КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет радіофізики,електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

ВИРІШЕННЯ ТРАНСПОРТНОЇ ЗАДАЧІ ЗА ДОПОМОГОЮ КВАНТОВОГОКОМП'ЮТЕРА

Дипломна робота бакалавра
студента 4року навчання
Спеціальність: 123 «Комп'ютерна інженерія»
Турченко Євгенія Олександровича

Науковий керівник:
к. ф.-м. н., доцент кафедри МТРФ
Іваненко Дмитро Олександрович

Рецензент:
д. ф.-м. н., професор кафедри алгебри механіко-математичного факультету
Шевченко Георгій Михайлович

До захисту допускаю

Завідувач кафедрою

Протокол засідання кафедри від
«___»_____2022 р. №____

КИЇВ - 2022

Зміст

0. Вступ

Вважається, що в недалекому майбутньому квантові комп’ютери здійснять так звану «квантову революцію» і тоді настане їх квантова перевага над звичайними комп’ютерами.

Квантовий комп’ютер (КК) – це пристрій, що, користуючись принципами квантової механіки, дозволяє скористатися явищем квантового паралелізму, завдяки якому, деякі операції, для яких класичному комп’ютеру потрібно сотні років, квантовий комп’ютер, в теорії, може вирішити за секунди. Квантові обчислення досліджуються ще з 80-х років минулого століття, але досі не вдалося створити повноцінний КК: вони досі не змогли досягти очікуваної продуктивності та потребують для своєї роботи умов, що важко досягти.

Тим не менш, навіть зараз вони перевершують класичні комп’ютери в деяких класах задач. Одним із таких класів є задачі перебору, пошуку. Для цих цілей вже існують спеціальні квантові алгоритми такі як пошук Гровера, квантовий відпал та інші.

У цій роботі буде розроблено алгоритм, для вирішення однієї з класичних задач пошуку шляху – задачі комівояжера. Крім того, буде проведене порівняння різних способів вирішення задачі, та порівняння квантового та класичного комп’ютера.

Мета роботи: Вирішення задачі комівояжера за допомогою квантового комп’ютера

1. Квантові комп’ютери

Квантовий комп’ютер – обчислювальний пристрій, робота якого базується на принципах квантової механіки: на принципах квантової суперпозиції і заплутаності. Від транзисторного комп'ютера відрізняється тим, що оперує кубітами – частинками, що можуть перебувати в трьох станах: «0», «1» та невизначений.

В теорії, квантові комп’ютери здатні розв’язувати певні задачі значно швидше ніж класичні. Наприклад, це задачі факториації цілих чисел або моделювання квантової системи багатьох тіл, квантовий алгоритм Шора, Саймона.

Одна з найсерйозніших проблем у галузі квантових обчислень — крихкість кубітів. Заплутаність системи кубітів з її середовищем, включаючи налаштування вимірювань, може порушити узгодженість системи та призвести до декогеренції. Тому зараз розробляються удосконалення у процесах створення обладнання для квантових обчислень та методи виправлення помилок.

2. Кубіти

Кубіт – це квантова система, яка має два рівні. В якості кубіта можуть бути використані, наприклад, диполі, атоми або фотони. Стан кубіту відповідає орієнтація поля, спіну чи поляризація. В класичних комп’ютерах біти можуть приймати лише один з двох станів, в свою чергу, завдяки своїм квантовомеханічним властивостям, кубіт може також перебувати і в суперпозиції цих станів. Хвильова функція кубіта є лінійною комбінацією його станів $|0\rangle$ і $|1\rangle$:

$$\psi = a|0\rangle + b|1\rangle,$$

,де a і b - комплексні числа, які задовольняють умові нормування

$$|a|^2 + |b|^2 = 1.$$

Під час вимірювання стану кубіта, його хвильова функція колапсує, і він займає стан $|0\rangle$ чи $|1\rangle$, як і для звичайного біта. При цьому ймовірність, що кубіт буде в стані $|0\rangle$ рівна $|a|^2$, а стан $|1\rangle$ — $|b|^2$. Завдяки своїм властивостям, виконуючи дії над системою кубітів в стані суперпозиції, можна змінити вірогідність отримання набору $|0\rangle$ і $|1\rangle$ таким чином, що прочитавши значення кубіт можна з високою імовірністю отримати, наприклад, значення виразу чи оптимальний набір кроків.

Стани кубіта можна зобразити з використанням сфери блоха. Для звичайного біта на ній доступні лише дві позиції – вгорі (стан 0) або внизу (стан 1). Для чистих станів кубіта доступна будь-яка точка на поверхні сфери.

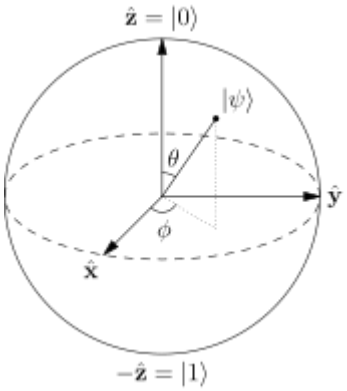


Рис.1 Сфера Блоха

Операції над кубітами можна розділити на два типи:

- Квантові вентилі – це такі перетворення, що відповідають поворотам вектора кубіта на сфері. Квантові вентилі – це унітарні перетворення кубіта.
- Вимірювання — це операція, завдяки якій можна дізнатися в якому стані знаходиться кубіт. При виконанні читання, кубіт займе один з двох станів $|0\rangle$ чи $|1\rangle$, з імовірностями $|a|^2$ і $|b|^2$ відповідно. При вимірюванні заплутаного кубіта, він переходить у мішаний стан. Мішані стани кубіта можна зобразити як точки, всередині сфери блоха.

3. Дії над кубітами

Для виконання дій над кубітами використовуються квантові вентилі.

Квантовий вентиль – це елемент квантового комп'ютера, який виконує операції над кубітами. Наприклад, це можуть бути операції повороту вектору кубіта чи логічні операції над кубітом.

Вентиль можна описати як унітарну матрицю. Результатом дії вентилю є добуток вектору стану кубіту і матриці вентилю.

Розмір матриці, яка описує дію вентиля: $2^n \times 2^n$, де n – кількість кубітів, над якими виконується операція. Так, для однокубітних вентилів матриця має розмір 2x2, а для двохкубітних – 4x4.

Наведемо декілька прикладів квантових вентилів в Q#:

1) Тотожне перетворення.

