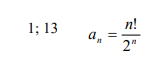
ЛР 2 – Фролов Глеб

**№1**  
. Разработать программу анализа арифметического выражения (рис.8). Программа должна: а) напечатать приглашение для ввода строки; прочитать с клавиатуры строку, введенную пользователем; б) с помощью стека проанализировать правильность расстановки круглых скобок: если в строке встретилась открывающая скобка, то записать ее в стек; если встретилась закрывающая скобка, то извлечь один символ из стека; операции записи и извлечения символа из стека реализовать в виде двух функций: PUSH и POP. Функция PUSH принимает символ и возвращает код возврата (0 - норма, 1 - переполнение стека), функция POP возвращает символ с вершины стека и код возврата (0 - норма, 1 - стек пуст); в) напечатать сообщение о правильности или ошибочности введенной строки; г) повторять действия, описанные в пп. а) - в), до тех пор, пока пользователь не введет пробел.

**№2**  
Написать программу для вычисления выражения an по формуле варианта, соответствующего номеру ЭВМ (варианты заданий). Вычисления организовать в виде рекурсивной функции. Программу выполнить по шагам, записать в конспекте последовательное изменение стека  
**изменение стека при выполнении программы для значения n = 4.**

Шаги:

1. Вызов `main()`

- Стек: `main()`

2. Ввод значения n пользователем

- Стек: `main()`

3. Вызов `calculate\_an(n)`

- Стек: `main(), calculate\_an(4)`

4. Рекурсивный вызов `calculate\_an(n - 1)` и `factorial(n)`

- Стек: `main(), calculate\_an(4), calculate\_an(3), factorial(4)`

5. Рекурсивный вызов `calculate\_an(n - 1)` и `factorial(n)`

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2), factorial(4), factorial(3)`

6. Рекурсивный вызов `calculate\_an(n - 1)` и `factorial(n)`

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2), calculate\_an(1), factorial(4), factorial(3), factorial(2)`

7. Рекурсивный вызов `calculate\_an(n - 1)` и `factorial(n)`

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2), calculate\_an(1), calculate\_an(0), factorial(4), factorial(3), factorial(2), factorial(1)`

8. Возврат значения выражения an

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2), calculate\_an(1), calculate\_an(0), factorial(4), factorial(3), factorial(2)`

9. Возврат значения выражения an

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2), calculate\_an(1), calculate\_an(0), factorial(4), factorial(3)`

10. Возврат значения выражения an

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2), calculate\_an(1), calculate\_an(0), factorial(4)`

11. Возврат значения выражения an

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2), calculate\_an(1), calculate\_an(0)`

