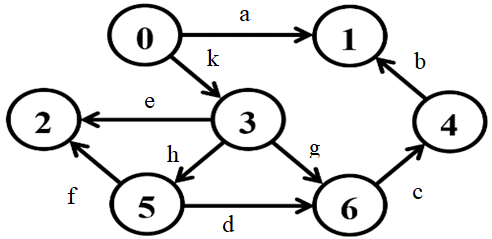
**Тараканов Никита**

**2-4 ПОИТ**

**Отчет**

**Вариант 13**

**Задание 1**

****

**Матрица смежности:**

Номер строки соответствует вершине, из которой идет ребро, а номер столбца соответствует вершине, в которую идет ребро.

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

**Матрица инцидентности:**

Строим матрицу вершин и ребер:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **d** | **e** | **f** | **h** | **g** | **k** |
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1** | -1 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | -1 | -1 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | -1 |
| **4** | 0 | 1 | -1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 1 | 0 | 1 | -1 | 0 | 0 |
| **6** | 0 | 0 | 1 | -1 | 0 | 0 | 0 | -1 | 0 |

**Список смежных вершин:**

[0]=[1, 3]

[1]=[]

[2]=[]

[3]=[5, 6]

[4]=[1]

[5]=[2, 6]

[6]=[4]

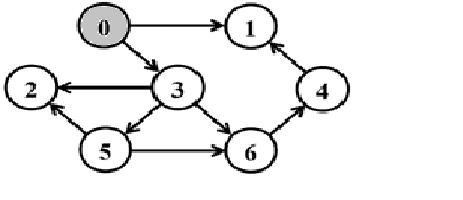
**Задание 2**

**Алгоритм поиска в ширину BFS(BREADTH-FIRST SEARCH)**

Алгоритм подразумевает, что задана исходная (стартовой) вершина, и основывается на простом правиле: при выборе очередной вершины предпочтение отдается ближайшей. При этом считается, что все дуги графа имеют единичную длину. Сначала посещается стартовая вершина, затем все вершины, смежные ей (т. е. находящиеся на расстоянии 1), после чего вершины, находящиеся на расстоянии 2 от стартовой и т.д.

Текущее состояние алгоритма хранится в следующих структурах памяти: **Q** – очередь вершин, **С** – массив окраски вершин, **D** – массив расстояний и **P** – массив предшествующих вершин.

Очередь **Q** (структура памяти, реализующая алгоритм «первый вошел − первый вышел»), используется для промежуточного хранения номеров вершин. На каждом шаге алгоритма, в очередь помещаются номера вершин в порядке их обнаружения. На каждом шаге, кроме первого, из очереди извлекается очередной номер вершины, подлежащей отметке о посещении. На первом шаге алгоритма в очередь помещается номер стартовой вершины. На последнем шаге очередь пуста.

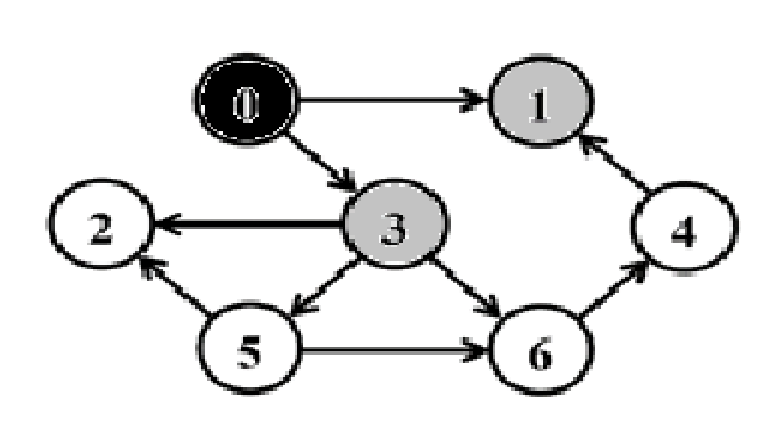


Q [0]

C [G] [W] [W] [W] [W] [W] [W]

D [0] [|] [|] [|] [|] [|] [|]

P [N] [N] [N] [N] [N] [N] [N]

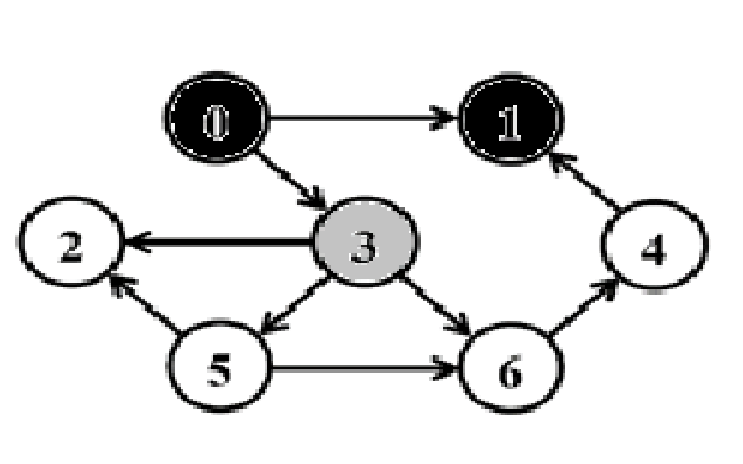


Q [1] [3]

C [B] [G] [W] [G] [W] [W] [W]

D [0] [1] [|] [1] [|] [|] [|]

P [N] [0] [N] [0] [N] [N] [N]

