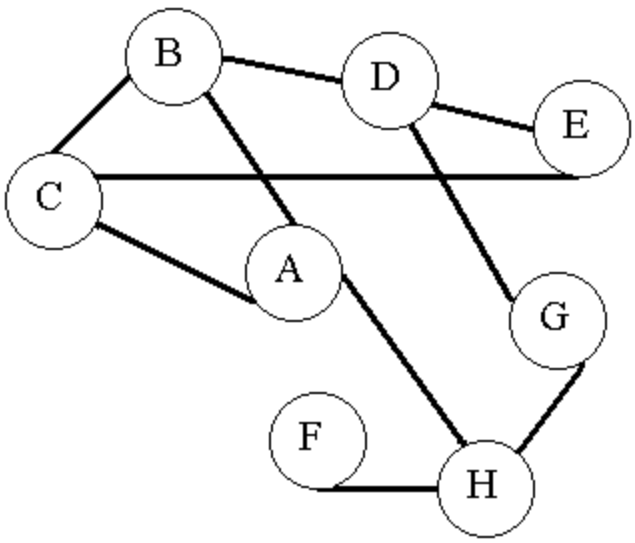
Алгоритмы обхода графа.

Представить граф в памяти компьютера можно по-разному. Мы будем говорить только о двух способах: матрице смежности и списках смежных вершин. Матрица смежности имеет размерность $N * N$, где N – число вершин графа. Элемент $A[i,j]$ этой матрицы считается равным 1, если вершины с номерами i и j связаны между собой ребром, и нулю, если не связаны. Матрица является удачным выбором в тех случаях, когда количество ребер в графе сравнимо с квадратом числа вершин (такие графы называют плотными). Если же граф разреженный, то обычно используют списки смежных вершин. Для каждой вершины заводится свой список, в который помещаются вершины (номера), связанные с данной вершиной ребрами. Все списки объединены в массив (одномерный, длины N).
Описанные представления могут быть использованы как для ориентированных, так и для неориентированных графов.
Графы возникают в ряде практически значимых задач. Эти задачи можно разделить на классы. В частности, можно выделить задачи поиска или обхода (в т.ч. установление связности, достижимости), задачи отыскания кратчайшего пути, задачи поиска подмножества вершин, удовлетворяющих определенным условиям, задачи о раскраске графа.
Изучение графов мы начнем с алгоритмов, позволяющих обойти все вершины графа. Разумеется, если граф несвязен, придется выполнять обход в каждой компоненте связности по отдельности. Алгоритмы обхода также называют алгоритмами поиска (действительно, в отличие от дерева, упорядоченность в графе в общем случае ввести нельзя, и обнаружить некоторый элемент можно только «последовательным поиском», в неудачном случае равнозначным обходу всего графа).

Пример.
Один из базисных алгоритмов, составляющий основу многих других – алгоритм поиска в ширину. Пусть задан граф G(V, E) и фиксирована **начальная вершина** s. Алгоритм поиска в ширину перечисляет все достижимые из s (если идти по ребрам) вершины в порядке возрастания расстояния от s. Расстоянием считается длина (число ребер) кратчайшего пути. В процессе поиска из графа выделяется часть, называемая «деревом поиска в ширину» с корнем s. Она содержит все достижимые из s вершины, и только их. Для каждой из них путь из корня в дереве поиска будет одним из кратчайших путей (из начальной вершины) в графе. Алгоритм применим и к ориентированным, и к неориентированным графам. Название объясняется тем, что в процессе поиска мы идем вширь, а не вглубь. Сначала перечисляются вершины, наиболее близкие к s (т.е. находящиеся на расстоянии 1), затем те, которые удалены от нее чуть больше (т.е. на расстояние 2), и т. д.
Для наглядности будем считать, что в процессе работы алгоритма вершины графа могут быть белыми, серыми и черными. Вначале они все белые, но в ходе работы алгоритма белая вершина может стать серой, а серая – черной (но не наоборот). Повстречав новую белую вершину, алгоритм красит ее в серый цвет и «запоминает», помещая в очередь уже обнаруженных вершин. Затем производится анализ запомненных вершин: из начала очереди выбирается очередная вершина, и перебираются все вершины, смежные с ней (т.е. непосредственно соединенные ребром). Если смежная вершина белая, ее перекрашивают в серый цвет и помещают в очередь, если же она имеет другой цвет, то это означает, что она уже была обнаружена на другом (как минимум, не более длинном) пути из вершины s. Когда все вершины, смежные с данной, будут просмотрены, ее удаляют из очереди и перекрашивают в черный цвет.

