**zĐẠI HỌC ĐÀ NẴNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**VÀ TRUYỀN THÔNG VIỆT – HÀN**

A logo with a black background

Description automatically generated

**BÁO CÁO CUỐI KỲ**

**HỌC PHẦN: LẬP TRÌNH SONG SONG**

|  |  |
| --- | --- |
| Sinh viên thực hiện: Giảng viên:  Lớp: | **Hoàng Hữu Tiến Đạt - 21AD011**  **TS. Nguyễn Đình Lầu**  **21AD** |

***Đà Nẵng, ngày 25 tháng 05 năm 2024***

**ĐẠI HỌC ĐÀ NẴNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**VÀ TRUYỀN THÔNG VIỆT – HÀN**

A logo with a black background

Description automatically generated

**BÁO CÁO CUỐI KỲ**

**HỌC PHẦN: LẬP TRÌNH SONG SONG**

***Đà Nẵng, ngày 25 tháng 05 năm 2024***

**LỜI CẢM ƠN**

Trong suốt thời gian thực hiện báo cáo cá nhân học phần Lập trình song song, em đã nhận được sự hỗ trợ, giúp đỡ của thầy. Với tình cảm sâu sắc, chân thành, cho phép chúng em được bày tỏ lòng biết ơn sâu sắc đến thầy đã tạo điều kiện giúp đỡ cho chúng em trong quá trình học tập và nghiên cứu đề tài.

Trước hết chúng em xin gửi tới thầy cô Khoa Kỹ thuật máy tính trường Đại học Công nghệ thông tin và Truyền thông Việt - Hàn lời chào trân trọng, lời chúc sức khỏe và lời cảm ơn sâu sắc. Với sự quan tâm dạy dỗ, chỉ bảo tận tình chu đáo của thầy cô để chúng em có thể hoàn thành báo cáo.

Đặc biệt chúng em xin được bày tỏ lòng biết ơn chân thành sâu sắc tới thầy giáo **TS.Nguyễn Đình Lầu** đã trực tiếp giúp đỡ, hướng dẫn em hoàn thành tốt báo cáo này trong thời gian qua.

Với điều kiện thời gian cũng như kinh nghiệm còn hạn chế, bài tập lớn này không thể tránh được những thiếu sót. Chúng em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các thầy cô để chúng em có thể bổ sung, nâng cao ý thức của mình, phục vụ tốt hơn công tác thực tế sau này.

***Em xin trân trọng cảm ơn.!***

**NHẬN XÉT CỦA GIẢNG VIÊN**

**(Của giảng viên hướng dẫn)**

**………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………**

Đà Nẵng, ngày 25 tháng 05 năm 2024.

Giảng viên hướng dẫn

TS. Nguyễn Đình Lầu

**MỤC LỤC**

**[MỞ ĐẦU](#_Toc167476800)** [1](#_Toc167476800)

**[1.](#_Toc167476801)****[Lý do chọn đề tài](#_Toc167476801)** [1](#_Toc167476801)

**[1.1.](#_Toc167476802)****[Mục tiêu của đề tài](#_Toc167476802)** [2](#_Toc167476802)

**[1.2.](#_Toc167476803)****[Đối tượng và phạm vi nghiên cứu](#_Toc167476803)** [2](#_Toc167476803)

**[1.2.1.](#_Toc167476804)****[Đối tượng:](#_Toc167476804)** [2](#_Toc167476804)

**[1.2.2.](#_Toc167476805)****[Phạm vi:](#_Toc167476805)** [2](#_Toc167476805)

**[1.3.](#_Toc167476806)****[Cách tiếp cận](#_Toc167476806)** [2](#_Toc167476806)

**[1.4.](#_Toc167476807)****[Phương pháp nghiên cứu](#_Toc167476807)** [3](#_Toc167476807)

**[2.](#_Toc167476808)****[Nội dung của đề tài](#_Toc167476808)** [3](#_Toc167476808)

**[CHƯƠNG 1: TỔNG QUAN VỀ XỬ LÝ SONG SONG VÀ PHÂN TÁN](#_Toc167476809)** [5](#_Toc167476809)

**[1.1.](#_Toc167476810)****[Xử lý Song Song](#_Toc167476810)** [5](#_Toc167476810)

**[1.2.](#_Toc167476811)****[Xử lý Phân Tán](#_Toc167476811)** [5](#_Toc167476811)

**[1.3.](#_Toc167476812)****[Các Mô hình Lập trình Song Song](#_Toc167476812)** [6](#_Toc167476812)

**[1.4.](#_Toc167476813)****[Ưu điểm và Thách thức của Xử lý Song Song và Phân Tán](#_Toc167476813)** [7](#_Toc167476813)

**[1.5.](#_Toc167476814)****[Giới thiệu về Hadoop](#_Toc167476814)** [7](#_Toc167476814)

**[CHƯƠNG 2: NGHIÊN CỨU MÔ HÌNH HADOOP](#_Toc167476815)** [9](#_Toc167476815)

**[2.1.](#_Toc167476816)****[Giới thiệu về Hadoop](#_Toc167476816)** [9](#_Toc167476816)

**[2.2.](#_Toc167476817)****[Kiến trúc của HDFS](#_Toc167476817)** [10](#_Toc167476817)

**[2.3.](#_Toc167476818)****[Xử lý dữ liệu lớn với MapReduce](#_Toc167476818)** [10](#_Toc167476818)

**[2.4.](#_Toc167476819)****[Nguyên tắc hoạt động của MapReduce](#_Toc167476819)** [11](#_Toc167476819)

**[2.5.](#_Toc167476820)****[Cài đặt Hadoop](#_Toc167476820)** [11](#_Toc167476820)

**[2.5.1.](#_Toc167476821)****[Cài đặt JDK](#_Toc167476821)** [11](#_Toc167476821)

**[2.5.2.](#_Toc167476822)****[Cài đặt Hadoop](#_Toc167476822)** [19](#_Toc167476822)

**[2.5.3.](#_Toc167476823)****[Cấu hình các tập tin cho Hadoop](#_Toc167476823)** [20](#_Toc167476823)

**[2.5.4.](#_Toc167476824)****[Hoàn thành và chạy thử nghiệm](#_Toc167476824)** [24](#_Toc167476824)

**[CHƯƠNG 3: BÀI TOÁN TÌM ĐƯỜNG ĐI TRÊN](#_Toc167476825)** [26](#_Toc167476825)

**[CẤU TRÚC MAPREDUCE](#_Toc167476826)** [26](#_Toc167476826)

**[3.1.](#_Toc167476827)****[Cấu hình các tập tin cho Hadoop](#_Toc167476827)** [26](#_Toc167476827)

**[3.1.1.](#_Toc167476828)****[Thuật toán Dijkstra tìm đường đi ngắn nhất từ 1 đỉnh đến tất cả các đỉnh](#_Toc167476828)** [26](#_Toc167476828)

