МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**ЛАБОРАТОРНАЯ РАБОТА**

### «**Нахождение минимально охватывающего дерева**

### **(алгоритм Краскала)**»

**Выполнил:** студент группы 381506-1

Никифорова Екатерина Александровна

Нижний Новгород

2017

### Содержание

[Постановка задачи 3](#_Toc501828223)

[Метод решения 4](#_Toc501828224)

[Схема распараллеливания 5](#_Toc501828225)

[Описание программной реализации 6](#_Toc501828226)

[Подтверждение корректности 8](#_Toc501828227)

[Результаты экспериментов и оценка масштабируемости 9](#_Toc501828228)

### Постановка задачи

Минимальным остовным (охватывающим) деревом (МОД) связного взвешенного графа называется его связный подграф, состоящий из всех вершин исходного дерева и некоторых его ребер, причем сумма весов ребер максимально возможная. Существует несколько алгоритмов позволяющих найти МОД: алгоритм Борувки, алгоритм Прима, алгоритм Краскала.

В данной лабораторной работе будет представлена реализация последнего алгоритма. Программа должная содержать последовательную и параллельную реализации, сравнивать результат их выполнения и выводить время работы.

### Метод решения

Зададим граф матрицей смежности:

* Если две вершины не смежны между собой, то ставим 0.
* Если же вершины между собой смежны, храним вес ребра, соединяющего их.

Постоим по матрице массив вершин, которые будут хранить: имя самой вершины и её цвет (множество, к которому она принадлежит). Изначально цвет всех вершин 0.

Заведём массив рёбер, хранящих в себе имена конца\начала ребра и его вес.

Приступим к самому алгоритму:

* Упорядочиваем рёбра в порядке возрастания веса.
* Просматриваем упорядоченный массив и выбираем ребро с минимальным весом не включенное в результирующее дерево и не образующее цикла с рёбрами, которые уже включены.
* Если все возможные вершины включены в дерево и количество рёбер на единицу меньше количества вершин, то алгоритм выполнен.

Как отследить, чтобы ребро не образовывало цикла? Изначально, перед работой алгоритма, первую вершину красим в другой цвет. При выборе ребра, с началом в этой вершине, красим в этот же цвет её конец. Допускается, что в ходе работы алгоритма может быть выделено ещё множество вершин другого цвета. Мы проверяем, принадлежат ли концы отрезка разным множествам. Если они из одного множества, то получится цикл и ребро не подходит. При соединении подграфов разных цветов, второй подграф целиком окрашивается в цвет первого. В результате работы алгоритма мы получаем остов, вершины которого окрашены в один цвет.

Время работы алгоритма Краскала сильно зависит от работы сортировки, используемой для упорядочивания рёбер. В программе представлена сортировка чёт-нечет. Это относительно простой алгоритм сортировки, модификация сортировки пузырьком. Причина выбора данной сортировке заключается в удобстве её распараллеливания. Для последовательной реализации суть алгоритма в том, что мы сравниваем элементы сначала под чётными индексами, потом под нечётными независимо друг от друга.

### Схема распараллеливания

В случае распараллеливания сама суть сортировки не меняется: мы работаем сначала с чётными, потом с нечётными. Но сейчас это относится к парам процессов.

Изначально делим массив между процессами, рассылаем и сортируем свой кусочек в каждом процессе. Далее происходит обмен данными между парами процессов:

* Внутри чётных пар (0,1), (2,3), (4,5) и т.д.

Либо:

* Внутри нечётных пар (1,2), (3,4), (5,6) и т.д.

После обмена, объединяем упорядоченные данные из обоих процессов в один большой упорядоченный массив, делим его пополам, рассылаем обратно и переходим к следующему обмену.

### Описание программной реализации

**Руководство пользователя**

На вход программы поступает 1 параметр: аrg1 (количество вершин в графе). Граф генерируется рандомно. Нужно понимать, что программа будет работать в основном с рёбрами, количество которых в полном графе будет n\*(n-1)/2 (при n вершин), т.е. при вводе 1000 вершин мы будем работать с массивом около 500 тысяч вершин.

При воде большого количества вершин (больше 30) программа выведет:

1. Количество рёбер
2. Время работы последовательного алгоритма
3. Количество процессов
4. Эквивалентны ли результаты работы
5. Время работы параллельного алгоритма