$\sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

2) Заперечення – вентиль Паулі X

$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ - X в Q#

3) Вентиль Паулі Y

$\sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ - Y в Q#

4) Вентить Паулі Z

$\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ - Z в Q#

Зобразити роботу цих вентилів можна за допомогою сфери Блоха:

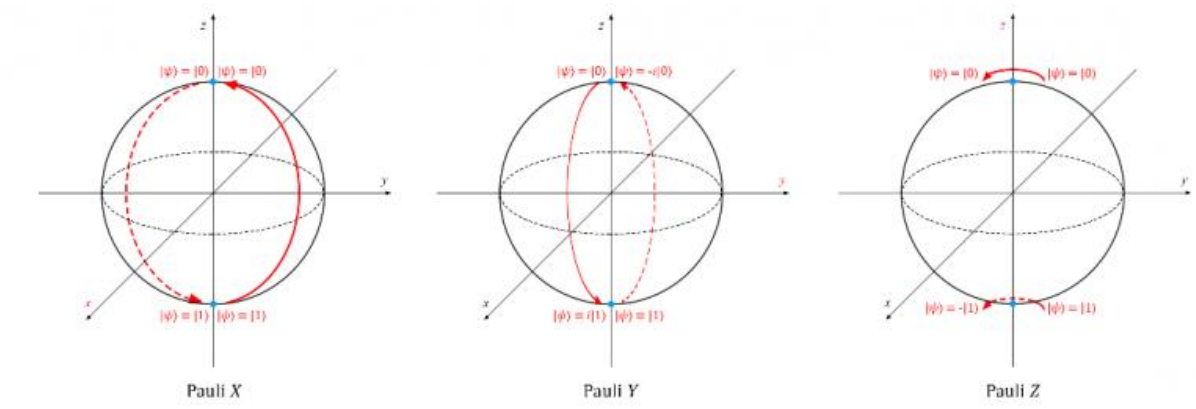


Рис. 2 Вентилі Паулі

5) Вентиль Адамара

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

Вентиль Адамара дозволяє перевести кубіт в суперпозицію станів |0⟩ і |1⟩.

6) Фазові зсуви

$R_\phi=\begin{bmatrix}1 & 0\\ 0 & e^{i\phi}\end{bmatrix}$ - R1 (R) в Q#

Крім звичайних вентилів, є і контрольовані. Такі вентилі мають два вхідних кубіти, і, за вимогою унітарності, два виходи.

Контрольовані операції позначаються літерою “C” перед символом операції. На вхід контрольованої операції O передаються контролюючий і контрольований кубіт. Якщо стан контролюючого кубіта рівний одиниці, то операція O виконується над контрольованим кубітом, інакше – стан кубіту не змінюється. Нехай, матриця операції O має вигляд:

$O=\begin{bmatrix}x_{00} & x_{01}\\ x_{10} & x_{11}\end{bmatrix}$

Тоді матриця перетворень CO має вигляд:

$C(O)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & x_{00} & x_{01}\\ 0 & 0 & x_{10} & x_{11}\end{bmatrix}$

4. Q# і Azure Quantum

Q# (QSharp) - це мова програмування з відкритим кодом, створена корпорацією Майкрософт для розробки та виконання квантових алгоритмів. Він є частиною пакету засобів розробки Quantum (QDK), який також містить бібліотеки Q#, квантові симулятори, розширення інших середовищ програмування та документацію по API. Окрім стандартної бібліотеки Q#,

пакет включає бібліотеки для квантової хімії, машинного навчання та числових розрахунків.

Програму Q# можна скомпілювати в автономну програму або викликати з ведучої програми, написаної мовою Python або в середовищі .NET.

При компіляції та запуску програми створюється екземпляр квантового симулятора, в який передається код Q#. Симулятор використовує код Q# для створення кубітів (що імітують квантові частинки) і виконує операції зі зміни їхнього стану. Результати квантових операцій у симуляторі потім повертаються до програми.

Ізоляція коду Q# у симуляторі гарантує, що алгоритми дотримуються законів квантової фізики та можуть правильно працювати на квантових комп'ютерах.

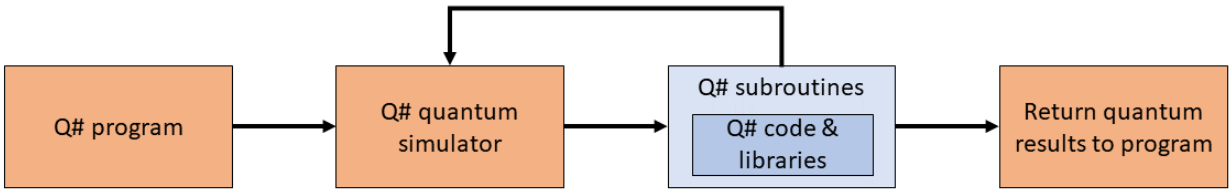


Рис 3. Схема взаємодії програми Q# з симулятором

Azure Quantum — це повноцінна хмарна служба, розроблена для надання користувачам віддаленого доступу до квантових комп’ютерів. Azure Quantum зосереджується на інтеграції інструментів квантових обчислень і його хмарного сервісу Azure. Щоб усунути проблеми, пов'язані

з крихкістю кубітів, Майкрософт використовує топологічні кубіти, які стабілізуються за рахунок маніпулювання їхньою структурою та оточення їх хімічними сполуками, що захищають кубіти від зовнішнього забруднення. Топологічні кубіти захищені від шуму завдяки топологічним властивостям квазічастинок,

що підвищує стійкість квантового обладнання до помилок. Ця підвищена стабільність дозволяє квантовому комп'ютеру масштабуватись, щоб виконувати більш тривалі та складні обчислення, а також спростити реалізацію більш комплексних рішень.

5. Квантові алгоритми

Квантовий алгоритм – це алгоритм, що виконується над кубітами у квантовому комп’ютері.

Квантовий алгоритм задає послідовність унітарних операцій (вентилів), які виконуються над кубітами. Квантовий алгоритм можна представити як у вигляді тексту, що описує послідовність дій, так і у графічному вигляді в якості системи вентилів.

Результат роботи квантового алгоритму має імовірнісний характер, тому задля збільшення ймовірності отримати правильний результат під час вимірювання, кількість операцій над кубітами збільшують, або проводять квантову систему через алгоритм декілька разів.

Головним типом завдань, які прискорюються квантовими алгоритмами, є завдання типу перебору. Їх можна розділити на дві основні групи:

1) Завдання моделювання динаміки складних систем

2) Математичні завдання, що зводяться до перебору варіантів:

Загальний випадок перебору: пошук Гровера, задача комівояжера

Завдання пошуку прихованих періодів: схема Шора використання швидкого квантового перетворення Фур'є та її аналоги.

