Лабораторная работа №7.

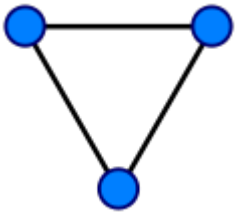
Базовые алгоритмы на графах.

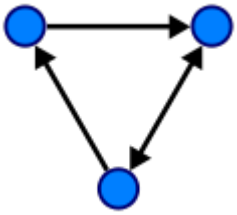
Цель работы: изучение алгоритмов поиска на графах, получение навыков разработки программ с использованием нелинейных структур данных.

Теоретические сведения

Основные определения

Граф G часто определяется как пара множеств (V, E), где V – непустое множество вершин, а E –множество пар различных элементов множества V. Если речь идет о неупорядоченных парах, то граф G называется *неориентированным* (а множество E называется множеством *ребер*). Если же E состоит из упорядоченных пар, это означает, что мы имеем дело с *ориентированным графом* (орграфом), вершины которого обычно называют *узлами*, а упорядоченные пары – *дугами*. На следующем рисунке показаны примеры неориентированного (слева) и ориентированного (справа) графов





Пусть имеются две вершины v_1 и v_2, причем $e = (v_1, v_2)$ – соединяющее эти вершины ребро. Тогда ребро e называется *инцидентным* вершинам v_1 и v_2, а сами вершины – *смежными*. Множество вершин, смежных с некоторой вершиной, называется *множеством смежности* этой вершины.

Чередующаяся последовательность вершин и ребер $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$, в которой любые два соседних элемента инцидентны, называется *маршрутом* длины k. При этом, если все ребра различны, то маршрут называется *цепью*, а если все вершины различны – *простой цепью*. В орграфе цепь часто называют *путем* (соответственно, элементами пути являются узлы и дуги). Замкнутая цепь называется *циклом*. *Расстоянием* между двумя вершинами является цепь кратчайшей длины. Если граф или орграф является *взвешенным* (то есть каждому ребру или дуге приписано число, называемое *весом ребра*), то можно ввести понятие *стоимости маршрута*, как суммы весов всех ребер маршрута. Таким образом, кратчайшим путем в орграфе является путь минимальной стоимости.

Две вершины графа называются *связанными*, если существует соединяющая их цепь. Граф, в котором все вершины связаны, называется *связным*. Поиск компонент связности графа является важной задачей, решаемой при построении отказоустойчивых сетевых и вычислительных структур.

Деревом называется связный граф без циклов. Пусть p – число вершин в графе G, а q – число ребер. Тогда для древовидного графа справедливо соотношение $q = p - 1$. Особое значение для решения многих задач имеет *минимальное остовное дерево*. Минимальным остовным деревом связного взвешенного графа называется дерево, состоящее из всех вершин графа, и некоторых его ребер, причем сумма весов ребер является наименьшей среди всех возможных остовных деревьев.

Способы программного представления графов

Существуют различные способы задания графа для компьютерной обработки. Выбор конкретного способа зависит от специфики решаемой задачи, структуры алгоритма, от объема памяти, доступной для хранения информации о графе и от скорости выполнения конкретных операций по доступу к элементам определения графа. Рассмотрим некоторые из этих способов.

Матрица смежности

Матрица смежности A размером $p \times p$ состоит из элементов $A[i][j]$ таких, что $A[i][j] = 1$, если вершина v_i является смежной с вершиной v_j. В противном случае, $A[i][j] = 0$. Объем памяти, необходимый для хранения данных в матрице A оценивается величиной $O(p^2)$. В общем случае, применение матрицы смежности можно считать разумным для сильно связных графов.

Для взвешенного орграфа матрицу смежности обычно определяют иначе. Во взвешенном графе $A[i][j] = \infty$, если v_i не является смежной с вершиной v_j. Если же из узла v_i в узел v_j ведет дуга с весом w_{ij}, то $A[i][j] = w_{ij}$. В частности, $A[i][i] = 0$. Для неориентированного графа $A[i][j] = A[j][i]$.

Матрица инцидентности

Матрица инцидентности представляет собой матрицу I размером $p \times q$, в которой $I[i][j] = 1$, если вершина v_i инцидентна ребру e_j. В противном случае, $I[i][j] = 0$. Объем памяти, необходимый для хранения данных в матрице смежности оценивается величиной $O(pq)$. Очевидно, что если граф имеет много вершин, но относительно мало ребер, хранение его в виде матрицы инцидентности более экономично, чем использование матрицы смежности.