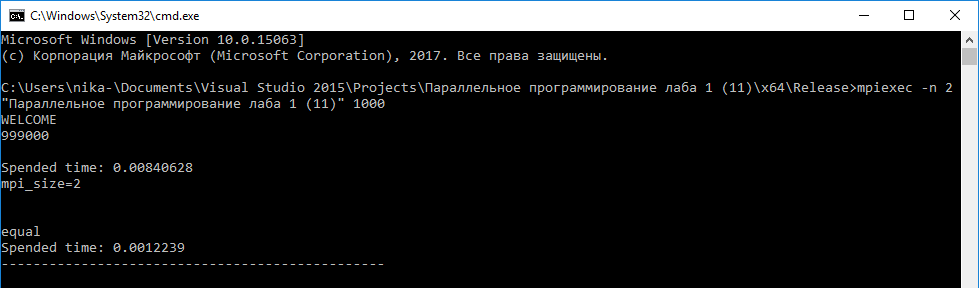
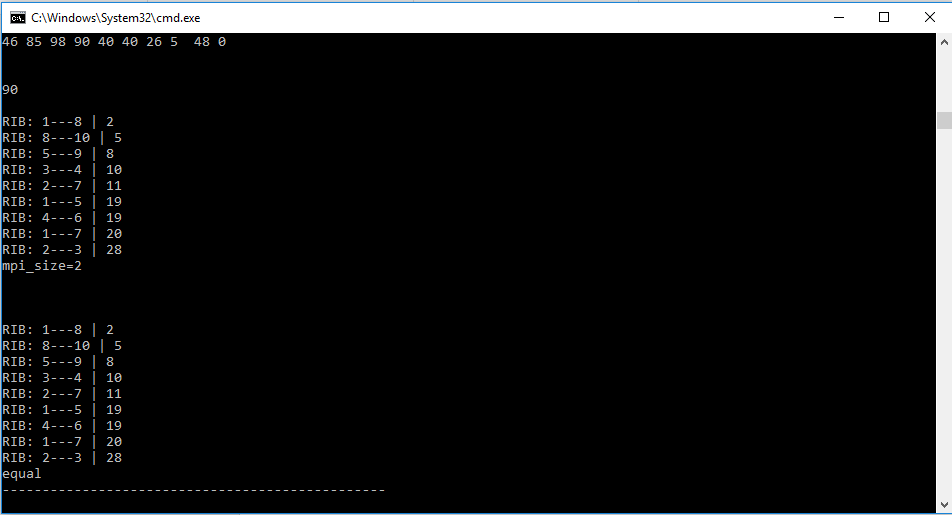


Рис. 1. Пример работы программы.

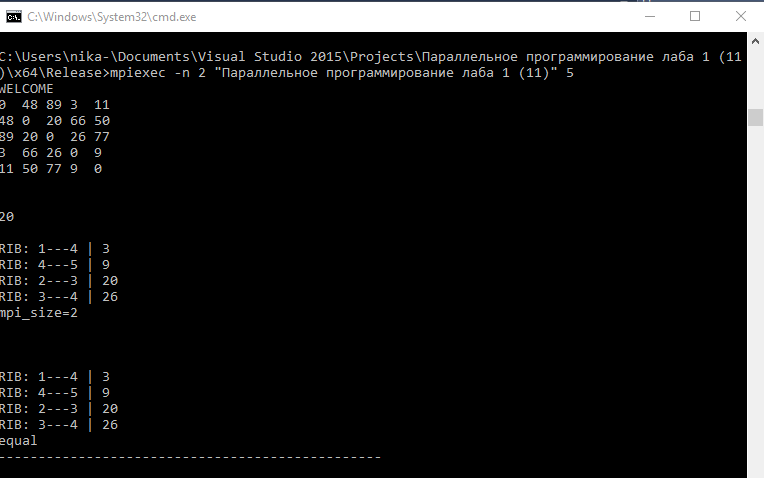
При запуске программы на небольшом графе, программа дополнительно выведет результаты работы обоих алгоритмов.

**Руководство программиста**

|  |  |
| --- | --- |
| void main(int argc, char\*\* argv) | Параметры argc, argv – параметры, передаваемые в приложение через консоль.  Основная функция в программе. Отвечает за вызов функций, вывод результатов работы. |
| void generatematrix(int\*\* mat, int len, int edges, int first\_value, int last\_value) | Функция генерации матрицы смежности, принимает число вершин, необходимое число рёбер, границы для рандома. |
| void generategraph(node\* gr, int\*\* mat, int tops, int edges) | Генерирует по матрице смежности массив со структурами (node- вершина), нужно для отслеживания циклов. |
| void fullmasofribs(rib \*Kr, int tops,node \*graph) | По данным с вершин формирует массив с рёбрами, с которым будет работать непосредственно сам алгоритм. |
| void OddEvenSort(rib \*A, int n) | Последовательная реализация сортировки, работает с массивом рёбер длины n. |
| int MPI\_OddEven\_Sort(int n, rib \*a, int root, MPI\_Comm comm) | Параллельная реализация сортировки, организует раздачу данных и распределение процессов по парам. Использует дополнительную функцию (см. ниже) |
| void MPI\_Pairwise\_Exchange(int localn, rib \*locala, int sendrank, int recvrank, MPI\_Comm comm) | Дополнительная функция, реализует обмен между парами, слияние данных и их перераспределение. |

### Подтверждение корректности

Для проверки корректности на небольших объёмах данных выводится матрица смежности и результат работы последовательной и параллельной реализации. Также результаты автоматически сравниваются. Количество рёбер в остове должно быть меньше на единицу, чем количество рёбер, это тоже способствует проверке.



### Результаты экспериментов и оценка масштабируемости

Все эксперименты проводились на кластере ННГУ им. Лобачевского. Максимальная длинна отреза равна 10.

Ниже представлены:

* график зависимости времени работы от количества процессов и объёма данных (количества рёбер).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 8 | 16 |
| 100000 | 0,000660419 | 0,00052828 | 0,000669081 | 0,000732217 | 0,000172117 |
| 300000 | 0,00220251 | 0,000344333 | 0,00023327 | 0,000682767 | 0,000907939 |
| 500000 | 0,00358872 | 0,000173505 | 0,000240814 | 0,00027797 | 0,000527438 |
| 700000 | 0,00499225 | 0,000503119 | 0,000227962 | 0,000295009 | 0,00068333 |
| 1000000 | 0,00706263 | 0,000284566 | 0,000281663 | 0,000281323 | 0,00058974 |

* График ускорения на различных количествах рёбер.

По результатам понятно, что на наибольшем количестве процессов максимальное ускорение не достигается, наиболее оптимальна работа на 4 процессах. Это можно объяснить тем, что на большем количестве процессов передача данных обходится дороже.