Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Fizyki, Matematyki i Informatyki

Marcin Bocheński

Numer albumu: 104844

Algorytm kwantowy Grovera

Grover's quantum algorithm

Praca magisterska
na kierunku FIZYKA TECHNICZNA

Praca wykonana pod kierunkiem:
dr Radosław Kycia

Uzgodniona ocena:.....

.....po
dpisy promotora i recenzenta

Kraków 2019

Spis treści

Streszczenie

Postęp technologiczny na przełomie XX i XXI wieku umożliwił szybki rozwój komputerów i pozwolił między innymi na zwiększenie ich mocy obliczeniowej, zmniejszenie gabarytów oraz spadek ceny. Komputery oparte na klasycznym podejściu do informacji nie są w stanie rozwiązać szybko i wydajnie 'złożonych' problemów obliczeniowych takich jak na przykład faktoryzacja liczb naturalnych oraz wyszukiwanie określonego elementu w nieuporządkowanej bazie danych. Jednak komputery kwantowe oparte o zasady mechaniki kwantowej są w stanie rozwiązać te problemy efektywnie.

W pracy opisano implementację, na komputerze klasycznym w oparciu o symulację bramek kwantowych, algorytmu Grovera do przeszukiwania bazy danych, który pozwala na kwadratowe przyspieszenie czasu wykonania zadania względem algorytmów klasycznych. Program napisany w języku Python został z powodzeniem użyty do przeszukania zbioru danych. Przedstawiona została również implementacja dla komputera kwantowego IBM Q.

Abstract

Technological progress at the turn of the 20th and 21st centuries enabled rapid computers development and allowed to enhance their computing power, reduce size and decrease prices. Computers based on classical information are not able to solve many present 'difficult' computing problems like for example prime factorization or searching of specific element in unsorted database quickly and efficiently. However, quantum computers based on quantum mechnics can solve such problems effectively.

In this thesis Grover’s algorithm implementation on classical computer was desrcibed in order to search database. The algorithm speeds up the task by square root comparing to classical algorithms. Code written in Python was successfuly used to search dataset. Implementation for IBM Q quantum computer was also presented.

Wyjaśnienie użytych skrótów i nazw

ACID – Atomicity, Consistency, Isolation, Durability

CERN – Europejska Organizacja Badań Jądrowych

CMOS - Complementary Metal–Oxide–Semiconductor

CPU – Central Processing Unit

DBMS – Database Management System

GIS – Geographic Information System

NASA – Narodowa Agencja Aeronautyki i Przestrzeni Kosmicznej rządu USA

PC – Personal Computer

Python –wysokopoziomowy język programowania ogólnego przeznaczenia

SQL – Structured Query Language

SZBD – System Zarządzania Bazą Danych

Cel, zakres i metodyka pracy

Celem pracy jest implementacja kwantowego algorytmu Grovera i zastosowanie go do przeszukania bazy danych. Platformą uruchomieniową będzie komputer klasyczny klasy PC, a środowiskiem uruchomieniowym będzie język programowania Python, w którym zostanie zasymulowane działanie komputera kwantowego oraz wykonanie omawianego algorytmu. Na potrzeby projektu wygenerowany zostanie prosty zbiór danych, który następnie będzie przeszukiwany za pomocą algorytmu Grovera. W rzeczywistej implementacji należałoby sprząc bazę danych z komputerem kwantowym.

W części teoretycznej przedstawione zostaną podstawowe zagadnienia informatyki klasycznej oraz kwantowej – opisany zostanie sposób przechowywania danych oraz wykonywania operacji. Wyjaśniona zostanie zasada działania kwantowego algorytmu Grovera oraz jego zastosowanie, a także istota systemów bazodanowych.

Część praktyczna zawierać będzie opis wykonanej pracy implementacyjnej oraz prezentację i dyskusję otrzymanych rezultatów. Omówione zostanie zagadnienie, a także zaproponowane podejście do jego rozwiązania. W języku Python zostanie zaimplementowana logika komputera kwantowego – bramek kwantowych, a także przedstawione zostanie działanie algorytmu Grovera. Przedstawiona zostanie również implementacja algorytmu dla komputera kwantowego IBM Q.

Niniejsza praca ma interdyscyplinarny charakter – poruszy zagadnienia z zakresu informatyki, fizyki oraz matematyki.

I. Część teoretyczna

1. Wstęp

Jednym z problemów, często spotykanym we współczesnej informatyce jest wyszukiwanie, w strukturze zawierającej dane, elementów o określonym wzorcu. W zależności od liczby informacji jakimi dysponujemy na temat kolekcji, można posłużyć się wieloma algorytmami w celu szybszego osiągnięcia rezultatu. Własnościami, które wpływają na szybkość rozwiązania problemu są między innymi: uporządkowanie, zgrupowanie oraz unikalność danych.

W przypadku nieposortowanego zestawu danych (struktura taka często nazywana jest listą), na temat którego nie dysponujemy żadną dodatkową wiedzą, jednym rozwiązaniem jest sprawdzenie wszystkich jego elementów po kolei. Wiąże się to z wysoką, liniową, złożonością obliczeniową algorytmu równą $O(n)$, gdzie n jest liczbą danych w zestawie[1]. Zatem złożoność jest wprost proporcjonalna do rozmiaru przeszukiwanego zbioru danych. W sytuacji, gdy nie posiadamy żadnej dodatkowej informacji o bazie danych, jest to jedyna droga rozwiązania problemu.

Inne podejście do rozwiązania tego problemu umożliwia informatyka kwantowa, która w XXI wieku dynamicznie rozwija się zarówno w obszarze teoretycznym jak i praktycznym. W 1996 roku Lov Grover[2] przedstawił algorytm kwantowy przeszukiwania nieuporządkowanej bazy danych w poszukiwaniu określonego elementu. Pozwala on na wyznaczenie pozycji elementu o danej właściwości w zadanym zbiorze danych. Złożoność czasowa tego algorytmu jest równa $O(\sqrt{n})$, umożliwia zatem kwadratowe przyspieszenie czasu wykonania operacji względem algorytmów wykonywanych na komputerach klasycznych. Algorytm kwantowy wyszukiwania może być także użyty jako podprogram bardziej złożonych algorytmów kwantowych.

W kolejnym rozdziale opisane zostaną podstawowe zagadnienia współczesnej informatyki.

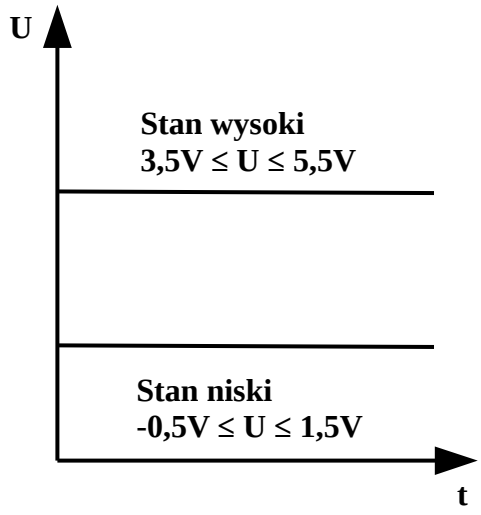
2. Podstawy informatyki

Informatyka jest interdyscyplinarną dziedziną nauki zajmującą się informacją oraz jej przetwarzaniem. Jej nieodłączną część stanowią obecnie komputery, przy użyciu których takie przetwarzanie najczęściej się odbywa. Wraz z postępem technologicznym i wzrostem liczby użytkowników komputerów, nauka ta rozwinęła się bardzo szybko i nadal dynamicznie rozwija.

2.1. Nośnik informacji

Podstawowym pojęciem związanym z przetwarzaniem informacji jest jej nośnik – przedmiot fizyczny umożliwiający zapisanie informacji oraz jej późniejsze odczytanie[3]. Informację możemy rozumieć jako wynik oddziaływania/odpytania źródła. Każdą taką odpowiedź możemy zakodować używając elementów zbioru nazywanego alfabetem A . W przypadku informacji binarnej alfabet składa się z dwóch elementów - zazwyczaj jest to $A=\{0,1\}$. Poszczególne elementy nazywają się bitami (b). Wówczas informacja zakodowana w sekwencji bitów jest sekwencją odpowiedzi na pytania, w których wynikiem może być jedna z dwóch odpowiedzi, np. Tak i Nie, stowarzyszone z 0 i 1 z alfabetu. Współczesne komputery posługują się takim sposobami binarnym zapisu informacji. Bit przyjmuje wyłącznie jedną umowną wartość: 0 lub 1; jest to odpowiednik dwóch stanów znanych z logiki: prawdy i fałszu.

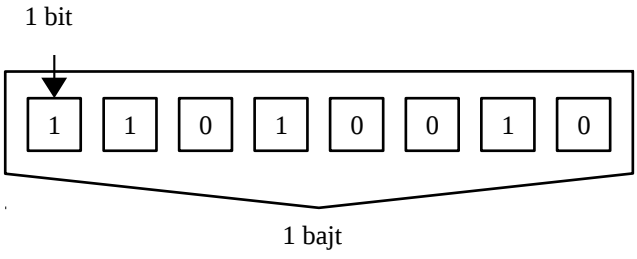
Kodowanie informacji w układach elektronicznych możliwe jest przez wybranie dwóch rozróżnialnych wartości parametru służącego do reprezentacji bitu. Mogą to być na przykład dwa różne stany napięcia lub natężenia prądu, magnetyzacji lub natężenia światła. W obwodach logicznych opartych na technologii CMOS[4] do reprezentacji bitu używa się dwóch przedziałów napięć: wysokiego oraz niskiego. Wartość potencjału elektrycznego stanu wysokiego, w stosunku do uziemienia, jest zawsze większa niż stanu niskiego.



Rysunek 1. Dwa poziomy napięcia, służące do reprezentacji bitu (przykład dla układu CMOS).

Komórki pamięci, które są w stanie przechowywać jeden bit, organizowane są w ciąg zdolny do przechowywania kilku bitów i nazywane są rejestrami. Rejestr zdolny pomieścić N bitów przechowuje elementy z iloczynu kartezjańskiego $A^N = A \times A \times ... \times A$.

Jednostką opisującą 8 bitów jest 1 bajt (1 B). Bajt umożliwia zapisanie $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ unikalnych wartości większych od zera. Jeden bajt pozwala zatem na przedstawienie liczby z przedziału $\langle 0, 255 \rangle$ [5].



Rysunek 2. Reprezentacja 1 bajtu.

Bajt oraz jego wielokrotności używany jest obecnie do opisu miary wielkości pamięci. Najczęściej spotykanymi obecnie jednostkami pamięci są: $1KB = 10^3 B$,

$1\,MB = 10^6\,B$ oraz $1\,GB = 10^9\,B$. Poza przedrostkami jednostek miar układu SI, w informatyce stosuje się również przedrostki dwójkowe o mnożniku $2^{10} = 1024$ (zamiast $10^3 = 1000$) , na przykład: $1\,KiB = 2^{10}\,B$ oraz $1\,GiB = 2^{30}\,B$ [6].

2.2. Logika układów cyfrowych

Ze względu na odpowiedniość pomiędzy literami alfabetu $A = \{0,1\}$, a wartościami logicznymi Prawda i Fałsz możemy zaadoptować logiczne operacje na gruncie informatyki i elektroniki.

Do ujęcia logiki w sposób algebraiczny używa się algebry Boole'a[7]. Jest to rodzaj algebry abstrakcyjnej, zdefiniowany w następujący sposób:

$$B = (b, \wedge, \vee, \neg, 0, 1) \ ,$$

gdzie:

b - zbiór;

$\wedge, \vee$ - operacje dwuargumentowe na elementach b ;

$\neg$ - negacja,

$0,1$ - wyróżnione elementy zbioru b , spełniające następujące prawa dla wszystkich $x, y, z \in b$:

łączności	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
przemienności	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
absorpcji	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$
tożsamości	$x \vee 0 = x$	$x \wedge 1 = x$
rozdzielności	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
pochłaniania	$x \vee \neg x = 1$	$x \wedge \neg x = 0$

W logice (będącej przykładem algebry Boole'a) operacje koniunkcji, alternatywy i negacji zdefiniowane są w następujący sposób:

I. koniunkcja $\wedge : A \times A \rightarrow A$ jest funkcją dwuargumentową określoną następująco (funkcja *min()* wybiera minimalną wartość x,y traktując elementy/symbole 0 (fałsz) i 1 (prawda) jako liczby całkowite):

$$x \wedge y = min(x, y) \quad .$$

Tablica prawdy:

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

II. alternatywa $\vee : A \times A \rightarrow A$ jest funkcją dwuargumentową prawdziwą, wtedy i tylko wtedy gdy co najmniej jeden z argumentów jest prawdziwy.

Tablica prawdy:

x	y	$x \vee y$
0	0	0
1	0	1
0	1	1
1	1	1

III. negacja $\neg : A \to A$ jest funkcją jednoargumentową określoną następująco (tutaj również 0 oraz 1 traktujemy jako liczy naturalne, w których jest zdefiniowana operacja odejmowania):

$$\neg x = 1 - x \quad .$$

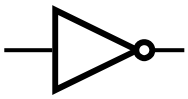
Tablica prawdy:

x	$\neg x$
0	1
1	0

Trzy powyższe operacje pozwalają odtworzyć wszystkie inne operacje używane w logice[8].

Układy elektroniczne realizujące wyżej wymienione operacje noszą nazwę bramek logicznych. Bramki o jednym wejściu i wyjściu noszą nazwę jednobitowych. Bramki mające więcej wejść i wyjść nazywamy wielobitowymi, np. AND lub OR. Warunkiem koniecznym do odwracalności wykonanej operacji jest odwracalność funkcji logicznej realizowanej przez daną bramkę.

Jedną z najprostszych bramek logicznych jest bramka NOT, posiadająca tylko jedno wejście oraz jedno wyjście i realizująca operację negacji. Bramka NOT jest odwracalna, gdyż definiuje bijekcję pomiędzy wejściem i wyjściem.

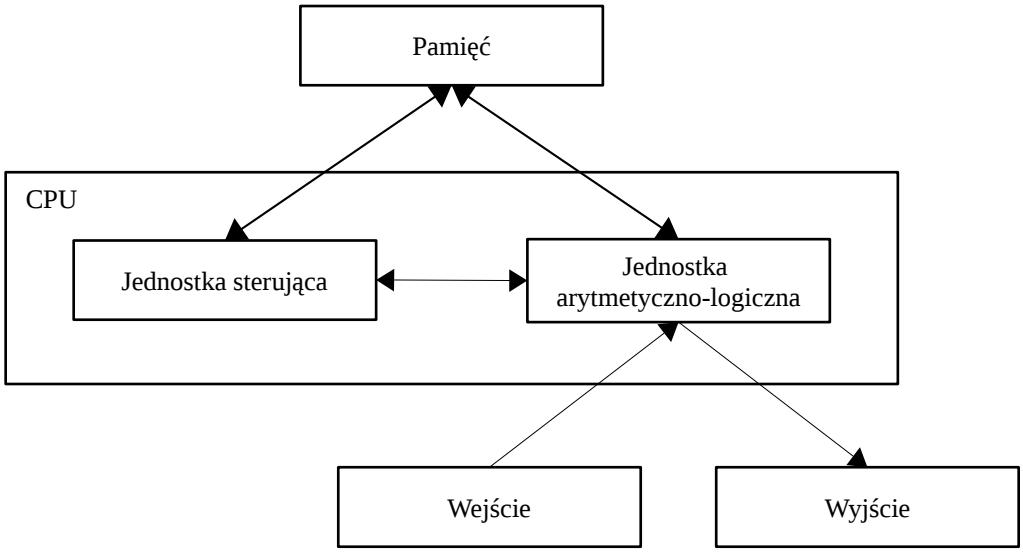


Rysunek 3: Symbol bramki NOT.

Bramki możemy łączyć ze sobą spinając wyjścia bramek z wejściami innych. To umożliwia tworzenie skomplikowanych elementów będących częściami każdego elektronicznego komputera.

Algorytm jest sposobem wykonania konkretnego zadania zapisanym w postaci jednoznacznych kroków. Algorytm przełożony na język zrozumiały dla komputera - program, może być wykonany przez komputer. Program ten jest wykonywany poprzez operacje na bramkach logicznych. Współczesne komputery, zbudowane w oparciu o *architekturę von Neumanna*[9], wykonują obliczenia przy pomocy jednostki centralnej (CPU). W skład CPU wchodzi jednostka sterująca, która nadzoruje wykonanie algorytmu/programu, oraz jednostka arytmetyczno-logiczna wykonująca obliczenia[10].

Architektura von Neumanna realizuje koncepcję składowania programu i danych we wspólnej pamięci, a przez to komputer może być uniwersalny - łatwo przeprogramowany do wykonywania nowego zadania. Poprzednie architektury miały rozdzielone miejsca w których składowano dane i programu, a często programy były 'ustawiane' przez manualną zmianę połączeń elektronicznych komputera.



Rysunek 4: Architektura von Neumanna.

Wyróżniamy kilka paradygmatów pisania programów. Najczęściej stosowanym jest paradygmat imperatywny, w którym specyfikujemy jakie operacje komputer ma wykonać aby wyprodukować wynik algorytmu. Jest to paradygmat najczęściej wykorzystywany we współczesnych, popularnych w zastosowaniach językach programowania. Innym podejściem jest paradygmat deklaratywny, w którym program składa się z opisu tego, co chcemy otrzymać, natomiast komputer (na którym działa

interpreter języka) ma to zadanie za nas wykonać. Ten paradygmat wymaga stworzenia narzędzia do interpretacji zadań. Ostatnim paradygmatem jest paradygmat funkcyjny, w którym program jest ciągiem wywołań (matematycznych) funkcji na danych wejściowych. Ten paradygmat ma solidne podstawy matematyczne i jest wykorzystywany najczęściej w zadaniach ‘trudnych’ i rozważaniach teoretycznych.

2.3. Złożoność obliczeniowa

Do opisu efektywności algorytmu stosuje się pojęcie złożoności obliczeniowej, określając ilość zasobów potrzebnych do wykonania obliczeń[11][12]. W przypadku czasowej złożoności obliczeniowej – liczbę wykonanych operacji podstawowych w zależności od rozmiaru danych wejściowych, a w przypadku złożoności pamięciowej – ilość pamięci wykorzystanej do wykonania obliczeń. Do zapisu złożoności obliczeniowej algorytmów stosuje się *notację dużego O*, która asymptotycznie opisuje jak zachowuje się funkcja w konkretnej granicy:

$$f(x)=O(g(x)), \text{ dla } x\to a,$$

wtedy i tylko wtedy, gdy:

$$\limsup_{x\to a}\left|\frac{f(x)}{g(x)}\right|<\infty.$$

Zazwyczaj ‘a’ jest przyjmowane jako nieskończoność w obliczeniach złożoności. Za jej pomocą oszacować można możliwy czas wykonania danego algorytmu lub maksymalne zapotrzebowanie na zasoby w trakcie jego działania.