12. Возврат значения выражения an

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2), calculate\_an(1)`

13. Возврат значения выражения an

- Стек: `main(), calculate\_an(4), calculate\_an(3), calculate\_an(2)`

14. Возврат значения выражения an

- Стек: `main(), calculate\_an(4), calculate\_an(3)`

15. Возврат значения выражения an

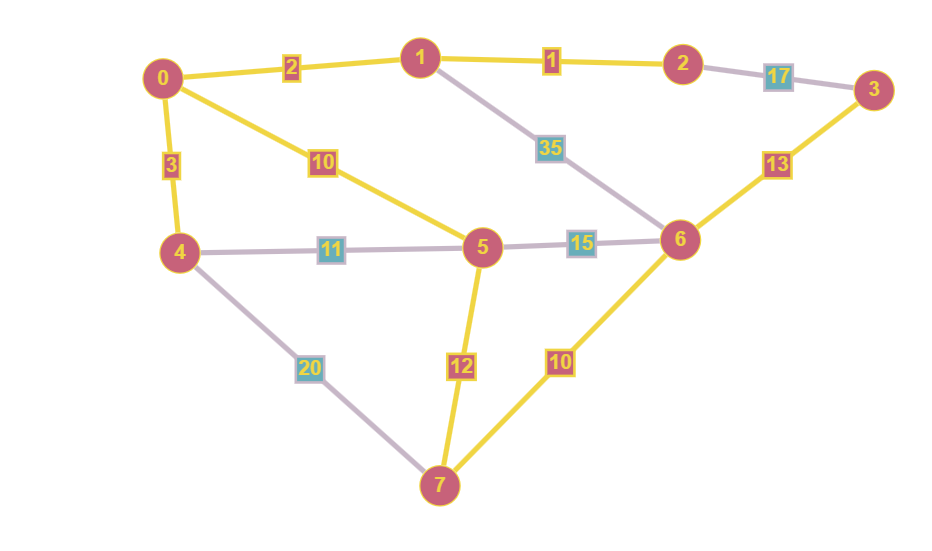
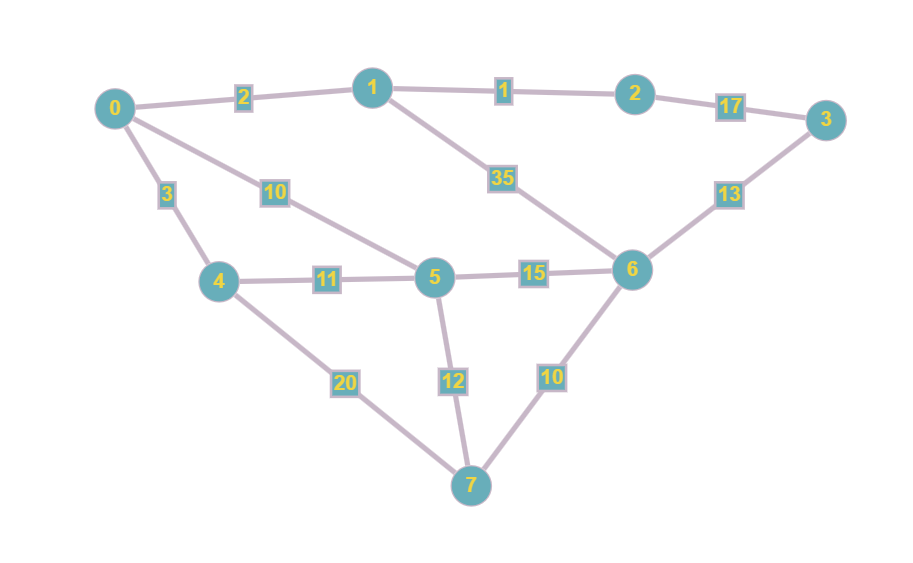
- Стек: `main(), calculate\_an(4)`

