МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Брянский государственный технический университет

Утверждаю
Ректор университета

_____О.Н. Федонин

«____»_____2020 г.

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

АЛГОРИТМЫ НА ГРАФАХ ОБЩЕГО ВИДА
(поиск в глубину и поиск в ширину)

Методические указания
к выполнению лабораторной работы №16
для студентов очной, очно-заочной и заочной форм обучения
по направлениям подготовки
230100 «Информатика и вычислительная техника»,
010500 «Математическое обеспечение и администрирование информационных систем»,
231000 «Программная инженерия»

Брянск 2020

УДК 006.91

Структуры и алгоритмы обработки данных. Алгоритмы на графах общего вида (Поиск в глубину и поиск в ширину) [Электронный ресурс]: методические указания к выполнению лабораторной работы №16 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия». – Брянск: БГТУ, 2020.– 18 с.

Разработали:
канд. техн. наук, проф.
В.К. Гулаков
канд. техн. наук, доц.
А.О. Трубаков
канд. техн. наук, доц.
Е.О. Трубаков

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №1 от 13.09.20)

1. ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Целью лабораторной работы является приобретение практических навыков обхода графа на основе алгоритмов поиска в ширину и поиска в глубину.

Продолжительность лабораторной работы − 4 часа.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Теория графов сегодня широко используется в различных отраслях науки и техники. Быстрое развитие данная теория получила с созданием электронно-вычислительной техники, которая позволяла решить многие задачи алгоритмизации.

Выбор структуры данных для хранения графа в памяти компьютера имеет принципиальное значение при разработке эффективных алгоритмов. При решении задач используются следующие пять основных способов представления графа:

матрица инциденций;

матрица смежности;

списки ребер;

структура смежности;

векторы смежности.

Рассмотрим эти представления графа более подробно.

Пусть задан граф (например, рис. 1), у которого число вершин равно n, а число ребер – m. Каждое ребро и каждая вершина имеют вес – целое положительное число. Если граф не является помеченным, то считается, что его вес равен единице.

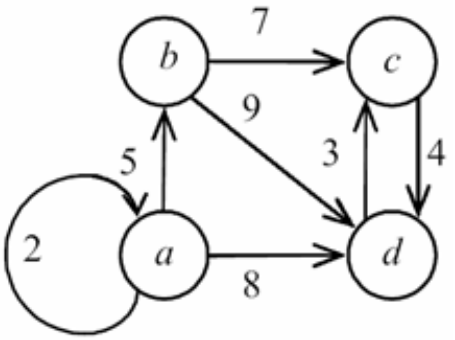

Рис. 1. Граф

1. **Матрица инцидентности** – это двумерный массив размерности $n \times m$, в котором указываются связи между инцидентными элементами графа (ребро и вершина). Столбцы матрицы соответствуют ребрам, строки – вершинам (рис. 2). Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром. Данный способ хранения является самым емким для хранения связей, но облегчает нахождение циклов в графе.

	a	b	c	d
a-a	2			
a-b		5		
a-d				8
b-c			7	
b-d				9
c-d				4
d-c			3	

	a	b	c	d
a-a	1			
a-b	-1	1		
a-d	-1			8
b-c		-1	1	
b-d		-1		9
c-d			-1	4
d-c			1	

	a	b	c	d
a-a	1			
a-b	1	1		
a-d	1			8
b-c		1	1	
b-d		1		1
c-d			1	1
d-c				

Рис. 2. Матрица инцидентности графа

Если рёбра не имеют веса для ориентированного графа, то столбец, соответствующий дуге $(x,y) \in E$, содержит -1 в строке, соответствующей вершине *x,* 1 в строке, соответствующей вершине *y,* и нули во всех остальных строках.

В случае неориентированного графа строка, соответствующий ребру *(x, y),* содержит 1 в столбцах, соответствующих узлам *x* и *y,* и нули в остальных столбцах.

С алгоритмической точки зрения матрица инциденций является, вероятно, самым худшим способом представления графа, который только можно себе представить. Во-первых, он требует n**m* ячеек памяти, причем большинство этих ячеек вообще занято нулями. Неудобен также доступ к информации. Ответ на элементарные вопросы типа “существует ли дуга *(x,* y)?”, “к каким вершинам ведут ребра из x?” требует в худшем случае перебора всех столбцов матрицы, а следовательно, *m* шагов.