**[3.1.2.](#_Toc167476829)****[Ví dụ đồ thị cụ thể](#_Toc167476829)** [26](#_Toc167476829)

**[3.2.](#_Toc167476830)****[Thuật toán tìm đường đi ngắn nhất từ 1 đỉnh đến tất cả các đỉnh trên MapReduce](#_Toc167476830)** [28](#_Toc167476830)

**[3.2.1.](#_Toc167476831)****[Thuật toán trên Map](#_Toc167476831)** [28](#_Toc167476831)

**[3.2.2.](#_Toc167476832)****[Thuật toán trên Reduce](#_Toc167476832)** [30](#_Toc167476832)

**[3.2.3.](#_Toc167476833)****[Ví dụ đồ thị cụ thể cho thuật toán Map và Reduce ở ví dụ được để cập ở phần 3.1.2](#_Toc167476833)** [31](#_Toc167476833)

**[CHƯƠNG 4: XÂY DỰNG DỮ LIỆU VÀ DEMO](#_Toc167476834)** [32](#_Toc167476834)

**[4.1. Demo kết quả ở ví dụ của đồ thị đã trình bày 3.2.3](#_Toc167476835)** [32](#_Toc167476835)

**[4.2. Xây dựng thuật toán tạo đồ thị ngẫu nhiên](#_Toc167476836)** [34](#_Toc167476836)

**[4.3. Demo kết quả đồ thị ngẫu nhiên](#_Toc167476837)** [35](#_Toc167476837)

**[4.3.1. Tạo đồ thị ngẫu nhiên](#_Toc167476838)** [35](#_Toc167476838)

**[4.3.2. Chạy demo trên hadoop](#_Toc167476839)** [35](#_Toc167476839)

**[KẾT LUẬN](#_Toc167476840)** [36](#_Toc167476840)

**[TÀI LIỆU THAM KHẢO](#_Toc167476841)** [37](#_Toc167476841)

# **MỞ ĐẦU**

1. **Lý do chọn đề tài**

**Tầm quan trọng của dữ liệu lớn:** Trong thời đại số hiện nay, dữ liệu được tạo ra với tốc độ chóng mặt từ nhiều nguồn khác nhau như mạng xã hội, thiết bị di động, cảm biến, giao dịch thương mại điện tử,... Việc xử lý và khai thác hiệu quả lượng dữ liệu khổng lồ này mang lại giá trị to lớn cho các doanh nghiệp và tổ chức, giúp họ đưa ra quyết định sáng suốt, cải thiện sản phẩm và dịch vụ, tối ưu hóa quy trình kinh doanh.

**Hadoop - Công cụ xử lý dữ liệu lớn mạnh mẽ:** Hadoop nổi lên như một giải pháp hàng đầu cho việc xử lý dữ liệu lớn nhờ tính mở rộng, khả năng chịu lỗi và chi phí hợp lý. Kiến trúc phân tán của Hadoop cho phép xử lý song song trên nhiều máy tính, giúp tăng tốc độ và khả năng xử lý lượng dữ liệu khổng lồ.

**Ứng dụng của thuật toán tìm đường đi ngắn nhất**: Thuật toán tìm đường đi ngắn nhất, đặc biệt là thuật toán Dijkstra, có ứng dụng rộng rãi trong nhiều lĩnh vực như:

* **Hệ thống định vị GPS:** Tìm đường đi ngắn nhất giữa hai địa điểm.
* **Mạng máy tính:** Tối ưu hóa việc định tuyến gói tin.
* **Lập kế hoạch vận tải:** Tìm lộ trình tối ưu cho việc vận chuyển hàng hóa.
* **Phân tích mạng xã hội:** Phân tích mối quan hệ giữa các cá nhân.

**Kết hợp sức mạnh:** Việc kết hợp sức mạnh của Hadoop và thuật toán tìm đường đi ngắn nhất mở ra khả năng giải quyết các bài toán tìm đường đi trên quy mô lớn, với lượng dữ liệu khổng lồ mà các phương pháp truyền thống khó có thể đáp ứng.

* 1. **Mục tiêu của đề tài**
* Nghiên cứu chuyên sâu: Tìm hiểu sâu về mô hình lập trình MapReduce, một trong những thành phần cốt lõi của Hadoop, và cách thức hoạt động của nó trong việc xử lý dữ liệu phân tán.
* Áp dụng thực tế: Áp dụng kiến thức về MapReduce để giải quyết một bài toán cụ thể là tìm đường đi ngắn nhất trên đồ thị, qua đó chứng minh tính hiệu quả và khả năng ứng dụng của mô hình này.
* Đánh giá hiệu năng: Triển khai thuật toán trên Hadoop và thực hiện các thử nghiệm để đánh giá hiệu năng, khả năng mở rộng và so sánh với các phương pháp khác.
* Đóng góp kiến thức: Đề tài này không chỉ giúp người thực hiện nắm vững kiến thức về Hadoop và MapReduce mà còn đóng góp vào việc mở rộng ứng dụng của mô hình này trong việc giải quyết các bài toán thực tế khác.
  1. **Đối tượng và phạm vi nghiên cứu**
     1. **Đối tượng:**
* Mô hình lập trình MapReduce
* Nền tảng Hadoop (HDFS, YARN)
* Thuật toán tìm đường đi ngắn nhất Dijkstra
* Đồ thị (có hướng, có trọng số)
  + 1. **Phạm vi:**
* Nghiên cứu lý thuyết về MapReduce, Hadoop và thuật toán Dijkstra.
* Cài đặt và cấu hình môi trường Hadoop.
* Triển khai thuật toán tìm đường đi ngắn nhất trên Hadoop bằng MapReduce.
* Thực hiện các thử nghiệm và đánh giá hiệu năng.

* 1. **Cách tiếp cận**
* **Nghiên cứu lý thuyết:** Tìm hiểu sâu về các khái niệm, nguyên lý hoạt động và các thành phần của MapReduce, Hadoop và thuật toán Dijkstra.
* **Cài đặt và cấu hình:** Cài đặt và cấu hình môi trường Hadoop trên một hoặc nhiều máy tính.
* **Triển khai thuật toán:** Thiết kế và triển khai thuật toán tìm đường đi ngắn nhất trên Hadoop bằng MapReduce.
* **Thử nghiệm và đánh giá:** Chuẩn bị dữ liệu đầu vào, thực hiện các thử nghiệm và đánh giá hiệu năng của thuật toán trên các tập dữ liệu khác nhau.
  1. **Phương pháp nghiên cứu**
* **Nghiên cứu tài liệu:** Đọc sách, bài báo khoa học, tài liệu hướng dẫn về Hadoop, MapReduce và thuật toán Dijkstra.
* **Nghiên cứu mã nguồn mở:** Tham khảo các dự án mã nguồn mở liên quan trên GitHub hoặc các nền tảng khác để hiểu rõ hơn về cách triển khai.
* **Thực hành:** Cài đặt và thực hiện các thử nghiệm trên môi trường Hadoop để kiểm chứng lý thuyết và đánh giá hiệu quả.