Список смежности

С точки зрения удобства использования при записи алгоритмов во многих случаях наиболее привлекательным оказывается способ хранения информации о графе в виде списков смежности, или списков примыканий. С точки зрения структур данных, используемых в программировании, списочная структура представляет собой массив из p указателей на списки смежных вершин. Для неориентированного графа затраты на хранение оцениваются величиной $O(p+2q)$. Для орграфа получим $O(p+q)$. Использование списков смежности позволяет хранить только информацию о реально присутствующих ребрах или дугах, однако за это приходится платить накладными расходами, связанными с организацией списка. Требуется дополнительная память для указателей на элементы списка, кроме того, доступ к информации требует больше времени. Если требуется обеспечить возможность частых изменений структуры графа в ходе работы программы (например, добавление или удаление вершин и ребер), то использование списочных структур, безусловно, является, наиболее эффективным вариантом.

Простым (и во многих случаях – вполне приемлемым с точки зрения затрат) способом представления графа также является *массив ребер* (или дуг для орграфа).

На рис. 1 приведена диаграмма неориентированного графа и способ его задания в форме списка смежности. Вершины и ребра здесь нумеруются с нуля.

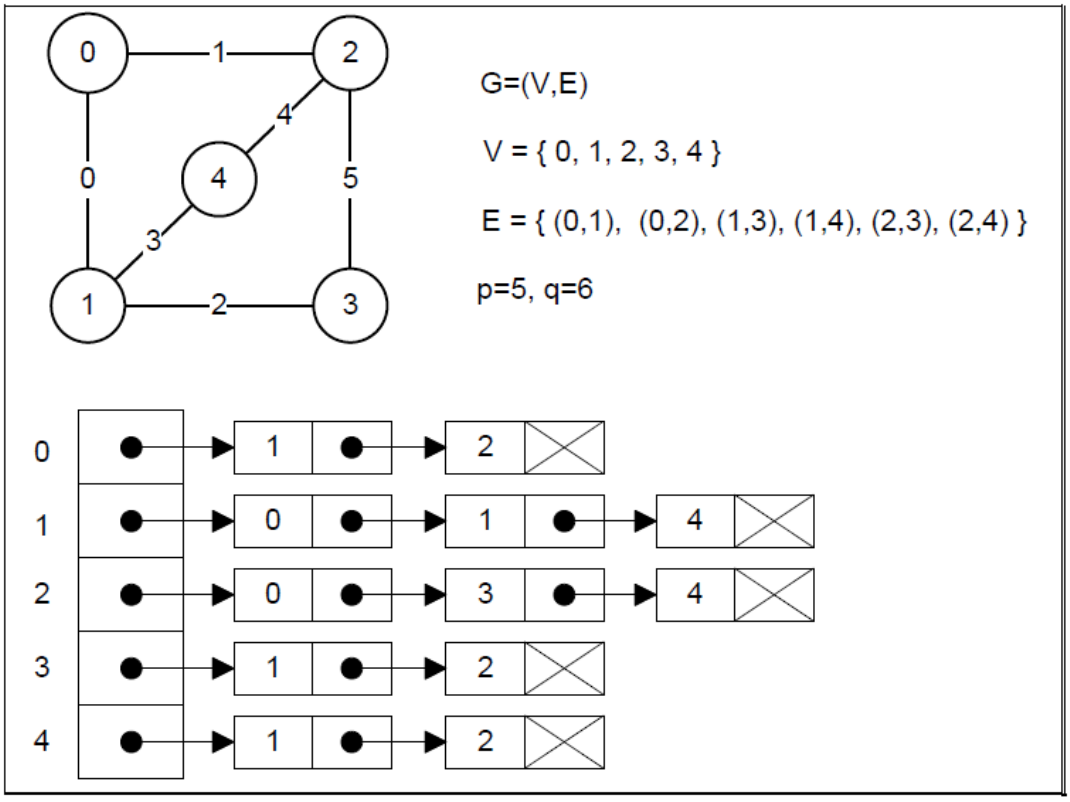


Рис. 1. Взвешенный неориентированный граф и его представление.

