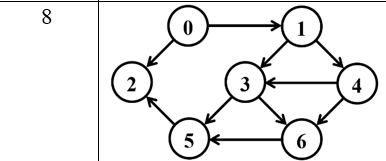
**Каракозова Регина, 2 ФИТ 7/2**

**Лабораторная работа №6. Алгоритмы на графах**

**Цель работы:** освоить сущность и программную реализацию: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов. Разобрать алгоритм Прима и алгоритм Крускала

**Задание 1.** Ориентированный граф G взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.







**Матрица смежности**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **V/V** | **V0** | **V1** | **V2** | **V3** | **V4** | **V5** | **V6** |
| **V0** | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **V1** | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| **V2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **V3** | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **V4** | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **V5** | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **V6** | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

**Матрица инцидентности**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **V/E** | **E0** | **E1** | **E2** | **E3** | **E4** | **E5** | **E6** | **E7** | **E8** | **E9** |
| **V0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **V1** | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **V2** | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 |
| **V3** | 0 | 0 | 0 | -1 | -1 | 0 | 1 | 1 | 0 | 0 |
| **V4** | 0 | 0 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **V5** | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | -1 | 1 |
| **V6** | 0 | 0 | 0 | 0 | 0 | -1 | 0 | -1 | 1 | 0 |

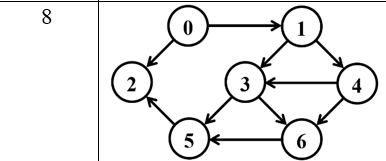
**Список смежных вершин**

S0={1, 2}; S1={3, 4}; S2={}; S3={5, 6}; S4={3, 6}; S5={2}; S6={5};

**Задание 2.** Осуществить алгоритмы поиска в ширину и глубину, а также алгоритма топологической сортировки аналогично примерам, рассмотренным на лекциях. Оформить отчет, включив в него каждый шаг выполнения алгоритмов.

**Алгоритм поиска в ширину**

*Алгоритм позволяет найти кратчайший (содержащий наименьшее число ребер) путь из одной вершины графа до всех остальных вершин. В нем сначала посещаются все вершины, смежные с текущей, а затем их потомки.*



**1.** Выбираем стартовую точку:0

Добавляем её в очередь

**Очередь:**0

**Посещенная очередь**: нет

**2.** Извлекаем из очереди вершину V0 и заносим в очередь посещения. Смотрим смежных вершин V0. Вершины: 1, 2. Добавляем их в очередь.

**Очередь:**1, 2

**Посещенная очередь**: 0

**3.** Извлекаем из очереди вершину V1 и заносим в очередь посещения. Смотрим смежных вершин V1. Вершины: 3, 4. Добавляем их в очередь.

**Очередь:** 2, 3, 4

**Посещенная очередь**: 0, 1

**4.** Извлекаем из очереди вершину V2 и заносим в очередь посещения. Смотрим смежных вершин V2. Нет смежных вершин. Продолжаем.

**Очередь:** 3, 4

**Посещенная очередь**: 0, 1, 2

**5.** Извлекаем из очереди вершину V3 и заносим в очередь посещения. Смотрим смежных вершин V3. Вершины: 5, 6. Добавляем их в очередь.

**Очередь:** 4, 5, 6

**Посещенная очередь**: 0, 1, 2, 3

**6.** Извлекаем из очереди вершину V4 и заносим в очередь посещения. Смотрим смежных вершин V4. Вершины: 3(уже в очереди- пропускаем), 6(уже в очереди- пропускаем).

**Очередь:** 5, 6,

**Посещенная очередь**: 0, 1, 2, 3, 4

**7.** Извлекаем из очереди вершину V5 и заносим в очередь посещения. Смотрим смежных вершин V5. Вершины:2 (уже в очереди- пропускаем).

**Очередь:** 6

**Посещенная очередь**: 0, 1, 2, 3, 4, 5

**8.** Извлекаем из очереди вершину V6 и заносим в очередь посещения. Смотрим смежных вершин V6. Вершины:5 (уже в очереди- пропускаем).

