Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Математическое программирование»

**Лабораторная работа №6**

**Тема «Алгоритмы на графах»**

Выполнил:

Студент 2 курса 7 группы ФИТ

Володькин Н. Д.   
 Проверил:   
 Доц. Буснюк Н. Н.

Минск 2023

**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 Краскала.

**Задание №1**

**Условие:** ориентированный граф **G** взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.



**Выполнение:**

Матрица смежности:

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

Матрица инцидентности:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **(0; 1)** | **(0; 3)** | **(1; 4)** | **(3; 1)** | **(3; 2)** | **(3; 5)** | **(4; 6)** | **(5; 2)** | **(6; 5)** |
| **0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | -1 | 0 | 1 | -1 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | -1 | 0 | 0 | -1 | 0 |
| **3** | 0 | -1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| **4** | 0 | 0 | -1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 1 | -1 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 1 |

Список смежных вершин:

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

**Задание №2**

**Условие:** 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.



**Выполнение:**

**Поиск в ширину**



1. Выбираем стартовую вершину 0. Добавляем ее в очередь.

**Очередь: [0]**

**Посещенная очередь: []**

1. Извлекаем из очереди вершину 0 и заносим в очередь посещения. У вершины 0 имеются смежные вершины – 1 и 3. Добавляем их в очередь.

**Очередь: [1, 3]**

**Посещенная очередь: [0]**

1. Извлекаем из очереди первую вершину (1) и добавляем ее в очередь посещения. У вершины 1 имеется смежная вершина 4, добавляем ее в очередь.

**Очередь: [3, 4]**

**Посещенная очередь: [0, 1]**

1. Извлекаем из очереди первую вершину (3) и добавляем ее в очередь посещения. У вершины 3 имеются смежные вершины 2 и 5, заносим их в очередь.

**Очередь: [4, 2, 5]**

**Посещенная очередь: [0, 1, 3]**

1. Извлекаем из очереди первую вершину (4) и добавляем ее в очередь посещения. У вершины 4 смежная вершина 6, добавляем ее в очередь.

**Очередь: [2, 5, 6]**

**Посещенная очередь: [0, 1, 3, 4]**

1. Извлекаем из очереди первую вершину (2) и добавляем ее в очередь посещения. У вершины 2 нет смежных вершин, продолжаем алгоритм.

**Очередь: [5, 6]**

**Посещенная очередь: [0, 1, 3, 4, 2]**

1. Извлекаем из очереди первую вершину (5) и добавляем ее в очередь посещения. У вершины 5 имеется смежная вершина 2, но она уже в очереди, продолжаем алгоритм.

**Очередь: [6]**

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

1. Извлекаем из очереди первую вершину (6) и добавляем ее в очередь посещения. У вершины 5 имеются смежная вершина 2, однако она есть в очереди посещения, пропускаем ее.

**Очередь: []**

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

1. Все вершины пройдены. Конец алгоритма.

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

**Поиск в глубину**



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

**Стек: [0]**

**Посещенные вершины: {0}**

У вершины 0 имеется 2 смежных вершины - 1 и 3. Добавляем первую смежную вершину (1) в стек.

**Стек: [0, 1]**

**Посещенные вершины: {0, 1}**

У вершины 1 имеется смежная вершина 4, добавляем ее в стек, отмечаем 1 как посещенную.

**Стек: [0, 1, 4]**

**Посещенные вершины: {0, 1, 4}**

У вершины 4 смежная вершина 6, добавляем ее в стек, отмечаем 4 как посещенную.

**Стек: [0, 1, 4, 6]**

**Посещенные вершины: {0, 1, 4, 6}**

У вершины 6 смежная вершина 5, добавляем ее в стек, отмечаем 6 как посещенную.

**Стек: [0, 1, 4, 6, 5]**

**Посещенные вершины: {0, 1, 4, 6, 5}**

У вершины 5 смежная вершина 2, добавляем ее в стек, отмечаем 6 как посещенную.

**Стек: [0, 1, 4, 6, 5, 2]**

**Посещенные вершины: {0, 1, 4, 6, 5, 2}**

У вершины 2 нет смежных вершин, отмечаем 2 как посещенную.

**Стек: [0, 1, 4, 6, 5]**

**Посещенные вершины: {0, 1, 4, 6, 5, 2}**

Убираем вершину 2 из стека.

**Стек: [0, 1, 4, 6]**

**Посещенные вершины: {0, 1, 4, 6, 5, 2}**

Убираем вершину 5 из стека.

**Стек: [0, 1, 4]**

**Посещенные вершины: {0, 1, 4, 6, 5, 2}**

Убираем вершину 6 из стека.

**Стек: [0, 1]**

**Посещенные вершины: {0, 1, 4, 6, 5, 2}**

Убираем вершину 4 из стека.

**Стек: [0]**

**Посещенные вершины: {0, 1, 4, 6, 5, 2}**

Убираем вершину 1 из стека.

**Стек: [0, 3]**

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

1. У вершины 3 нет смежных вершин, отмечаем 3 как посещенную.

**Стек: [0]**

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

1. У вершины 0 нет не посещенных смежных вершин. Конец алгоритма.