Существует много алгоритмов на графах, основой которых является систематический перебор вершин графа, такой, что каждая вершина просматривается (посещается) в точности один раз. Поэтому существенной задачей является нахождение хороших методов поиска в графе.

2. **Матрица смежности** – это двумерный массив размерности $n \times n$, значения элементов которого характеризуются смежностью вершин графа (рис. 3). При этом значению элемента матрицы присваивается вес ребер, которые соединяют соответствующие вершины. Такой подход используется, когда необходимо проверять смежность или находить вес ребра по двум заданным вершинам.

	a	*b*	*c*	*d*
a	2	5	0	8
b	0	0	7	9
c	0	0	0	4
d	0	0	3	0

Рис. 3. Матрица смежности графа

Если рёбра не имеют веса, то все не нулевые элементы заменяются на единицы, что позволяет экономить память

Матрица смежности данного орграфа G=(V,E) зависит от упорядоченных элементов V.

Для различных упорядочений элементов V получаются различные матрицы смежности одного и того же графа.

Тем не менее любая матрица смежности графа G может быть получена из другой матрицы смежности этого же графа путем перестановки некоторых строк и соответствующих им столбцов.

Основным преимуществом матрицы смежности является тот факт, что за один шаг можно получить ответ на вопрос “существует ли ребро из *x* в *y?*

Недостатком же является тот факт, что независимо от числа ребер объем занятой памяти составляет n^2. На практике это неудобство можно иногда уменьшить, храня целую строку (столбец) матрицы в одном машинном слове. Это возможно для малых *n.* Для простых неориентированных графов эта матрица смежности симметрична.

Трудно внести изменения в граф

Понятие матричного представления можно расширить на мультиграфы и взвешенные графы.

В случае мультиграфа или взвешенного графа $a_{ij} = w_{ij}$, где w_{ij} - либо кратность, либо вес ребра. В этом случае матрица смежности называется **матрицей весов.**

Если в каждой вершине графа имеются только петли, матрица смежности является диагональной.

Список ребер – это множество, образованное парами смежных вершин (рис. 4). Для его хранения обычно используют одномерный массив размером *m*, содержащий список пар вершин, смежных с одним ребром графа. Список ребер более удобен для реализации различных алгоритмов на графах по сравнению с другими способами.

a	*a*	*a*	*b*	*b*	*c*	*d*
a	*b*	*d*	*c*	*d*	*d*	*c*
2	5	8	7	9	4	3

Рис. 4. Список ребер графа

Если вес рёбер не задаётся, то третий элемент в списке отсутствует. Списки в массиве требуют указателей. Это представление можно реализовать двумя массивами g=(g_1, g_2, g_3, ...g_E)-начальные вершины ребер, h=(h_1, h_2, h_3..., h_E)-конечные вершины ребер.

Очевидно, что объем памяти в этом случае составляет *2m (m - количество рёбер)*.

Неудобством является большое число шагов (порядка *m* в худшем случае), необходимое для получения множества вершин, к которым ведут ребра из данной вершины.

Структура смежности - список всех последователей (соседей) каждой вершины.

Для каждой вершины задается список всех последователей (соседей) этой вершины.

Для графа представленного на рис. 1 эта структура выглядит следующим образом:

Для ориентированного графа

a: a, b, d
b: c, d
c: d
d: c

для неориентированного графа

a: a, b, d
b: a, c, d
c: b, d
d: a, b, c

Если для хранения метки вершины используется одно машинное слово, то структура смежности ориентированного графа требует V+E слов, а для неориентированного V+2*E слов.

Структуры смежности могут быть удобно реализованы списком списков или массивом из V линейно связанных списков, где каждый список содержит последователей некоторой вершины.

Такое представление желательно для алгоритмов, в которых в графе добавляются или удаляются вершины.

Вектор смежности. Номер строки соответствует номеру узла. Каждая строка содержит номера смежных узлов.

Для ориентированного графа

a	b	d
c	d	
d		
c		

Для неориентированного графа

a	b	d
a	c	d
b	d	
a	b	c

Рис. 5. Вектор смежности для графа на рис.1.