**Очередь:**

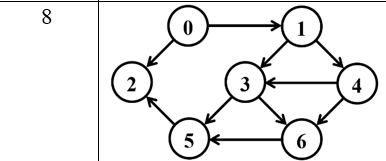
**Посещенная очередь**: 0, 1, 2, 3, 4, 5, 6

**9.** Все вершины пройдены. Конец алгоритма.

**Поиск в ширину: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6**

**Алгоритм поиска в глубину**

*В процессе поиска мы идем «вглубь» графа настолько, насколько возможно. Следуя алгоритму, мы последовательно обойдем все вершины графа, которые доступны из начальной вершины. Если ребро ведет в не пройдённую до этого момента вершину, то алгоритм запускается с нее. В случае если ребер, которые ведут в не рассмотренную вершину, больше нет, то происходит возврат назад.*



**1.** Создаем стек и помещаем в него начальную вершину 0. Помечаем вершину 0 как посещенную.

**Стек:** 0

**Посещенные вершины:** 0

**2.** У вершины 0 имеется 2 смежных вершины - 1, 2. Добавляем первую смежную вершину (1) в стек.

**Стек:** 0, 1

**Посещенные вершины:** 0

**3.** Помечаем вершину 1 как посещенную.

**Стек:** 0, 1

**Посещенные вершины:** 0, 1

**4.** У вершины 1 имеется 2 смежных вершины - 3, 4. Добавляем первую смежную вершину (3) в стек.

**Стек:** 0, 1, 3

**Посещенные вершины:** 0, 1

**5.** Помечаем вершину 3 как посещенную.

**Стек:** 0, 1,3

**Посещенные вершины:** 0, 1, 3

**6.** У вершины 3 имеется 2 смежных вершины - 5, 6. Добавляем первую смежную вершину (5) в стек.

**Стек:** 0, 1, 3, 5

**Посещенные вершины:** 0, 1

**7.** Помечаем вершину 5 как посещенную.

**Стек:** 0, 1, 3,5

**Посещенные вершины:** 0, 1, 3, 5

**8.** У вершины 5 имеется 1 смежных вершины - 2. Добавляем её в стек.

**Стек:** 0, 1, 3, 5, 2

**Посещенные вершины:** 0, 1, 3, 5

**9.** Помечаем вершину 2 как посещенную.

**Стек:** 0, 1, 3, 5, 2

**Посещенные вершины:** 0, 1, 3, 5, 2

**10.** У вершины 2 нет смежных вершин. Возвращаемся в вершину 3.

**Стек:**

**Посещенные вершины:** 0, 1, 3, 5, 2

**11.** У вершины 3 есть еще вершина 6. У вершины 6 нет не посещенных вершин. Возвращаемся в вершину 1 и добавляем в посещенные вершину 6

**Стек:**

**Посещенные вершины:** 0, 1, 3, 5, 2, 6.

**12.** У вершины 1 есть еще вершина 4. У вершины 4 нет не посещенных вершин. Возвращаемся в вершину 0 и добавляем в посещенные вершину 4

**Стек:**

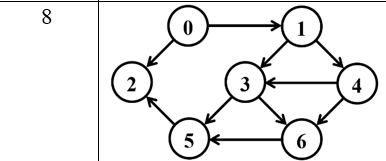
**Посещенные вершины:** 0, 1, 3, 5, 2, 6, 4.

**13.** У вершины 0 нет не посещенных смежных вершин. Конец алгоритма.

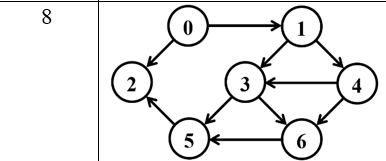
**Поиск в глубину: 0 -> 1 -> 3 -> 5 -> 2 -> 6 -> 4**

**Алгоритм топологической сортировки**

*Является упорядочиванием вершин бесконтурного ориентированного графа. Его цель состоит в том, чтобы перенумеровать вершины так, чтобы каждое ребро из вершины с меньшим номером вело в вершину с большим. Иными словами, нужно найти перестановку вершин, которая соответствует порядку, задаваемому всеми ребрами графа.*

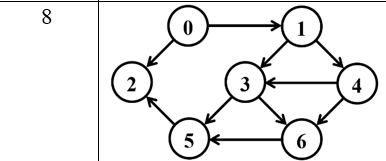


1. Выбираем вершину без исходящих дуг. Это вершина **2**.
2. Убираем все входящие в вершину 2 дуги.



