Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №5**

**Дисциплина: Распределенные задачи и алгоритмы**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Нагалевский

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.И. Шиян

**Тема работы:** Разработать алгоритм определения того, является ли граф гиперкубом.

**Постановка задачи:**

Гиперкуб — это абстрактный математический объект в теории графов и комбинаторике. В контексте теории графов гиперкуб обозначает граф, вершинами которого являются битовые строки фиксированной длины, а ребра соединяют те вершины, которые отличаются по одному биту. Гиперкуб также известен как n-мерный куб.

Давайте рассмотрим пример гиперкуба для случая n = 3. В этом случае гиперкуб будет трехмерным, и каждая вершина будет представлять битовую строку длиной 3 бита (в двоичной системе). Ребра гиперкуба соединяют вершины, которые отличаются ровно в одном бите.

000

/ | \

001-010-110

| | /

011-101

\ |

100

В этом графе вершины представляют битовые строки 000, 001, 010, 011, 100, 101, 110. Ребра соединяют только те вершины, которые отличаются ровно в одном бите. Например, вершины 000 и 001 соединены ребром, потому что они отличаются только в одном бите.

Гиперкубы являются важным объектом изучения в комбинаторике, теории графов и в приложениях в компьютерных науках, так как они представляют собой примеры объектов с фиксированным числом измерений, в которых связи между элементами легко определить.

Теперь рассмотрим подробно алгоритм проверки является ли граф гиперкубом. Алгоритм проверки графа на гиперкуб осуществляется в функции checkIsomorphism (GraphData data):

1. Определение числа вершин графа:

Алгоритм начинается с определения числа вершин в графе. Он проверяет, что количество вершин соответствует степени двойки (), иначе сразу возвращается false, поскольку гиперкубы имеют точное количество вершин, равное степени двойки (рисунок 1).

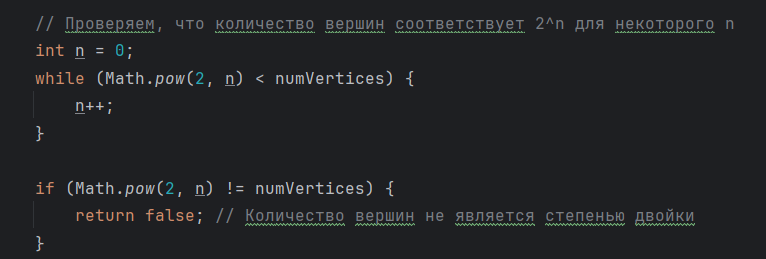


Рисунок 1 – Проверка на степень двойки.

1. Проверка ребер графа:

Далее, алгоритм проверяет, что ребра соединяют вершины, которые отличаются ровно в одном бите. Это осуществляется путем проверки битового XOR (исключающее ИЛИ) между исходной и конечной вершиной каждого ребра. Если количество установленных битов в результате XOR не равно 1, то граф не является гиперкубом (рисунок 2).

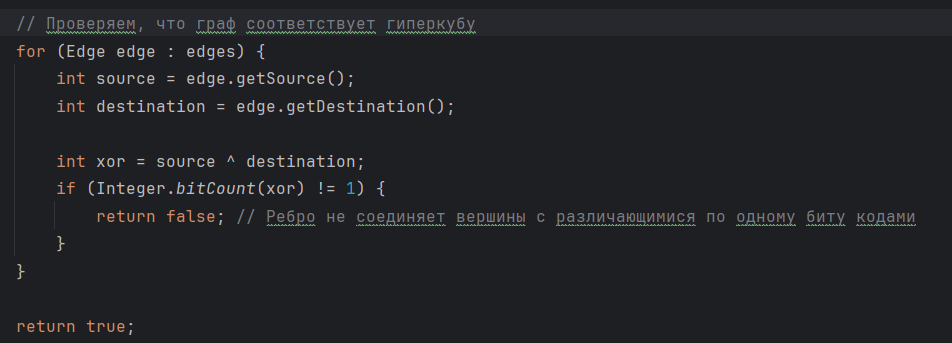


Рисунок 2 – проверка графа на соответствие гиперкубу.

1. Возвращение результата:

Если обе проверки пройдены, функция возвращает true, что означает, что граф является гиперкубом.

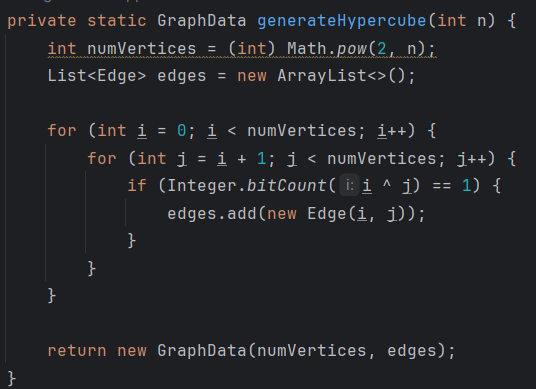
Также была написана функция, которая генерирует граф, являющийся гиперкубом, для проверки правильности работы алгоритма (рисунок 3).  


