

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з освітнього компоненту

«Технології паралельних обчислень. Курсова робота»

Тема: Алгоритм пошуку в ширину та його паралельна реалізація мовою програмування Java

|  |  |
| --- | --- |
| **Керівник**:  асп. Дифучина Олександра Юріївна  «Допущено до захисту»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 р.  Захищено з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Члени комісії:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Виконавець**:  Середюк Валентин Васильович  студент групи ІП-02  залікова книжка № ІП-0222  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «23» травня 2023 р.  Інна СТЕЦЕНКО  Олександра ДИФУЧИНА |

**Київ – 2023**

**ЗАВДАННЯ**

1. Виконати огляд існуючих реалізацій алгоритму пошуку в ширину, послідовних та паралельних, з відповідними посиланнями на джерела інформації (статті, книги, електронні ресурси). Зробити висновок про актуальність дослідження.
2. Виконати розробку послідовного алгоритму пошуку в ширину та реалізувати мовою програмування Java. Дослідити швидкодію алгоритму при зростанні складності обчислень та зробити висновки про необхідність паралельної реалізації алгоритму.
3. Виконати розробку паралельного алгоритму пошуку в ширину та реалізувати мовою програмування Java. Забезпечити зручне введення даних для початку обчислень.
4. Виконати тестування алгоритму пошуку в ширину, що доводить коректність результатів обчислень.
5. Виконати дослідження швидкодії алгоритму пошуку в ширину при зростанні кількості даних для обчислень.
6. Виконати експериментальне дослідження прискорення розробленого алгоритму пошуку в ширину при зростанні кількості даних для обчислень. Реалізація алгоритму вважається успішною, якщо прискорення більше 1,2.
7. Зробити висновки про переваги паралельної реалізації обчислень для алгоритму пошуку в ширину та програмних засобів, які використовувались.

**АНОТАЦІЯ**

У даній роботі досліджується алгоритм пошуку в ширину (BFS) та його паралельна реалізація мовою програмування Java. Метою роботи є оцінка швидкодії паралельної версії алгоритму порівняно з послідовною. Буде проводиться огляд алгоритму BFS, його принципу роботи. Далі розглядатиметься паралельна реалізація BFS, визначаться основні кроки та способи розділення простору пошуку між потоками. Алгоритм BFS буде представлено мовою програмування Java. Також буде демонструватися використання потоків або пулу потоків у Java для покращення продуктивності BFS.

Буде проводитися оцінка продуктивності та оптимізація паралельної версії алгоритму. Вимірюватиметься час виконання паралельної версії у порівнянні з послідовною, будуть наведені графіки порівняння двох алгоритмів. Очікується, що паралельна реалізація дозволить прискорити час роботи в порівнянні з послідовною реалізацією мінімум в 1,2 раза.

ЗМІСТ

[ВСТУП 5](#_Toc136311478)

[1 ОПИС ПОСЛІДОВНОГО АЛГОРИТМУ ПОШУКУ В ШИРИНУ ТА ЙОГО ВІДОМИХ ПАРАЛЕЛЬНИХ РЕАЛІЗАЦІЙ 6](#_Toc136311479)

[1.1 Загальна інформація 6](#_Toc136311480)

[1.2 Концепція алгоритму 6](#_Toc136311481)

[1.3 Реалізація послідовного алгоритму 7](#_Toc136311482)

[1.4 Реалізація паралельного алгоритму 9](#_Toc136311483)

[2 РОЗРОБКА ПОСЛІДОВНОГО АЛГОРИТМУ ТА АНАЛІЗ ЙОГО ШВИДКОДІЇ 12](#_Toc136311484)

[2.1 Розробка послідовного алгоритму 12](#_Toc136311485)

[3 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ ТА ЙОГО КОРОТКИЙ ОПИС 13](#_Toc136311486)

[4 РОЗРОБКА ПАРАЛЕЛЬНОГО АЛГОРИТМУ З ВИКОРИСТАННЯМ ОБРАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ПРОЕКТУВАННЯ, РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ 14](#_Toc136311487)