Для представления ориентированных графов можно также использовать списковые структуры. Списковые структуры в основном используются при обработке символов. При этом характерны две черты:

1. Непредсказуемые требования памяти

2. Высокая интенсивность обработки данных

Во многих задачах на графах выбор представления является решающим для эффективности алгоритма. С другой стороны, переход от одного представления к другому относительно прост и имеет сложность $O(V^2)$.

Под **обходом графов (поиском на графах)** понимается процесс систематического просмотра всех ребер или вершин графа с целью нахождения ребер или вершин, удовлетворяющих некоторому условию.

При решении многих задач, использующих графы, необходимы эффективные методы регулярного обхода вершин и ребер графов. К стандартным и наиболее распространенным методам обхода вершин и ребер графов относятся:

- грубый метод
- поиск в глубину (*Depth First Search*, *DFS*);
- поиск в ширину (*Breadth First Search*, *BFS*).

Эти методы чаще всего применяются для ориентированных графов, но они применимы и для неориентированных, ребра которых считаются двунаправленными. Алгоритмы этих методов применяют при решении различных задач обработки графов, например

построении остовного леса, проверки связности, ацикличности, вычисления расстояний между вершинами и других.

• *Определение:* Остовное дерево (стягивающее дерево, каркас) – это граф, в котором каждая пара вершин связана одной и только одной цепью.

• *Теорема:* Граф G является связным тогда и только тогда, когда он содержит остовное дерево.

Число различных каркасов полного связного неориентированного графа с N вершинами было найдено впервые Коли и равно $N^{(N-2)}$.

Грубый метод

выбираем самое дешевое ребро, затем к нему добавляется самое дешевое из оставшихся и т.д.

При этом окончательная подсеть должна содержать:

все вершины;

должна быть связной;

не должна содержать циклов;

должна иметь минимально возможный вес.

Подалгоритмы определения связности и наличия циклов могут сделать этот алгоритм неэффективным.

Поиск в глубину

При поиске в глубину посещается первая вершина, затем необходимо идти вдоль ребер графа, до попадания в тупик. Вершина графа является тупиком, если все смежные с ней вершины посещены. После попадания в тупик необходимо возвращаться назад вдоль пройденного пути, пока не будет обнаружена вершина, у которой есть еще не посещенная смежная вершина, а затем необходимо двигаться в этом новом направлении. Процесс оказывается завершенным при возвращении в начальную вершину, причем все смежные с ней вершины уже должны быть посещены.

Таким образом, основная идея поиска в глубину – когда возможные пути по ребрам, выходящим из вершин, разветвляются, необходимо сначала полностью исследовать одну ветку и только потом переходить к другим веткам (если они останутся нерассмотренными).

Алгоритм поиска в глубину

Шаг 1. Все вершины графа помечаются как не посещенные. Выбирается первая вершина и помечается как посещенная.

Шаг 2. Для последней вершины, помеченной как посещенная, выбирается смежная вершина, являющаяся первой помеченной как не посещенная, и она помечается как посещенная. Если таких вершин нет, то берется предыдущая помеченная вершина.

Шаг 3. Повторить шаг 2 до тех пор, пока все вершины не будут помечены как посещенные (рис. 5).