Рисунок 3 – генерация гиперкуба.

Также были проведены измерения времени работы программы на заданном количестве вершин графа, различном количестве процессов и блокирующим и неблокирующем способом. На рисунке 4 представлен график различия времени, который показывает, что неблокирующий способ выигрывает по времени, чем блокирующий.

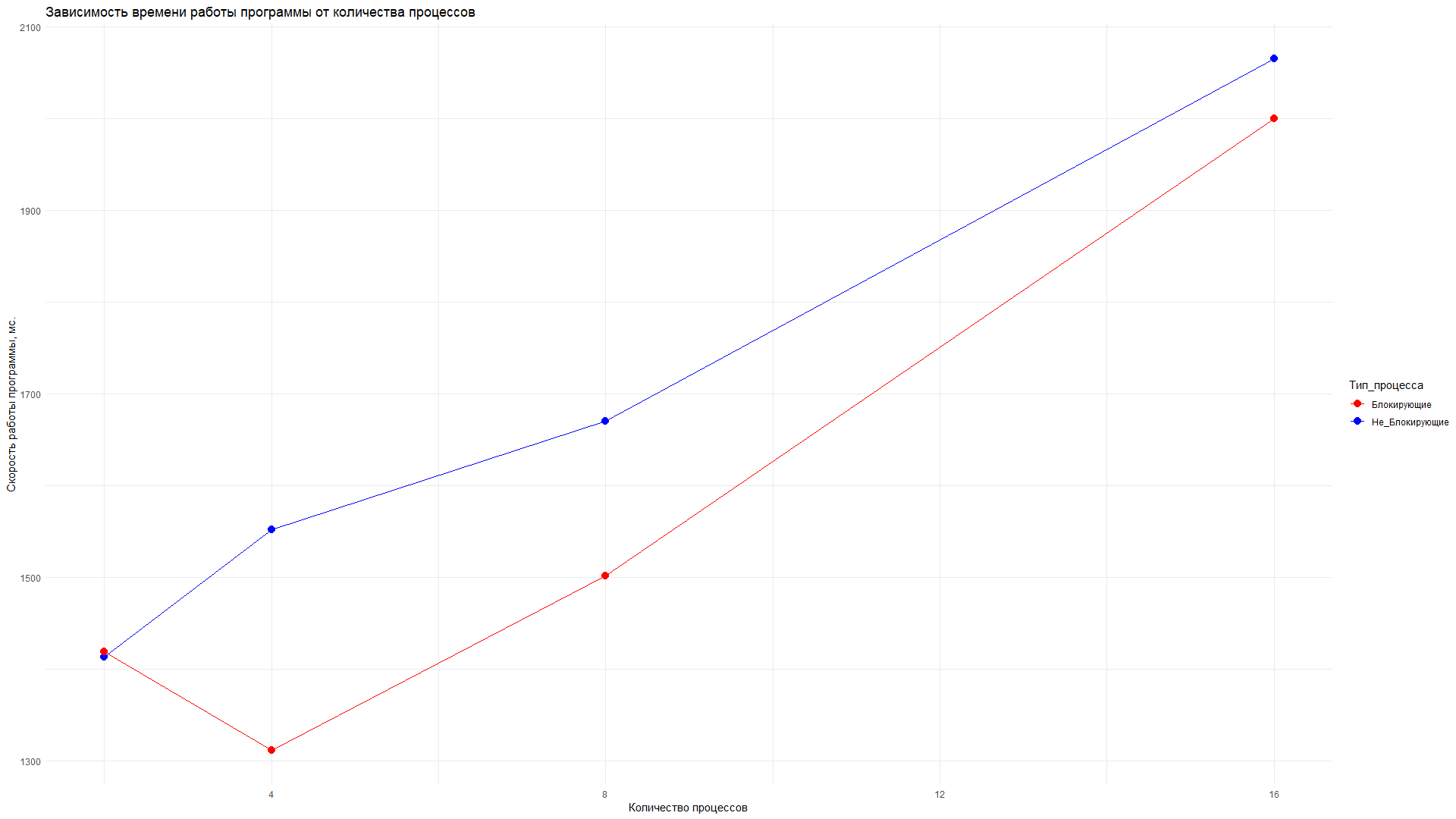


Рисунок 4 – график зависимости времени работы программы от количества процессов

**Листинг программ**

Файл Main.java

import mpi.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Main {  
 public static void main(String[] args) {  
 MPI.*Init*(args);  
  
 int rank = MPI.*COMM\_WORLD*.Rank();  
 int size = MPI.*COMM\_WORLD*.Size();  
  
 if (size < 2) {  
 System.*out*.println("Этот код должен быть запущен как минимум на двух процессах.");  
 MPI.*Finalize*();  
 return;  
 }  
  
 if (rank == 0) {  
 GraphData graphData = *generateGraphData*();  
  