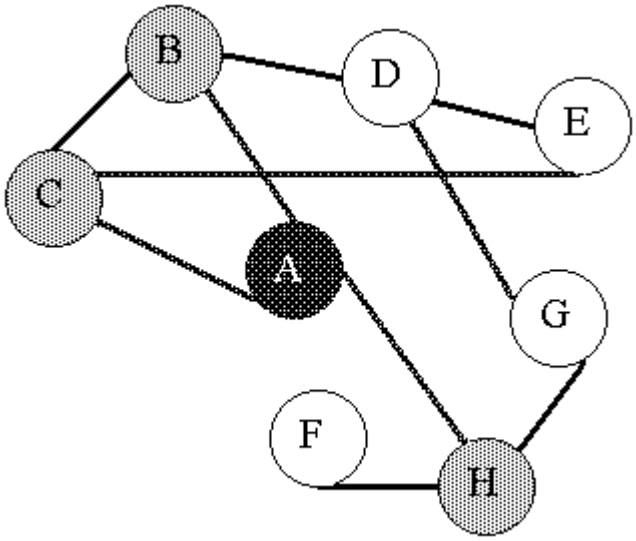


После этого из очереди выбирается следующая вершина, и описанный выше процесс продолжается – до тех пор, пока очередь не опустеет.
Таким образом, окрашенные (серые или черные) вершины – это те, которые уже обнаружены. Поддерживается следующее свойство: если $(u,v) \in E$ и u – черная, то v – серая или черная вершина. Поэтому только серые вершины могут иметь смежные необнаруженные вершины (и только серые вершины находятся в очереди).

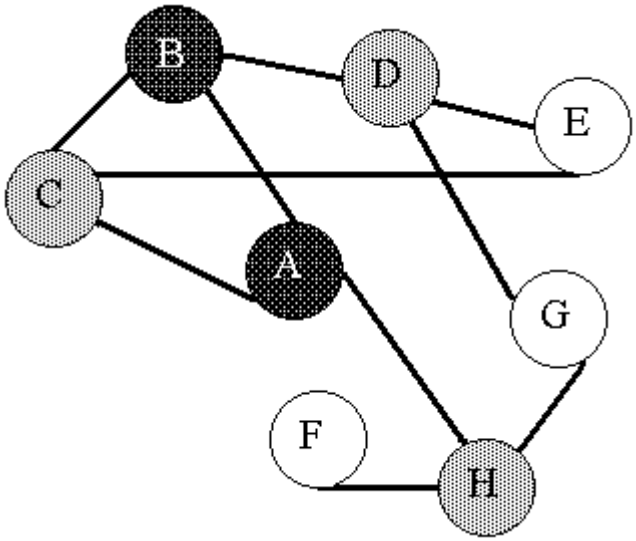
Вначале дерево поиска состоит только из корня – начальной вершины *s*. Как только алгоритм обнаруживает новую белую вершину *v*, смежную с ранее найденной вершиной *u*, вершина *v* (вместе с ребром (*u*,*v*)) добавляется к дереву поиска. Каждая вершина обнаруживается только однажды.

Продемонстрируем работу алгоритма поиска в ширину на следующем графе (см. рис.1).

Предположим, что «стартовой» вершиной будет вершина A. Все вершины, включая A, должны быть изначально покрашены в белый цвет. Затем мы красим стартовую вершину в серый цвет и помещаем ее в очередь. На этом этапе никакие другие вершины в очередь не помещаются: можно считать, что на этом этапе мы перебираем вершины, находящиеся на нулевом расстоянии от стартовой (как понятно, такая вершина только одна – она сама).