Przykładowo dla nieuporządkowanej listy zawierającej N unikalnych elementów, w najgorszym przypadku, gdy poszukiwany element jest ostatnim elementem listy, musimy odczytać i porównać ze wzorcem wszystkie elementy zbioru - złożoność obliczeniowa jest równa $O(N)$. Zatem algorytm skaluje się liniowo w stosunku do wielkości listy N.

Warto zauważyć, że notacja O nie uwzględnia czynników stałych, na przykład $O(2N)=O(N)$, a także członów niższego rzędu np. $O(N^2)=O(N^2+2N)$.

3. Baza danych

Baza danych jest zbiorem wzajemnie powiązanych ze sobą informacji, służącym do przechowywania określonych danych, często o podobnym do siebie charakterze. Rozwiązanie to jest znane i stosowane od dawna. Zanim powstały elektroniczne bazy danych, różne instytucje posiadały zbiory najczęściej w formie papierowej, na przykład urzędy, biblioteki itp. Wraz z postępem technologicznym i cyfryzacją społeczeństwa tradycyjne bazy danych zaczęto zastępować elektronicznymi systemami.

W ujęciu informatycznym baza danych to zbiór materiałów cyfrowych zapisanych zgodnie z określonymi regułami, kontrolowany przez dedykowany program komputerowy – tak zwany system zarządzania bazą danych, w skrócie SZBD (ang. DBMS – Database Management System), pozwalający na zapis, odczyt oraz modyfikację danych. Służy on do zarządzania i obsługi bazy danych, czyli wykonywania operacji takich jak między innymi tworzenie nowych elementów w obrębie bazy oraz manipulowanie danymi. Systemy zarządzania bazą danych odpowiada za mechanizmy do odczytu, definiowania i dodawania nowych danych, a także edycji lub usuwania istniejących[13]. Operowanie na danych odbywa się poprzez wykonywanie zapytań bazodanowych. Tworzone zapytania nazywane są kwerendami. System zarządzania bazą danych może również odpowiadać za monitorowanie wydajności bazy oraz administrację dostępem do danych.

W celu sprostania zróżnicowanym wymaganiom dotyczącym funkcjonalności systemu bazodanowego, powstało wiele systemów baz danych, oferujących określone możliwości. Z technologicznego punktu widzenia, wśród najpopularniejszych rodzajów baz danych można wyróżnić następujące rodzaje baz danych:

- relacyjne - dane mają strukturę tabel powiązanych relacjami; obecnie najpopularniejszy typ baz danych; językiem komunikacji z taką bazą jest SQL,
- nierelacyjne (ang. NoSQL) - dane nie mają wyraźnej struktury,
- grafowe - powiązanie danych przedstawione jest za pomocą grafów,
- obiektowe - baza przechowuje obiekty - instancje klas;
- obiektowo-relacyjne - obiekty są składowane w strukturze relacyjnej bazy danych.

Umożliwiają one, w zależności od dostarczanego rozwiązania, przechowywanie danych w różnych strukturach powiązanych w odmienny sposób.

Najpopularniejszym rodzajem bazy danych jest baza relacyjna[14]. Dane wewnątrz bazy zapisywane są w formie powiązanych relacjami tabel, które zawierają informacje (encje) o określonym przez kolumnę charakterze i typie. Poniżej przedstawiona została przykładowa tabela, zawierająca listę kontaktów:

Telefon	**Imię**	**Nazwisko**
808080	Jan	Kowalski
505050	Anna	Nowak
707070	Maria	Lewandowska
606060	Dariusz	Wiśniewski

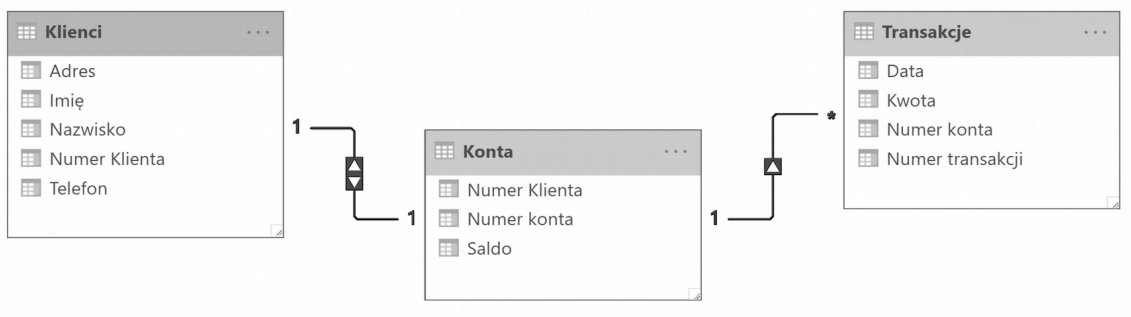
Najpopularniejszym językiem do tworzenia zapytań bazodanowych jest deklaratywny język SQL, będący również oficjalnym międzynarodowym standardem[15]. W oparciu o powyższą tabelę kontaktów, przykładowe zapytanie zwracające informacje o posiadaczu wyszczególnionego numeru telefonu, można zapisać w postaci:

***SELECT** Imię, Nazwisko **FROM** kontakty **WHERE** Telefon = 606060; .*

Wynikiem powyższego zapytania będzie *Imię* i *Nazwisko* rekordów, dla których wartość pola *Telefon* jest równa *606060*.

W celu ułatwienia referencji do danych często do tabeli dodaje się pole z unikalnym identyfikatorem, bądź używa jednego z istniejących pól jako identyfikatora. W relacyjnych bazach danych jednoznaczna pole jednoznacznie identyfikujące wiersz/rekord nazywa się kluczem głównym.

Integralną częścią każdej relacyjnej bazy danych jest jej model logiczny - schemat, który definiuje tabele oraz relacje pomiędzy nimi. W celu uniknięcia powtarzania się tych samych informacji wewnątrz bazy, relacje projektuje się zgodnie z pierwszą, drugą oraz trzecią postacią normalną[16]. Jako przykład, poniżej przedstawiony został schemat prostej bazy danych, składającej się z trzech tabel, w których zawarte są informacje o klientach, kontach i transakcjach bankowych.



Rysunek 5. Przykładowy schemat logiczny relacyjnej bazy danych.

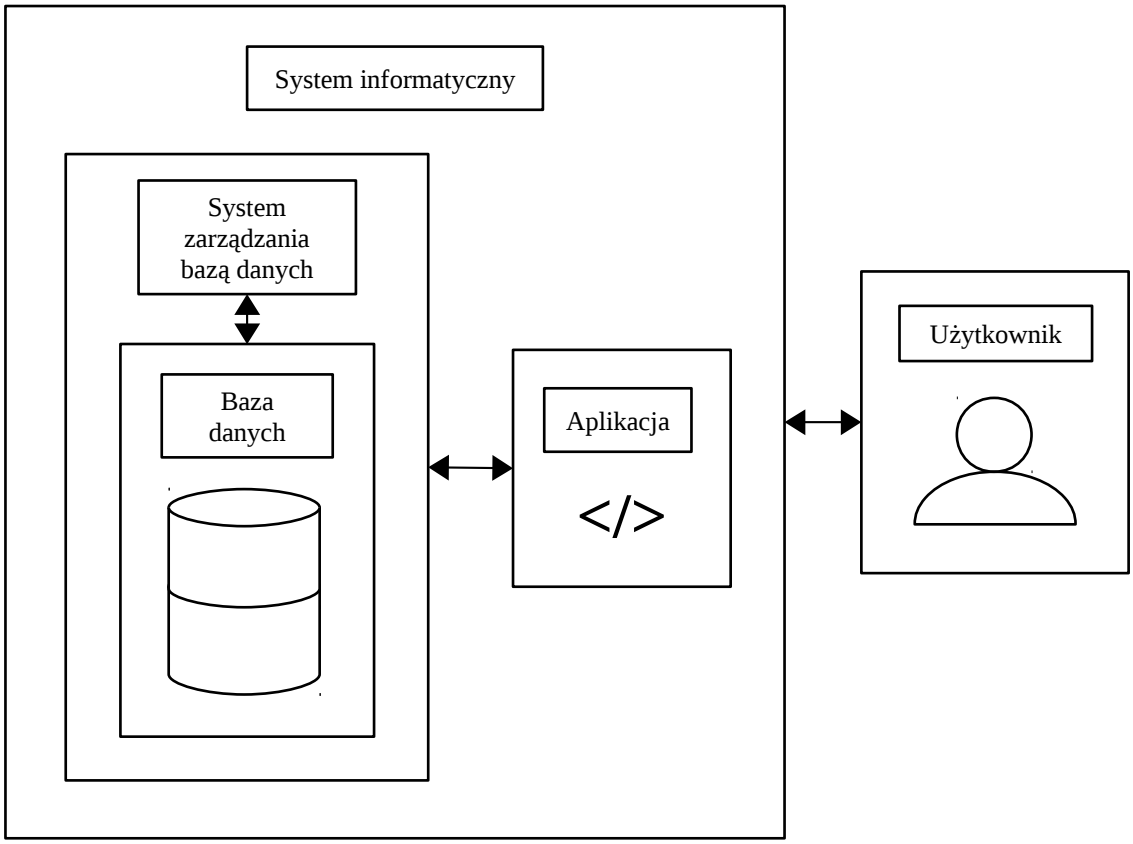
Sekwencja operacji modyfikujących zawartość bazy danych agreguje się w postaci transakcji, przy czym każdy taki zbiór jest niepodzielny, to znaczy muszą zostać wykonane wszystkie zdefiniowane w nim operacje lub żadna z nich. Zasady, jakie powinna spełniać każda transakcja, definiują reguły ACID (ang. **A**tomicity, **C**onsistency, **I**solation, **D**urability) – niepodzielności, spójności, izolacji i trwałości. Dotyczą one nie tylko operacji pojedynczej transakcji, ale także zależności pomiędzy różnymi transakcjami. Większość współczesnych SZBD wspiera, w różnym stopniu, operacje w postaci transakcji.

W obrębie bazy może być zdefiniowana także dodatkowa warstwa dostępu do danych poprzez elementy takie jak na przykład: widoki, procedury składowane i wyzwalacze. Ułatwiają one wykonywanie predefiniowanych operacji na bazie danych i umożliwiają na przykład automatyzację zadań oraz kontrolowane ograniczenie dostępu do danych.

3.1. Zastosowanie

W zależności od zastosowanego systemu zarządzania bazą danych oraz jego przeznaczenia w bazie mogą znajdować się różne typy danych, takich jak na przykład: dane liczbowe, dane znakowe oraz bardziej złożone struktury, na przykład: obrazy, utwory dźwiękowe, pliki, dane Systemu Informacji Geograficznej itp. Z punktu widzenia informatyki nie jest istotny charakter przechowywanych danych tylko ich typ.

Elektroniczne bazy danych są obecnie powszechnie stosowane w systemach informatycznych. Na ogół stanowią one określoną część większych, bardziej złożonych, systemów informatycznych takich jak aplikacje czy strony internetowe.



Rysunek 6. Diagram systemu informatycznego zawierającego bazę danych.

Bazy danych umożliwiają przechowywanie różnych treści, na przykład informacji odnośnie zbioru książek i czytelników w bibliotece, listy oferowanych artykułów w sklepie internetowym, historii transakcji w systemie bankowym czy danych geograficznych (systemy GIS).

4. Informatyka kwantowa

Jako rozwinięcie dotychczas istniejącego dziedziny nauki - informatyki, poprzez połączenie jej z mechaniką kwantową zapoczątkowano erę informatyki kwantowej, która obecnie dynamicznie się rozwija. Ideę obliczeń kwantowych, jako pierwszy, zaprezentował w 1982 roku Richard Feynman w swojej pracy[17] poświęconej teorii symulacji zjawisk fizycznych przy pomocy komputera i koncepcji nowego rodzaju maszyny, zdolnego do efektywnej symulacji zjawisk kwantowych.

Dzięki wykorzystaniu zjawisk kwantowych zamiast praw klasycznych i zastąpieniu tradycyjnych układów układami kwantowymi możliwe jest zastosowanie nowej elementarnej jednostki informacji – kwantowego bitu, zwanego *kubitem*, który może być realizowany przez dowolny kwantowy układ dwustanowy. W przeciwieństwie do klasycznego bitu, nowa jednostka może przyjmować więcej niż tylko dwa stany - unormowaną zespoloną kombinację liniową dwóch stanów bazowych, tzn. wszystkie możliwe stany takiego układu leżą na sferze Blocha[18]. Dzięki temu zamiast dwóch stanów klasycznych mamy możliwość użycia nieprzeliczalnej liczby punktów na sferze Blocha jako stanów.

W ramach informatyki kwantowej opracowuje się algorytmy kwantowe[19], które na nowo, w sposób szybszy i bardziej efektywny rozwiązują istniejące problemy obliczeniowe, takie jak na przykład przeszukiwanie zbioru danych (algorytm Grovera[2]), rozkład liczb na czynniki pierwsze (algorytm Shora[20]) czy szybka kwantowa transformacja Fouriera (algorytm Kitajewa[21]). Obecnie lista algorytmów kwantowych jest obszerna[22] - najbardziej aktualne informacje można znaleźć na stronach NIST[23].

Ze względu na szybkość algorytmy te z powodzeniem wykorzystać można w wielu istniejących systemach informatycznych na przykład do przyspieszenia operacji bazodanowych i przetwarzania sygnałów, a także w kryptografii. Algorytmy kwantowe są w większości algorytmami probabilistycznymi – zwracają poprawną odpowiedź z pewnym prawdopodobieństwem. W celu zwiększenia poprawności wyniku wielokrotnie powtarza się algorytm. Zazwyczaj w łatwy sposób można również zweryfikować otrzymany wynik przed ponownym uruchomieniem algorytmu.

4.1. Jednostka informacji kwantowej

W informatyce kwantowej podstawową jednostką informacji jest kubit – zespolona kombinacja liniowa (zwanej dalej superpozycją) stanów bazowych, czyli stanu zero oraz stanu jeden, znanych z informatyki klasycznej.

W mechanice kwantowej stan układu jest opisywany wektorem (a dokładniej kierunkiem wyznaczanym przez ten wektor i nazywanym promieniem) z zespolonej przestrzeni Hilberta H.

Jest to zupełna przestrzeń liniowa nad ciałem liczb zespolonych, na której zdefiniowany jest nieosobliwy *iloczyn skalarny*:

$$\langle \phi, \psi \rangle : H \times H \to \mathbb{C} \quad ,$$

gdzie:

$\phi, \psi \in H$,

spełniający warunki:

$\langle \phi, \psi \rangle = \overline{\langle \psi, \phi \rangle}$,

$\langle \alpha\phi + \beta\psi, \eta \rangle = \alpha \langle \phi, \eta \rangle + \beta \langle \psi, \eta \rangle$; $\alpha, \beta \in \mathbb{C}; \eta \in H$,

$\langle \phi, \phi \rangle \geq 0$, $\langle \phi, \phi \rangle = 0 \Leftrightarrow \phi = 0$.

Wówczas normę wektora, mająca wartość rzeczywistą, możemy zdefiniować przy użyciu iloczyny wektorowego przy użyciu wzoru:

$$\|\phi\| = \sqrt{\langle \phi, \phi \rangle} \quad .$$

Długość wektora w mechanice kwantowej nie ma znaczenia (gdyż wektory stanu opisują zespoloną przestrzeń rzutową), ale ze względu na interpretację probabilistyczną przyjmuje się unormowanie do jedności.

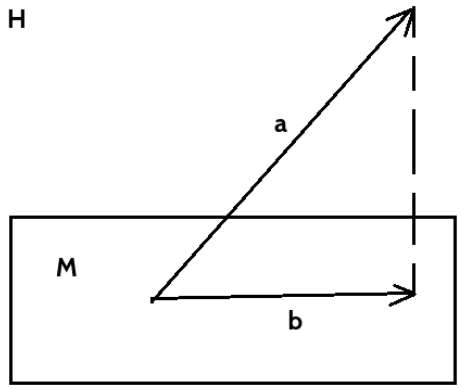
Iloczyn $\langle \phi, \psi \rangle$ interpretuje się jako ‘ilość’ stanu ϕ w stanie ψ lub vice versa jeżeli stany ϕ i ψ są unormowane do jedności.

Zupełność jest własnością przestrzeni topologicznej mówiącą, iż jeżeli nieskończony ciąg x_n z tej przestrzeni spełnia *warunek Cauchy'ego*:

$$\forall \epsilon > 0 \exists N \forall n,m > N |x_n - x_m| < \epsilon \quad ,$$

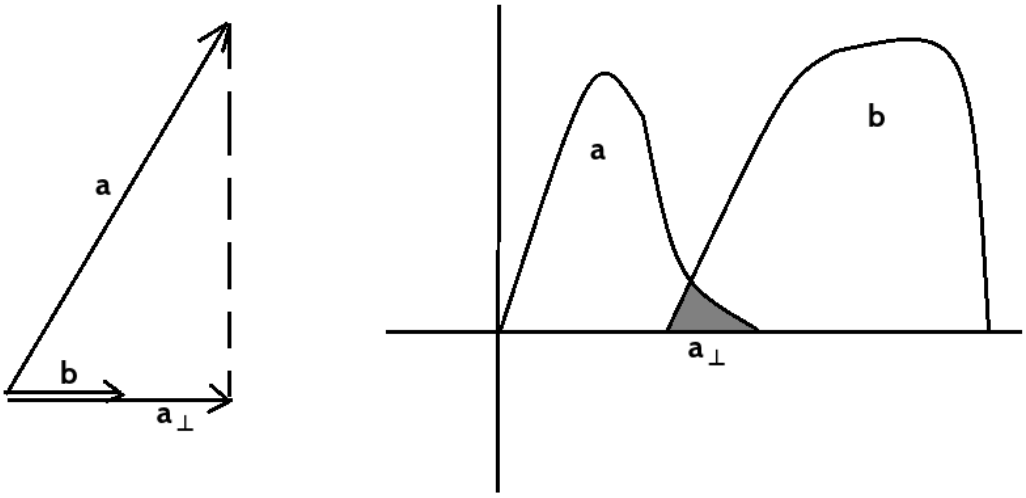
to w przestrzeni istnieje granica tego ciągu. Innymi słowy, jeżeli ciąg 'chce się zbiegnąć' spełniając warunek Cauchy'ego, to przestrzeń posiada element graniczny. Każdą przestrzeń możemy uzupełnić o brakujące granice tworząc zupełną przestrzeń. Najprostszym przypadkiem jest zbiór liczb wymiernych, który nie jest zupełny. Jednak granice ciągów liczb wymiernych zbiegają się do liczb wymiernych lub dają nowe liczby będące liczbami rzeczywistymi nie mającymi reprezentacji wymiernej, na przykład $\sqrt{2}$. Dołączając te granice do zbioru liczb wymiernych otrzymamy zbiór liczb rzeczywistych, który jest zupełny. Podobnie zbiór liczb zespolonych jest zupełny. Skończenie wymiarowe przestrzenie wektorowe (w tym przestrzenie Hilberta) są zupełne, gdyż każda taka przestrzeń jest izomorficzna do przestrzeni R^n w przypadku przestrzeni rzeczywistej lub C^n w przypadku zespolonej przestrzeni wektorowej, a te ostatnie są zupełne, gdyż iloczyn kartezjański przestrzeni zupełnych jest zupełny. W przypadku przestrzeni nieskończenie wymiarowych zazwyczaj przestrzeń należy uzupełnić, np. przestrzeń funkcji całkowalnych z kwadratem L^2, gdzie funkcje tak naprawdę są klasami, w których dwie funkcje w danej klasie różnią się na podzbiorze miary zero.

