Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки, факультет радіоелектроніки та телекомунікацій
(повне найменування інституту, факультету)

Кафедра «Захист інформації»
(повне найменування кафедри)

Пояснювальна записка

до дипломної проекту (роботи)

магістр
(ступінь вищої освіти)

на тему Дослідження квантових методів криптоаналізу

Виконав: студент 5 курсу, групи РТ-811м
Спеціальності 125 Кібербезпека
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Безпека інформаційних і комунікаційних систем
Аль-Хамад Н.А.
(прізвище та ініціали)
Керівник Неласа Г.В.
(прізвище та ініціали)
Рецензент Самойлик С.С.
(прізвище та ініціали)

2022

Форма № 23

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет Інститут інформатики та радіоелектроніки, Факультет радіоелектроніки та телекомунікацій
Кафедра захисту інформації
Ступінь вищої освіти магістр
Спеціальність 125 «Кібербезпека»
(код і найменування)
Освітня програма (спеціалізація) Безпека інформаційних і комунікаційних систем
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Завідувач кафедри Воскобойник В.О.,
кандидат технічних наук, доцент
«10» 12 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНУ ПРОЕКТ (РОБОТУ) СТУДЕНТА

Аль-Хамада Науараса Аднановича
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження квантових методів криптоаналізу
керівник (роботи) Неласа Ганна Вікторівна, к.т.н, доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «08» листопада 2022 року № 371

2. Строк подання студентом роботи 10 грудня 2022 р.

3. Вихідні дані до роботи Квантові обчислення, логічні квантові вентилі, квантове перетворення Фур'є, алгоритм Шора, Qiskit.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Розглянути можливі квантові алгоритми, які є небезпечними для криптографії; проаналізувати математичну логіку квантових обчислень і алгоритмів; розробити в квантовому середовищі програму факторизації чисел; проаналізувати залежність кількості кубітів від числа факторизації і проаналізувати час виконання квантового алгоритму факторизації Шора.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Ілюстративний матеріал та презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1, 2, 3	Неласа Г. В., к.т.н., доцент кафедри ЗІ	13.09.22	
нормоконтроль	Корольков Р. Ю., старш. викл. кафедри ЗІ		

7. Дата видачі завдання «13» вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітки
1	Складання та затвердження завдання на дипломну роботу.	13.09.22 – 19.09.22	
2	Підбір літератури. Збір інформації, що відповідає отриманому завданню.	20.09.22 – 26.09.22	
3	Аналіз квантових обчислень і алгоритмів	27.09.22 – 03.10.22	
4	Аналіз робочого середовища для виконання програм з використанням квантових обчислень	04.10.22 – 24.10.22	
5	Розробка програми факторизації невеликих чисел в квантовому середовищі	25.10.22 – 07.11.22	
6	Дослідження залежності кількості кубітів від числа і час виконання алгоритму факторизації Шора	08.11.22 – 14.11.22	
7	Формулювання висновків.	15.11.22 – 21.11.22	
8	Оформлення пояснювальної записки.	22.11.22 – 28.11.22	
9	Оформлення графічної частини, презентації.	29.11.22 – 05.12.22	

Студент _____ Аль-Хамад Н.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Неласа Г.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

ПЗ: 78 сторінок, 53 рисунки, 3 таблиці, 32 джерела.

Мета роботи – дослідження вразливостей асиметричних криптографічних алгоритмів з використанням методів квантового криптоаналізу. Розробка квантової програми факторизації чисел з використанням алгоритму Шора в IBM Qiskit з використанням трьох реальних кубітів.

Об’єкт дослідження – процеси класичних прямих криптографічних перетворень та квантового криптоаналізу алгоритмів асиметричної криптографії.

Предмет дослідження – квантові алгоритми факторизації та дискретного логарифмування, які можна використовувати для криптоаналізу алгоритмів класичної криптографії.

Методи дослідження – теорія квантових обчислень, криптографія та криптоаналіз, програмування.

В ході роботи розроблено програму факторизації чисел з використанням квантового алгоритму Шора, дослідження необхідної кількості кубітів і часу її виконання у порівнянні з відповідною функції IBM Qiskit, яка має більше обмежень до вихідних даних.

АЛГОРИТМ ШОРА, ЗАПЛУТАНІСТЬ КУБІТІВ, КВАНТОВЕ ПЕРЕТВОРЕННЯ ФУР’Є, КВАНТОВІ АЛГОРИТМИ, КВАНТОВІ ВЕНТИЛІ, КВАНТОВІ ОБЧИСЛЕННЯ, КРИПТОАНАЛІЗ, КРИПТОГРАФІЯ, КУБІТИ

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КПФ – Квантове перетворення Фур’є

КРК – Квантовий розподіл ключів

МН – Машинне навчання

ОКФ – Оцінка квантової фази

ШІ – Штучний інтелект

AES (Advanced Encryption Standard) – симетричний блоковий шифр для захисту секретної інформації

ECDH (Elliptic Curve Diffie-Hellman) – прокол, який використовує групову властивість еліптичних кривих для встановлення спільного секретного ключа, не надсилаючи його безпосередньо один одному

ECDSA (Elliptic Curve Digital Signature Algorithm) – складний криптографічний алгоритм шифрування з відкритим ключем

IBM (International Business Machines Corporation) – глобальна технологічна компанія, яка надає апаратне, програмне забезпечення та хмарні послуги

IBM Q (IBM Quantum Computing) – технологія квантових обчислень

Qiskit – програмна основа, що фінансується IBM, щоб дати змогу суспільству увійти у світ квантового обчислення

RSA (Rivest-Shamir-Adleman) – алгоритм асиметричного шифрування

ВСТУП

Актуальність дослідження. Зі стрімким розвитком науки і технологій люди, як правило, мають більший попит на комп’ютери. Класичні комп’ютери більше не можуть задовольнити частину обчислювальних потреб, тому квантові інформаційні системи привертають все більше уваги. Квантові комп’ютери мають хороший паралелізм, високу швидкість обробки інформації, великий обсяг зберігання інформації та продуктивність у багатьох сферах значно перевершують класичні комп’ютери. Класичні комп'ютери, засновані на схемах, отримали широке застосування. Наукові дослідження, військова сфера, економіка та життя все більше залежать від комп’ютерів. Зі створенням і вдосконаленням квантової механіки було зроблено багато областей, які поєднуються з квантовою механікою, і були зроблені прориви. Деякі властивості квантової механіки придатні для обробки інформації. Так народився міждисциплінарний предмет квантової механіки та існуючих інформаційних технологій, який отримав назву квантової інформатики. Квантовий комп'ютер є частиною квантової інформатики. Вони перевершують класичні комп'ютери за швидкістю паралельної роботи, ефективністю зберігання інформації та безпекою шифрування інформації.

Через деякі принципові відмінності квантові комп’ютери в певних сферах легко перевершують продуктивність класичних комп’ютерів. Це робить численні системи, такі як системи шифрування та передачі інформації, засновані на класичних комп’ютерах, складною позицією. Квантова інформатика має хороші перспективи розвитку, її цінують дослідницькі установи, уряди, підприємства та збройні сили.

Метою дипломної роботи є дослідження квантових обчислень, алгоритмів та створення алгоритму факторизації чисел з використанням алгоритму Шора.

1 КВАНТОВІ ОБЧИСЛЕННЯ. ЇХ НЕБЕЗПЕКА ДЛЯ СУЧАСНОЇ КРИПТОГРАФІЇ

1.1 Класифікація комп’ютерних алгоритмів

Щоб зрозуміти роль квантових комп’ютерів серед сучасних традиційних комп’ютерів, спочатку потрібно дізнатися, як можливо вимірювати продуктивність різних алгоритмів.

В інформатиці алгоритми класифікують за тим, як ресурси, які вони використовують, зростають із розміром вхідних даних. Це називають скалдністю алгоритму.

Складність алгоритмів можна класифікувати за такими часами (рис. 1.1):

– лінійний час (зростає лінійно (пропорційно) з довжиною вхідного числа);
– постійний час (не впливає на довжину введеного числа);
– квадратичний час (зростає з квадратом довжини введеного числа);
– експоненціальний час (час роботи експоненціально зростає з довжиною входу).

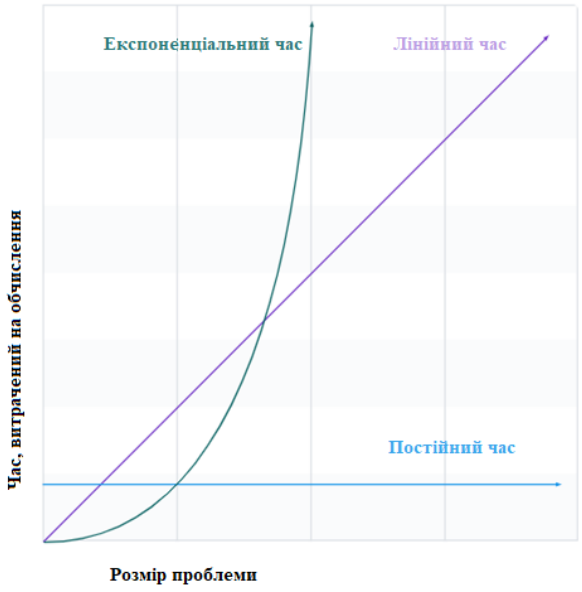

Рисунок 1.1 – Складність алгоритмів

1.2 Виникнення квантових обчислень

У 1994 році Пітер Шор поклав початок розробки квантових обчислень. Шор виявив, що квантовий комп'ютер міг вирішити математичну задачу, яка, у свою чергу, дала б можливість ефективно факторизувати великі числа. Оскільки велика частина сучасного шифрування залежить від припущення, що таке факторування займає надто багато часу навіть для того, щоб усі комп'ютери світу працювали разом, щоб зламати один ключ, розуміння Шора призвело до величезних державних і приватних інвестицій у квантові обчислення. Ненавмисно це також змусило багатьох спостерігачів розглядати квантові обчислення переважно як загрозу шифруванню.

Сьогодні національні держави та навіть приватні компанії змагаються за «перевагу в квантових обчисленнях», квантовий комп'ютер, який є настільки швидким, що може вирішувати проблеми, які неможливо реально вирішити за допомогою класичних комп'ютерів (комп'ютерів, які ми використовуємо щодня). Сполучені Штати, Китай, Європейський Союз та окремі європейські країни (Франція, Німеччина, Великобританія) вкладають мільярди в цю сферу.

Тим не менш, тепер громадськість може придбати акції IonQ Inc., компанії квантових обчислень, яка зареєстрована на Нью-Йоркській фондовій біржі через суперечливу, але все більш поширену практику використання SPAC (спеціальна цільова компанія з придбання). І навіть відносно невеликі стартапи мають робочі, але однаково маленькі квантові комп'ютери. Китайські вчені можуть сміливо стверджувати, що лідирують у розробці квантових обчислень, завдяки рецензованим публікаціям у провідних журналах [1], де детально описується їхня технічна потужність. Одна з китайських машин навіть претендує на «квантову перевагу», хоча вона надзвичайно здатна вирішити проблему, яка практично не цікавить нікого, крім дослідників квантових обчислень та їхніх інвестиційних банкірів [1].

1.3 Квантовий комп’ютер

Квантові комп’ютери імітують поведінку атомів і субатомних частинок, щоб різко збільшити швидкість обробки. Ці частинки можуть існувати в кількох станах одночасно, таке явище називається квантовою суперпозицією. Атоми, наприклад, можуть бути збуджені, не збуджені, і тим, і іншим. Квантові об’єкти також утворюють нерозривні зв’язки один з одним і впливають на поведінку один одного навіть на великій відстані. Два або більше зв'язаних квантових об'єкти створюють тендітну екосистему, яка називається складеною квантовою системою. Це означає, що якщо один об’єкт у системі порушується, то всі об’єкти, з якими він зв’язаний, також будуть порушені.

Квантові комп’ютери – це зображення складених квантових систем у натуральну величину. У той час як класичні комп’ютери обробляють інформацію, послідовно перемикаючи цифрові перемикачі, що представляють 0 і 1, квантові комп’ютери використовують одиниці, які називаються кубітами, які представляють кілька значень одночасно. Оскільки їм не потрібно обробляти інформацію послідовно, кубіти можуть виконувати обчислення значно швидше, ніж біти, які можуть робити це лише за допомогою дискретних значень. Тому дослідники квантової інформації намагаються заплутати якомога більше кубітів, і коли їм це вдається, обчислювальна потужність квантових комп’ютерів зростає експоненціально.

Однак одна проблема полягає в тому, що перешкоди між заплутаними кубітами можуть призвести до розпаду всієї системи. Характеристики, які роблять квантові системи потужними, також роблять їх тендітними. Це явище, яке вчені називають декогеренцією, створює значні проблеми для надійності квантових комп’ютерів. Сучасні кубіти надзвичайно чутливі до впливів навколишнього середовища, таких як температура та пил, і збої в будь-якій частині складеної системи можуть каскадом поширюватися на всю систему. Через цю крихкість поточний час когерентності – кількість часу,

протягом якого кубіт може зберігати пам’ять перед тим, як піддатися декогерентності – становить менше 1 хвилини [2].

1.4 Стан науки пов’язаної з квантовими обчисленнями

Сьогодні квантові комп’ютери є як у дослідницьких лабораторіях, так і на виробництві. Але навіть найпродуктивніші сучасні пристрої ще не здатні здійснювати атаки на системи шифрування з відкритим ключем, такі як RSA. Вчені досягли стабільного прогресу, але недостатньо, щоб бути реальною загрозою.

У той час як сучасні комп’ютери вимірюються в бітах і байтах, квантові комп’ютери вимірюються в кубітах. Хоча в багатьох популярних описах квантових комп’ютерів сказано, що кубіти одночасно мають значення 0 і 1, це не зовсім так. Натомість кубіти можуть мати значення 0 або 1, але значення не визначається до кінця квантового обчислення.

У той час як звичайні комп’ютери складаються зі схем, які обробляють дані, квантові комп’ютери правильніше розглядати як сукупність невизначених даних – кубітів – які зазнають програми: наприкінці виконання програми вимірюється кожен кубіт. Зазвичай програму потрібно відтворювати кілька разів із багатьма окремими вимірюваннями, перш ніж результат обчислення можна буде дізнатися з високою точністю.

У 2001 році 7-кубітний квантовий комп’ютер, створений групою Ісаака Чуанга в дослідницькому центрі IBM Almaden [3], успішно розклав число 15 на прості множники 3 і 5. Число 15 представлено у двійковій системі чотирма бітами: 1111. Число 15 також є, не випадково, найменшим числом, яке є непростим, непарним і неповним квадратом. Таким чином, це найменше число, яке команда IBM могла б розрахувати на множники. «Квантовий комп’ютер» був заснований на хімічній речовині, яка була спеціально синтезована, щоб мати 7 кубітів.

Після демонстрації IBM інші дослідники розклали більші числа на квантових комп'ютерах. Жоден із цих підходів не зміг розкласти число, недосяжне для звичайного комп'ютера. Більшість розкладених чисел можна розкласти за допомогою ручки та паперу. Наприклад, у 2012 році команда під керівництвом Наньян Сю з Університету науки і технологій Китаю, Хефей, успішно розклала [4] число 143.

Більш свіжі приклади, які привернули увагу громадськості, включають розкладання 7-значного (20-бітного) числа 1 005 973 у січні 2019 року за допомогою спеціального типу квантового пристрою, відомого як машина для відпалу. Це була захоплююча подія, тому що раніше вважалося, що квантові комп'ютери з відпалом не можуть враховувати фактори. Здавалося, це розширило загрозу криптоаналізу. Вчені припустили, що оскільки виробник пристрою різко масштабував його всього за сім років, можливо, машину, здатну розраховувати на множники типи чисел, які використовуються для захисту сучасного комерційного Інтернету, можна буде створити роками, а не десятиліттями.

Поточний рекорд являє собою 13-значне число, 1 099 551 473 989 [5, 6].

1.5 Можливості та ризики для національної безпеки

Через свою чутливість до впливів навколишнього середовища сьогоднішні квантові комп'ютери є дуже нестабільними, і їх потрібно зберігати в дорогих холодильниках, охолоджених до температур, близьких до абсолютного нуля. Маючи 70 кубітів, для сучасних машин ще не вистачає мільйона кубітів, необхідних для того, щоб зробити квантові комп'ютери комерційно життєздатними. Тим не менш, деякі дослідники прогнозують, що така віха може бути досягнута протягом 10 років, і коли вони стануть зрілими, ці технології матимуть безліч бойових, комерційних і стратегічних застосувань.

По-перше, квантові обчислення можуть посилити штучний інтелект/машинне навчання. Квантова технологія може обробляти та виявляти

шаблони в даних швидше, ніж класичні машини, роблячи квантові інструменти ШІ/МН більш точними та масштабованими. Інструменти квантового ШІ, наприклад, можуть надати автономну зброю та мобільні платформи, такі як дрони, з розширеними можливостями зондування, навігації та позиціонування в зонах, де відсутній GPS. Оснащені інструментами квантового ШІ, такі системи можуть також самостійно змінювати курс, щоб уникнути контрзаходів противника.

