Обход графов

Поиск в глубину (depth-first search) отвечает на вопрос: какие части графа достижимы из заданной вершины? Это похоже на задачу обхода лабиринта: имея граф как список смежности, мы в каждой вершине «видим» её соседей — как в лабиринте, где мы видим соседние помещения и можем в них перейти. Но если мы будем это делать без продуманного плана, то можем хо- дить кругами, а в какую-то часть так и не попасть.

Обойти лабиринт можно с помощью клубка ниток (закрепив конец нитки в исходной точке и нося клубок с собой, мы всегда сможем вернуться) и мела (чтобы отмечать, где мы уже были, и не ходить туда ещё раз). Примерно так же работает алгоритм поиска в глубину. Чтобы хранить пометки, для каждой вершины будем хранить бит, показывающий, были мы в этой вершине или нет. Клубок ниток можно было бы заменить стеком (идя в новый коридор, мы добавляем его в стек, а возвращаясь обратно, забираем), но в нашем алгоритме мы вместо явного стека используем рекурсию.

Explore(v)

{Вход: вершина v графа G = (V, E).}

{Выход: visited[u] = true для всех вершин u, достижимых из v.}

visited[v] = true

для каждого ребра (v, u) ∈ E:

  если visited[u]=false: Explore(u)

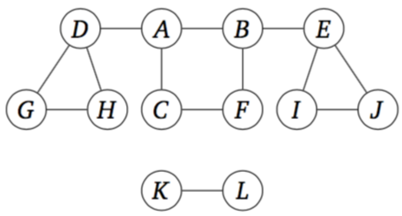


Рис. 1. Пример графа

На рис. 3.2 показан вызов Explore для графа и вершины A. Считаем, что соседи вершины перебираются в алфавитном порядке. Сплошными нарисованы рёбра, которые ведут в ранее не встречавшиеся вершины. Например, когда процедура Explore находилась в вершине B, она прошла по ребру B–E, и, поскольку в E она до этого не бывала, был произведён вызов Explore для E. Сплошные рёбра образуют дерево (связный граф без циклов) и поэтому называются древесными рёбрами (tree edges). Пунктирные же рёбра ведут в вершины, которые уже встречались. Такие рёбра называются обратными (back edges).

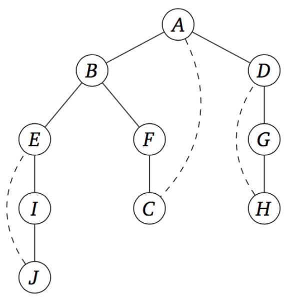


Рис. 2. Вызов Explore для вершины A

Процедура Explore обходит все вершины, достижимые из данной. Для обхода всего графа алгоритм поиска в глубину последовательно вызывает Explore для всех вершин (пропуская уже посещённые).

DFS(G)

для всех вершин $v \in V$*:*

  visited[v] = false

для всех вершин $v \in V$*:*

  если visited[v]=false:

    Explore(v)

Сразу же отметим, что процедура Explore вызывается для каждой вершины ровно один раз благодаря массиву visited (пометки в лабиринте). Она включает в себя:

1) O(1)-операции: пометка вершины;

2) перебор соседей.

На рис. 3 показан поиск в глубину на графе с двенадцатью вершинами. Опять считаем, что соседи перебираются в алфавитном порядке. Во внешнем цикле процедура Explore вызывается трижды — для вершин A, C и F. Результатом является лес (forest) из трёх деревьев с корнями в этих вершинах.

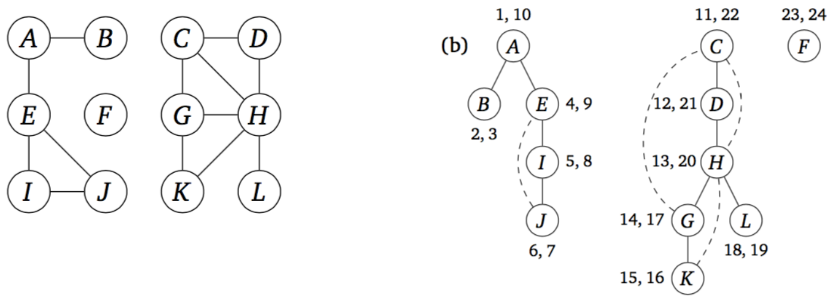


Рис. 3. Обход графа в глубину.