1. **Nội dung của đề tài**

* Giới thiệu tổng quan về xử lý song song và phân tán, cũng như các khái niệm cơ bản về dữ liệu lớn và Hadoop.
* Đi sâu vào mô hình MapReduce, giải thích chi tiết về cách thức hoạt động, các thành phần và cách thức triển khai trên Hadoop.
* Trình bày chi tiết về thuật toán Dijkstra, từ ý tưởng đến các bước thực hiện và minh họa bằng ví dụ cụ thể.
* Hướng dẫn chi tiết cách cài đặt và cấu hình Hadoop để chuẩn bị cho việc triển khai thuật toán.
* Mô tả cách thức áp dụng MapReduce để giải quyết bài toán tìm đường đi ngắn nhất, bao gồm cả việc thiết kế các hàm Map và Reduce.
* Thực hiện demo trên một đồ thị cụ thể để minh họa cách thức hoạt động của thuật toán trên Hadoop.
* Xây dựng thuật toán tạo đồ thị ngẫu nhiên để kiểm tra tính hiệu quả của thuật toán trên các loại đồ thị khác nhau.
* Thực hiện các thử nghiệm trên các tập dữ liệu lớn để đánh giá hiệu năng và khả năng mở rộng của thuật toán.

**CHƯƠNG 1: TỔNG QUAN VỀ XỬ LÝ SONG SONG VÀ PHÂN TÁN**

Trong kỷ nguyên dữ liệu lớn (Big Data), việc xử lý hiệu quả lượng thông tin khổng lồ trở thành một thách thức lớn. Xử lý song song và phân tán nổi lên như những giải pháp quan trọng để giải quyết vấn đề này. Chương này sẽ cung cấp một cái nhìn tổng quan về các khái niệm, mô hình và công cụ liên quan đến xử lý song song và phân tán, đồng thời giới thiệu về Hadoop, một framework mạnh mẽ trong lĩnh vực này.

* 1. **Xử lý Song Song**

Xử lý song song là một mô hình tính toán trong đó nhiều hoạt động hoặc tác vụ được thực hiện đồng thời, nhằm mục đích tăng tốc độ xử lý và giải quyết các bài toán phức tạp một cách hiệu quả. Thay vì thực hiện tuần tự từng bước, xử lý song song cho phép chia nhỏ công việc thành các phần nhỏ hơn và thực hiện chúng đồng thời trên nhiều bộ xử lý hoặc máy tính khác nhau.

**Các cấp độ song song:**

* **Song song cấp lệnh (Instruction-level Parallelism - ILP):** Ở cấp độ này, nhiều lệnh trong một chương trình được thực thi đồng thời, tận dụng các kỹ thuật như pipelining và superscalar để tăng tốc độ xử lý.
* **Song song cấp dữ liệu (Data-level Parallelism - DLP):** Cùng một thao tác được áp dụng trên nhiều phần tử dữ liệu cùng lúc, thường được sử dụng trong các ứng dụng xử lý mảng, ma trận hoặc hình ảnh.
* **Song song cấp tác vụ (Task-level Parallelism - TLP):** Các tác vụ hoặc tiến trình độc lập được thực thi đồng thời trên các bộ xử lý khác nhau.
  1. **Xử lý Phân Tán**

Xử lý phân tán là một mô hình tính toán trong đó các thành phần của một hệ thống được phân bố trên nhiều máy tính khác nhau, kết nối với nhau thông qua mạng. Mục tiêu của xử lý phân tán là tận dụng tài nguyên của nhiều máy tính để tăng cường khả năng tính toán, lưu trữ và xử lý dữ liệu.

**Các mô hình phân tán:**

* **Phân tán tập trung (Centralized Distributed System):** Trong mô hình này, một máy chủ trung tâm đóng vai trò điều khiển và quản lý các máy trạm. Các máy trạm gửi yêu cầu đến máy chủ và nhận kết quả từ máy chủ.
* **Phân tán phi tập trung (Decentralized Distributed System):** Không có máy chủ trung tâm, các máy tính trong hệ thống tương tác với nhau một cách bình đẳng. Mỗi máy tính đều có khả năng xử lý và lưu trữ dữ liệu.
  1. **Các Mô hình Lập trình Song Song**
* **Chia để trị (Divide and Conquer):** Đây là một kỹ thuật cổ điển trong đó một bài toán lớn được chia thành các bài toán con nhỏ hơn và dễ giải quyết hơn. Các bài toán con được giải quyết độc lập, và kết quả của chúng được kết hợp để tạo ra giải pháp cho bài toán ban đầu.
* **Lập trình hướng luồng (Thread-based Programming):** Mô hình này sử dụng các luồng (thread), là các đơn vị thực thi nhỏ nhất trong một tiến trình, để thực hiện các tác vụ đồng thời. Các luồng chia sẻ cùng một không gian địa chỉ và có thể giao tiếp với nhau thông qua các biến dùng chung.
* **Lập trình hướng thông điệp (Message Passing):** Trong mô hình này, các tiến trình giao tiếp với nhau bằng cách gửi và nhận thông điệp qua các kênh truyền thông. Mỗi tiến trình có không gian địa chỉ riêng và không thể truy cập trực tiếp vào bộ nhớ của các tiến trình khác.
  1. **Ưu điểm và Thách thức của Xử lý Song Song và Phân Tán**

**Ưu điểm:**

* **Tăng tốc độ xử lý:** Tận dụng sức mạnh của nhiều bộ xử lý để thực hiện các tác vụ đồng thời, giúp giảm thời gian xử lý.
* **Khả năng mở rộng:** Dễ dàng thêm tài nguyên tính toán (bộ xử lý, bộ nhớ) để đáp ứng nhu cầu xử lý ngày càng tăng.
* **Tính chịu lỗi:** Nếu một thành phần trong hệ thống bị lỗi, các thành phần khác vẫn có thể tiếp tục hoạt động, đảm bảo tính liên tục của hệ thống.
* **Giải quyết bài toán lớn:** Cho phép xử lý các bài toán có dữ liệu quá lớn không thể xử lý trên một máy tính đơn lẻ.