Квантове обчислення також має потенціал для значного підвищення зв'язку, безпеки та швидкості Інтернету. Так званий квантовий Інтернет з'єднує квантові пристрої за допомогою заплутування. Вчені з Нідерландів [7], наприклад, переплутали три однокубітні пристрої, які успішно обмінювалися інформацією та зберігали інформацію теоретично незламним способом. У великому масштабі ця архітектура, яка використовує квантову криптографію, може створити надбезпечну комунікаційну інфраструктуру, яка захищає підключені до Інтернету пристрої, включаючи критичну інфраструктуру, від кібератак.

Крім того, квантову криптографію та інструменти ШІ можна поєднати для покращення збору та аналізу розвідувальних даних. Розвідувальні служби, оснащені квантовими комп'ютерами, наприклад, можуть за 8 годин [8] або менше, функція, на виконання якої найшвидшим суперкомп'ютерам у світі за допомогою методів грубої сили знадобиться близько 300 трильйонів років. Для виконання цього завдання квантовим комп'ютерам, ймовірно, знадобиться близько 20 мільйонів кубітів, але, виходячи з поточної траєкторії розвитку квантової техніки, такі машини можуть бути доступні протягом 25 років [8]. Інструменти квантового штучного інтелекту тоді дозволять аналітикам відсортувати та розібратися в великих даних, доступних завдяки квантовому дешифруванню.

Тим не менш, змагальне використання квантових комп'ютерів створює кілька ризиків для національної безпеки, особливо якщо прогрес у квантовому дешифруванні випереджає прогрес у квантовому шифруванні. Наприклад, зловмисник із можливостями квантового дешифрування теоретично міг би

отримати доступ до зашифрованої інформації з легкістю, поставивши під загрозу використання сучасної комунікаційної інфраструктури. Для дипломатів це означає, що спілкування між ними та їхніми іноземними колегами більше не буде безпечним [9]. Для представників розвідувального співтовариства квантовий криптоаналіз може викрити найглибші державні таємниці таких країн як США [10], спричинивши кризу. І квантовий криптоаналіз може дозволити супротивникам декодувати цінні комунікації на полі бою, істотно підриває військову стратегію.

Держави також, ймовірно, змагатимуться за контроль над квантовим Інтернетом. Традиційний Інтернет був заснований на наборі спільних стандартів, принципів і протоколів. Однак під час зародження квантового Інтернету країни-союзники не бажали співпрацювати в квантових дослідженнях, а супротивники не дійшли згоди щодо спільних принципів управління квантовою епохою. У час, коли уряди все більше прагнуть регулювати потік інформації [11] та локалізвати дані в межах своїх кордонів, квантові дослідницькі силоси можуть прискорити перехід до формування кількох «міні-інтернетів», які держава контролює у своїх власних інтересах.

1.6 Класифікація квантових алгоритмів

Навіть не маючи доступу до квантових комп’ютерів, можна припускати, як ці комп’ютери можна використовувати для злому поточних схем шифрування. Зокрема, є два алгоритми, які можна використовувати для атаки на криптографію. Це алгоритм пошуку квантового періоду Шора, який можна використовувати лише для криптографії з відкритим ключем, такої як RSA та еліптичної кривої Діффі-Хеллмана (ECDH), і алгоритм квантового пошуку Гровера, який можна використовувати проти будь-яких алгоритмів, таких як блокові шифри, такі як AES або хеш-функції (SHA).

Хоча криптографічна спільнота розуміє широкий вплив цих алгоритмів, точні вимоги до ресурсів завжди досліджуються. Ефекти, які є відомими:

1) Алгоритм Шора може зламати поточні алгоритми з відкритим ключем, такі як RSA, ECDH, ECDSA тощо, і тому ці алгоритми потрібно замінити задовго до появи квантових комп'ютерів.

2) Алгоритм Гровера ефективно скорочує розмір ключа вдвічі, тому такі алгоритми, як AES і SHA, повинні подвоїти розмір ключа, щоб залишатися безпечними в майбутньому.

1.7 Загрози для сучасної криптографії

Більшість порушень безпеки сталася через несанкціонований доступ до даних. Згідно з даними дослідження [12], до березня 2020 року було зламано 832 мільйони записів (рис. 1.2). Відповідно до звіту, опублікованого cybint solution, 62% підприємств стикаються з фішинговими атаками та атаками соціальної інженерії. Починаючи з 2014 року кількість витоків даних зросла на 67%, що становить 2/3 зростання від загальної кількості витоків за останні 5 років з моменту існування даних.

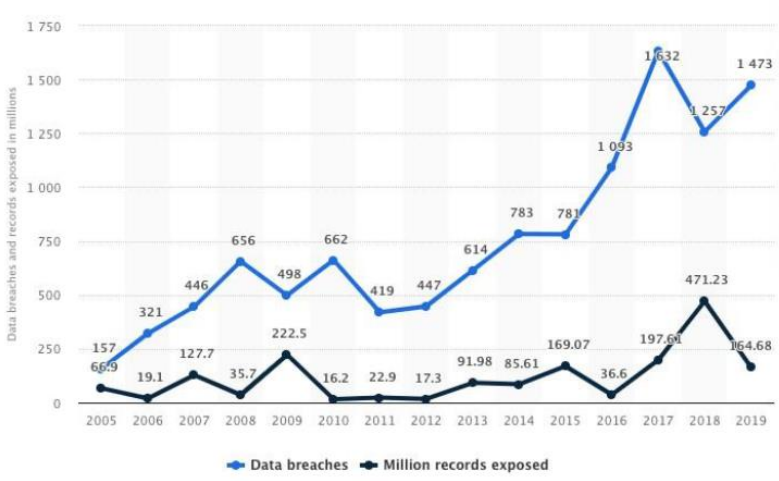

Рисунок 1.2 – Графік виявлення порушень даних і записів [12]

Збільшення витоку даних виникає через технічний прогрес зловмисників. Це допомогло їм використати вразливість мережі, створивши потужне програмне забезпечення, яке дозволяє автоматизувати атаки. У більшості випадків організації не знають про витік даних.

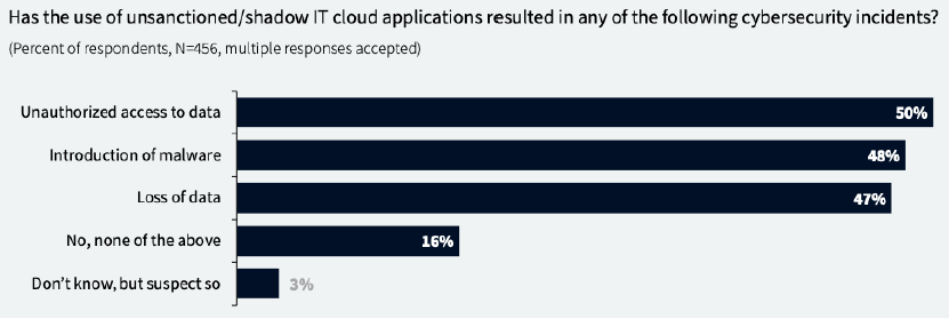

Рисунок 1.3 – Дослідження інцидентів у кібербезпеці [13]

Згідно з дослідженням безпеки [13], понад 75% респондентів втратили гроші через ці атаки. З цієї причини криптографія впроваджується для забезпечення безпечного зв'язку між мережами. Це допомагає компаніям захищати все: від електронної пошти до банківських транзакцій або навіть даних соціальних мереж чи онлайн-покупок.

Квантові обчислення є загрозою для сучасної криптографії з відкритим ключем.

Коли дослідники передбачають крах шифрування, вони описують атаки на системи, засновані на криптографії з відкритим ключем, наприклад RSA з використанням алгоритму Шора. Статистика, яка наводиться вище, зосереджена на атаках RSA.

Криптографія з відкритим ключем рідко використовується сама по собі в сучасних обчислювальних системах. Натомість відкритий ключ використовується для шифрування ключів шифрування Advanced Encryption Standard (AES): саме ключі AES використовуються для шифрування фактичних повідомлень електронної пошти, веб-сторінок, банківських операцій і масових даних на жорсткому диску.

У 1996 році Лов Гровер винайшов спосіб використовувати продуктивний квантовий комп'ютер, щоб значно скоротити час, потрібний для пошуку ключа шифрування AES. Завдяки алгоритму Гровера та ексклюзивному використанню квантового комп'ютера можна зламати 128-бітний ключ AES десь від 5 годин до

одного року, залежно від багатьох деталей. Ось чому багато урядів і виробників модернізували свої системи з AES-128 на AES-256. Оновлення, яке, по суті, є безкоштовним, виводить злом AES за межі можливостей навіть найдосконалішого квантового комп'ютера, який використовує сучасні криптоаналітичні методи.

1.8 Поняття і класифіація сучасної криптографії

У наш час дистанційне спілкування відіграє вирішальну роль у повсякденному житті. Безпечний зв'язок стає все більш важливим у багатьох сферах, наприклад, онлайн-покупки, електронні листи та відеочати.

Криптографія – це практика та вивчення кодування та декодування секретних повідомлень для забезпечення безпечного зв'язку. Існує дві основні галузі криптографії: криптографія з секретним (симетричним) ключем і криптографія з відкритим (асиметричним) ключем.

Ключ – це інформація (параметр), яка керує роботою криптографічного алгоритму. У шифруванні ключ визначає конкретне перетворення відкритого тексту в зашифрований текст або навпаки під час дешифрування. Ключі також використовуються в інших криптографічних алгоритмах, таких як схеми цифрового підпису та коди автентифікації повідомлень.

На практиці через значні труднощі розподілу ключів у криптографії з секретним ключем криптоалгоритми з відкритим ключем широко використовуються в звичайних криптосистемах. Безпеку цих схем шифрування можна підтвердити лише на підставі передбачуваної складності математичної задачі, наприклад розкладання добутку двох великих простих чисел. Жодна схема шифрування з відкритим ключем не може бути захищеною від перехоплювачів з необмеженою обчислювальною потужністю.

Одним із найвідоміших алгоритмів квантових обчислень є алгоритм Шора [14], який може розкладати число N на множники за $O((\log N)^3)$ часу та $O(\log N)$ простору. Алгоритм важливий, оскільки передбачає, що криптографію з

відкритим ключем можна легко зламати, враховуючи досить великий квантовий комп'ютер. RSA [15], наприклад, використовує відкритий ключ N, який є добутком двох великих простих чисел. Одним із способів зламати шифрування RSA є розкладання N на множники, але з класичними алгоритмами розкладання на множники стає все більш трудомістким, оскільки N зростає; точніше, невідомо жодного класичного алгоритму, який би міг факторизувати за час $O((\log N)^k)$ для будь-якого k. Навпаки, алгоритм Шора може зламати RSA за поліноміальний час. Його також було розширено для атаки на багато інших криптосистем із відкритим ключем.

У криптографії шифр Вернама (одноразовий блокнот) – це алгоритм шифрування, у якому відкритий текст поєднується з випадковим ключем або "блокнотом", який має таку ж довжину, як і відкритий текст, і використовується лише один раз. Для поєднання відкритого тексту з блокнотом використовується модульне доповнення. У 1917 році Вернам запропонував схему шифрування – одноразовий блокнот [16]. У 1949 році Шеннон довів, що одноразовий блокнот є інформаційно-теоретично захищеною, незалежно від того, яка обчислювальна потужність доступна для переслухувача [17]. Тобто, якщо ключ справді випадковий, ніколи не використовується повторно та зберігається в секреті, одноразовий блокнот забезпечує повну секретність.

Незважаючи на те, що Шеннон підтвердив свою безпеку, одноразовий блокнот має серйозні недоліки на практиці:

1) алгоритм вимагає абсолютно випадкового ключа;

2) безпечна генерація та обмін ключем повинні бути принаймні такими ж, як і повідомлення.

Ці труднощі впровадження призвели до того, що системи одноразових блокнотів є непрактичними та настільки серйозними, що вони перешкодили застосуванню одноразових блокнотів як широко поширеного інструменту інформаційної безпеки.

Квантова фізика пропонує вирішення двох вищезазначених труднощів для одноразового блокнота. По-перше, природа суперпозиції (невизначеності)

квантової механіки може генерувати справжню випадковість. По-друге, квантова криптографія дозволяє двом віддалених сторонам генерувати безпечні ключі.

1.9 Технологія квантового розподілу ключів

Розподіл ключів – це спосіб розподілу зашифрованих ключів між двома сторонами. Простий спосіб розповсюдження ключів – це особиста зустріч у безпечному середовищі та обмін ключами. Але тепер ми можемо обмінюватися на будь-якій відстані за допомогою відкритих ключів, таких як шифри, RSA, Діффі-Хеллмана та криптографії на основі еліптичних кривих для обміну ключами [18].

Проблеми зі звичайним розподілом ключів полягають у тому, що вони використовують прості математичні обчислення для передачі даних, їх легко обчислити і до них можуть легко отримати доступ сторонні особи [18].

Цей класичний підхід розподілу ключів має багато проблем.

Вони, як і класичний ключ, можуть генерувати лише слабкі випадкові числа, які можуть бути легко отримані сторонніми особами. Крім того, потужність процесора та ці ключі вразливі до нових стратегій атак, оскільки їх потрібно перепрограмувати заново, якщо використовувалися нові стратегії атак. Квантові комп’ютери можуть зробити ці класичні стратегії шифрування небезпечними, оскільки квантові комп’ютери можуть легко декодувати дані, присутні в класичних ключах, якщо квантові комп’ютери стануть реальністю. Отже, щоб залишатися попереду, потрібно використовувати великі асиметричні ключі для безпечного зберігання та розповсюдження симетричних ключів. Усе це змушує переглянути безпеку криптографічних ключів [18].

КРК вирішує ці проблеми, з якими стикаються криптографічні ключі, використовуючи всю квантову механіку для передачі даних або інформації з однієї точки в іншу [18]. КРК використовує власний квантовий канал для передачі даних від передавача до приймача. Йому також потрібне загальнодоступне посилання зв’язку, щоб мати доступ до постобробки. Він

також має портал, який обчислює кількість даних, втрачених через перехоплення [18].

1.10 Поняття квантового криптоаналізу

Квантовий криптоаналіз – це розуміння значення зашифрованої інформації без доступу до вихідної інформації. У цьому можна знайти секретний ключ [19]. Також вивчаються атаки, щоб отримати зашифровану інформацію. Це дослідження атак можна провести шляхом практичного дослідження безпеки. Загалом, практичною проблемою в системі КРК є виявлення лазівок за допомогою криптоаналізу [19].

Криптоаналіз – це дослідження методів отримання значення зашифрованої інформації без доступу до секретної інформації, яка зазвичай потрібна для цього. Як правило, це передбачає пошук секретного ключа.

У квантовому аналогу потрібно розглянути лазівки, які існують у системах КРК і різних стратегіях атак. Вивчення атак має подвійне значення. По-перше, він досліджує безпеку в практичному сенсі. По-друге, це принципово цікаво в квантовій механіці.

Наприклад, загальною фізичною проблемою в практичній системі КРК з двома детекторами є лазівка ефективності детектування [20, 21]. Ця лазівка лежить в основі не тільки прикладної технології, такої як КРК, але й фундаментальної фізики, такої як перевірка нерівності Белла. Крім того, на практиці важко побудувати два детектори, які б мали абсолютно однакові характеристики.

Найбільший виклик, який стоїть перед квантовою індустрією, полягає в тому, щоб привести квантові комп’ютери або їх різноманітні додатки до комерційного та широкого використання, незважаючи на їх обмеження. Основна мета полягає в тому, щоб квантові комп’ютери вирішували невирішені питання, які ми не змогли вирішити.

2 МАТЕМАТИЧНА МОДЕЛЬ КВАНТОВИХ ОБЧИСЛЕНЬ, АЛГОРИТМІВ І КРИПТОАНАЛІЗУ

2.1 Представлення інформації у квантових обчисленнях

Квантові комп'ютери використовують квантові явища, такі як поляризація фотонів, рівні атомної енергії або ядерний оберт для представлення інформації та використовують квантові системи для виконання обчислень. Фундаментальний блок квантових обчислень відрізняється від класичних комп'ютерів. Квантові обчислення – це тип обробки інформації, який використовує принципи квантової механіки, такі як квантова заплутаність і суперпозиція. Головною особливістю квантових обчислень є те, що вони мають імовірнісний характер. Якщо ми намагаємося побачити результати, ми можемо отримати певний результат лише з певною ймовірністю [22].

Основною одиницею інформації у квантовому комп'ютері є кубіти, як і класичні біти. На практиці цей кубіт є надпровідною частинкою, що зберігається в ізольованому середовищі, вільному від декогерентності, при температурі абсолютного нуля. Ця ізольована частинка може бути електроном або фотоном деяких вибраних елементів, які виявляють надпровідність, у якій її спін визначає ймовірність 0 (розкручування вгору) та 1 (обертання вниз). Ми можемо описати кубіт вектором у суперпозиції станів, де вони перекриваються, на відміну від бітів, які можуть мати лише одне значення за раз. Завдяки такій природі суперпозиції кубіт може представляти більше інформації, ніж класичні біти. Він відомий як квантовий паралелізм [22, 23].

