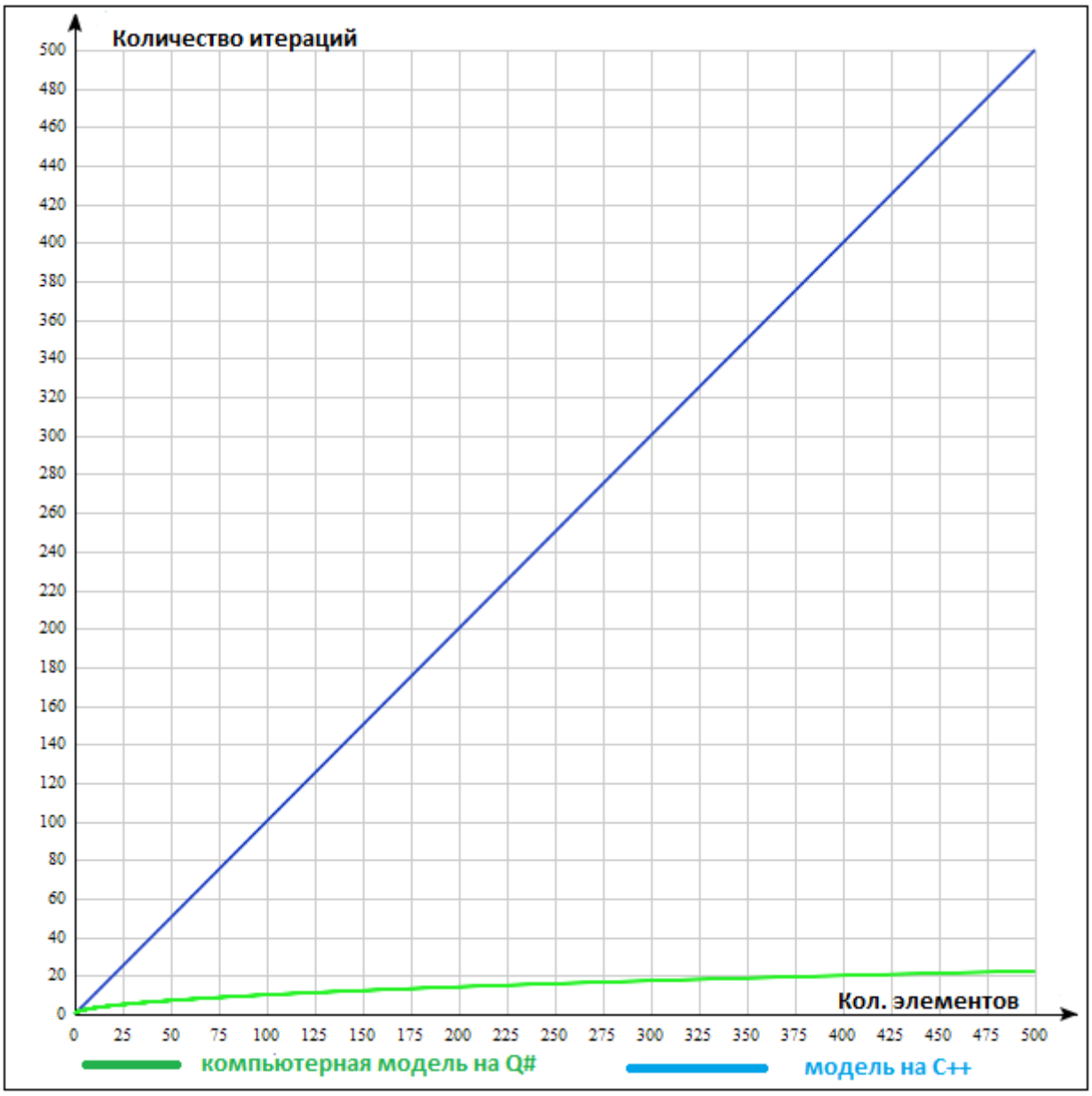

Рисунок 4.3 - Компьютерные модели, разработанные в данной работе

Как видно на рисунке 4.3 компьютерная модель на языке программирования Q# находит ответ на за $\overline{n}$ итерации , а модель на языке программирования C++ за n итераций , где n- количество элементов массиве.