16. Возврат значения выражения an

- Стек: `main()`

**№3**

Составить программы для исследования методов Крускала и Прима

**-МИН ОСТ**

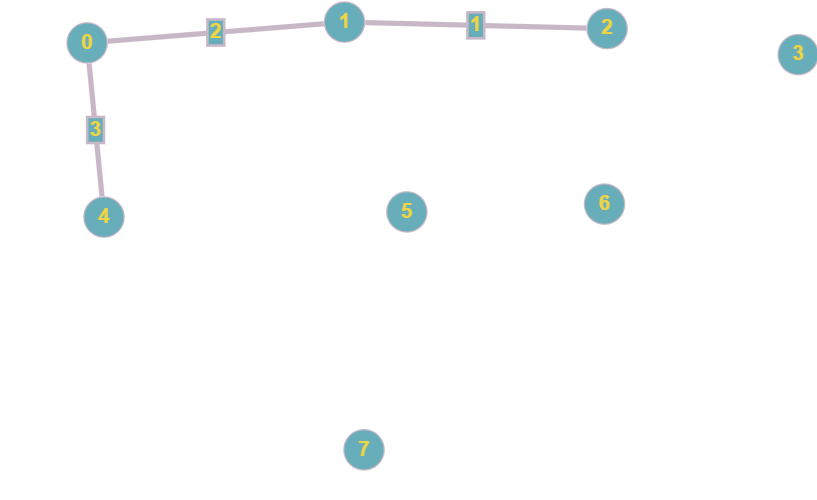
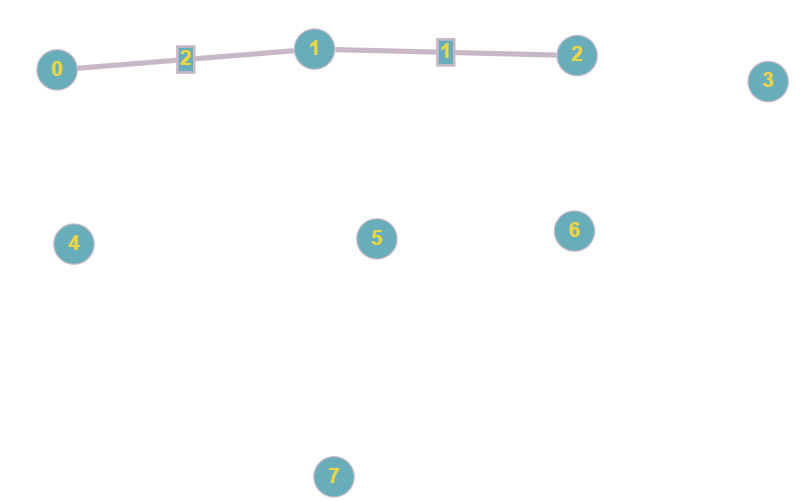
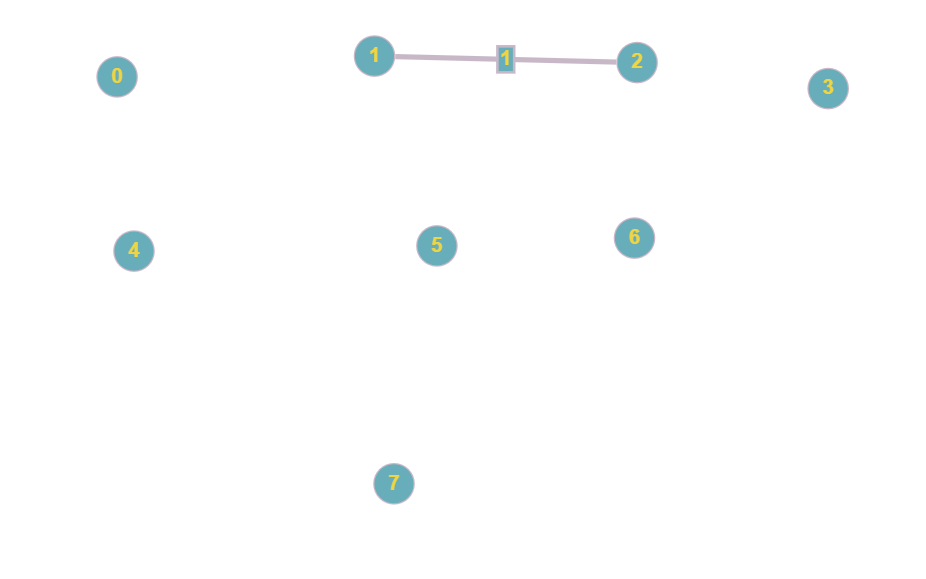
Метод Крускала

Отсортируем ребра в порядке возрастания:

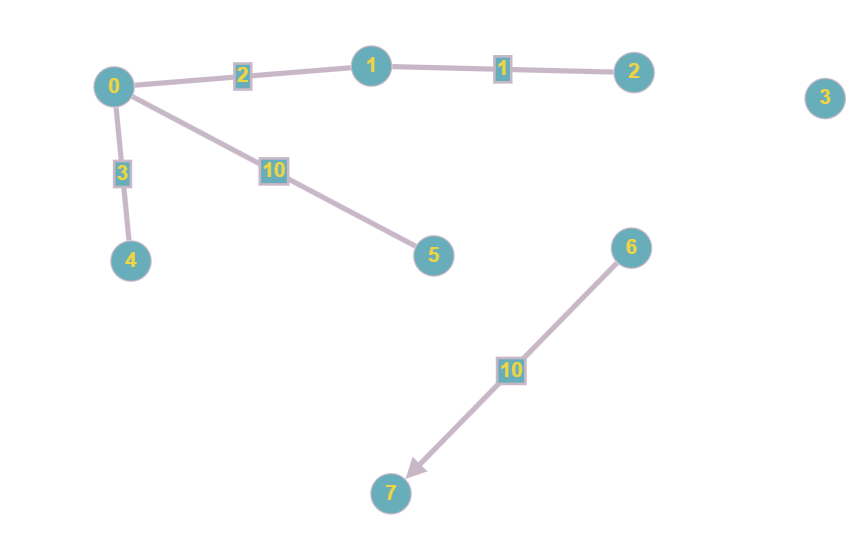
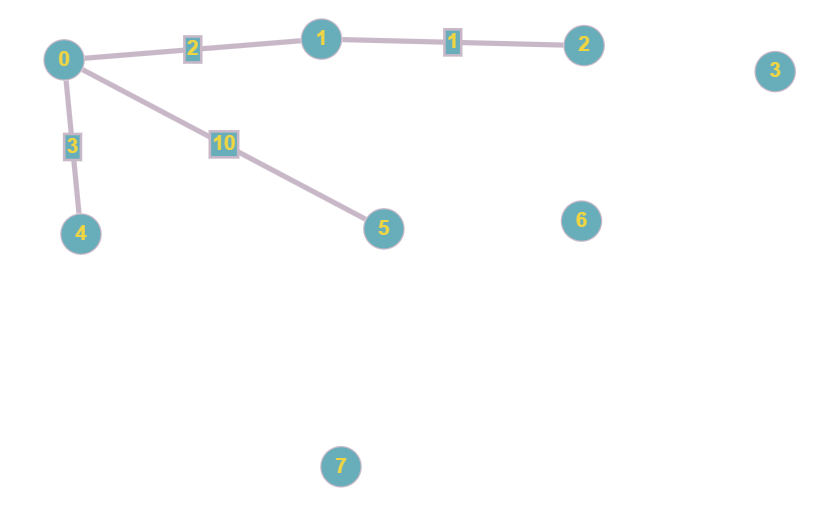
1: |1-2| 10: |0-5| 12: |5-7| 17: |2-3|  
2: |0-1| 10: |6-7| 13: |3-6| 20: |4-7|