З математичної точки зору кубіт – це вектор, що представляє ймовірності 0 і 1 за допомогою коефіцієнтів величини α і β відповідно, які обмежені ймовірністю 1, тобто $\alpha^2 + \beta^2 = 1$. Якщо α досягає 1, то β буде прагнути до 0.

Загальні обчислювальні стани представлені у векторній формі наступним чином:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Таким чином, одиничний вектор суперпозиції станів описує кубіт як лінійну комбінацію.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle.$$

Параметри α і β є комплексними числами, які називаються амплітудами та містять обмеження $|\alpha|^2 + |\beta|^2 = 1$. Стани $|0\rangle$ та $|1\rangle$ утворюють ортонормований базис для векторного простору [23, 24].

2.2 Представлення станів кубіту

Імовірності корисні, коли існує багато можливих результатів, і ми не маємо достатньо інформації, щоб визначити, що станеться, як підкидання кубика чи підкидання монети. Зазвичай, кожному результату надається ймовірність і використовується, щоб визначити ймовірність того, що щось станеться. Імовірності дуже добре працюють для речей, які зазвичай можна побачити в навколишньому світі, але з «квантовими» речами цей підхід не працює, і необхідно оновити його, щоб пояснити таку поведінку.

Отже, для опису квантової механіки можна використовувати амплітуди ймовірностей. Амплітуди ймовірностей подібні до нормальних ймовірностей у тому, що:

- амплітуди мають величину;
- кожен можливий результат має амплітуду ймовірності;
- величина амплітуди цього результату говорить нам, наскільки ймовірно цей результат відбудеться.

Але амплітуди також мають додаткову властивість, яка називається фазою. Фазу можна розглядати як кут, тому в той час, як звичайні ймовірності мають лише величину, амплітуди також мають напрямок (рис. 2.1).

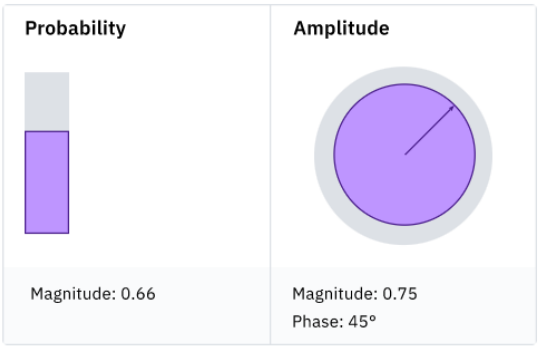

Рисунок 2.1 – Порівняння ймовірності та амплітуди

Оскільки амплітуда має напрямок, її можна віднести до комплексних чисел.

Результатом фази є те, що коли ми додаємо дві з цих амплітуд разом, вони можуть компенсувати одна одну, як це роблять додатні та від'ємні числа. Така поведінка називається інтерференцією і пояснює всю поведінку, характерну для квантової механіки, яку не можливо побачити в класичній механіці.

Щоб знайти ймовірність вимірювання результату, зводимо в квадрат величину амплітуди цього результату (рис. 2.2).

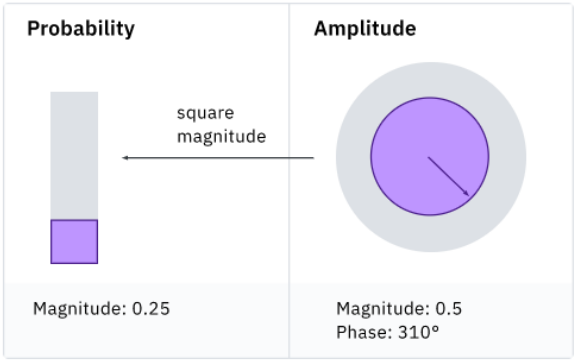

Рисунок 2.2 – Ймовірність вимірювання результату

Фазу не можливо побачити безпосередньо, але вона існує, оскільки створює ефект перешкод.

Таким чином, можна змоделювати поведінку, наприклад, вентиля Адамара (рис. 2.3). Дана модель показує різницю між ймовірністю та амплітудою ймовірності, а також пояснює чому при застосуванні повторно вентиля Адамара результат повертається у вихідний стан.

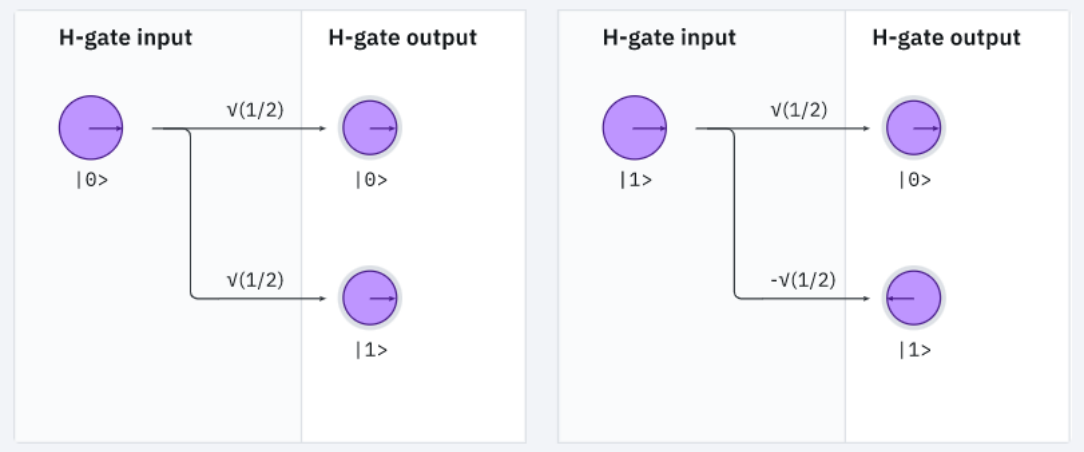

Рисунок 2.3 – Модель гейта Адамара

Але з цією амплітудною моделлю не можна зробити висновок, що кубіт пішов певним шляхом («0-0-0 або 0-1-0»), оскільки не можливо побачити ефект інтерференції. Це змушує людей говорити, що «кубіт може бути 0 і 1 одночасно», що не обов’язково є неправильним способом опису поведінки кубіту, але також не є особливо корисним для людей, які вивчають квантові обчислення. Важливо пам’ятати, що, оскільки ми не бачимо такої поведінки в нашому повсякденному житті, у нас насправді немає спільних слів для цього, тому вчені створили нові слова.

Квантові дослідники використовують математичні об’єкти, які називаються векторами, які для їхніх цілей є просто списками чисел. У кожній точці обчислення можна використовувати вектор, щоб відстежувати амплітуди всіх можливих результатів (рис. 2.4).

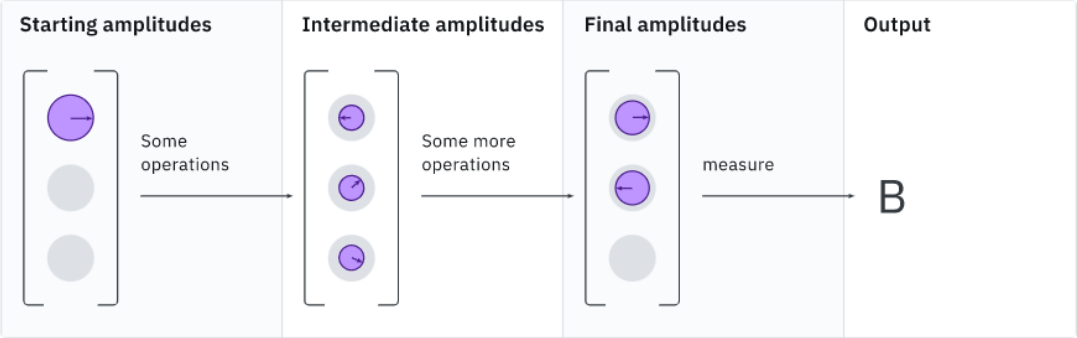

Рисунок 2.4 – Використання вектора у кожній точці обчислення для опису стану системи

Кількість можливих результатів подвоюється з кожним доданим додатковим кубітом, і якщо використовується дана техніка, розмір таких векторів також зростатиме експоненціально. Це не вірно для кожної квантової схеми, наприклад, якщо схема починається з усіма кубітами в стані 0 і ніякі перетворення з ними не проводяться, тоді досить легко передбачити, яким буде результат. Але здається, що складніші квантові схеми можна моделювати лише на класичних комп'ютерах за допомогою алгоритмів, які експоненціально зростають разом із кількістю кубітів у схемі. Верхня межа для моделювання складної квантової схеми, як правило, знаходиться десь між позначкою 30-40 кубітів.

Отже, можна зробити висновок, що:

– неможливо робити біти з об'єктів, які відповідають правилам квантової механіки, і їх називають кубітами;
– ці кубіти можна описати за допомогою амплітуд імовірностей, які схожі на класичні ймовірності, але з фазою (напрямком);
– ці амплітуди можуть компенсувати одна одну (цей ефект називається інтерференцією), і саме це спричиняє раніше незрозумілу поведінку;
– найкращі алгоритми, які зараз існують для моделювання кубітів, використовують експоненціальні ресурси з кількістю кубітів, тому симуляція великої кількості кубітів недоступна для класичних комп'ютерів.

2.3 Вектори станів

У квантовій фізиці використовується вектори стану для опису стану квантової системи.

Кожен елемент у векторі стану містить ймовірність знаходження кубіту в певному місці.

У розділі 2.2 показано, що можливо передбачити поведінку квантової системи, відстежуючи амплітуди ймовірностей для кожного результату в кожній точці квантового обчислення. Також очевидно, що для n кубітів є 2^n можливих результатів, і можна зберігати ці амплітуди в списках довжиною 2^n, які називаються векторами. Оскільки ці вектори описують стан кубітів, вони називаються векторами стану.

Ось приклад вектора стану для квантового комп'ютера з двома кубітами:

$$|x\rangle = \begin{bmatrix} \sqrt{\frac{1}{2}} \\ \sqrt{\frac{1}{2}} \\ 0 \\ 0 \end{bmatrix}.$$

2.3.1 Множення векторів на скаляри

Загальне правило для вектора з N елементів це:

$$s\begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ \cdots \\ e_{N-1} \end{bmatrix} = \begin{bmatrix} s \times e_0 \\ s \times e_1 \\ s \times e_2 \\ \cdots \\ s \times e_{N-1} \end{bmatrix}.$$

Отже, вектор стану |x⟩, який був визначений вище, можна записати більш акуратно:

$$|x\rangle = \sqrt{\frac{1}{2}}\begin{bmatrix}1\\1\\0\\0\end{bmatrix}.$$

2.3.2 Додавання двох векторів

Друге правило стосується додавання двох векторів. Це визначається лише тоді, коли два вектори мають однакову кількість елементів, і дає новий вектор із такою ж кількістю елементів. Ось загальне правило:

$$\begin{bmatrix}a_0\\a_1\\a_2\\\cdots\\a_{N-1}\end{bmatrix}+\begin{bmatrix}b_0\\b_1\\b_2\\\cdots\\b_{N-1}\end{bmatrix}=\begin{bmatrix}a_0+b_0\\a_1+b_1\\a_2+b_2\\\cdots\\a_{N-1}+b_{N-1}\end{bmatrix}.$$

Отже, |x⟩ можна записати у вигляді:

$$|x\rangle = \sqrt{\frac{1}{2}}(|00\rangle + |01\rangle).$$

Значить, будь-який квантовий стан можна записати як суперпозицію цих векторів стану, якщо помножити кожен вектор на правильне число (амплітуду) та скласти їх разом:

$$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle,$$

де a_{00}, ..., a_{11} – амплітуди стану.

Оскільки будь-який вектор можна представити як комбінацію цих чотирьох векторів, це означає, що ці чотири вектори утворюють базис, який називають

обчислювальним базисом. Обчислювальний базис – це не єдина основа. Для одиничних кубітів популярну основу утворюють вектори |+⟩ і |-⟩:

$$|+\rangle = \sqrt{\frac{1}{2}}\begin{bmatrix}1\\1\end{bmatrix},$$

$$|-\rangle = \sqrt{\frac{1}{2}}\begin{bmatrix}1\\-1\end{bmatrix}.$$

2.4 Правила вимірювання

Є просте правило вимірювання. Щоб знайти ймовірність вимірювання стану |ψ⟩ у стані |x⟩:

$$p(|x\rangle) = |\langle x|\psi\rangle|^2.$$

Символи ⟨ та | означають, що ⟨x| – вектор-рядок, символи | та ⟩ означають, що |ψ⟩ є вектором-стовпцем. У квантовій механіці вектори-стовпці називаються кет (ket), а вектори-рядки бра (bra). Разом вони складають нотацію бра-кет. Будь-який кет |a⟩ має відповідний бра ⟨a|, їх можна перетворювати між собою за допомогою сполученого транспонування.

2.5 Наслідки правила вимірювання

Це правило керує тим, як отримується інформація з квантових станів. Тому це дуже важливо для всього, що можна робити в квантових обчисленнях. З цього також відразу випливає кілька важливих фактів.

2.5.1 Нормалізація

Правило показує, що амплітуди пов’язані з ймовірностями. Якщо треба, щоб сума ймовірностей дорівнювала 1, потрібно, щоб вектор стану був правильно нормалізований (). Зокрема, необхідно, щоб величина вектора стану дорівнювала 1:

$$\langle\psi|\psi\rangle = 1.$$

Якщо $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, тоді $|\alpha|^2 + |\beta|^2 = 1$.

2.5.2 Альтернативне вимірювання

Правило вимірювання дає ймовірність p(|x⟩), що стан |ψ⟩ вимірюється як |x⟩. Ніде не сказано, що |x⟩ може бути лише |0⟩ або |1⟩.

Вимірювання насправді є лише одним із нескінченної кількості можливих способів вимірювання кубіта. Для будь-якої ортогональної пари станів можна визначити вимірювання, яке змусило б кубіт вибирати між двома.

2.5.3 Глобальна фаза

Іншим фактором є глобальна фаза вектора стану. Оскільки фаза існує лише завдяки ефектам інтерференції, які вона створює, можливо вимірювати лише різницю фаз. Якби всі амплітуди були повернуті у векторі стану на однакову величину, все одно можна побачити у векторі стану точно таку саму поведінку.

2.5.4 Ефект спостерігача

Амплітуди містять інформацію про ймовірність того, що кубіт буде знайдений у певному стані, але як тільки кубіт вимірюється, ми точно знаємо,

який стан кубіта. Наприклад, якщо вимірюється кубіт у стані суперпозиції, і в результаті отримаємо кубіт в стані |0⟩, якщо ми виміряємо ще раз, є 100% ймовірність знайти кубіт у стані |0⟩. Це означає, що акт вимірювання змінює стан кубітів.

Це називається руйнуванням стану кубіта. Щоб досягти справді квантових обчислень, необхідно дозволити кубітам досліджувати більш складні стани. Тому вимірювання використовуються лише тоді, коли потрібно витягти вихід. Це означає, що всі вимірювання розміщуються в кінці квантової схеми.

2.6 Логічні квантові вентилі

Подібно до класичних логічних вентилів для класичних сигналів, квантові логічні вентилі (або так звані квантові вентилі) використовуються для обробки квантових сигналів і формування квантових обчислювальних систем. На відміну від класичних логічних вентилів, квантові логічні вентилі оборотні і не втрачають вхідних сигналів. Крім того, квантова логіка не має проблем з розсіюванням тепла, що принципово уникає проблеми класичних логічних елементів, які не можуть нормально працювати через недостатнє розсіювання тепла. Відповідно до кількості вхідних кубітів квантові логічні вентилі поділяються на однокубітові та багатокубітові. Базові квантові логічні вентилі складають складні метричні логічні вентилі, і ці логічні вентилі складають всю квантову інформаційну систему [25].

2.6.1 Вентилі Паулі

До вентилей Паулі відносять:

1) X-гейт, який відповідає матриці Паулі:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0|.$$

X-гейт також часто називають NOT-гейтом, маючи на увазі його класичний аналог.

2) Y-гейт, який відповідає матриці Паулі:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = -i|0\rangle\langle 1| + i|1\rangle\langle 0|.$$

3) Z-гейт, який відповідає матриці Паулі:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1|.$$

Вентилі Y і Z є еквівалентом повороту на 180 навколо відповідної осі координат на сфері Блоха.

2.6.2 Вентиль Адамара

Вентиль Адамара є вентилем перетворення над одним кубітом, він відповідає матриці H:

$$H = \sqrt{\frac{1}{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Вентиль Адамара (H-гейт) є фундаментальним квантовим вентилем у квантових обчисленнях. Це дозволяє відійти від полюсів сфери Блоха і створити суперпозицію між $|0\rangle$ і $|1\rangle$. Він має матрицю:

Цей логічний елемент насправді є перетворенням Гільбертового простору від власних векторів $|0\rangle$ і $|1\rangle$ матриці Паулі Z до власних векторів $|+\rangle$ і $|–\rangle$ матриці Паулі X.