[5 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ АЛГОРИТМУ 15](#_Toc136311488)

[ВИСНОВКИ 16](#_Toc136311489)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 17](#_Toc136311490)

[ДОДАТКИ 18](#_Toc136311491)

[Додаток А. Нзава додатку 18](#_Toc136311492)

[Додаток Б. Назва додатку 19](#_Toc136311493)

[Додаток В. Назва додатку 20](#_Toc136311494)

# ВСТУП

У сучасному світі, де обчислювальні завдання стають все більш складними і об'ємними, паралельні обчислення набувають особливої актуальності. З кожним роком технології розвиваються швидкими темпами, дозволяючи реалізувати паралельність на все більш широкій шкалі. Такі області, як штучний інтелект, обробка великих об'ємів даних, комп'ютерне зору, наукові дослідження та інші, потребують швидкого та ефективного вирішення складних завдань. Однією з основних тенденцій у розвитку паралельних обчислень є зростання кількості ядер та потоків в процесорах. Замість того, щоб прискорювати окремі ядра до надзвичайно високих швидкостей, виробники процесорів почали ставити на акцент на збільшення кількості ядер, що дозволяє виконувати більше завдань паралельно. Це відкриває шлях до ефективного використання паралельних обчислень для прискорення обчислювальних процесів.

Одним з найбільш популярних алгоритмів, що використовуються в різних областях обчислювальної науки, є алгоритм пошуку в ширину (BFS – Breadth-First Search). Він використовується для пошуку найкоротшого шляху між двома вузлами графа, перебираючи сусідні вузли на кожному рівні глибини перед переходом до наступного рівня. Завдяки зростанню потужності сучасних обчислювальних систем, реалізація алгоритму BFS у паралельній середовищі стає важливою задачею. Паралельні обчислення дозволяють розподілити завдання між багатьма процесорами або ядрами, що дозволяє прискорити обчислення та забезпечити ефективне використання обчислювальних ресурсів.

Мова програмування Java, з її широким спектром бібліотек та засобів, є однією з найпоширеніших мов для розробки паралельних програм. Вона надає розширені можливості для створення багатопоточних додатків та ефективного використання паралельних обчислень.

# 1 ОПИС ПОСЛІДОВНОГО АЛГОРИТМУ ПОШУКУ В ШИРИНУ ТА ЙОГО ВІДОМИХ ПАРАЛЕЛЬНИХ РЕАЛІЗАЦІЙ

## 1.1 Загальна інформація

Алгоритм пошуку в ширину (BFS, Breadth-First Search) є одним з базових алгоритмів графового пошуку. Він дозволяє обходити всі вершини графа, починаючи з заданої стартової вершини, і відшукати найкоротші шляхи до всіх досяжних вершин. Алгоритм гарантує знаходження найкоротшого шляху до будь-якої досяжної вершини, при умові, що всі ребра мають однакову вагу. Він також може бути використаний для пошуку найкоротшого шляху між двома вершинами.

Складність алгоритму пошуку в ширину залежить від кількості вершин (n) та ребер (m) у графі. У найгіршому випадку, коли граф є повністю зв'язним, складність алгоритму можна оцінити як O(n + m), оскільки кожна вершина та кожне ребро відвідується рівно один раз. У випадку розріджених графів, де кількість ребер (m) набагато менша за кількість вершин (n), складність алгоритму можна приблизно оцінити як O(n), оскільки n >> m.

Алгоритм пошуку в ширину є ефективним і широко використовується у багатьох областях, включаючи графові структури даних, алгоритми маршрутизації в комп'ютерних мережах, графічні системи та штучний інтелект.