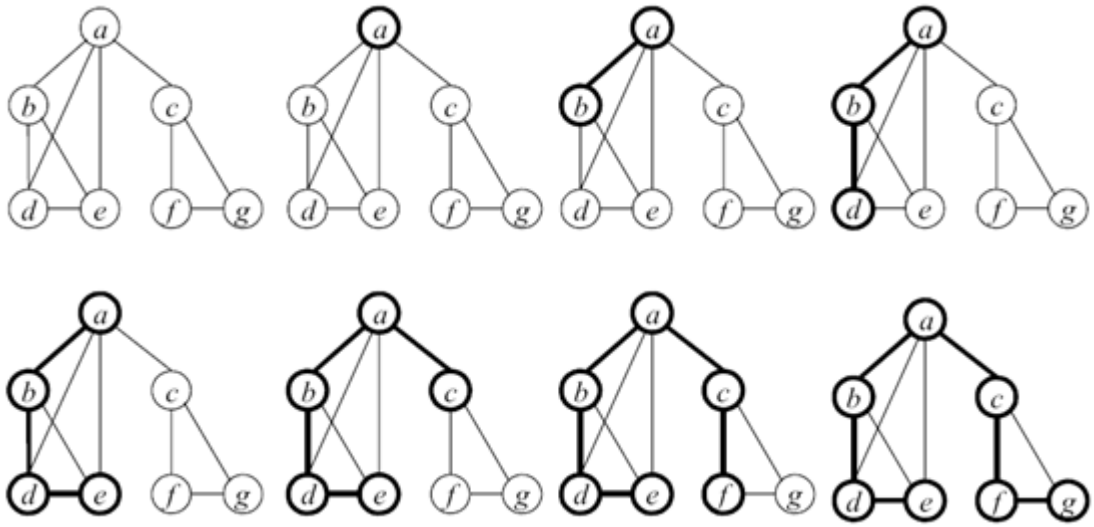

Рис. 5. Демонстрация алгоритма поиска в глубину

```
//Описание функции алгоритма поиска в глубину
void Depth_First_Search(int n, int **Graph, bool
*Visited,
                        int Node){
  Visited[Node] = true;
  cout << Node + 1 << endl;
  for (int i = 0 ; i < n ; i++)
    if (Graph[Node][i] && !Visited[i])
      Depth_First_Search(n,Graph,Visited,i);
}
```

Также часто используется не рекурсивный алгоритм поиска в глубину. В этом случае рекурсия заменяется на стек. Как только вершина просмотрена, она помещается в стек, а использованной она становится, когда больше нет новых вершин, смежных с ней.

Временная сложность зависит от представления графа. Если применена матрица смежности, то временная сложность равна $O(n^2)$, а если нематричное представление – $O(n+m)$: рассматриваются все вершины и все ребра.

Поиск в ширину

При поиске в ширину, после посещения первой вершины, посещаются все соседние с ней вершины. Потом посещаются все вершины, находящиеся на расстоянии двух ребер от начальной. При каждом новом шаге посещаются вершины, расстояние от которых до начальной на единицу больше предыдущего. Чтобы предотвратить повторное посещение вершин, необходимо вести список посещенных вершин. Для хранения временных данных, необходимых для работы алгоритма, используется очередь – упорядоченная последовательность элементов, в которой новые элементы добавляются в конец, а старые удаляются из начала.

Таким образом, основная идея поиска в ширину заключается в том, что сначала исследуется все вершины, смежные с начальной вершиной (вершина с которой начинается обход). Эти вершины находятся на расстоянии 1 от начальной. Затем исследуется все вершины на расстоянии 2 от начальной, затем все на расстоянии 3 и т.д. Обратим внимание, что при этом для каждой вершины сразу находятся длина кратчайшего маршрута от начальной вершины.

Алгоритм поиска в ширину

Шаг 1. Все вершины графа помечаются как непосещенные. Выбирается первая вершина и помечается как посещенная (и заносится в очередь).

Шаг 2. Посещается первая вершина из очереди (если она не помечена как посещенная). Все ее соседние вершины заносятся в очередь. После этого она удаляется из очереди.

Шаг 3. Повторяется шаг 2 до тех пор, пока очередь не станет пустой (рис. 6).