Поиск в глубину в ориентированных графах

Рассмотренный нами алгоритм поиска в глубину может быть использован и для ориентированных графов (в процедуре Explore надо перебирать выходящие из вершины рёбра). На рис. 4 показан пример ориентированного графа и дерева, построенного поиском в глубину (напомним, что соседи перебираются в алфавитном порядке).

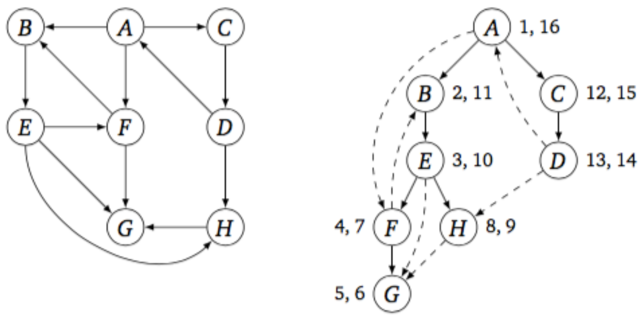


Рис. 4. Обход ориентированного графа

При поиске в глубину в неориентированных графах мы различали древесные рёбра дерева и обратные (все остальные). Для ориентированных графов типов рёбер будет уже четыре.

Древесные рёбра (tree edges) — рёбра леса, построенного поиском в глубину.

Прямые рёбра (forward edges) ведут от вершины к её потомку, не являющемуся при этом её ребёнком.

Обратные рёбра (back edges) ведут от вершины к её предку.

Перекрёстные рёбра (cross edges) ведут от вершины к другой вершине, не являющейся ни предком, ни потомком первой (такая вершина уже полностью обработана в момент обнаружения перекрёстного ребра).

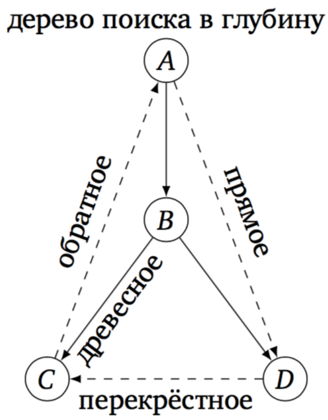


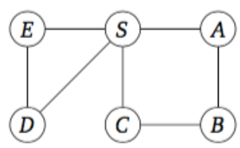
Рис. 5. Виды ребер

Поиск в глубину не только быстро находит все вершины, достижимые из начальной, но и строит дерево, содержащее пути к ним. Однако эти пути не обязательно будут кратчайшими

Расстоянием (*distance*) между двумя вершинами неориентированного графа будем называть длину кратчайшего пути между ними, измеренную в рёбрах. У этого понятия есть простой физический смысл. Представим себе граф из шариков (вершин), соединённых нитками одинаковой длины (рёбрами). Потянем граф вверх за вершину *s*; за ней потянутся все вершины, достижимые из *s*. Чтобы найти расстояния от них до *s*, мы можем теперь просто измерить, насколько они ниже *s*.

Вершины графа, подвешенного за *S*, разбиваются на слои: сама S, вершины на расстоянии 1 от *S*, вершины на расстоянии 2 и так далее. Можно находить расстояния от S до всех вершин, переходя от уровня к уровню. Когда вершины уровней 0, 1, 2, ..., *d* определены, легко найти вершины уровня $d + 1$: это просто ещё не просмотренные вершины, смежные с вершинами уровня d. Эти рассуждения наталкивают нас на алгоритм, работающий в каждый момент с двумя уровнями: некоторым уровнем *d*, который уже полностью известен, и уровнем $d + 1$, который находится просмотром соседей вершин уровня *d*.