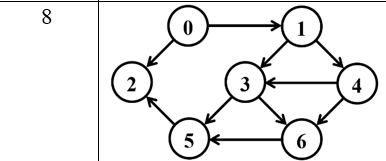


1. Снова выбираем вершину без исходящих дуг. Это вершина **5**.
2. Убираем все входящие в вершину 5 дуги.



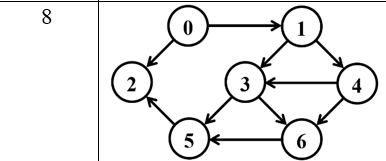


1. Снова выбираем вершину без исходящих дуг. Это вершина **6**.
2. Убираем все входящие в вершину 6 дуги.



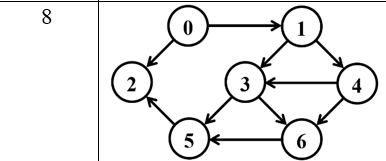


1. Снова выбираем вершину без исходящих дуг. Это вершина **3**.
2. Убираем все входящие в вершину 3 дуги.



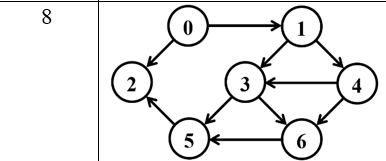


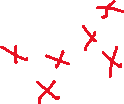
1. Снова выбираем вершину без исходящих дуг. Это вершина **4**.
2. Убираем все входящие в вершину 4 дуги.





1. Снова выбираем вершину без исходящих дуг. Это вершина **1**.
2. Убираем все входящие в вершину 1 дуги.

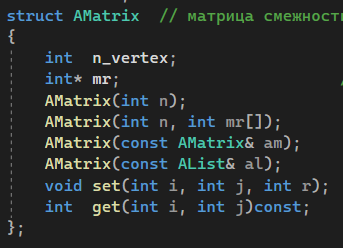




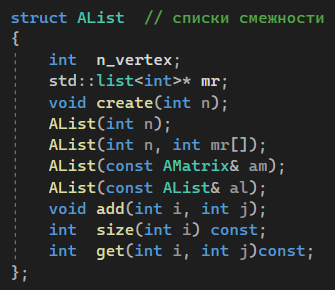
1. Дошли в вершину **0.** Это конечная точку.

**Топологическая сортировка: 2 -> 5 -> 6 -> 3 -> 4 -> 1 -> 0**

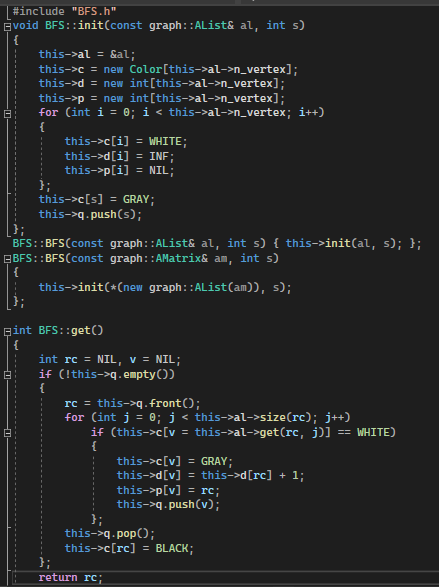
**Задание 3.** Осуществить программную реализацию алгоритмов на C++. Разработать структуры AMatrix и АList для представления ориентированного графа матричным и списковым способом. Разработать функции преобразования из одного способа представления в другой. Разработать функцию BFS обхода вершин графа, используя метод поиска в ширину. Продемонстрировать работу функции. Копии экрана вставить в отчет.