 Status[] sendStatus = new Status[size - 1];  
 Request[] sendRequests = new Request[size - 1];  
  
 GraphData[] dataToSend = new GraphData[]{graphData};  
 for (int i = 1; i < size; i++) {  
 sendRequests[i - 1] = MPI.*COMM\_WORLD*.Isend(dataToSend, 0, 1, MPI.*OBJECT*, i, 0);  
 }  
  
 for (int i = 0; i < size - 1; i++) {  
 sendStatus[i] = sendRequests[i].Wait();  
 }  
 } else {  
 GraphData[] dataReceived = new GraphData[1];  
 MPI.*COMM\_WORLD*.Irecv(dataReceived, 0, 1, MPI.*OBJECT*, 0, 0).Wait();  
 GraphData graphDataRecv = dataReceived[0];  
  
  
 boolean isHypercube = *checkIsomorphism*(graphDataRecv);  
  
 byte[] sendBuffer = new byte[]{(byte) (isHypercube ? 1 : 0)};  
 MPI.*COMM\_WORLD*.Isend(sendBuffer, 0, 1, MPI.*BYTE*, 0, 0);  
  
 }  
  
 if (rank == 0) {  
  
 byte[] recvResults = new byte[size - 1];  
 Status[] recvStatus = new Status[size - 1];  
 Request[] recvRequests = new Request[size - 1];  
  
 for (int i = 1; i < size; i++) {  
 recvRequests[i - 1] = MPI.*COMM\_WORLD*.Irecv(recvResults, i - 1, 1, MPI.*BYTE*, i, 0);  
 }  
  
 for (int i = 0; i < size - 1; i++) {  
 recvStatus[i] = recvRequests[i].Wait();  
 }  
  
 boolean isHypercubeOverall = true;  
 for (byte result : recvResults) {  
 isHypercubeOverall &= (result != 0);  
 }  
  
  
 if (isHypercubeOverall) {  
 System.*out*.println("Граф является гиперкубом.");  
 } else {  
 System.*out*.println("Граф не является гиперкубом.");  
 }  
 }  
  
 MPI.*Finalize*();  
 }  
  
 private static GraphData generateGraphData() {  
 int numVertices = 6;  
 List<Edge> edges = new ArrayList<>();  
 edges.add(new Edge(0, 1));  
 edges.add(new Edge(0, 2));  
 edges.add(new Edge(1, 3));  
 edges.add(new Edge(2, 4));  
 edges.add(new Edge(3, 5));  
 edges.add(new Edge(4, 5));  
  
 //return new GraphData(numVertices, edges);  
 return *generateHypercube*(numVertices);  
 }  
  
 private static boolean checkIsomorphism(GraphData data) {  
 int numVertices = data.getNumVertices();  
 List<Edge> edges = data.getEdges();  
  
   
 int n = 0;  
 while (Math.*pow*(2, n) < numVertices) {  
 n++;  
 }  
  
 if (Math.*pow*(2, n) != numVertices) {  
 return false;  
 }  
  
 for (Edge edge : edges) {  
 int source = edge.getSource();  
 int destination = edge.getDestination();  
  
 int xor = source ^ destination;  
 if (Integer.*bitCount*(xor) != 1) {  
 return false;  
 }  
 }  
  
 return true;  
 }  
  
 private static GraphData generateHypercube(int n) {  
 int numVertices = (int) Math.*pow*(2, n);  
 List<Edge> edges = new ArrayList<>();  
  
 for (int i = 0; i < numVertices; i++) {  
 for (int j = i + 1; j < numVertices; j++) {  
 if (Integer.*bitCount*(i ^ j) == 1) {  
 edges.add(new Edge(i, j));  
 }  
 }  
 }  
  
 return new GraphData(numVertices, edges);  
 }  
  
}

Файл GraphData.java:

import java.io.Serializable;  
import java.util.List;  
  
public class GraphData implements Serializable {  
 private int numVertices;  
 private List<Edge> edges;  
  
 public GraphData(){  
  
 }  
  
 public GraphData(int numVertices, List<Edge> edges) {  
 this.numVertices = numVertices;  
 this.edges = edges;  
 }  
  