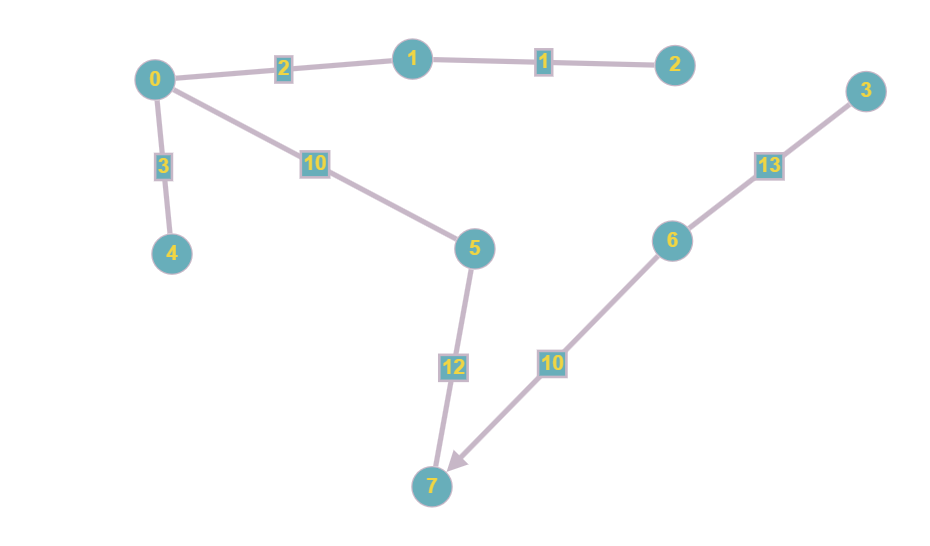
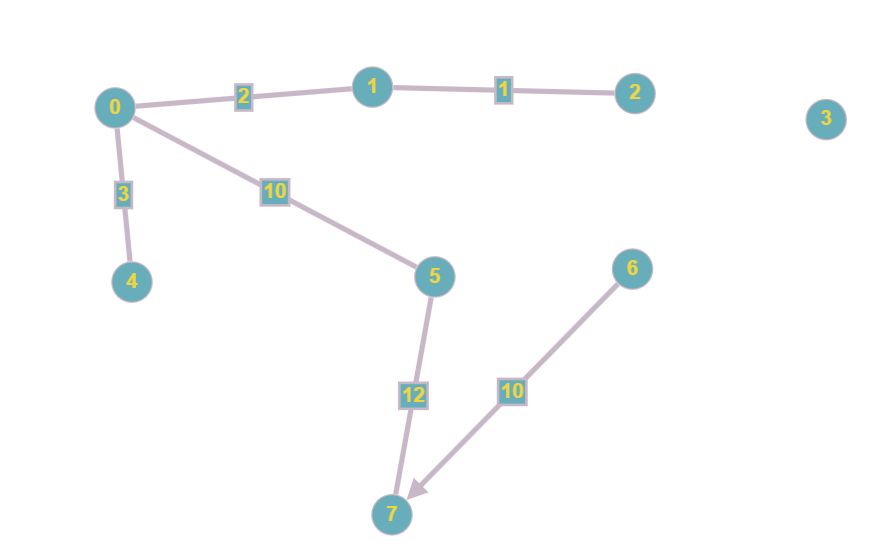
3: |0-4| 11: |4-5| 15: |5-6| 35: |1-6|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шаг 1: 1: |1-2| Шаг 2: 2: |0-1| Шаг 3: 3: |0-4|