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

**Топологическая сортировка**



1. Выбираем вершину без исходящих дуг. Это вершина 2.
2. Убираем все входящие в вершину 2 дуги.
3. Снова выбираем вершину без исходящих дуг. После удаления это вершина 5.
4. Убираем все входящие в вершину 5 дуги.
5. Снова выбираем вершину без исходящих дуг. После удаления это вершина 6.
6. Убираем все входящие в вершину 6 дуги.
7. Снова выбираем вершину без исходящих дуг. После удаления это вершина 4.
8. Убираем все входящие в вершину 4 дуги.
9. Снова выбираем вершину без исходящих дуг. После удаления это вершина 1.
10. Убираем все входящие в вершину 1 дуги.
11. Снова выбираем вершину без исходящих дуг. После удаления это вершина 3.
12. Убираем все входящие в вершину 3 дуги.
13. Снова выбираем вершину без исходящих дуг. После удаления это вершина 0.
14. Убираем все входящие в вершину 0 дуги.

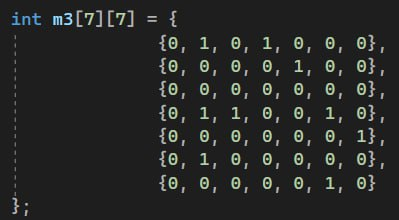
**Топологическая сортировка: 2 -> 5 -> 6 -> 4 -> 1 -> 3 -> 0**

**Задание №3**

**Условие:** 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.

**Выполнение:**







**Задание №4**

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

**Выполнение:**





Конечный ответ отличается от моего, однако в данной программной реализации метод начинает действовать также как и я, в то время как я осуществлял поиск по принципу увеличения номера вершины и поэтому я пошел сначала в вершину 1.

**Задание №5**

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

**Выполнение:**





**Задание №6**

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

Веса ребер принять:

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;

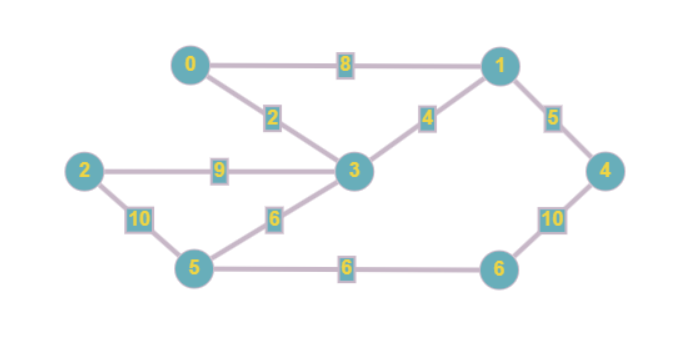
W(e4,3)=4; W(e3,4)=1;

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;



**Выполнение:**

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

**Дерево: {0}**

1. Доступны два ребра: (0, 3) и (0, 1), выбираем ребро с наименьшим весом 2 (0, 3), добавляем вершину 3 в дерево.

**Дерево: {0, 3}**

1. Из доступных ребер выбираем ребро с наименьшим весом 4 (3, 1), добавляем вершину 1 в дерево.

**Дерево: {0, 3, 1}**

1. Из доступных ребер выбираем ребро с наименьшим весом 5 (1, 4), добавляем вершину 4 в дерево.

**Дерево: {0, 3, 1, 4}**

1. Из доступных ребер выбираем ребро с наименьшим весом 6 (3, 5), добавляем вершину 5 в дерево.

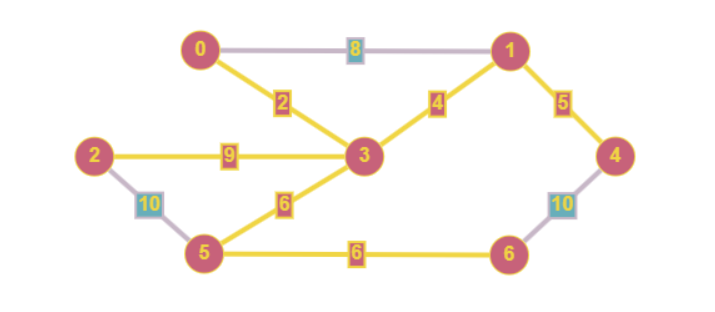
**Дерево: {0, 3, 1, 4, 5}**

1. Из доступных ребер выбираем ребро с наименьшим весом 6 (5, 6), добавляем вершину 6 в дерево.

**Дерево: {0, 3, 1, 4, 5, 6}**

1. Из доступных ребер выбираем ребро с наименьшим весом 9 (3, 2), добавляем вершину 2 в дерево.

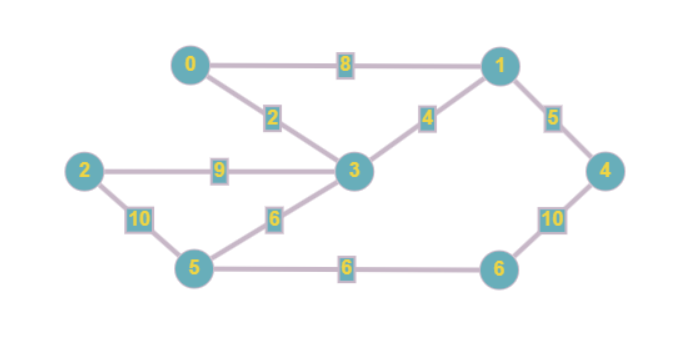
**Дерево: {0, 3, 1, 4, 5, 6, 2}**



Вес минимального остовного дерева равен 32.