 public int getNumVertices() {  
 return numVertices;  
 }  
  
 public List<Edge> getEdges() {  
 return edges;  
 }  
}

Файл Edge.java:

import java.io.Serializable;  
  
public class Edge implements Serializable {  
 private final int source;  
 private final int destination;  
  
 public Edge(int source, int destination) {  
 this.source = source;  
 this.destination = destination;  
 }  
  
 public int getSource() {  
 return source;  
 }  
  
 public int getDestination() {  
 return destination;  
 }  
}

Файл Nonblocking.java:

import mpi.\*;  
import java.util.ArrayList;  
import java.util.List;  
public class Nonblocking {  
 public static void main(String[] args) {  
 MPI.*Init*(args);  
  
 int rank = MPI.*COMM\_WORLD*.Rank();  
 int size = MPI.*COMM\_WORLD*.Size();  
  
 if (size < 2) {  
 System.*out*.println("This code should be run with at least two processes.");  
 MPI.*Finalize*();  
 return;  
 }  
  
 long startTime, endTime;  
 startTime = System.*currentTimeMillis*();  
  
 if (rank == 0) {  
 // Generate the graph data on process with rank 0.  
 GraphData graphData = *generateGraphData*();  
  
 // Send data to all processes using blocking communication.  
 GraphData[] dataToSend = new GraphData[]{graphData};  
 for (int i = 1; i < size; i++) {  
 MPI.*COMM\_WORLD*.Send(dataToSend, 0, 1, MPI.*OBJECT*, i, 0);  
 }  
 } else {  
 // Processes with rank 1 and above receive data.  
 GraphData[] dataReceived = new GraphData[1];  
 MPI.*COMM\_WORLD*.Recv(dataReceived, 0, 1, MPI.*OBJECT*, 0, 0);  
 GraphData graphDataRecv = dataReceived[0];  
  
 boolean isHypercube = *checkIsomorphism*(graphDataRecv);  
  
 // Send the result to process with rank 0.  
 byte[] sendBuffer = new byte[]{(byte) (isHypercube ? 1 : 0)};  
 MPI.*COMM\_WORLD*.Send(sendBuffer, 0, 1, MPI.*BYTE*, 0, 0);  
 }  
  
 if (rank == 0) {  
 // Process with rank 0 collects the results.  
 byte[] recvResults = new byte[size - 1];  
 for (int i = 1; i < size; i++) {  
 MPI.*COMM\_WORLD*.Recv(recvResults, i - 1, 1, MPI.*BYTE*, i, 0);  
 }  
  
 boolean isHypercubeOverall = true;  
 for (byte result : recvResults) {  
 isHypercubeOverall &= (result != 0);  
 }  
  
 endTime = System.*currentTimeMillis*();  
  
 if (isHypercubeOverall) {  
 System.*out*.println("The graph is a hypercube.");  
 } else {  
 System.*out*.println("The graph is not a hypercube.");  
 }  
  
 System.*out*.println("Program completed in " + (endTime - startTime) + " ms with " + size + " processes.");  
 }  
  
 MPI.*Finalize*();  
 }  
  
 private static GraphData generateGraphData() {  
 int numVertices = 10;  
 return *generateHypercube*(numVertices);  
 }  
  
 // функция для проверки изоморфизма гиперкубов  
 private static boolean checkIsomorphism(GraphData data) {  
 int numVertices = data.getNumVertices();  
 List<Edge> edges = data.getEdges();  
  
 // Проверяем, что количество вершин соответствует 2^n для некоторого n  
 int n = 0;  
 while (Math.*pow*(2, n) < numVertices) {  
 n++;  
 }  
  
 if (Math.*pow*(2, n) != numVertices) {  
 return false; // Количество вершин не является степенью двойки  
 }  
  
 // Проверяем, что граф соответствует гиперкубу  
 for (Edge edge : edges) {  
 int source = edge.getSource();  
 int destination = edge.getDestination();  
  
 int xor = source ^ destination;  
 if (Integer.*bitCount*(xor) != 1) {  
 return false; // Ребро не соединяет вершины с различающимися по одному биту кодами  
 }  
 }  
  
 return true;  
 }  
  
 private static GraphData generateHypercube(int n) {  
 int numVertices = (int) Math.*pow*(2, n);  
 List<Edge> edges = new ArrayList<>();  
  
 for (int i = 0; i < numVertices; i++) {  
 for (int j = i + 1; j < numVertices; j++) {  
 if (Integer.*bitCount*(i ^ j) == 1) {  
 edges.add(new Edge(i, j));  
 }  
 }  
 }  
  
 return new GraphData(numVertices, edges);  
 }  
}