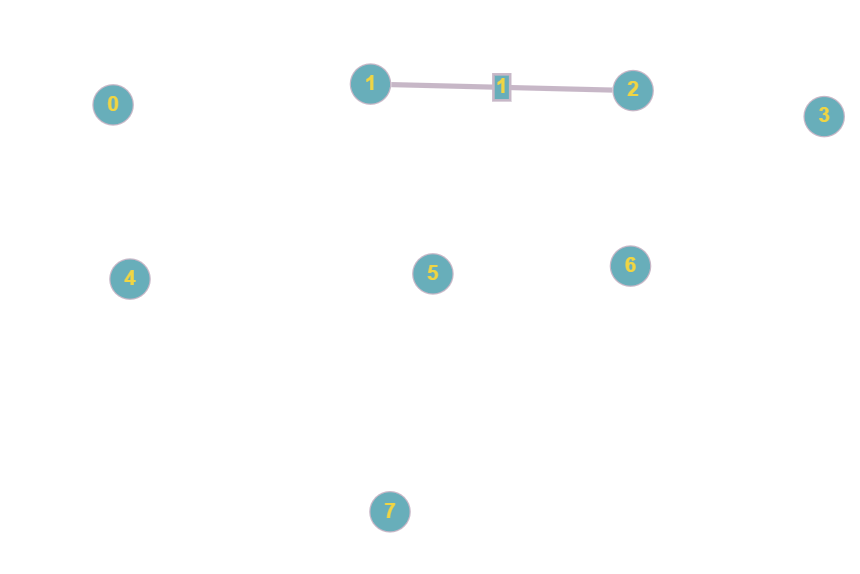
Шаг 4: 10: |0-5| Шаг 5: 10: |6-7|

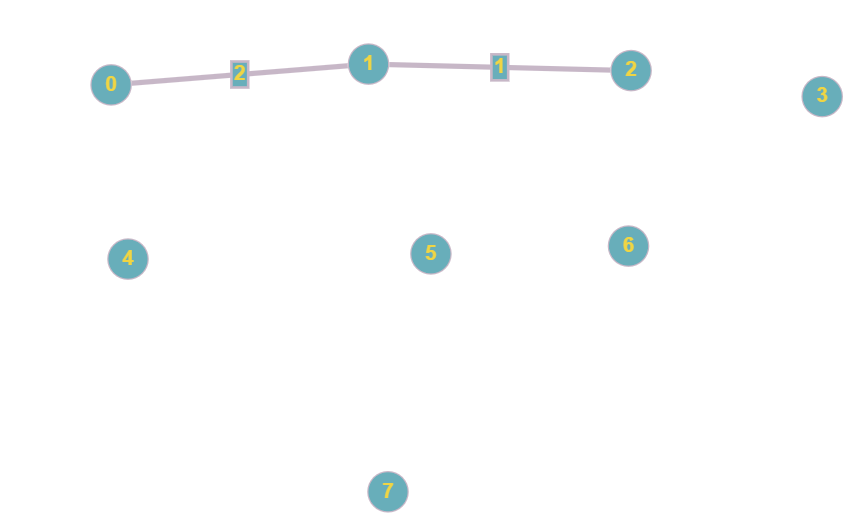


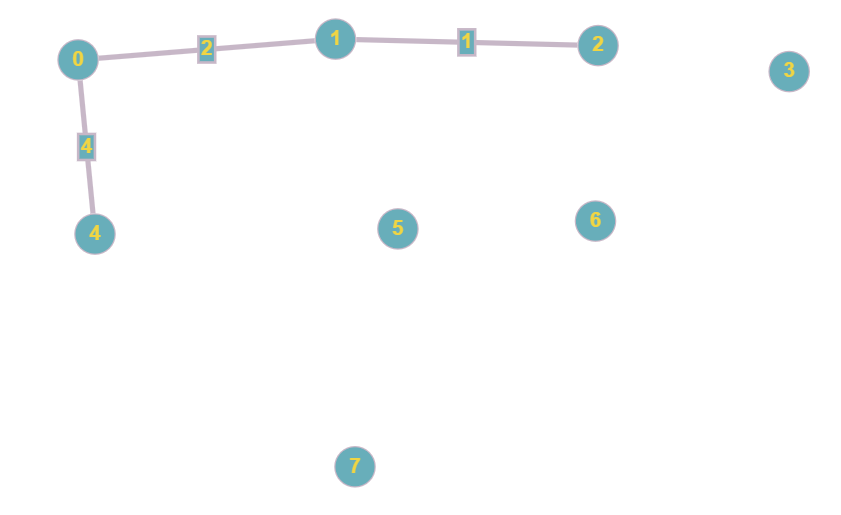
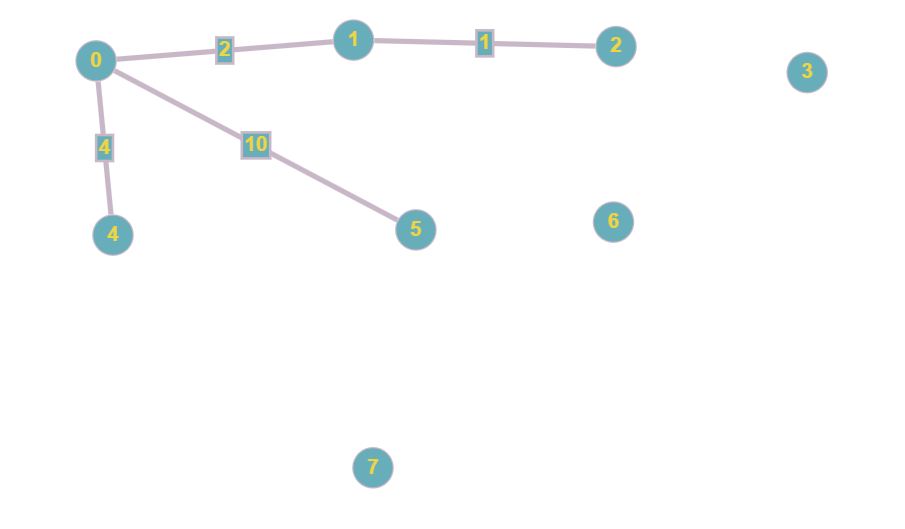
Шаг 6: 11: |4-5|- этот ребро добавлять нельзя   
Шаг 7: 12: |5-7| Шаг 8 13: |3-6|   
  