Задачи на графах

Наиболее распространенными задачами теории графов являются следующие:

1. нахождение кратчайшего пути из одной вершины графа в другую;
2. нахождение минимального остовного дерева графа;
3. поиск вершины, удовлетворяющей заданным требования (обход графа);
4. нахождение компонент связности графа.

Поиск кратчайших путей на графе

Путь, соединяющий две вершины графа v_i и v_j, будем считать *кратчайшим*, если суммарный вес всех ребер, по которым проходит путь, минимален. Задача поиска кратчайшего пути является одной из наиболее распространенных. Легко понять, что в общем случае эффективность алгоритма поиска кратчайшего пути будет в основном определяться числом его вершин и ребер.

В качестве примера вкратце рассмотрим здесь алгоритм Дейкстры (Dijkstra, 1959) как один из наиболее распространенных. Для изучения других алгоритмов и более глубокого анализа их эффективности см. лекционные материалы по курсу АСД и специальную литературу.

Алгоритм Дейкстры используется для поиска кратчайшего пути на взвешенных неориентированных графах. Идея метода состоит в том, что каждой вершине графа присваивается метка, которая показывает минимальное известное (на данном этапе алгоритма) расстояние от текущей вершины до начальной. На каждом шаге алгоритм «посещает» одну из вершин и пытается уменьшить значения меток. Он завершается, когда все вершины посещены.

Алгоритм можно разделить на несколько этапов. При инициализации для начальной вершины устанавливается метка 0, для всех остальных вершин — метки, равные бесконечности. Все вершины графа при этом помечаются как непосещённые. На следующем шаге из непосещённых вершин выбирается вершина v, имеющая минимальную метку. Рассматриваются все маршруты, в которых v является предпоследним пунктом. Для каждого соседа вершины v (кроме посещённых) рассчитывается новая длина пути – это сумма значений текущей метки v и длины ребра. Если полученное значение меньше метки соседа, метка заменяется. После

обхода всех соседей, вершина v отмечается как посещенная и мы переходим к следующему шагу. Работа алгоритма иллюстрируется на следующем рисунке.

