Министерство образования Российской Федерации

Пермский Национальный Исследовательский Политехнический Университет

Кафедра «Информационные технологии и автоматизированные системы»

Лабораторная работа № 2

Тема: «Диаграммы Хассе»

Выполнил:

Студент группы ИВТ-19-2б

Шеретов Марк Алексеевич

Принял:

Доцент кафедры ИТАС

А.Ф. Хабибулин

Пермь 2020

ПОСТАНОВКА ЗАДАЧИ

Диаграмма Хассе. Вводится, подмножество. Найти максимум, минимум, супремум, инфимум

КОД ПРОГРАММЫ НА ЯП C#:

Класс Program:

using System;

using System.Collections.Generic;

using System.Linq;

namespace diskret\_lab2

{

    class Program

    {

        static void Main(string[] args)

        {

            CreateSet(out Dictionary<string, Element> set);

            AddRelation(set);

            CreateSubSet(set, out HashSet<Element> subSet);

            PrintMinMax(set);

            Console.WriteLine($"Supremum: {FindSupremumInSubset(subSet)}");

            Console.WriteLine($"Infimum: {FindInfimumInSubset(subSet)}");

        }

        /// <summary>

        /// Создание множества

        /// </summary>

        /// <param name="set">Множество для создания</param>

        static void CreateSet(out Dictionary<string,Element> set)

        {

            set = new Dictionary<string, Element>(); //Множество

            string[] buffer = EnterSet(); //Ввод элементов множества

            foreach(string key in buffer)//Добавление элементов в словарь-множество

                if(!set.TryGetValue(key, out \_)) set.Add(key,new Element(key));

        }

        /// <summary>

        /// Ввод строчек в массив строчек

        /// </summary>

        /// <returns>Массив введённых строчек</returns>

        static string[] EnterSet()

        {

            Console.Write("Введите элементы множества: ");

            return Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

        }

        static string[] EnterSubSet()

        {

            Console.Write("Введите элементы подмножества: ");

            return Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

        }

        /// <summary>

        /// Создание подмножества

        /// </summary>

        /// <param name="set">Множество</param>

        /// <param name="subSet">Подмножество</param>

        static void CreateSubSet(Dictionary<string,Element> set, out HashSet<Element> subSet)

        {

            subSet = new HashSet<Element>();

            string[] elementKeys = EnterSubSet();

            foreach(string key in elementKeys)

            {

                if(set.ContainsKey(key)) subSet.Add(set[key]);

                else Console.WriteLine($"Элемент {key} не существует в исходном множестве");

            }

        }

        /// <summary>

        /// Добавление отношений

        /// </summary>

        /// <param name="set">Множество элементов</param>

        static void AddRelation(Dictionary<string, Element> set)

        {

            int relsCount = IntInput("Введите количество отношений в диаграмме: ");

            do

            {

                string[] rels = RelationInput();

                if(set.TryGetValue(rels[0], out Element lower) & set.TryGetValue(rels[1], out Element upper))

                {

                    if(lower.AddRelation(upper))//если удалось добавить отношение

                    {

                        Console.WriteLine("Added");//Сообщение об успешном добавлении

                        relsCount--;

                    }

                    else//Сообщение, что отношение уже существует

                    {

                        Console.WriteLine($"Отношение между {lower} и {upper} уже существует");

                    }

                }

                else //Если какого-то элемента нет в множестве - вывод собщения

                {

                    if(lower == null) Console.WriteLine($"Элемента {rels[0]} нет в множестве");

                    if(upper == null) Console.WriteLine($"Элемента {rels[1]} нет в множестве");

                }

            }while(relsCount != 0);

        }

        /// <summary>

        /// Ввод строки отношения

        /// </summary>

        /// <returns></returns>

        static string[] RelationInput()

        {

            Console.WriteLine("Введите отношение в формате А-B");

            return Console.ReadLine().Split(new char[] { '-' }, StringSplitOptions.RemoveEmptyEntries);