Отже, H-гейт виконує над кубітами перетворення:

$$H|0\rangle = |+\rangle,$$
$$H|1\rangle = |-\rangle.$$

2.6.3 Вентиль зсуву фази

Вентиль зсуву фази – це квантовий вентиль, який працює на одному кубіті, він відповідає матриці S:

$$S = \begin{bmatrix} 1 & 1 \\ 0 & e^{\frac{i\pi}{2}} \end{bmatrix}.$$

Він робить чверть оберту навколо сфери Блоха.

2.6.4 Вентиль π/8

Фазовий вентиль з кутом θ = π/8 називається вентилем π/8, це особливий вид вентиля фазового зсуву, який відповідає матриці T:

$$T = \begin{bmatrix} 1 & 1 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix}.$$

Це еквівалентно обертанню сфери Блоха на θ градусів.

2.6.5 Вентиль CNOT

Вентиль CNOT, який можна побачити на рис. 2.5. CNOT є багатокубітним вентилем, який відповідає матриці CNOT:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Функція цього вентиля не змінює керуючий кубіт, але виконує ексклюзивну операцію XOR над керованим кубітом (коли керуючий кубіт дорівнює 0, керований кубіт не змінюється. Коли керуючий кубіт дорівнює 1, керуючий кубіт виконує операцію NOT).

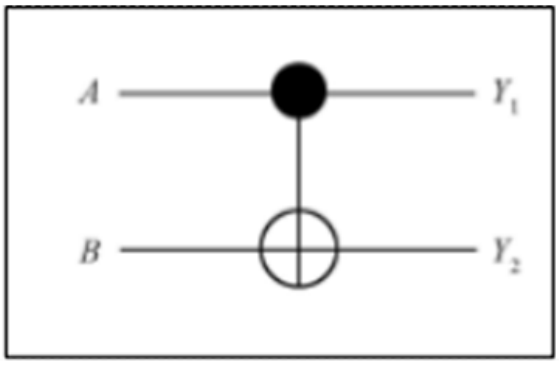

Рисунок 2.5 – Вентиль CNOT, де A – керуючий кубіт, B – керований кубіт

2.6.6 Вентиль SWAP

Вентиль SWAP є багатокубітним вентилем, він відповідає матричному SWAP:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Функція цього вентиля полягає в обміні двома вхідними кубітами. Вентиль SWAP складається з трьох вентилей CNOT. За допомогою цих процесів відбувається обмін двома вхідними сигналами (рис. 2.6 і рис. 2.7) [26].

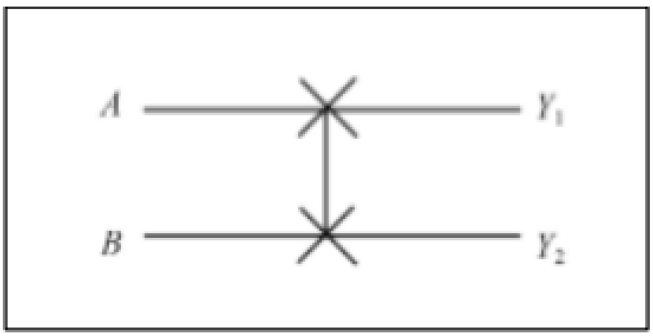

Рисунок 2.6 – Вентиль SWAP

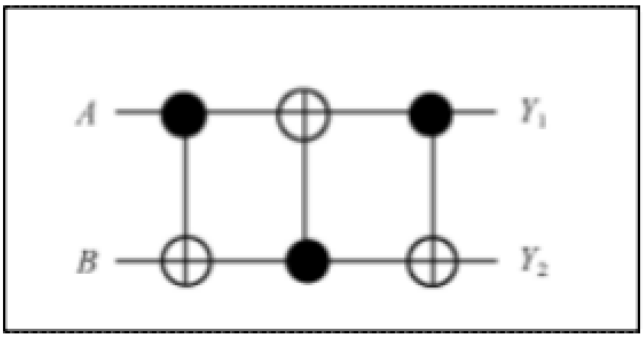

Рисунок 2.7 – Вентиль SWAP з трьох вентилей CNOT

2.7 Заплутаність кубітів

Електрони або кубіти, якщо вони взаємодіють у будь-якій точці, матимуть зв’язок між цими частинками. Якщо будь-які електрони або кубіти демонструють ці особливості, тоді вони називаються, оскільки вони корелюються або вони переплутані разом. Після того, як буде відомий спін одного електрона, інший заплутаний електрон матиме свій спін у протилежному напрямку до іншого електрона.

Через використання квантової суперпозиції частинки не мають спіну чи напрямку до обчислень, але частинка існує альтернативно як у верхньому, так і в нижньому станах або в 0s та 1s станах. Спин частинки вимірюється під час вимірювання, і вона одночасно повідомляється своїй заплутаній частинці, вона має напрямок обертання, протилежний напрямку обертання початкового спіну

вимірюваної частинки. Завдяки цій властивості кубіти можуть взаємодіяти. Незалежно від того, наскільки велика відстань між взаємопов'язаними частинками, вони залишатимуться заплутаними, доки вони ізольовані [27].

Простим прикладом квантової заплутаності кубітів є стан Белла:

$$|\beta_{00}\rangle = \sqrt{\frac{1}{2}} \cdot (|00\rangle + |11\rangle),$$

$$|\beta_{01}\rangle = \sqrt{\frac{1}{2}} \cdot (|01\rangle + |10\rangle),$$

$$|\beta_{10}\rangle = \sqrt{\frac{1}{2}} \cdot (|00\rangle - |11\rangle),$$

$$|\beta_{11}\rangle = \sqrt{\frac{1}{2}} \cdot (|01\rangle - |10\rangle).$$

Стани Белла утворюють ортонормований базис у чотиривимірному гільбертовому просторі, відомий як базис Белла.

2.8 Квантові алгоритми

Квантовий алгоритм – це алгоритм, який використовується в квантових обчисленнях. Цей алгоритм використовує характеристики квантової механіки для роботи, і він має свої переваги перед класичними алгоритмами в багатьох аспектах.

В розділі 1.6 було визначено два алгоритми, які представляють загрозу для сучасної криптографії: алгоритм Гровера і алгоритм Шора.

2.8.1 Алгоритм Гровера

Алгоритм Гровера – високоефективний квантовий алгоритм, який називається алгоритмом квантового пошуку. Цей алгоритм дуже підходить для пошуку даних з багатьох невпорядкованих даних. Часова складність класичного алгоритму дорівнює O(N), тоді як часова складність алгоритму Гровера дорівнює O(√N). Це означає, що при розв'язуванні тієї самої задачі вибір алгоритму Гровера для її розв'язання заощадить багато часу, ніж вибір класичного алгоритму. Алгоритм Гровера має широкий спектр використання і може бути застосований для математичних розрахунків, злому паролів, ядерного магнітного резонансу, оптики тощо [28].

2.8.2 Алгоритм Шора

Алгоритм Шора є дуже ефективним квантовим алгоритмом, який в основному використовується для факторизації великих чисел. Факторизація великих чисел застосована до криптосистеми з відкритим ключем RSA, яка є алгоритмом шифрування, який використовується в багатьох банках і онлайн-рахунках. Алгоритм Шора може перетворити факторизацію великих чисел на обчислення періоду функції. Швидкість обчислення квантового комп'ютера набагато вище, ніж у класичного комп'ютера при обчисленні періоду. Алгоритм Шора дозволяє зламати криптосистему з відкритим ключем RSA, і безпека банків і веб-сайтів, які використовують RSA, буде під загрозою [29].

2.9 Квантове перетворення Фур'є

.

Перетворення Фур'є зустрічається в багатьох різних версіях у класичних обчисленнях, починаючи від обробки сигналів до стиснення даних і закінчуючи теорією складності. Квантове перетворення Фур'є (QFT) – це квантова реалізація дискретного перетворення Фур'є за амплітудами хвильової функції. Він є

частиною багатьох квантових алгоритмів, особливо алгоритму факторизації Шора та квантової оцінки фази.

Дискретне перетворення Фур'є діє на вектор (x_0,…, x_{N-1}) і відображає його на вектор (y_0,…, y_{N-1}) згідно з формулою:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j w_N^{jk},$$

де $w_N^{jk} = e^{2\pi i \frac{jk}{N}}$.

Подібним чином квантове перетворення Фур'є діє на квантовий стан $|X\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$ і відображає його на квантовий стан $|Y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle$ відповідно до формули:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j w_N^{jk}.$$

Можна звернути увагу, що це перетворення впливає лише на амплітуди стану. Це також можна виразити як:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} w_N^{jk} |k\rangle.$$

Або як унітарна матриця:

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} w_N^{jk} |k\rangle\langle j|.$$

Квантове перетворення Фур'є виконує перетворення між двома базисами, обчислювальним (Z) базисом і базисом Фур'є. H-гейт є однокубітним КПФ, і він

перетворюється між Z-базисними станами $|0\rangle$ і $|1\rangle$ до X-базисних станів $|+\rangle$ і $|-\rangle$. Таким же чином усі багатокубітні стани в обчислювальній базі мають відповідні стани в базисі Фур'є. КПФ – це просто функція, яка перетворює ці основи.

$$|\text{Стан в обчислювальному базисі}\rangle \xrightarrow{\text{КПФ}} |\text{Стан у базисі Фур}'\text{є}\rangle.$$

$$QFT|x\rangle = |\widetilde{x}\rangle.$$

Наприклад, розглянемо, як оператор КПФ, як визначено вище, діє на одиничний стан кубіта $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. У цьому випадку $x_0 = \alpha$, $x_1 = \beta$ і $N = 2$. Тоді:

$$y_0 = \frac{1}{\sqrt{2}}\left(\alpha \cdot \exp\left(2\pi i \frac{0 \times 0}{2}\right) + \beta \cdot \exp\left(2\pi i \frac{1 \times 0}{2}\right)\right) = \frac{1}{\sqrt{2}}(\alpha + \beta),$$

$$y_1 = \frac{1}{\sqrt{2}}\left(\alpha \cdot \exp\left(2\pi i \frac{0 \times 1}{2}\right) + \beta \cdot \exp\left(2\pi i \frac{1 \times 1}{2}\right)\right) = \frac{1}{\sqrt{2}}(\alpha - \beta).$$

Кінцевим результатом є наступний стан:

$$U_{QFT}|\psi\rangle = \frac{1}{\sqrt{2}}(\alpha + \beta)|0\rangle + \frac{1}{\sqrt{2}}(\alpha - \beta)|1\rangle.$$

Ця операція є саме результатом застосування оператора Адамара (H) на кубіті. Якщо застосувати оператор H до стану $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, ми отримаємо новий стан:

$$\frac{1}{\sqrt{2}}(\alpha + \beta)|0\rangle + \frac{1}{\sqrt{2}}(\alpha - \beta)|1\rangle \equiv \tilde{\alpha}|0\rangle + \tilde{\beta}|1\rangle.$$

Квантове перетворення для більшого N знаходиться за формулою:

$$QFT_N|x\rangle = \frac{1}{\sqrt{N}}\left(|0\rangle + e^{\frac{2\pi i}{2}x}|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle\right) \ldots \left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right).$$

2.9.1 Схема, яка реалізує КПФ

Схема, яка реалізує КПФ, використовує два вентилі. Перший – це однокубітовий вентиль Адамара. У розділі 2.9 було продемонстровано, що дія H-гейта на однокубітний стан $|x_k\rangle$ є:

$$H|x_k\rangle = \frac{1}{\sqrt{N}}\left(|0\rangle + \exp(\frac{2\pi i}{2}x_k)|1\rangle\right).$$

Другий вентиль є двокубітним керованим обертанням $CROT_k$, заданий у блочно-діагональній формі як:

$$CROT_k = \begin{bmatrix} I & 0 \\ 0 & UROT_k \end{bmatrix}, \text{ де}$$

$$UROT_k = \begin{bmatrix} I & 0 \\ 0 & \exp(\frac{2\pi i}{2^k}) \end{bmatrix}.$$

Враховуючи ці два вентилі, схема, яка реалізує n-кубітне КПФ, показана на рис. 2.8.

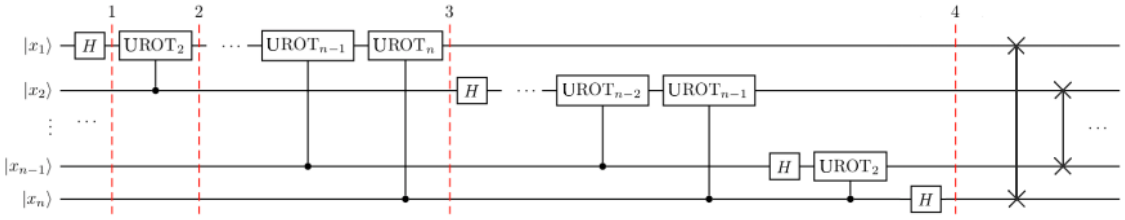

Рисунок 2.8 – Схема реалізації КПФ для n кубітів

Схема працює наступним чином:

1) Все починається з n-кубітового вхідного стану $|x_1, x_2, ..., x_n\rangle$.

2) Після першого вентиля Адамара на першому кубіті стан перетворюється з вхідного стану на стан:

$$H_1|x_1, x_2, \dots, x_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + \exp(\frac{2\pi i}{2}x_1)|1\rangle\right) \otimes |x_2, x_3, \dots, x_n\rangle.$$

3) Після вентиля $UROT_2$, на кубіті 1, керованому кубітом 2, стан трансформується на стан:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + \exp\left(\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1\right)|1\rangle\right) \otimes |x_2, x_3, \dots, x_n\rangle.$$

Після застосування останнього вентиля $UROT_n$ на кубіті 1, керованому кубітом n, стан буде мати наступну форму:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + \exp\left(\frac{2\pi i}{2^n}x_n + \frac{2\pi i}{2^{n-1}}x_{n-1} + \cdots + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1\right)|1\rangle\right) \otimes |x_2, x_3, \dots, x_n\rangle.$$

Відзначаючи, що:

$$x = 2^{n-1}x_1 + 2^{n-2}x_2 + \cdots + 2^1x_{n-1} + 2^0x_n,$$

можна записати наведений вище стан як:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + \exp\left(\frac{2\pi i}{2^n}x\right)|1\rangle\right) \otimes |x_2, x_3, \dots, x_n\rangle.$$

4) Після застосування подібної послідовності вентилей для кубітів 2,..., n, знаходимо кінцевий стан таким:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + \exp\left(\frac{2\pi i}{2^n}x\right)|1\rangle\right) \otimes \dots \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + \exp\left(\frac{2\pi i}{2^2}x\right)|1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + \exp\left(\frac{2\pi i}{2^1}x\right)|1\rangle\right).$$

Такий стан точно відповідає КПФ вхідного стану, який було отримано вище, з тим застереженням, що порядок кубитів у вихідному стані зворотний.

Оскільки, порядок кубітів у вихідному стані зворотній, застосовується вентиль SWAP в кінці схеми.

2.10 Алгоритм Шора

Алгоритм Шора відомий розкладанням цілих чисел за поліноміальний час. Оскільки найвідоміший класичний алгоритм потребує суперполіноміального часу для розкладання добутку двох простих чисел, широко використовувана криптосистема RSA покладається на неможливість розкладання на множники для достатньо великих цілих чисел.

Алгоритм Шора складається з двох частин:

- зведення, яке можна зробити на класичному комп'ютері, задачі розкладання до задачі пошуку порядку (класична частина).
- квантовий алгоритм для вирішення проблеми пошуку порядку (квантова частина).

У наступних розділах буде акцентовано увагу на квантовій частині алгоритму Шора, яка фактично вирішує проблему знаходження періоду. Оскільки задачу розкладання на множники можна перетворити на задачу знаходження періоду за поліноміальний час, ефективний алгоритм знаходження періоду також можна використовувати для ефективного розкладання цілих чисел.

Наразі цього достатньо, щоб показати, що якщо ми можемо обчислити період $a^x \pmod N$ ефективно, тоді ми також можемо ефективно факторизувати.

2.10.1 Проблема пошуку періоду

Розглянемо періодичну функцію:

$$f(x) = a^x \bmod N,$$

де a і N – натуральні числа і вони не мають спільних множників, $a < N$.

Період або порядок (r) – це найменше (не нульове) ціле число, яке:

$$a^r \bmod N = 1.$$

Можна побачити приклад цієї функції на графіку (рис. 2.9). Лінії між точками допомагають побачити періодичність і не представляють проміжні значення між x-маркерами.