6. Задачі оптимізації

Оптимізація – це група задач, які є найбільш імовірними претендентами для вирішення на квантових комп'ютерах, що завдяки своїм особливостям можуть мати значну перевагу над звичними комп'ютерами. Суть задачі зводиться того, щоб з усіх варіантів розв'язання підібрати найкращий (оптимальний). Це можуть бути задачі з балансування навантаження чи пошуку маршруту.

Для того, щоб квантовий комп'ютер зміг розв'язати таку задачу, потрібно створити її математичну модель. Математичну модель зводять до задання функції, яку необхідно мінімізувати чи максимізувати – це функція вартості (затрат).

До функції вартості можна додавати обмеження і «заохочення» - це її елементи, що впливають на рішення задачі, роблячи його більш або менш бажаним. Найпростішою задачею оптимізації можна вважати задачу пошуку глобального максимуму чи мінімуму функції однієї змінної.

При моделюванні задачі також користуються таким поняттям, як *простір пошуку* – це такий простір, що містить всі можливі рішення задачі. Кожна точка такого простору є допустимим рішенням задачі і відповідає одній конфігурації (*конфігурація* – набір значень змінних функції).

Ландшафт оптимізації – це значення функції вартості при кожній можливій конфігурації в просторі пошуку.

Виділяють три види ландшафтів оптимізації:

1) Ландшафт без нерівностей

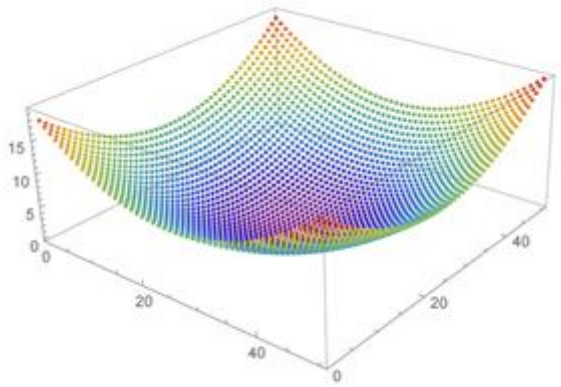


Рис 4. Ландшафт без нерівностей

Задачі з таким ландшафтом легко вирішуються за допомогою традиційних алгоритмів, таких як градієнтний спуск. Глобальний мінімум легко знайти, починаючи з будь-якої точки. Квантовий

комп'ютер не дає жодних переваг в таких задачах.

2) Структурований ландшафт

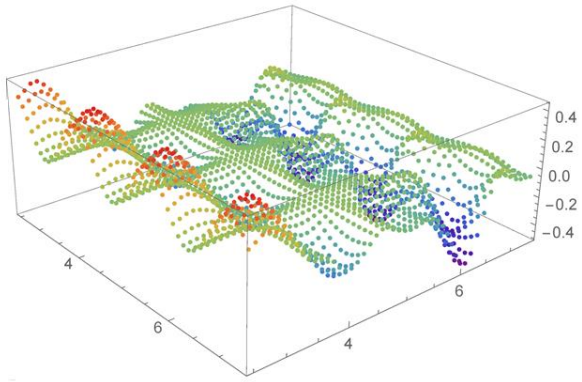


Рис 5. Структурований ландшафт

В такому ландшафті може бути багато локальних мінімумів. Найбільшою складністю є продовження пошуку після знаходження локального мінімуму. На такому ландшафті квантова оптимізація може мати перевагу над класичними алгоритмами.

3) Точковий ландшафт

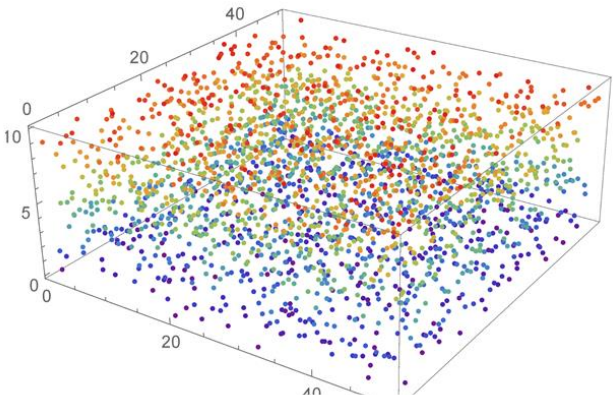


Рис 6. Точковий ландшафт

Такий ландшафт є хаотичним. У випадку точкового ландшафту, найефективнішим методом залишається перебір.

Сенс використання квантової оптимізації з’являється з виконанням умов:

- Задачу можна виразити через функцію затрат
- Ландшафт має бути структурованим і нерівним, мати декілька локальних мінімумів.
- Ефективність підвищується зі збільшенням кількості змінних функції затрат. Наприклад, якщо змінних менше сотні, то можна скористатися й класичним алгоритмом. Якщо ж змінних сотні – квантова оптимізація дозволяє покращити час обчислення на порядки.

7. Методи оптимізації azure quantum

- Квантовий відпал – це алгоритм, в якому амплітуда стану системи кубітів з кожним кроком змінюється зі зміною температури відповідно до функції вартості. На початку виконання алгоритму система знаходиться в стані з високою температурою, де задовільні та незадовільні конфігурації трапляються з однаковою імовірністю. Далі система повільно охолоджується, доки не досягне своєї мінімальної заданої температури. Функція вартості встановлює зв'язки між кубітами таким чином, що при охолодженні система з більшою ймовірністю займатиме більш «вигідний» стан – стан з найменшою внутрішньою енергією. При досягненні мінімальної температури вважається, що конфігурація системи і є рішенням.

- Паралельна закалка (відпуск) – алгоритм, схожий на квантову нормалізацію, але замість того, щоб проводити операції над однією системою кубітів, використовуються декілька систем (репліки), які ініціалізуються з різними початковими параметрами – температурою та випадковою конфігурацією. Далі над кожною з реплік виконуються ті ж самі взаємодії, що й при квантовій нормалізації, але репліки можуть бути переставлені між температурами, в ході пошуку рішення.
- Відпал популяції – алгоритм, що виконує перебір можливих значень функції за допомогою імітації блукаючої популяції. Під час виконання алгоритму в просторі пошуку виконуються розподілені «блукання», при чому початкова точка для кожного з них вибирається випадково. При кожному кроці зміни температури чи напруженості поля, виконується вибірка популяції: деякі «блукання» видаляються, а деякі дублюються. При цьому зберігається тенденція до збереження «блукань» з меншими витратами. Таким чином виконується об'єднання «блукань» в області з найменшою енергією.
- Алгоритм Монте-Карло – алгоритм, схожий на квантову нормалізацію, але використовується при наявності високих і тонких

бар’єрів на ландшафті. Для долання перешкод використовується імітація квантового тунелювання

- Субстохастичний алгоритм Монте-Карло – алгоритм, що об’єднує в собі особливості відпалу популяції і квантового алгоритму Монте-Карло.