        }

        /// <summary>

        /// Ввод целого числа

        /// </summary>

        /// <param name="invMsg">Строка - текст приглашения</param>

        /// <returns>целое число, введённое пользователем</returns>

        static int IntInput(string invMsg)

        {

            int buff;

            do

            {

                try

                {

                    Console.WriteLine(invMsg);

                    int.TryParse(Console.ReadLine(), out buff);

                    return buff;

                }

                catch (System.TypeInitializationException ex)

                {

                    Console.WriteLine(ex.Message);

                    Console.WriteLine("Попробуйте ещё раз");

                }

            }while(true);

        }

        static void PrintMinMax(Dictionary<string, Element> set)

        {

            HashSet<Element> min = FindMin(set.Values); //Поиск минимумов

            HashSet<Element> max = FindMax(set.Values); //Поиск максимумов

            PrintSet("min", min);

            PrintSet("max", max);

        }

        /// <summary>

        /// Поиск минимумов

        /// </summary>

        /// <param name="set">множество</param>

        /// <returns></returns>

        static HashSet<Element> FindMin(Dictionary<string, Element>.ValueCollection set)

        {

            HashSet<Element> min = new HashSet<Element>();

            foreach (Element elem in set)

            {

                if(elem.LowerRelations.Count == 0 && elem.UpperRelations.Count != 0)

                    min.Add(elem);

            }

            return min;

        }

        /// <summary>

        /// Поиск максимумов

        /// </summary>

        /// <param name="set">множество</param>

        /// <returns></returns>

        static HashSet<Element> FindMax(Dictionary<string, Element>.ValueCollection set)

        {

            HashSet<Element> max = new HashSet<Element>();

            foreach (Element elem in set)

            {

                if(elem.UpperRelations.Count == 0 && elem.LowerRelations.Count != 0)

                    max.Add(elem);

            }

            return max;

        }

        /// <summary>

        /// Печать множества

        /// </summary>

        /// <param name="name">Название множества</param>

        /// <param name="set">множество</param>

        static void PrintSet(string name, HashSet<Element> set)

        {

            Console.WriteLine($"{name}: {{ ");

            foreach (Element elem in set) Console.Write($" {elem}");

            Console.WriteLine(" }");

        }

        /// <summary>

        /// Поиск супремума в подмножестве

        /// </summary>

        /// <param name="subSet">Подмножество</param>

        /// <returns></returns>

        static Element FindSupremumInSubset(HashSet<Element> subSet)

        {

            Dictionary<Element, HashSet<Element>> helpDict = new Dictionary<Element, HashSet<Element>>();

            foreach(Element elem in subSet) helpDict.Add(elem, elem.GetAllUpperRelations());

            foreach(Element elem in subSet) helpDict[elem].Add(elem);

            HashSet<Element> helpSet = helpDict[subSet.ElementAt(0)];

            foreach(Element elem in subSet)

            {

                helpSet.IntersectWith(helpDict[elem]);

            }

            if(helpSet.Count != 0)

            {

                foreach(Element elem in subSet)

                {

                    if(helpSet.Contains(elem))

                        return elem;

                    return helpSet.ElementAt(0);

                }

            }

            return null;

        }

        /// <summary>

        /// Поиск инфимума в подмножестве

        /// </summary>

        /// <param name="subSet">Подмножество</param>

        /// <returns></returns>

        static Element FindInfimumInSubset(HashSet<Element> subSet)

        {

            Dictionary<Element, HashSet<Element>> helpDict = new Dictionary<Element, HashSet<Element>>();

            foreach (Element elem in subSet) helpDict.Add(elem, elem.GetAllLowerRelations());

            foreach (Element elem in subSet) helpDict[elem].Add(elem);

            HashSet<Element> helpSet = helpDict[subSet.ElementAt(0)];

            foreach (Element elem in subSet)

                helpSet.IntersectWith(helpDict[elem]);

            if (helpSet.Count != 0)

            {

                foreach (Element elem in subSet)

                    if (helpSet.Contains(elem))

