**1. Методические указания: Операции над множествами**

Теоретическая часть

**1. Основные операции:**

- Объединение (A ∪ B): Все элементы из A и B без повторений.
- Пересечение (A ∩ B): только общие элементы.
- Разность (A \ B): Элементы из A, которых нет в B.
- Симметрическая разность (A Δ B): Элементы, принадлежащие только A или только B.

**2. Свойства операций:**

- Ассоциативность: `(A ∪ B) ∪ C = A ∪ (B ∪ C)`
- Дистрибутивность: `A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)`

**Практическая часть**

Шаг 1: Реализация на Python

python

A = {1, 2, 3, 4, 5}

B = {3, 4, 5, 6, 7}

C = {5, 6, 7, 8, 9}

**Объединение**

union = A.union(B, C)  # или A | B | C

**Пересечение**

intersection = A.intersection(B, C)  # или A & B & C

**Разность**

diff = A.difference(B)  # или A - B

**Симметрическая разность**

sym_diff = B.symmetric_difference(C)  # или B ^ C

**Шаг 2: Проверка свойств**

```python

# Ассоциативность
```

```
left = (A | B) | C
right = A | (B | C)
print(left == right)  # Должно быть True

# Дистрибутивность
left = A & (B | C)
right = (A & B) | (A & C)
print(left == right)  # Должно быть True
```

**Шаг 3: Визуализация (диаграммы Эйлера-Венна)**

Используйте библиотеку `matplotlib_venn`:

```python
from matplotlib_venn import venn3
venn3([A, B, C], ('A', 'B', 'C'))
plt.show()
```

---

### **2. Методические указания: Логические функции**

#### **Теоретическая часть**

1. **Таблица истинности**: Построить для всех комбинаций `x, y, z`.
2. **Нормальные формы**:
   - **СДНФ**: Дизъюнкция конъюнкций, где функция = 1.
   - **СКНФ**: Конъюнкция дизъюнкций, где функция = 0.
3. **Упрощение**: Использовать законы де Моргана, дистрибутивность и т.д.

#### **Практическая часть**

**Шаг 1: Таблица истинности**

| x | y | z | ¬z | x∧y | (x∧y)∨¬z | F |
|---|---|---|----|-----|----------|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... |

**Шаг 2: СДНФ и СКНФ**

Для `F = (x∧y)∨¬z`:

- **СДНФ**: `(¬x∧¬y∧¬z) ∨ (¬x∧y∧¬z) ∨ (x∧¬y∧¬z) ∨ (x∧y∧¬z) ∨ (x∧y∧z)`
- **СКНФ**: `(x∨y∨¬z) ∧ (x∨¬y∨¬z) ∧ (¬x∨y∨¬z)`

**Шаг 3: Упрощение**

```python
F = (x and y) or (not z)  # Упрощается до F = ¬z ∨ (x∧y)
```

**Шаг 4: Программа на Python**

```python
def F(x, y, z):
    return (x and y) or (not z)

# Пример вызова
print(F(True, False, True))  # False
```

---

### **3. Методические указания: Алгоритмы на графах**

#### **Теоретическая часть**

1. **Обход графа**:
   - **DFS (в глубину)**: Рекурсия или стек.
   - **BFS (в ширину)**: Очередь.
2. **Алгоритм Дейкстры**: Кратчайший путь из одной вершины во все остальные.
3. **Двудольность**: Граф можно раскрасить в 2 цвета.

#### **Практическая часть**

**Шаг 1: Реализация графа**

```python
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D'],
    'C': ['A', 'D', 'E'],
    'D': ['B', 'C', 'E'],
    'E': ['C', 'D']
}
```

**Шаг 2: DFS и BFS**

```python
from collections import deque

def DFS(graph, start):
    visited = []
    stack = [start]
    while stack:
        node = stack.pop()
        if node not in visited:
            visited.append(node)
```

```
            stack.extend(reversed(graph[node]))
    return visited

def BFS(graph, start):
    visited = []
    queue = deque([start])
    while queue:
        node = queue.popleft()
        if node not in visited:
            visited.append(node)
            queue.extend(graph[node])
    return visited
```

**Шаг 3: Дейкстра (для взвешенного графа)**

```python
import heapq

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    heap = [(0, start)]

    while heap:
        current_dist, node = heapq.heappop(heap)
        for neighbor, weight in graph[node].items():
            distance = current_dist + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(heap, (distance, neighbor))
```