Алгоритм емулює блукання популяції, але переміщення відбуваються з урахуванням квантових флуктуацій. Рішення про видалення чи дублювання «блукань» відбувається відповідно до функції вартості.

- Табу-пошук (або пошук з заборонами) – алгоритм, що нагадує локальний пошук, але дозволяє використання переміщень, що ведуть до підвищення вартості, якщо немає кращого рішення. Також встановлюються заборони, на вже розглянуті рішення.

Методи слід використовувати в залежності від особливостей ландшафту: кількості локальних мінімумів, висоти та ширини бар'єрів.

Для всіх перерахованих методів задачу (функцію вартості) необхідно представити в одному з двох виглядів: *PUBO* чи *Ising*.

PUBO – задача двійкової оптимізації з необмеженими многочленами.

Функція вартості такої задачі виглядає наступним чином:

$$H = \sum_{i} c_i x_i + \sum_{i,j} c_{i,j} x_i x_j + \sum_{i,j,k} c_{i,j,k} x_i x_j x_k \dots$$

,де $c \in R$, $x \in \{0,1\}$

Ising – має функцію затрат у вигляді:

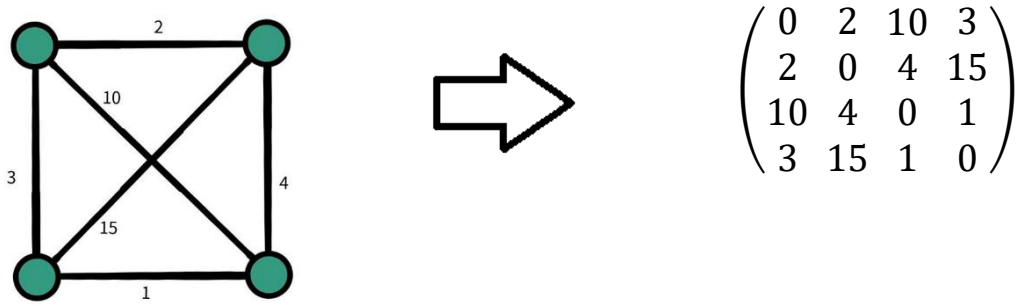
$$H = \sum_{i} c_i s_i + \sum_{i,j} c_{i,j} s_i s_j + \sum_{i,j,k} c_{i,j,k} s_i s_j s_k \dots$$

, де $c \in R$, $s \in \{-1,1\}$

8. Задача комівояжера

Задача комівояжера – задача знаходження найшвидшого, найвигіднішого чи найдешевшого маршруту для обходу всіх точок хоча б один раз. Існує велика кількість формулювань і рішень задачі. Наприклад: обійти всі точки і повернутися в початкову, відвідати точку максимум один раз, обійти всі точки з обмеженим об'ємом вантажу, що можна перевезти за раз чи з використанням декількох транспортів.

Зазвичай, задачу зображають у вигляді графу, де вершини відповідають місцям, які необхідно відвідати, а ребра – вартості переходу. В свою чергу, граф можна легко представити у вигляді матриці суміжності:



$$\begin{pmatrix} 0 & 2 & 10 & 3 \\ 2 & 0 & 4 & 15 \\ 10 & 4 & 0 & 1 \\ 3 & 15 & 1 & 0 \end{pmatrix}$$

Рис 7. Граф і його матриця суміжності

8.1 Вирішення задачі комівояжера за допомогою квантового комп'ютера

Розглянемо варіант задачі комівояжера, в якому необхідно обійти всі вершини графа, відвідавши їх максимум один раз та повернутися в початкову точку. Задачу розглянемо на повному, зваженому, орієнтованому графі.

Нехай, маємо N вершин, які необхідно відвідати.

Нехай, вартість переходу від вершини $i \in \{0, N-1\}$ до вершини $j \in \{0, N-1\}$

складає $c_{i,j}$.

Таким чином, матриця суміжності для N вершин матиме вигляд:

$$C = \begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,N-2} & c_{0,N-1} \\ c_{1,0} & \ddots & & \ddots & c_{1,N-1} \\ \vdots & & \ddots & & \vdots \\ c_{N-2,0} & \ddots & & \ddots & c_{N-2,N-1} \\ c_{N-1,0} & c_{N-1,1} & \cdots & c_{N-1,N-2} & c_{N-1,N-1} \end{pmatrix}$$

Використовуючи матрицю суміжності, можливо створити перелік усіх можливих переходів в графі:

$$c = (x_0, x_1, \dots x_{N-1}) \times \begin{pmatrix} c_{0,0} & \cdots & c_{0,N-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \cdots & c_{N-1,N-1} \end{pmatrix} \times \begin{pmatrix} x_N \\ \vdots \\ x_{2N-1} \end{pmatrix}$$

, де c – вартість переходу $x_i = 1, i \in \{0, N-1\}$ та $x_j = 1, j = \{N, 2N-1\}$, якщо відбувся перехід від вершини з номером i до вершини з номером $j - N$, інакше $x_i = 0, x_j = 0$.

Виразимо вартість $n-$ го переходу в маршруті:

$$c_n = (x_{nN}, \dots, x_{nN+i}, \dots, x_{nN+N-1}) \times \begin{pmatrix} c_{0,0} & \cdots & c_{0,N-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \cdots & c_{N-1,N-1} \end{pmatrix} \times \begin{pmatrix} x_{N(n+1)} \\ \vdots \\ x_{N(n+1)+j} \\ \vdots \\ x_{N(n+1)+N-1} \end{pmatrix}$$

, де n – номер переходу, c_n- вартість $n-$ го переходу.

Враховуючи, що кількість переходів в даних обставинах завжди рівна N, можемо записати вираз для обчислення вартості всього маршруту:

$$H = \sum_{n=0}^{N-1} c_n = \sum_{n=0}^{N-1} \left((x_{nN}, \dots x_{nN+N-1}) \times \begin{pmatrix} c_{0,0} & \cdots & c_{0,N-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \cdots & c_{N-1,N-1} \end{pmatrix} \times \begin{pmatrix} x_{N(n+1)} \\ \vdots \\ x_{N(n+1)+N-1} \end{pmatrix} \right)$$

$$= \sum_{n=0}^{N-1} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} c_{i,j} x_{Nn+i} x_{N(n+1)+j}$$

,де H – гамільтоніан – енергія системи при будь-якій конфігурації (функція вартості), $n \in \{0, N-1\}$ – номер кроку, $i, j \in \{0, N-1\}$ – номер початкової та кінцевої вершини переходу.