**Thách thức:**

* **Độ phức tạp:** Việc thiết kế và lập trình các ứng dụng song song và phân tán thường phức tạp hơn so với các ứng dụng tuần tự, đòi hỏi phải quản lý đồng bộ, truyền thông và xử lý lỗi giữa các thành phần.
* **Cân bằng tải:** Đảm bảo rằng công việc được phân phối đều giữa các bộ xử lý để tránh tình trạng một số bộ xử lý bị quá tải trong khi các bộ xử lý khác lại nhàn rỗi.
* **Hiệu suất:** Đôi khi việc chia nhỏ công việc và xử lý song song có thể dẫn đến chi phí liên lạc và đồng bộ hóa giữa các thành phần, làm giảm hiệu suất tổng thể của hệ thống.
  1. **Giới thiệu về Hadoop**

Hadoop là một framework mã nguồn mở được phát triển bởi Apache Software Foundation, được thiết kế để lưu trữ và xử lý dữ liệu lớn trên các cụm máy tính phân tán. Hadoop cung cấp một nền tảng đáng tin cậy, mở rộng và hiệu quả về chi phí để xử lý các bài toán dữ liệu lớn trong nhiều lĩnh vực khác nhau.

**Các thành phần chính của Hadoop:**

* **HDFS (Hadoop Distributed File System):** Một hệ thống tập tin phân tán được tối ưu hóa để lưu trữ dữ liệu lớn trên các cụm máy tính. HDFS chia nhỏ dữ liệu thành các khối và phân phối chúng trên nhiều máy tính, đảm bảo tính khả dụng và chịu lỗi cao.
* **YARN (Yet Another Resource Negotiator):** Một thành phần quản lý tài nguyên và lập lịch các tác vụ trên cụm Hadoop. YARN cho phép chạy nhiều ứng dụng khác nhau trên cùng một cụm Hadoop và đảm bảo việc sử dụng tài nguyên một cách hiệu quả.
* **MapReduce:** Một mô hình lập trình song song và phân tán để xử lý dữ liệu lớn. MapReduce chia nhỏ công việc thành các tác vụ nhỏ hơn (map) và sau đó tổng hợp kết quả (reduce) để tạo ra kết quả cuối cùng.

**CHƯƠNG 2: NGHIÊN CỨU MÔ HÌNH HADOOP**

* 1. **Giới thiệu về Hadoop**

Hadoop là một framework mã nguồn mở được phát triển bởi Apache Software Foundation, cho phép lưu trữ và xử lý dữ liệu lớn trên các cụm máy tính phân tán. Hadoop được thiết kế để giải quyết các bài toán dữ liệu lớn mà các hệ thống truyền thống không thể đáp ứng được về khả năng mở rộng, tính chịu lỗi và hiệu suất.

* **HDFS (Hadoop Distributed File System):** Hệ thống tập tin phân tán của Hadoop cho phép lưu trữ dữ liệu lớn trên nhiều máy tính khác nhau. Dữ liệu được chia thành các khối và phân phối trên các node trong cụm, đảm bảo tính khả dụng và chịu lỗi cao. HDFS còn hỗ trợ các tính năng như sao chép dữ liệu, cân bằng tải và phục hồi dữ liệu khi có sự cố.
* **YARN (Yet Another Resource Negotiator):** Đây là thành phần quản lý tài nguyên và lập lịch các ứng dụng trên cụm Hadoop. YARN chịu trách nhiệm phân bổ tài nguyên (CPU, bộ nhớ) cho các ứng dụng, theo dõi tiến trình thực thi và đảm bảo việc sử dụng tài nguyên một cách hiệu quả.
* **MapReduce:** Đây là mô hình lập trình cốt lõi của Hadoop, cho phép xử lý dữ liệu lớn một cách song song và phân tán. MapReduce hoạt động theo hai giai đoạn chính là Map và Reduce. Giai đoạn Map chia nhỏ dữ liệu đầu vào thành các cặp key-value và áp dụng một hàm xử lý (Mapper) lên từng cặp. Giai đoạn Reduce tổng hợp các kết quả từ giai đoạn Map theo key và áp dụng một hàm xử lý khác (Reducer) để tạo ra kết quả cuối cùng.

**Lịch sử phát triển:**

Hadoop được lấy cảm hứng từ các bài báo nghiên cứu của Google về Google File System (GFS) và MapReduce. Doug Cutting và Mike Cafarella đã khởi tạo dự án Hadoop vào năm 2005 và sau đó được Apache Software Foundation tiếp quản và phát triển thành một dự án mã nguồn mở.

* 1. **Kiến trúc của HDFS**

HDFS là một hệ thống tập tin phân tán được thiết kế để lưu trữ dữ liệu lớn trên các cụm máy tính. Dữ liệu được chia thành các khối (block) và lưu trữ trên các DataNode. Một NameNode đóng vai trò quản lý metadata của hệ thống, bao gồm thông tin về vị trí của các khối dữ liệu, tên tập tin, thư mục, quyền truy cập,...

**Các thành phần chính của HDFS:**

* **NameNode:** Lưu trữ metadata của hệ thống tập tin, bao gồm cấu trúc thư mục, tên tập tin, quyền truy cập và vị trí của các khối dữ liệu trên các DataNode.
* **DataNode:** Lưu trữ các khối dữ liệu thực tế. Mỗi DataNode chịu trách nhiệm quản lý các khối dữ liệu trên máy tính của nó và phục vụ các yêu cầu đọc/ghi dữ liệu từ client.
  1. **Xử lý dữ liệu lớn với MapReduce**

MapReduce là một mô hình lập trình song song và phân tán được sử dụng để xử lý dữ liệu lớn trên Hadoop. Mô hình này chia công việc thành hai giai đoạn chính là Map và Reduce.

**Giai đoạn Map:**

1. Dữ liệu đầu vào được chia thành các phần nhỏ và phân phối đến các Mapper.
2. Mỗi Mapper đọc một phần dữ liệu, xử lý và tạo ra các cặp key-value trung gian.
3. Các cặp key-value trung gian được sắp xếp theo key.

**Giai đoạn Reduce:**

1. Các cặp key-value trung gian được nhóm theo key và gửi đến các Reducer.
2. Mỗi Reducer nhận một tập hợp các giá trị có cùng key, xử lý và tạo ra kết quả cuối cùng.
   1. **Nguyên tắc hoạt động của MapReduce**

MapReduce hoạt động theo nguyên tắc chia để trị (divide and conquer), trong đó một bài toán lớn được chia thành các bài toán con nhỏ hơn và giải quyết độc lập.