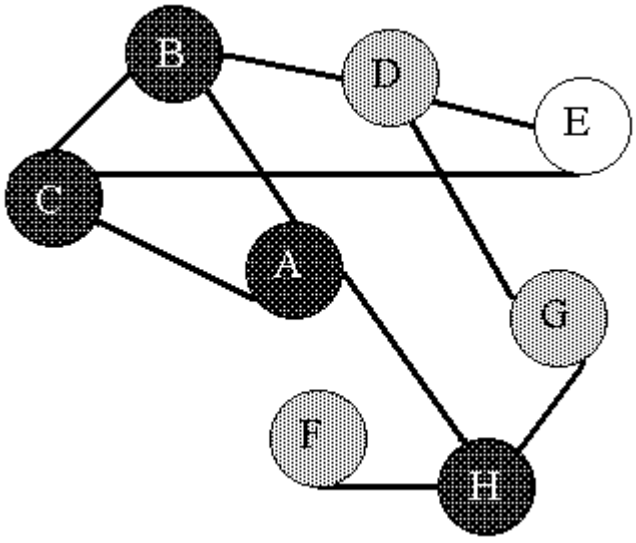


Далее мы начинаем просматривать очередь и выбираем из нее первую (и единственную на данный момент) вершину – A. Теперь, используя список смежных вершин для A (или матрицу смежности, или какое-то другое представление графа – какое именно, значения не имеет), перебираем вершины, находящиеся от нее на расстоянии 1. Как видно из рисунка, это будут вершины B, C и H. Поочередно красим эти вершины в серый цвет и помещаем их в очередь. Когда список смежных вершин для вершины A окажется исчерпанным, ее нужно будет удалить из очереди и покрасить в черный цвет – чтобы исключить дальнейшие обращения к ней



(рис. 2).

Итак, на этот момент список обхода состоит из (черной) вершины A, очередь – из (серых) вершин B, C, H, остальные вершины (белые) – пока не просмотрены. Теперь выбираем из очереди вершину B (напомним, что очередь – это линейный список с дисциплиной FIFO) и анализируем ее список смежных вершин. В этом список входят вершины A, C и D, но только вершина D – белая. Поэтому помещаем ее в конец очереди (после вершины H), а вершину B после этого удаляем из очереди и красим в черный цвет (включаем в список обхода – после A). Таким образом, список обхода состоит из вершин A и B, очередь – из C, H, D (именно в такой последовательности), вершины E, F, G пока остаются белыми (рис. 3). Следующая вершина, которую мы выбираем из очереди – C – не имеет смежных вершин, которые бы еще оставались белыми. Смежными с ней являются только вершины A и B, которые уже покрашены в черный цвет. Поэтому мы просто перекрашиваем вершину C в черный цвет и удаляем ее из очереди. Теперь в списке обхода вершины A, B, C, а в очереди – H и D. Список смежности вершины H позволяет нам пополнить очередь вершинами F и G, вершину H после этого можно будет перенести в список обхода (рис. 4).



На следующем шаге, анализируя список смежности вершины D, обнаружим последнюю белую вершину – E, перекрасим ее в серый цвет и поместим в очередь. Вершина D, в свою очередь, будет покрашена в черный

цвет и помещена в список обхода (A, B, C, H, D). В очереди останутся вершины F, G, E, которые последовательно будут выбраны из нее, покрашены в черный цвет и добавлены к списку обхода (поскольку белых вершин в графе после перекрашивания E уже не останется).
Сформированный в итоге список обхода графа выглядит так: A(0), B(1), C(1), H(1), D(2), F(2), G(2), E(3) (в скобках указаны длины наиболее коротких путей до той или иной вершины от A). Действительно, сначала были пройдены все вершины, отстоящие на расстояние 1, затем – 2, и лишь затем –3. Благодаря окрашиванию вершин повторный проход через ту или иную вершину исключается.

Задача. Напишите процедуру поиска в ширину.

Решение.
Будем считать, что у нас имеется объект «граф», представленный набором списков смежных вершин, причем каждая вершина характеризуется цветом, расстоянием от стартовой вершины, а также указателем на «родителя», которым будет узел-предшественник этой вершины в списке обхода. Первоначально «родителя» ни у одного узла не будет, для чего будет использовано обозначение **nil**. Затем «родителей» обретут все узлы, кроме стартового (можно использовать список с заголовочным звеном).
Также будем считать, что имеется объект «очередь», у которого есть методы «поместить в очередь вершину» и «удалить из очереди вершину» (разумеется, все это делается с соблюдением дисциплины FIFO). Кроме того, у объекта «очередь» имеется свойство «голова очереди», которое позволяет обратиться к вершине, которая в данный момент является в очереди первой, не удаляя ее. Определен и способ проверки очереди на пустоту (в зависимости от реализации это может быть и метод, и свойство логического типа).
Первоначально все расстояния до стартовой вершины будем полагать бесконечными. Конечно же, в программе нельзя написать известный значок бесконечности, но можно – по договоренности – использовать либо максимально возможное целое число, либо число, превосходящее (заранее известное) число вершин графа. Наконец, можно использовать отрицательное число или вовсе пропустить инициализацию расстояний для вершин графа (это можно делать, если граф заведомо связен).
Приведем решение на псевдокоде (параметрами процедуры являются собственно граф G и стартовая вершина s в нем):

```
procedure Breadth_First_Search (G, s)
begin
  for каждая вершина графа do
    begin
      цвет вершины := белый
      расстояние := бесконечность
      родитель вершины := nil;
    end
  цвет вершины s := серый
  расстояние := 0;
  родитель вершины := nil;
  очередь. Поместить в очередь вершину s;
  while очередь не пуста do
    begin
      u := очередь.Голова очереди
      for каждая вершина v из списка смежных вершин для u do
        if цвет вершины v = белый
          then begin
            цвет вершины v := серый
            расстояние v := расстояние u + 1
            родитель v := u
            очередь. Поместить в очередь вершину v
          end;
      очередь. Удалить из очереди (вершину u)
      цвет вершины u := черный
    end; // while
end; // Breadth_First_Search
```

