МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное бюджетное образовательное учреждение высшего образования

«Севастопольский государственный университет»

кафедра Информационных систем

Голикова Александра Антоновна

Институт информационных технологий и управления в технических системах

курс 4 группа ИС/б-16-2-о

09.03.02 Информационные системы и технологии (уровень бакалавриата)

ОТЧЕТ

по лабораторной работе №8

по дисциплине «Теория распределенных систем и параллельных вычислений»

на тему «Исследование алгоритмов поиска кратчайшего пути на графе»

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

(дата)

Руководитель практикума

ст. преподаватель   А. Ю. Дрозин

(должность) (подпись) (инициалы, фамилия)

Севастополь

2019

1. ЦЕЛЬ РАБОТЫ

## Реализовать программно и исследовать эффективность алгоритмов поиска кратчайшего пути на графе с использованием функций библиотеки MPI.

2. ПОРЯДОК ВЫПОЛНЕНИЯ

Вариант 1

Выполнить разработку программы, реализующей поиск кратчайшего пути на графе при помощи алгоритма Дейкстры. Применить разработанную процедуру к графу на Рис. 2.1 для поиска кратчайшего пути от вершины 0 к вершине 9.

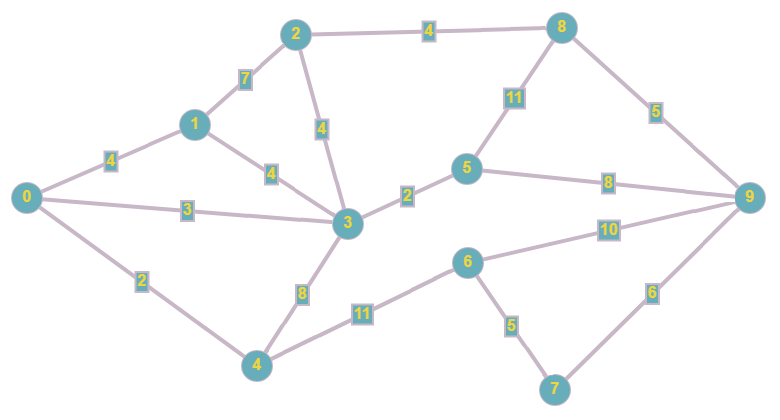


Рисунок 2.1 – Исходный граф для реализации алгоритма Дейкстры

3. ТЕКСТ ПРОГРАММЫ

#include <iostream>

#include<stdio.h>

#include <cfloat>

#include<stdlib.h>

#include <time.h>

#include "mpi.h"

using namespace std;

#define MASTER 0

FILE \*fp;

double \*\*graph=NULL;

double \*minDistance=NULL;

int \*transitionals=NULL;

int \*visited=NULL;

int num=0;

int rankNum, // айди процесса

numtasks, // количество процессов

start, // начальная вершина

size, // размер процесса

initNode=0, //корневая вершина

currentNode, //текущая вершина

totalVisited=0;

char fileName[]="graphs/graph.txt";

void readFile(char c[]) {

if((fp = fopen (c, "r" ))==NULL) {

printf("Не удалось прочитать файл\n");

}

fscanf(fp, "%d" ,&num);

graph = (double\*\*)malloc(num\*sizeof(double\*));

minDistance=(double\*)malloc(num\*sizeof(double));

transitionals=(int \*)malloc(num\*sizeof(int));

if(rankNum==MASTER)

visited=(int \*)malloc(num\*sizeof(int));

// создаем матрицу графа

for(int i=0;i<num;i++){

graph[i] = (double\*)malloc(num\*sizeof(double));

for(int j=0;j<num;j++){

// инициализируем массив графа на бесконечностях

graph[i][j]=DBL\_MAX;

}

// инициализируем вектор расстояний на бесконечностях

minDistance[i]=DBL\_MAX;

transitionals[i]=i;

if(rankNum==MASTER)

visited[i]=0;

}

int a,b;

double d;

while(feof(fp)==0){

fscanf(fp,"%d\t%d\t%lf",&a,&b,&d);

graph[a][b]=d;

graph[b][a]=d;

}

}

void calcTaskAmount() {

int nmin, nleft, nnum;

//определяем размер процесса

nmin=num/numtasks;

nleft=num%numtasks;

int k=0;

for (int i = 0; i < numtasks; i++) {

nnum = (i < nleft) ? nmin + 1 : nmin;

if(i==rankNum){

start=k;

size=nnum;

printf ("Процесс№%2d считает с вершины %2d до %2d \n", rankNum,start, size);

}

k+=nnum;

}

}

void printGraph(){

for(int i = 0; i < num; i++){

for (int j = 0; j < num; ++j){

if(graph[i][j]!=DBL\_MAX)

printf("{%.0f}\t",graph[i][j]);

else

printf("{-}\t");

}

printf("\n");