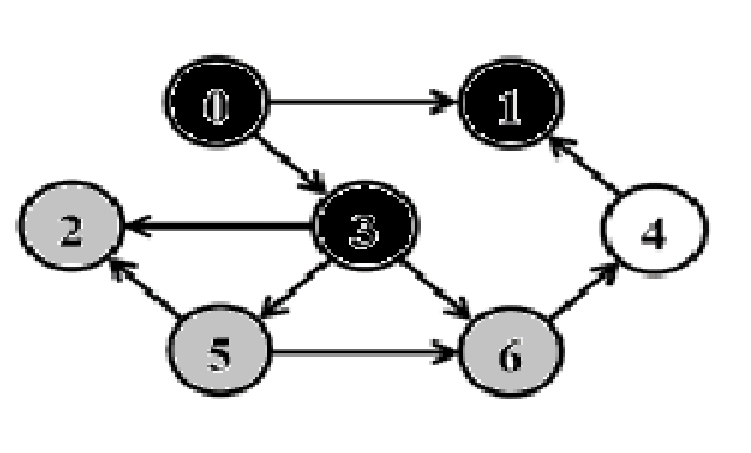


Q [3]

C [B] [B] [W] [G] [W] [W] [W]

D [0] [1] [|] [1] [|] [|] [|]

P [N] [0] [N] [0] [N] [N] [N]

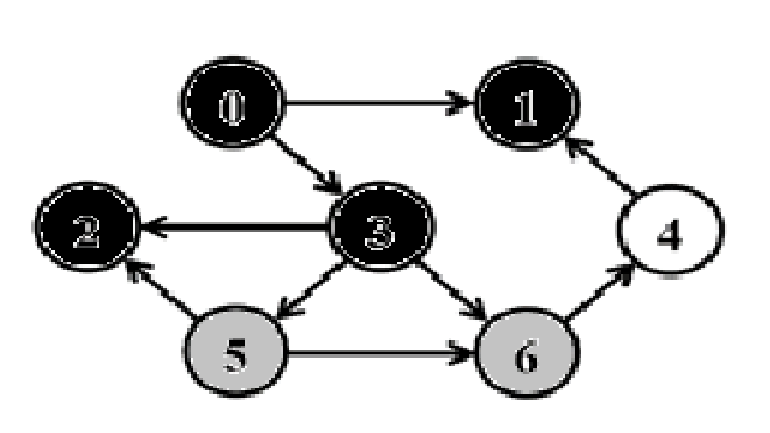


Q [2] [5] [6]

C [B] [B] [G] [B] [W] [G] [G]

D [0] [1] [2] [1] [|] [2] [2]

P [N] [0] [3] [0] [N] [3] [3]

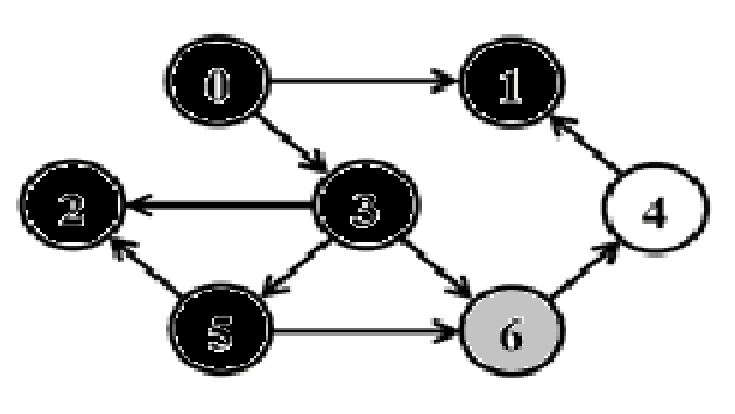


Q [5] [6]

C [B] [B] [B] [B] [W] [G] [G]

D [0] [1] [2] [1] [|] [2] [2]

P [N] [0] [3] [0] [N] [3] [3]

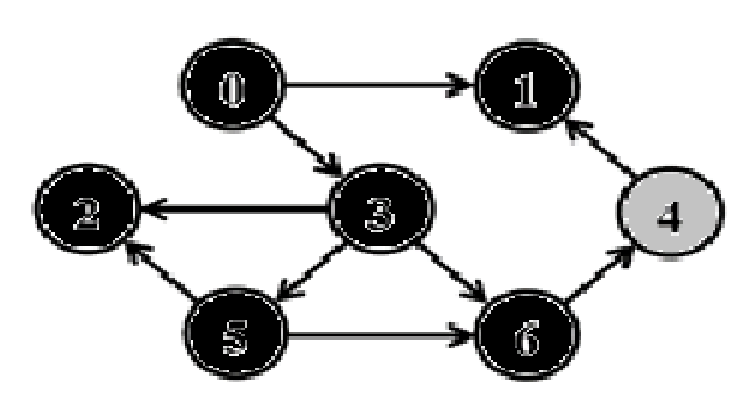


Q [6]

C [B] [B] [B] [B] [W] [B] [G]

D [0] [1] [2] [1] [|] [2] [2]

P [N] [0] [3] [0] [N] [3] [3]

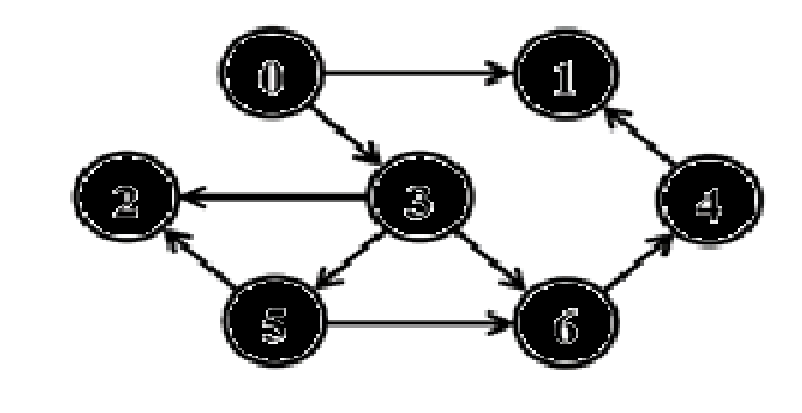


Q [4]

C [B] [B] [B] [B] [G] [B] [B]

D [0] [1] [2] [1] [3] [2] [2]

P [N] [0] [3] [0] [6] [3] [3]



Q []