Комментарий.

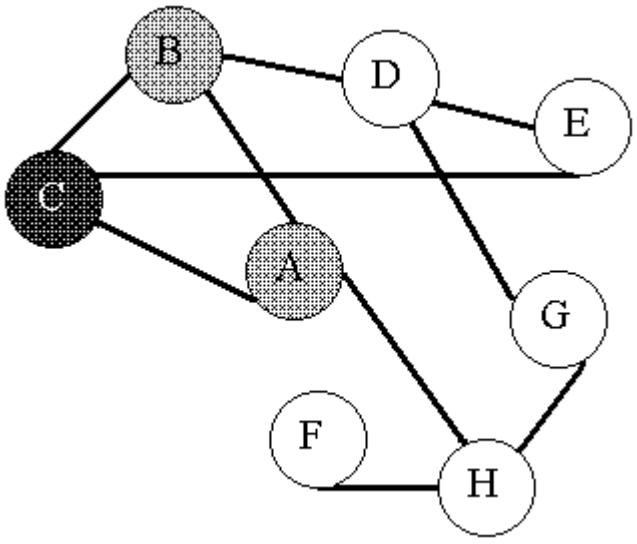
В первом цикле все вершины становятся белыми, все значения расстояний бесконечными, а родителем каждой из них объявляется **nil**. Следующие строки красят начальную вершину в серый цвет и помещают ее в очередь. Эта очередь будет в дальнейшем содержать только серые вершины.
Основной цикл программы выполняется, пока очередь непуста, т.е. существуют серые вершины (вершины, которые уже обнаружены, но списки смежности которых еще не просмотрены). Первая из вершин в очереди помещается во вспомогательную переменную u. Вложенный цикл **for** просматривает все смежные с u вершины. Когда среди них обнаружится белая, объявляем u ее родителем, красим в серый цвет и устанавливаем расстояние, большее на единицу (по сравнению с u). Затем эта вершина добавляется в хвост очереди. Когда все смежные с u вершины будут просмотрены, ее можно будет удалить из очереди и перекрасить в черный цвет.

Задачи.

1. Перепишите алгоритм поиска в ширину на Pascal, определив все необходимые структуры данных (либо объекты и/или динамические структуры данных, либо реализовать очередь на базе массива, а граф представлять матрицей смежности).
2. Еще одним базисным алгоритмом является поиск в глубину. Его стратегия такова: идти вглубь, пока это возможно (есть непройденные исходящие ребра), и возвращаться и искать другой путь, когда таких ребер нет. Если после этого остаются необнаруженные вершины, можно выбрать одну из них, и повторять процесс, пока не обнаружим все вершины графа.

Комментарий (к задаче 2).

Рассмотрим обход следующего графа.
Для наглядности будем считать, что в процессе работы алгоритма вершины графа могут быть белыми, серыми и черными. Вначале они все белые, но в ходе работы алгоритма белая вершина может стать серой, а серая – черной (но не наоборот).



Таким образом, окрашенные (серые или черные) вершины – это те, которые уже обнаружены. Поддерживается следующее свойство: если $(u,v) \in E$ и u – черная, то v – серая или черная вершина. Поэтому только серые вершины могут иметь смежные необнаруженные вершины (и только серые вершины находятся в очереди).
Алгоритм поиска в глубину может быть описан следующим образом. Каждая из вершин вначале белая. Будучи обнаруженной, она становится серой. Когда она будет полностью обработана (список смежных с ней вершин просмотрен), она становится черной.
Будем считать стартовой вершиной A. Первоначально в серый цвет будет перекрашена именно она. Первой из вершин в списке смежности вершины A будет вершина B. Ее также покрасим в серый цвет, после чего сразу приступим к анализу списка ее смежных вершин. Вершину A (которая является членом этого списка) рассматривать не следует, поэтому сразу перейдем к вершине C. Она будет сначала покрашена в серый цвет, а затем – сразу же, так как в ее списке смежных вершин белых вершин уже обнаружить не удастся – в черный цвет (рис. 5). Поместим вершину C в список обхода и вернемся к вершине B и изучим, есть ли у нее пока еще белые смежные вершины. Таковая найдется – это вершина D, у которой, в свою очередь, также найдутся смежные вершины, которые еще не были просмотрены. Сначала обратимся