* **Chia nhỏ dữ liệu:** Dữ liệu đầu vào được chia thành các phần nhỏ và phân phối đến các Mapper.
* **Xử lý song song:** Các Mapper xử lý các phần dữ liệu một cách độc lập và đồng thời.
* **Tổng hợp kết quả:** Các kết quả trung gian từ các Mapper được tổng hợp và gửi đến các Reducer.
* **Kết hợp kết quả:** Các Reducer kết hợp các kết quả trung gian để tạo ra kết quả cuối cùng.
  1. **Cài đặt Hadoop**
     1. **Cài đặt JDK**

Hadoop yêu cầu Java Development Kit (JDK) để hoạt động. Các bước cài đặt JDK bao gồm:

* 1. Tải xuống JDK từ trang chủ của Oracle.

Ví dụ cài bản Java SE Development Kit 8u202:

[A screenshot of a computer

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-2.png)

Bấm vào link để tải, chương trình xuất hiện như bên dưới:

[A screenshot of a computer

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-3.png)

Tick vào “I reviewed and accept the Oracle….”

Rồi bấm download

Chương trình yêu cầu đăng nhập:

[A screenshot of a computer

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-4.png)

Nếu chưa có tài khoản thì cứ đăng ký “Create Account”

* 1. Cài đặt JDK theo hướng dẫn.
* Tiến hành cài đặt:

[A screenshot of a computer

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-7.png)

Nhấn Next để cài đặt

[A screenshot of a software update

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-8.png)

Tới chỗ này nhớ chỉnh vào Ổ C, không có dấu và không khoảng trắng

A screenshot of a software update

Description automatically generated

Chọn được nơi cài JDK không có khoảng trắng, nhấn Next để tiếp tục

[A screenshot of a software update

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-11.png)

Chờ chương trình cài đặt hoàn tất

[A screenshot of a computer

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-12.png)

Nếu jre yêu cầu cài thì cũng chỉnh vào ổ C như trên

Bấm Next

và Tiếp tục chờ

[A screenshot of a computer

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-13.png)

Khi xuất hiện màn hình dưới đây tức là đã hoàn tất quá trình cài đặt JDK

[A screenshot of a software update

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-14.png)

Bấm Close để hoàn tất

Như vậy đã cài đặt xong JDK 1.8

* 1. Cấu hình biến môi trường JAVA\_HOME để chỉ đến thư mục cài đặt JDK.
* Vào Start menu, search “Edit the system enviroment”

A grey background with white text

Description automatically generated

* HIện cửa sổ như bên dưới, chọn “Environment Variables…”

A screenshot of a computer program

Description automatically generated

* Ở mục “User variables for..” chọn vào mục “New”

A screenshot of a computer

Description automatically generated

* Hiện cửa số, ta đặt tên như bên dưới hình, Nhấn OK.

A screenshot of a computer

Description automatically generated

* Ở mục “System variables” chọn mục “Path” và nhấn “Edit”.

A screenshot of a computer program

Description automatically generated

* Chọn New và thêm các đường dẫn sau “**C:\Java\jdk\bin”**

A screenshot of a computer program

Description automatically generated

* + 1. **Cài đặt Hadoop**

Các bước cài đặt Hadoop bao gồm:

* Tải xuống phiên bản Hadoop từ trang chủ của Apache Hadoop ở link sau:

<https://mirror.downloadvn.com/apache/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz>

* Giải nén gói Hadoop vừa tải xuống vào ổ C.

[A screenshot of a computer

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-26.png)

* Cấu hình các tập tin cần thiết để Hadoop hoạt động.

Tương tự với các bước setup JDK, thiết lập các biến và đường dẫn mới cho Hadoop

A screenshot of a computer

Description automatically generated

* Ở mục “System variables” chọn mục “Path” và nhấn “Edit”.

A screenshot of a computer program

Description automatically generated

* Chọn New và thêm các đường dẫn sau “**C:\hadoop\bin”** và “**C:\hadoop\sbin”**

A screenshot of a computer program

Description automatically generated

* + 1. **Cấu hình các tập tin cho Hadoop**

Các tập tin cấu hình chính của Hadoop bao gồm:

* core-site.xml: Cấu hình các thiết lập chung cho Hadoop.
* hdfs-site.xml: Cấu hình các thiết lập cho HDFS.
* mapred-site.xml: Cấu hình các thiết lập cho MapReduce.
* yarn-site.xml: Cấu hình các thiết lập cho YARN (Yet Another Resource Negotiator).
* hadoop-env.cmd

1. Cấu hình **core-site.xml**như dưới đây:

*<configuration>*

*<property>*

*<name>fs.defaultFS</name>*

*<value>hdfs://localhost:9000</value>*

*</property>*

*</configuration>*

1. Cấu hình **mapred-site.xml**như dưới đây:

*<configuration>*

*<property>*

*<name>mapreduce.framework.name</name>*

*<value>yarn</value>*

*</property>*

*</configuration>*

1. Cấu hình **hdfs-site.xml** như dưới đây:

A screenshot of a computer

Description automatically generated

* Tạo thư mục “data” trong “C:/hadoop”
* Tạo thư mục con “datanode” trong “C:/hadoop/data”
* Tạo thư mục con “namenode” trong “C:/hadoop/data”

1. Sau đó cấu hình **hdfs-site.xml**như sau:

*<configuration>*

*<property>*

*<name>dfs.replication</name>*

*<value>1</value>*

*</property>*

*<property>*

*<name>dfs.namenode.name.dir</name>*

*<value>C:\hadoop\data\namenode</value>*

*</property>*

*<property>*

*<name>dfs.datanode.data.dir</name>*

*<value> C:\hadoop\data\ datanode</value>*

*</property>*

*</configuration>*

1. Cấu hình **yarn-site.xml**như dưới đây:

*<configuration>*

*<property>*

*<name>yarn.nodemanager.aux-services</name>*

*<value>mapreduce\_shuffle</value>*

*</property>*

*<property>*

*<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>*

*<value>org.apache.hadoop.mapred.ShuffleHandler</value>*

*</property>*

*</configuration>*

1. Cấu hình **hadoop-env.cmd**:

Mở file này lên và tìm tới lệnh:

* set JAVA\_HOME=%JAVA\_HOME%

sửa %JAVA\_HOME% thành đường dẫn cài JDK trong ổ C:

* set JAVA\_HOME= C:\Java\jdk

Sau đó format lại **namenode và datanode**: mở command line lên, gõ lệnh sau:

* **hdfs namenode –format**
* **hdfs datanode -format**

[A screenshot of a computer program

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-33.png)

Bước format này chỉ cần làm 1 lần.