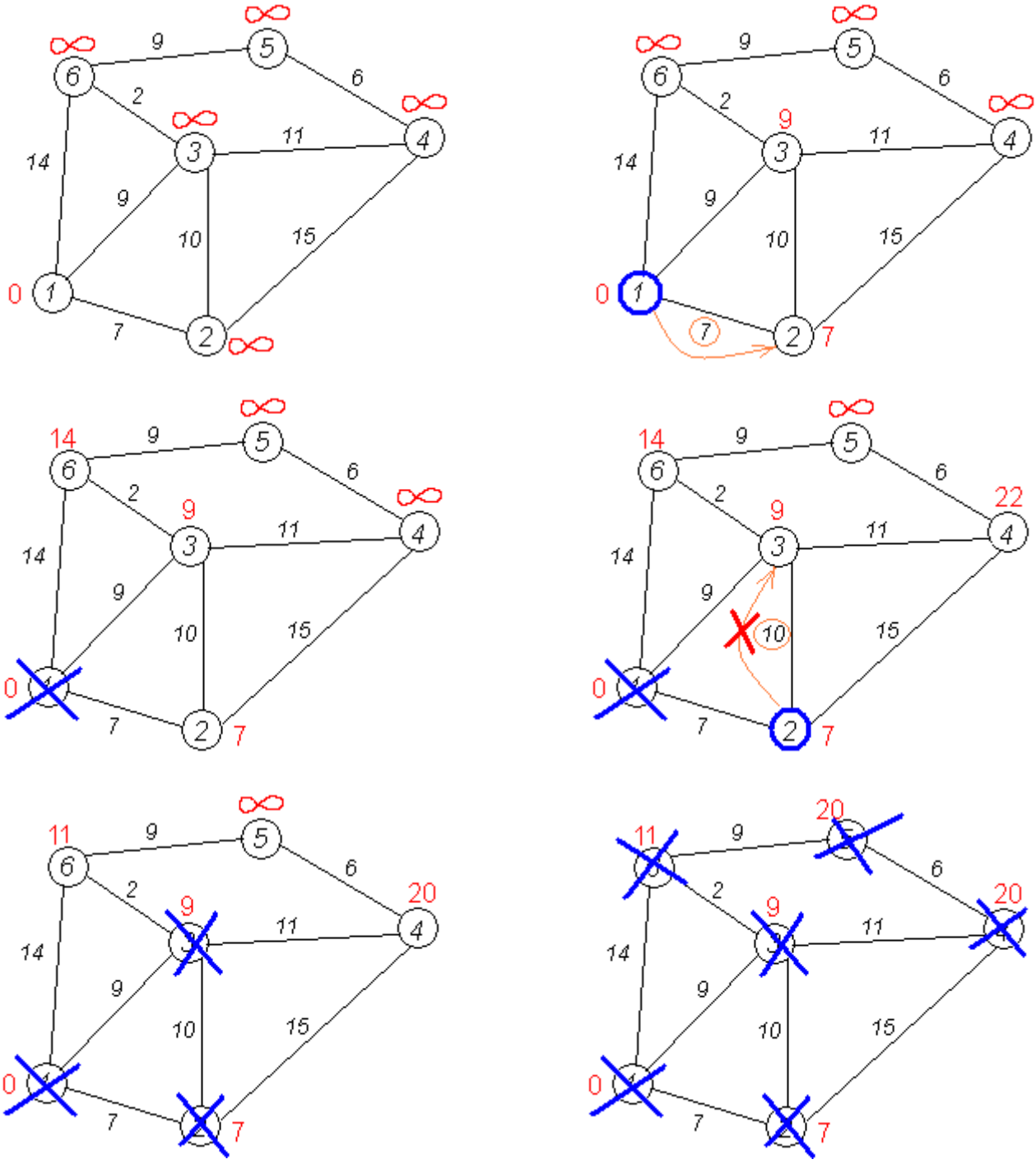


Рис. 2. Иллюстрация работы алгоритма Дейкстры для поиска кратчайших путей на графе. Черным цветом обозначены номера вершин графа и значения весов всех его ребер. Красным цветом обозначены текущие значения меток всех вершин. Подробное описание всех шагов алгоритма для данного случая можно найти в презентации к лекции 9.

Построение минимального остовного (покрывающего) дерева

Остовным (или покрывающим) деревом связного неориентированного графа называют ациклический связный подграф, в который входят все его вершины. Это понятие иллюстрируется следующим рисунком, на котором изображен неориентированный взвешенный граф и выделена часть ребер, образующих остовное дерево.

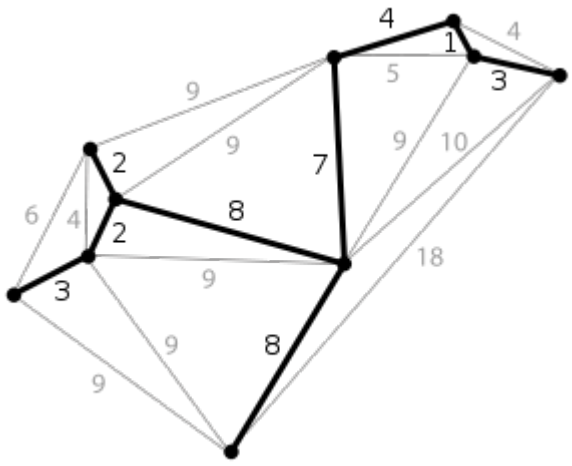


Рис. 3. Неориентированный граф и его остовное (покрывающее) дерево.

Из рисунка хорошо видно, что для одного и того же графа можно построить несколько различных остовных деревьев. В теории графов доказывается, что любое остовное дерево для графа, содержащего p вершин, включает в себя ровно $p-1$ ребер. Согласно формуле Кэли, общее количество различных остовных деревьев в полном графе равно p^{p-2}.

Среди всех остовных деревьев заданного графа выделяется одно, называемое *минимальным*. Минимальное остовное дерево (МОД) — это остовное дерево, имеющее минимальный возможный вес, то есть сумму весов всех входящих в него рёбер. Задача о нахождении МОД часто встречается в подобной постановке. Предположим, что имеется сеть населенных пунктов, которые необходимо соединить дорогами, причем так, чтобы можно было добраться из одного пункта в другой напрямую или через другие пункты. Дороги строятся между парами населенных пунктов и известна стоимость строительства каждой такой дороги. Требуется определить, какие именно дороги нужно строить, чтобы минимизировать общую стоимость строительства.

Существует несколько методов нахождения МОД. Наиболее известные среди них – это алгоритмы Прима, Краскала и Борувки. Далее мы кратко рассмотрим метод Краскала, более подробное изложение этого и других алгоритмов имеется в материалах лекции 8.