**Задание №7**

**Условие:** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Краскала. Шаги построения отразить в отчете.



**Выполнение:**

Отличие алгоритма Краскала от алгоритма Прима в том, что в алгоритме Прима мы проходимся по тому же графу и строим единое дерево, а также мы выбираем какую-то стартовую вершину и от нее ищем ребра с минимальными весами, связывая таким образом вершины. В алгоритме Краскала может получиться так(в большинстве случаев), что мы создаем несколько деревьев и объединяем их в одно по ходу алгоритма. 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.

|  |  |
| --- | --- |
| Неотсортированные веса | Отсортированные по неубыванию веса |
| 0 🡨🡪 1; W = 8 0 🡨🡪 3; W = 2 1 🡨🡪 3; W = 4 1 🡨🡪 4; W = 5 3 🡨🡪 2; W = 9 3 🡨🡪 5; W = 6 4 🡨🡪 6; W = 10 5 🡨🡪 2; W = 10 5 🡨🡪 6; W = 6 | 0 🡨🡪 3; W = 2 1 🡨🡪 3; W = 4 1 🡨🡪 4; W = 5 3 🡨🡪 5; W = 6 5 🡨🡪 6; W = 6 0 🡨🡪 1; W = 8 3 🡨🡪 2; W = 9 4 🡨🡪 6; W = 10 5 🡨🡪 2; W = 10 |

1. Выбираем вершину 0, так как к ней прилегает ребро с минимальным весом 2. Создаем дерево с корнем в вершине 0. Ребро ведет к вершине 3, записываем эту вершину как вершину первого дерева.

**Дерево 1: {0, 3}**

1. Выбираем следующее ребро с минимальным весом, это ребро с весом 4, которое идет между вершинами 1 и 3, в котором уже имеется дерево. Записываем эту вершину как вершину первого дерева.

**Дерево 1: {0, 3, 1}**

1. Выбираем следующее ребро с минимальным весом, это ребро с весом 5, которое идет между вершинами 1 и 4, в котором уже имеется дерево. Записываем эту вершину как вершину первого дерева.

**Дерево 1: {0, 3, 1, 4}**

1. Выбираем следующее ребро с минимальным весом, это ребро с весом 6, которое идет между вершинами 3 и 5, вершина 3 уже привязана к первому дереву, поэтому добавляем к этому дереву вершину 5.

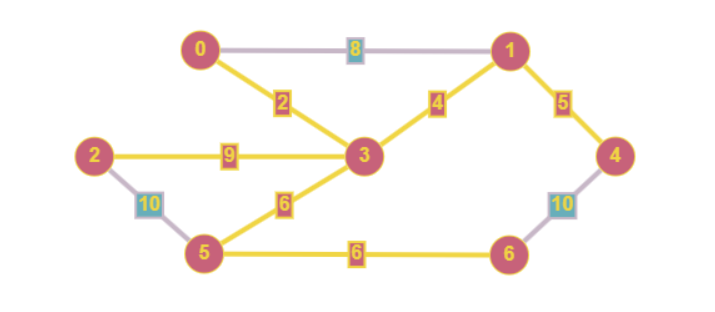
**Дерево 1: {0, 3, 1, 4, 5}**

1. Выбираем следующее ребро с минимальным весом, это ребро с весом 6, которое идет между вершинами 5 и 6, вершина 5 уже привязана к первому дереву, поэтому добавляем к этому дереву вершину 6.

**Дерево 1: {0, 3, 1, 4, 5, 6}**

1. Пропускаем следующее ребро с минимальным весом, так как оно идет между вершинами 0 и 1, которые уже включены в первое дерево, и берем следующее ребро, которое имеет вес 9 и лежит между вершинами 3 и 2, вершина 3 уже находится в дереве, включаем вершину 2 в дерево.

**Дерево 1: {0, 3, 1, 4, 5, 6, 2}**



Вес минимального остовного дерева равен 32.

**Вывод:** Алгоритмы поиска в глубину и ширину применяются для обхода графов и поиска путей между вершинами. Они также могут использоваться для проверки наличия циклов в графах и для поиска компонент связности.

Алгоритм топологической сортировки применяется для упорядочивания вершин в ориентированном ациклическом графе таким образом, чтобы все дуги указывали от более ранней вершины к более поздней. Это может быть полезно для определения последовательности выполнения задач в проектах или для обнаружения зависимостей между задачами.

Алгоритм Прима и Краскала применяются для построения минимального остовного дерева в связном взвешенном графе. Они могут использоваться для оптимизации сетевых и транспортных систем, в телекоммуникационных сетях и в других областях, где требуется выбрать наиболее экономичный маршрут или связь между точками.

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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.