* + 1. **Hoàn thành và chạy thử nghiệm**

A screen shot of a computer

Description automatically generated

Sau khi gõ lệnh trên, hệ thống sẽ chạy Hadoop

Phải đảm bảo các ứng dụng sau được chạy:

– Hadoop Namenode

– Hadoop datanode

– YARN Resource Manager

– YARN Node Manager

[A screenshot of a computer program

Description automatically generated](https://duythanhcse.files.wordpress.com/2021/01/hadoop-35.png)

Như vậy là ta đã khởi chạy thành công Hadoop.

**CHƯƠNG 3: BÀI TOÁN TÌM ĐƯỜNG ĐI TRÊN**

**CẤU TRÚC MAPREDUCE**

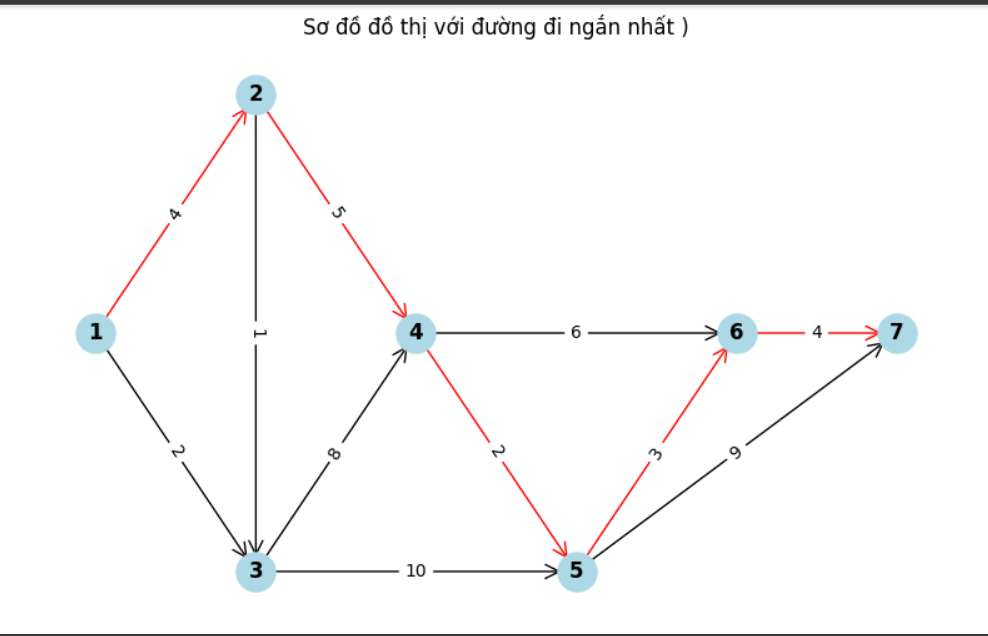
* 1. **Cấu hình các tập tin cho Hadoop**
     1. **Thuật toán Dijkstra tìm đường đi ngắn nhất từ 1 đỉnh đến tất cả các đỉnh**

Thuật toán Dijkstra được sử dụng để tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại trong đồ thị với các cạnh không âm. Thuật toán sử dụng một tập hợp các đỉnh đã biết khoảng cách ngắn nhất và mở rộng nó cho đến khi bao gồm tất cả các đỉnh.

Ý tưởng cơ bản của thuật toán như sau:

* *Bước 1*: Từ đỉnh gốc, khởi tạo khoảng cách tới chính nó là 00, khởi tạo khoảng cách nhỏ nhất ban đầu tới các đỉnh khác là +∞. Ta được danh sách các khoảng cách tới các đỉnh.
* *Bước 2*: Chọn đỉnh a có khoảng cách nhỏ nhất trong danh sách này và ghi nhận. Các lần sau sẽ không xét tới đỉnh này nữa.
* *Bước 3*: Lần lượt xét các đỉnh kề b của đỉnh a. Nếu *khoảng cách từ đỉnh gốc* tới đỉnh b nhỏ hơn khoảng cách hiện tại đang được ghi nhận thì cập nhật giá trị và đỉnh kề a vào khoảng cách hiện tại của b.
* *Bước 4*: Sau khi xét tất cả đỉnh kề b của đỉnh a. Lúc này ta được danh sách khoảng cách tới các điểm đã được cập nhật. Quay lại *Bước 2* với danh sách này. Thuật toán kết thúc khi chọn được khoảng cách nhỏ nhất từ tất cả các điểm.
  + 1. **Ví dụ đồ thị cụ thể**

Để dễ dàng hiểu ý tưởng của thuật toán. Chúng ta cùng xem ví dụ với đồ thị vô hướng *G*. Thuật toán Dijkstra sẽ tìm đường đi ngắn nhất từ đỉnh 1 tới đỉnh 7 trong đồ thị 𝐺*.*



Giải thuật:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **k** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| 0 | {0,1}\* | {,1} | {,1} | {,1} | {,1} | {,1} | {,1} |
| 1 | - | {4,1} | {2,1}\* | {,1} | {,1} | {,1} | {,1} |
| 2 | - | {4,1}\* | - | {10,3} | {12,3} | {,1} | {,1} |
| 3 | - | - | - | {9,2}\* | {12,3} | {,1} | {,1} |
| 4 | - | - | - | - | {11,4}\* | {15,4} | {,1} |
| 5 | - | - | - | - | - | {14,5}\* | {20,5} |
| 6 | - | - | - | - | - | - | {18,6}\* |
| 7 | - | - | - | - | - | - | - |

Thuật toán kết thúc khi chọn được khoảng cách nhỏ nhất cho tất cả các đỉnh.

Vậy kết quả đi từ đỉnh 1 đến 7 là: 1 → 2 →4 → 5 → 6 → 7 với trọng số bằng 18.