Przestrzeń Hilberta ze względu na zupełność i posiadanie iloczynu skalarnego jest często używana do różnego rodzaju aproksymacji, gdyż wspomniane struktury matematyczne pozwalają znaleźć element z podprzestrzeni najlepiej przybliżający dany element – zob. rysunek.



Wektor a jest przybliżany przez wektor b należący do (domkniętej) podprzestrzeni M przy użyciu rzutu prostopadłego na podprzestrzeń M, który daje iloczyn skalarny. Zupełność H i domkniętość M gwarantuje, że jeżeli weźmiemy ciąg Cauchy'ego wektorów z M, które zbiegają się do b, to wektor b również należy do M. Przykład ten pokazuje istotę pojęć wchodzących w skład definicji przestrzeni Hilberta.

Jeżeli więc mamy układ opisywany wektorem ψ , to domieszka stanu ϕ w stanie ψ jest zwykłym rzutem i określa jak bardzo nakładają się/oddziałują wektory stanu – zob. rysunek.



Na rysunku po lewej przedstawiono rzut wektora a na b, który określa jaka część stanu a nakłada się (oddziałuje) na stan b. Po prawej pokazano to samo dla nieskończonej przestrzeni Hilberta realizowanej przez funkcje całkowalne z kwadratem. Wówczas zacieniona część odpowiada iloczynowi skalarnemu stanów zadanych funkcją a i b. Określa to jaka część wektorów stanu/funkcji się nakłada/przekrywa, a przez to wyznacza wielkość obszaru potencjalnego oddziaływania.

W notacja Diraca[24] wektor z n wymiarowej zespolonej przestrzeni Hilberta można zapisać jako wektor kolumnowy w postaci $|\psi\rangle$, nazywanego ket:

$$|\psi\rangle = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{bmatrix}$$

Natomiast stan sprzężony hermitowsko do niego można zapisać w postaci wektora wierszowego (nazywanego bra), poprzez transpozycję i sprzężenie zespolone wektora kolumnowego:

$$\langle\psi| = \begin{bmatrix} c_1^* & c_2^* & \dots & c_i^* & \dots & c_n^* \end{bmatrix}$$

gdzie:

$i = 1, 2, \dots, n$

$c_i, c_i^* \in \mathbb{C}$.

Z matematycznego punktu widzenia wektory bra są funkcjonałami liniowymi na przestrzeni Hilberta należącymi do dualnej przestrzeni Hilberta H^* , to znaczy:

$$\langle\psi| : H \rightarrow \Re$$

zgodnie ze wzorem:

$$\langle\psi|(|\chi\rangle) = \langle\psi|\,\chi\rangle \quad .$$

Odpowiedniość pomiędzy przestrzenią H, a jej przestrzenią dualną zadaje iloczyn skalarny < , > zgodnie z podstawowym twierdzeniem Riesza o reprezentacji funkcjonału liniowego w przestrzeni Hilberta[25].

Stany bazowe kubitu można zapisać w dwuwymiarowej przestrzeni $dim(H)=2$ za pomocą notacji Diraca w następujący sposób:

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

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

Powyższe dwa stany bazowe odpowiadają klasycznej jednostce informacji – bitowi, z tą różnicą, że są wielkościami wektorowymi a nie skalarnymi.

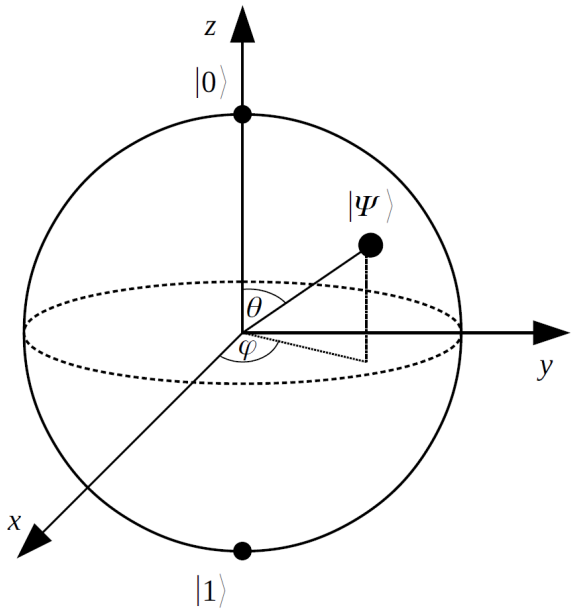
Kubit to unormowana zespolona kombinacja liniowa stanów bazowych w dwuwymiarowej przestrzeni Hilberta, którą można zapisać za pomocą następującego wyrażenia:

$$|\psi\rangle=\alpha|0\rangle+\beta|1\rangle=\begin{bmatrix}\alpha\\\beta\end{bmatrix} \quad ,$$

gdzie:

$\alpha,\beta\in\mathbb{C}$	- amplitudy prawdopodobieństwa,
$\|0\rangle,\|1\rangle$	- baza standardowa dwuwymiarowej, zespolonej przestrzeni Hilberta H,
$\|\alpha\|^2+\|\beta\|^2=1$	- unormowanie amplitud prawdopodobieństwa.

Takie stany leżą na wspomnianej we wstępie sferze Blocha przedstawionej na rysunku poniżej.



Rysunek 7: Sfera Blocha.

Uogólnioną jednostką informacji kwantowej jest **kudit**, czyli wektor opisujący układ kwantowy w $dim(H)=n$ wymiarowej przestrzeni Hilberta.

Kubit jest kuditem o $n=2$ stopniach swobody. Wówczas mamy n stanów bazowych $|0\rangle,\ldots,|n-1\rangle$, z których możemy tworzyć unormowane liniowe kombinacje zespolone.

Analogicznie jak w przypadku kubitu, stan kuditu można opisać w następujący sposób:

$$|\psi\rangle=\sum_{i=1}^{n}\alpha_i|i\rangle=\begin{bmatrix}\alpha_i\\ \vdots\\ \alpha_n\end{bmatrix} \quad ,$$

gdzie:

$i=1,2,\ldots,n$,

$\sum_{i=1}^{n}|\alpha_i|^2=1$.

Pomiar wartości kubitu jest realizowany przez rzutowanie go na wybrany stan, którego występowanie 'domieszki' chcemy w kubicie sprawdzić. Jest to operacja nieodwracalna.

Operacje na układzie kwantowym są reprezentowane przez operatory, które zachowują iloczyn skalarny, a przez to prawdopodobieństwo zdefiniowane przez ten iloczyn skalarny. Takie operatory zachowujące iloczyn skalarny nazywamy izometriami, w szczególności, dla przestrzeni Hilberta:

$$\langle U\psi | U\chi \rangle = \langle \psi | \chi \rangle \quad ,$$

co daje warunek definiujący izometrie zwane w przypadku przestrzeni zespolonych operatorami unitarnymi:

$$U^{\dagger}U = 1 \ ,$$

gdzie symbol † oznacza sprzężenie hermitowskie, które w przypadku reprezentacji operatorów w postaci macierzowej jest złożeniem sprzężenia zespolonego i transpozycji macierzy.

Po wybraniu bazy przestrzeni Hilberta możemy taki operator liniowy zapisać w postaci macierzy, a przez to wszystkie operacje sprowadzają się do operacji macierzowych znanych z algebry liniowej.

Operacja pomiaru jest nieodwracalna, a przez to nie może być operacją unitarną w której nie jest zachowywane prawdopodobieństwo. Taki „kolaps” nieunitarny funkcji falowej podczas pomiaru jest nie do końca zrozumianą zagadką mechaniki kwantowej.

Podobnie jak w przypadku klasycznym, kubity (kudity) możemy magazynować w układach nazwanych rejestrami kwantowymi. Odpowiadają one złożeniu układów kwantowych opisywanych przestrzeniami Hilberta $H_1, \ldots, H_N$ i są opisywane wektorami w iloczynie tensorowym tych przestrzeni:

$$H_1 \otimes \ldots \otimes H_N \quad .$$

W zapisie stanów takiego złożonego układu zazwyczaj pomija się iloczyn tensorowy i zamiast:

$$|\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle$$

piszemy:

$$|\psi_1\rangle \ldots |\psi_N\rangle \quad .$$

Podobnie operatory unitarne A i B operujące na przestrzeniach H_1 oraz H_2 możemy połączyć w jeden operator $A \otimes B : H_1 \otimes H_2 \rightarrow H_1 \otimes H_2$, który działa na stanach w następujący sposób:

$$A \otimes B(|\psi\rangle \otimes |\chi\rangle) = (A|\psi\rangle) \otimes (B|\chi\rangle) \quad .$$

W przypadku reprezentacji operatorów jako macierze:

$$A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \quad , \quad B = \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} \quad ,$$

iloczyn tensorowy macierzy, zwany czasami iloczynem Kroneckera, obliczany jest według wzoru:

$$A \otimes B = \begin{bmatrix} a_1 \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} & a_2 \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} \\ a_3 \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} & a_4 \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_2 b_1 & a_2 b_2 \\ a_1 b_3 & a_1 b_4 & a_2 b_3 & a_2 b_4 \\ a_3 b_1 & a_1 b_2 & a_4 b_1 & a_2 b_2 \\ a_3 b_3 & a_1 b_4 & a_4 b_3 & a_2 b_4 \end{bmatrix} \quad .$$

Natomiast dla wektorów:

$$\psi_1 = \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} \quad , \quad \psi_2 = \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} \quad , \quad \psi_3 = \begin{bmatrix} \alpha_3 \\ \beta_3 \end{bmatrix} \quad ,$$

iloczyn tensorowy obliczany jest według wzoru:

$$\psi_1 \otimes \psi_2 \otimes \psi_3 = \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} \otimes \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} \otimes \begin{bmatrix} \alpha_3 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} \alpha_1 \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} \\ \beta_1 \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} \end{bmatrix} \otimes \begin{bmatrix} \alpha_3 \\ \beta_3 \end{bmatrix} =$$

$$= \begin{bmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{bmatrix} \otimes \begin{bmatrix} \alpha_3 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} \alpha_1\alpha_2\alpha_3 \\ \alpha_1\alpha_2\beta_3 \\ \alpha_1\beta_2\alpha_3 \\ \alpha_1\beta_2\beta_3 \\ \beta_1\alpha_2\alpha_3 \\ \beta_1\alpha_2\beta_3 \\ \beta_1\beta_2\alpha_3 \\ \beta_1\beta_2\beta_3 \end{bmatrix} .$$

Jest to istotna różnica w porównaniu z iloczynem kartezjańskim używanym przy opisie rejestrów klasycznych. Własności iloczynu tensorowego implikują charakterystyczne cechy stanów kwantowych. Jednymi z najbardziej fundamentalnych jest twierdzenie o zakazie klonowania stanu kwantowego Wootersa i Żurka[26] oraz twierdzenie o nieusuwalności stanu kwantowego. To drugie twierdzenie możemy uważać za odwrócenie czasowe twierdzenia o zakazie klonowania.

W celu sformułowania pierwszego twierdzenia[27] wprowadzamy *unitarny operator kopiowania*, który w najprostszym przypadku układu złożonego z dwóch przestrzeni Hilberta H ma postać:

$$U: H \otimes H \rightarrow H \otimes H, U|\psi\rangle|\chi\rangle = |\psi\rangle|\psi\rangle \quad .$$

Twierdzenie o klonowaniu wówczas mówi:

Twierdzenie 1.
Dla $dim(H) > 1$ nie istnieje operator kopiujący.

Dowód (za pozycją [27]):
Załóżmy, że operator kopiujący U istnieje; wykorzystując dwa stany $|\psi_1\rangle$ i $|\psi_2\rangle$:

$$U(|\psi_1\rangle|\psi_1\rangle) = |\psi_1\rangle|\psi_1\rangle \quad ,$$

$$U(|\psi_2\rangle|\psi_1\rangle) = |\psi_2\rangle|\psi_2\rangle \quad ,$$

zatem:

$$U\left(\frac{1}{\sqrt{2}}(|\psi_1\rangle+|\psi_2\rangle)|\psi_1\rangle\right)=\left(\frac{1}{\sqrt{2}}(|\psi_1\rangle+|\psi_2\rangle)\right)\left(\frac{1}{\sqrt{2}}(|\psi_1\rangle+|\psi_2\rangle)\right)= \\ =\frac{1}{2}(|\psi_1\rangle|\psi_1\rangle+|\psi_1\rangle|\psi_2\rangle+|\psi_2\rangle|\psi_1\rangle+|\psi_2\rangle|\psi_2\rangle) \quad ,$$

$$U\left(\frac{1}{\sqrt{2}}(|\psi_1\rangle+|\psi_2\rangle)|\psi_1\rangle\right)=\frac{1}{\sqrt{2}}U(|\psi_1\rangle|\psi_1\rangle)+\frac{1}{\sqrt{2}}U(|\psi_2\rangle|\psi_2\rangle)= \\ =\frac{1}{\sqrt{2}}|\psi_1\rangle|\psi_1\rangle+\frac{1}{\sqrt{2}}|\psi_2\rangle|\psi_2\rangle \quad .$$

Jak widać otrzymane powyżej reprezentacje stanu po przekształceniach nie są tożsame, zatem nie istnieje operator kopiujący.

Twierdzenie o nieusuwalności z kolei ma postać:

Twierdzenie 2.

Jeżeli dla wektorów $v_1, v_2, \ldots, v_i \in H_n$ oraz wektora zerowego $0 \in H_n$ istnieje całkowicie dodatnie odwzorowanie:

$$|v_i\rangle\langle v_i| \otimes |v_i\rangle\langle v_i| \rightarrow |v_i\rangle\langle v_i| \otimes |0\rangle\langle 0| \quad ,$$

czyli odwzorowanie usuwające kopię wektora v_i , to kopię tę będzie można odtworzyć z otoczenia. Dowód powyższego twierdzenia można znaleźć w [27].

Te dwa fundamentalne twierdzenia wpływają na użyteczność stanów kwantowych -stan kwantowy jest unikalny i niemożliwy do skopiowania, a przez to możemy go użyć jako klucza kryptograficznego lub wykryć potencjalną próbę kopiowania informacji zakodowanej w takim stanie. Z drugiej strony stan kwantowy jest wrażliwy na wpływ otoczenia, a mimo to, podczas jego zniszczenia możemy stan odtworzyć ze stanu kwantowego otoczenia.

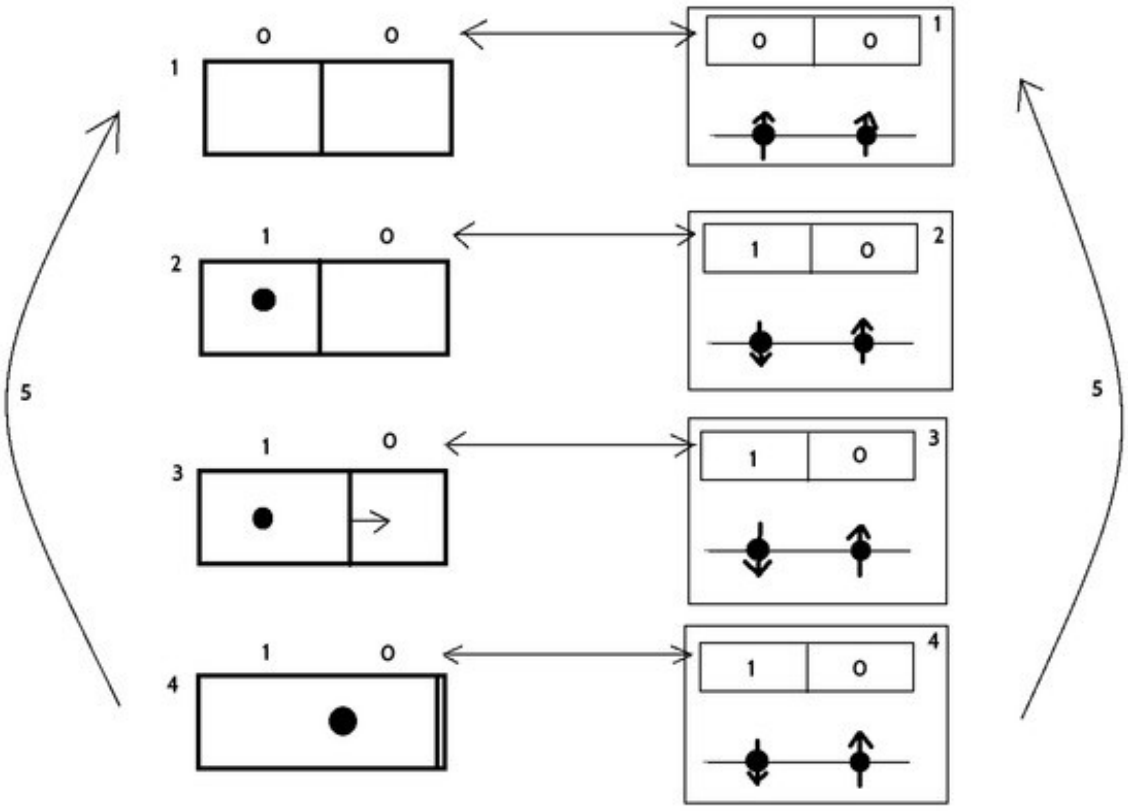
4.2. Odwracalność i reguła Landauera

W przypadku nieodwracalnych operacji binarnych Landauer[28] wykazał, że operacja taka, np. kasowanie bitów, generuje ciepło w wielkości nie mniejszej niż:

$$Q = k_B T \ln 2 \quad ,$$

gdzie:
k_B - stała Boltzmanna,
T – temperatura pracy układu.