к вершине E: также, как и в случае с C, ее сначала перекрасим в серый цвет, а затем, выяснив, что смежных белых вершин у нее нет, – в черный (рис. 6). Она пополнит список обхода вслед за C, а мы перейдем к рассмотрению вершины G, следующей в списке смежности для D.
Из D мы пройдем в вершину G, из вершины G – в вершину H, из вершины H – в вершину F, последовательно перекрашивая каждую из упомянутых вершин в серый цвет. Рассмотрение F покажет, что белых смежных с ней вершин уже нет, и она будет покрашена в черный цвет, а список обхода будет выглядеть как C, E, F. Возвращение в вершину H также приведет к перекрашиванию ее в черный цвет (список обхода C, E, F, H). Затем в черный цвет последовательно будут перекрашены вершины G, D, B и, наконец, A. Список обхода получится следующим: C, E, F, H, G, D, B, A. Здесь мы не указываем расстояние от стартовой вершины – большого смысла оно не имеет (разве что в качестве справочной информации). Однако часто используют так называемые «временные метки», показывающие, в какой последовательности проходились вершины. Поэтому в реализации алгоритма поиска в глубину будем ставить на вершинах метки времени. Каждая вершина имеет две метки: в d[v] записано, когда эта вершина была обнаружена (и сделана серой), а в f[v] – когда была закончена обработка списка смежных с ней вершин, и она стала черной. Для постановки этих меток используется глобальная переменная time. Массивы для меток не обязательно должны быть частью объекта «граф».
Будем считать, что у нас имеется объект «граф», представленный набором списков смежных вершин, причем каждая вершина характеризуется цветом и указателем на «родителя», которым будет узел-предшественник этой вершины в списке обхода. Первоначально «родителя» ни у одного узла не будет, для чего будет использовано обозначение **nil**. Затем «родителей» обретут все узлы, кроме стартового (можно использовать список с заголовочным звеном).
Приведем псевдокод:

```
procedure Depth_First_Search (G)
begin
  for каждая вершина графа do
    begin
      цвет вершины := белый
      родитель вершины := nil;
    end
  time := 0; // глобальная переменная, показывающая время
  for каждая вершина u графа do
    begin
      if цвет вершины = белый
        then DFS_Visit(u); // рекурсивная процедура, описана ниже
    end
end

procedure DFS_Visit (u)
begin
  цвет вершины u := серый
  d[u] := time;
  time := time + 1;
  for каждая вершина v из списка смежных вершин для u do
    if цвет вершины v = белый
      then begin
        родитель v := u
        DFS_Visit(v);
      end;
  end;
  цвет вершины u := черный
  f[u] := time;
  time := time + 1;
end;
```

Домашнее задание:

1. Разработайте алгоритмы поиска в ширину и в глубину для представления графа в виде матрицы смежности (если в аудитории решалась задача для списков смежных вершин) или, напротив, для представления в виде списков смежных вершин (если для матриц смежности задача была рассмотрена).
2. Работают ли алгоритмы поиска в глубину и в ширину только для связных графов или для несвязных тоже? Напишите программу, определяющую количество связных компонент графа.

3. Мостом графа называется такое ребро, удаление которого приводит к увеличению числа связных компонент графа. Напишите программу нахождения всех мостов графа.
4. Напишите нерекурсивную программу топологической сортировки.

(Топологическая сортировка). Представьте себе *n* чиновников, каждый из которых выдает справки определенного вида. Нужно получить все эти справки, соблюдая установленные ограничения: у каждого чиновника есть список справок, которые нужно собрать перед обращением к нему. Разработайте подходящую структуру данных и напишите программу.

5. Источником в ориентированном графе назовем вершину, от которой достижимы все другие вершины, стоком – вершину, достижимую из всех других вершин. Напишите программу поиска всех источников и стоков данного ориентированного графа.