Поиск в ширину (breadth-first search, BFS) непосредственно реализует эту простую идею. Изначально очередь Q содержит только вершину *S*, то есть вершину на расстоянии 0. Для каждого последующего расстояния $d = 1, 2, 3, \ldots$ найдётся момент времени, в который Q содержит все вершины на расстоянии d и только их. Когда все эти вершины будут обработаны (извлечены из очереди), их непросмотренные соседи окажутся добавленными в конец очереди, то есть мы перейдём к следующему значению d.



порядок посещения	содержание очереди после обработки вершины
	[*S*]
S	[*A C D E*]
A	[*C D E B*]
C	[*D E B*]
D	[*E B*]
E	[*B*]
B	[ ]

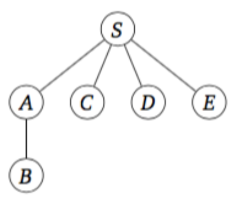


Рис. 6. Обход в ширину

Запустим этот алгоритм для вершины S. Считаем, что соседи каждой вершины обрабатываются в алфавитном порядке. На рис. 6 слева показан порядок обхода вершин, а справа изображено дерево поиска в ширину. Оно содержит только рёбра, при переходе по которым обнаруживались новые вершины. В отличие от дерева поиска в глубину все пути данного дерева с началом в S являются кратчайшими. Поэтому оно называется деревом кратчайших путей (shortest-path tree).

BFS (G, s)

{Вход: граф G(V, E), вершина s 2 V.}

{Выход: для всех вершин u, достижимых из s,

dist[u] будет равно расстоянию от s до u.}

для всех вершин u $\in$ *V:*

  dist[u] = $\propto$

  dist[s] = 0

Q = {s} {очередь из одного элемента}

пока Q не пусто:

  u = Eject (Q)

  для всех рёбер (u, v) $\in$ *E:*

    если dist[v] = $\propto$*:*

      Inject(Q, v)

    dist[v] = dist[u] + 1

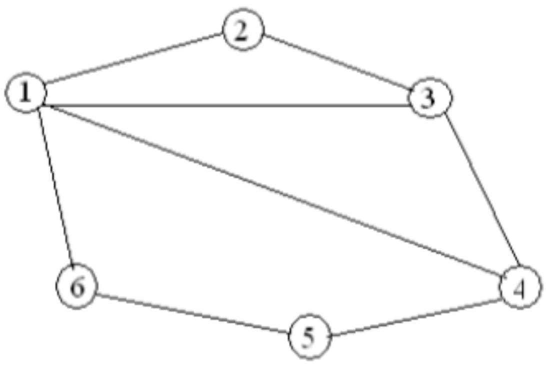
Этот алгоритм можно применять и к ориентированным графам. В них тоже можно определить расстояние от вершины S до вершины T как минимальное число рёбер, которое надо пройти (в их направлении), чтобы из S попасть в T. Расстояние при этом уже не будет симметрично, но понятие кратчайшего пути имеет смысл, и наш алгоритм по-прежнему годится.

2.1. Выполнение лабораторной работы

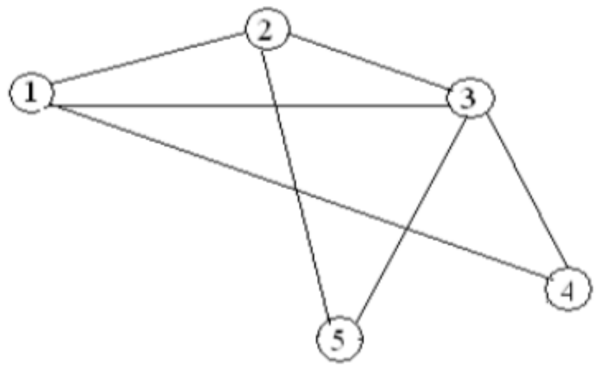
1. Согласно индивидуальному варианту реализовать обход графа в глубину
2. Согласно индивидуальному варианту реализовать обход графа в ширину
3. Реализовать оба алгоритма на любом выбранном языке программирования. Вывести на экран результаты обходов
4. Составить отчет в электронном виде.
5. Ответить на дополнительные вопросы преподавателя.

