Лабораторная работа № 8

Методические указания

Графы

Цель работы – реализовать алгоритмы обработки графовых структур: поиск различных путей, проверку связности, построение остовых деревьев минимальной стоимости.

Краткие теоретические сведения

Граф – это конечное множество вершин и ребер, соединяющих их, т. е.:

$$G = < V,E >,$$

где V – конечное непустое множество вершин; E – множество ребер (пар вершин).

Если пары E (ребра) имеют направление, то граф называется ориентированным (орграф), если иначе - неориентированный (неорграф). Если в пары E входят только различные вершины, то в графе нет петель. Если ребро графа имеет вес, то граф называется взвешенным. Степень вершины графа равна числу ребер, входящих и выходящих из нее (инцидентных ей). Неорграф называется связным, если существует путь из каждой вершины в любую другую.

Обозначим количество вершин как $n = ¦ V ¦$, а количество ребер как $m = ¦ E ¦$.

Графы в памяти могут представляться различным способом. Один из видов представления графов – это матрица смежности B(n*n); В этой матрице элемент b[i,j]=1, если ребро, связывающее вершины Vi и Vj существует и b[i,j]=0, если ребра нет. У неориентированных графов матрица смежности всегда симметрична.

Во многих случаях удобнее представлять граф в виде так называемого списка смежностей. Список смежностей содержит для каждой вершины из множества вершин V список тех вершин, которые непосредственно связаны с этой вершиной. Каждый элемент (ZAP[u]) списка смежностей является записью, содержащей данную вершину и указатель на следующую запись в списке (для последней записи в списке этот указатель – пустой). Входы в списки смежностей для каждой вершины графа хранятся в таблице (массиве) (BEG [u])

Пример неориентированного графа приведен на рис. 1, матрица смежности для этого графа – на рис. 2, а список смежности – на рис. 3.

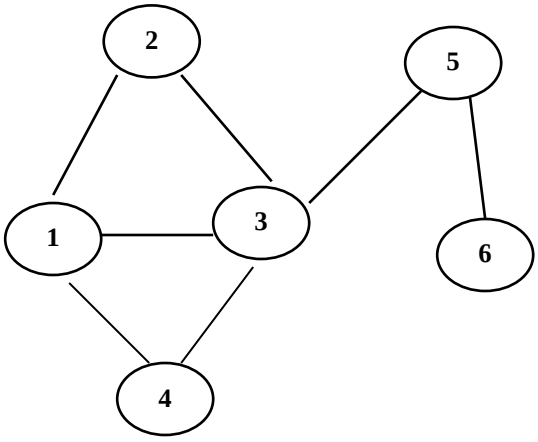

Рис. 1

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

Рис. 2

BEG[u] ZAP[u]

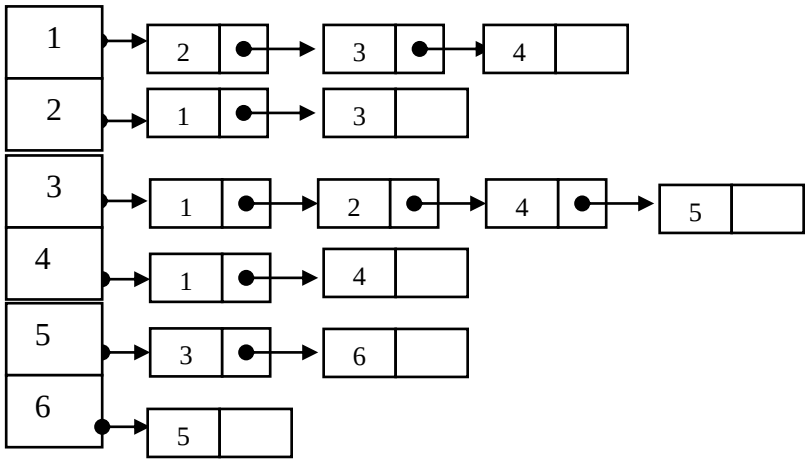

Рис. 3

Перечислим основные операции по работе с графовыми структурами:

- поиск кратчайшего пути от одной вершины к другой (если он есть);
- поиск кратчайшего пути от одной вершины ко всем другим;
- поиск кратчайших путей между всеми вершинами;
- поиск эйлерова пути (если он есть);
- поиск гамильтонова пути (если он есть).

В основе построения большинства алгоритмов на графах лежит систематический перебор вершин графа, при котором каждая вершина просматривается в точности один раз, а количество просмотров ребер графа ограничено заданной константой (лучше – не более одного раза).