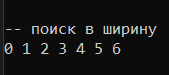
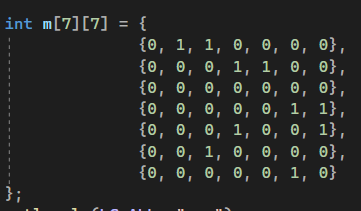
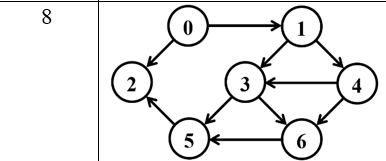
AMatrix



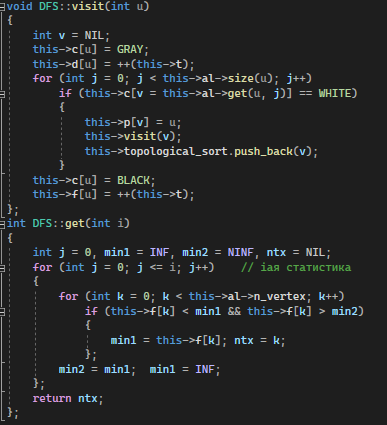
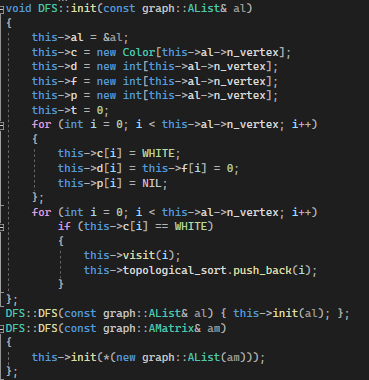
АList



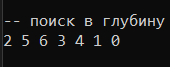
функция BFS



**Задание 4.** Разработать функцию DFS обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.



функция DFS



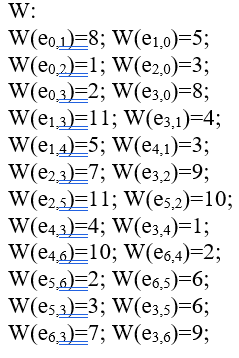
0-1-4-3-6-5-2

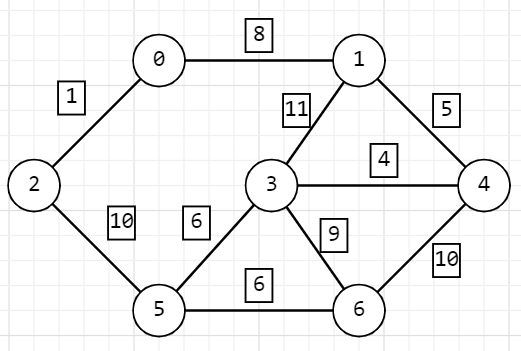
**Задание 5.** Доработайте функцию DFS, для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.



**Задание 6.** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Прима. Шаги построения отразить в отчете.

Веса ребер принять:





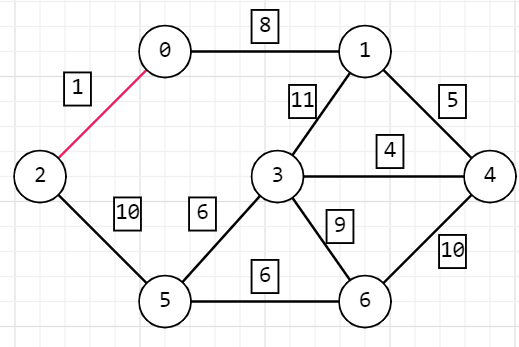
Для данного алгоритма не учитывается направление стрелок.

1. Выбираем стартовую вершину, пусть она будет 0. Добавляем вершину 0 в дерево.