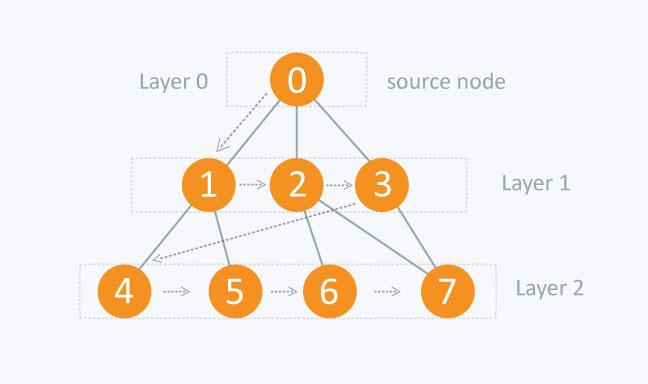
## 1.2 Концепція алгоритму

Алгоритм пошуку в ширину (BFS) працює на основі принципу «перебування у шарах». Він починає з заданої стартової вершини і розглядає всі сусідні вершини цієї стартової вершини перед переходом до сусідніх вершин наступного шару.

Опис роботи алгоритму:

1. Початкова вершина ставиться у чергу.
2. Позначаємо початкову вершину як відвідану.
3. Поки черга не порожня:
   1. Беремо вершину з початку черги.
   2. Перевіряємо всіх сусідніх вершин для поточної вершини, які ще не відвідані.
   3. Додаємо сусідні вершини до кінця черги та позначайте їх як відвідані.
4. Повторюємо крок 3 до тих пір, поки черга не порожня.

Алгоритм продовжує просуватися шар за шаром по графу, знаходячи всі вершини на поточному шарі, перш ніж переходити до наступного шару (рис 1.1).

  
Рисунок 1.1 – Приклад обходу графа за допомогою BFS

Це гарантує знаходження найкоротшого шляху від початкової вершини до всіх досяжних вершин. Процес продовжується до тих пір, поки не будуть відвідані всі досяжні вершини або досягнутий певний критерій завершення. Наприклад, відвідані всі вершини, знайдена шукана вершина або досягнуто певної глибини пошуку.

## 1.3 Реалізація послідовного алгоритму

Для кращого розуміння, зобразимо роботу алгоритму у вигляді блок-схеми (рис 1.2).

Зображення, що містить текст, квитанція, чорно-білий, схема

Автоматично згенерований опис  
Рисунок 1.2 – Блок-схема роботи послідовного алгоритму

У загальному, алгоритм можна поділити на 2 основні частини. Першою є знаходження сусідів початкової вершини, а другою це обробка цих самих вершин по принципу, що і для початкової. Головною умовою, що і пришвидшує виконання алгоритму, є перевірка вершини на відвідуваність перед тим, як її добавляти у чергу.

Для реалізації алгоритму мовою програмування Java, напишемо псевдокод послідовної реалізації алгоритму пошуку в ширину (код 1.1):

**procedure** BFS(graph, startVertex):

**create** an **empty** queue

**create** a visited **array** and **initialize** all elements **to** false

**enqueue** startVertex **into** the queue

**set** visited[startVertex] **to** true

**while** the queue **is not empty**:

currentVertex = **dequeue** from the queue

**for each** neighborVertex **of** currentVertex:

**if** neighborVertex **is not** visited:

**enqueue** neighborVertex **into** the queue

**set** visited[neighborVertex] **to** true

**end procedure**

Код 1.1 – Псевдокод послідовного алгоритму пошуку в ширину

## 1.4 Реалізація паралельного алгоритму

Серед відомих паралельних реалізацій алгоритму BFS можна виділити 4 основних, а саме:

1. Паралельна обробка вершин: У цьому методі граф розділяється на декілька незалежних частин, і кожна частина обробляється окремим потоком або процесом. Кожен потік виконує BFS на своїй частині графа. Після завершення обробки кожної частини результати можуть бути об'єднані.
2. Розподілення черги: Цей підхід включає розподілення черги вершин між потоками або процесами. Кожен потік бере вершину з черги і обробляє її сусідів. Це дозволяє розпаралелити обробку сусідів вершин і скоротити загальний час виконання.
3. Метод «Wavefront»: У цьому методі граф обробляється «хвилями». Кожна хвиля представляється як окремий етап обробки, де всі вершини на деякій відстані від початкової вершини обробляються паралельно. Після завершення однієї хвилі переходить до наступної хвилі з вершинами, що знаходяться на більшій відстані. Цей підхід дозволяє паралелізувати обробку вершин на кожній відстані.
4. Гібридна паралелізація: Цей підхід поєднує різні методи паралелізації для досягнення кращої продуктивності. Наприклад, можна поєднати паралельну обробку вершин з розподіленим чергуванням для більш ефективного використання ресурсів та покращення часу виконання.

