Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

за 1 семестр

По дисциплине: «Дискретная математика»

Тема: «Множества»

Выполнил:

Студент 2 курса

Группы ПО-4(1)

Иваненко И.Л.

Проверил:

Глущенко Т.А.

2020

Лабораторная работа №1

Множества

Цель: изучить базовые понятия теории множеств и операции над множествами.

Вариант 10

Задание 1.

На универсуме U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11} заданы множества:

А = {1, 3, 5, 6, 7, 8}

B = {2, 3, 4, 5, 8}

C = {1, 2, 3, 5, 6}

Выражение: (A \ B) \ C

1. Для заданного множества А построить булеан

Код программы:

#ФАЙЛ functions.py

#СОЗДАНИЕ И ВЫВОД БУЛЕАНА И БИНАРНОГО КОДА МНОЖЕСТВ

def boolean(array):

boolean = []

boolean\_part = []

boolean\_pattern = []

boolean\_binary\_code = []

boolean\_size = 2 \*\* len(array)

print("\nБИНАРНЫЙ КОД МНОЖЕСТВ:\n")

for n in range(boolean\_size):

while n > 0:

boolean\_binary\_code = [n % 2] + boolean\_binary\_code

n = n // 2

if len(boolean\_binary\_code) < len(array):

for i in range(len(boolean\_binary\_code), len(array)):

boolean\_binary\_code = [0] + boolean\_binary\_code

print(boolean\_binary\_code)

boolean\_pattern.append(boolean\_binary\_code)

boolean\_binary\_code = []

print("\nБУЛЕАН:\n")

for boolean\_binary\_code in boolean\_pattern:

for index in range(len(array)):

if boolean\_binary\_code[index] == 1:

boolean\_part.append(array[index])

print(boolean\_part)

boolean.append(boolean\_part)

boolean\_part = []

return boolean

#ФАЙЛ main.py

import functions

#ОБЪЯВЛЕНИЕ ВХОДНЫХ ДАННЫХ

universum = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

A = [1, 3, 5, 6, 7, 8]

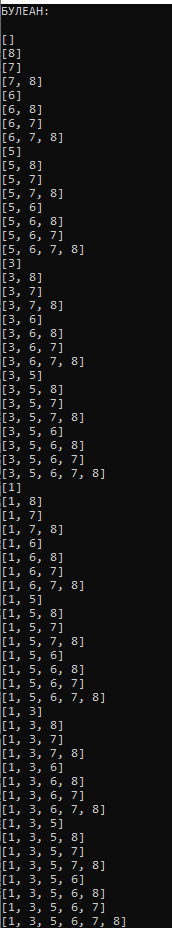
B = [2, 3, 4, 5, 8]

C = [1, 2, 3, 5, 6]

#ЗАДАНИЕ 1 - ПОСТРОИТЬ БУЛЕАН

boolean = functions.boolean(array = A)

Результат выполнения:



2. Реализовать сортировку слиянием

Код программы:

#СЛИВАЕТ ДВА СПИСКА В ОДИН ОТСОРТИРОВАННЫЙ СПИСКА И ВОЗВРАЩАЕТ ЕГО

def merge(left\_array, right\_array):

sorted\_array = []

left\_array\_index = 0

right\_array\_index = 0

for i in range(len(left\_array) + len(right\_array)):

if left\_array\_index < len(left\_array) and right\_array\_index < len(right\_array):

if left\_array[left\_array\_index] < right\_array[right\_array\_index]:

sorted\_array.append(left\_array[left\_array\_index])

left\_array\_index += 1

else:

sorted\_array.append(right\_array[right\_array\_index])

right\_array\_index += 1

elif left\_array\_index == len(left\_array):

sorted\_array.append(right\_array[right\_array\_index])

right\_array\_index += 1

elif right\_array\_index == len(right\_array):

sorted\_array.append(left\_array[left\_array\_index])

left\_array\_index += 1

return sorted\_array

#ФУНКЦИЯ БЫСТРОЙ СОРТИРОВКИ

def merge\_sort(array):

if len(array) < 2:

return array

mid = len(array) // 2

left\_array = merge\_sort(array = array[:mid])

right\_array = merge\_sort(array = array[mid:])

return merge(left\_array = left\_array, right\_array = right\_array)

#ЗАДАНИЕ 2 - СОРТИРОВКА СЛИЯНИЕМ

print("\nОСТСОРТИРОВАННОЕ МНОЖЕСТВО:\n")

print(functions.merge\_sort(A + B))

Результат выполнения:



3. Реализовать операцию пересечения множеств, используя маску

Код программы:

#СОЗДАНИЕ БИТОВОЙ МАСКИ

def bitmask\_generation(array, universum):

bitmask = []

for element in universum:

if element in array:

bitmask.append(1)

else:

bitmask.append(0)

return bitmask

#ФУНКЦИЯ ПЕРЕСЕЧЕНИЯ МНОЖЕСТВ

def array\_crossing(array1, array2, array3, universum):

array1\_bitmask = bitmask\_generation(array = array1, universum = universum)

array2\_bitmask = bitmask\_generation(array = array2, universum = universum)

array3\_bitmask = bitmask\_generation(array = array3, universum = universum)

crossing\_array = []

for index in range(len(universum)):

if array1\_bitmask[index] == array2\_bitmask[index] == array3\_bitmask[index] == 1:

crossing\_array.append(universum[index])

print("\nПЕРЕСЕЧЕНИЕ МНОЖЕСТВ:\n")

print(crossing\_array

#ЗАДАНИЕ 3 - РЕАЛИЗОВАТЬ ОПЕРАЦИЮ ПЕРЕСЕЧЕНИЯ МНОЖЕСТВ, ИСПОЛЬЗУЯ МАСКУ

functions.array\_crossing(array1 = A, array2 = B, array3 = C, universum = universum)

Результат выполнения:



4. Найти прямое произведение А х В и его мощность

Код программы:

#ПРЯМОЕ ПРОИЗВЕДЕНИЕ МНОЖЕСТВ

def array\_union(array1, array2):

union = []

union\_part = []

for i in range(len(array1)):

for j in range(len(array2)):

union\_part.append(array1[i])

union\_part.append(array2[j])

union.append(union\_part)

union\_part = []

union\_power = len(union) \* 2

print("\nПРОИЗВЕДЕНИЕ:" + str(union) + "\n")

print("МОЩНОСТЬ:" + str(union\_power) + "\n")

#ЗАДАНИЕ 4 - НАЙТИ ПРЯМОЕ ПРОИЗВЕДЕНИЕ А х В И ЕГО МОЩНОСТЬ

functions.array\_union(array1 = A, array2 = B)

Результат выполнения:



5. Вычислить программно выражение согласно варианту, проиллюстрировать результат диаграммой Эйлера-Венна

Код программы:

#ФУНКЦИЯ ВОЗВРАТА РАЗНОСТИ 2 МНОЖЕСТВ

def array\_differ(array1, array2, universum):

array1\_bitmask = bitmask\_generation(array = array1, universum = universum)

array2\_bitmask = bitmask\_generation(array = array2, universum = universum)

differ\_array = array1

for index in range(len(universum)):

if array1\_bitmask[index] == array2\_bitmask[index] == 1:

differ\_array.remove(universum[index])

return differ\_array

#ФУНКЦИЯ ВОЗВРАТА РЕЗУЛЬТАТА ВЫПОЛНЕНИЯ ОПЕРАЦИИ ПО ВАРИАНТУ

#(A \ B) \ C

def lab\_operation(array1, array2, array3, universum):

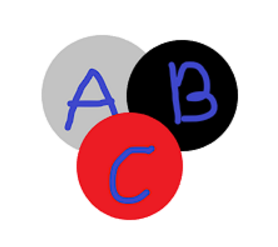
return array\_differ(array1 = array\_differ(array1, array2, universum), array2 = array3, universum = universum)

#ЗАДАНИЕ 5 - ВЫЧИСЛИТЬ ПРОГРАММНО ВЫРАЖЕНИЕ СОГЛАСНО ВАРИАНТУ

print("РЕЗУЛЬТАТ ВЫЧИСЛЕНИЯ ВЫРАЖЕНИЯ:\n")

print(functions.lab\_operation(array1 = A, array2 = B, array3 = C, universum = universum))

Результат выполнения:

6. Указать примеры покрытий и разбиения множества A.

A = [1, 3, 5, 6, 7, 8]

Разбиение:

[] [1, 3, 5, 6, 7, 8]

[1] [3, 5, 6, 7, 8]

[3] [1, 5, 6, 7, 8]

[5] [1, 3, 6, 7, 8]

[6] [1, 3, 5, 7, 8]

[7] [1, 3, 5, 6, 8]

[8] [1, 3, 5, 6, 7]

[1, 3] [5, 6, 7, 8]

[1, 5] [3, 6, 7, 8]

[1, 6] [3, 5, 7, 8]

[1, 7] [3, 5, 6, 8]

[1, 8] [3, 5, 6, 7]

[3, 5] [1, 6, 7, 8]

[3, 6] [1, 5, 7, 8]

[3, 7] [1, 5, 6, 8]

[3, 8] [1, 5, 6, 7]

[5, 6] [1, 3, 7, 8]

[5, 7] [1, 3, 6, 8]

[5, 8] [1, 3, 6, 7]

[6, 7] [1, 3, 5, 8]

[6, 8] [1, 3, 5, 7]

[7, 8] [1, 3, 5, 6]

[1, 3, 5] [6, 7, 8]

[1, 3, 6] [5, 7, 8]

[1, 3, 7] [5, 6, 8]

[1, 3, 8] [5, 6, 7]

[1, 5, 6] [3, 7, 8]

[1, 5, 7] [3, 6, 8]

[1, 5, 8] [3, 6, 7]

[1, 6, 7] [3, 5, 8]

[1, 6, 8] [3, 5, 7]

[1, 7, 8] [3, 5, 6]