Общая формулировка алгоритма Краскала:

1) Множество рёбер дерева устанавливаем пустым (вершины без ребер).
2) Из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появления в нём цикла:
   2.1) выбираем ребро минимального веса
   2.2) добавляем его к уже имеющемуся множеству.
3) Если больше рёбер нет - алгоритм завершён, иначе переходим к п. 2.

Пошаговая работа этого алгоритма иллюстрируется следующим рисунком.

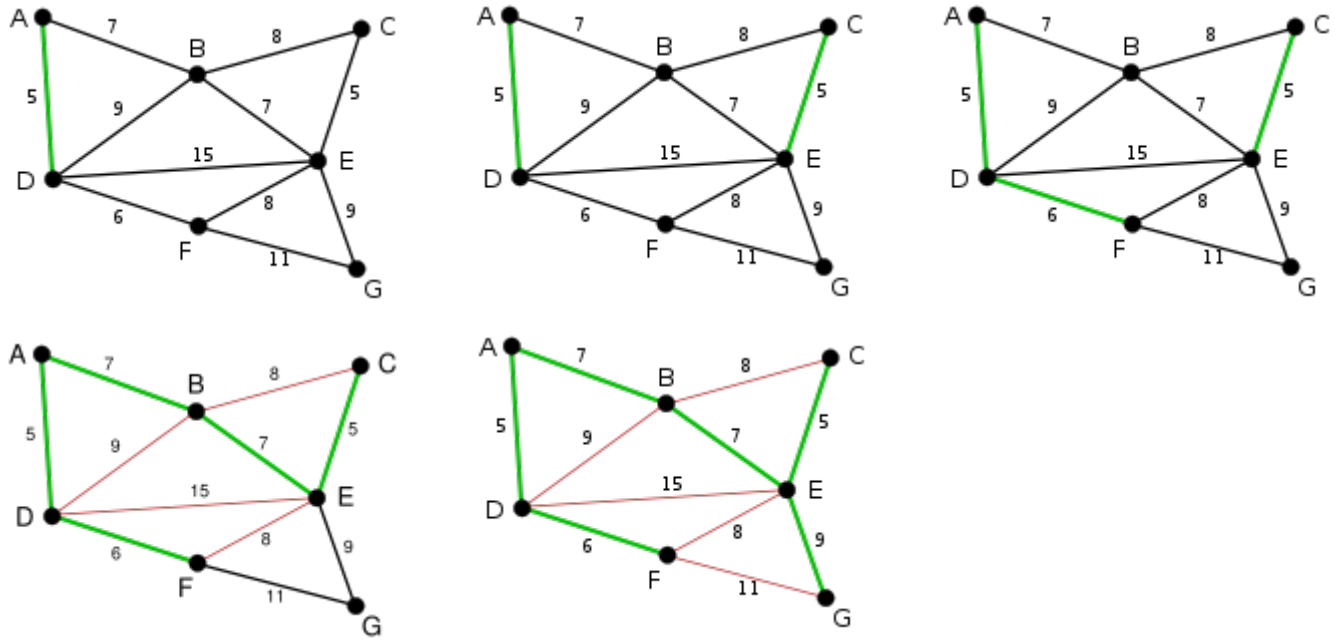


Рис. 4. Алгоритм Краскала для нахождения минимального остовного дерева. Зеленым цветом выделены ребра, входящие в МОД.

Неинформированный поиск в графе

К наиболее часто встречающимся задачам на графах можно также отнести задачу поиска элементов, удовлетворяющих наперед заданным условиям. В качестве простейшего примера можно рассмотреть нахождение вершины, хранящей некоторое заданное значение. Если никакой дополнительной информации о расположении элементов в графе не известно, для решения этой задачи часто используют методы неинформированного, или «слепого» поиска. Наиболее популярные алгоритмы – поиск в ширину, поиск в глубину, поиск в глубину с итеративным углублением и др. Как правило, эти алгоритмы формулируются для деревьев, однако с небольшими модификациями могут применяться на графах с любой структурой. Рассмотрим далее кратко два ключевых алгоритма, другие методы могут быть найдены в литературе и материалах лекции 10.