C [B] [B] [B] [B] [G] [B] [B]

D [0] [1] [2] [1] [3] [2] [2]

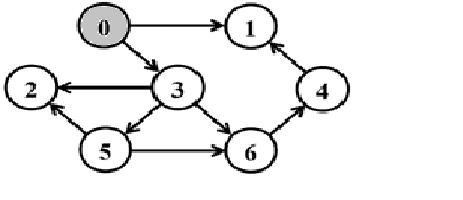
P [N] [0] [3] [0] [6] [3] [3]

Итоговый обход:

0 –> 1 -> 3 -> 2 -> 5 -> 6 -> 4

**Алгоритм обхода в глубину DFS(Depth-first search):**

Назначение и размерность массивов С (массив окраски вершин) и P (массив предшествующих вершин) такие же, как и в алгоритме BFS. В массиве D для каждой вершины записывается время обнаружения (шаг окраски в серый цвет). Массив F предназначен для хранения времени фиксации (шага окраски в черный цвет) вершины. Кроме того, используется переменная t, текущее значение которой – номер шага алгоритма.



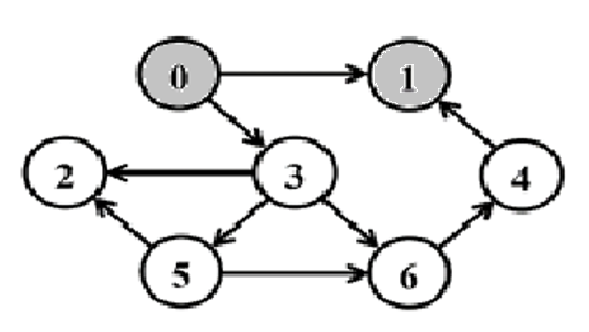
t 1

C [G] [W] [W] [W] [W] [W] [W]

D [1] [|] [|] [|] [|] [|] [|]

P [N] [N] [N] [N] [N] [N] [N]

F [0] [0] [0] [0] [0] [0] [0]



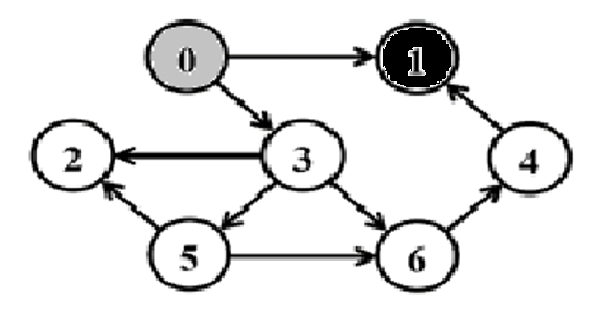
t 2

C [G] [G] [W] [W] [W] [W] [W]

D [1] [2] [|] [|] [|] [|] [|]

P [N] [0] [N] [N] [N] [N] [N]

F [0] [0] [0] [0] [0] [0] [0]



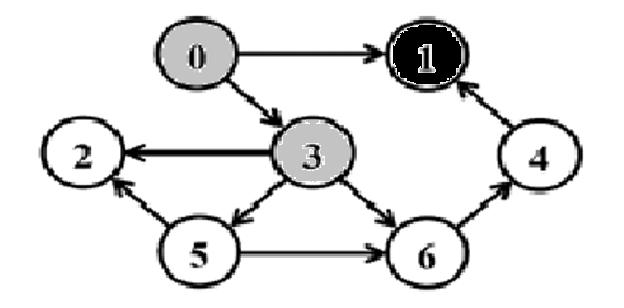
t 3

C [G] [B] [W] [W] [W] [W] [W]

D [1] [2] [|] [|] [|] [|] [|]

P [N] [0] [N] [N] [N] [N] [N]

F [0] [3] [0] [0] [0] [0] [0]



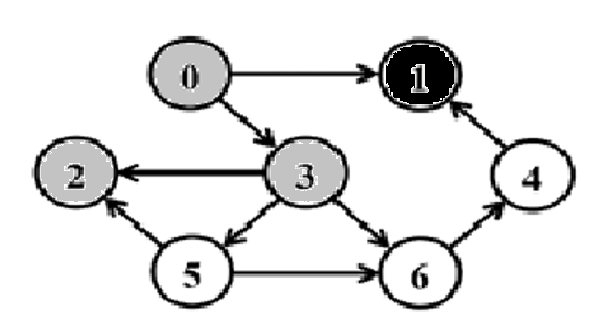
t 4

C [G] [B] [W] [G] [W] [W] [W]

D [1] [2] [|] [4] [|] [|] [|]

P [N] [0] [N] [0] [N] [N] [N]

F [0] [3] [0] [0] [0] [0] [0]



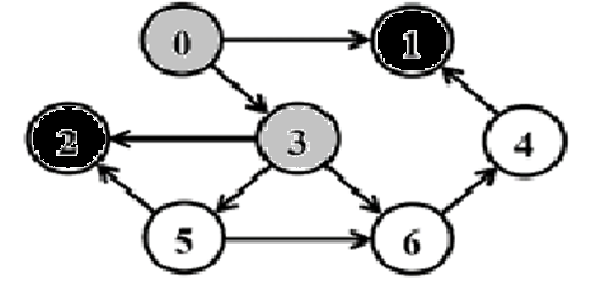
t 5

C [G] [B] [G] [G] [W] [W] [W]

D [1] [2] [5] [4] [|] [|] [|]

P [N] [0] [3] [0] [N] [N] [N]

F [0] [3] [0] [0] [0] [0] [0]



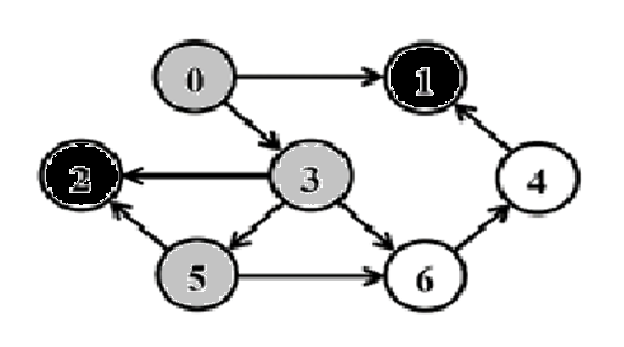
t 6

C [G] [B] [B] [G] [W] [W] [W]