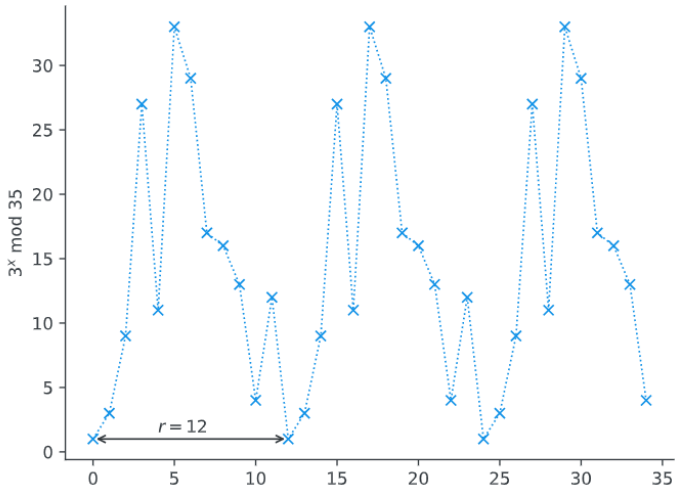

Рисунок 2.9 – Приклад періодичної функції в алгоритмі Шора

2.10.2 Рішення проблеми пошуку періоду

Рішення Шора полягало у використанні оцінки квантової фази на унітарному операторі:

$$U|y\rangle = |ay \bmod N\rangle.$$

Якби ми почали в стані$|1\rangle$, ми побачимо, що кожне послідовне застосування U буде множити стан регістру на a (mod N), і після r застосувань ми дійдемо до стану $|1\rangle$ знову.

Таким чином, суперпозиція станів у цьому циклі (|u0⟩) буде власним станом U:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \left|a^k \bmod N\right\rangle.$$

Цей власний стан має власне значення 1, що не дуже цікаво. Більш цікавим власним станом може бути той, у якому фаза є різною для кожного з цих базових станів обчислення. Зокрема, розглянемо випадок, коли фаза k-го стану пропорційна k:

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k}{r}} \left|a^k \bmod N\right\rangle.$$

Звідси:

$$U|u_1\rangle = e^{\frac{2\pi i}{r}} |u_1\rangle.$$

Це особливо важливе власне значення, оскільки воно містить *r*. Насправді *r* має бути включено, щоб переконатися, що різниці фаз між *r* обчислювальними базовими станами рівні. Це не єдиний власний стан із такою поведінкою; щоб узагальнити це далі, можна помножити ціле число *s* на цю різницю фаз, яка відображатиметься у власному значенні:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} \left|a^k \bmod N\right\rangle.$$

$$U|u_s\rangle = e^{\frac{2\pi i s}{r}} |u_s\rangle.$$

Тепер ми маємо унікальний власний стан для кожного цілого значення s $(0 \leq s \leq r-1)$.

Далі, якщо підсумувати всі ці власні стани, різні фази скасовують усі обчислювальні базові стани, крім |1⟩:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle.$$

Оскільки базовий стан обчислення |1⟩ є суперпозицією цих власних станів, це означає, що якщо ми робимо ОКФ на U, використовуючи стан |1⟩, виміряємо фазу:

$$\phi = \frac{s}{r},$$

де s – випадкове ціле число від 0 до r – 1.

Отже, використовуємо алгоритм неперервних дробів на ϕ, щоб знайти r. Принципова схема показана на рис. 2.10.

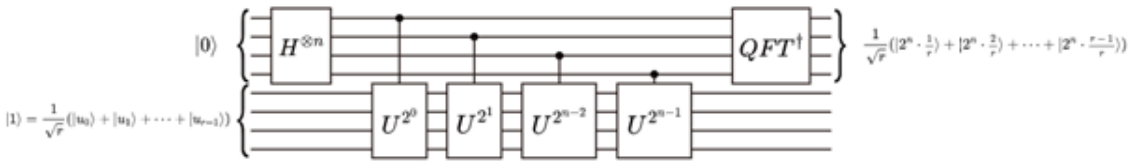

Рисунок 2.10 – Квантова схема алгоритму Шора

2.11 Переваги квантових обчислень

По-перше, квантові комп’ютери мають природний паралелізм, який може впоратися з проблемою простими кроками, які буде важко обчислити при використанні класичних комп’ютерів. Цей вид паралелізму призводить до зменшення складності часу, що призводить до широкого спектру застосувань квантових комп’ютерів у багатьох галузях і навіть може вплинути на майбутній

розвиток певних галузей. По-друге, квантові комп’ютери мають застосування в багатьох сферах. Його можна використовувати для злому кодів, що вплине на фінанси та військові справи. Його також можна використовувати для вирішення математичних задач. Висока ефективність квантових комп’ютерів у певних аспектах зробить можливими деякі нерозв’язні математичні проблеми. По-третє, порівняно з класичними бітами кубіти можна використовувати для зберігання та обчислення більшої кількості інформації. За тих самих умов кубіти можуть заощадити більше часу та місця [30].

2.12 Недоліки квантових обчислень

Спочатку квантовий комп’ютер може обробляти квант суперпозиційного стану паралельно, і після обробки виходить суперпозиційний стан багатьох результатів. Цей стан суперпозиції потребує подальших спостережень, щоб отримати певний стан, і спостереження спричинять колапс хвильової функції, і її неможливо буде виміряти знову. Іншими словами, за одну операцію можна отримати лише один результат, а проблеми з декількома рішеннями вимагають кількох операцій. Це призводить до фундаментальних недоліків квантових комп’ютерів у вирішенні проблем, які мають кілька рішень.

Далі, у класичному методі зберігання інформації 1 біт інформації може бути лише 0 або 1, і цей метод має відносно високий рівень терпимості до помилок. Для квантового зберігання інформації необхідний стан суперпозиції інформації $|0\rangle$ і $|1\rangle$, що зберігається в одному кубіті. Деякі стани суперпозиції мають дуже невеликі відмінності. Квантове зберігання інформації вимагає високого ступеня точності для роботи квантових комп’ютерів.

Нарешті, класичне зберігання інформації зберігає інформацію у формі напруги, і інформація, що зберігається таким чином, мало змінюється з часом. Квантова пам'ять зберігає когерентні накладені частинки, але ці частинки з часом будуть декогерентними, що означає, що ймовірність спотворення інформації відносно висока.

3 ПРОГРАМУВАННЯ КВАНТОВИХ АЛГОРИТМІВ У QISKIT

3.1 Qiskit як програмне забезпечення для квантових обчислень

Qiskit – це програмне забезпечення, яке знаходиться між квантовими алгоритмами з одного боку та фізичним квантовим пристроєм з іншого. Він перекладає такі поширені мови програмування, як Python, на мову квантової машини. Це означає, що будь-хто за межами лабораторії IBM Q може запрограмувати квантовий комп'ютер. Qiskit – чудовий навчальний інструмент для розвитку інтуїції щодо концепцій квантової інформації. Це також може бути шляхом для розвитку науки, гарантуючи, що квантові пристрої можуть надійно використовуватися різноманітною аудиторією, і покращуючи відтворюваність результатів. Завдяки такому широкому доступу IBM також сподівається сприяти створенню спільноти, яка зможе відкривати нові методи та революційні програми [31].

Все ще існує великий розрив між обчислювальними ресурсами, доступними на поточному апаратному забезпеченні, та ресурсами, необхідними для деяких із часто обіцяних застосувань квантових комп'ютерів, таких як цілочисельне факторування та моделювання молекул. Швидко просуваються дослідження того, як збільшити обчислювальні ресурси на чіпі (тобто кількість кубітів, точність вентилів і час когерентності) і розробити менш ресурсомісткі програми (наприклад, схеми малої глибини). Qiskit є третьою віссю цих зусиль, що дає можливість керувати накладними витратами на ресурси та адаптувати програми для конкретних пристроїв. Завдяки Qiskit розробникам, можливо, не доведеться чекати роками на появу більш потужного апаратного забезпечення. Зокрема, важливим завданням на наступні кілька років буде краще зрозуміти вплив шуму та його вплив на кінцеві результати обчислень. Доступ до реальної квантової машини є неоціненним для реалістичного виконання цих досліджень.

IBM Quantum пропонує кілька справжніх квантових комп’ютерів і високопродуктивних класичних обчислювальних симуляторів через свою IBM Quantum Lab.

3.2 Компоненти та елементи Qiskit

До основних елементів Qiskit належать:

– Qiskit Terra;
– Qiskit Aer;
– Qiskit Ignis;
– Qiskit Aqua.

Terra, елемент «земля», є основою, на якій лежить решта Qiskit. Він забезпечує основу для складання квантових програм на рівні схем і імпульсів, щоб оптимізувати їх для обмежень конкретного пристрою та керувати виконанням серій експериментів на пристроях віддаленого доступу. Terra визначає інтерфейси для бажаного досвіду кінцевого користувача, а також ефективне керування рівнями оптимізації, планування імпульсів і внутрішнього зв’язку.

Qiskit Terra складається з шести основних модулів [32]:

– qiskit.circuit;
– qiskit.pulse;
– qiskit.transpiler;
– qiskit.quantum_info;
– qiskit.visualization;
– qiksit.providers.

Квантова схема (qiskit.circuit) – це модель квантових обчислень, у якій обчислення виконується шляхом виконання послідовності квантових операцій (зазвичай вентилів) над регістром кубітів.

Розклад імпульсів (qiskit.pulse) – це набір імпульсів, які надсилаються в квантовий експеримент і застосовуються до каналу (експериментальний вхідний

рядок). Це нижчий рівень, ніж схеми, і вимагає, щоб кожен вентиль у схемі був представлений як набір імпульсів. На цьому рівні експерименти можуть бути розроблені для зменшення помилок (динамічна розв'язка, пом'якшення помилок та оптимальні форми імпульсів).

Основна частина досліджень квантових обчислень полягає в розробці того, як запустити квантові схеми на реальних пристроях. У цих пристроях експериментальні помилки та декогеренція вносять помилки під час обчислення. Таким чином, щоб отримати надійну реалізацію, важливо зменшити кількість вентилів і загальний час роботи квантової схеми. Транспайлер (qiskit.transpiler) представляє концепцію менеджера проходів, щоб дозволити користувачам досліджувати оптимізацію та знаходити кращі квантові схеми для заданого алгоритму.

Для виконання більш просунутих алгоритмів і аналізу схем, запущених на квантовому комп'ютері, важливо мати інструменти для реалізації простих квантових інформаційних завдань (qiskit.quantum_info). Вони включають методи як для оцінки показників, так і для генерації квантових станів, операцій і каналів.

У Terra є багато інструментів для візуалізації квантової схеми (qiskit.visualization). Це дозволяє швидко перевірити квантову схему, щоб переконатися, що це те, що користувач хотів реалізувати. Після запуску схеми важливо мати можливість переглядати вихід. Існує проста функція (plot_histogram()) для побудови результатів квантової схеми, включаючи інтерактивну версію. Існує також функція plot_state() і plot_bloch_vector(), які дозволяють побудувати графік квантового стану.

Після того, як користувач створив схеми для роботи на сервері, він повинен мати зручний спосіб роботи з ним (qiksit.providers). У Terra це робиться за допомогою чотирьох частин:

– провайдер (provider);
– сервер (backend);
– job екземпляри;
– результат (result).

Провайдер (provider) – це сутність, яка реалізує абстрактний базовий клас (BaseProvider) і надає доступ до групи різних серверних програм (наприклад, серверних програм, доступних через IBM Quantum). Він взаємодіє з цими бекендами, щоб, наприклад, дізнатися, які з них доступні, або отримати екземпляр певного бекенда.

Сервер (backend) – це сутність, яка реалізує абстрактний базовий клас (BaseBackend), що представляє симулятор або справжній квантовий комп'ютер і відповідає за роботу квантових схем і повернення результатів. У них є метод запуску, який приймає qobj як вхідні дані та повертає об'єкт BaseJob. Цей об'єкт дозволяє асинхронний запуск завдань для отримання результатів із серверної частини після завершення завдання.

Job екземпляри є реалізаціями абстрактного базового класу (BaseJob), і їх можна розглядати як «квиток» для поданої роботи. Вони з'ясовують стан виконання в певний момент часу (наприклад, якщо завдання стоїть у черзі, виконується чи не вдалося), а також дозволяють контролювати завдання.

Після завершення роботи Terra дозволяє отримати результати (result) з віддалених серверних програм за допомогою result = job.result(). Цей об'єкт результату містить квантові дані, і найпоширенішим способом взаємодії з ним є використання result.get_counts(circuit). Цей метод дозволяє користувачеві отримати необроблені підрахунки з квантової схеми та використовувати їх для додаткового аналізу за допомогою інструментів квантової інформації, наданих Terra.

Aer, елемент «повітря», пронизує всі елементи Qiskit. Щоб дійсно прискорити розробку квантових комп'ютерів, IBM потрібні кращі симулятори, емулятори та налагоджувачі. Aer допомагає зрозуміти обмеження класичних процесорів, демонструючи, якою мірою вони можуть імітувати квантові обчислення. Крім того, можна використовувати Aer, щоб переконатися, що поточні та найближчі квантові комп'ютери функціонують правильно. Це можна зробити, розширивши межі моделювання та імітуючи вплив реалістичного шуму на обчислення.

Aer надає високоефективну структуру симулятора для квантових схем, використовуючи стек програмного забезпечення Qiskit. Він містить оптимізовані модулі симулятора C++ для виконання схем, скомпільованих у Terra. Aer також надає інструменти для побудови моделей шуму з можливістю конфігурації для виконання реалістичного шумового моделювання помилок, які виникають під час виконання на реальних пристроях.

Qiskit Aer містить три високопродуктивні модулі симулятора:

– QasmSimulator;
– StatevectorSimulator;
– UnitarySimulator.

QasmSimulator дозволяє ідеальне та шумне багаторазове виконання схем qiskit і повертає підрахунки чи пам’ять. Є кілька методів, які можна використовувати для більш ефективного моделювання різних схем. До них належать:

– вектор станів (statevector) – використовує моделювання щільного вектора стану;
– стабілізатор (stabilizer) – використовує симулятор стану стабілізатора Кліффорда, який дійсний лише для схем Кліффорда та моделей шуму;
– подовженний стабілізатор (extended_stabilizer) – використовує наближений симулятор, який розкладає схеми на члени стану стабілізатора, кількість яких зростає разом із кількістю некліффордівських вентилів;
– matrix_product_state – використовує симулятор Matrix Product State (MPS).

StatevectorSimulator дозволяє ідеально одноразове виконання схем qiskit і повертає остаточний вектор стану симулятора після застосування.

UnitarySimulator дозволяє ідеальне одноразове виконання схем qiskit і повертає кінцеву унітарну матрицю самої схеми. Зауважте, що схема не може містити операції вимірювання або скидання для цього сервера.

Ignis, елемент «вогонь», присвячений боротьбі з шумом і помилками та створенню нового шляху. Це включає кращу характеристику помилок, покращення воріт і обчислення за наявності шуму. Ignis призначений для тих, хто хоче розробити коди квантової корекції помилок, або хто хоче вивчити способи характеристики помилок за допомогою таких методів, як томографія, або навіть знайти кращий спосіб використання вентилів шляхом вивчення динамічного розв'язування та оптимального керування.

Aqua, елемент «вода», є елементом життя. Щоб квантові обчислення виправдали очікування, потрібно знайти застосування в реальному світі. Aqua – це місце, де створюють алгоритми для квантових комп'ютерів. Ці алгоритми можна використовувати для створення програм для квантових обчислень. Aqua доступний для експертів у галузі хімії, оптимізації, фінансів та ШІ, які хочуть вивчити переваги використання квантових комп'ютерів як прискорювачів для конкретних обчислювальних завдань.

Проблеми, які можуть виграти від потужності квантових обчислень, були виявлені в багатьох областях, таких як хімія, ШІ, оптимізація та фінанси. Однак квантові обчислення потребують дуже спеціальних навичок. Щоб задовольнити потреби величезної кількості практиків, які хочуть використовувати квантові обчислення на різних рівнях програмного забезпечення та робити свій внесок у них, IBM створили Qiskit Aqua.

3.3 Реалізація КПФ у Qiskit

В Qiskit реалізація вентиля CROT, який визначений у розділі 2.9.1, є вентилем з керованим обертанням фази. Цей вентиль визначено в OpenQasm як:

$$CP(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{bmatrix}.$$

Отже, відображення вентиля $CROT_k$ в CP-гейт знайдено з рівняння:

$$\theta = \frac{2\pi}{2^k} = \frac{\pi}{2^{k-1}}.$$

Приклад реалізації КПФ на 3-х кубітах зображено на рис. 3.1 і 3.2.

```
import numpy as np
from numpy import pi
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute, transpile, Aer, BasicAer, IBMQ
from qiskit.tools.jupyter import *
from qiskit.visualization import plot_histogram, plot_bloch_multivector
from ibm_quantum_widgets import *
from qiskit.providers.aer import QasmSimulator
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
```

Рисунок 3.1 – Імпорт бібліотек Qiskit

```
qc = QuantumCircuit(3) # Створення квантової схеми з трьома кубітами
qc.h(2) # Вентиль Адамара до кубіта 2
qc.cp(pi/2, 1, 2) # CROT від кубіта 1 до кубіта 2
qc.cp(pi/4, 0, 2) # CROT від кубіта 0 до кубіта 2
qc.h(1) # Вентиль Адамара до кубіта 1
qc.cp(pi/2, 0, 1) # CROT від кубіта 0 до кубіта 1
qc.h(0) # Вентиль Адамара до кубіта 0
qc.swap(0, 2) # Вентиль SWAP між кубітами 0 і 2
qc.draw()
```

Рисунок 3.2 – Реалізація КПФ на 3-х кубітах у Qiskit

На рис. 3.2 спочатку необхідно визначити квантову схему (qc). Застосовуємо H-гейт до кубіта 2. Далі ми хочемо повернути це на додаткову чверть оберту, якщо кубіт 1 знаходиться в стані $|1\rangle$. І ще, якщо найменш значущий кубіт 0 знаходиться в стані $|1\rangle$.