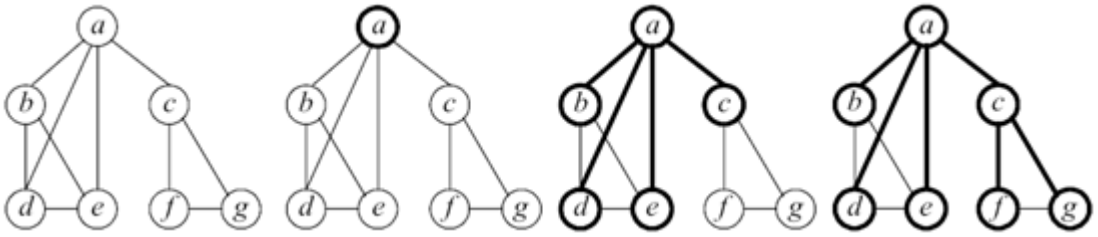

Рис. 6. Демонстрация алгоритма поиска в ширину

```
//Описание функции алгоритма поиска в ширину
void Breadth_First_Search(int n, int **Graph,
                          bool *Visited, int Node){
  int *List = new int[n]; //очередь
  int Count, Head;        // указатели очереди
  int i;
  // начальная инициализация
  for (i = 0; i < n ; i++)
    List[i] = 0;
  Count = Head = 0;
  // помещение в очередь вершины Node
  List[Count++] = Node;
  Visited[Node] = true;
  while ( Head < Count ) {
    //взятие вершины из очереди
    Node = List[Head++];
    cout << Node + 1 << endl;
    // просмотр всех вершин, связанных с вершиной Node
    for (i = 0 ; i < n ; i++)
      // если вершина ранее не просмотрена
      if (Graph[Node][i] && !Visited[i]){
        // заносим ее в очередь
        List[Count++] = i;
        Visited[i] = true;
      }
  }
}
```

Сложность поиска в ширину при нематричном представлении графа равна $O(n+m)$, ибо рассматриваются все n вершин и m ребер. Использование матрицы смежности приводит к оценке $O(n^2)$

1. Графы являются моделью представления данных, основанных на отношениях между элементами множеств.

2. Для представления графов используется несколько способов: список ребер, матрица смежности, матрица инцидентности.

3. Для организации поиска на графах используются обходы в глубину и в ширину.

4. Реализацию обходов можно реализовать рекурсивными и не рекурсивными алгоритмами.

5. От вида графа и способа его представления зависит временная сложность выполнения алгоритма.

3. ТЕХНИКА БЕЗОПАСНОСТИ ПРИ ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Меры безопасности при работе с электротехническими устройствами соответствуют мерам безопасности, принимаемым при эксплуатации установок с напряжением до 1000 В и разработанным в соответствии с «Правилами техники безопасности при эксплуатации электроустановок потребителей», утвержденными Главгосэнергонадзором 21 декабря 1984 г.

Студенту не разрешается приступать к выполнению лабораторной работы, если замечены какие-либо неисправности в лабораторном оборудовании.

Студент не должен прикасаться к токоведущим элементам электрооборудования, освещения и электропроводке, открывать двери электрошкафов и корпусов системных блоков, мониторов.

Студенту запрещается прикасаться к неизолированным или поврежденным проводам и электрическим устройствам, наступать на переносные электрические провода, лежащие на полу, самостоятельно ремонтировать электрооборудование и инструмент.

Обо всех замеченных неисправностях электрооборудования студент должен немедленно поставить в известность преподавателя или лаборанта.

При выполнении лабораторной работы необходимо соблюдать осторожность и помнить, что только человек, относящийся серьезно к своей безопасности, может быть застрахован от несчастного случая.

4. ЗАДАЧИ К ЛАБОРАТОРНОЙ РАБОТЕ

1. На основании приведенной в п. 2 функции реализуйте программу, в которой выполняется алгоритм обхода графа на основе поиска в глубину.

2. На основании приведенной в п. 2 функции реализуйте программу, в которой выполняется алгоритм обхода графа на основе поиска в ширину.

3. Используйте обход графа в ширину для определения всех вершин графа, находящихся на фиксированном расстоянии d от данной вершины.

4. Перенумеруйте вершины графа в порядке обхода в глубину и определите среднюю плотность графа как частное от деления количества его ребер на число вершин. Можно ли оба эти действия выполнить за один обход графа?

5. В вершинах неориентированного графа хранятся положительные целые числа. Подсчитайте число пар дружественных чисел в вершинах графа, которые соединены ребрами.

6. В общежитии живет N студентов. При поселении каждый студент представил список своих знакомых. Каждое воскресенье организуется вечер знакомств, когда знакомые любого студента знакомятся между собой. Выяснить, через сколько недель познакомятся два указанных студента.