Надалі задача полягатиме в тому, щоб знайти таку конфігурацію $x_i, i \in \{0, N^2 + N - 1\}$, при якій H матиме найменше можливе значення.

8.1.1 Відсіювання неправильних рішень

В даній реалізації функції Гамільтоніана, виникає низка проблем. Під час пошуку рішення, до простору пошуку входять конфігурації, при яких значення функції вартості буде в глобальному мінімумі, проте рішення не буде нас влаштовувати.

Такими конфігураціями можуть бути:

- $\sum_{i=0}^{N^2+N-1} x_i = 0$ - всі x рівні нулю. В такому випадку $H = 0$, глобальний мінімум, який відповідає даній конфігурації.
- Конфігурації, при яких відвідується більше однієї точки за раз
- Одна вершина відвідана два і більше разів

Щоб відсіяти конфігурації, при яких всі x рівні нулю, опустимо ландшафт у всіх точках, окрім тих, в який всі x рівні нулю:

$$-\lambda \sum_{i=0}^{N^2+N-1} x_i$$

, де λ – коефіцієнт, що є достатньо великим, для того щоб відсіяти конфігурації, що нас не влаштовують, проте не занадто великим, щоб вплинути на рішення, які знаходяться поруч з незадовільними.

В такому випадку, найменшу вартість буде мати маршрут, в якому відвідуються всі точки одночасно на кожному кроці. Збільшимо вартість конфігурацій, в яких відвідується більше однієї точки за раз:

$$\lambda \sum_{n=0}^{N} \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} x_{Nn+i} x_{Nn+j}$$

, де n – номер переходу. $n \in \{0, N\}$, а не $\{0, N-1\}$, оскільки один перехід додає повернення в початкову точку.

Далі потрібно відсіяти конфігурації, при яких одна й та ж вершина відвідується більше, ніж один раз.

$$\lambda \sum_{i=0}^{N^2+N-1} \sum_{j=N+i}^{N^2-1} x_i x_j$$

, при чому j збільшується з кроком N. Таким чином, якщо вершина була повторно відвідана на будь-якому з j-их кроків, то вартість збільшиться.

Наступним кроком необхідно задати початкову та кінцеву точки:

$$-(\alpha x_0 + \beta x_{N^2})$$

, де α і β - коефіцієнти, що є достатньо великими, для того щоб відсіяти конфігурації, що нас не влаштовують, проте не занадто великими, щоб вплинути на рішення, які знаходяться поруч з незадовільними, x_0 і x_{N^2} рівні одиниці, якщо перша вершина була відвідана на першому і останньому кроці.

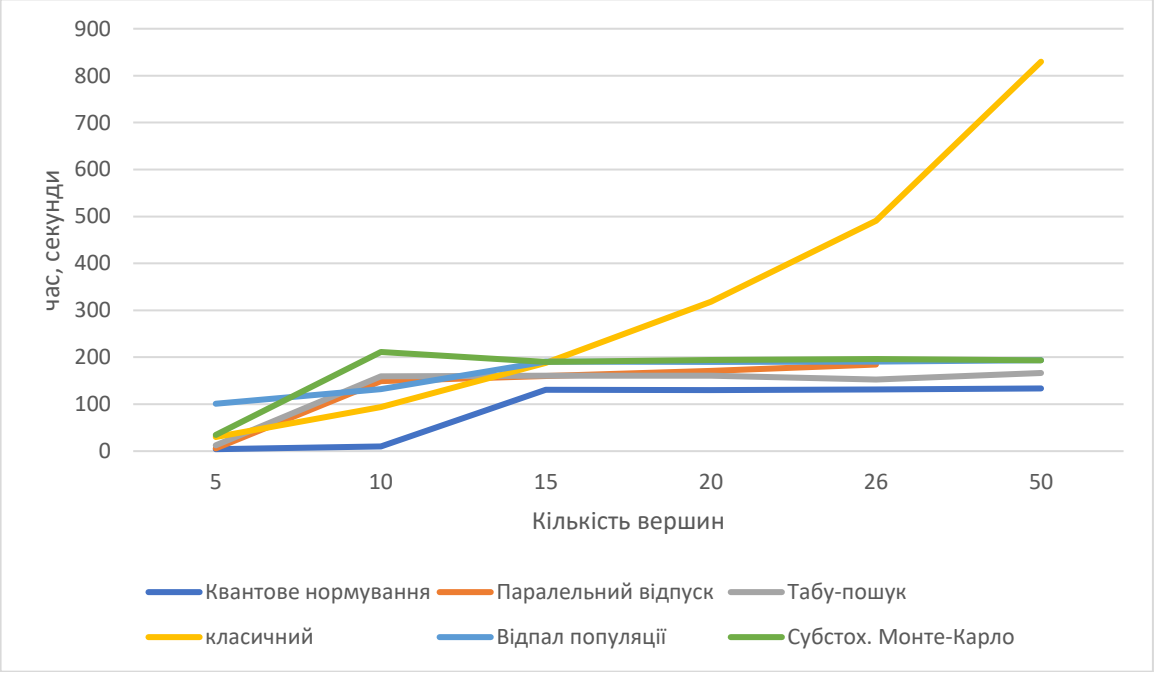
Просумувавши всі вище описані вирази, отримуємо кінцеву функцію вартості:

$$H = \sum_{n=0}^{N-1} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} c_{i,j} x_{Nn+i} x_{N(n+1)+j} - \alpha \sum_{i=0}^{N^2+N-1} x_i + \beta \sum_{n=0}^{N} \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} x_{Nn+i} x_{Nn+j} + \gamma \sum_{i=0}^{N^2+N-1} \sum_{j=N+i}^{N^2-1} x_i x_j - (\lambda x_0 + \mu x_{N^2})$$