Давайте детальніше розглянемо метод розподілення черги, так як він є найбільш використовуваним.

Метод розподілення черги в паралелізації алгоритму пошуку в ширину включає розподіл черги вершин між потоками або процесами. Ідея полягає в тому, щоб кожен потік брав вершину з черги і обробляв її сусідів, що дозволяє паралелізувати обробку сусідніх вершин та прискорити алгоритм.

Основний алгоритм з методом розподілення черги виглядає наступним чином:

1. Створити загальну чергу вершин та поділити її між доступними потоками або процесами. Це можна зробити, наприклад, шляхом розподілу вершин у черзі на рівні підмножини для кожного потоку або процесу.
2. Кожен потік або процес бере вершину зі своєї частини черги.
3. Обробляється вершина, яку взяв потік або процес. Виконується потрібна обробка або зберігається необхідна інформація про вершину.
4. Потік або процес перевіряє всіх сусідніх вершин, які ще не були відвідані. Якщо сусідня вершина не відвідана, вона додається до власної частини черги потоку або процесу.
5. Кроки 2-4 повторюються, поки черга потоку або процесу не порожня, або досягнутий певний критерій завершення.
6. Після завершення обробки усіх вершин потоки або процеси можуть об'єднати результати своєї обробки.

Метод розподілення черги може суттєво покращити продуктивність алгоритму BFS, особливо при роботі з великими графами, де обробка вершин може бути ресурсоємкою операцією. Проте, необхідно правильно розподіляти завдання та враховувати синхронізацію для досягнення правильності та ефективності паралельної обробки.

Для кращого розуміння роботи даного методи, зобразимо його використання у вигляді псевдокоду (код 1.2):

**procedure** ParallelBFS(graph, startVertex, numThreads):

**create** an **empty** shared queue

**create** a shared visited **array** and **initialize** all elements **to** false

**enqueue** startVertex **into** the shared queue

**set** visited[startVertex] **to** true

**create** numThreads worker threads/processes

**for** **each** thread **of** numThreads:

**create** a worker thread/process

**wait** for all worker threads/processes **to** complete

**end procedure**

**procedure** WorkerThread():

**while** the shared queue **is not** **empty**:

**if** the shared queue **is not empty**:

currentVertex = **dequeue** from the shared queue

**if** currentVertex **is not null**:

**for each** neighborVertex **of** currentVertex:

**if** neighborVertex **is not** visited:

**set** visited[neighborVertex] **to** true

**enqueue** neighborVertex **into** the shared queue

**end procedure**

Код 1.2 – Псевдокод паралельного алгоритму пошуку в ширину методом розподілення черги

# 2 РОЗРОБКА ПОСЛІДОВНОГО АЛГОРИТМУ ТА АНАЛІЗ ЙОГО ШВИДКОДІЇ

## 2.1 Розробка послідовного алгоритму

# 3 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ ТА ЙОГО КОРОТКИЙ ОПИС

# 4 РОЗРОБКА ПАРАЛЕЛЬНОГО АЛГОРИТМУ З ВИКОРИСТАННЯМ ОБРАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ПРОЕКТУВАННЯ, РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ

# 5 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ АЛГОРИТМУ

Таблиця 5.1. – Назва таблиці

|  |  |  |
| --- | --- | --- |
| Кількість елементів | Час паралельного алгоритму, мікросекунд | Час послідовного алгоритму, мікросекунд |
| 1000 | 2477 | 475 |
| 3000 | 2338 | 837 |

Рисунок 5.1. – Назва рисунку

Рисунок 5.3. – Результат виконання програми у консолі

# ВИСНОВКИ

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. …
2. …
3. …
4. …
5. …

# ДОДАТКИ

## Додаток А. Нзава додатку

## Додаток Б. Назва додатку

## Додаток В. Назва додатку