В данной главе были рассмотрены результаты компьютерных моделей на языке программирования Q# и C++ , которые решают задачу поиска в неотсортированном массиве и был произведен сравнительный анализ результатов компьютерных моделей.

ЗАКЛЮЧЕНИЕ

Для реализации модели квантовых вычислений вполне можно обойтись «обычным компьютером», однако, конечно же, программы на нём будут исполняться несоизмеримо долго, занимая огромное количество памяти. Всё, что необходимо реализовать, — это векторная и матричная алгебра, причём элементами векторов и матриц являются комплексными числами.

Наиболее приемлемой для реализации модели квантовых вычислений видится парадигма функционального программирования, поскольку все ключевые понятия этой новой вычислительной модели естественным образом ложатся на идиомы функционального программирования.

Подводя итоги, было сделано следующее:

1. Исследованы методы решения задачи поиска данных;
2. Проанализированы квантовые алгоритмы поиска, приводилось сравнение с классическими алгоритмами;
3. Описаны модели квантового вычисления;
4. Рассмотрена и описана математическая модель квантового алгоритма Гровера;
5. На основе математической модели, была преобразована компьютерная модель посредством языка программирования Q#;
6. Была получена гистограмма результата алгоритма;

В результате компьютерного моделирования полученных результатов, алгоритм Гровера давал правильный результат в 94.5 % случаев. Этого вполне достаточно для того, чтобы иметь возможность запустить алгоритм Гровера три раза и выбрать из этих трёх запусков результат, повторившийся по крайней мере дважды.

С научной точки зрения, показано, как применение квантовых алгоритмов поиска помогает при решении задачи поиска данных и на сколько эффективно.

Образовательным аспектом исследовательской работы является то, что разработанная программа, поможет студентам быстрее освоить навыки математического и компьютерного моделирования квантовых вычислений.

Созданная и описанная компьютерная модель алгоритма Гровера имеет недостатки. В ней не реализовано много того, что должно быть сделано для полноценного описания модели квантовых вычислений на каком-либо языке программирования. Например, из самого банального, — здесь не реализована возможность применения гейта к части кубитов в квантовой схеме

Не реализована работа со вспомогательными битами, не созданы операции для инициализации кубитов и их уничтожения. Однако главная цель достигнута — разработана компьютерная модель алгоритма Гровера на языке программирования Q#.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

1. ГОСТ 7.1-2003. Библиографическая запись. Библиографическое описание документа.

2. ГОСТ 7.32-2001. Отчет о научно-исследовательской работе. Структура и правила оформления

3. ГОСТ 7.82-2001. Библиографическая запись. Библиографическое описание электронных ресурсов.

4. ГОСТ Р 7.05-2008 Библиографическая ссылка.

5. ГОСТ 9327-60 Бумага и изделия из бумаги. Потребительские форматы.

6. ГОСТ 2.105-95 Единая система конструкторской документации. Общие требования к текстовым документам.

7. ГОСТ 6.38-90 Унифицированные системы документации. Система организационно-распорядительной документации. Требования к оформлению документов.

Учебники и учебные пособия

8. Душкин Р.В. Квантовые вычисления и функциональное программирование. / Р.В. Душкин — 2014. — 318 с., Ил.

9. Сысоев С.С. Квантовые вычисления / С.С. Сысоев —2017. — 16 с., ил.

10. Конова Е.А. Алгоритмы и программы. Язык С++: Учебное пособие. — 2-е изд., стер. / Е.А. Конова, Г.А. Поллак Г.А. — СПб.: Издательство «Лань», 2017.—384 с.: ил.

11. Томас К. Алгоритмы построение и анализ: 3-е изд. / К. Томас, Л. Чарльз, Р. Рональд, Ш. Клиффорд.- Пер.с англ. – М. ООО «И.Д. Вильямс», 2013.- 1328с.:ил . – Парал. тит. англ.

12. Хайнеман, Д. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд./ - Д. Хайнеман, П. Гэри,С. Стэнли.: —СпБ.: ООО “Альфа-книга”, 2017. — 432 с .: ил. — Парал. тит. англ.

13. Ричард Б. Жемчужины проектирования алгоритмов: функциональный подход / Пер. с англ. В. Н. Брагилевского и А. М. Пеленицына. - М.; ДМ К Пресс, 2013. — 330 с.: ил.