To prawo zostało zweryfikowane w różnych układach (klasycznych i kwantowych)[29]. Prawo to również wyjaśnia pozorny paradoks demona Maxwella przedstawiony na rysunku poniżej.



(Rysunek z pracy [29])

Na rysunku przedstawiono po lewej stronie naczynie z jednym atomem gazu idealnego. Naczynie podzielone jest przegrodą kontrolowaną przez demona Maxwella. Stan pamięci demona wraz z elektroniczną implementacją tej pamięci przedstawione jest po prawej stronie. W stanie 1 demon nie wie, w której części naczynia znajduje się atom gazu. W sytuacji 2 demon zlokalizował gaz w lewej części naczynia – stan ten został zapamiętany w pamięci demona. W sytuacji 3 demon puszcza przegrodę i gaz może wykonać pracę swobodnie i adiabatycznie rozprężając się do pełnej objętości naczynia. Praca wykonana to:

$$W = k_B T \int_{V/2}^{V} \frac{dV}{V} = k_B T \ln 2 \quad .$$

Ostatnia część jest przedstawiona na w stanie 4. Jeżeli teraz tę sytuację zepniemy w cykl przy użyciu 5, gdzie demon wyjmuje przegrodę i w losowej chwili wkłada ją w środek naczynia, wówczas traci on informację na temat początkowego położenia atomu gazu. Wówczas informacja w pamięci demona jest w kolejnym cyklu bezużyteczna i należałoby ją skasować.

Paradoks termodynamiki demona Maxwella polega na tym, że taki cykl powtarzany wiele razy czerpie pracę wykorzystując jedynie źródło ciepła jakim jest atom gazu bez udziału chłodnicy. W oczywisty sposób przeczy to drugiej zasadzie termodynamiki sformułowanej dla silników cieplnych. Dlatego Landauer zaproponował, aby zachować drugą zasadę termodynamiki, że podczas operacji kasowania pamięci demona wydziela się ciepło $Q \geqslant W$ sprawiające, że entropia dla takiego procesu rośnie, a przez to zasady termodynamiki nie są łamane.

Wydzielane ciepło jest małe ze względu na wielkość stałej Boltzmanna, jednak w przyszłych układach elektronicznych i kwantowych może mieć istotne konsekwencje.

4.3. Kwantowy zapis danych

W przypadku komputera kwantowego ilość danych jakie możemy zapisać w rejestrze jest uzależniona od liczby kubitów, którymi dysponujemy. W przypadku N-kubitowego (N-kuditów) baza składa się z 2^N elementów, więc tworząc unormowane zespolone

kombinację liniowe jesteśmy w stanie zapisać nieprzeliczalną liczbę informacji: a liczb zespolonych.

W przypadku rejestru klasycznego w N bitach jesteśmy w stanie zapisać 2^N możliwych kombinacji informacji 2^N-1 bitów.

4.4. Bramki kwantowe

Bramka kwantowa pozwala w kontrolowany sposób wykonywać operacje na kubitach w celu zmiany ich stanu. W modelu obliczeń kwantowych opartych o bramki kwantowe jest ona podstawowym elementem umożliwiającym konstrukcję komputera kwantowego.

Bramki kwantowe są reprezentowane przez macierze unitarne, co wymagane jest przez formalizm mechaniki kwantowej (konieczność zachowania prawdopodobieństwa) [30]. Z racji, że operacje kwantowe opisywane są za pomocą operacji unitarnych, które są odwracalne, operacje kwantowe również są odwracalne, w odróżnieniu od operacji logicznych dla których ta własność nie zawsze zachodzi. Z uwagi na to, że przetworzenie rejestru przez bramkę jest tożsame z działaniem operatora unitarnego, reprezentującego daną bramkę, na wektor opisujący stan rejestru, liczba kubitów rejestru oraz liczba wejść/wyjść bramki musi być jednakowa.

Poniżej, w kolejnych podrozdziałach, opisane zostaną najpopularniejsze bramki kwantowe.

4.4.1. Bramka negacji

Jedną z najprostszych jednokubitowych bramek kwantowych – *bramkę negacji* można przedstawić w postaci następującej reprezentacji macierzowej:

$$NOT = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} .$$

Działa ona w taki sam sposób jak klasyczna bramka NOT, która zamienia stan sygnału wejściowego na przeciwny, na przykład dla wzbudzonego stanu bazowego $|1\rangle$ powoduje zmianę do stanu podstawowego $|0\rangle$:

$$NOT|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \quad .$$

Dla dowolnego kubitu $|\psi\rangle$ powoduje ona wzajemną zamianę amplitud prawdopodobieństwa na komplementarną, w sensie współczynniki przy wektorach bazowych zmieniają się według zasady $0 \rightarrow 1$ oraz $1 \rightarrow 0$:

$$NOT|\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \quad .$$

Bramki kwantowe można również zapisać także w notacji Diraca, na przykładzie powyższej bramki NOT :

$$NOT = |0\rangle\langle 1| + |1\rangle\langle 0| \quad .$$

Dla stanu bazowego $|1\rangle$:

$$NOT|1\rangle = |0\rangle\langle 1|1\rangle + |1\rangle\langle 0|1\rangle = |0\rangle \quad .$$

4.4.2. Zestaw uniwersalnych bramek kwantowych

Bardzo istotnym z punktu widzenia informatyki kwantowej jest istnienie i realizacja zestawu czterech uniwersalnych bramek kwantowych, niezbędnych do konstrukcji komputera kwantowego (kryterium DiVincenzo o istnieniu uniwersalnego zbioru bramek kwantowych, opisane w rozdziale 5.1).

Zgodnie z twierdzeniem o uniwersalności bramek kwantowych, każda kwantowa operacja unitarna może być przybliżona z dowolną dokładnością za pomocą kombinacji wielu bramek uniwersalnych oraz ich iloczynu tensorowego, dokładniej - każdą operację unitarną daje się przybliżać (przy użyciu mnożenia macierzy i iloczynu tensorowego) przez bramki H, T, CNOT oraz macierz jednostkową opisane poniżej[31].

1. Bramka identyczności

Pierwszą z zestawu uniwersalnych bramek kwantowych jest jednokubitowa bramka identyczności, reprezentowana przez poniższą dwuwymiarową macierz unitarną:

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

2. Bramka Hadamarda

Kolejną z uniwersalnych bramek kwantowych jest jednokubitowa bramka Hadamarda H reprezentowana przez iloczyn $\sqrt{2}^{-1}$ i dwu wymiarowej macierzy Hadamarda:

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

Działanie bramki na dwuwymiarową bazę $\{|0\rangle,|1\rangle\}$ powoduje transformację układu do stanu, w którym oba stany bazowe mogą być zaobserwowane z takim samym prawdopodobieństwem, wynoszącym $\frac{1}{2}$:

$$H|0\rangle=\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle \quad ,$$

$$H|1\rangle=\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle \quad .$$

Ponadto podwójne zadziałanie bramką Hadamarda powoduje powrót układu do stanu pierwotnego:

$$H^2|0\rangle=H\left(\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle\right)=|0\rangle \quad ,$$

$$H^2|1\rangle=H\left(\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle\right)=|1\rangle \quad .$$

Warto zauważyć, że operatory takie, które spełniają zależność $H^2=I$ mają wartości własne ± 1 .

3. Bramka T

Bramka T, odpowiadająca za przesunięcie w fazie, inaczej nazywana bramką $\frac{\pi}{8}$:

$$T=\begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{i\pi}{4}\right) \end{bmatrix} .$$

Poniżej przedstawiono działanie bramki na stany bazowe:

$$T|0\rangle=\begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{i\pi}{4}\right) \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix}=\begin{bmatrix} 1 \\ 0 \end{bmatrix}=|0\rangle ,$$

$$T|1\rangle=\begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{i\pi}{4}\right) \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}=\begin{bmatrix} 0 \\ \exp\left(\frac{i\pi}{4}\right) \end{bmatrix} .$$

4. Bramka kontrolowanej negacji

Ostatnią z zestawu uniwersalnych bramek kwantowych jest *dwukubitowa bramka kwantowa kontrolowanej negacji cNOT* . Można ją przedstawić za pomocą następującej reprezentacji macierzowej:

$$cNOT=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

w bazie iloczynu tensorowego: $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$.

Pierwszy z dwóch kubitów wejściowych nazywany jest kubitem sterującym i jego stan wpływa na drugi kubit – docelowy. Wtedy i tylko wtedy, gdy kubit sterujący jest w stanie bazowym $|1\rangle$ następuje odwrócenie drugiego kubitu, natomiast dla stanu $|0\rangle$ negacja nie następuje.

Formalnie bramkę cNOT można zapisać jako:

$$|x\, y\rangle = |x\,, x \oplus y\rangle \quad ,$$

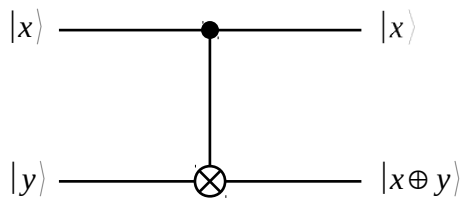
gdzie:

x - kubit sterujący (kontrolny),

y - kubit docelowy,

$\oplus$ - suma modulo 2.

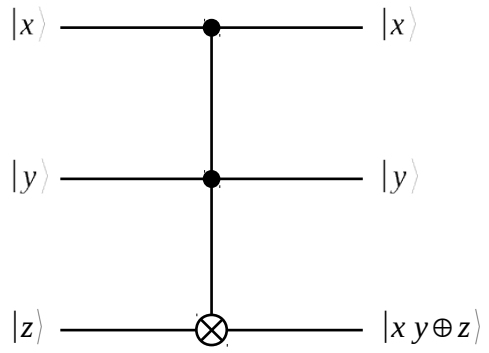
Graficznie bramkę kwantową kontrolowanej negacji można przedstawić następująco:



Rysunek 8. Reprezentacja graficzna bramki cNOT.

4.4.3. Bramka Toffoliego

Rozwinięciem bramki kwantowej kontrolowanej negacji jest *trzykubitowa bramka Toffoliego*, która posiada dodatkowo drugą linię sterującą. Nazywana jest ona kontrolowaną kontrolowaną bramką negacji (ccNOT).



Rysunek 9. Reprezentacja graficzna bramki Toffoliego.

Stan dwóch pierwszych kubitów steruje kubitem docelowym w następujący sposób: wtedy i tylko wtedy, gdy równocześnie oba kubity sterujące znajdują się w stanie bazowym $|1\rangle$ następuje odwrócenie stanu trzeciego kubitu, natomiast dla wszystkich innych kombinacji kubitów wejściowych negacja nie następuje.

Analogicznie do bramki *cNOT* , bramkę *ccNOT* można przedstawić za pomocą następującej reprezentacji macierzowej:

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

4.4.4. Bramka Pauliego X

W algorytmie Grovera będziemy również używać bramki Pauliego X, która działa dla jednego kubitu i wyraża się przez macierz Pauliego:

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

Bramka ta działając na wybrany stan zamienia go na przeciwny:

$$X|\psi\rangle=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}=\begin{bmatrix}\beta\\ \alpha\end{bmatrix} \quad .$$

Dla stanów bazowych $|0\rangle$, $|1\rangle$ powoduje wzajemną ich zamianę:

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

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

5. Komputery kwantowe

Narodziny i rozwój rozważań o komputerach kwantowych miały miejsce pod koniec XX wieku. Wynikały z dostrzeżenia ograniczeń komputerów klasycznych oraz nowych możliwości, które mogła zapewnić mechanika kwantowa.

Pojęcie po raz pierwszy pojawiło się w pracach/wykładzie fizyka-noblisty Richarda Feynmana[16].

5.1. Kryteria konstrukcji komputera kwantowego

Wymagania, które muszą spełniać systemy fizyczne do przechowywania i przetwarzania informacji kwantowych sformułował w 1995 roku amerykański fizyk teoretyczny David DiVincenzo[32]. Obecnie znane są one jako *kryteria DiVincenzo* i wymagają od komputera kwantowego następujących cech:

1. możliwość skalowania do systemów z wystarczającą liczbą dobrze zdefiniowanych kubitów,
2. możliwość przygotowania kubitów w stanie początkowym $|0\rangle$,
3. możliwość realizacji kubitów za pomocą stanów fizycznych ze stosunkowo długim średnim czasem życia w celu zapewnienia koherencji stanów kwantowych podczas obliczeń,
4. możliwość realizacji zestawu uniwersalnych bramek kwantowych: transformacji unitarnych na pojedynczych kubitach oraz sterowalnych bramek cNOT,
5. skuteczna procedura mierzenia stanu kubitu w końcowym etapie obliczeń (odczyt wyniku).
6. możliwość wewnętrznej konwersji kubitu stacjonarnego w kubit mobilny,
7. możliwość koherentnej transmisji kubitów pomiędzy lokalizacjami.

Łączne spełnienie co najmniej pierwszych pięciu kryteriów DiVincenzo jest konieczne do zbudowania maszyny zdolnej do realizacji algorytmów kwantowych[33].

W przypadku komunikacji kwantowej spełnione muszą zostać dodatkowo kolejne dwa kryteria.

5.2. Modele obliczeń kwantowych

Obecnie istnieje kilka różnych podejść do realizacji fizycznej kwantowej najprostszej jednostki informacji oraz konstrukcji komputera kwantowego. W tym celu wykorzystać można na przykład następujące zjawiska[28]: magnetyczny rezonans jądrowy, pułapki jonowe, kropki kwantowe, a także obwody nadprzewodzące na złączach Josephsona.

W pracy skupimy się na abstrakcyjnym komputerze kwantowym, a nie na jego konkretnej realizacji fizycznej.

Obecnie komputery kwantowe realizują obliczenia w oparciu o kilka różnych modeli[27]:

1. model obwodowy - obliczenia są realizowane przy użyciu bramek kwantowych; jest to podejście podobne do realizacji komputera kwantowego ,
2. model adiabatyczny - komputer jest realizowany zgodnie z twierdzeniem adiabatycznym; obliczenia są realizowane przy pomocy zjawiska kwantowego wyżarzania,
3. model topologiczny - opiera się na nietrywialnych topologicznych stanach materii,
4. kwantowa maszyna Turinga - model analogiczny do klasycznej maszyny Turinga,
5. kwantowa teoria automatów - model analogiczny do klasycznej teorii automatów skończonych.

Najpopularniejszymi obecnie modelami są: model obwodowy oraz równoważny do niego model adiabatyczny. Istniejące algorytmy kwantowe można łatwo wyrazić za pomocą modelu obwodowego – używając bramek kwantowych. Model ten jest najbardziej zbliżony do klasycznego modelu bramek logicznych w komputerze kwantowym. W dalszej części pracy skupimy się właśnie na tym modelu.

5.3. Komercyjne realizacje fizyczne

Pierwszy w historii, dostępny komercyjnie, komputer kwantowy został stworzony przez firmę D-Wave Systems[34]. Zrealizowany został w oparciu o model adiabatyczny obliczeń kwantowych, który jest równoważny modelowi obwodowemu[35]. W 2011 roku firma zaprezentowała 128 kubitowy komputer kwantowy D-Wave One[36]. Dwa lata później do sprzedaży wprowadzony został 512 kubitowy D-Wave Two[37]. Następnie, w 2015 roku firma przedstawiła kolejną ulepszoną wersję swojego produktu – 1000 kubitowy model 2X[38].

Kolejny model, zaprezentowany w 2017 roku to D-Wave 2000Q[39], dysponuje dwoma tysiącami kubitów. Rok później, za pośrednictwem dedykowanego portalu internetowego D-Wave Leap, firma udostępniła darmowy dostęp w czasie rzeczywistym do swojego komputera kwantowego[40].

Inną firmą obecną na rynku komercyjnych komputerów kwantowych jest IBM. W styczniu 2019 roku firma IBM zaprezentowała swój pierwszy komercyjny komputer kwantowy o nazwie IBM Q System One[41], zbudowany na bazie 20 kubitowego rejestru. Komputer jest oparty na zjawisku nadprzewodnictwa. Podobnie jak D-Wave Systems, IBM również udostępnia za darmo dostęp do komputera kwantowego w chmurze (ograniczenie do pięciu lub szesnastu kubitów, w zależności od wybranego komputera) o nazwie IBM Q Experience[42].

Produkcją oraz rozwojem obwodów kwantowych, używanych do konstrukcji komputerów kwantowych zajmuje się również firma Rigetti Computing[43].

Wyżej wymienione realizacje komputerów kwantowych znalazły odbiorców wśród firm takich jak Google, a także wielu instytucji naukowych i rządowych, między innymi: NASA, CERN oraz Los Alamos National Laboratory.

6. Algorytm kwantowy Grovera

Wyszukiwanie wyróżnionego elementu w zbiorze danych jest współcześnie jednym z częściej spotykanych problemów związanych z przetwarzaniem danych. W przypadku, gdy nie dysponujemy dodatkowymi informacjami na temat zbioru, dla nieuporządkowanej bazy danych, składającej się z N rekordów, algorytmy klasyczne polegają na sprawdzeniu całej bazy danych kolejno element po elemencie i potrzebują średnio $\frac{N}{2}$ prób na znalezienie żądanego elementu. Jeśli poszukiwany rekord będzie ostatnim elementem kolekcji, konieczne będzie wykonanie aż N operacji w celu poznania wyniku. We wspomnianej notacji O, która nie jest czuła na czynniki liczbowe, taki algorytm ma złożoność $O(N)$.

Szybsze rozwiązanie problemu przeszukiwania nieposortowanego zbioru danych podaje informatyka kwantowa. Jeden z historycznie pierwszych algorytmów kwantowych - algorytm Grovera może zostać użyty do wyszukiwania określonego elementu w nieuporządkowanej bazie danych, potrzebując na to tylko średnio $O(\sqrt{N})$ prób.