Подбавши про цей кубіт, тепер можна ігнорувати його та повторити процес, використовуючи ту саму логіку для кубітів 0 і 1. Нарешті, необхідно поміняти місцями кубіти 0 і 2, щоб завершити КПФ.

Побудована схема зображена на рис. 3.3.

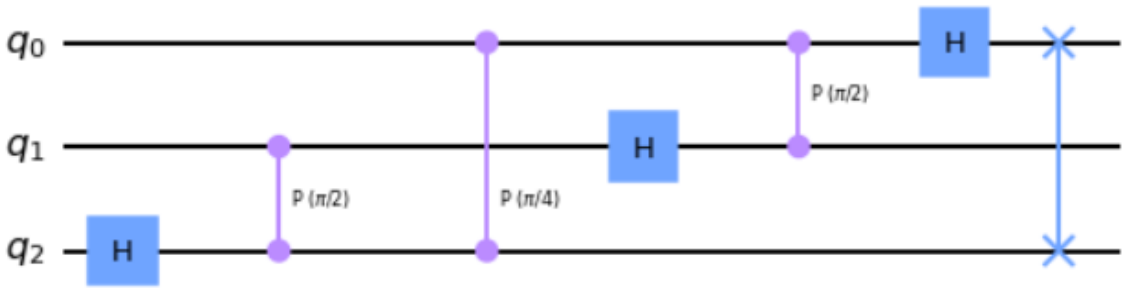

Рисунок 3.3 – Схема КПФ на 3-х кубітах у Qiskit

Для запуску на реальному квантовому пристрої спочатку необхідно створити загальну функцію КПФ.

Почнемо зі створення функції, яка правильно обертає кубіти. Отже, спочатку, як було з прикладом із 3-ма кубітами, треба правильно обертати найважливіший кубіт (кубіт із найвищим індексом).

Це перша частина КПФ. Після правильно повернутого найважливішого кубіта, потрібно правильно повернути другий за значимістю кубіт. Потім необхідно мати справу з третім за значимістю і так далі. Щоб не писати більше коду, можна використовувати той самий код, коли програма підходить до кінця функції QFT_rotations(), щоб повторити процес для наступних n – 1 кубітів. Остаточна функція і схема продемонстрована на рис. 3.4 і 3.5.

```
def QFT_rotations(circuit, n):
    if n == 0: # Вихід з функції, якщо схема порожня
        return circuit
    n -= 1 # Індекси починаються з 0
    circuit.h(n) # Застосування H-гейта до найбільш важливого кубіта
    for qubit in range(n):
        # Для кожного менш значущого кубіта потрібно виконати кероване обертання під меншим кутом
        circuit.cp(pi/2**(n - qubit), qubit, n)
    # Наприкінці функції знову викликається та сама функція для наступних кубітів
    QFT_rotations(circuit, n)
```

Рисунок 3.4 – Функція, яка правильно обертає n кубітів

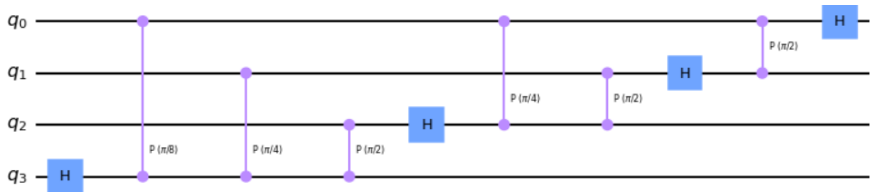

Рисунок 3.5 – Схема при використанні функції обертання на 4 кубітах

Нарешті, потрібно додати вентилі SWAP в кінці функції КПФ, щоб відповідати визначенню КПФ. Все це було об’єднано в остаточну функцію QFT() на рис. 3.6. Схема використання даної функції зображена на рис. 3.7.

```
def swap_registers(circuit, n):
    for qubit in range(n//2):
        circuit.swap(qubit, n - qubit - 1)
    return circuit

def QFT(circuit, n):
    QFT_rotations(circuit, n)
    swap_registers(circuit, n)
    return circuit
```

Рисунок 3.6 – Функція з вентилем SWAP і загальна функція КПФ

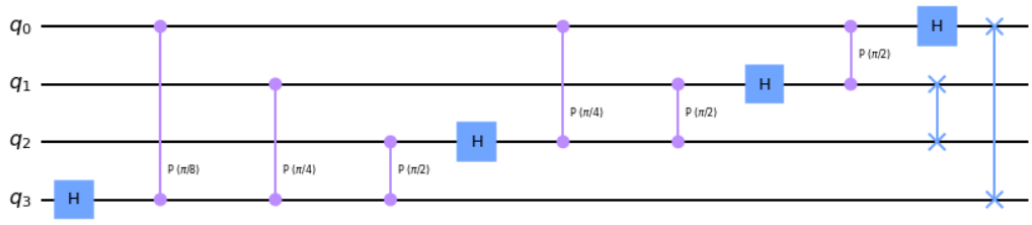

Рисунок 3.7 – Схема при використанні функції КПФ на 4 кубітах

Тепер залишилось продемонструвати, що ця схема працює правильно. Для цього треба спочатку закодувати число в базі обчислень, наприклад число 6. Число 6 у двійковій системі відповідає значенню 110.

Закодуємо це в кубіти, і перевіримо стан кубіта за допомогою симулятора aer (рис. 3.8).

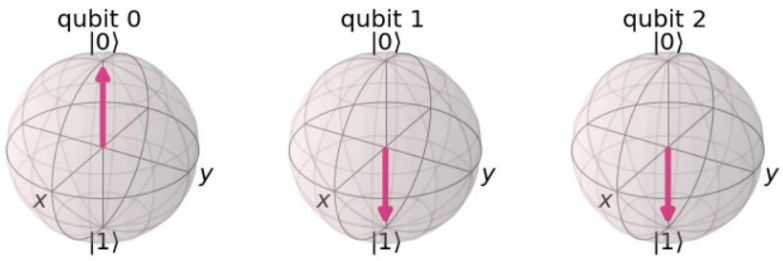

Рисунок 3.8 – Стан кубіта, який відповідає стану |6⟩ в базі обчислень

Нарешті, скористаємося функцією QFT() (рис. 3.9) і переглянемо остаточний стан кубітів (рис. 3.10).

```
qc = QuantumCircuit(3)
qc.x([1, 2])
QFT(qc, 3)
sim = Aer.get_backend('aer_simulator')
qc.save_statevector()
statevector = sim.run(qc).result().get_statevector()
plot_bloch_multivector(statevector)
```

Рисунок 3.9 – Використання функції QFT() на 3 кубітах та симуляція за допомогою aer симулятора

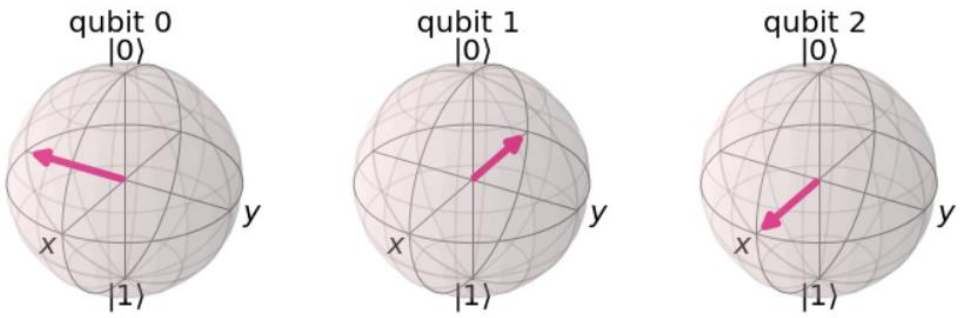

Рисунок 3.10 – Стан кубіта, який відповідає стану $|\tilde{6}\rangle$ в базі Фур'є

Можна побачити, що функція QFT() працювала правильно. Кубіт 0 повернуто на 6/8 повних оберти, кубіт 1 – на 12/8 повні оберти (еквівалентно 3/2 повному оберту), а кубіт 2 – на 24/8 повних оберти (еквівалентно 3 повних обертів).

3.3.1 Запуск КПФ на реальному квантовому пристрої

Якби ми спробували запустити схему в кінці розділу 3.3 на реальному пристрої, результати були б абсолютно випадковими, оскільки всі кубіти знаходяться в однаковій суперпозиції станів $|0\rangle$ і $|1\rangle$.

Якщо треба продемонструвати та дослідити КПФ, що працює на реальному апаратному забезпеченні, можна натомість створити, наприклад стан $|\tilde{7}\rangle$,

запустити КПФ у зворотному порядку та перевірити, що результатом є стан $|7\rangle$, як і очікувалося.

По-перше, використаємо Qiskit, щоб легко змінити операцію КПФ (рис. 3.11).

```
def inverse_QFT(circuit, n):
    # Спочатку створюємо схему КПФ правильного розміру:
    qft_circ = QFT(QuantumCircuit(n), n)
    # Далі візьмемо зворотну схему від цієї схеми
    invqft_circ = qft_circ.inverse()
    # І додаємо її до перших n кубітів в існуючій схемі
    circuit.append(invqft_circ, circuit.qubits[:n])
    return circuit.decompose()  # .decompose() дозволяє бачити окремі вентилі
```

Рисунок 3.11 – Зміна функції QFT() на зворотню функцію inverse_QFT()

Тепер переведемо кубіти в стан $|\tilde{7}\rangle$ (рис. 3.12).

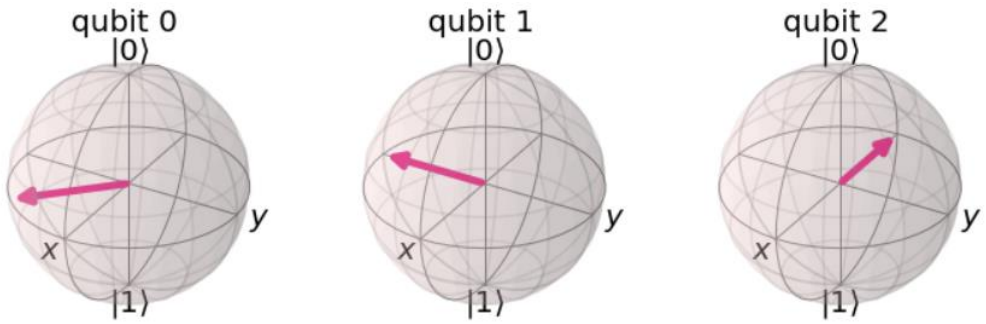

Рисунок 3.11 – Стан кубіта, який відповідає стану $|\tilde{7}\rangle$ в базі Фур'є

Нарешті, застосуємо зворотнє КПФ. Результатом є схема на рис. 3.12

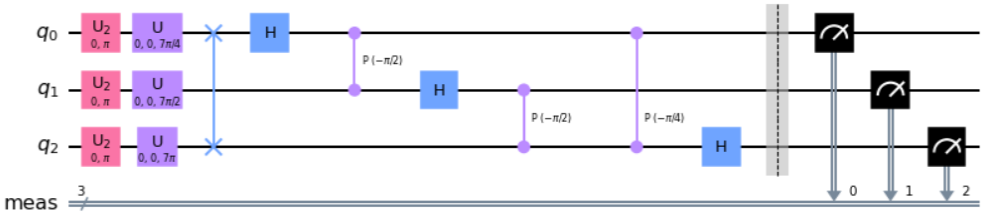

Рисунок 3.12 – Схема реалізації КПФ на реальному квантовому пристрої на 3 кубітах

На рис. 3.13 зображено завантаження облікового запису і отримання найменш завантаженого серверного пристрою з кількістю nqubits та вимірювання побудованої схеми (рис. 3.12) на реальному квантовому пристрої. Результат вимірювання зображено на рис. 3.14.

```
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits >= nqubits
                                       and not x.configuration().simulator
                                       and x.status().operational==True))
print("least busy backend: ", backend)
```

```
least busy backend:  ibmq_quito
```

```
shots = 2048
transpiled_qc = transpile(iqc, backend, optimization_level=3)
job = backend.run(transpiled_qc, shots=shots)
job_monitor(job)
counts = job.result().get_counts()
plot_histogram(counts)
```

```
Job Status: job has successfully run
```

Рисунок 3.13 – Вимірювання на реальному квантовому пристрої

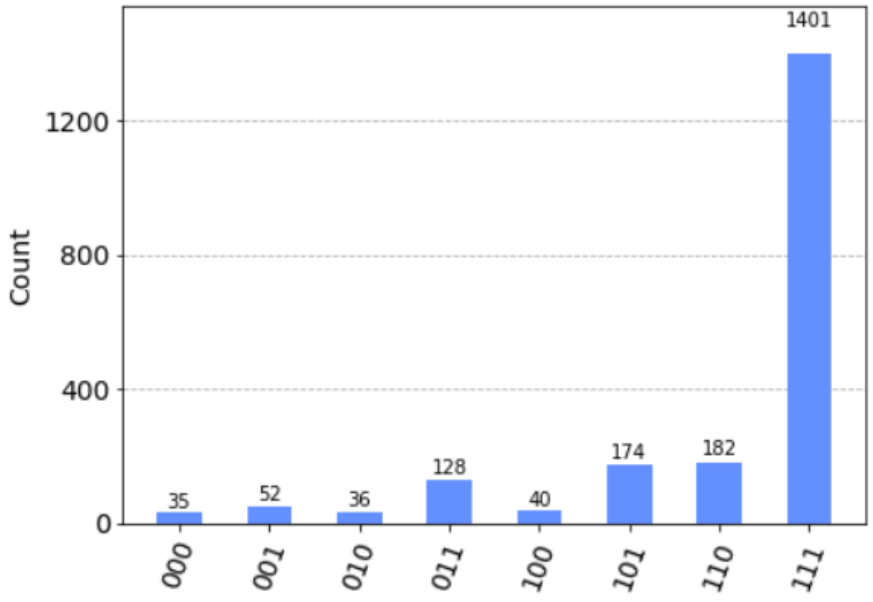

Рисунок 3.14 – Результат використання КПФ на реальному квантовому пристрої

В результаті видно, що найімовірніший результат є 111 що відповідає стану $|7\rangle$.

3.4 Реалізація алгоритму Шора в Qiskit

Розгянемо два значення 15 і 21, а також розв'яжемо задачу знаходження періоду при a = 2, N = 15 і N =21. Надаємо схеми для U де:

$$U|y\rangle = |ay \bmod(15)\rangle,$$
$$U|y\rangle = |ay \bmod(21)\rangle.$$

Щоб створити U^x, просто необхідно повторити схему x разів. Функції c_amod15 (рис. 3.15) та c_amod21 (рис. 3.16) повертає контрольований U-гейт для a, повторюваних power разів.

```
def c_amod15(a, power):
    U = QuantumCircuit(4)
    for i in range(power):
        U.swap(2, 3)
        U.swap(1, 2)
        U.swap(0, 1)
    U = U.to_gate()
    U.name = "%i^%i mod 15" % (a, power)
    c_U = U.control()
    return c_U
```

Рисунок 3.15 – Функція, яка створює котрольований U-гейт для N = 15

```
def c_amod21(a, power):
    U = QuantumCircuit (5)
    for i in range(power):
        U.ccx(0,3,4)
        U.ccx(0,3,1)
        U.ccx(0,4,3)
        U.ccx(0,4,2)
        U.cx(4,3)
        U.cx(4,2)
        U.cx(4,0)
        U.swap(3,4)
        U.swap(2,3)
        U.swap(1,2)
        U.swap(0,1)
        U.cx(0,4)
    U = U.to_gate()
    U.name = "%i^%i mod 21" %(a, power)
    c_U = U.control()
    return c_U
```

Рисунок 3.16 – Функція, яка створює котрольований U-гейт для N = 21

Також імпортуємо схему для КПФ, яка була розглянута в розділі 3.3 (рис. 3.17).

```
def qft_dagger(n):
    """n-qubit QFTdagger the first n qubits in circ"""
    qc = QuantumCircuit(n)
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT†"
    return qc
```

Рисунок 3.17 – Функція зворотнього КПФ

Для пошуку періоду для N = 15 потрібен 4-кубітний робочий регістр, а для N = 21 – 5-кубітний. Потрібно працювати принаймні з $\lceil \log_2 N \rceil$ кубітів. Інакше не є можливим зберігати проміжні значення під час модульного піднесення до степеня.