Поиск в ширину (breadth-first search) — это стратегия поиска, при которой вначале развёртывается (то есть посещается) корневой узел дерева, затем — все его преемники (по очереди), далее – преемники этих преемников и т.д. Прежде чем происходит развёртывание (посещение) каких-либо узлов дерева на следующем уровне, развёртываются все узлы на заданной глубине поиска (см. рис. 5).

Данный алгоритм является полным. Если все действия имеют одинаковую стоимость, поиск в ширину является оптимальным. Общее количество развернутых узлов (т.е. временная сложность алгоритма) равно $O(b^{d+1})$, где b — коэффициент ветвления, d — глубина поиска.

Для реализации поиска в ширину удобно использовать очередь queue. В начале работы алгоритма в очередь помещается только корневой узел. На каждой итерации основного цикла из начала очереди извлекается узел. Если этот узел является целевым (т.е. хранит искомое значение), то поиск на этом останавливается. В противном случае извлеченный узел развёртывается, то есть все его преемники по очереди добавляются в конец очереди. Алгоритм работает до тех пор, пока не будет найден целевой узел (успешное завершение), или пока в очереди не останется ни одного узла (безуспешное завершение).

Поиск в глубину (depth-first search) — иная стратегия поиска, при которой развёртывается всегда самый глубокий узел в текущей периферии дерева поиска. То есть, вначале развертывается (анализируется) первый по списку преемник текущего узла, далее — первый преемник приемника и т.д. Развёрнутые узлы удаляются из периферии, поэтому в дальнейшем поиск «возобновляется» со следующего самого поверхностного узла, который всё ещё имеет неисследованных преемников. Поиск в глубину удобнее всего может быть реализована с помощью стека stack или с помощью рекурсивной функции (см. материалы лекции 10).

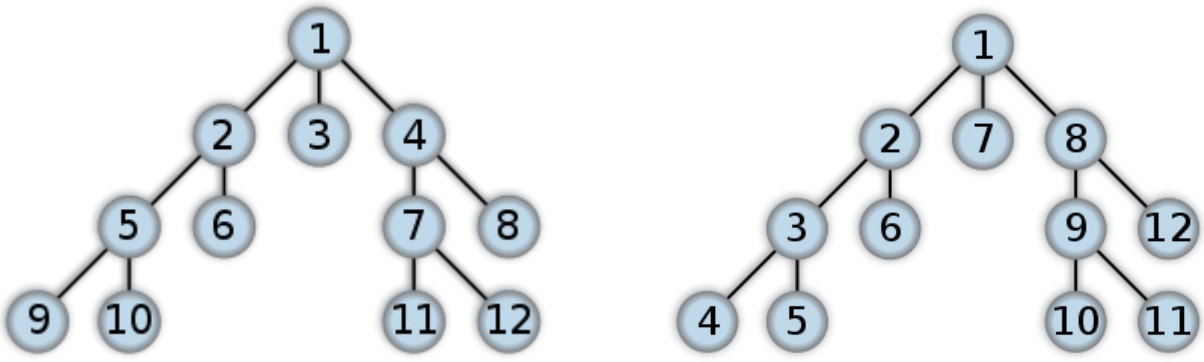


Рис. 5. Порядок обхода элементов дерева при поиске в ширину (слева) и в глубину (справа).

Задание

1) Реализуйте неориентированный односвязный граф в виде программной структуры данных. Для реализации используйте любой из описанных выше способов (матрица смежности, матрица инцидентности, список смежности, множество вершин и ветвей).
2) Реализуйте алгоритм по индивидуальному заданию.
3) Добавьте в программу код, иллюстрирующий пошаговую работу алгоритма, например, выводящий на экран порядок посещения узлов (вершин) графа, последовательность построения МОД и т.д.
4) Проверьте корректность работы алгоритма на разных наборах входных данных.

Варианты заданий

1. Алгоритм Краскала для построения минимального остовного дерева.
2. Алгоритм Прима для построения минимального остовного дерева.
3. Алгоритм Борувки для построения минимального остовного дерева.
4. Алгоритм Дейкстры для поиска кратчайшего пути на графе.
5. Алгоритм Левита для поиска кратчайшего пути на графе.
6. Волнового алгоритма трассировки пути.
7. Неинформированный поиск с обходом в ширину.
8. Неинформированный поиск с обходом в глубину.
9. Поиск в глубину с итеративным углублением.
10. Поиск с ограничением глубины.