14. HenryS. Warren, JR. Алгоритмические трюки для программистов, 2-е изд.: Пер с англ.— М.:ООО «И.Д.Вильямс», 2014 – 512 с.:ил. —Парал. тит. англ.

15. Бхаргава А. Г. Изучаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих./А.Г. Бхаргава - СПб.: Питер, 2017. - 288 с. : ил.

16. ДЖ. Макконелл ,Основы современных алгоритмов 2-е дополненное издание/ Перевод с английского под редакцией С.К. Ландо Дополнение М.В.Ульянова - Москва: Техносфера, 2004. - 368с.

17. Игошин В.И. Теория алгоритмов: Учеб. пособие./ В.И. Игошин — М .: И Н Ф РА -М , 2016. — 318 с

18. Клейнберг. Дж Алгоритмы: разработка и применение. Классика ComputersScience /Дж. Клейнберг, Е. Тардос: — Пер. с англ. Е. Матвеева. — СПб.: Питер, 2016. — 800 с.: ил. — (Серия «Классика computer science»).

19. Лафоре Р. Структуры данных и алгоритмы в Java. Классика Computers Science. 2-еизд./ Р. Лафоре — СПб.: Питер, 2013. — 704 с.: ил.

20. Седжвик.Р, Алгоритмы на Java, 4-е издание / Р. Седжвик, К. Уэйн : - Пер. с англ. — М.: ООО «И.Д. Вильямс», 2013 .— 848 с.: ил. — Парал. тит. англ.

21. Стивенс Р, Алгоритмы. Теория и практическое применение / Род Стивенс. — Москва: Издательство «Э» , 2016. —544 с.

22. Скиена С. Алгоритмы. Руководство по разработке. – 2-е изд. / С.Скиена — СПб.:БХВ-Петербург,2011.— 720 с.: ил

23. Кормен. Т.Х. Алгоритмы: вводный курс./ Т.Х Кормен Пер. с англ. - М.: ООО "И.Д. Вильяме", 2014. - 208 с. : ил. - Парал. тит. англ.

24. Бахвалов, Н.С. Численные методы [Текст]: учебное пособие / Н.С. Бахвалов. - М.: Наука, 2006. 631 с.

25. Боев, В.Д. Компьютерное моделирование. Пособие для курсового и дипломного проектирования [Текст]: учебное пособие / В.Д. Боев, Д.И. Кирик, Р.П. Сыпченко. — М.: Военная Академия связи, Санкт-Петербург, 2011. — 348 с.

26. Бураков, П.В., Косовцева, Т.Р. Информатика. Алгоритмы и программирование. [Текст]: учебное пособие / П.В. Бураков, Т.Р. Косовцева. — Санкт-Петербург: НИУ ИТМО, 2014. — 83 с.

27. Воеводин, В. В. Параллельные вычисления: учебное пособие / В.В. Воеводин, В. В. Воеводин — Санкт-Петербург: БХВ-Петербург, 2002. — 608 с.

28. Вороненко, Б.А. Введение в математическое моделирование [Текст]: учебно-методическое пособие / Б.А. Вороненко и др. —СПб.:НИУ ИТМО; ИХиБТ, 2014. — 44 с.

29. Гринь, А.Г. Вероятность и статистика [Текст]: учебное пособие/ А.Г. Гринь.— Омск: Омский государственный университет, 2013.— 304 с.

30. Карпов, В.Е. Численные методы, алгоритмы и программы. Введение в распараллеливание [Текст]: учебное пособие для вузов / В.Е. Карпов, А.И. Лобанов. — М.: Издат-во Москов. физ.-технич. института, 2014. — 190 с.

31. Колесов, Ю.Б. Компонентные технологии математического моделирования [Текст]: учебное пособие / Ю.Б. Колесов, Ю.Б. Сениченков. — Санкт-Петербург: Издательство Политехнического университета, 2013. — 233 с..

32. Петров, В.Ю. Информатика. Алгоритмизация и программирование. [Текст]: учебное пособие / В.Ю. Петров. — Часть 1 — Санкт-Петербург: Университет ИТМО, 2015. — 91 с.

33. Рендольф, Н. Microsoft Visual Studio 2010 для профессионалов [Текст]: учебник / Н. Рендольф, Д. Гарднер, М. Минутилло, К. Андерсон. — М.: MicrosoftPress, 2011. — 1184 с.