* 1. **Thuật toán tìm đường đi ngắn nhất từ 1 đỉnh đến tất cả các đỉnh trên MapReduce**
     1. **Thuật toán trên Map**

class AlgorithmMapper extends MapReduceBase implements Mapper<Text, Text, Text, Text>

{

  private static final String MARKED                 = "MARKED";

  private static final String UNMARKED               = "UNMARKED";

  private static final String INFINITY                   = "INFINITY";

  private static final int    MINIMUM\_SOURCE\_NODE\_TOKENS = 3;

  private static final int    MAXIMUM\_SOURCE\_NODE\_TOKENS = 4;

  public static final int     INDEX\_SOURCE\_NODE\_NUMBER   = 0;

  public static final int     INDEX\_SOURCE\_NODE\_WEIGHT   = 1;

  private static final int    INDEX\_SOURCE\_NODE\_STATUS   = 2;

  private static final int    INDEX\_SOURCE\_NODE\_PATH     = 3;

  private Text                emptyText                  = new Text();

  @Override

  public void map(Text key, Text value, OutputCollector<Text, Text> outputMapper, Reporter reporter) throws IOException

  {

    String[] sourceNodeDetails = key**.**toString()**.**substring(1, key**.**toString()**.**length() - 1)**.**split(",");

    if (sourceNodeDetails**.**length >= MINIMUM\_SOURCE\_NODE\_TOKENS

        && !sourceNodeDetails[INDEX\_SOURCE\_NODE\_WEIGHT]**.**equalsIgnoreCase(INFINITY)

        && !sourceNodeDetails[INDEX\_SOURCE\_NODE\_STATUS]**.**equalsIgnoreCase(MARKED))

    {

      String currentPath = "";

      if (sourceNodeDetails**.**length == MAXIMUM\_SOURCE\_NODE\_TOKENS)

      {

        currentPath = sourceNodeDetails[INDEX\_SOURCE\_NODE\_PATH];

      }

      currentPath += (currentPath**.**length() == 0 ? "" : "-") + sourceNodeDetails[INDEX\_SOURCE\_NODE\_NUMBER];

      outputMapper**.**collect(new Text("{" + sourceNodeDetails[INDEX\_SOURCE\_NODE\_NUMBER] + "," + sourceNodeDetails[INDEX\_SOURCE\_NODE\_WEIGHT] + "," + MARKED + "," + currentPath + "}"), value);

      System**.**out**.**println(">>> MAP1: Key = [" + new Text("{" + sourceNodeDetails[INDEX\_SOURCE\_NODE\_NUMBER] + "," + sourceNodeDetails[INDEX\_SOURCE\_NODE\_WEIGHT] + "," + MARKED + "," + currentPath + "}") + "] Value = [" + value +"]");

      if (value**.**toString()**.**trim()**.**length() > 0)

      {

        String[] tokens = value**.**toString()**.**trim()**.**split("\t");

        String[][] adjacentNodeDetails = new String[tokens**.**length][2];

        for (int index = 0; index < tokens**.**length; index++)

        {

          adjacentNodeDetails[index] = tokens[index]**.**substring(1, tokens[index]**.**length() - 1)**.**split(",");

        }

        int sourceNodeWeight = Integer**.**parseInt(sourceNodeDetails[INDEX\_SOURCE\_NODE\_WEIGHT]);

        for (int index = 0; index < tokens**.**length; index++)

        {

          int number = sourceNodeWeight + Integer**.**parseInt(adjacentNodeDetails[index][1]);

          outputMapper**.**collect(new Text("{" + adjacentNodeDetails[index][0] + "," + Integer**.**toString(number) + "," + UNMARKED + "," + currentPath + "}"), emptyText);

          reporter**.**incrCounter(ShortestPath**.**CUSTOM\_COUNTERS, ShortestPath**.**NUMBER\_OF\_UNMARKED\_NODES\_TO\_BE\_PROCESSED, 1);

          System**.**out**.**println(">>> MAP2: Key = [" + new Text("{" + adjacentNodeDetails[index][0] + "," + Integer**.**toString(number) + "," + UNMARKED + "," + currentPath + "}") + "] Value = [" + emptyText +"]");

        }

      }

    }

    else

    {

      outputMapper**.**collect(key, value);

      System**.**out**.**println(">>> MAP3: Key = [" + key + "] Value = [" + value +"]");

    }

  }

}

* + 1. **Thuật toán trên Reduce**

class AlgorithmReducer extends MapReduceBase implements Reducer<Text, Text, Text, Text>

{

  @Override

  public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text> outputReducer, Reporter reporter)throws IOException

  {

    int index = 0;

    String newValue = "";

    while (values**.**hasNext())

    {

      if (index > 0)

      {

        newValue += "\t";

      }

      newValue += values**.**next()**.**toString();

    }

    outputReducer**.**collect(key, new Text(newValue));

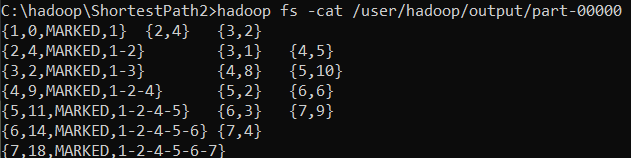
    System**.**out**.**println(">>> REDUCE: Key = [" + key + "], Value = [" + new Text(newValue) + "]");

  }

}

* + 1. **Ví dụ đồ thị cụ thể cho thuật toán Map và Reduce ở ví dụ được để cập ở phần 3.1.2**
* Với đồ thị ở mục 3.1.2 ta có file input.txt tương ứng:
* **{1,0,UNMARKED} {2,4} {3,2}**
* **{2,INFINITY,UNMARKED} {3,1} {4,5}**
* **{3,INFINITY,UNMARKED} {4,8} {5,10}**
* **{4,INFINITY,UNMARKED} {5,2} {6,6}**
* **{5,INFINITY,UNMARKED} {6,3} {7,9}**
* **{6,INFINITY,UNMARKED} {7,4}**

{7,INFINITY,UNMARKED}Kết quả chạy với thuật toán Map và Reduce như sau:



**CHƯƠNG 4: XÂY DỰNG DỮ LIỆU VÀ DEMO**

**4.1. Demo kết quả ở ví dụ của đồ thị đã trình bày 3.2.3**

* Mở cửa sổ lệnh dưới quyền *Administrator.*
* Khởi động các tiến trình hadoop: chạy *start-all.cmd*
* Chuyển vào thư mục hiện hành

cd C:\hadoop\ShortestPath2

* Tạo Project java build với Maven
* Khai báo dependencies
* Biên dịch mã nguồn java thành các file jar

Maven clean ; Maven install

* Chạy chương trình Mapreduce với dữ liệu trong thư mục /user/hadoop/input, kết quả xuất ra /user/hadoop/output

C:\hadoop\ShortestPath2>hadoop jar ShortestPath2-1.0-SNAPSHOT.jar /user/hadoop/input /user/hadoop/output

* Quá trình Map/Reduce sẽ lặp đi lặp lại nhiều lần. Kết quả của Map sẽ là input cho Reduce.
* Xem kết quả trong /user/hadoop/output