8.2 Реалізація та аналіз алгоритму

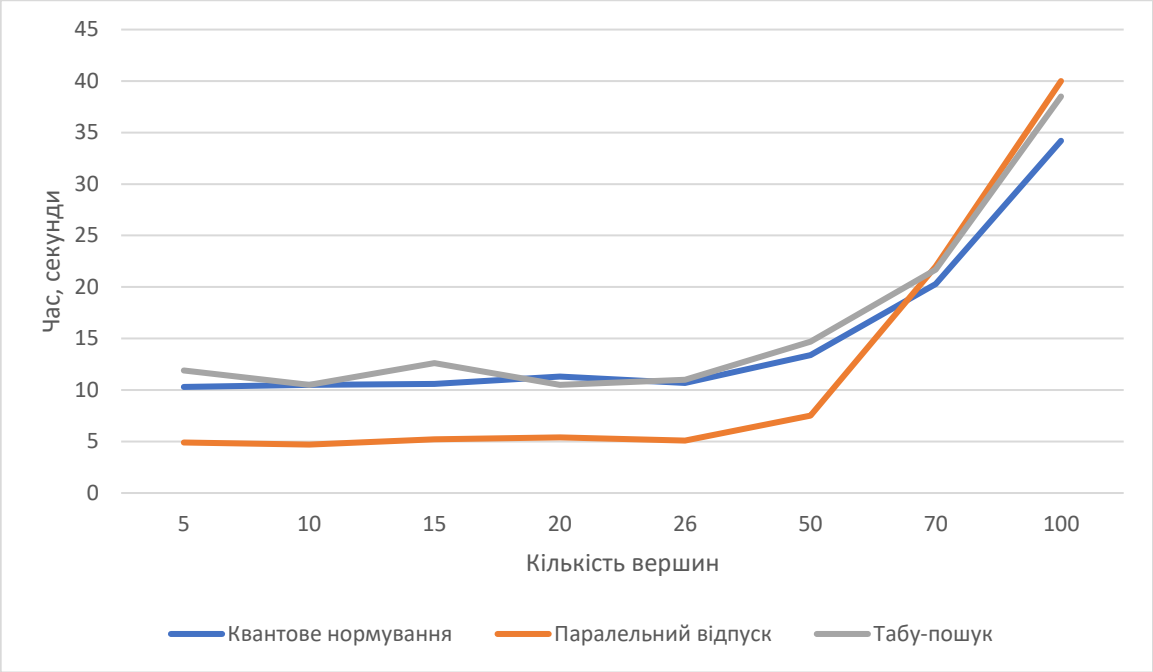
Для деяких мов програмування, серед яких python, наявні бібліотеки для роботи Q# та взаємодії з хмарним сервісом azure. Завдяки цьому, наявна можливість задати всі необхідні дані для роботи ядра Q# в середовищі python, відправити задачу на віддалений квантовий комп'ютер та інтерпретувати результат роботи КК мовою python.

Ознайомитися з блок схемою алгоритму та кодом мовою python можна в додатках №1 та №2 відповідно.
В якості методу вирішення проблеми можна обрати будь-який п.7.
Визначити ефективність використання кожного з них для пошуку маршруту можна виконавши низку тестів із засіканням часу, необхідного для пошуку конфігурації з найменшою енергією. Платформа azure надає варіанти методів з та без параметрів за замовчуванням. Тести проведені для повних, орієнтованих, зважених графів з кількістю вершин 5,10,15,20,26,50.
Для наочності, додатково проведено тест з класичним комп'ютером і використанням генетичного алгоритму. В кожному тесті вартість переходів визначається випадково.
Методи з параметрами за умовчанням:



Із графіку залежності часу виконання від кількості вершин графу, видно, що квантовий комп'ютер, навіть з стандартними параметрами методів, має значну перевагу над класичним. Зі збільшенням кількості вершин час пошуку оптимальної конфігурації зростає дуже повільно, в той час як класичний комп'ютер потребує все більше часу із кожно новою вершиную.
Із графіків можна зробити висновок, що складність класичного алгоритму складає близько $O(n\text{^}2)$, а квантового $O(n \log(n))$

Покращити час виконання алгоритму на квантовому комп'ютері можна підібравши параметри для вирішувачів.



Висновок

В роботі був написаний та проаналізований алгоритм для квантового комп’ютера, що виконує задачу комівояжера, тобто шукає найбільш оптимальний маршрут обходу графу із поверненням в початкову точку. Були проведені тести програми із різними варіантами вирішення задачі і порівняний час виконання алгоритму квантовим комп’ютером і класичним. Із результатів тестів видно, що навіть із усіма недоліками теперішніх квантових комп’ютерів, вони вже можуть значно краще виконувати такого роду задачі. Так, для пошуку шляху в графі з 50 вершин, квантовому комп’ютеру знадобилося всього 7.5 секунд, в той час як класичному – 829 секунд. Крім того, навіть із стандартними параметрами методу оптимізації, складність алгоритму для квантового комп’ютера складає $O(n \log(n))$, а в класичного з використанням генетичного алгоритму - $O(n\text{^}2)$.

Завдяки тому, що алгоритм реалізовано з використанням мови програмування Python, його можна легко змінювати для виключення маршрутів, зміни початкової і кінцевої точки маршруту. Крім того, можливо передавати результат виконання алгоритму зовнішнім програмам, для їх подальшого використання, наприклад, для візуалізації маршруту на карті.

Джерела

1. Введение в оптимизацию. Имитация отжига – Веб-сайт Habr

URL: https://habr.com/ru/post/209610/

3. Квантовые цепи и вентили — вводный курс – Веб-сайт Habr URL:https://habr.com/ru/company/microsoft/blog/351628/

4. Общие сведения о языке программирования Q# и пакете средств разработки Quantum (QDK) – Веб-сайт, документація Microsoft.

URL: https://docs.microsoft.com/ru-ru/azure/quantum/overview-what-is-qsharp-and-qdk

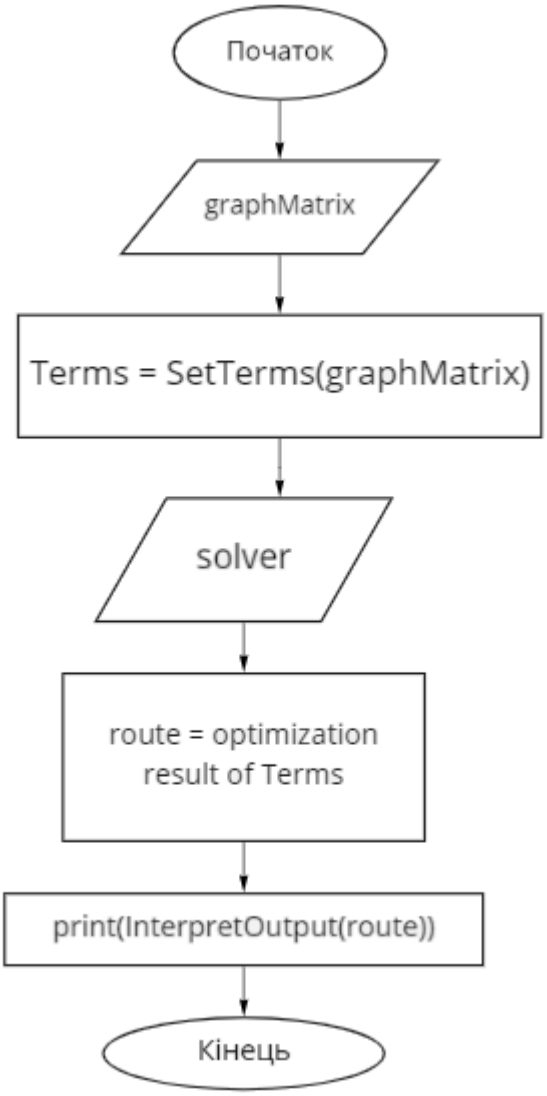
5. Azure Quantum documentation – Веб-сайт, документація Microsoft
URL: https://docs.microsoft.com/en-us/azure/quantum/