34. Сизова, Т.М. Статистика [Текст]: учебное пособие / Т.М. Сизова. — Санкт-Петербург: НИУ ИТМО, 2013. — 176 с.

35. Чивилихин. С.А. Квантовая информатика [Текст]: учебное пособие/С. А. Чивилихин.— СПб:СПбГУ ИТМО , 2009. – 80с.

36. Горбачев В.Н., Учебное пособие по квантовой телепортации, квантовым вычислениям и другим вопросам квантовой информации/ В.Н. Горбачев , А.И. Жилиба[Текст]: учебное пособие/—СПб.: БХВ-Петербург,2001

37. Нильсен М., Квантовые вычисления и квантовая информация / М. Нильсен , И. Чанг. — Пер. с англ— М:Мир, 2006 г.- 824с.

38. А.Мена Изучаем Haskell. Библиотека программиста./ Мена. А. — СПб.: Питер, 2015. — 464 с.:

Электронные ресурсы

39. Астафьев О. В., Устинов, А.В., Впервые появившийся сверхпроводящий кубит / О.В Астафьев, А.В Устинов [Электронный ресурс]: http://www.rqc.ru/news/?ELEMENT_ID=1087

40. История о квантовых компьютерах и о том, изменят ли они нашу жизнь [Электронный ресурс]:https://geektimes.ru/company/ua-hosting/blog/247424/

41. D-wave начала продажи 2000-кубитного квантового вычислителя [Электронный ресурс]:https://nplus1.ru/news/2017/01/26/d-wave-2000

42. Язык программирования для квантовых компьютеров Q# от Microsoft [Электронный ресурс]: http://digitrode.ru/articles/

43. Создан 50-кубитный квантовый компьютер [Электронный ресурс]: http://www.cnews.ru/news

44. Аль-Рефан В.А. Разработка математической модели конкурентных процессов / В.А. Аль-Рефан, И.А. Наумейко, 2014. [Электронный ресурс]:

http://cyberleninka.ru/article/n/razrabotka-matematicheskoy-modeli-konkurentnyh-protsessov

45. D-wave начала продажи 2000-кубитного квантового вычислителя [Электронный ресурс]:https://nplus1.ru/news/2017/01/26/d-wave-2000

Литература на иностранном языке

46. Herlihy, M. The Art of Multiprocessor Programming. / M. Herlihy, N. Shavit. —Revised 1st Edition. — Morgan Kaufmann, 2012. — 537 p.

47. Herlihy M., Shavit N. The Art of Multiprocessor Programming. – Revised 1st Edition. — Morgan Kaufmann, 2012. – 537 p.

48. € 135 miljoen for on twikkeling super computer [Электронный ресурс]:https://www.rijksoverheid.nl/actueel/nieuws/

49. Klint Finley. Quantum computing is real, and D-WAVE just open-sourced it.[Электронный ресурс]:https://www.wired.com/2017/

50. Learn about IBM Q and quantum computing.[Электронный ресурс]:https://www.research.ibm.com/ibm-q/

51. Edward F.Invariant Quantum Algorithms for Insertion into an Ordered List /, F. Edward , G. Jerey ,G. Sam, S. Michael,[Электронный ресурс]:https://arxiv.org/pdf/quant-ph/

52. PellandP., Haines K. Moving to Microsoft Visual Studio 2010 – Microsoft Press, 2011. –336 p.

53. Microsoft Quantum Development Kit[Электронный ресурс]:https://www.microsoft.com/en-us/quantum/development-kit

54. A Preview of Bristlecone, Google's New Quantum Processor [Электронный ресурс]: https://ai.googleblog.com/2018/03

55. Microsoft Quantum Development Kit[Электронный ресурс]:https://docs.microsoft.com/en-gb/quantum/?view=qsharp-preview

56. Andrew M. C. Optimal quantum adversary lower bounds for ordered search. / M.C. Andrew, Troy L. —2007. —16 с., ил.[Электронный ресурс]:https://arxiv.org/pdf/0708.3396.pdf

57. Michael B.O Quantum search in an ordered list via adaptive learning. / - B.O Michael, Aviatan. H—2007.—10 c., ил.[Электронный ресурс]:https://arxiv.org/pdf/quant-ph/0703231.pdf