7. Список членов клуба любителей анекдотов включает N человек. Каждый из членов клуба имеет несколько слушателей, которым он рассказывает анекдоты. В клубе действуют следующие правила:

а) если один человек рассказывает другому неизвестный тому анекдот, то он получает премию 1000 рублей;

б) если рассказанный анекдот оказывается известным, то рассказчик платит в качестве штрафа 1000 рублей;

в) член клуба, узнавший на некотором заседании клуба новый анекдот обязан на следующем заседании пересказать его своим слушателям;

г) на каждом заседании члены клуба с меньшими номерами в списке рассказывают новые анекдоты в первую очередь;

д) каждый член клуба рассказывает анекдот любому из своих слушателей не более одного раза.

Член клуба с номером *K* узнал новый анекдот и рассказал его своим слушателям. Определить, какую сумму должен получить или выплатить в конце концов каждый член клуба.

8. Организовать на диске информацию о сети автомобильных дорог, включающую наименование пунктов и расстояния участков между пунктами. Обеспечить корректировку информации для следующих случаев:

а) появился новый пункт, связанный дорогами с некоторыми из имеющихся пунктов;

б) закрыт некоторый участок дороги;

в) появился новый участок дороги;

г) перестал существовать некоторый пункт, но остались все дороги, идущие через него.

9. Постройте алгоритмы для преобразования друг в друга всех пар следующих представлений графа:

а) матрица смежности;

б) список ребер;

в) структура смежности;

г) матрица инциденций; сколько операций необходимо для каждого из обращений?

10. Мост в связном неориентированном графе $G=(V,E)$ − это ребро, удаление которого делает граф несвязным. Постройте алгоритм поиска в глубину для определения мостов в G, требующий $O(|V|+|E|)$ операций.

11. Поиск в ширину исследует ребра графа $G=(V,E)$ и порождает *BFS*-дерево следующим образом. Начинаем в произвольно выбранной вершине r (в корне) и проходим все ребра, инцидентные r; пусть эти ребра будут $(r, a[1])$, $(r, a[2])$,... $(r, a[k])$. Затем исследуем ребра, инцидентные $a[1], a[2], \ldots a[k]$. Пусть эти ребра будут $(a[1], a[1, 1])$, $(a[1], a[1, 2])$,... $(a[1], a[1, t1])$, $(a[2], a[2, t2])$, $(a[k], a[k, tk])$. Затем исследуем ребра, инцидентные $a[1, 1], a[1, 2], \ldots a[k,tk]$, и т.д. Этот процесс продолжаем до тех пор, пока не будут исследованы все ребра дерева. (Если G − несвязный граф, то процесс можно повторить и получить *BFS*-лес.) Постройте алгоритм для исследования неориентированного графа согласно данной схеме. (Указание: используйте очередь). Используйте этот тип прохождения графа для определения всех вершин, находящихся на фиксированном

расстоянии d от вершины s, в неориентированном графе, у которого каждое ребро имеет длину 1.

12. Разработать программу поиска остовного дерева грубым методом и методом поиска в глубину. Оценить эффективность программ на произвольных графах.

13. Разработать программу поиска остовного дерева грубым методом и методом поиска в ширину. Оценить эффективность программ на произвольных графах.

14. Эйлеров путь в неориентированном связном графе- это путь, который проходит по ребру в точности один раз. Разработайте алгоритм и программу отыскания эйлерова пути.

15. На плоскости заданы координаты N элементов, являющихся выводами печатной платы. Некоторые элементы связаны между собой. Требуется от элемента, имеющего наибольшую суммарную длину связей с другими элементами, построить контур, т.е. путь, не содержащий циклов. При построении контура использовать следующее правило: из всех возможных элементов, связанных с данным, в контур включается ближайший элемент.

16. Дана матрица смежности графа, содержащего n вершин. Требуется выяснить существует ли такой замкнутый путь, проходящий по ребрам графа, который проходит через все вершины по одному разу (так называемый Гамильтонов цикл) . Если такой путь существует, то построить его.

17. Написать программу формирования матрицы смежности для ориентированного графа.

18. Определить полустепени захода и исхода заданной вершины ориентированного графа.

19. Написать процедуру удаления вершины графа.

20. Написать процедуру обхода графа G (выдачи на печать всех его вершин) .

21. Написать процедуры объединения двух графов.

22. Разработать программу поиска остовного дерева грубым методом и с помощью алгоритма Прима Краскала. Оценить эффективность программ на произвольных графах.