**Дерево: {0}**

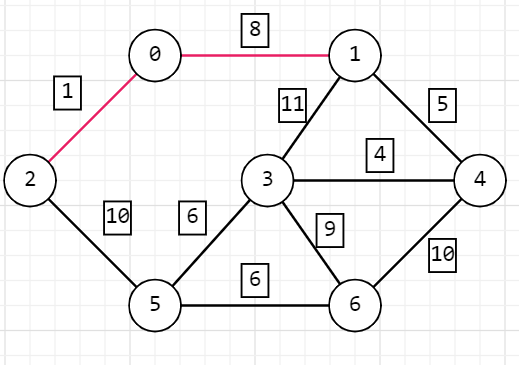
1. Находим все ребра, соединяющие вершину 0 с другими вершинами. Получаем (0,1) (0,2). Выбираем с меньшим весом: 2. Добавляем вершину 2 в дерево.

**Дерево: {0 2}**



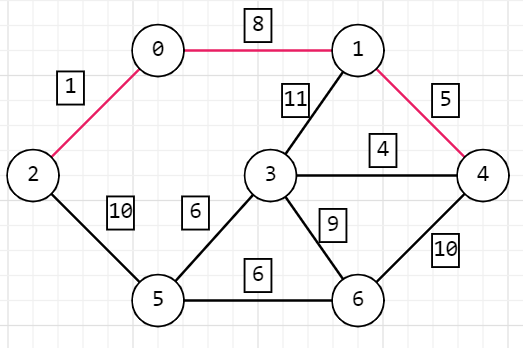
1. Находим все ребра, соединяющие вершину 0 и 2 с другими вершинами. Получаем (0,1) (0,2) (2,5). Выбираем с меньшим весом: 8. Добавляем вершину 1 в дерево.

**Дерево: {0 2 1}**



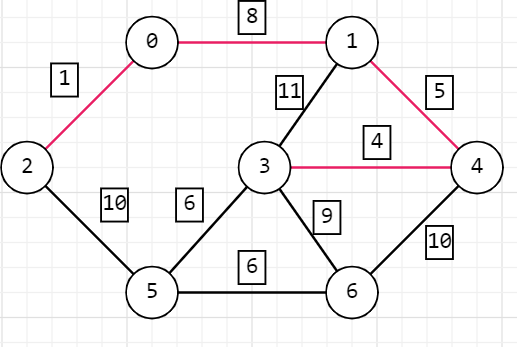
1. Находим все ребра, соединяющие вершину 0 и 2 и 1 с другими вершинами. Получаем (0,1) (0,2) (2,5) (1,4) (1,3). Выбираем с меньшим весом: 5. Добавляем вершину 4 в дерево.

**Дерево: {0 2 1 4}**



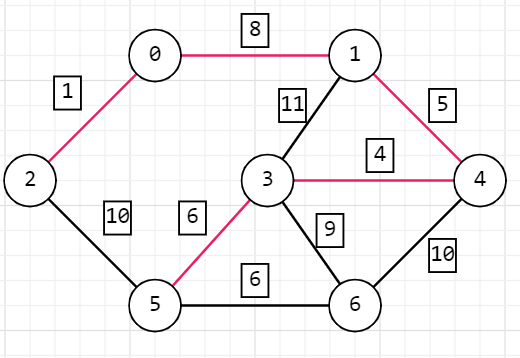
1. Находим все ребра, соединяющие вершину 0 и 2 и 1 и 4 с другими вершинами. Получаем (0,1) (0,2) (2,5) (1,4) (1,3) (4,3) (4,6). Выбираем с меньшим весом: 4. Добавляем вершину 3 в дерево.

**Дерево: {0 2 1 4 3}**



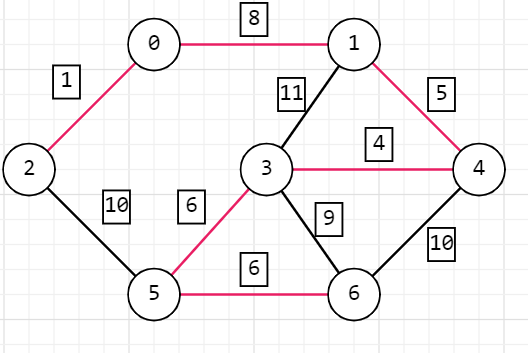
1. Находим все ребра, соединяющие вершину 0 и 2 и 1 и 4 и 3 с другими вершинами. Получаем (0,1) (0,2) (2,5) (1,4) (1,3) (4,3) (4,6) (3,5) (3,6). Выбираем с меньшим весом: 6. Добавляем вершину 5 в дерево.