Варианты заданий

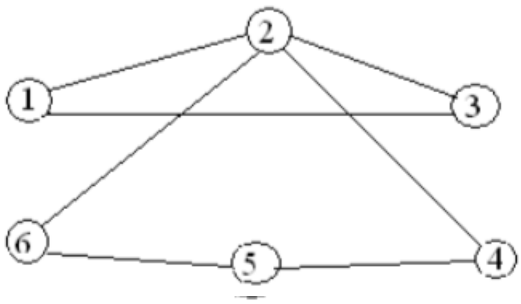
1.



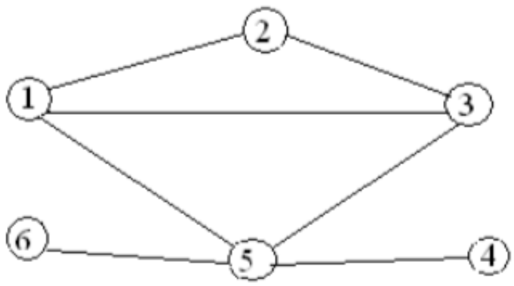
2.



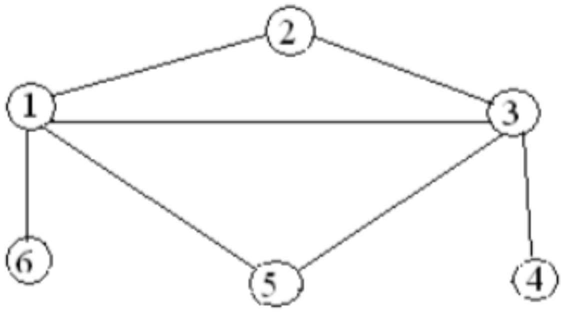
3.



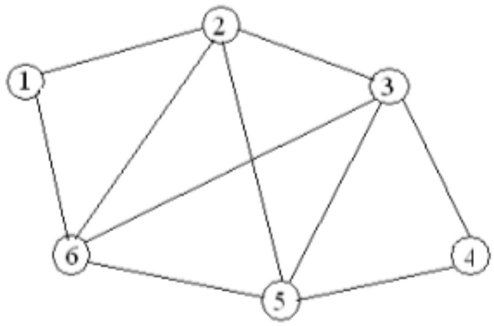
4.



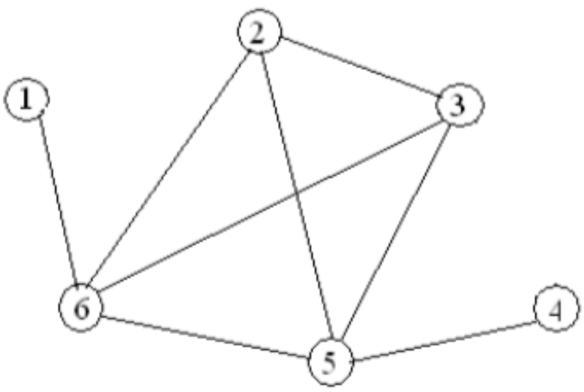
5.



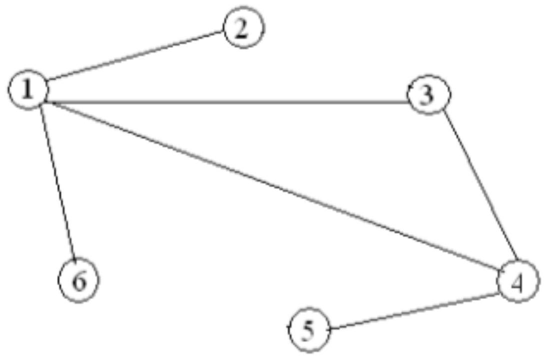
6.



