МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №3.4

по дисциплине: Дискретная математика
тема: «Упорядоченные множества»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
ст. пр. Рязанов Юрий Дмитриевич
ст. пр. Бондаренко Татьяна Владимировна

Белгород 2023 г.

Лабораторная работа №3.4
Упорядоченные множества

Цель работы: изучить упорядоченные множества, алгоритм топологической сортировки, научиться представлять множества диаграммами Хассе, находить минимальные (максимальные) и наименьшие (наибольшие) элементы упорядоченного множества.

1. Написать программы, формирующие матрицы отношений в соответствии с вариантом задания, на множествах $_1$ и $_2$.
   alg.h

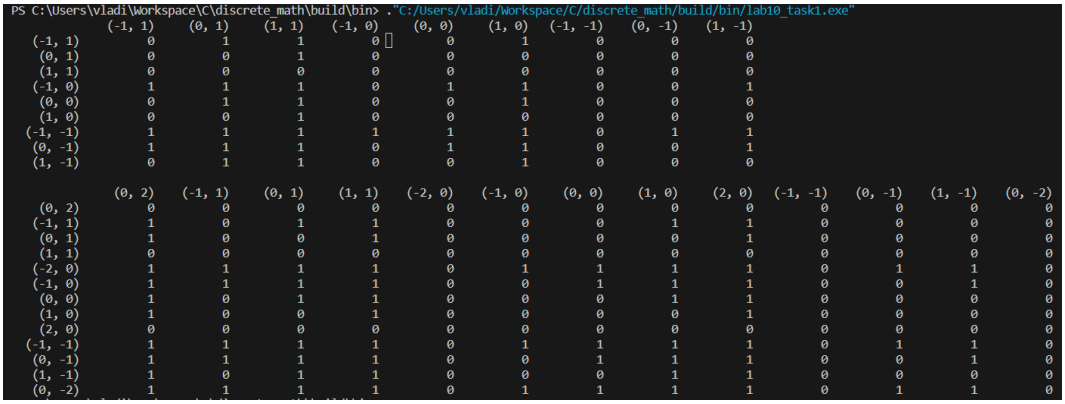
```
template <typename T>
class Relation : public BoolMatrixRelation
{
protected:
    std::vector<T> origin;

public:
    Relation(std::vector<T> origin, std::function<bool(T, T)> pred) :
        BoolMatrixRelation(origin.size(),
        [&origin, &pred](int x, int y) {
                return pred(origin[x - 1], origin[y - 1]);
        })
    {
        this->origin = origin;
    };

    friend std::ostream& operator<<(std::ostream& out, Relation<T>& val) {
        int blockSize = 10;
        out << std::setw(blockSize) << ""
            << " ";
        for (int i = 0; i < val.size; i++)
        {
            std::stringstream ss;
            ss << val.origin[i];
            std::string buf = ss.str();

            out << std::setw(blockSize) << buf << " ";
        }
        out << "\n";

        for (int x = 0; x < val.size; x++)
        {
            std::stringstream ss;
            ss << val.origin[x];
            std::string buf = ss.str();

            out << std::setw(blockSize) << buf << " ";
            for (int y = 0; y < val.size; y++)
            {
                out << std::setw(blockSize) << val.data[x][y] << " ";
            }
```

```
            out << "\n";
        }

        return out;
    }
};
```

main.cpp

```
#include <vector>
#include <iostream>

typedef std::pair<int, int> Vec2;

std::ostream& operator<<(std::ostream& out, Vec2& o) {
    out << "(" << o.first << ", " << o.second << ")";

    return out;
}

#include "../../libs/alg/alg.h"

bool predicate(Vec2 a, Vec2 b) {
    return (a.first - b.first) < (b.second - a.second);
}

int main() {
    std::vector<Vec2> M1 = {
    {-1,  1}, {0,  1}, {1,  1},
    {-1,  0}, {0,  0}, {1,  0},
    {-1, -1}, {0, -1}, {1, -1}
    };

    Relation<Vec2> m1Ordered(M1, predicate);


    std::vector<Vec2> M2 = {
                        { 0,  2},
                {-1,  1}, { 0,  1}, { 1,  1},
    {-2,  0}, {-1,  0}, { 0,  0}, { 1,  0}, { 2,  0},
                {-1, -1}, { 0, -1}, { 1, -1},
                        { 0, -2}
    };

    Relation<Vec2> m2Ordered(M2, predicate);

    std::cout << m1Ordered << "\n" << m2Ordered;

    return 0;
}
```