D [1] [2] [5] [4] [|] [|] [|]

P [N] [0] [3] [0] [N] [N] [N]

F [0] [3] [6] [0] [0] [0] [0]



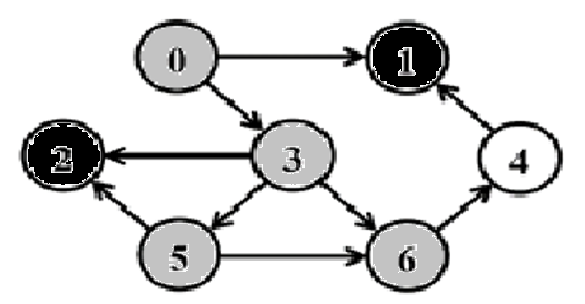
t 7

C [G] [B] [B] [G] [W] [G] [W]

D [1] [2] [5] [4] [|] [7] [|]

P [N] [0] [3] [0] [N] [3] [N]

F [0] [3] [6] [0] [0] [0] [0]



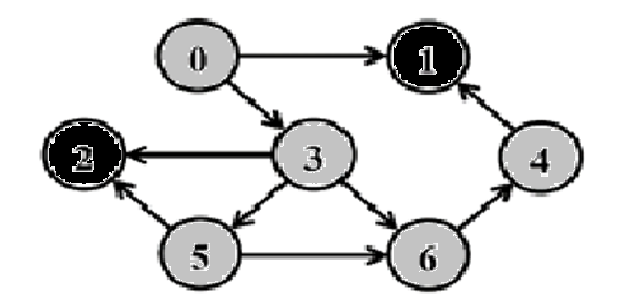
t 8

C [G] [B] [B] [G] [W] [G] [G]

D [1] [2] [5] [4] [|] [7] [8]

P [N] [0] [3] [0] [N] [3] [5]

F [0] [3] [6] [0] [0] [0] [0]



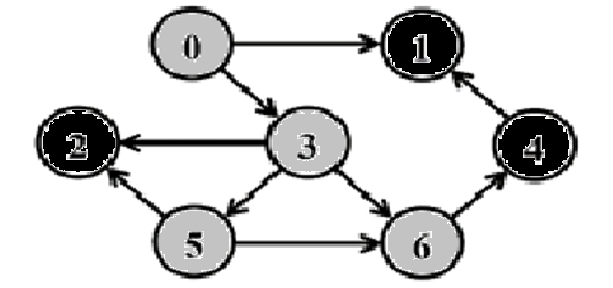
t 9

C [G] [B] [B] [G] [G] [G] [G]

D [1] [2] [5] [4] [9] [7] [8]

P [N] [0] [3] [0] [6] [3] [5]

F [0] [3] [6] [0] [0] [0] [0]



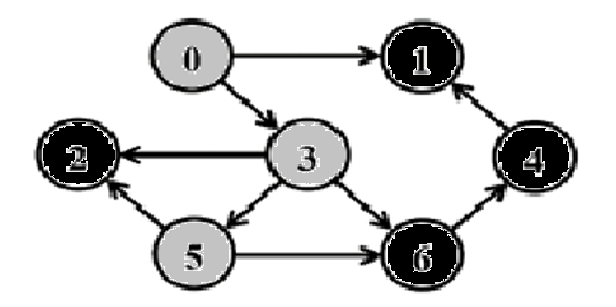
t 10

C [G] [B] [B] [G] [B] [G] [G]

D [1] [2] [5] [4] [9] [7] [8]

P [N] [0] [3] [0] [6] [3] [5]

F [0] [3] [6] [0] [10] [0] [0]



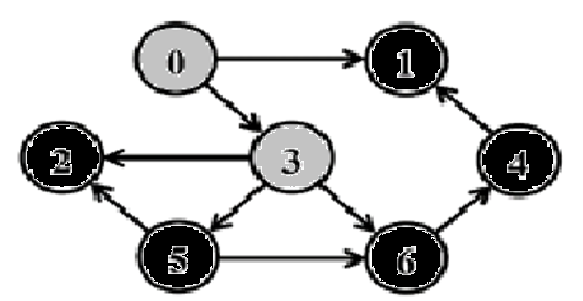
t 11

C [G] [B] [B] [G] [B] [G] [B]

D [1] [2] [5] [4] [9] [7] [8]

P [N] [0] [3] [0] [6] [3] [5]

F [0] [3] [6] [0] [10] [0] [11]



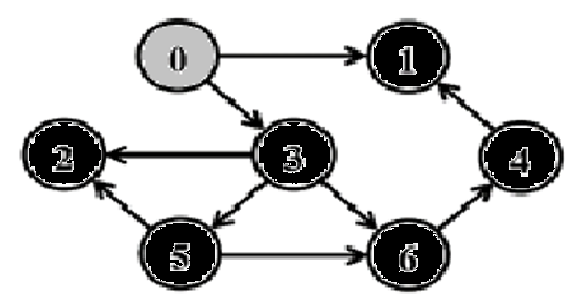
t 12

C [G] [B] [B] [G] [B] [B] [B]

D [1] [2] [5] [4] [9] [7] [8]

P [N] [0] [3] [0] [6] [3] [5]

F [0] [3] [6] [0] [10] [12] [11]