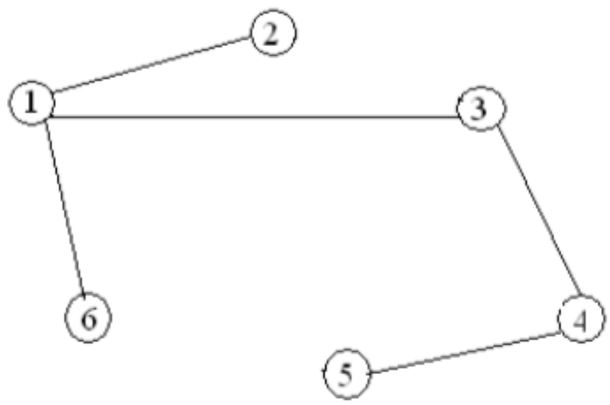
7.



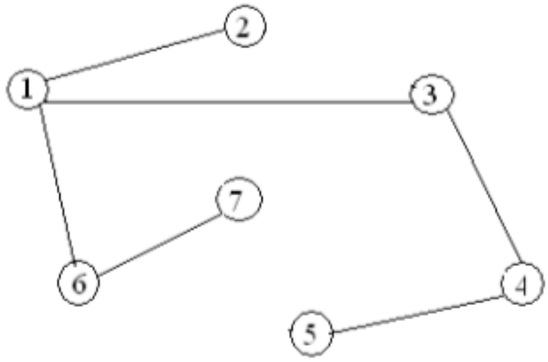
8.



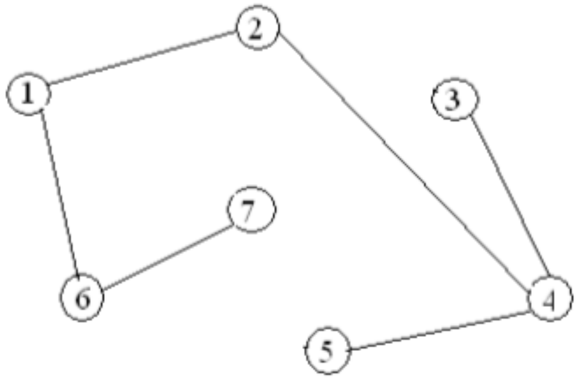
9.



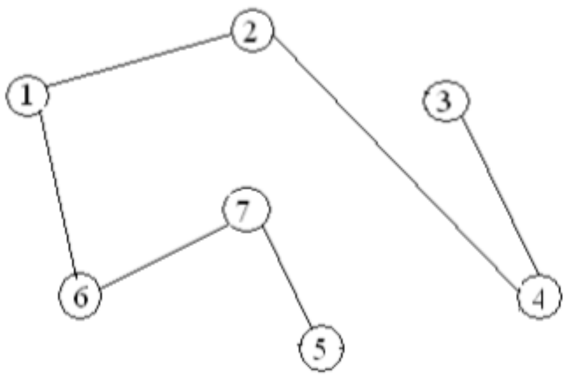
10.



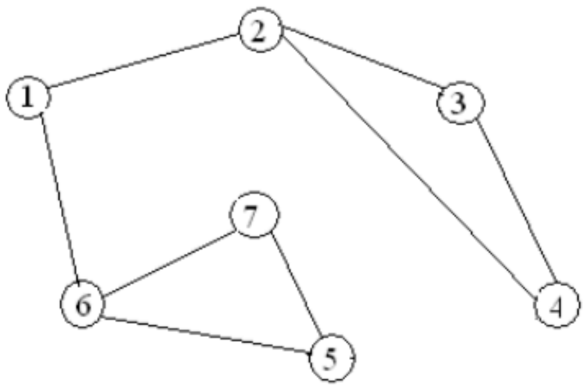
11.



12.



13.



14.