                        return elem;

                return helpSet.ElementAt(0);

            }

            return null;

        }

        /// <summary>

        /// Поиск инфимума

        /// </summary>

        /// <param name="subSet">Подмножество</param>

        /// <returns></returns>

        static Element FindInfimum(HashSet<Element> subSet)

        {

            HashSet<Element> lowerEdge = subSet.ElementAt(0).GetAllLowerRelations();

            foreach (Element element in subSet)

                lowerEdge.IntersectWith(element.GetAllLowerRelations());

            if (lowerEdge.Count != 0) return lowerEdge.ElementAt(0);

            else return null;

        }

        /// <summary>

        /// Поиск супремума

        /// </summary>

        /// <param name="subSet">подмножество</param>

        /// <returns></returns>

        static Element FindSupremum(HashSet<Element> subSet)

        {

            HashSet<Element> upperEdge = subSet.ElementAt(0).GetAllUpperRelations(); //Верхняя грань первого элемента

            foreach (Element element in subSet) //Затем поиск пересечений ВГ первого элемента с ВГ других элементов

                upperEdge.IntersectWith(element.GetAllUpperRelations());

            if (upperEdge.Count != 0) return upperEdge.ElementAt(0);

            else return null;

        }

    }

}

Класс Element:

using System.Collections.Generic;

namespace diskret\_lab2

{

    public class Element

    {

        public string Value { get; private set;} //Значение элемента

        public HashSet<Element> UpperRelations{get;private set;} //Связи на уровень выше на диаграмме

        public HashSet<Element> LowerRelations{ get; private set;} //Связи на уровент ниже на диаграмме

        /// <summary>

        /// Конструктор с параметром

        /// </summary>

        /// <param name="value">значение для элемента</param>

        public Element(string value)

        {

            this.Value = value;

            UpperRelations = new HashSet<Element>();

            LowerRelations = new HashSet<Element>();

        }

        /// <summary>

        /// Получение  всех отношений выше

        /// </summary>

        /// <returns></returns>

        public HashSet<Element> GetAllUpperRelations()

        {

            HashSet<Element> relations = new HashSet<Element>(UpperRelations);

            foreach (Element elem in UpperRelations) relations.UnionWith(elem.GetAllUpperRelations());

            return relations;

        }

        /// <summary>

        /// Получение всех отношений ниже

        /// </summary>

        /// <returns></returns>

        public HashSet<Element> GetAllLowerRelations()

        {

            HashSet<Element> relations = new HashSet<Element>(LowerRelations);

            foreach( Element elem in LowerRelations) relations.UnionWith(elem.GetAllLowerRelations());

            return relations;

        }

        /// <summary>

        /// Добавление отношения к элементу

        /// </summary>

        /// <param name="element">Элемент</param>

        /// <returns></returns>

        public bool AddRelation(Element element)

        {

            return AddUpperRelation(element);

        }

        /// <summary>

        /// Добавить связь на уровень выше

        /// </summary>

        /// <param name="element">Элемент</param>

        /// <returns></returns>

        private bool AddUpperRelation(Element element)

        {

            if(!this.UpperRelations.Contains(element) && !this.LowerRelations.Contains(element))

            {

                this.UpperRelations.Add(element);//Добавление отношения

                element.AddLowerRelation(this); // Добавление противоположного отношения новому элементу

                return true;//Добавление успешно

            }

            else return false;//если отношение есть - то не успешно

        }

        /// <summary>

        /// Добавить связь на уровень ниже

        /// </summary>

        /// <param name="element">Элемент</param>

        /// <returns></returns>

        private bool AddLowerRelation(Element element)

        {

            if(!this.LowerRelations.Contains(element))

            {

                this.LowerRelations.Add(element);

                element.AddUpperRelation(this);

                return true;

            }

            else return false;

        }

        /// <summary>

        ///

        /// </summary>

        /// <returns>Значение Value</returns>

        public override string ToString()

        {

            return Value;

        }

    }

}