Один из основных методов проектирования графовых алгоритмов – это поиск (или обход графа) в глубину (depth first search, DFS), при котором, начиная с произвольной вершины **v0**, ищется ближайшая смежная вершина **v**, для которой, в свою очередь, осуществляется поиск в глубину (т.е. снова ищется ближайшая, смежная с ней вершина) до тех пор, пока не встретится ранее просмотренная вершина, или не закончится список смежности вершины **v** (то есть вершина полностью обработана). Если нет новых вершин, смежных с **v**, то вершина **v** считается использованной, идет возврат в вершину, из которой попали в вершину **v,** и процесс продолжается до тех пор, пока не получим **v = v0**. Иными словами, поиск в глубину из вершины **v** основан на поиске в глубину из всех новых вершин, смежных с вершиной **v**.

Путь, полученный методом поиска в глубину, в общем случае не является кратчайшим путем из вершины **v** в вершину **u**. Это общий недостаток поиска в глубину [1,2]. На рис. 4 показан путь, полученный обходом графа в глубину.

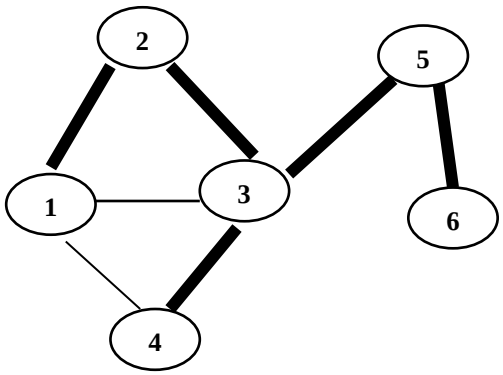

Рис. 4

Указанного недостатка лишен другой метод обхода графа – поиск в ширину (breadth first search, BFS). Обработка вершины **v** осуществляется путем просмотра сразу всех новых соседей этой вершины. При этом полученный путь является кратчайшим путем из одной вершины в другую [1].

На рис. 5 показан путь, найденный методом поиска в ширину

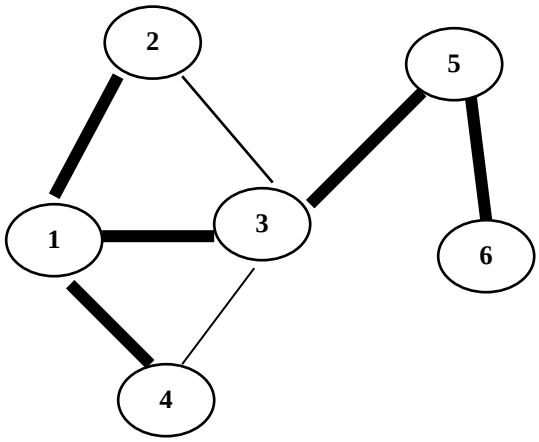

Рис. 5

При использовании алгоритмов DFS и BFS графы обходят разными способами, получая при этом некоторые подграфы, которые имеют специфические названия: каркасы, остовы или стягивающие деревья [1]. Все вершины исходного графа входят в полученное стягивающее дерево, а все ветви дерева – это множество ребер из всех

множеств ребер графа (см. рис. 4, 5).

Произвольный путь в графе, проходящий через каждое ребро графа точно один раз, называется эйлеровым путем. При этом, если по некоторым вершинам путь проходит неоднократно, то он является непростым. Если путь замкнут, то имеем эйлеров цикл. Для существования эйлерова пути в связном графе необходимо и достаточно, чтобы граф содержал не более двух вершин нечетной степени [2,3]

Путь в графе, проходящий в точности один раз через каждую вершину графа (а не каждое ребро) и соответствующий цикл называются гамильтоновыми и существуют не для каждого графа, как и эйлеров путь. В отличие от эйлеровых путей неизвестно ни одного простого необходимого и достаточного условия для существования гамильтоновых путей. Неизвестен даже алгоритм полиномиальной сложности, проверяющий существование гамильтонова пути в произвольном графе.

Проблема существования гамильтонова пути принадлежит к классу так называемых NP-полных задач [2, 3].

Поиск кратчайших путей до всех вершин из одной указанной вершины для взвешенного орграфа (имеющего значение, т.е. вес или стоимость, ребра) с неотрицательными ребрами осуществляется с использованием алгоритма Дейкстры. Алгоритм Дейкстры основан на выборе для включения в путь всякий раз той вершины, которая имеет наименьшую оценку кратчайшего пути (по весам ребер), то есть наименьший путь до этой вершины из всех возможных путей, которые были рассмотрены ранее [1].