Wyobraźmy sobie funkcję $f:Q\rightarrow\{0,1\}$ określoną następująco:

$$f(x)=\begin{cases}1, & x=a\\ 0, & x\neq a\end{cases},$$

gdzie:

a - element poszukiwany, taki że:

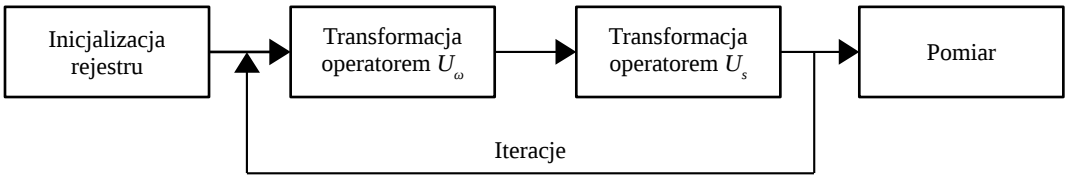
$$\exists!_{a\in Q} f(a)=1 \quad .$$

Taką funkcję nazywamy wyrocznią (ang. *oracle*), gdyż wyróżnia (zwracając wartość 1) dokładnie jeden, poszukiwany element a.

Algorytm Grovera umożliwia znalezienie argumentu, dla którego funkcja $f(x)$ przyjmuje wartość 1. W rozpatrywanym przypadku zbiorem argumentów Q jest baza danych, a funkcja realizuje operację porównania z elementem wyróżnionym a.

Algorytm, jako dane wejściowe, przyjmuje rejestr kwantowy złożony z n kubitów, którego $N=2^n-1$ stanów bazowych odpowiada za reprezentację zbioru danych który chcemy przeszukać. Następnie iteracyjnie wzmacniana jest amplituda prawdopodobieństwa poszukiwanego rozwiązania, a na końcu wykonywany jest pomiar.

Poszczególne kroki algorytmu Grovera zostały przedstawione na poniższym diagramie. Ich szczegółowy opis zawarty został w kolejnej sekcji.



Rysunek 10: Kroki algorytmu Grovera.

Jak zobaczymy niżej, istnieje optymalna liczba iteracji dająca najwyższą amplitudę dla szukanego elementu. Jeżeli wykonamy więcej iteracji ponad optymalną liczbę, wówczas algorytm może nie znaleźć rozwiązania.

6.1. Kroki algorytmu

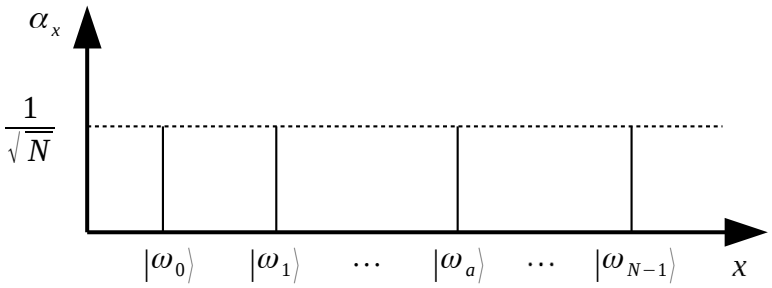
Kroki algorytmu Grovera składają się z następującej sekwencji:

1. Inicjacja rejestru kwantowego reprezentującego bazę danych. Najczęściej używa się binarnej reprezentacji danych z bazy kodując je w bazie iloczynu tensorowego kubitów. Jeżeli baza danych zawiera dane reprezentowane przez N bitów klasycznych, to takie dane możemy zapisać przy pomocy systemu złożonego z N kubitów o bazie |0...0> aż do |1...1>.
2. Wprowadzenie każdego ze stanów wejściowych ω , za pomocą transformaty Hadamarda:

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

w superpozycję stanów bazowych:

$$|s_0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |\omega_i\rangle \quad .$$



Ten krok odpowiada za wyrównanie wartości amplitud α wszystkich stanów bazowych ω , co przedstawione zostało na powyższym schematycznym wykresie.

3. Transformacja operatorami U_ω i U_s w kolejnych iteracjach:

$$|s_{n+1}\rangle = U_s U_\omega |s_n\rangle \quad ,$$

gdzie:

$$U_\omega = I - 2|\omega_a\rangle\langle\omega_a| \quad ,$$

$$U_s = 2|s\rangle\langle s| - I \quad .$$

Operator U_ω , nazywany zmodyfikowanym operatorem sprawdzenia, powoduje zmianę znaku amplitudy prawdopodobieństwa poszukiwanego stanu $|\omega_a\rangle$:

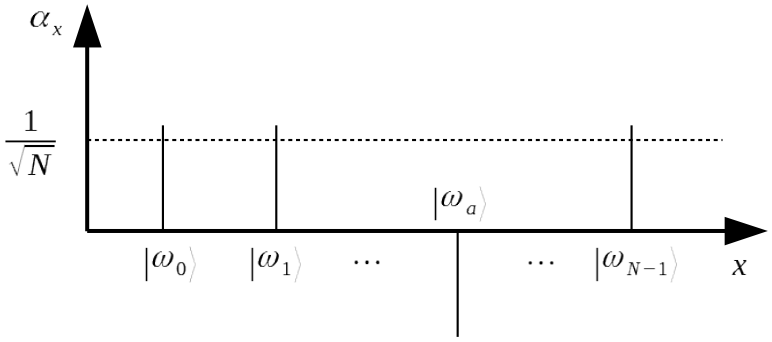
$$U_\omega = I - 2|\omega_a\rangle\langle\omega_a| \quad ,$$

która efektywnie działa w następujący sposób:

$$U_\omega|\omega\rangle=(-1)^{f(\omega)}|\omega\rangle \quad ,$$

gdzie $f(x)$, jest wspomnianą ‘wyrocznią’ zwracającą 1 dla x będącego szukanym elementem i 0 w przeciwnym wypadku. Operator U_ω również w literaturze nazywany jest wyrocznią.

Operacja ta zmienia znak amplitudy odpowiadającej szukanemu elementowi ω_a . Jest to schematycznie przedstawione na poniższym rysunku.



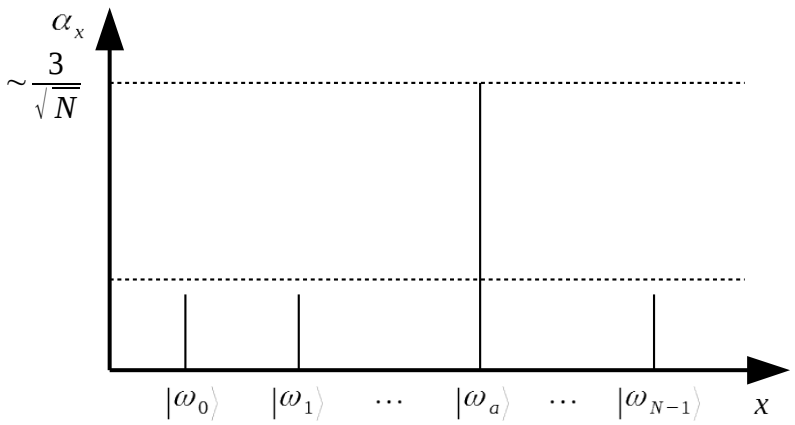
Fakt ten wykorzystany zostaje w kolejnym kroku iteracji – transformacji operatorem dyfuzji Grovera U_s . Operator ten ma reprezentację macierzową:

$$D=2A-I\otimes I\otimes\ldots\otimes I=2A-I^{\otimes N} \quad ,$$

gdzie:

$$A=\begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{1}{N} & & & \frac{1}{N} \\ \vdots & & & \vdots \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix} \quad .$$

Operator dyfuzji Grovera odpowiada za obrót wszystkich amplitud prawdopodobieństwa wokół ich średniej wartości. Nazwa pochodzi od tego, że Grover inspirował się dyskretną wersją operatora dla równania dyfuzji. Operator ten ma również szereg zastosowań w innych algorytmach kwantowych[23].



Operator ten wykorzystuje fakt, iż szukana amplituda ma przeciwną fazę uzyskaną w poprzednim kroku. To sprawia, że operator dyfuzji powiększa amplitudę dla szukanego elementu zmniejszając amplitudy dla innych elementów bazy, co przedstawia powyższy rysunek.

Dla lepszego zrozumienia działania tego operatora rozważmy przykład dla N=2. W celu uproszczenia przykład rozpatrzmy w zbiorze liczb rzeczywistych. Załóżmy, że mamy wektor $[0.25, -0.25, 0.25, 0.25]$ w bazie: |00>, |01>, |10>, |11>. Amplitudy są równe, poza faktem, że amplituda dla |01> ma odwróconą fazę. Jeżeli teraz zastosujemy operator dyfuzji Grovera otrzymamy wektor $[0.0, 1.0, 0.0, 0.0]$ (dla N=2 jedna iteracja jest optymalna). Jeśli jednak wykonamy dodatkową iterację otrzymamy wektor falowy $[-0.5, 0.5, -0.5, -0.5]$ więc otrzymujemy równe prawdopodobieństwa dla każdej amplitudy wynoszące $[0.25, 0.25, 0.25, 0.25]$. Zatem wykonując zbyt wiele iteracji nie będziemy w stanie odnaleźć poszukiwanego elementu. Jest to charakterystyczna cecha algorytmu Grovera, a także innych algorytmów kwantowych.

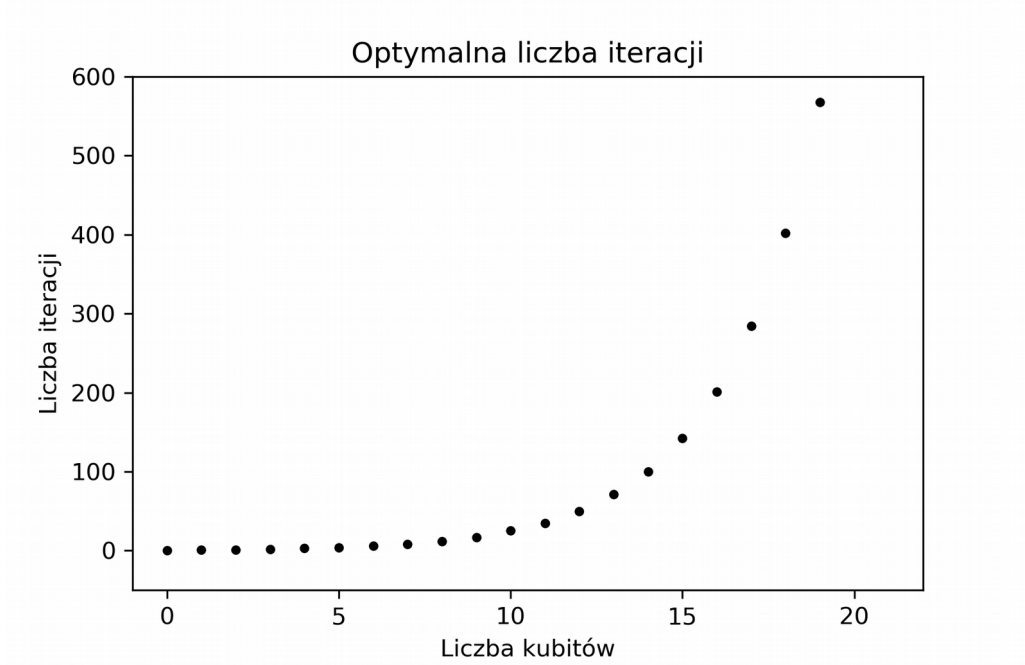
Dla ogólnego przypadku ponowna transformacja powyższymi operatorami (iteracja) powoduje wzmocnienie amplitudy prawdopodobieństwa poszukiwanego stanu $|\omega_a\rangle$ o około $\frac{2}{\sqrt{N}}$.

Optymalna liczba iteracji i , których wykonanie jest potrzebne do uzyskania poprawnego rezultatu z najwyższym możliwym prawdopodobieństwem, dana jest wyrażeniem[2]:

$$i \leqslant \frac{\pi \sqrt{2^n}}{4} \quad ,$$

gdzie n jest liczbą kubitów.

Wykonanie zbyt dużej liczby iteracji, jak widzieliśmy powyżej powoduje zmniejszenie amplitudy poszukiwanego stanu, a zatem wpływa negatywnie na otrzymane rozwiązanie. Poniżej przedstawiona została zależność optymalnej liczby iteracji od liczby kubitów.



Jak widać na powyższym wykresie mamy do czynienia ze wzrostem wykładniczym - nawet dla stosunkowo niedużej liczby kubitów musimy wykonać kilkadziesiąt lub nawet kilkaset iteracji, w celu osiągnięcia możliwie największego prawdopodobieństwa poprawności rezultatu.

4. Pomiar rejestru, po wykonaniu dostatecznej liczby iteracji, w celu znalezienia poszukiwanego stanu $|\omega_a\rangle$. W przybliżeniu, stan wynikowy jest z pewnym prawdopodobieństwem równy stanowi poszukiwanemu:

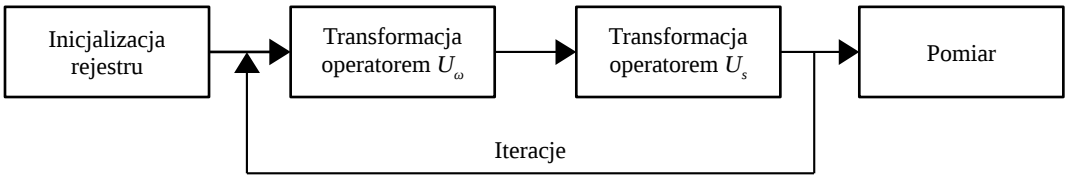
$$|s_i\rangle = U_s U_\omega |s_{i-1}\rangle ; |\langle s_i | \omega_a \rangle| \gg |\langle s_i | \omega_l \rangle|, \text{ dla } l \neq a \quad .$$

W kolejnym rozdziale przedstawiona zostanie praktyczna implementacja poszczególnych kroków algorytmu Grovera.

II. Część praktyczna

7. Implementacja algorytmu

Kroki algorytmu Grovera zostały szczegółowo opisane w rozdziale 6.1. niniejszej pracy. Kolejność koniecznych do wykonania operacji przed stawiona jest na poniższym diagramie.



Rysunek 11: Kroki algorytmu Grovera.

W dalszej części przedstawione zostaną główne fragmenty implementacji algorytmu Grovera. Na początku zostanie pokazane jak krok po kroku zaimplementować najprostszą wersję algorytmu przy użyciu obliczeń macierzowych z biblioteki NumPy, a następnie skupimy się na konkretnym przykładzie mającym już bardziej realistyczne zastosowanie.

7.1. Implementacja w Pythonie

Na początku opiszemy w jaki sposób przy użyciu języka Python 3.6 oraz biblioteki NumPy wykonać proste obliczenia kwantowe[44][45]. Biblioteka NumPy służy do szybkich obliczeń numerycznych i macierzowych w Pythonie[46]. Do wizualizacji wyników wykorzystana została dodatkowo biblioteka matplotlib[47].

Wybierzmy n=2 kubitowy system z bazą 00,01,10,11, która ma wymiar $N=2^n=4$. Kod programu przedstawiony został poniżej. Komentarze w języku Python rozpoczynamy symbolem # i ciągnął się one do końca linii. W poniższym kodzie opisują one konkretne kroki w kodzie.

Na początku importujemy potrzebne biblioteki oraz definiujemy parametry wejściowe:

```
import math
import numpy as np  #biblioteka NumPy jest dostępna pod nazwą np
```

#Wymiar bazy 00,01,10,11

N=4

#Wektor bazowy (0,0,1,0) poszukiwanego elementu

omega=np.array([0.0,0.0,1.0,0.0], dtype=complex)

#Konstrukcja wyroczni dla stanu 10 (np.eye zwraca macierz jednostkową)

oracle=np.eye(N)

oracle[2,2]=-1

print("U_omega=", oracle)

U_omega= [[1. 0. 0. 0.]

[0. 1. 0. 0.]

[0. 0. -1. 0.]

[0. 0. 0. 1.]]

#Wektor początkowy dla algorytmu Grovera (1,0,0,0)

psi=np.array([1.0, 0.0,0.0, 0.0], dtype=complex)

print("psi0=", psi)

psi0= [1.+0.j 0.+0.j 0.+0.j 0.+0.j]

#Konstrukcja bramki Hadamarda dla jednego kubitu – H2, a następnie wykorzystanie iloczynu tensorowego (np.kron) do stworzenia bramki Hadamarda dla 2 kubitów

H2=1.0 / 20.5 * np.array([[1.0, 1.0],[1.0, -1.0]])**

H4=np.kron(H2,H2)

print("H=", H4)

H= [[0.5 0.5 0.5 0.5]

[0.5 -0.5 0.5 -0.5]

[0.5 0.5 -0.5 -0.5]

[0.5 -0.5 -0.5 0.5]]

#Konstrukcja operatora dyfuzji Grovera – np.ones tworzy macierz złożoną z samych jedynek

D=2*(1.0/float(N))*np.ones((N,N), dtype=complex)-np.eye(N)

print("D=", D)

D= [[-0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]

[0.5+0.j -0.5+0.j 0.5+0.j 0.5+0.j]

[0.5+0.j 0.5+0.j -0.5+0.j 0.5+0.j]

[0.5+0.j 0.5+0.j 0.5+0.j -0.5+0.j]]

##Początek algorytmu – jedna iteracja

#Krok 1 - Aplikacja bramki Hadamarda na stanie początkowym

psi=H4.dot(psi)

print("H.psi=", psi)

H.psi= [0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]

#Krok 2 – aplikacja wyroczni

psi=oracle.dot(psi)

print("oracle.psi=", psi)

oracle.psi= [0.5+0.j 0.5+0.j -0.5+0.j 0.5+0.j]

#Krok 3 – aplikacja operatora dyfuzji Grovera

psi=D.dot(psi)

print("D.psi=", psi)

D.psi= [0.+0.j 0.+0.j 1.+0.j 0.+0.j]

#Pomiar stanu psi w stanie omega

print("Pomiar w stanie omega=", math.sqrt(omega.dot(psi)))

P*omiar w stanie omega= 0.9999999999999999*

Jako wynik otrzymamy 1. Zatem w tym prostym przypadku algorytm Grovera po jednej iteracji wskazał szukany element.

Gdybyśmy ponownie wykonali iterację:

psi=oracle.dot(psi)

psi=D.dot(psi)

print("D.psi=", psi)

D.psi= [-0.5+0.j -0.5+0.j 0.5+0.j -0.5+0.j]

Wówczas wektor falowy to [-0.5+0.0j, -0.5+0.0j, 0.5+0.0j, -0.5+0.0j], (j=sqrt(-1)), więc otrzymujemy równe prawdopodobieństwa dla każdej amplitudy, a algorytm nie daje wskazania. Jest to przykład ogólnego zjawiska dla tego algorytmu – występuje optymalna liczba iteracji po przekroczeniu której algorytm może nie dać wyniku.