**Дерево: {0 2 1 4 3 5}**



1. Находим все ребра, соединяющие вершину 0 и 2 и 1 и 4 и 3 и 5 с другими вершинами. Получаем (0,1) (0,2) (2,5) (1,4) (1,3) (4,3) (4,6) (3,5) (3,6) (5,6) (5,2). Выбираем с меньшим весом: 6. Добавляем вершину 6 в дерево.

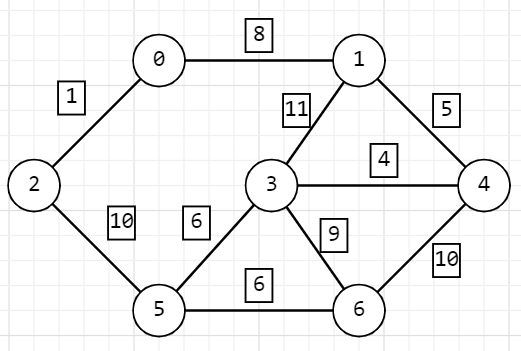
**Дерево: {0 2 1 4 3 5 6}**



Вес минимального остовного дерева равен 30.

**Задание 7.** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала. Шаги построения отразить в отчете.

*Мы сортируем по неубыванию веса всех ребер и в зависимости от того, где располагается ребро со следующим минимальным весом, мы можем либо продолжать строить текущее дерево, либо строим новое (его корень лежит в вершине, к которой ведет ребро с текущим минимальным весом), а затем их соединяем.*



1. Выбираем вершину с наименьшим прилагающимся весом. В нашем случае вершина 0. Ребро с наименьшим весом от этой вершины идет в вершину 2. Добавляем эти вершины в дерево. **Дерево 1: {0, 2}**
2. Ищем вершину с наименьшим весом :3 и 4. Создаем новое дерево.

**Дерево 1: {0, 2}**

**Дерево 2: {3, 4}**

1. Повторяем алгоритм. Следующее минимальное это 5 от верш 1 и 4. Добавляем к соответствующему дереву.

**Дерево 1: {0, 2}**

**Дерево 2: {3, 4, 1}**

1. Повторяем алгоритм. Следующее минимальное это 6 от верш 3 и 5. Добавляем к соответствующему дереву.

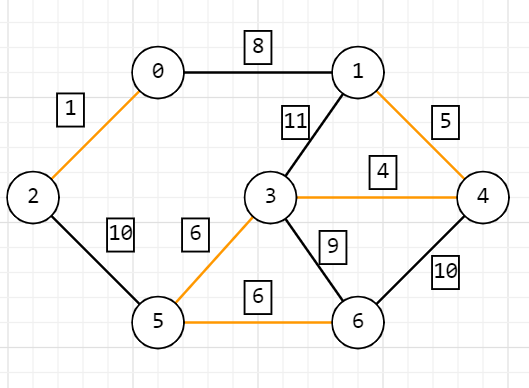
**Дерево 1: {0, 2}**

**Дерево 2: {3, 4, 1, 5}**

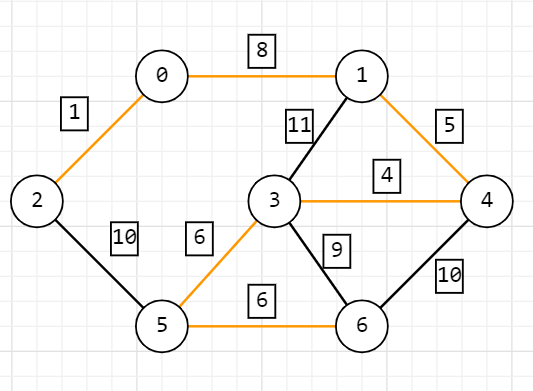
1. Повторяем алгоритм. Следующее минимальное это 6 от верш 5 и 6. Добавляем к соответствующему дереву.