Поиск МОД закончен так как все компонент связности – 1; вес=51

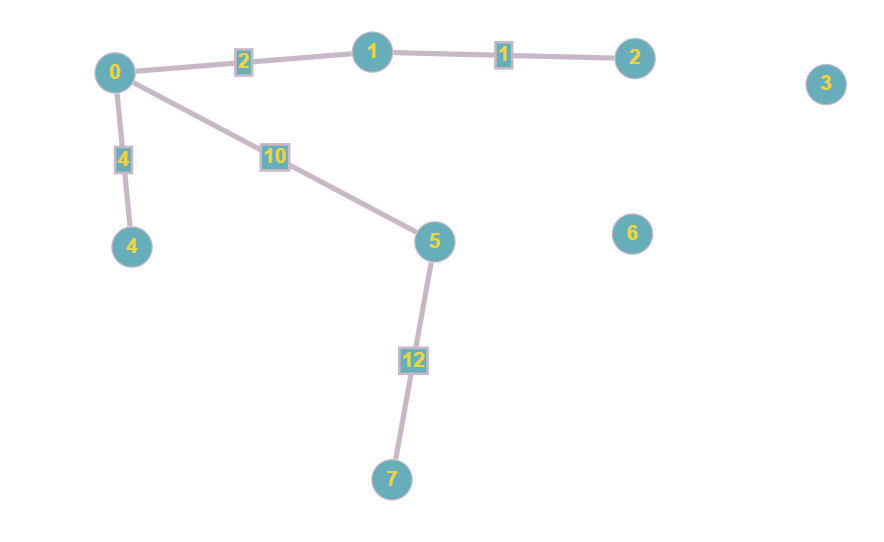
Метод Прима:

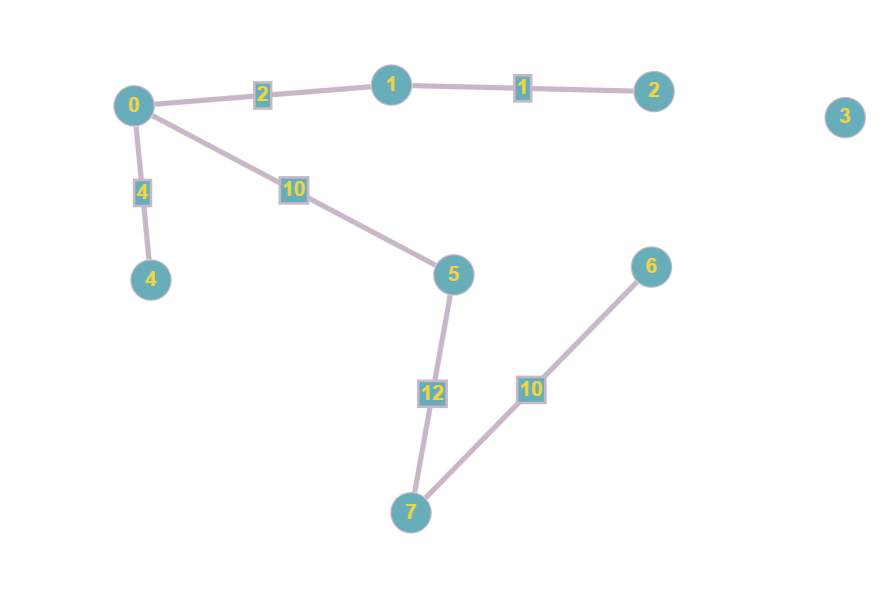
Шаг 1-найдем минимальное по весу ребро - 1: |1-2|