Примеры покрытий:

[1] [3, 5, 6, 7, 8]

[1] [1, 3, 5, 6, 7, 8]

[1, 3] [1, 3, 5, 6, 7, 8]

[1, 3] [5, 6, 7, 8]

[1, 3] [1, 5, 6, 7, 8]

[1, 3] [3, 5, 6, 7, 8]

7. Реализовать программно алгоритм разбиения множеств

Код программы:

#ФУНКЦИЯ РАЗБИЕНИЯ МНОЖЕСТВА

def partition(array):

if len(array) == 1:

yield [array]

return

first = array[0]

for smaller in partition(array[1:]):

for n, subset in enumerate(smaller):

yield smaller[:n] + [[first] + subset] + smaller[n+1:]

yield [[first]] + smaller

#ФУНКЦИЯ ВЫВОДА РАЗБИЕНИЯ

def partition\_output(array):

print("\nРАЗБИЕНИЕ МНОЖЕСТВА:\n")

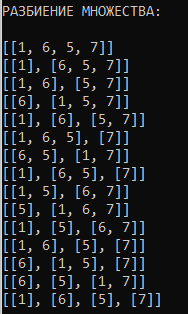
for subset in partition(array = array):

print(subset)

#ЗАДАНИЕ 7 - РЕАЛИЗОВАТЬ ПРОГРАММНО АЛГОРИТМ РАЗБИЕНИЯ МНОЖЕСТВ

functions.partition\_output(array = [1, 6, 5, 7])

Результат выполнения:



8. Построить код Грея для множества

def gray\_code(array):

gray\_code = []

print("\nБИНАРНЫЙ КОД ГРЕЯ ДЛЯ КАЖДОГО ЭЛЕМЕНТА МНОЖЕСТВА\n")

for element in array:

element\_to\_gray = element ^ (element >> 1)

while(element\_to\_gray > 0):

gray\_code = [element\_to\_gray % 2] + gray\_code

element\_to\_gray = element\_to\_gray // 2

if len(gray\_code) < 4:

for i in range(len(gray\_code), 4):

gray\_code = [0] + gray\_code

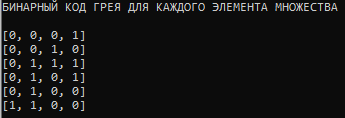
print(gray\_code)

gray\_code = []

#ЗАДАНИЕ 8 - ПОСТРОИТЬ КОД ГРЕЯ ДЛЯ МНОЖЕСТВА

functions.gray\_code(array = A)

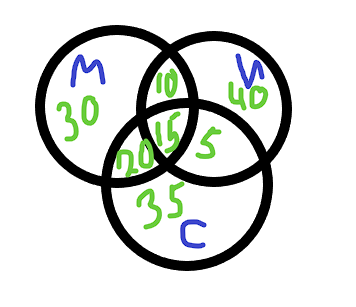
Результат выполнения:



9. Составить таблицу для различных сортировок (быстрой, слиянием, выбором, вставки, пятая на выбор) с указанием «О большого», достоинств и недостатков данного алгоритма сортировки.

|  |  |  |  |
| --- | --- | --- | --- |
| Сортировка | Преимущества | Недостатки | О |
| Быстрая | - при частичной упорядоченности повышаются шансы разделения массива на более равные части  - одна из самых быстрых внутренних сортировок | - использует доп. память  - неустойчивая | Л: O(nlogn)  С: O(nlogn)  Х: O(n2) |
| Слиянием | - устойчивая  - быстрая | - использует доп. память  - если входная последовательность почти отсортирована время работы то же | Л: O(nlogn)  С: O(nlogn)  Х: O(nlogn) |
| Выбором | - устойчивая  - не использует доп. память | - если входная последовательность почти отсортирована, количество сравнений то же | Л: O(n2)  С: O(n2)  Х: O(n2) |
| Вставками | - устойчивая  - эффективна на небольших наборах данных  - эффективна на частично-от-сортированных наборах данных  - не использует доп. память | - высокая вычислительная сложность | Л: O(n)  С: O(n2)  Х: O(n2) |
| Пирамидальная | - быстрая  - не использует доп. память | - неустойчивая  - если входная последовательность почти отсортирована, количество сравнений то же | Л: O(nlogn)  С: O(nlogn)  Х: O(nlogn) |

Задание 2.



В группе из 200 студентов 75 изучают математику, 70 - историю, 75 -

социологию, 35 изучают математику и социологию, 20 - историю и

социологию, 25 изучают математику и историю, 15 студентов - все

три предмета.

а) Сколько студентов изучают хотя бы один из трех предметов?

Ответ:155

б) Сколько студентов изучают только один из трех предметов?

Ответ: 105

в) Сколько студентов изучают историю или математику, но не

изучают социологию?

Ответ: 80

г) Сколько студентов не изучают ровно два из трех предметов?

Ответ: 35

д) Сколько студентов не выбрали историю или математику?

Ответ: 130

Вывод: В ходе выполнения данной лабораторной работы изучил основы работы с множествами.