Алгоритм Беллмана-Форда позволяет решить задачу о поиске кратчайших путях из одной выбранной вершины ко всем остальным вершинам при любых весах ребер, в том числе и отрицательных. Сначала ищется путь от выбранной вершины ко всем вершинам, связными с ней (аналогично поиску в ширину), а затем – кратчайший путь ко всем остальным вершинам, с попыткой последовательно пройти к ним, т.е. сначала через первую вершину, затем через вторую, через третью и так далее до последней вершины. Кроме того, алгоритм возвращает TRUE, если в графе нет цикла отрицательного веса, достижимого из данной вершины [2].

Для поиска кратчайших путей между всеми вершинами используется алгоритм Флойда-Уоршалла. По алгоритму Флойда-Уоршалла сначала ищется кратчайший путь от одной вершины ко всем вершинам, доступным из нее, затем проводятся те же действия, но пытаясь пройти от этой вершины ко всем доступным из нее, проходя каждый раз через новую вершину (сначала через первую, затем – через вторую и т.д.). Таким образом обрабатываются все вершины. [2].

Как уже было сказано, остовым деревом графа является дерево, содержащие все вершины графа. Стоимостью этого дерева является сумма стоимостей всех ребер [1].

Наиболее часто для построения остовых деревьев минимальной стоимости используют два алгоритма: алгоритм Крускала и алгоритм Прима.

Алгоритм Крускала, который при просмотре графа в лесу ребер, состоящему из нескольких связных компонент (деревьев), добавляет каждый раз минимальное ребро, концы которого лежат в разных компонентах леса.

Алгоритм Прима при просмотре графа строит одно дерево, в которое каждый раз добавляется ребро минимального веса [1].

Например, типичное применение остовых деревьев минимальной стоимости – это построение коммуникационных линий между городами, где стоимости ребер – это стоимость коммуникационных сетей.

Задание

Обработать графовую структуру в соответствии с заданным вариантом. Обосновать выбор необходимого алгоритма и выбор структуры для представления графов. Ввод данных осуществить на усмотрение программиста. Результат выдать в графической форме.

Указания к выполнению работы

При отладке программы необходимо проверить правильность ввода данных. Программа должна адекватно реагировать на неверный ввод, в том числе на «пустой» ввод. Необходимо проверять поиск несуществующих путей в графе, использовать различные варианты графов для проверки правильности работы программы.

При разработке интерфейса программы следует предусмотреть:

- указание типа, формата и диапазона вводимых данных,
- указание действий, производимых программой,
- наличие пояснений при выводе результата,
- вывод графов осуществить в графическом виде (или предложить иную визуализацию в виде графа)
- вывод времени выполнения программы и объема требуемой памяти при использовании различных структур для представления графа.

При тестировании программы необходимо:

- проверить правильность ввода и вывода данных (т.е. их соответствие требуемому типу и формату), обеспечить адекватную реакцию программы на неверный ввод данных;
- обеспечить вывод сообщений при отсутствии входных данных («пустой ввод»);
- проверить правильность выполнения операций;
- предусмотреть вывод сообщения при поиске несуществующих путей в графе.

Содержание отчета

В отчете по лабораторной работе должен быть обоснован выбор формы представления графа и алгоритма, использованного для решения поставленной задачи. Следует указать сложность используемого алгоритма. Необходимо также предложить вариант реальной задачи, для которой можно использовать разработанную программу.

После выполнения работы студент должен уметь ответить на следующие вопросы**:**

1. Что такое граф?
2. Как представляются графы в памяти?
3. Какие операции возможны над графами?
4. Какие способы обхода графов существуют?
5. Где используются графовые структуры?
6. Какие пути в графе Вы знаете?
7. Что такое каркасы графа?

Отчет представляется в электронном или печатном виде.

Список рекомендуемой литературы

1. *Ахи А., Хопкрофт Д., Ульман Д.* Структуры данных и алгоритмы.: Пер. с англ. М.: Издат. дом «Вильямс», 2000. С. 183–225
2. *Кормен Т., Лейзерсон Ч., Ривест Р.*., Алгоритмы: построение и анализ: Пер. с англ., М.: МЦНМО, 2001. С. 88–91, 435–523
3. *Седжвик Р.* Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск/Алгоритмы на графах: Пер. с англ. СПб.: «ДиаСофтЮП», 2003. С. 673–1000