23. Разработать программу поиска остовного дерева грубым методом и с помощью метода поиска в глубину. Оценить эффективность программ на произвольных графах.

24. Разработать программу поиска остовного дерева грубым методом и методом поиска в ширину. Оценить эффективность программ на произвольных графах.

25. Для заданного графа определите, является ли он связным.

26. Найдите все мосты заданного графа.

27. Для заданного графа определите, является ли он эйлеровым, и если да, то вывести эйлеров цикл.

28. Пусть $G= (V,E)$ – неориентированный связный граф. Предположим, что имеется только одна вершина, не имеющая заходящих в нее ребер, и одна вершина, не имеющая исходящих из нее ребер. Разработать алгоритм и программу для отыскания разреза между этими вершинами, который имеет наименьшее число ребер.

5. УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

Каждую задачу необходимо решить в соответствии с изученными алгоритмами обхода графа, реализовав программный код на языке C++. Рекомендуется воспользоваться материалами лекции 44, где подробно рассматриваются описания алгоритмов обхода графа, примеры разработки функций, реализующих алгоритмы обхода графа, на языке C++. Программу для решения каждого задания разработать методом процедурной абстракции, используя функции. Этапы решения сопроводить комментариями в коде. В отчете о лабораторной работе отразить разработку и обоснование математической модели решения задачи, представить результаты тестирования программ.

Каждую задачу реализовать в соответствии с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке C++;

- разработать контрольный тест к программе;
- отладить программу;
- составить отчет о лабораторной работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как связаны между собой различные способы представления графов?
2. Как от вида или представления графа зависит временная сложность алгоритмов поиска в глубину и в ширину?
3. Как при реализации в коде выполняется возвращение из тупиковых вершин при обходе графа?
4. Как выполняется обход в несвязном графе?
5. Распространяются ли понятия «поиск в глубину» и «поиск в ширину» на несвязный граф? Ответ обоснуйте.
6. Охарактеризуйте трудоемкость рекурсивного и нерекурсивного алгоритмов обхода графа.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Кнут, Д. Искусство программирования для ЭВМ Т.1, Основные алгоритмы / Д. Кнут – М.: Диалектика Вильямс 2020. – 760 с. ISBN 978-5-907144-23-1

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт – М.: ДМК Пресс, 2010 – 272 с.: ил. ISBN 978 5 94074 584 6

3. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М: Издательский дом «Вильямс», 2005. –1296 с. ISBN 5-8459-0857-4(рус)

4. Ахо, А. В. Структуры данных и алгоритмы / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. – М.: Вильямс, 2003. – 382 с. ISBN 10:0-201-00023-7

5. Седжвик, Р. Фундаментальные алгоритмы на C++. Анализ. Структуры данных. Сортировка. Поиск: пер. с англ. /Р. Седжвик. – К.: ДиаСофт, 2001. – 688 с. ISBN 5-93772-054-7

6. Хэзфилд, Р. Искусство программирования на C. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: пер. с англ. / Р. Хэзфилд,

Л. Кирби [и др.]. – К.: ДиаСофт, 2001. – 736 с. ISBN 966-7393-82-8, 0-672-31896-2

7. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си (+ CD): учеб. Пособие / Б.С. Хусаинов. – М.: Финансы и статистика, 2004. – 464 с. ISBN: 5-279-02775-8

Структуры и алгоритмы обработки данных. Алгоритмы на графах общего вида (Поиск в глубину и поиск в ширину): методические указания к выполнению лабораторной работы №14 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки: 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия»

ВАСИЛИЙ КОНСТАНТИНОВИЧ ГУЛАКОВ
АНДРЕЙ ОЛЕГОВИЧ ТРУБАКОВ
ЕВГЕНИЙ ОЛЕГОВИЧ ТРУБАКОВ

Научный редактор В.В. Конкин
Редактор издательства Л.Н. Мажугина
Компьютерный набор В.К. Гулаков

Темплан 2020 г., п. 205

Подписано в печать Формат 1/16
Усл.печ.л. 1,04 Уч.-изд.л. 1,04

Издательство Брянского государственного технического университета
241035, Брянск, бульвар им.50-летия Октября, 7, БГТУ, тел. 58-82-49
Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16