58. D-Wave Systems announce samulti-year agreement to provide its technology to Google, NASA. [Электронный ресурс]: https://finance.yahoo.com/news/

59. Jamie Condliffe. Google's Quantum Dream May Be Just Around the Corner. [Электронный ресурс]: https://www.technologyreview.com/s/602283/

ПРИЛОЖЕНИЕ А

```
{-# OPTIONS_HADDOCKprune, ignore-exports #-}

{-------------------------------------------------------------------------
------}
{- | Модуль с описанием функций для реализации квантового алгоритма
Гровера.

-}
{-------------------------------------------------------------------------
------}

moduleGrover
(
grover,
main
)
where

{-[ СЕКЦИЯИМПОРТА ]---------------------------------------------------
-----}

importControl.Arrow((&&&))
importControl.Monad(replicateM)
importData.Complex(Complex,realPart)
importData.Function(on)
importData.List(sort,group)

importCircuit
importGate
importQubit

{-[ ФУНКЦИИ ]---------------------------------------------------------
------}

-- | Специально подготовленный оракул для демонстрации алгоритма
Гроверана
```

```
--    трёхкубитах.
oracle::Matrix(ComplexDouble)
oracle=matrixToComplex[[1,0,0,0,0,0,0,0],
[0,1,0,0,0,0,0,0],
[0,0,1,0,0,0,0,0],
[0,0,0,1,0,0,0,0],
[0,0,0,0,1,0,0,0],
[0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,-1]]

-- | Ещё один оракул, который соответствует функции, возвращающей 1 для
--    нескольких (трёх) значений.
oracle'::Matrix(ComplexDouble)
oracle'=matrixToComplex[[1,0,0,0,0,0,0,0],
[0,-1,0,0,0,0,0,0],
[0,0,1,0,0,0,0,0],
[0,0,0,1,0,0,0,0],
[0,0,0,0,-1,0,0,0],
[0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,-1]]

-- | Функция, реализующая гейт диффузии.
diffusion::Matrix(ComplexDouble)
diffusion=2<*>(qubitPlus3|><|qubitPlus3)<->gateIn3
where
qubitPlus3=toVector$foldl1entangle$replicate3qubitPlus

-- | Основнаяфункциямодуля, демонстрирующаяалгоритмГроверанатрёхкубитах.
grover::Matrix(ComplexDouble)->IOString
groverf=initial|>gateHn3
|>f|>diffusion
|>f|>diffusion
>>>(measure.fromVector3)
where
initial=toVector$foldrentanglequbitZero$replicate2qubitZero
```

```
-- | Главнаяфункциямодуля, котораястроитгистограммурезультатовизмерения
--   квантового регистра, запуская алгоритм Гровера заданное количество
раз.
mainfn=dol<-replicateMn$groverf
return$map(length&&&head)$group$sortl
{-[ КОНЕЦ МОДУЛЯ ]---------------------------------------------------------
------
```

ПРИЛОЖЕНИЕ Б

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h> #include <stdlib.h> // для переключения на русский
#define _CRT_SECURE_NO_WARNING
#include <stdio.h>
#include <stdlib.h> // для переключения на русский язык - функция
system() // Функция поиска в массиве k размерности n // элемента со
значением key
int search(int *k, int n, int key)
{
for (int i = 0; i < n; i++) // просматриваем все элементы в цикле
{
       if (k[i] == key) // если находим элемент со значением key,
       return i; // возвращаем его индекс } return -1; // возвращаем -1 -
элемент не найден
}
int main()
{
       int k[8]; // описываем массив из 8 элементов
       int point; // индекс элемента, равного указанному значению (3)
       system("chcp 1251"); // переходим на страницу 1251 для поддержки
     русского языка
       system("cls"); // очищаем окно консоли
       // В цикле вводим элементы массива
       for (int i = 0; i<8; i++)
       {
              printf("Введите k[%d]: ", i);
               scanf("%d", &k[i]);
       }
// Вызываем функцию поиска в массиве элемента, равного 3
       point = search(k, 8, 3);
     if (point == -1) // если функция вернула -1, такого элемента в
     массиве нет
     printf("Элементов равных 3 в массиве нет!\n");
     else // иначе выводим полученный индекс элемента
     printf("Элемент с индексом %d равен 3", point);
     getchar();
```

```
getchar();
return 0; }
```