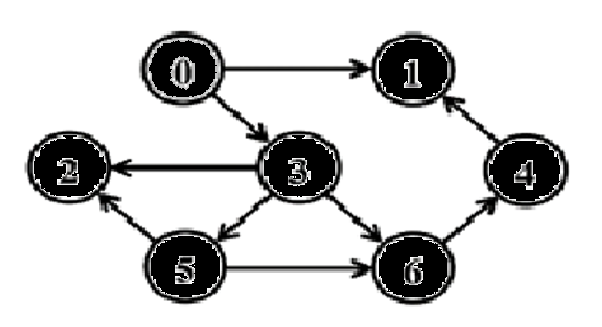
t 13

C [G] [B] [B] [B] [B] [B] [B]

D [1] [2] [5] [4] [9] [7] [8]

P [N] [0] [3] [0] [6] [3] [5]

F [0] [3] [6] [13] [10] [12] [11]



t 14

C [B] [B] [B] [B] [B] [B] [B]

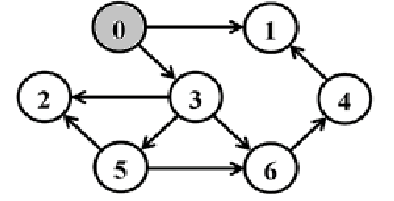
D [1] [2] [5] [4] [9] [7] [8]

P [N] [0] [3] [0] [6] [3] [5]

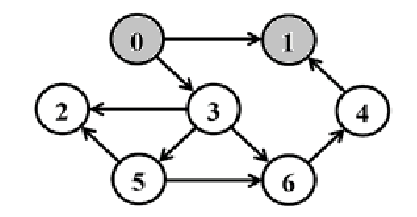
F [14] [3] [6] [13] [10] [12] [11]

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

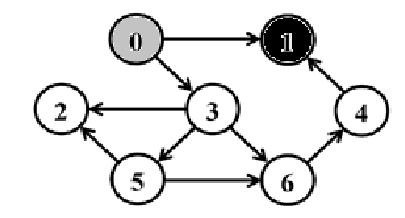
**Алгоритм топологической сортировки**



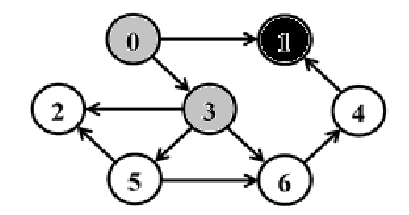
[]



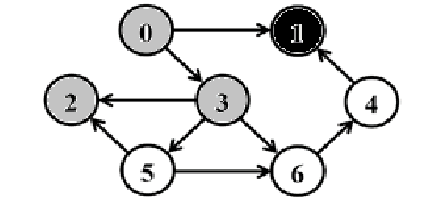
[]



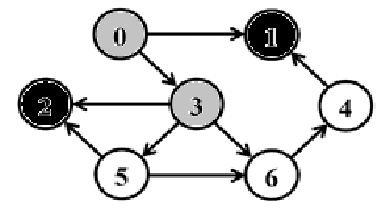
[1]



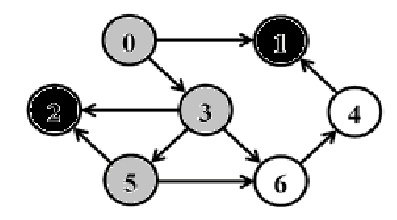
[1]



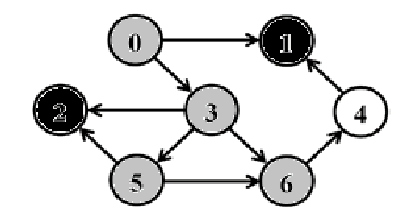
[2] [1]



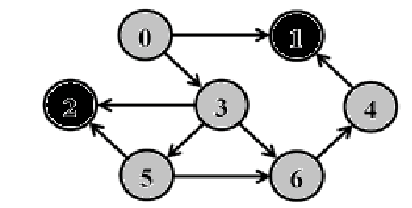
[2] [1]



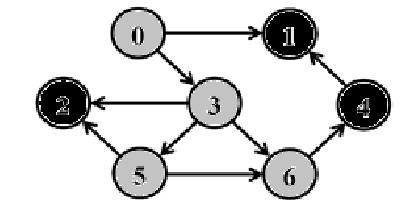
[2] [1]



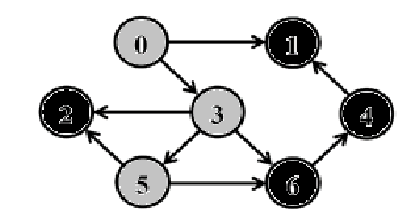
[2] [1]



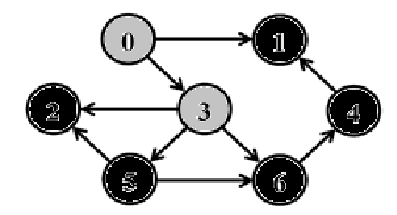
[4] [2] [1]



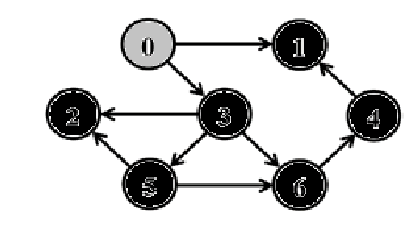
[4] [2] [1]



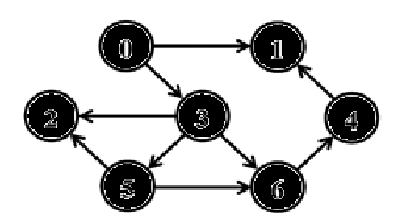
[6] [4] [2] [1]



[5] [6] [4] [2] [1]

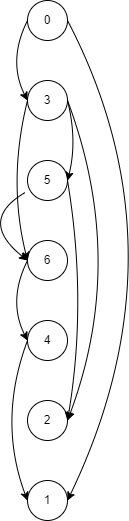


[3] [5] [6] [4] [2] [1]



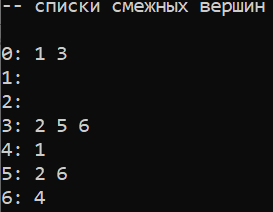
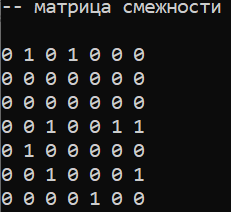
[0] [3] [5] [6] [4] [2] [1]

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



**Задание 3**

Демонстрация работы BFS и вывод графа в форме разных хранений





Вывод программы сходится с результатом работы алгоритма вручную.