Також потрібно принаймні 8 і 10 елементів керування. При використанні менше $\lceil 2 \log_2 N \rceil$ елементів керування, результат фактично не матиме необхідної роздільної здатності для розрізнення різних періодів.

Отже, будемо використовувати 8 рахункових кубітів і 4 робочих регістри для N = 15, а також 10 рахункових кубітів і 5 робочих регістри для N = 21.

За допомогою цих будівельних блоків тепер можна легко побудувати схему для алгоритму Шора (рис. 3.18, 3.19).

```
n_count = 10
a = 2
qc = QuantumCircuit(n_count + 5, n_count)
for q in range(n_count):
    qc.h(q)
qc.x(n_count)
for q in range(n_count):
    qc.append(c_amod21(a, 2**q), [q]+[i+n_count for i in range(5)])

qc.append(qft_dagger(n_count), range(n_count))
qc.measure(range(n_count), range(n_count))
qc.draw(fold = -1)
```

Рисунок 3.18 – Побудова схеми алгоритму Шора для N = 21

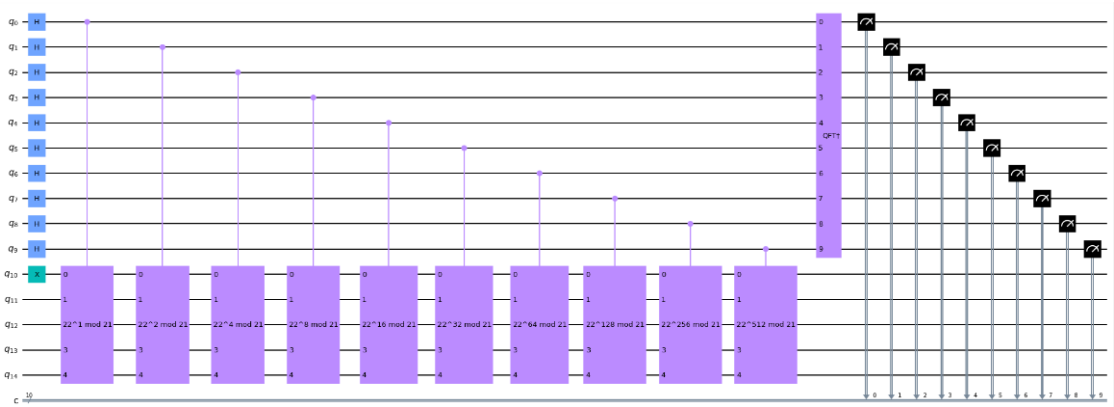

Рисунок 3.19 – Схема алгоритму Шора

Далі подивимося, які результати вимірюються, використовуючи симулятор aer (рис. 3.20 і 3.21).

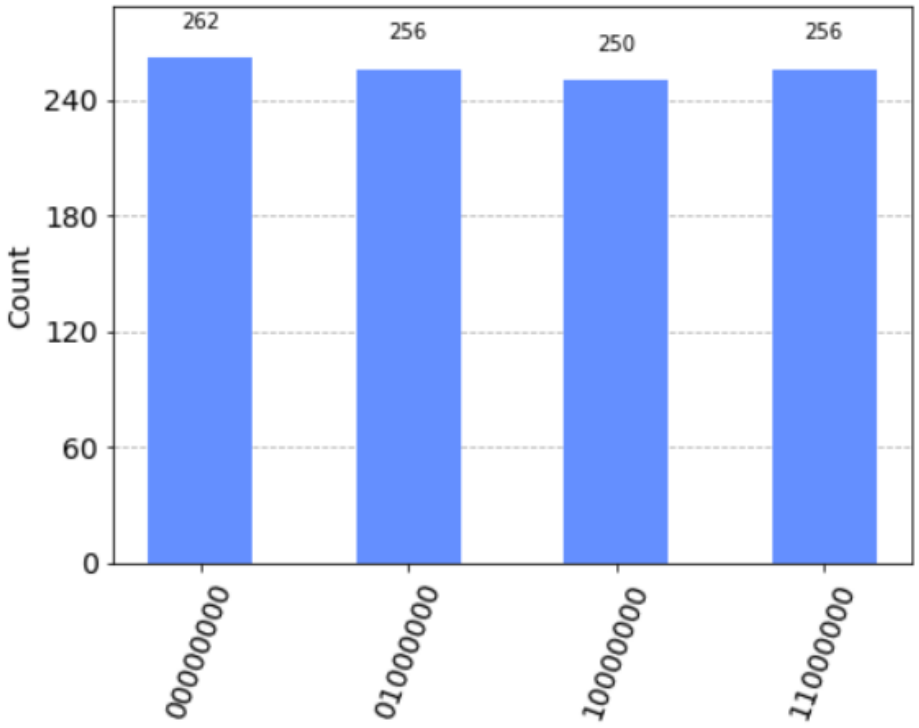

Рисунок 3.20 – Результат вимірювання схеми з N = 15

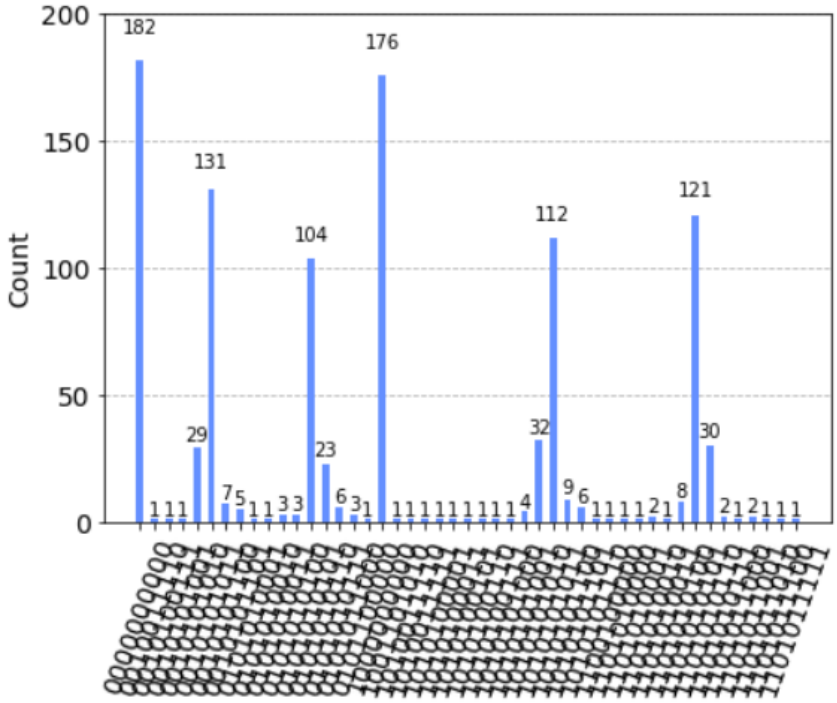

Рисунок 3.21 – Результат вимірювання схеми з N = 21

Ці результати відповідають виміряним фазам зображеним на рис. 3.22 і 3.23.

```
                Register Output                Phase
0   10000000(bin) = 128(dec)   128/256 = 0.50
1   11000000(bin) = 192(dec)   192/256 = 0.75
2   01000000(bin) =  64(dec)    64/256 = 0.25
3   00000000(bin) =   0(dec)     0/256 = 0.00
```

Рисунок 3.22 – Знайдені фази при вимірюванні схеми для N = 15

```
30  0101011000(bin) = 344(dec)  344/1024 = 0.34
31  1101010110(bin) = 854(dec)  854/1024 = 0.83
32  1010101100(bin) = 684(dec)  684/1024 = 0.67
33  0101010101(bin) = 341(dec)  341/1024 = 0.33
34  0101010111(bin) = 343(dec)  343/1024 = 0.33
35  1101010111(bin) = 855(dec)  855/1024 = 0.83
36  1010011110(bin) = 670(dec)  670/1024 = 0.65
37  1101011001(bin) = 857(dec)  857/1024 = 0.84
38  0010101101(bin) = 173(dec)  173/1024 = 0.17
39  1100100000(bin) = 800(dec)  800/1024 = 0.78
40  0101010011(bin) = 339(dec)  339/1024 = 0.33
41  1101010011(bin) = 851(dec)  851/1024 = 0.83
42  1001101010(bin) = 618(dec)  618/1024 = 0.60
43  1010011111(bin) = 671(dec)  671/1024 = 0.66
44  0010101100(bin) = 172(dec)  172/1024 = 0.17
45  1101010010(bin) = 850(dec)  850/1024 = 0.83
46  1010101001(bin) = 681(dec)  681/1024 = 0.67
```

Рисунок 3.23 – Знайдені фази при вимірюванні схеми для N = 21

Тепер залишилось використати алгоритм ланцюгових дробів, щоб спробувати знайти *s* та *r*. Python має таку вбудовану функціональність: ми можемо використовувати модуль fractions, щоб перетворити float на Fraction об'єкт.

Порядок (*r*) має бути меншим за *N*, тому встановимо максимальний знаменник рівним 15 і 21. Остаточний результат зображений на рис. 3.24 і 3.25.

	Phase	Fraction	Guess for r
0	0.50	1/2	2
1	0.75	3/4	4
2	0.25	1/4	4
3	0.00	0/1	1

Рисунок 3.24 – Знайдений порядок *r* для N = 15

30	0.335938	1/3	3
31	0.833984	5/6	6
32	0.667969	2/3	3
33	0.333008	1/3	3
34	0.334961	1/3	3
35	0.834961	5/6	6
36	0.654297	13/20	20
37	0.836914	5/6	6
38	0.168945	1/6	6
39	0.781250	7/9	9
40	0.331055	1/3	3
41	0.831055	5/6	6
42	0.603516	3/5	5
43	0.655273	13/20	20
44	0.167969	1/6	6
45	0.830078	5/6	6
46	0.665039	2/3	3

Рисунок 3.25 – Знайдений порядок *r* для N = 21

Ми бачимо, що для N = 15 два з виміряних власних значень дали правильний результат: *r* = 4 , а також, що алгоритм Шора має ймовірність невдачі. Ці погані результати тому, що *s* = 0, або тому, що *s* і *r* не є взаємно простими і замість *r* нам надано множник *r*. Найпростішим рішенням є просто повторювати експеримент, доки не отримаємо задовільний результат для порядку *r*.

4 ДОСЛІДЖЕННЯ НЕОБХІДНОЇ КІЛЬКОСТІ КУБІТІВ ДЛЯ ФАКТОРИЗАЦІЇ ТА ЧАСУ ВИКОНАННЯ ПРОГРАМИ

4.1 Створення програми факторизації чисел при a = 2 і N ≤ 21

Не всі проблеми факторизації складні, можна миттєво помітити парне число й знати, що один із його множників дорівнює 2. Насправді існують певні критерії вибору чисел, які важко розкласти, але основна ідея полягає в тому, щоб вибрати добуток двох великих простих чисел.

Загальний алгоритм розкладання на множники спочатку перевірить, чи існує швидкий спосіб розкладання цілого числа на множники, перш ніж використовувати пошук періоду Шора для найгіршого сценарію.

Використання алгоритму Шора для факторизації має сенс, коли N є непарним, розкладається лише на прості множники і є періодичним з парним періодом *r*. Тому для N ≤ 21 алгоритм Шора буде використовуватися лише для чисел 15 і 21 як в розділі 3.4. Для інших чисел буде використовуватися функція factelse (рис. 4.1), яка використовує функції правильного розподілу x_gate() (рис. 4.3) та swap_gate() (рис. 4.3).

```
def factelse(a, N):
    I = QuantumCircuit(5)
    sim = Aer.get_backend('aer_simulator')
    I = x_gate(I, N)
    factors = list()
    while (a <= N):
        if (N % a) == 0:
            if (N % 2) == 0:
                factors.append(a)
                I = swap_gate(I, N)
                I.measure_all()
                result = sim.run(I, memory = True).result().get_memory()
                x = int(result[0], 2)
                I.remove_final_measurements()
                N = x
            elif ((N % 2) != 0) and ((sqrt(N) % 1) == 0):
                I.swap(1, 3)
                I.measure_all()
                result = sim.run(I, memory = True).result().get_memory()
                x = int(result[0], 2)
                I.remove_final_measurements()
                N = x
                factors.append(a)
            else:
                factors.append(a)
                a += 1
        else:
            a += 1
    return factors
```

Рисунок 4.1 – Функція для факторизації інших чисел в Qiskit

```
def x_gate(circuit, N):
    if (N % 2) == 0:
        if N in [2, 6, 10, 14, 18]:
            circuit.x(1)
        if N in [4, 6, 12, 14, 20]:
            circuit.x(2)
        if N in [8, 10, 12, 14]:
            circuit.x(3)
        if N in [16, 18, 20]:
            circuit.x(4)
    else:
        circuit.x(0)
        if N in [3, 7, 11, 19]:
            circuit.x(1)
        if N in [5, 7, 13]:
            circuit.x(2)
        if N in [9, 11, 13]:
            circuit.x(3)
        if N in [17, 19]:
            circuit.x(4)
    return circuit
```

Рисунок 4.2 – Функції x_gate()

```
def swap_gate(circuit, N):
    if (N % 2) == 0:
        circuit.swap(0, 1)
        circuit.swap(1, 2)
        circuit.swap(2, 3)
        circuit.swap(3, 4)
    return circuit
```

Рисунок 4.3 – Функції swap_gate()

Для побудови програми факторизації чисел N буде використовуватися дві функції: factorn_21 і factoring_n21.

Перша функція (рис. 4.4) будує та вимірює квантові схеми алгоритму Шора, якщо N = 15 або N = 21. Для інших чисел використовується функція factelse. Отже, перша функція виконує квантову частину алгоритму Шора.

```
def factorn_21(a, N):
    if N == 15:
        n_count = 8
        qc = QuantumCircuit(n_count + 4, n_count)
        for q in range(n_count):
            qc.h(q)
        qc.x(n_count)
        for q in range(n_count):
            qc.append(c_amod15(a, 2**q), [q] + [i + n_count for i in range(4)])
        qc.append(qft_dagger(n_count), range(n_count))
        qc.measure(range(n_count), range(n_count))
        sim = Aer.get_backend('aer_simulator')
        t_qc = transpile(qc, sim)
        qobj = assemble(t_qc, shots = 1)
        result = sim.run(qobj, memory = True).result()
        readings = result.get_memory()
        phase = int(readings[0], 2)/(2**n_count)
        return phase
    elif N == 21:
        n_count = 10
        qc = QuantumCircuit(n_count + 5, n_count)
        for q in range(n_count):
            qc.h(q)
        qc.x(n_count)
        for q in range(n_count):
            qc.append(c_amod21(a, 2**q), [q] + [i + n_count for i in range(5)])
        qc.append(qft_dagger(n_count), range(n_count))
        qc.measure(range(n_count), range(n_count))
        sim = Aer.get_backend('aer_simulator')
        t_qc = transpile(qc, sim)
        qobj = assemble(t_qc, shots = 1)
        result = sim.run(qobj, memory = True).result()
        readings = result.get_memory()
        phase = int(readings[0], 2)/(2**n_count)
        return phase
    else:
        factors = factelse(a, N)
        return factors
```

Рисунок 4.4 – Функція factorn_21 для факторизації чисел N

Друга функція (рис. 4.5) виконує класичну частину алгоритму Шора і повторює алгоритм, поки не буде знайдено принаймні один множник для чисел 15 або 21.

```
def factoring_n21(a, N):
    if a != 2:
        raise ValueError("'a' must be 2")
    if N > 21:
        raise ValueError("'N' must be 21 or less")
    if N == 15 or N == 21:
        factor_found = False
        attempt = 0
        while not factor_found:
            attempt += 1
            print("\nAttempt %i:" % attempt)
            phase = factorn_21(a, N)
            frac = Fraction(phase).limit_denominator(N)
            r = frac.denominator
            print("Result: r = %i" % r)
            if phase != 0:
                guesses = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]
                print("Guessed Factors: %i and %i" % (guesses[0], guesses[1]))
                for guess in guesses:
                    if guess not in [1,N] and (N % guess) == 0:
                        print("*** Non-trivial factor found: %i ***" % guess)
                        factor_found = True
    else:
        factor = factorn_21(a, N)
        print ("Factors found:", factor)
        return factor
```

Рисунок 4.5 – Функція factoring_n21 для факторизації чисел N

Результати показані на рис. 4.6 і 4.7.

```
a = 2
N1 = 10
N2 = 12
N3 = 18
N4 = 20
fac1 = factoring_n21(a, N1)
fac2 = factoring_n21(a, N2)
fac3 = factoring_n21(a, N3)
fac4 = factoring_n21(a, N4)
```

```
Factors found: [2, 5]
Factors found: [2, 2, 3]
Factors found: [2, 3, 3]
Factors found: [2, 2, 5]
```

Рисунок 4.6 – Факторизація парних чисел N1, N2, N3 і N4

```
a = 2
N = 15
fac = factoring_n21(a, N)
```

```
Attempt 1:
Result: r = 1

Attempt 2:
Result: r = 4
Guessed Factors: 3 and 5
*** Non-trivial factor found: 3 ***
*** Non-trivial factor found: 5 ***
```

```
a = 2
N = 21
fac = factoring_n21(a, N)
```

```
Attempt 1:
Result: r = 6
Guessed Factors: 7 and 3
*** Non-trivial factor found: 7 ***
*** Non-trivial factor found: 3 ***
```

Рисунок 4.7 – Факторизація чисел з використанням алгоритму Шора

В результаті була створена програма для факторизації чисел $N \leq 21$, яка використовує квантовий алгоритм Шори при необхідності, якщо N відповідає значенням 15 і 21.