**Дерево 1: {0, 2}**

**Дерево 2: {3, 4, 1, 5, 6}**



Потом мы соединяем два дерева при помощи ребра с наименьшим весом. Это вес равен 8, то есть соединяем вершину 0 и 1.



Вес минимального остовного дерева равен 30.

**Вывод**: в ходе лабораторной работы я освоила сущность и программную реализацию: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов. Разобрать алгоритм Прима и алгоритм Крускала.

**Вопросы для защиты**

1. **Какие представления графов Вы знаете?**

• Матрица смежности, где каждый элемент i,j матрицы обозначает наличие или отсутствие ребра между вершинами i и j.

• Список смежности, где каждая вершина графа представляется списком вершин, с которыми она соединена ребром.

• Матрица инцидентности, где каждый элемент i,j матрицы обозначает наличие или отсутствие ребра между вершиной i и ребром j.

1. **В чем заключается поиск в ширину? Где рационально его использовать?**

Поиск в ширину - это алгоритм обхода графа, который начинает с вершины и постепенно расширяет посещаемые вершины в порядке увеличения расстояния от исходной. Алгоритм использует очередь для хранения вершин, которые нужно посетить.

Поиск в ширину рационально использовать, когда нужно найти кратчайший путь между двумя вершинами, так как он гарантирует нахождение кратчайшего пути при условии, что ребра имеют одинаковый вес. Также алгоритм может использоваться для поиска всех вершин в графе, достижимых из заданной вершины.

1. **В чем заключается поиск в глубину? В каких ситуациях рационально его использовать?**

Поиск в глубину - это алгоритм обхода графа, который идет "вглубь" структуры, посещая как можно более глубокие узлы, пока не будет достигнута целевая вершина или пока все узлы не будут обойдены. Алгоритм используется в ситуациях, когда необходимо проверить наличие пути между двумя вершинами, проверить, является ли граф деревом, или выполнить другие задачи, связанные с обходом и анализом графов. Также поиск в глубину может использоваться для нахождения всех компонент связности в графе.

1. **В чем смысл топологической сортировки? Для чего она применяется?**

Топологическая сортировка - это алгоритм сортировки вершин в ориентированном ациклическом графе таким образом, что для каждой дуги (u, v) вершина u следует вершине v. Она используется для нахождения порядка выполнения задач, когда некоторые задачи могут быть выполнены только после завершения других задач. Топологическая сортировка применяется, например, при оптимизации компиляторов, планировании задач в проектах, управлении проектами и т.д.

1. **Что такое минимальное остовное дерево?**

Минимальное остовное дерево — это подграф связного неориентированного взвешенного графа, содержащий все его вершины и имеющий минимальную сумму весов ребер. Оно используется для решения задач, связанных с оптимизацией, например, для оптимизации системы передачи электроэнергии или связи в компьютерных сетях.

1. **В чем заключается стандартный алгоритм построения минимального остовного дерева?**

Стандартный алгоритм построения минимального остовного дерева основан на алгоритме Краскала или Прима. Алгоритм Краскала заключается в построении остовного дерева по шагам: сначала создается лес, состоящий из отдельных деревьев, затем на каждом шаге находится минимальное ребро, соединяющее два разных дерева, и эти деревья объединяются в одно. Алгоритм Прима начинается с выбора случайной вершины, затем на каждом шаге выбирается ближайшее к уже построенной части графа ребро и добавляется в остовное дерево. Оба алгоритма приводят к построению минимального остовного дерева.

1. **К какой категории алгоритмов относятся алгоритмы Прима и Крускала?**

К жадным алгоритмам. Они выбирают на каждом шаге ребро с минимальным весом из доступных, не проверяя при этом весь оставшийся граф на наличие более выгодных решений.

1. **Опишите один шаг алгоритма Крускала? Когда алгоритм прекращает свою работу?**

Шаг алгоритма Крускала состоит в выборе минимального ребра, которое соединяет две компоненты связности графа, и добавлении его в остовное дерево. Далее происходит объединение этих двух компонент в одну. Алгоритм прекращает работу, когда все вершины графа соединены в одну компоненту связности.