**Задание 4**

Демонстрация работы DFS



Вывод программы сходится с результатом работы алгоритма вручную.

**Задание 5**

Демонстрация работы топологической сортировки



Вывод программы сходится с результатом работы алгоритма вручную.

**Задание 6**

Составление минимального остовного дерева по алгоритму Прима

Веса ребер принимаем:

W:

W(e0,1)=8; W(e1,0)=5;

W(e0,2)=1; W(e2,0)=3;

W(e0,3)=2; W(e3,0)=8;

W(e1,3)=11; W(e3,1)=4;

W(e1,4)=5; W(e4,1)=3;

W(e2,3)=7; W(e3,2)=9;

W(e2,5)=11; W(e5,2)=10;

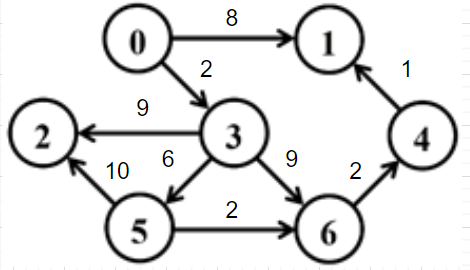
W(e4,3)=4; W(e3,4)=1;

W(e4,6)=10; W(e6,4)=2;

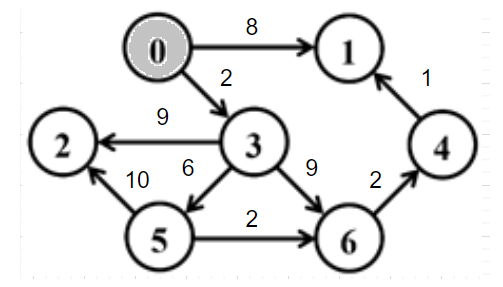
W(e5,6)=2; W(e6,5)=6;

W(e5,3)=3; W(e3,5)=6;

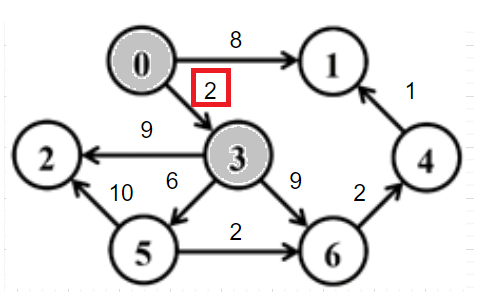
W(e6,3)=7; W(e3,6)=9;



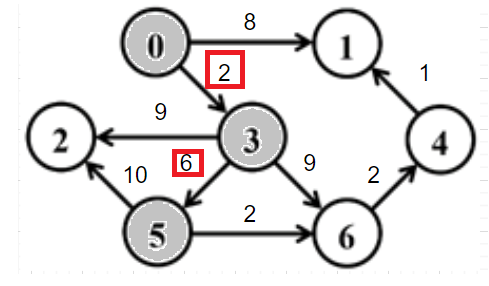
Выбираем произвольную вершину и окрашиваем:

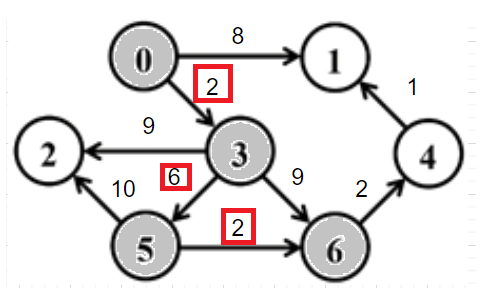


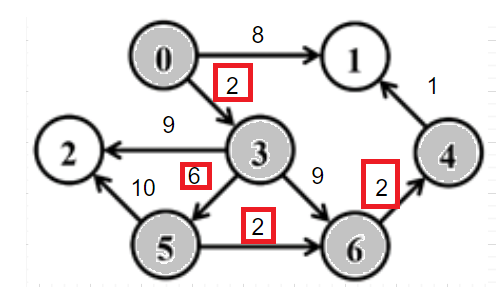
Среди всех ребер, инцидентных стартовой вершине, отыскивается ребро, имеющее наименьшую длину. Вторая (неокрашенная) вершина ребра окрашивается, а само ребро вместе с концевыми вершинами включается в будущее минимальное остовное дерево.

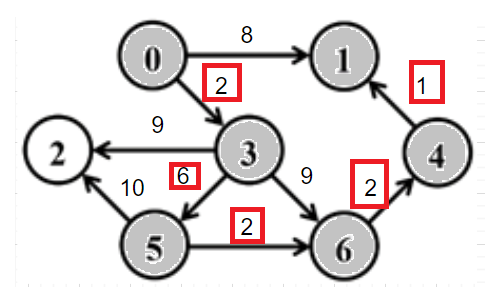


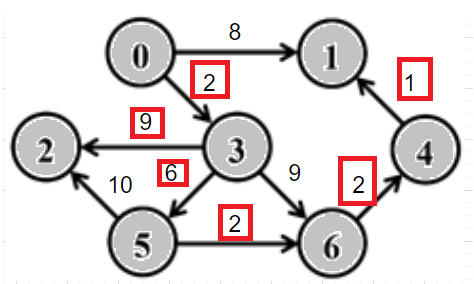
На последующих шагах алгоритма выбирается по одному ребру с минимальной длиной и одной неокрашенной концевой вершиной. Неокрашенные вершины окрашиваются, выбранные ребра пополняют строящееся минимальное остовное дерево. Цикл построения дерева продолжается до тех пор, пока не будут окрашены все вершины исходного графа.