4.2 Аналіз необхідної кількості кубітів для алгоритмів та витраченого часу на факторизацію

В Qiskit вбудований алгоритм Шора за кількістю кубітів і часом факторизації відрізняється від алгоритму побудованого в розділі 3.4. На рис. 4.8 показана кількість кубітів, яка використовувалась для факторизації числа 21.

```
print(f'Computed of qubits for circuit: {4 * math.ceil(math.log(N, 2)) + 2}')
print(f'Actual number of qubits of circuit: {shor.construct_circuit(N).num_qubits}')
```

```
Computed of qubits for circuit: 22
Actual number of qubits of circuit: 22
```

Рисунок 4.8 – Кількість кубітів в схемі Qiskit для факторизації числа 21

Вбудований алгоритм Шора при факторизації числа N використовує $4\log_2(N) + 2$ кубітів. Алгоритм представлений в розділі 3.4 використовує $3\log_2(N)$, а побудована функція factelse() в розділі 4.1 – $\log_2(N)$. Отже, результати використання цих алгоритмів для різних N показані в табл. 4.1.

Таблиця 4.1 – Порівняння використаних кубітів для факторизації

Число N	Вбудований алгоритм Шора ($4\log_2(N) + 2$)	Створений алгоритм Шора ($3\log_2(N)$)	Функція factelse() ($\log_2(N)$)
16	18	12	4
32	22	15	5
64	26	18	6
128	30	21	7
256	34	24	8
512	38	27	9
1024	42	30	10
2048	46	33	11

Графіки залежності числа, яке можна факторизувати, від необхідної кількості кубітів були побудовані в системі Mathcad і зображені на рис. 4.9 і 4.10.

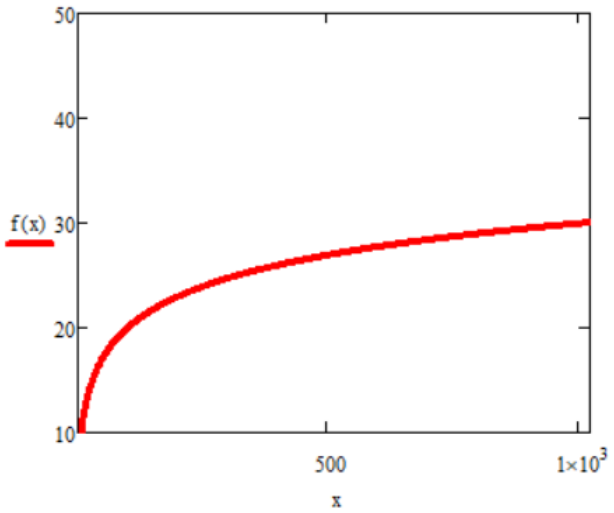

Рисунок 4.9 – Графік залежності кількісті кубітів від числа N в алгоримі Шора ($4\log_2(N) + 2$)

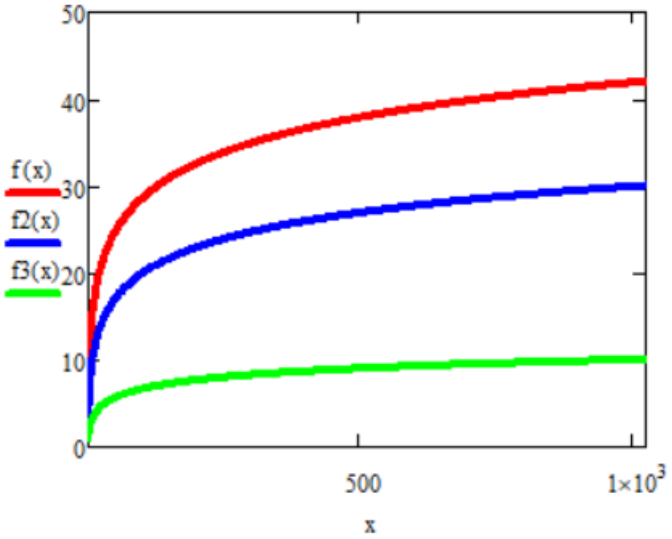

Рисунок 4.10 – Порівняння графіків залежності кількісті кубітів від числа N в алгоритмі Шора($4\log_2(N) + 2$, $3\log_2(N)$ і $\log_2(N)$)

Залежність для функції factelse() зображена на рис. 4.11.

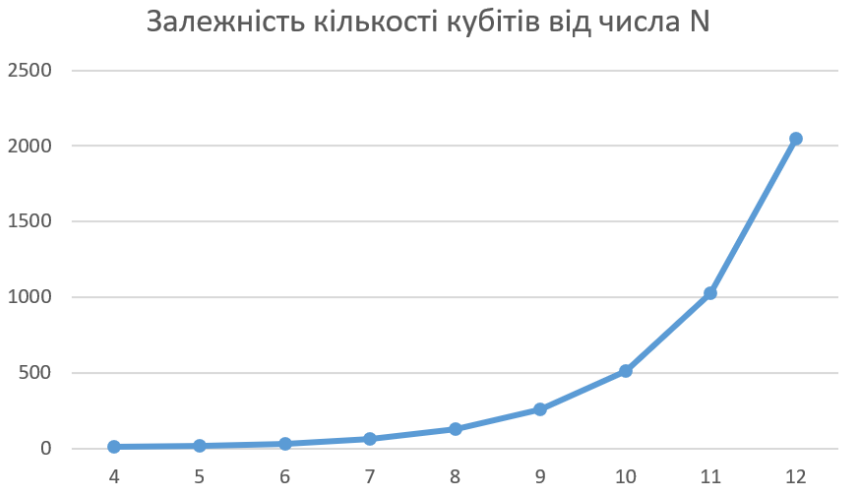

Рисунок 4.11 – Графік залежності кількісті кубітів від числа N в функції factelse()

Дослідження часу виконання алгоритмів продемонстровані на рис. 4.12, 4.13 та 4.14.

```
import time
a = 2
N = 15
st = time.time()
fac = factoring_n21(a, N)
time.sleep(3)
et = time.time()
res = et - st
final_res = res / 60
print("Time execution:", res, final_res)
```

```
Attempt 1:
Result: r = 4
Guessed Factors: 3 and 5
*** Non-trivial factor found: 3 ***
*** Non-trivial factor found: 5 ***
Time execution: 4.044835805892944 0.06741393009821574
```

Рисунок 4.12 – Час виконання створеної функції факторизації factoring_n21() при факторизації числа 15

```
import time
a = 2
N = 21
st = time.time()
fac = factoring_n21(a, N)
time.sleep(3)
et = time.time()
res = et - st
final_res = res / 60
print("Time execution:", res, final_res)
```

```
Attempt 1:
Result: r = 6
Guessed Factors: 7 and 3
*** Non-trivial factor found: 7 ***
*** Non-trivial factor found: 3 ***
Time execution: 43.766950607299805 0.7294491767883301
```

Рисунок 4.13 – Час виконання створеної функції факторизації factoring_n21() при факторизації числа 21

```
import time
N = 21
backend = Aer.get_backend('aer_simulator')
quantum_instance = QuantumInstance(backend, shots=1024)
shor = Shor(quantum_instance=quantum_instance)
st = time.time()
result = shor.factor(N)
time.sleep(3)
print(f"The list of factors of {N} as computed by the Shor's algorithm is {result.factors[0]}.")
et = time.time()
res = et - st
final_res = res/60
print("Time execution:", res, final_res)
```

```
The list of factors of 21 as computed by the Shor's algorithm is [3, 7].
Time execution: 61.23547697067261 1.0205912828445434
```

Рисунок 4.14 – Час виконання вбудованого алгоритму Шора при факторизації числа 21

На цих рисунках перше число відповідає часу в секундах, а друге – в хвилинах.

Результати дослідження занесені до табл. 4.2 і 4.3. В табл. 4.2 результати при N = 15, в табл. 4.3 при N = 21.

Таблиця 4.2 – Порівняння витраченого часу виконання

Кількість спроб	Вбудований алгоритм Шора (с.)	Створений алгоритм Шора (с.)
1	11.48	3.34
2	11.87	1.07
3	11.28	1.04
4	12.02	1.28
5	11.54	1.36

Таблиця 4.3 – Порівняння витраченого часу виконання

Кількість спроб	Вбудований алгоритм Шора (с.)	Створений алгоритм Шора (с.)
1	61.59	43.76
2	62.01	43.59
3	61.68	44.69
4	63.05	44.37
5	62.44	43.68

В результаті дослідження можна зробити висновок, що створений алгоритм факторизації витрачає трохи менше часу для виконання ніж вбудований алгоритм. Також створений алгоритм використовує менше кубітів, що і зменшує час виконання. Оскільки не було знайдено коду, який представляє собою вбудований алгоритм, можна лише припустити, що цей алгоритм використовує більше кубітів для зменшення кількості виміряних результатів, які є неприйнятними і вважаються помилкою. Недоліком створеного алгоритму є саме його ймовірнісний характер, що призводить до зростання часу виконання при невдалій спробі вимірювання.

ВИСНОВОК

В результаті виконання дипломного проекту було створено квантову програму факторизації цілих не більших 21. В цій програмі застосовуються дві функції factorn_21 і factoring_n21.

Функція factorn_21() базується на алгоритмі Шора для чисел, які є непарними та з парною періодичністю. В цій функції для інших чисел використовується функція factelse(), яка не використовує алгоритм Шора, але все одно будує квантові схеми і використовує кубіти для розкладання чисел на множники.

Функція factoring_n21() виконує класичну частину алгоритму Шора і повторює алгоритм, поки не буде знайдено принаймні один множник для чисел, які задовольняють критерії необхідних для алгоритму Шора.

В кінці роботи було досліджено, яка кількість кубітів необхідна для створення квантової схеми, що факторизує числа як для алгоритму Шора, так і для створеної функції factelse(). Досліджено, яка кількість часу витрачається на виконання побудованого алгоритму Шора і порівнянно з часом виконання вбудованого алгоритму Шора.

В результаті дослідження можна зробити висновок, що створений алгоритм факторизації втрачає менше часу на розкладання чисел з використанням алгоритму Шора, використовує менше кубітів для побудови квантової схеми у порівнянні з вбудованим алгоритмом. Але створений алгоритм факторизації може призводити до неприйнятних результатів, оскільки він має ймовірнісний характер. Можна лише припустити, що вбудований алгоритм Шора в IBM Q використовує більше кубітів для того, щоб ці результати відкинути та підвищити ймовірність вимірювання потрібного результату.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Barry C. Sanders Institute for Quantum Science and Technology, University of Calgary, Calgary, AB, Canada [Електронний ресурс]. – Режим доступу: https://physics.aps.org/articles/v14/147
2. Wang, Y., Um, M., Zhang, J. et al. Single-qubit quantum memory exceeding ten-minute coherence time. Nature Photon 11, 646–650 (2017). [Електронний ресурс]. – Режим доступу: https://doi.org/10.1038/s41566-017-0007-1
3. IBM's Test-Tube Quantum Computer Makes History; First Demonstration Of Shor's Historic Factoring Algorithm [Електронний ресурс]. – Режим доступу: https://www.sciencedaily.com/releases/2001/12/011220081620.htm
4. Nanyang Xu, et al. “Quantum Factorization of 143 on a Dipolar-Coupling Nuclear Magnetic Resonance System.” PRL 108, 130531 (2012) [Електронний ресурс]. – Режим доступу: https://phys.org/news/2012-04-largest-factored-quantum-algorithm.html
5. Amir H. Karamlou, William A. Simon, Amara Katabarwa, Travis L. Scholten, Borja Peropadre, Yudong Cao Analyzing the performance of variational quantum factoring on a superconducting quantum processor (2021). [Електронний ресурс]. – Режим доступу: https://www.nature.com/articles/s41534-021-00478-z
6. Quantum computer sets new record for finding prime number factors [Електронний ресурс]. – Режим доступу: https://www.newscientist.com/article/2227387-quantum-computer-sets-new-record-for-finding-prime-number-factors/
7. Realization of a multimode quantum network of remote solid-state qubits M. Pompili, S. L. N. Hermans, S. Baier, H. K. C. Beukers, P. C. Humphreys, R. N. Schouten, R. F. L. Vermeulen, M. J. Tiggelman, L. dos Santos Martins, B. Dirkse, S. Wehner, R. Hanson Science, Vol. 372, Issue 6539 (2021) [Електронний ресурс]. – Режим доступу: https://www.science.org/doi/10.1126/science.abg1919

8. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits Craig Gidney, Martin Ekera 021-04-15, volume 5, с. 433 [Електронний ресурс]. – Режим доступу: https://quantum-journal.org/papers/q-2021-04-15-433/
9. Scott Buchholz, Joe Mariani, Adam Routh, Akash Keval, Pankaj Kamleshkumar Kishnani The realist's guide to quantum technology and national security [Електронний ресурс]. – Режим доступу: https://www2.deloitte.com/us/en/insights/industry/public-sector/the-impact-of-quantum-technology-on-national-security.html
10. Fred Guterl As China Leads Quantum Computing Race, U.S. Spies Plan for a World with Fewer Secrets (2022) [Електронний ресурс]. – Режим доступу: https://www.newsweek.com/2020/12/25/china-leads-quantum-computing-race-us-spies-plan-world-fewer-secrets-1554439.html
11. The Turn to Infrastructure in Internet Governance Francesca Musiani, Derrick L. Cogburn, Laura DeNardis, Nanette S. Levinson Springer, 2 бер. 2016 р. - 268 с.
12. Kamra, S. and J. Scott (2019). "Impact of Data Breaches to Organizations and Individuals." [Електронний ресурс]. – Режим доступу: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3510590
13. Quantum cryptography based on Bell's theorem Artur K. Ekert Phys. Rev. Lett. 67, 661 – 5 August 1991 [Електронний ресурс]. – Режим доступу: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.67.661
14. P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, с. 124, 1994.
15. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120, 1978.
16. G. S. Vernam. Cipher printing telegraph systems for secret wire and racho telegraphm communications. J. AIEE, с. 109, 1926.
17. C. Shannon. Communication theory of secrecy systems. Bell System Technical Journal, 28(4):656, 1949.
18. GUANCO, F. (2015). "What is Quantum Key Distribution." Cloud Security Alliance.

19. Ma, X. (2008). "Quantum cryptography: theory and practice." [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/0808.1385.pdf
20. T. W. Marshall, E. Santos, and F. Selleri. Local realism has not been refuted by atomic cascade experiments. Phys. Lett. A, 98:5, 1983.
21. M. Curty, M. Lewenstein, and N. Lutkenhaus. Entanglement as precondition for secure quantum key distribution. Phys. Rev. Lett., 92:217903, 2004.
22. S. Akama, Elements of quantum computing. Springer, 2015.
23. A. Gepp and P. Stocks, “A review of procedures to evolve quantum algorithms,” Genetic programming and evolvable machines, vol. 10, no. 2, с. 181–228, 2009.
24. M. A. Nielsen and I. Chuang, “Quantum computation and quantum information,” 2002.
25. Zhang D. Y. and Tan Y. K. 2019 Classical logic gates and quantum logic gates [J] Journal of Hengyang Normal University 40(03) с. 16-23
26. Xia P. S. 2001 Quantum Computing [J] Computer Research and Development, 10 с. 1153-1171
27. Jennewein, T., et al. (2000). "Quantum cryptography with entangled photons." Physical review letters 84(20): 4729.
28. Wu N. and Song F. M. 2007 Quantum computing and quantum computers [J] Computer Science and Exploration 01 с. 1-16
29. Samuel J. and Lomonaco Jr. 2002 Shor’s Quantum Factoring Algorithm. Proceedings of symposium in Applied Mathematics 58 с. 161-180
30. Guo G. C. and Zhang H. and Wang Q. 2017 Overview of Quantum Information Technology Development [J] Journal of Nanjing University of Posts and Telecommunications (Natural Science Edition) 37(03) с. 1-14
31. IBM Research Editorial Staff. (2018). “Qiskit for quantum computation” [Електронний ресурс]. – Режим доступу: https://www.ibm.com/blogs/research/2018/02/qiskit-index/
32. Qiskit documentation. “The Qiskit Elements” [Електронний ресурс]. – Режим доступу: https://qiskit.org/documentation/stable/0.24/the_elements.html