Результат выполнения программы:



2. Написать программы, формирующие матрицы отношения доминирования по матрицам отношения порядка.
alg.h

```
template <typename T>
class DominantRelation;

template <typename T>
class Relation : public BoolMatrixRelation
{
protected:
    std::vector<T> origin;

public:
    Relation(std::vector<T> origin, std::function<bool(T, T)> pred) : BoolMatrixRelation(origin.size(), [&origin,
    ↪  &pred](int x, int y)
                                                                                    { return pred(origin[x -
                                                                                    ↪  1], origin[y - 1]);
                                                                                    ↪  })
    {
        this->origin = origin;
    };

    DominantRelation<T> getDominantRelation();

    friend std::ostream& operator<<(std::ostream& out, Relation<T>& val) {
        int blockSize = 10;
        out << std::setw(blockSize) << ""
            << " ";
        for (int i = 0; i < val.size; i++)
        {
            std::stringstream ss;
            ss << val.origin[i];
            std::string buf = ss.str();

            out << std::setw(blockSize) << buf << " ";
        }
        out << "\n";

        for (int x = 0; x < val.size; x++)
        {
            std::stringstream ss;
            ss << val.origin[x];
```

```
            std::string buf = ss.str();

            out << std::setw(blockSize) << buf << " ";
            for (int y = 0; y < val.size; y++)
            {
                out << std::setw(blockSize) << val.data[x][y] << " ";
            }

            out << "\n";
        }

        return out;
    }
};

template <typename T>
class DominantRelation : public Relation<T>
{
public:
    DominantRelation(std::vector<T> origin, std::function<bool(T, T)> pred) : Relation<T>(origin, pred) {};
};

#include "lab10/task2.tpp"
```

task2.tpp

```
template <typename T>
DominantRelation<T> Relation<T>::getDominantRelation()
{
    if (!isOrdered())
        throw std::invalid_argument("Original relation should be relation of order");

    DominantRelation<T> result(origin, [](T, T)
                               { return false; });
    for (int x = 0; x < size; x++)
    {
        for (int y = 0; y < size; y++)
        {
            if (!data[x][y])
                continue;
            bool anyZ = false;

            for (int z = 0; z < size && !anyZ; z++)
            {
                if (data[x][z] && data[z][y])
                    anyZ = true;
            }

            result.data[x][y] = !anyZ;
        }
    }

    return result;
```

```
}
```

main.cpp

```
#include <vector>
#include <iostream>

typedef std::pair<int, int> Vec2;

std::ostream& operator<<(std::ostream& out, Vec2& o) {
    out << "(" << o.first << ", " << o.second << ")";

    return out;
}

#include "../../libs/alg/alg.h"

bool predicate(Vec2 a, Vec2 b) {
    return (a.first - b.first) < (b.second - a.second);
}

int main() {
    std::vector<Vec2> M1 = {{-1,  1}, {0,  1}, {1,  1},
                                      {-1,  0}, {0,  0}, {1,  0},
                                      {-1, -1}, {0, -1}, {1, -1}};

    Relation<Vec2> m1Ordered(M1, predicate);
    auto dominationRelM1 = m1Ordered.getDominantRelation();

    std::cout << dominationRelM1 << std::endl;


    std::vector<Vec2> M2 = {
                                          { 0,  2},
                                {-1,  1}, { 0,  1}, { 1,  1},
                          {-2,  0}, {-1,  0}, { 0,  0}, { 1,  0}, { 2,  0},
                                {-1, -1}, { 0, -1}, { 1, -1},
                                          { 0, -2}};

    Relation<Vec2> m2Ordered(M2, predicate);
    auto dominationRelM2 = m2Ordered.getDominantRelation();

    std::cout << dominationRelM2 << std::endl;

    return 0;
}
```