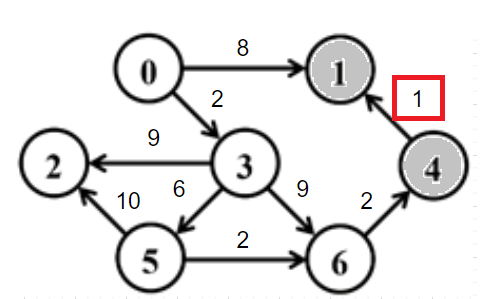


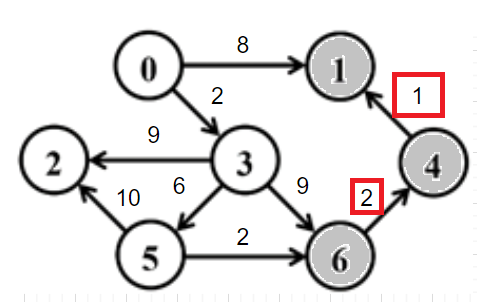
0 –> 3 [2], 3 -> 2 [9], 3 -> 5 [6], 5 -> 6 [2], 6 -> 4 [2], 4 -> 1 [1]

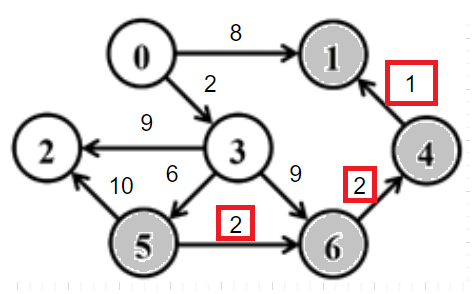
Минимальный путь: 2 + 9 + 6 + 2 + 2 + 1 = 22

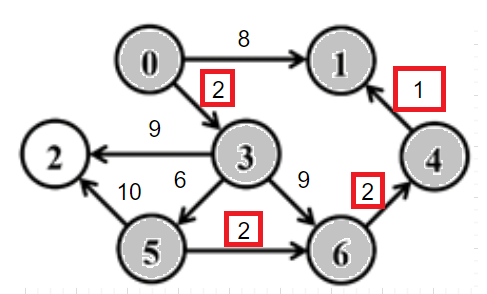
**Алгоритм Крускала**

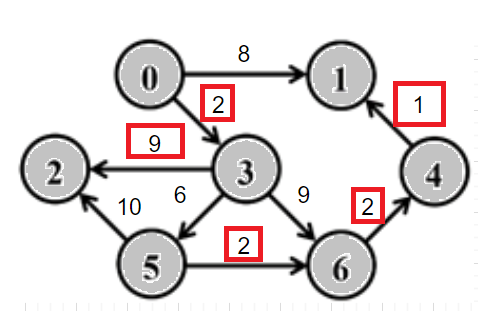
Отыскивается ребро исходного графа, имеющее минимальную длину. Вершины, соединенные выбранным ребром, окрашиваются.

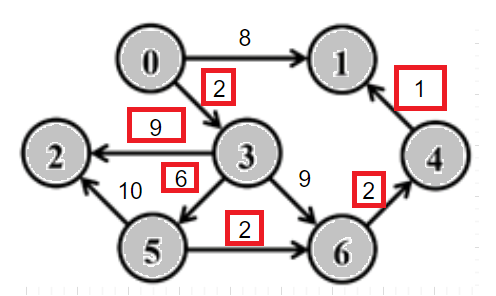












0 -> 3 [2], 3 -> 2 [9], 3 -> 5 [6], 5 -> 6 [2], 6 -> 4 [2], 4 -> 1 [1]

Минимальный путь: 2 + 9 + 6 + 2 + 2 + 1 = 22

**Вопросы для защиты**

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

Матрица смежности: это квадратная матрица, где элемент в позиции (i, j) равен 1, если вершины i и j соединены ребром, и 0 в противном случае. Для невзвешенного графа матрица смежности симметрична, а для взвешенного графа вместо 1 в ячейке может быть указан вес ребра.

Списки смежности: это представление графа, где каждая вершина имеет список смежных с ней вершин. Для каждой вершины создается список, содержащий все вершины, с которыми она соединена ребром.

Матрица инцидентности: это матрица, где строки соответствуют вершинам графа, а столбцы — ребрам. Элемент матрицы равен 1, если соответствующая вершина является началом или концом соответствующего ребра. В невзвешенных графах значение элемента обычно равно 1, а взвешенных графах может быть указан вес ребра.

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

Поиск в ширину (BFS, Breadth-First Search) — это алгоритм обхода или поиска в графе, который начинает с заданной вершины и идет "вширь", посещая сначала все соседние вершины текущей, затем все их соседние вершины и так далее.

Процесс работы алгоритма BFS можно описать следующим образом:

- Начинается с выбранной начальной вершины и помещения её в очередь.

- Пока очередь не пуста, извлекается вершина из начала очереди.

- Посещается извлеченная вершина и выполняются необходимые действия (например, сбор информации, проверка условий и т.д.).

- Для каждой смежной с посещенной вершиной, которая еще не была посещена, она добавляется в конец очереди.

- Повторяются шаги 2-4 до тех пор, пока очередь не станет пустой.

Поиск в ширину позволяет обойти все вершины графа, начиная с заданной

начальной вершины, при этом сначала посещаются все вершины на одном уровне (расстоянии) от начальной вершины, а затем переходят на следующий уровень. Алгоритм также помогает найти кратчайшие пути от начальной вершины ко всем остальным вершинам, при условии, что ребра графа имеют одинаковую длину.

Поиск в ширину рационально использовать в следующих случаях:

- Поиск кратчайшего пути в невзвешенном графе (где все ребра имеют одинаковую длину).

- Проверка связности графа и поиск всех достижимых вершин из заданной вершины.

- Поиск ближайших соседей для каждой вершины в графе.