Powyższy przykład pokazuje, jak implementować proste bramki i wykonywać obliczenia kwantowe przy użyciu języka Python i biblioteki NumPy. Następnie zajmiemy się bardziej realistycznym przykładem, w którym dane wejściowe do przeszukiwania znajdują się w książce telefonicznej, wygenerowanej na potrzeby niniejszego przykładu.

W dalszej części przedstawione zostaną główne fragmenty implementacji algorytmu Grovera dla $n=3$ kubitowego rejestru. Z racji, że wszystkie wektory i operatory są rzeczywiste, dla uproszczenia będziemy operować wyłączenia na liczbach rzeczywistych.

Pełny kod programu, umożliwiający zmianę parametrów wejściowych, w tym m.in. rozmiaru rejestru kwantowego, został załączony w Appendix.

Dla niniejszego przykładu możemy za jego pomocą w systemie binarnym zakodować 8 różnych wartości z przedziału $N=\langle 0,2^3-1\rangle=\langle 0,7\rangle$, przedstawionych tabeli poniżej.

N	Stan rejestru
0	$\lvert 000\rangle$
1	$\lvert 001\rangle$
2	$\lvert 010\rangle$
3	$\lvert 011\rangle$
4	$\lvert 100\rangle$
5	$\lvert 101\rangle$
6	$\lvert 110\rangle$
7	$\lvert 111\rangle$

Niech zbiór danych wejściowych będzie nieposortowaną tabelą zawierającą unikalne numery telefonów, dodatkowo posiadającą pole z indeksem wiersza.

Indeks	Numer telefonu
0	70707070
1	20202020
2	40404040
3	10101010
4	30303030
5	80808080
6	60606060
7	50505050

Niech poszukiwanym elementem będzie numer $a = 60606060$. Numery telefonów zapisane są w *wektorze phoneBook*, do elementów którego można odwołać się poprzez podanie wartości indeksu. Podobnie jak poprzednio na początku importujemy bibliotekę NumPy, a następnie definiujemy zbiór danych oraz wartość poszukiwanego elementu.

```
#Import bibliotek
import math
import numpy as np

#Zbiór danych
phoneBook=np.asarray([70707070, 20202020, 40404040, 10101010, 30303030,
80808080, 60606060, 50505050])

#Wybór poszukiwanego elementu
a=60606060
```

Zdefiniujmy również funkcję obliczającą optymalną liczbę iteracji oraz zestaw zmiennych wejściowych.

```
#Funkcja obliczająca optymalną liczbę iteracji na podstawie rozmiaru (size) rejestru kwantowego
def iterations(size):
    return math.floor(0.25*math.pi*math.sqrt(2**size))

#Rejestr kwantowy – parametry wejściowe
n=3
N=2**n
iMax=iterations(n)
print("Optymalna liczba iteracji:", iMax)
Optymalna liczba iteracji: 2

#Macierz identyczności
identityMatrix=np.eye(N)
print(identityMatrix)
[[1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]]
```

Pierwszym krokiem algorytmu jest inicjalizacja rejestru – uśrednienie wartości amplitud wszystkich stanów (jest ona równoważna aplikacji bramki Hadamarda **H8=np.kron(H4,H2)** z poprzedniego przykładu na stan |000>=(1,0,0,0,0,0,0,0)):

```
#Stworzenie zainicjalizowanego rejestru
s=[]
```

s.append((1.0/math.sqrt(N))*np.ones((N,1)))

print(s[0])

[[0.35355339]
[0.35355339]
[0.35355339]
[0.35355339]
[0.35355339]
[0.35355339]
[0.35355339]
[0.35355339]]

Następnie konstruujemy macierz odpowiedzialną za operację sprawdzenia, bazując na stworzonej uprzednio macierzy jednostkowej *identityMatrix*:

U_omega=np.copy(identityMatrix)

for i in range (N):
    if(phoneBook[i]==a):
        U_omega[i,i]=-1.0

print(U_omega)

[[1. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 0. -1. 0.]
[0. 0. 0. 0. 0. 0. 0. 1.]]

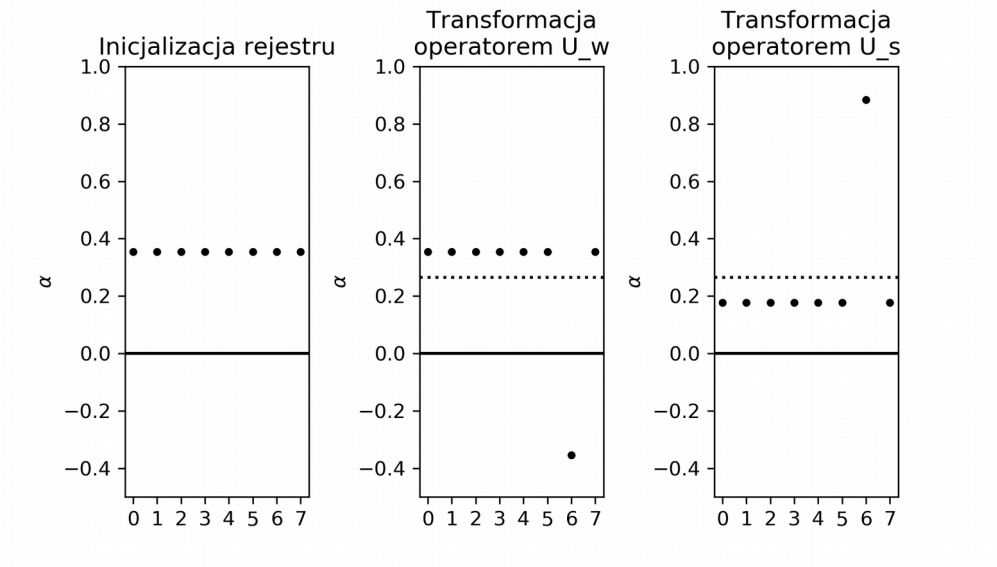
W kolejnym kroku tworzymy macierz dyfuzji Grovera, odpowiedzialną za obrót amplitud wokół ich wartości średniej. Do obliczenia iloczynu zewnętrznego dwóch wektorów użyjemy funkcji *outer* z biblioteki NumPy:

```
U_s=2*np.outer(s[0],s[0])-identityMatrix
print(U_s)
[[-0.75  0.25  0.25  0.25  0.25  0.25  0.25  0.25]
 [ 0.25 -0.75  0.25  0.25  0.25  0.25  0.25  0.25]
 [ 0.25  0.25 -0.75  0.25  0.25  0.25  0.25  0.25]
 [ 0.25  0.25  0.25 -0.75  0.25  0.25  0.25  0.25]
 [ 0.25  0.25  0.25  0.25 -0.75  0.25  0.25  0.25]
 [ 0.25  0.25  0.25  0.25  0.25 -0.75  0.25  0.25]
 [ 0.25  0.25  0.25  0.25  0.25  0.25 -0.75  0.25]
 [ 0.25  0.25  0.25  0.25  0.25  0.25  0.25 -0.75]]
```

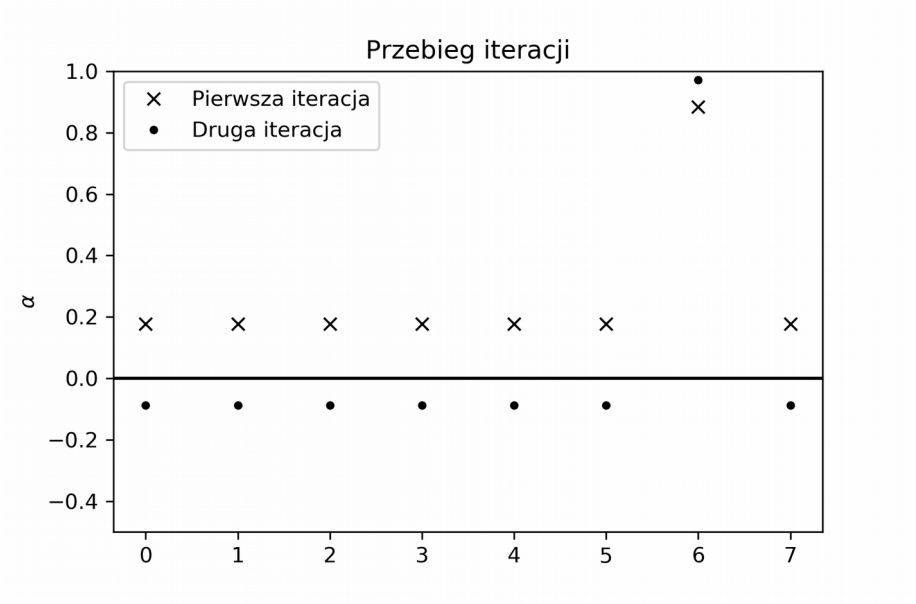
Następnie w pętli wykonujemy odpowiednią ilość operacji transformacji operatorami *U_omega* oraz *U_s* korzystając z funkcji *dot* z biblioteki NumPy, służącej do obliczania iloczynu skalarnego. Dla 3 kubitowego rejestru optymalna liczba iteracji *iMax* wynosi 2. Wynik każdej iteracji dołączamy do listy *s*.

```
#Pętla główna
for i in range(iMax):
    s.append(U_s.dot(U_omega.dot(s[i])))
```

Przebieg inicjalizacji rejestru oraz pierwszej iteracji przedstawiony jest na wykresach poniżej. Kod służący do obliczenia wartości widocznych na wykresie oraz ich wizualizacji załączony jest w Appendix.



Dla niewielkiej liczby kubitów (w tym przypadku równej 3) już po wykonaniu jednej iteracji z dużym prawdopodobieństwem sukcesu można wskazać wynik. Dla danego rozmiaru rejestru kwantowego optymalna liczba iteracji wynosi 2. Wynik działania kolejnych iteracji widoczny jest na poniższym wykresie.



Ostatnim krokiem algorytmu jest pomiar:

#Krok 3 - pomiar

res=s[iMax]*s[iMax]

#Wybór maksymalnej wartości z listy res

max=np.where(res==np.amax(res))[0]

print(res)

[[0.0078125]

[0.0078125]

[0.0078125]

[0.0078125]

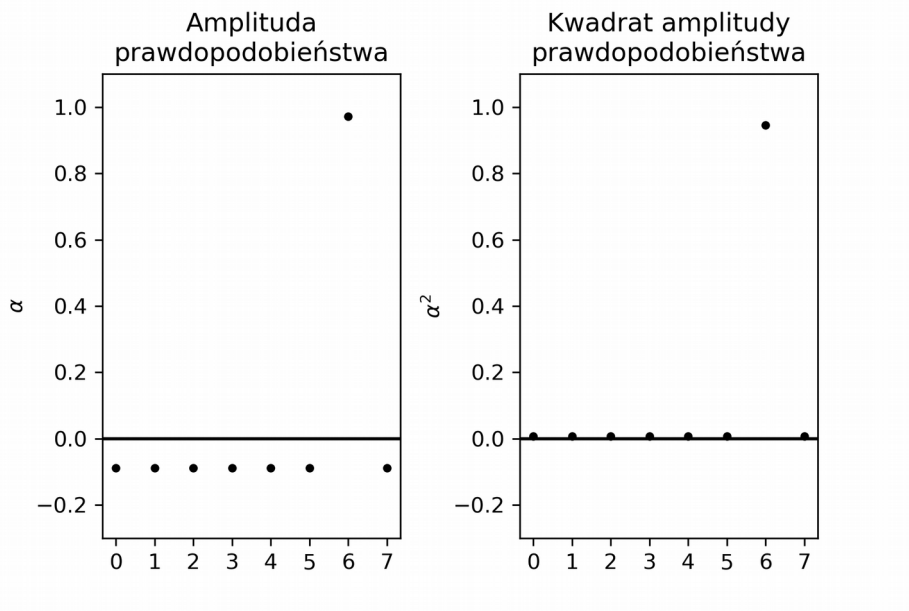
[0.0078125]

[0.0078125]

[0.9453125]

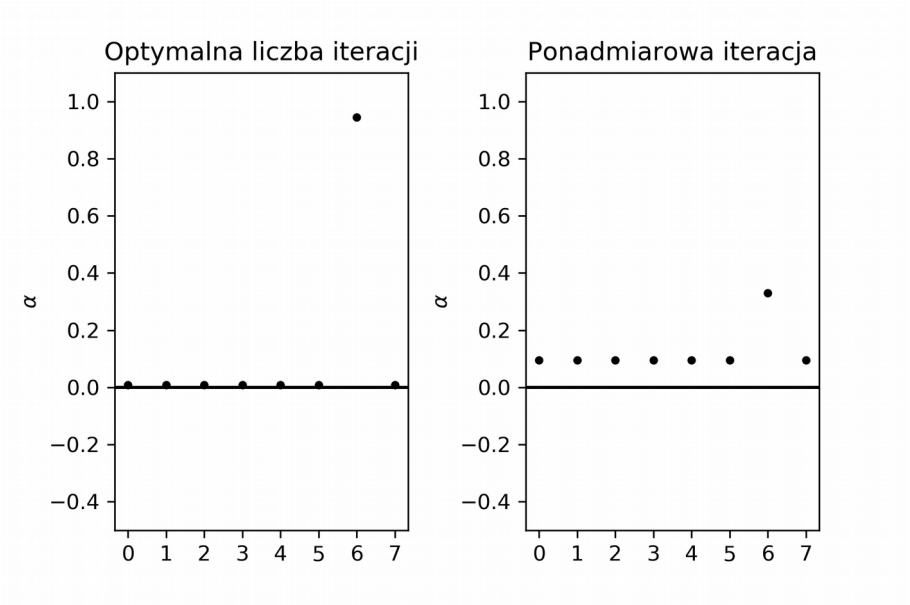
[0.0078125]]

Wynik działania algorytmu przedstawiony został poniżej.



Jak widać na powyższym wykresie, po wykonaniu dwóch iteracji udało się znaleźć poszukiwany element z prawdopodobieństwem wynoszącym około 0,95.

Ponadto wykonanie większej ilości iteracji niż optymalna ich liczba powoduje zmniejszenie amplitudy poszukiwanego stanu, a zatem wpływa negatywnie na dokładność otrzymanego wyniku.



7.2. Obwód kwantowy IBM Q

Dostęp do komputera kwantowego IBM Q, realizującego obliczenia w oparciu o model obwodowy, możemy uzyskać za pomocą strony IBM Q Experience[42]. Umożliwia ona tworzenie obwodów kwantowych na kilka różnych sposobów.

Pierwszym sposobem tworzenia obwodów kwantowych jest interfejs graficzny, za pomocą którego można tworzyć obwody korzystając z dostępnych bramek kwantowych oraz operacji. Sposób tworzenia algorytmów kwantowych w tym interfejsie przypomina komponowanie muzyki używając zapisu podobnego do znanego z muzyki zapisu nutowego, tzw. ‘pięciolini’, dlatego też ten interfejs nazywa się *Composer* (kompozytor).

Drugim sposobem jest otwarta platforma programistyczna (ang. framework) Qiskit do obliczeń kwantowych, umożliwiająca komunikację z IBM Q między innymi z poziomu języka Python.

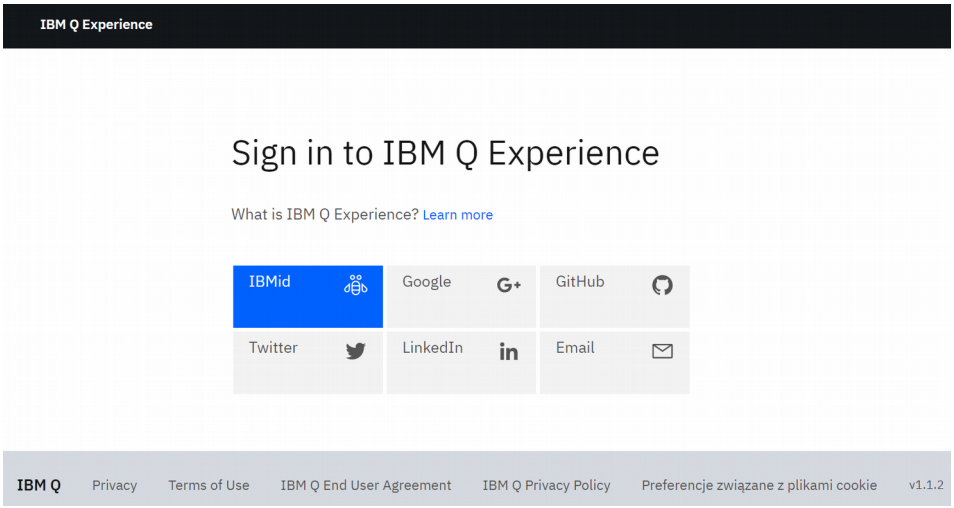
Obwody stworzone za pomocą obu powyższych metod mogą zostać wyrażone za pomocą kodu ‘kwantowego asemblera’ nazywanego OpenQASM, służącego do reprezentacji obwodów kwantowych IBM Q, który jest zarazem trzecią możliwą

metodą implementacji obwodów kwantowych. Jest to najbardziej niskopoziomowa metoda.

W celu implementacji algorytmu Grovera na komputerze kwantowym konieczne jest stworzenie obwodu kwantowego, który realizuje rutynę opisaną w rozdziale 6. W dalszej części przedstawiony zostanie przykład implementacji dla komputera kwantowego IBM Q.