Результат выполнения программы:

```
PS C:\Users\vladi\Workspace\C\discrete_math\build\bin> ."C:/Users/vladi/Workspace/C/discrete_math/build/bin/lab10_task2.exe"
          (-1, 1)    (0, 1)    (1, 1)    (-1, 0)    (0, 0)    (1, 0)   (-1, -1)   (0, -1)   (1, -1)
  (-1, 1)       0         1         0          0         0         1          0         0         0
   (0, 1)       0         0         1          0         0         0          0         0         0
   (1, 1)       0         0         0          0         0         0          0         0         0
  (-1, 0)       1         0         0          0         1         0          0         0         1
   (0, 0)       0         1         0          0         0         1          0         0         0
   (1, 0)       0         0         1          0         0         0          0         0         0
 (-1, -1)       0         0         0          1         0         0          0         1         0
  (0, -1)       1         0         0          0         1         0          0         0         1
  (1, -1)       0         1         0          0         0         1          0         0         0

          (0, 2)   (-1, 1)    (0, 1)    (1, 1)   (-2, 0)   (-1, 0)    (0, 0)    (1, 0)    (2, 0)  (-1, -1)   (0, -1)   (1, -1)   (0, -2)
  (0, 2)       0         0         0          0         0         0          0         0         0         0         0         0         0
 (-1, 1)       0         0         1          0         0         0          0         1         0         0         0         0         0
  (0, 1)       1         0         0          1         0         0          0         0         1         0         0         0         0
  (1, 1)       0         0         0          0         0         0          0         0         0         0         0         0         0
 (-2, 0)       0         0         0          0         0         1          0         0         0         0         1         0         0
 (-1, 0)       0         1         0          0         0         0          1         0         0         0         0         1         0
  (0, 0)       0         0         1          0         0         0          0         1         0         0         0         0         0
  (1, 0)       1         0         0          1         0         0          0         0         1         0         0         0         0
  (2, 0)       0         0         0          0         0         0          0         0         0         0         0         0         0
 (-1, -1)      0         0         0          0         0         1          0         0         0         0         1         0         0
  (0, -1)      0         1         0          0         0         0          1         0         0         0         0         1         0
  (1, -1)      0         0         1          0         0         0          0         1         0         0         0         0         0
  (0, -2)      0         0         0          0         0         1          0         0         0         0         1         0         0
```

3. Написать программу, реализующую алгоритм топологической сортировки по матрице отношения доминирования.
alg.h

```
template <typename T>
class DominantRelation;

template <typename T>
class Relation : public BoolMatrixRelation
{
protected:
    std::vector<T> origin;

public:
    Relation(std::vector<T> origin, std::function<bool(T, T)> pred) : BoolMatrixRelation(origin.size(), [&origin,
    ↪  &pred](int x, int y)
                                                                                { return pred(origin[x -
                                                                                ↪  1], origin[y - 1]);
                                                                                ↪  })
    {
        this->origin = origin;
    };

    DominantRelation<T> getDominantRelation();

    friend std::ostream& operator<<(std::ostream& out, Relation<T>& val) {
        int blockSize = 10;
        out << std::setw(blockSize) << ""
            << " ";
        for (int i = 0; i < val.size; i++)
        {
            std::stringstream ss;
            ss << val.origin[i];
            std::string buf = ss.str();

            out << std::setw(blockSize) << buf << " ";
        }
        out << "\n";

        for (int x = 0; x < val.size; x++)
        {
            std::stringstream ss;
```

```
            ss << val.origin[x];
            std::string buf = ss.str();

            out << std::setw(blockSize) << buf << " ";
            for (int y = 0; y < val.size; y++)
            {
                out << std::setw(blockSize) << val.data[x][y] << " ";
            }

            out << "\n";
        }

        return out;
    }
};

template <typename T>
class DominantRelation : public Relation<T>
{
public:
    DominantRelation(std::vector<T> origin, std::function<bool(T, T)> pred) : Relation<T>(origin, pred) {};

    std::vector<std::vector<T>> getTopologicalSort();
};

#include "lab10/task2.tpp"
#include "lab10/task3.tpp"
```