- Топологическая сортировка графа (если граф является ациклическим ориентированным).

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

Поиск в глубину (DFS, Depth-First Search) — это алгоритм обхода или поиска в графе, который исследует вершины "вглубь" перед тем, как переходить к следующей непосещенной вершине.

Процесс работы алгоритма DFS можно описать следующим образом:

- Начинается с выбранной начальной вершины.

- Посещается текущая вершина и выполняются необходимые действия (например, сбор

информации, проверка условий и т.д.).

- Для каждой смежной с посещенной вершиной, которая еще не была посещена, рекурсивно запускается алгоритм DFS от этой вершины.

- Повторяется шаг 2-3 для каждой непосещенной вершины.

Алгоритм DFS идет "глубже" в структуру графа, посещая вершины до тех пор, пока не достигнет конечной вершины или пока не будет выполнено определенное условие остановки. При этом, если в графе встречается тупиковая ветвь (вершина, из которой нет путей к непосещенным вершинам), алгоритм возвращается к ближайшей предыдущей непосещенной вершине, от которой есть другие непосещенные вершины, и продолжает исследование.

Поиск в глубину рационально использовать в следующих случаях:

- Поиск пути между двумя вершинами (возможно, не кратчайшего пути).

- Проверка наличия циклов в графе.

- Генерация и обход всех возможных комбинаций или перестановок в графе.

- Топологическая сортировка графа (если граф является ациклическим ориентированным).

- Поиск компонент связности в неориентированном графе.

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

Топологическая сортировка — это алгоритмический процесс упорядочивания вершин ориентированного ациклического графа (DAG) в линейный порядок, который удовлетворяет условию, что каждое ребро в графе идет от вершины с меньшим порядковым номером к вершине с большим порядковым номером. Другими словами, это нахождение линейного порядка, в котором каждая вершина предшествует всем своим потомкам в графе.

Смысл топологической сортировки заключается в определении порядка выполнения задач или операций, которые зависят друг от друга. Конкретно, топологическая сортировка полезна в следующих случаях:

- Зависимости задач: Если в графе существуют зависимости между задачами, то топологическая сортировка может определить порядок выполнения этих задач. Например, если задача B зависит от выполнения задачи A, то A должна быть выполнена до B.

- Сборка программного кода: В программировании топологическая сортировка может быть использована при сборке программного кода с учетом зависимостей между модулями или компонентами.

- Графические редакторы: В графических редакторах топологическая сортировка может быть полезна для определения порядка отрисовки объектов, чтобы обеспечить правильное перекрытие и видимость элементов.

- Управление задачами и процессами: В различных областях управления проектами или процессами, где существуют зависимости между задачами, топологическая сортировка может помочь определить последовательность выполнения задач.

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

Минимальное остовное дерево (Minimum Spanning Tree, MST) — это подграф связного неориентированного взвешенного графа, который содержит все вершины исходного графа и является деревом (не содержит циклов), а сумма весов его ребер минимальна.

То есть, MST представляет собой подмножество ребер исходного графа такое, что они образуют дерево и имеют минимальную сумму весов среди всех возможных деревьев, соединяющих все вершины графа.

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

Стандартный алгоритм построения минимального остовного дерева (Minimum Spanning Tree, MST) включает два основных алгоритма: алгоритм Прима (Prim's algorithm) и алгоритм Краскала (Kruskal's algorithm). Оба алгоритма могут быть использованы для нахождения MST в связном неориентированном взвешенном графе.

Алгоритм Прима:

- Начинаем с выбора произвольной вершины в графе и помечаем ее как посещенную.

- Для каждой посещенной вершины находим ребро минимального веса, которое соединяет ее с непосещенной вершиной.

- Выбираем ребро с минимальным весом из найденных и добавляем его к остовному дереву. При этом связанная с ним непосещенная вершина становится посещенной.

- Повторяем предыдущий шаг, пока не будут посещены все вершины графа.

Алгоритм Краскала:

- Сортируем все ребра графа по возрастанию их весов.

- Начинаем с пустого остовного дерева.

- Последовательно добавляем ребра с минимальными весами к остовному дереву, при условии, что добавление данного ребра не создаст цикл в дереве. Для проверки цикла можно использовать структуру данных "независимые множества" (disjoint set).

- Повторяем предыдущий шаг, пока не будут добавлены все вершины минимального остовного дерева или пока не закончатся ребра.

Оба алгоритма гарантируют нахождение минимального остовного дерева. Однако алгоритм Прима обычно эффективнее для плотных графов (графов с большим количеством ребер), тогда как алгоритм Краскала лучше работает на разреженных графах (графах с небольшим количеством ребер). Выбор между алгоритмами может зависеть от конкретной структуры графа и его размера.

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

Алгоритмы Прима и Крускала относятся к категории алгоритмов для нахождения минимального остовного дерева (Minimum Spanning Tree, MST).

MST-алгоритмы решают задачу построения дерева, которое содержит все вершины связного неориентированного графа и имеет минимальную сумму весов ребер. Алгоритм Прима и алгоритм Крускала являются двумя широко используемыми методами для решения этой задачи.

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

Один шаг алгоритма Крускала включает следующие действия:

- Из отсортированного списка ребер выбирается ребро с минимальным весом, которое еще не было добавлено в остовное дерево.

- Проверяется, не создаст ли добавление данного ребра цикл в остовном дереве. Для этого можно использовать структуру данных "независимые множества" (disjoint set). Если ребро не создает цикл, то оно добавляется к остовному дереву.

- Повторяются шаги 1-2 до тех пор, пока не будут добавлены все вершины минимального остовного дерева или пока не закончатся ребра.

Алгоритм Крускала прекращает свою работу, когда все вершины графа становятся связанными и образуют минимальное остовное дерево, то есть когда добавлено достаточное количество ребер для соединения всех вершин без создания циклов. Это происходит, когда остовное дерево содержит (V-1) ребер, где V - количество вершин в графе.