7.2.1. Opis interfejsu

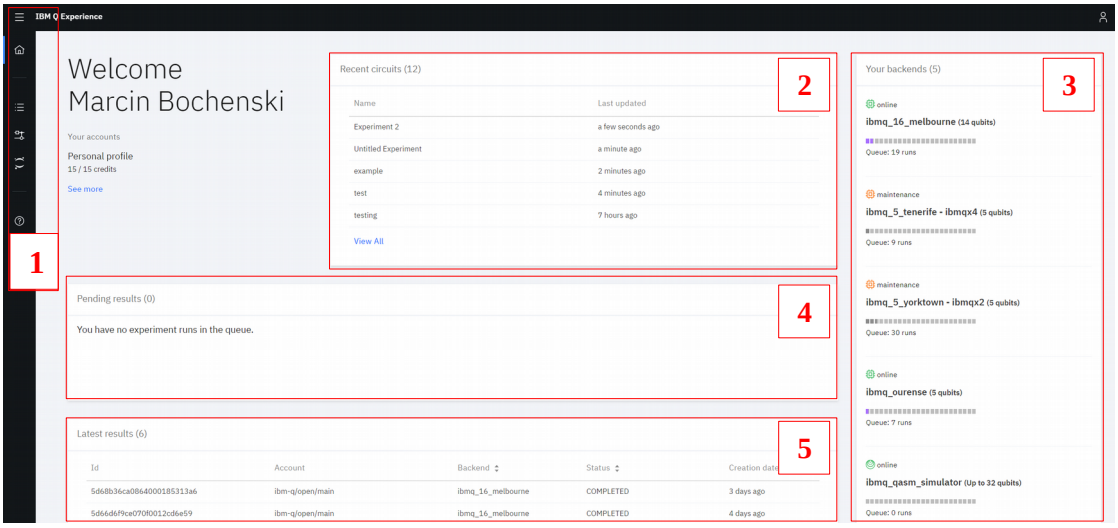
Interfejs graficzny IBM Q Experience[42] umożliwia tworzenie obwodów kwantowych z elementów dostępnych w zdefiniowanym zbiorze – bramek kwantowych oraz operacji. W publicznie dostępnej darmowej wersji można użyć między innymi pięcio- lub szesnasto-kubitowego procesora kwantowego. Dostęp do komputera kwantowego IBM jest możliwy przez stronę internetową https://quantum-computing.ibm.com. Ekran logowania przedstawiony jest poniżej; w celu autentykacji można użyć dedykowanego konta *IBMid* lub konta jednego z dostępnych portali.



Rysunek 12: Strona logowania do portalu IBM Q Experience.

Po zalogowaniu następuje automatyczne przekierowanie do panelu głównego IBM Q (ang. *Dashboard*). Panel boczny (1) umożliwia nawigację pomiędzy stroną główną, wynikami obliczeń (ang. *Results*), edytorem obwodów (ang. *Circuit Composer*), notatnikami Qiskit (ang. *Qiskit Notebooks*), a także stroną z dokumentacją i wsparciem

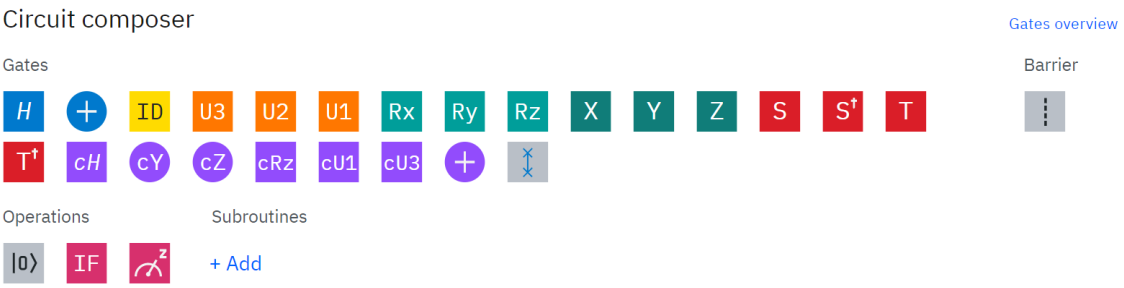
(ang. *Documentation & Support*). Za pośrednictwem strony głównej mamy dostęp do ostatnio używanych obwodów (2), listy komputerów kwantowych oraz ich aktualnego statusu (3), listy oczekujących obliczeń (4) oraz listy wykonanych obliczeń (5).



Rysunek 13: Strona główna IBM Q.

W edytorze możemy tworzyć obwód równocześnie na dwa sposoby: w trybie graficznym poprzez przeciągnie wybranych elementów oraz poprzez edycję kodu OpenQASM. Obie metody są równoważne i użycie jednej z nich automatycznie generuje tożsamy kod w języku drugiej.

Do konstrukcji obwodu kwantowego, jako elementów składowych, możemy użyć elementów widocznych poniżej.



Rysunek 14: Elementy dostępne w edytorze graficzny.

Poza bramkami kwantowymi mamy możliwość: wstawienia bariery, przywrócenia kubitu do stanu początkowego $|0\rangle$, użycia instrukcji warunkowej oraz wykonania

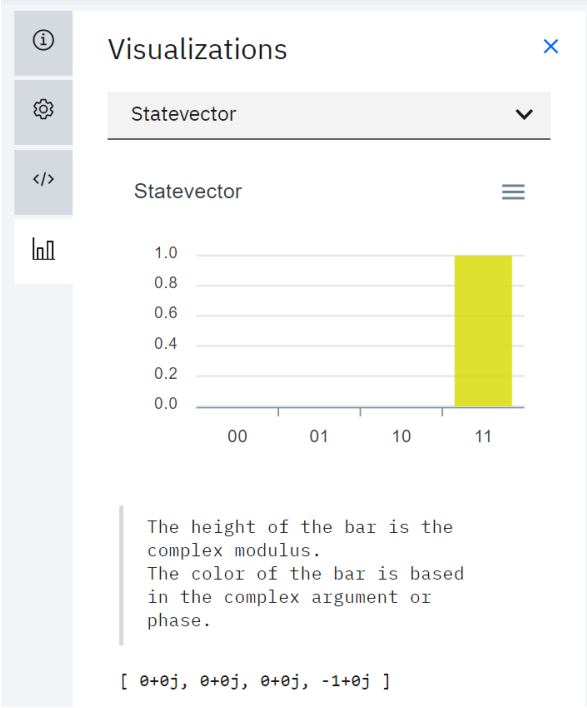
pomiaru. Dostępna jest również opcja stworzenia podprogramu, jednak na chwilę obecną funkcjonalność ta jest w fazie rozwoju i dostępna jest jedynie z poziomu edytora kodu OpenQASM.

7.2.2. Wysłanie na komputer kwantowy

IBM Q umożliwia obliczenie wyniku działania stworzonego obwodu zarówno w oparciu o symulację komputera kwantowego, jak również o jego wykonanie na procesorze kwantowym. W przypadku symulacji wynik zwracany jest natychmiastowo, natomiast w przypadku wykonania na procesorze kwantowym czas otrzymania rezultatu zależy od ilości obliczeń innych użytkowników, które oczekują w kolejce na wykonanie.

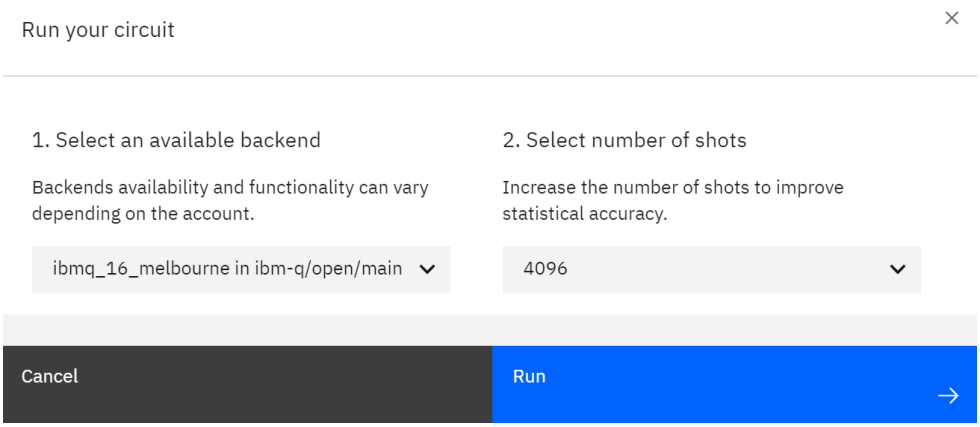
Dobrą praktyką jest testowanie algorytmu na symulatorze, a w przypadku gdy będziemy pewni, że nasz algorytm działa użycie komputera kwantowego. Dzięki temu wykorzystamy efektywnie zasoby.

W celu wyświetlenia wyniku symulacji należy otworzyć zakładkę *Visualizations* po lewej stronie interfejsu (zob, rysunek 15). Umożliwia ona wybór jednego z trzech predefiniowanych typów wykresów: *Statevector*, *Density Matrix* i *Measurement Probabilities*. Poza wykresami podana jest również numeryczna wartość wektora stanu.



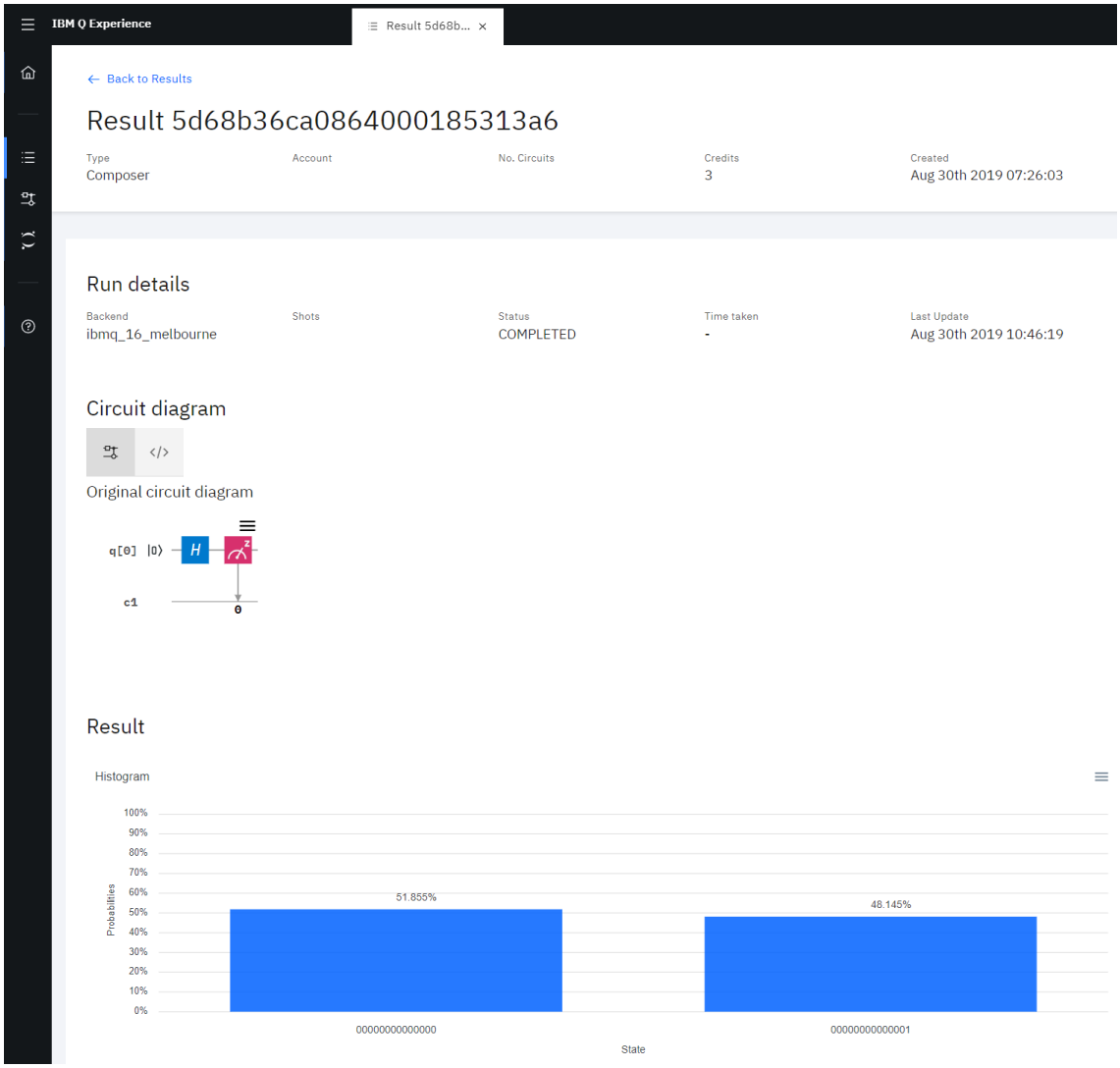
Rysunek 15: Wizualizacja wyniku symulacji.

Aby uruchomić stworzony obwód na procesorze kwantowym należy użyć opcji *Run your circuit*. Umożliwia ona wybór procesora kwantowego spośród dostępnych na liście oraz liczby prób spośród wartości: (1, 1024, 4096, 8192).



Po zatwierdzeniu zlecenie wykonania obliczeń zostanie przesłane do kolejki oczekującej do wybranego procesora. Wynik, gdy będzie już dostępny, zostanie dodany do zakładki *Results* → *Completed*. Wpis wynikowy zawiera schemat przesłanego do

obliczeń obwodu oraz histogram przedstawiający prawdopodobieństwo odczytu danego stanu.



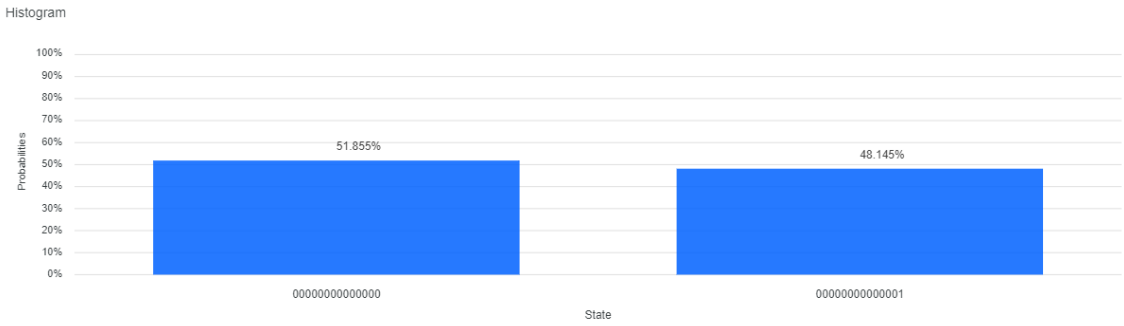
Rysunek 16: Karta z wynikiem obliczeń.

Jako bardzo prosty przykład algorytmu kwantowego i wstęp do użycia interfejsu IBM Q, wykonamy operację bramki Hadamarda na stanie kwantowym $|0\rangle$, a następnie wykonamy pomiar.

Dla danego stanu bramka Hadamarda transformuje układu do stanu:

$$H|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \quad ,$$

w którym stany $|0\rangle$ i $|1\rangle$ mogą być zaobserwowane w kilkunastu próbach (1024) z równym prawdopodobieństwem.



Rysunek 17: Wynik działania algorytmu na 16 kubitowym procesorze dla 1024 prób.

Zatem stan końcowy będzie miał równe prawdopodobieństwo (wynoszące 0,5) znalezienia się w stanie $|0\rangle$ oraz $|1\rangle$, co pokazuje wykres.

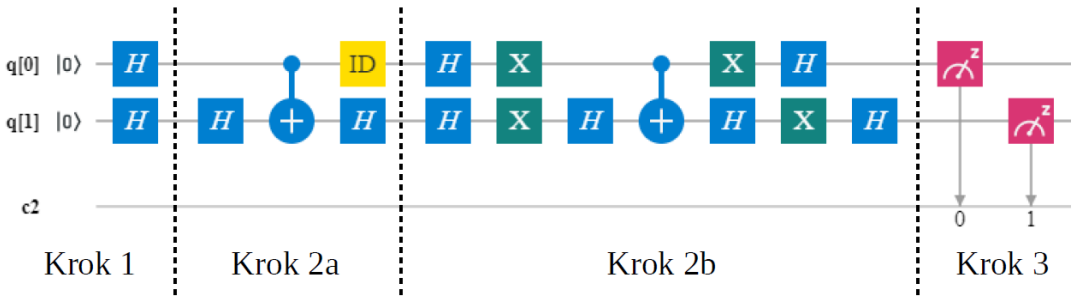
7.2.3. Opis implementacji

Poniżej przedstawiona jest implementacja algorytmu Grovera dla dwóch kubitów, gdzie poszukiwanym elementem jest $a=|11\rangle$. Obwód został stworzony za pomocą następujących bramek:

Bramka	Funkcja OpenQASM	Symbol graficzny
Hadamarda	h	
Pauliego X	x	
kontrolowanej negacji	cx	
identyczności	id	

Szczegółowy opis bramek użytych do implementacji algorytmu znajduje się w rozdziale 4.4. Bramka Pauliego X realizuje bramkę negacji opisaną wcześniej.

Bramka identyczności (ID) została użyta wyłącznie na potrzeby zoptymalizowania sfery wizualnej omawianego obwodu, w celu łatwiejszego odseparowania od siebie poszczególnych kroków algorytmu. Związane jest to z tym, że interfejs Composera automatycznie wyrównuje elementy do lewej strony.



Rysunek 18: Obwód kwantowy z wyszczególnionymi krokami algorytmu Grovera.

Wygenerowany kod OpenQASM (dostępny w zakładce *Circuit editor* oznaczonej symbolem </>) powyższego obwodu realizującego algorytm Grovera, a także komentarze i wyniki symulacji poszczególnych etapów, zamieszczone zostały poniżej.

