РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3.2

по дисциплине: Дискретная математика
тема: «Транзитивное замыкание отношения »

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Рязанов Юрий Дмитриевич
Бондаренко Татьяна Владимировна

Белгород 2022 г.

Цель работы: изучить и выполнить сравнительный анализ алгоритмов вычисления транзитивного замыкания отношения.

Выполнение заданий

1. Изучить и программно реализовать алгоритмы объединения степеней и Уоршалла для вычисления транзитивного замыкания отношения.

```
#include <iostream>
#include <vector>
#include <set>
#include <ctime>
#include "binaryRelation.h"


using binaryRelationMatrixRow = std::vector<bool>;
using binaryRelation = std::vector<binaryRelationMatrixRow>;


binaryRelation binaryRelation_transitiveClosure_PowerUnion(
  const binaryRelation &A) {
  auto C = A;
  auto A_pow = binaryRelation_composition(A, A);
  for (int i = 2; i < A.size(); ++i) {
    C = binaryRelation_union(C, A_pow);
    A_pow = binaryRelation_composition(A_pow, A);
  }
  return C;
}

binaryRelation binaryRelation_transitiveClosure_Warshall(
  const binaryRelation &A) {
  size_t dimension = A.size();
  auto C = A;
  for (int z = 1; z < dimension; ++z)
    for (int x = 1; x < dimension; ++x)
      for (int y = 1; y < dimension; ++y)
        C[x][y] = C[x][y] || C[x][z] && C[z][y];
  return C;
}
```

2. Разработать и программно реализовать генератор отношений на множестве мощности N и содержащих заданное число пар.

```
binaryRelation generateBinaryRelation(const size_t &start_x,
                                      const size_t &start_y,
                                      const size_t &nPairs,
                                      const size_t &setSize) {
  binaryRelation res(setSize + 1,
                     binaryRelationMatrixRow(setSize + 1, false));
  for (int i = 0; i < nPairs; ++i)
    res[(start_x + i / setSize)
        % setSize + 1][(start_y + i % setSize) % setSize + 1] = true;
  return res;
}
std::vector<binaryRelation> getGeneratedBR(const size_t &nPairs,
                                           const size_t &setSize,
                                           const size_t &nRelations = 1000) {
```

```
  std::vector<binaryRelation> res;
  srand(time(nullptr));
  for (int i = 0; i < nRelations; ++i) {
    size_t x = rand() % setSize + 1;
    size_t y = rand() % setSize + 1;
    res.push_back(generateBinaryRelation(
      x, y, nPairs, setSize));
  }
  return res;
}
```

3. Разработать и написать программу, которая генерирует 1000 отношений на множестве мощности N с заданным числом пар, для каждого отношения вычисляет транзитивное замыкание двумя алгоритмами и определяет время выполнения каждого алгоритма. Время вычисления транзитивного замыкания различных отношений на множестве мощности N с заданным числом пар может быть разным, поэтому программа так же должна определять минимальное и максимальное время вычисления транзитивного замыкания сгенерированных отношений. Выполнить программу при N = 50, 100 и 150. Результат для каждого N представить в виде таблицы (табл. 3).

```
#include <float.h>
#include <fstream>

#define TIME_TEST(testCode, time) { \
 clock_t start_time = clock(); \
 testCode \
 clock_t end_time = clock(); \
 clock_t sort_time = end_time - start_time; \
 time = sort_time * 1000.0 / CLOCKS_PER_SEC; \
}

void testTransitiveClosureAlgorithms(const size_t &setSize,
                                     std::ostream &out) {
  std::pair<binaryRelation (*)(const binaryRelation &),
    std::string> funcs[] = {
    {binaryRelation_transitiveClosure_PowerUnion, "Power Union"},
    {binaryRelation_transitiveClosure_Warshall, "Warshall"}
  };
  std::vector<binaryRelation>
    relations[] = {
    getGeneratedBR(1, setSize),
    getGeneratedBR(setSize * setSize / 4, setSize),
    getGeneratedBR(setSize * setSize / 2, setSize),
    getGeneratedBR(setSize * setSize * 2 / 3, setSize),
    getGeneratedBR(setSize * setSize, setSize)};
  std::cout << "Relations generated" << '\n' << std::endl;
  for (const auto &func: funcs) {
    out << func.second << ';';
    std::cout << "Testing: " << func.second << std::endl;
    for (const auto &rel_vector: relations) {
      double minTime = DBL_MAX;
      double maxTime = 0;
      for (const auto &rel: rel_vector) {
        double currTime;
        TIME_TEST(func.first(rel);, currTime)
        if (currTime > maxTime) maxTime = currTime;
        else if (currTime < minTime) minTime = currTime;
```

```
      }
      out << minTime << " ms;" << maxTime << " ms;";
      std::cout << "Pair test passed" << std::endl;
    }
    out << '\n';
    std::cout << std::endl;
  }
}

int main() {
  std::ofstream file("output.csv");
  for (int i = 50; i <= 150; i += 50) {
    testTransitiveClosureAlgorithms(i, file);
  }
  file.close();
  return 0;
}
```

N = 50

Pairs	1		625		1250		1666		2500	
	min	max	min	max	min	max	min	max	min	max
Power Union	7 ms	13 ms	6 ms	9 ms	5 ms	8 ms	4 ms	7 ms	3 ms	5 ms
Warshall	0 ms	1 ms	0 ms	1 ms	0 ms	1 ms	0 ms	2 ms	0 ms	1 ms

N = 100

Pairs	1		2500		5000		6666		10000	
	min	max	min	max	min	max	min	max	min	max
Power Union	90 ms	116 ms	72 ms	100 ms	54 ms	98 ms	42 ms	73 ms	18 ms	23 ms
Warshall	1 ms	3 ms	0 ms	3 ms	1 ms	3 ms	0 ms	3 ms	1 ms	3 ms

N = 150

Pairs	1		5625		11250		15000		22500	
	min	max	min	max	min	max	min	max	min	max
Power Union	424 ms	470 ms	332 ms	466 ms	241 ms	414 ms	181 ms	340 ms	60 ms	80 ms
Warshall	4 ms	8 ms	5 ms	7 ms	4 ms	7 ms	5 ms	7 ms	5 ms	7 ms

Вывод: в ходе работы были изучены алгоритмы транзитивного замыкания отношений и выполнен их сравнительный анализ.