task3.tpp

```
template <typename T>
std::vector<std::vector<T>> DominantRelation<T>::getTopologicalSort() {
    std::vector<int> W(this->size, 0);

    for (int x = 0; x < this->size; x++) {
        for (int y = 0; y < this->size; y++) {
            W[y] += this->data[x][y];
        }
    }

    int ind = 0;

    while (true)
    {
        std::vector<int> indices;

        // Поиск нулевых элементов и сохранение их индексов
        auto iter = W.begin();
        while ((iter = std::find(iter, W.end(), 0)) != W.end()) {
            indices.push_back(iter - W.begin());

            iter++;
        }
```

```
        if (indices.size() == 0) break;

        ind--;
        for (auto index : indices) {
            // Заменяем нулевой элемент
            W[index] = ind;

            for (int elementIndex = 0; elementIndex < this->data[index].size(); elementIndex++) {
                auto element = this->data[index][elementIndex];
                W[elementIndex] -= element;
            }
        }
    }

    std::vector<std::vector<T>> levels;
    for (int i = 0; i < this->size; i++) {
        std::vector<T> level;

        for (int j = 0; j < this->size; j++) {
            if (W[j] == -(1 + i))
                level.push_back(this->origin[j]);
        }

        if (level.empty()) break;

        levels.push_back(level);
    }

    return levels;
}
```

main.cpp

```
#include <vector>
#include <iostream>

typedef std::pair<int, int> Vec2;

std::ostream& operator<<(std::ostream& out, Vec2& o) {
    out << "(" << o.first << ", " << o.second << ")";

    return out;
}

#include "../../libs/alg/alg.h"

bool predicate(Vec2 a, Vec2 b) {
    return (a.first - b.first) < (b.second - a.second);
}

int main() {
    std::vector<Vec2> M1 = {{-1,  1}, {0,  1}, {1,  1},
```

```
                                        {-1,  0}, {0,  0}, {1,  0},
                                        {-1, -1}, {0, -1}, {1, -1}};

    Relation<Vec2> m1Ordered(M1, predicate);
    auto dominationRelM1 = m1Ordered.getDominantRelation();
    auto m1levels = dominationRelM1.getTopologicalSort();
    for (auto it = m1levels.end() - 1; it >= m1levels.begin(); it--) {
        for (auto element : *it) {
            std::cout << "(" << element.first << ", " << element.second << ") ";
        }

        std::cout << "\n";
    }

    std::cout << std::endl;


    std::vector<Vec2> M2 = {
                                            { 0,  2},
                                    {-1,  1}, { 0,  1}, { 1,  1},
                            {-2,  0}, {-1,  0}, { 0,  0}, { 1,  0}, { 2,  0},
                                    {-1, -1}, { 0, -1}, { 1, -1},
                                            { 0, -2}};

    Relation<Vec2> m2Ordered(M2, predicate);
    auto dominationRelM2 = m2Ordered.getDominantRelation();
    auto m2levels = dominationRelM2.getTopologicalSort();
    for (auto it = m2levels.end() - 1; it >= m2levels.begin(); it--) {
        for (auto element : *it) {
            std::cout << "(" << element.first << ", " << element.second << ") ";
        }

        std::cout << "\n";
    }

    return 0;
}
```

Результат выполнения программы:

```
PS C:\Users\vladi\Workspace\C\discrete_math\build\bin>
(1, 1)
(0, 1) (1, 0)
(-1, 1) (0, 0) (1, -1)
(-1, 0) (0, -1)
(-1, -1)

(0, 2) (1, 1) (2, 0)
(0, 1) (1, 0)
(-1, 1) (0, 0) (1, -1)
(-1, 0) (0, -1)
(-2, 0) (-1, -1) (0, -2)
```

Вывод: в ходе лабораторной работы научились формировать фактормножество для заданного отношения эквивалентности на ЭВМ.