C:\hadoop\ShortestPath2>hadoop fs -ls /user/hadoop/output/

C:\hadoop\ShortestPath2>hadoop fs -cat /user/hadoop/output/part-00000

**4.2. Xây dựng thuật toán tạo đồ thị ngẫu nhiên**

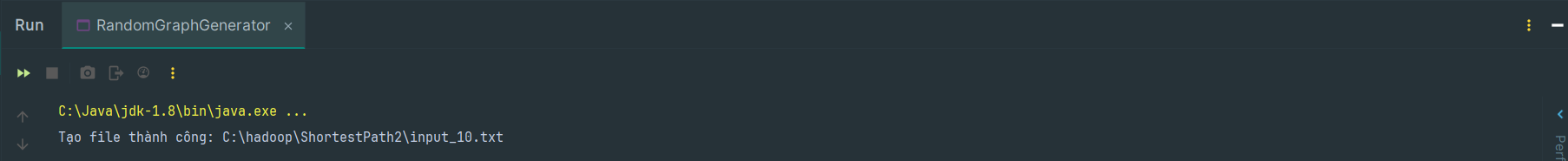
Thuật toán tạo đồ thị với số đỉnh, mật độ cạnh ngẫu nhiên được thiết lập trong file RandomGraphGenerator.java:

package org.code;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.Random;  
public class RandomGraphGenerator {  
 public static void main(String[] args) {  
 Random rand = new Random();  
 int numNodes = rand.nextInt(10) + 5; // 5 đến 14 đỉnh  
 String filePath = "C:\\hadoop\\ShortestPath2\\input\_" + numNodes + ".txt";  
 try (FileWriter writer = new FileWriter(filePath)) {  
 // Đỉnh đầu tiên với trọng số 0  
 writer.write("{" + 1 + ",0,UNMARKED}");  
 int numEdges = rand.nextInt(5) + 1;  
 for (int j = 2; j <= 1 + numEdges && j <= numNodes; j++) {  
 int weight = rand.nextInt(101); // Trọng số từ 0 đến 100  
 writer.write("\t{" + j + "," + weight + "}");  
 }  
 writer.write("\n");  
 // Các đỉnh còn lại  
 for (int i = 2; i <= numNodes; i++) {  
 writer.write("{" + i + ",INFINITY,UNMARKED}");  
 numEdges = rand.nextInt(5) + 1;  
 for (int j = 1; j <= numEdges; j++) {  
 int neighbor = rand.nextInt(numNodes) + 1;  
 int weight = rand.nextInt(101);  
 writer.write("\t{" + neighbor + "," + weight + "}");  
 }  
 writer.write("\n");  
 }  
  
 System.out.println("Tạo file thành công: " + filePath);  
 } catch (IOException e) {  
 System.err.println("Lỗi ghi file: " + e.getMessage());  
 }  
 }  
}

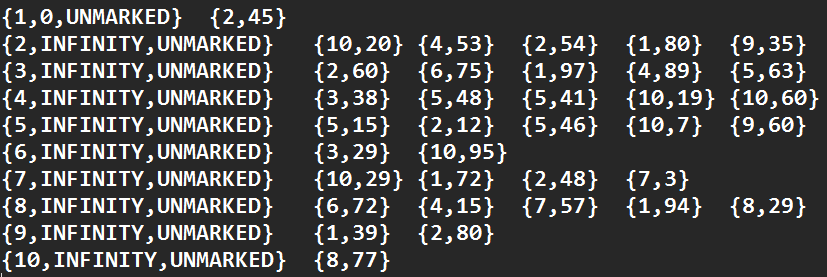
**4.3. Demo kết quả đồ thị ngẫu nhiên**

**4.3.1. Tạo đồ thị ngẫu nhiên**

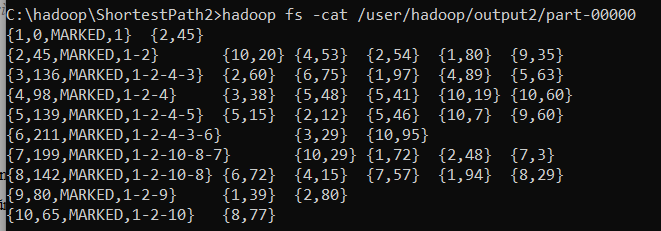
- Chạy file RandomGraphGenerator.java:



- File input.txt có kết quả sau



**4.3.2. Chạy demo trên hadoop**



**Như vậy đối với đồ thị trên, ta có kết quả đường đi ngắn nhất là 1 🡪 2 🡪 10**

# **KẾT LUẬN**

Trong báo cáo này em đã trình bày về lý thuyết, cài đặt và ứng dụng của mô hình lập trình MapReduce trên nền tảng Hadoop để giải quyết bài toán tìm đường đi ngắn nhất. Em cũng đã giới thiệu tổng quan về xử lý song song và phân tán, cũng như các khái niệm cơ bản về dữ liệu lớn và Hadoop. Tiếp theo, chúng tôi đi sâu vào mô hình MapReduce và thuật toán Dijkstra, giải thích chi tiết về cách thức hoạt động, các thành phần và cách triển khai trên Hadoop.

Ngoài ra báo cáo cũng đã hướng dẫn chi tiết cách cài đặt và cấu hình Hadoop, mô tả cách thức áp dụng MapReduce để giải quyết bài toán tìm đường đi ngắn nhất và thực hiện demo trên một đồ thị cụ thể. Kết quả thử nghiệm cho thấy thuật toán có hiệu năng tốt và khả năng mở rộng cao.

Tuy nhiên, vẫn còn một số hạn chế cần được cải thiện trong tương lai như:

* Tối ưu hóa thuật toán để giảm thời gian xử lý và sử dụng tài nguyên hiệu quả hơn.
* Mở rộng thuật toán để giải quyết các bài toán tìm đường đi khác như tìm đường đi ngắn nhất trên đồ thị có hướng, đồ thị có trọng số âm, hoặc tìm đường đi với nhiều ràng buộc.
* Ứng dụng thuật toán vào các bài toán thực tế khác trong lĩnh vực giao thông, vận tải, logistics, mạng xã hội, v.v.

Em hy vọng rằng báo cáo này đã cái nhìn tổng quan về mô hình lập trình MapReduce và cách ứng dụng nó để giải quyết bài toán tìm đường đi ngắn nhất trên nền tảng Hadoop. Và đây là một công cụ mạnh mẽ và hữu ích cho việc xử lý dữ liệu lớn và có thể được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau.

# **TÀI LIỆU THAM KHẢO**

*[1] <https://hadoop.apache.org/>*

*[2] <https://www.oreilly.com/library/view/hadoop-the-definitive/9781449361900/>*

*[3] <https://www.oreilly.com/library/view/mapreduce-design-patterns/9780596521678/>*

*[4] [Find Shortest Paths from Source to all Vertices using Dijkstra’s Algorithm (geeksforgeeks.org)](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)*

*[5] [The MapReduce based approach to improve the shortest path computation | Lau | J. Math. Comput. Sci. (scik.org)](https://scik.org/index.php/jmcs/article/view/8300)*