6. What is optimization? – Веб-сайт, документація Microsoft
URL: https://docs.microsoft.com/en-us/azure/quantum/optimization-overview-introduction

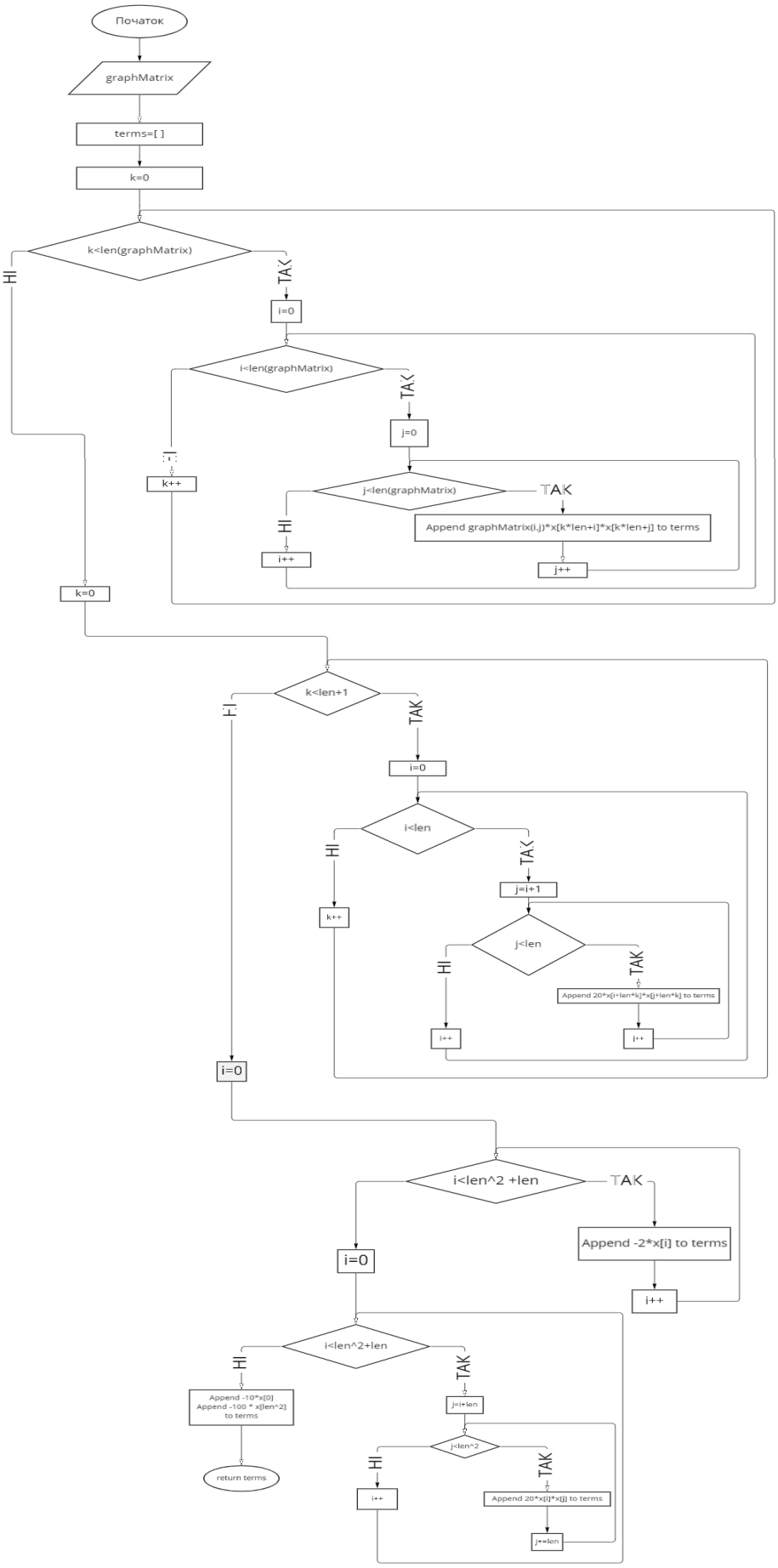
Додатки

Додаток №1: Блок-схема програми

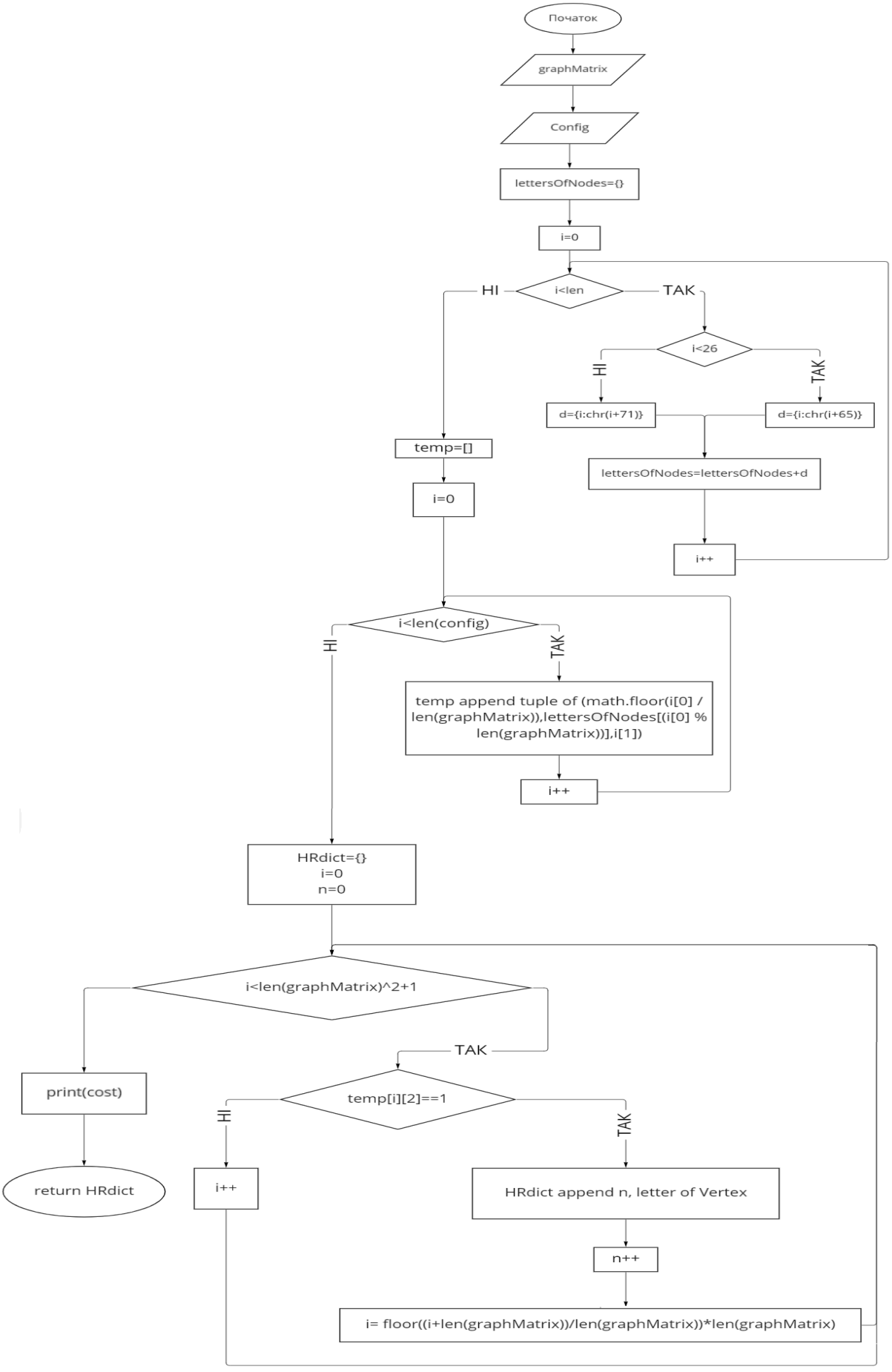
Тіло програми:



Функція, яка обчислює елементи Гамільтоніана, SetTerms:



Функція, що інтерпретує отриману конфігурацію, InterpretOutput:



Додаток 2: Код програми, мовою python

```
def SetTerms(graphMatrix):

    terms = []
    for k in range(0, len(graphMatrix)):
        for i in range(0, len(graphMatrix)):
            for j in range(0, len(graphMatrix)):
                terms.append(Term(c = graphMatrix.item((i,j)), indices = [i + (len(graphMatrix) *k ), j + (len(graphMatrix) * (k + 1))]))

    for k in range(0,len(graphMatrix)+1):
        for i in range(0,len(graphMatrix)):
            for j in range(i+1,len(graphMatrix)):
                terms.append(Term(c = int(20 * np.max(graphMatrix)), indices = [i + (len(graphMatrix) * k), j + (len(graphMatrix) * k)]))

    for i in range(0, len(graphMatrix) + len(graphMatrix) * (len(graphMatrix))):
        terms.append(Term(c = int(-2 * np.max(graphMatrix)), indices = [i]))

    for i in range(0, len(graphMatrix) + len(graphMatrix) * (len(graphMatrix))):
        for j in range(i + len(graphMatrix), len(graphMatrix) * (len(graphMatrix)), len(graphMatrix)):
            terms.append(Term(c = int(20 * np.max(graphMatrix)), indices = [i,j]))

    terms.append(Term(c = int(-10 * np.max(graphMatrix)), indices = [0]))

    terms.append(Term(c = int(-100 * np.max(graphMatrix)), indices = [len(graphMatrix) * (len(graphMatrix))]))

    return terms

def InterpretOutput(config, graphMatrix):

    lettersOfNodes={}
    for i in range(len(graphMatrix)):
        if i<26:
            d={i:chr(i+65)}
        else:
            d={i:chr(i+71)}
        lettersOfNodes.update(d)

    config = [(int(a), b) for a, b in config]
    config.sort(key=lambda tup: tup[0])

    temp=[]
    for i in config:
        tup = (math.floor(i[0] / len(graphMatrix)),lettersOfNodes[(i[0] % len(graphMatrix))],i[1])
        temp.append(tup)

    HRdict = {}
    HRdict['Route'] = {}
    i=0
    n=0
    while i < len(graphMatrix) * (len(graphMatrix) + 1):
        if temp[i][2] == 1:
            HRdict['Route'].update({n: temp[i][1]})
            n += 1
            i=math.floor((i+len(graphMatrix))/len(graphMatrix))*len(graphMatrix)
        else:
            i=i+1

    if (len(HRdict['Route']) - 1) != len(graphMatrix):
        print("Помилка! Відвідана неправильна кількість вершин!\n Очікувано: "+len(graphMatrix)+"\nВідвідано: "+len(HRdict['Route']) - 1)

    if HRdict['Route'][0] != HRdict['Route'][len(HRdict['Route'])-1]:
        print("Помилка! Не повернулися в початкову точку!")

    cost = 0
    for i in range(len(graphMatrix)):
        a=ord(HRdict['Route'][i])
        if a <= 90:
            a=a-65
        else:
            a=a-97
        b=ord(HRdict['Route'][i+1])
        if b <= 90:
            b=b-65
        else:
            b=b-97
        cost=cost+graphMatrix.item(a,b)
    print("route cost:"+str(cost))
    return HRdict
workspace = Workspace (
    subscription_id = "",
    resource_group = "",
    name = "",
    location = ""
)
nodes = 5
costRange = 10
```

```
graphMatrix = np.random.randint(costRange, size=(nodes,nodes))
nodes = len(graphMatrix)

print("Input matrix:")
for i in range(nodes):
    for j in range(nodes):
        print(graphMatrix[i][j],end='\t')
    print()

problem = Problem(name="TSP for diploma", problem_type=ProblemType.pubo, terms=SetTerms(graphMatrix))

#solver = SimulatedAnnealing(workspace, timeout = 120)
solver = ParallelTempering(workspace,sweeps=10,replicas=5,all_betas=[1.15,3.14,4.3,5.8,7.2], timeout = 300)
#solver = Tabu(workspace, timeout = 120)

curtime = time.time()
route = solver.optimize(problem)
endtime = time.time()
print("done"+"\n"+"Time spent (s):",end="")
print(endtime-curtime)
print(InterpretOutput(route['configuration'].items(), graphMatrix))
```