Stworzenie dwukubitowego rejestu

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
```

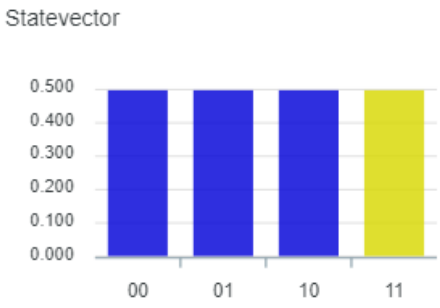
Krok 1 – inicjalizacja rejestru:

```
h q[0];
h q[1];
```

Krok 2 - jedna iteracja

a) aplikacja wyroczni - zmiana znaku amplitudy poszukiwanego elementu a :

```
h q[1];
cx q[0],q[1];
id q[0];
h q[1];
```



[0.5+0j, 0.5+0j, 0.5+0j, -0.5+0j]

b) transformacja operatorem dyfuzji Grovera:

```
h q[0];
x q[0];
```

h q[1];

x q[1];

h q[1];

cx q[0],q[1];

x q[0];

h q[1];

h q[0];

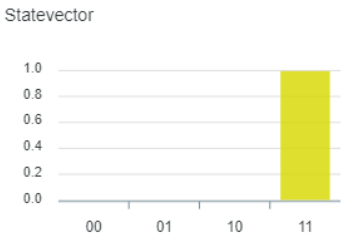
x q[1];

h q[1];

Krok 3 – pomiar:

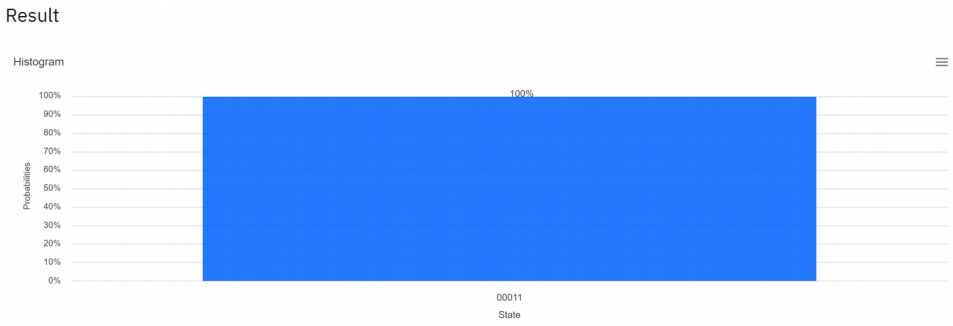
measure q[0] -> c[0];

measure q[1] -> c[1];



[0+0j, 0+0j, 0+0j, -1+0j]

Jak widać za pomocą przedstawionej implementacji udało się znaleźć poszukiwany element.

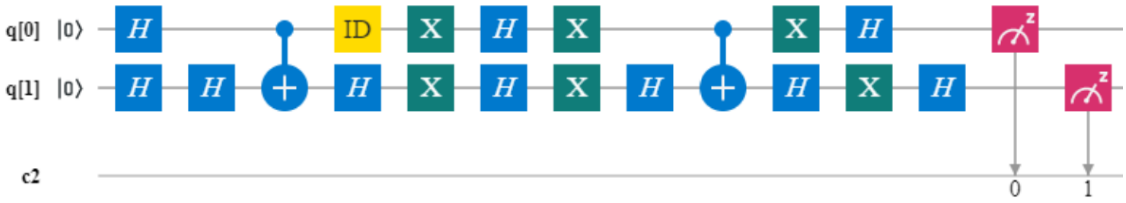


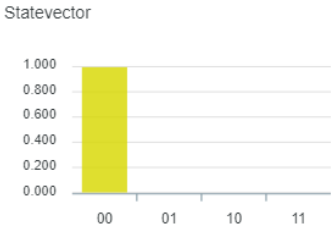
Ilustracja 1: Wynik wykonanych obliczeń kwantowych.

Dla rozpatrywanego przypadku wynik wykonanych obliczeń jest zgodny z wynikiem symulacji.

Poniżej przedstawione są również implementacje algorytmu dla poszukiwanego elementu a równego: $|00\rangle, |01\rangle, |10\rangle$ wraz z wynikami symulacji. Różnią się one między sobą implementacją wyroczni (krok 2a). Implementacja pozostałych kroków algorytmu jest uniwersalna dla wszystkich przedstawionych wartości a . Podobnie jak bramka identyczności, dwie następujące bezpośrednio po sobie bramki H dla nie zmieniają działania obwodu (wzajemnie się znoszą); zostały dodane dla wyraźnego wydzielenia kolejnych kroków algorytmu.

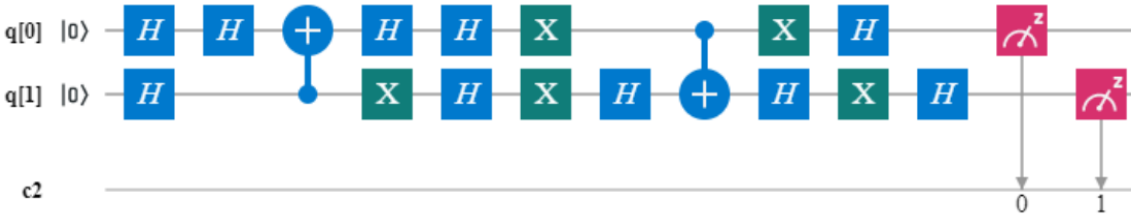
$a = |00\rangle$

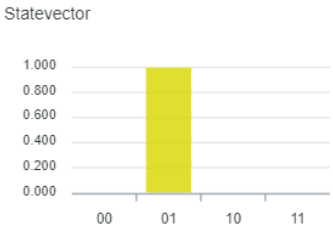




[-1+0j, 0+0j, 0+0j, 0+0j]

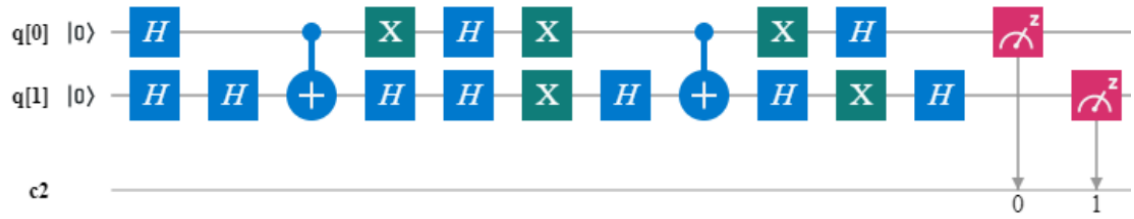
$a = |01\rangle$

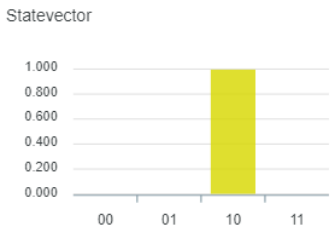




[0+0j, -1+0j, 0+0j, 0+0j]

$a = |10\rangle$





[0+0j, 0+0j, -1+0j, 0+0j]

Znane są również inne implementacje algorytmu Grovera dla komputera kwantowego IBM Q, w tym także dla większej liczby kubitów[48][49].

7.3. Wnioski

Implementacja prostego dwukubitowego algorytmu Grovera na rzeczywistym komputerze kwantowym IBM Q nie przedstawiała żadnych trudności. Graficzny interfejs sprawia, że wizualne programowanie jest proste i intuicyjne. Wyniki są również prezentowane w zrozumiały sposób.

Ważnym spostrzeżeniem jest to, że algorytm kwantowy daje wynik z pewnym prawdopodobieństwem, co jest zupełnie innym podejściem jak dla klasycznych komputerów.

Wszystkie algorytmy kwantowe działają szybko ze względu na niską liczbę kubitów używaną przez nie. Dlatego raczej nie są one użyteczne w zastosowaniach. Należy więc je uważać jedynie jako proste przykłady ilustrujące ideę.

Bardziej skomplikowane implementacje algorytmu Grovera o większej liczbie kubitów są opisane na przykład w pracach[48][49].

8. Podsumowanie

W niniejszej pracy opisano teoretyczne aspekty podstaw informatyki klasycznej oraz kwantowej, a także nakreślono podstawowe różnice między nimi. Przedstawione zostały kryteria dotyczące konstrukcji komputerów kwantowych oraz obecnie znane podejścia (modele obliczeń kwantowych) do ich realizacji. Wyjaśniona została istota oraz zasada działania algorytmu kwantowego Grovera. Opisane zostały również wybrane zagadnienia związane z współczesnymi systemami bazodanowymi.

W części praktycznej przedstawiono implementację algorytmu kwantowego Grovera w języku Python na komputerze klasycznym, w oparciu o symulację numeryczną bramek kwantowych. Umożliwiła ona skuteczne i efektywne przeszukanie niewielkiego zbioru danych. Przedstawiona została również przykładowa implementacja algorytmu Grovera dla komputera kwantowego IBM Q.

Możliwymi rozwinięciami niniejszej pracy mogą być: implementacja obwodu kwantowego dla większej ilości kubitów oraz komunikacja z bazą danych, w celu wierniejszego odwzorowania działania algorytmu. Ponadto istnieje możliwość rozszerzenia zakresu działania algorytmu Grovera do poszukiwania kilku elementów równocześnie[50].

Bibliografia

1: Cormen T. H., Leiserson C. E., Rivest R. L., Clifford S., PWN, Wprowadzenie do algorytmów., 2017

2: Grover L. K., Proceedings, 28th Annual ACM Symposium on the Theory of Computing, A fast quantum mechanical algorithm for database search., 1996

3: Pierce J. R., Dover Publications, An Introduction to Information Theory: Symbols, Signals & Noise., 1980

4: Baker J. R., Wiley-IEEE, CMOS: Circuit Design, Layout, and Simulation, Third Edition., 2010

5: Werewka J., Skrzyński P., Turek M., Wydawnictwo AGH, Podstawy programowania komputerów., 2008

6: , , IEC International Standard IEC 60027-2: Letter symbols to be used in electrical technology - Part 2: Telecommunications and electronics,

7: Kuratowski K., Mostowski A., Monografie Matematyczne, tom 27. Instytut Matematyczny PAN, Teoria mnogości., 1952

8: Rasiowa H., PWN, Wstęp do matematyki współczesnej, 2014

9: von Neumann J., von Neumann J., First Draft of a Report on the EDVAC., 1945

10: Stallings W., WN-T, Organizacja i architektura systemu komputerowego., 2000

11: Wróblewski P., Helion, Algorytmy - struktury danych i techniki programowania., 2003

12: Cormen T. H., Helion, Algorytmy bez tajemnic., 2013

13: Beynon-Davies P., WN-T, Systemy baz danych., 1998

14: Ullman J. D., Widom J., WN-T, Podstawowy wykład z systemów baz danych., 2000

15: , , ISO/IEC 9075-2:2016: Information technology -- Database languages -- SQL -- Part 2: Foundation (SQL/Foundation), 2016

16: Elmasri R., Navathe S.B., Helion, Wprowadzenie do systemów baz danych., 2005

17: Feynman R. P., International Journal of Theoretical Physics. 21 (6/7), Simulating Physics with Computers., 1982

18: Nielsen M. A., Chuang I. L., Cambridge University Press, Quantum Computation and Quantum Information., 2010

19: Hirvensalo M., WSiP, Algorytmy kwantowe., 2004

20: Shor P. W., Proceedings 35th Annual Symposium on Foundations of Computer Science. IEEE, Algorithms for quantum computation: discrete logarithms and factoring., 1994

21: Kitaev A. Y., Russian Mathematical Surveys, 52:6, Quantum computations: Algorithms and error correction., 1997

22: Mosca M., Rozprawa doktorska. Uniwersytet Oksfordzki, Quantumu Computer Algorithms., 1999

23: Oficjalna strona NIST, , math.nist.gov/quantum/zoo/, data dostępu: 2019.07.20

24: Shankar R., PWN, Mechanika kwantowa., 2006

25: Rudin W., McGraw-Hill, Real and Complex Analysis., 1996

26: Wootters W. K., Żurek W. H., Nature 299, A single quantum cannot be cloned., 1982

27: Sawerwain M., Wiśniewska J., PWN, Informatyka kwantowa. Wybrane obwody i algorytmy., 2015

28: Le Bellac M., PWN, Wstęp do informatyki kwantowej., 2012

29: Kycia R. A., Entropy 2018, 20(12), Landauer's Principle as a Special Case of Galois Connection, 2018

30: Berman G. P. et al., World Scientific, Introduction to Quantum Computers., 1999

31: Miesięcznik Delta., Uniwersytet Warszawski, http://www.deltami.edu.pl/delta/archiwum/2017/12/, 2017

32: DiVincenzo D. P., Science, 270, Quantum Computation, 1995

33: Gruber J. et al., Zeszyty naukowe Politechniki Śląskiej. Seria: Organizacja i Zarządzanie, zeszyt 74 , Kryteria budowy komputera kwantowego i algorytmy kryptografii postkwantowej., 2014

34: Oficjalna strona D-Wave Systems, https://www.dwavesys.com/home, , data dostępu: 2019.06.15

35: Aharonov D. et al., SIAM Review, 50(4), Adiabatic Quantum Computation Is Equivalent to StandardQuantum Computation., 2008

36: Nature, 474, 18 (2011), https://www.nature.com/news/2011/110531/full/474018a.html, data dostępu: 2019.08.04

37: Oficjalna strona D-Wave Systems, , http://www.dwavesys.com/sites/default/files/D-Wave-Investor%20Presentation-Web100814-2.pdf, data dostępu: 2019.08.04

38: Oficjalna strona D-Wave Systems, D-Wave 2X, https://www.dwavesys.com/sites/default/files/D-Wave%202X%20Tech%20Collateral_0915F.pdf, data dostępu: 2019.08.04

39: Oficjalna strona D-Wave Systems, , https://www.dwavesys.com/press-releases/d-wave%C2%A0announces%C2%A0d-wave-2000q-quantum-computer-and-first-system-order, data dostępu: 2019.08.04

40: Oficjalna strona D-Wave Systems, , https://www.dwavesys.com/press-releases/d-wave-launches-leap-first-real-time-quantum-application-environment, 2019.08.04

41: Oficjalna strona IBM, https://www.research.ibm.com/ibm-q/, , data dostępu: 2019.06.15

42: Oficjalna strona IBM Q Experience, , https://quantum-computing.ibm.com, data dostępu: 2019.08.04

43: Oficjalna strona Rigetti Computing, , https://www.rigetti.com/, data dostępu: 2019.06.24

44: Downey A., Helion, Myśl w języku Python! Nauka programowania., 2017

45: Boschetti A., Massaron L., Helion, Python. Podstawy nauki o danych., 2017

46: Oficjalna strona NumPy, , https://www.numpy.org, data dostępu: 2019.08.24

47: Oficjalna strona Matplotlib, , https://matplotlib.org, data dostępu: 2019.08.24

48: Blomkvist Karlsson V., Strömberg P., http://www.diva-portal.org/smash/get/diva2:1214481/FULLTEXT01.pdf, 4-qubit Grover's algorithm implemented for the ibmqx5 architecture, data dostępu: 2019.08.25

49: Mandviwalla A., Ohshiro K., Ji B., https://cis.temple.edu/~boji/papers/REU2018.pdf, Implementing Grover’s Algorithm on the IBM Quantum Computers, data dostępu: 2019.08.26

50: Zalka Ch., https://arxiv.org/abs/quant-ph/9902049, A Grover-based quantum search of optimal order for an unknown number of marked elements, 1999

Appendix

W załączniku został zamieszczony kod w języku Python, wraz z komentarzami, symulujący działanie algorytmu Grovera oraz umożliwiający wizualizację otrzymanych wyników.

```
#Import bibliotek
import math
import random
import numpy as np
np.set_printoptions(threshold=np.inf)

#Funkcja obliczająca optymalną liczbę iteracji na podstawie rozmiaru (size) zestawu danych
def iterations(size):
   return math.floor(0.25*math.pi*math.sqrt(2**size))

#Rejestr kwantowy – parametry wejściowe
n=3
N=2**n
iMax=iterations(n)

#Macierz identyczności
identityMatrix=np.eye(N)

#Zbiór danych
#Generacja książki telefonicznej
#phoneBook=(1+np.arange(N))*np.full(N,10101010)
#np.random.shuffle(phoneBook)

#Zapisanie jednej z wygenerowanych kombinacji
phoneBook=np.asarray([70707070, 20202020, 40404040, 10101010, 30303030, 80808080, 60606060, 50505050])

#Wybór poszukiwanego elementu
a=60606060
```

```
#Wyświetlenie zmiennych
print("[INFO]","qRegister dim:", n, ", qRegister size:", N, ", iMax:", iMax, ",
distinguishedElement:", a)

#Krok 1 – inicjalizacja rejestru
s=[]
s.append((1.0/math.sqrt(N))*np.ones((N,1)))

#Krok 2
#Konstrukcja wyroczni
U_omega=np.copy(identityMatrix)


for i in range (N):
   if(phoneBook[i]==a):
      U_omega[i,i]=-1.0

U_s=2*np.outer(s[0],s[0])-identityMatrix

#Pętla główna
for i in range(iMax):
   s.append(U_s.dot(U_omega.dot(s[i])))

#Krok 3 - pomiar
res=s[iMax]*s[iMax]

#Wybór maksymalnej wartości z listy res
max=np.where(res==np.amax(res))[0]


print("[INFO]","Result id:", max, "res:", res[max][0])

#Sprawdzenie otrzymanego rozwiązania
#Sprawdzenie zgodności wyniku
if(phoneBook[max]!=a):
   print("[ERROR] - SEARCH FAILED")

#Sprawdzenie warunku normalizacji
if(abs(1-res.sum())>0.00000001):
   print("[ERROR] - NORMALIZATION FAULT")
```

```
#Tworzenie wykresów
#Pierwsza iteracja
import matplotlib.pyplot as plt
%matplotlib inline

ylim=(-0.5,1.0)

plt.figure(figsize=(7,4))
plt.subplot(131)
plt.title("Inicjalizacja rejestru")
plt.ylim(ylim)
plt.ylabel(r'$ \alpha$')
plt.plot(s[0], marker='.', linestyle='none', color='black')
plt.axhline(y=0, color='black', linestyle='-')
plt.xticks(range(N))

plt.subplot(132)
plt.title("Transformacja\noperatorem U_w")
plt.ylim(ylim)
plt.ylabel(r'$ \alpha$')
plt.plot(U_omega.dot(s[0]), marker='.', linestyle='none', color='black')
plt.axhline(y=0, color='black', linestyle='-')
plt.axhline(y=np.mean(U_omega.dot(s[0])), marker='', linestyle=':', color='black')
plt.xticks(range(N))

plt.subplot(133)
plt.title("Transformacja\noperatorem U_s")
plt.ylim(ylim)
plt.ylabel(r'$ \alpha$')
plt.plot(U_s.dot(U_omega.dot(s[0])), marker='.', linestyle='none', color='black')
plt.axhline(y=0, color='black', linestyle='-')
plt.axhline(y=np.mean(U_omega.dot(s[0])), marker='', linestyle=':', color='black')
plt.xticks(range(N))

plt.subplots_adjust(wspace=0.6)
plt.rcParams['figure.dpi']=100
plt.savefig('charts/first_iteration.png', dpi = 300)
plt.show()
```

```
#Przebieg iteracji
plt.title("Przebieg iteracji")
plt.ylim(ylim)
plt.ylabel(r'$ \alpha$')
plt.plot(s[1], marker='x', linestyle='none', color='black')
plt.plot(s[2], marker='.', linestyle='none', color='black')
plt.axhline(y=0, color='black', linestyle='-')
plt.xticks(range(N))
plt.gca().legend(('Pierwsza iteracja','Druga iteracja'))

plt.rcParams['figure.dpi']=100
plt.savefig('charts/second_iteration.png', dpi = 300)
plt.show()

#Optymalna liczba iteracji
plt.title("Optymalna liczba iteracji")
plt.xlabel("Liczba kubitów")
plt.ylabel("Liczba iteracji")
plt.plot(np.transpose(np.array([list(map(iterations,range(0,20)))])), marker='.', linestyle='none',
color='black')
plt.xlim(-1,22)
plt.ylim(-50,600)
plt.rcParams['figure.dpi']=80
plt.savefig('charts/total_iterations.png', dpi = 300)
plt.show()
```