}

}

void updateminDistance(){

for(int i=start;i<start+size;i++){

if(graph[currentNode][i]<DBL\_MAX){

if((graph[currentNode][i]+minDistance[currentNode])<minDistance[i]){

minDistance[i]=graph[currentNode][i]+minDistance[currentNode];

transitionals[i]=currentNode;

}

}

}

}

void recieveDataFromSlave(){

MPI\_Status status;

int buffer[2];

for(int i=1;i<numtasks;i++){

MPI\_Recv(buffer,2,MPI\_INT,i,0,MPI\_COMM\_WORLD,&status);

int iStart=buffer[0];

int iSize=buffer[1];

MPI\_Recv(&minDistance[iStart],iSize,MPI\_DOUBLE,i,1,MPI\_COMM\_WORLD,&status);

MPI\_Recv(&transitionals[iStart],iSize,MPI\_INT,i,2,MPI\_COMM\_WORLD,&status);

}

}

void sendDataToRoot(){

int buffer[2];

buffer[0]=start;

buffer[1]=size;

MPI\_Send(buffer,2,MPI\_INT,0,0,MPI\_COMM\_WORLD);

MPI\_Send(&minDistance[start],size,MPI\_DOUBLE,0,1,MPI\_COMM\_WORLD);

MPI\_Send(&transitionals[start],size,MPI\_INT,0,2,MPI\_COMM\_WORLD);

}

int main(int argc, char \*argv[]) {

MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rankNum);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&numtasks);

readFile(fileName);

MPI\_Barrier(MPI\_COMM\_WORLD);

calcTaskAmount();

//начать мерять время

double ti= MPI\_Wtime();

if(rankNum==MASTER){

printGraph();

//currentNode=initNode;

}

//сказать всем, на какой я вершине

MPI\_Bcast(&currentNode, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

transitionals[currentNode]=-1;

minDistance[currentNode]=0;

while(totalVisited<num){

updateminDistance();

if(rankNum==MASTER)

recieveDataFromSlave();

else

sendDataToRoot();

if(rankNum==MASTER){

visited[currentNode]=1;

totalVisited++;

double min=DBL\_MAX;

int index=0;

for(int i=0;i<num;i++){

if(visited[i]!=1 && minDistance[i]<min){

min=minDistance[i];

index=i;

}

}

currentNode=index;

}

MPI\_Bcast(&currentNode,1,MPI\_INT,0,MPI\_COMM\_WORLD);

MPI\_Bcast(minDistance,num,MPI\_DOUBLE,0,MPI\_COMM\_WORLD);

MPI\_Bcast(transitionals,num,MPI\_INT,0,MPI\_COMM\_WORLD);

MPI\_Bcast(&totalVisited,1,MPI\_INT,0,MPI\_COMM\_WORLD);

}

//закончиить мерить время

double tf= MPI\_Wtime();

if(rankNum==MASTER){

printf("Минимальные расстояния [");

for(int i=0;i<num-1;i++)

printf("%.2f, ",minDistance[i]==DBL\_MAX ? -1:minDistance[i]);

printf("%.2f]\n",minDistance[num-1]==DBL\_MAX ? -1:minDistance[num-1]);

printf("Промежуточных вершин [");

for(int i=0;i<num-1;i++)

printf("%d, ",transitionals[i]);

printf("%d]\n",transitionals[num-1]);

printf("Время выполнения: %f\n",(tf-ti) );

}

MPI\_Barrier(MPI\_COMM\_WORLD);

free(graph);

free(minDistance);

free(transitionals);

free(visited);

MPI\_Finalize();

return 0;

}

4. РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

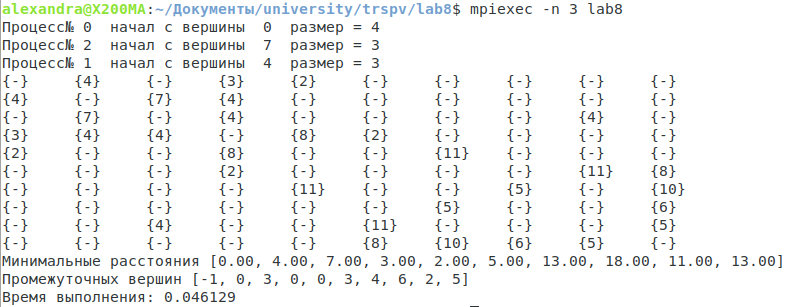


Рисунок 1 – Выполнение программы

ВЫВОДЫ

В рамках лабораторной работы была разработана программа, организующая механизм взаимодействия параллельно выполняющихся процессов — «Клиент - Сервер». Сервер реализован в соответствии со схемой управления использования рандеву. Программа реализует случай, когда сервер отделён от ресурса и к серверу идёт обращение на разрешение использования или освобождение ресурса, а клиент может непосредственно обратиться к ресурсу.