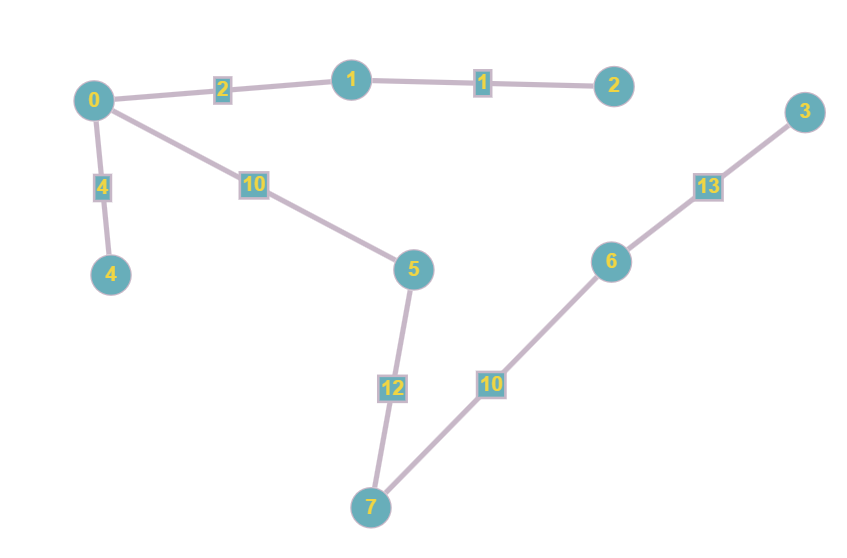


Шаг 2 – минимальное ребро инцидентное взятым вершинам (1, 2)- 2: |0-1|  


Шаг 3 – минимальное ребро инцидентное взятым вершинам (0, 1, 2)- 3: |0-4|  
  
Шаг 4 – минимальное ребро инцидентное взятым вершинам (0, 1, 2, 4)- 10: |0-5|  


Шаг 5 – минимальное ребро инцидентное взятым вершинам (0, 1, 2, 4, 5)- 12: |5-7|  


Шаг 6 – минимальное ребро инцидентное взятым вершинам (0, 1, 2, 4, 5, 7)- 10: |6-7|  


Шаг 7 – минимальное ребро инцидентное взятым вершинам (0, 1, 2, 4, 5, 6, 7)- 13: |3-6|  
  
Поиск МОД закончен так как все компонент связности – 1; вес=51

1. Под "остовным деревом" обычно понимается частичное поддерево ориентированного или неориентированного графа, которое содержит все вершины графа и является деревом (то есть не содержит циклов). Остовное дерево может быть использовано для моделирования различных систем, например, сетей передачи данных или древовидных структур данных.

2. Методы Крускала и Прима являются двумя популярными алгоритмами для построения минимального остовного дерева в связном взвешенном графе. Они оба используют жадный подход, но есть некоторые отличия:

- \*\*Метод Крускала\*\*:

- Начинает с пустого остовного дерева и последовательно добавляет самое короткое ребро, которое не создаст цикл в дереве.

- Рёбра сортируются по весу, а затем добавляются к текущему остовному дереву, если они не образуют цикл.

- Алгоритм завершается, когда все вершины соединены.

- Метод Крускала применяется, когда граф представлен в виде списка рёбер.

- \*\*Метод Прима\*\*:

- Начинает с одной произвольной вершины и постепенно строит остовное дерево, добавляя к нему рёбра с наименьшим весом, которые соединяют вершину дерева с вершиной, не входящей в дерево.

- В каждом шаге выбирается вершина с наименьшим весом ребра, инцидентного текущему остовному дереву.

- Алгоритм завершается, когда все вершины соединены.

- Метод Прима часто применяется, когда граф представлен в виде списка смежности.

Оба алгоритма гарантируют построение минимального остовного дерева, однако выбор между ними может зависеть от представления графа и других факторов, таких как эффективность выполнения.