Глава II. Алгоритмы на графах

§1. Алгоритмы: основные определения. Сложность по времени и памяти.

Теория алгоритмов – раздел математики, изучающий общие свойства алгоритмов.

Перечислим наиболее важные вопросы в теории алгоритмов.

1. Что такое алгоритм?

2. Существует ли алгоритм для конкретной проблемы?
«Алгоритмическая разрешимость проблемы».

3. Существует ли «эффективный» алгоритм для конкретной проблемы?
«Классы сложности алгоритмов».

Опр. Алгоритм – точное предписание, задающее вычислительный процесс. Начинается вычисление с исходных данных, выбранных из совокупности всех возможных («допустимых») исходных данных, заканчивается получением результата.

Замечание. С точки зрения современной практики, алгоритм это программа.

Основные требования к алгоритму:

1. Алгоритм применяется к исходным данным и выдает результат, т. е. имеет вход и выход. Кроме того, в ходе работы появляются промежуточные данные.

2. Данные для своего размещения требуют памяти.

3. Объем памяти (как количество ячеек) дает одну из характеристик алгоритма, называемую сложностью по памяти.

4. Алгоритм состоит из отдельных элементарных шагов (действий), и их количество конечно.

5. Количество шагов дает вторую характеристику алгоритма, называемую сложностью по времени.

6. Последовательность шагов алгоритма должна быть детерминированной, т. е. после каждого шага либо указывается следующий шаг, либо дается команда остановки.

7. Алгоритм должен быть результативным, т. е. для любого из допустимых входных данных алгоритм должен остановиться и выдать результат.

Обе сложности зависят от входных данных, т.е. являются функциями от размера входных данных.

Опр. Сигнализирующей функцией $\varphi(n)$ называют числовую функцию, равную сложности алгоритма (по памяти или по времени), где n = размер входа.

Опр. Порядком величины φ называют величину h, такую, что существует константа C: $\varphi \le C \cdot h$.

Обозначение: $O(h)$, где φ и h зависят от n.

Пример. Величина $\varphi(n) = 21n^2 + 500n - 1$ имеет порядок $O(n^2)$.

«Эффективными» алгоритмами часто называют:

- Линейные алгоритмы, имеют порядок $O(n)$.
- Полиномиальные алгоритмы, имеют порядок $O(n^p)$.

«Плохими» алгоритмами считают:

- Экспоненциальные алгоритмы, имеют порядок $O(e^n), O(2^n), O(10^n)...$

Для анализа сложности алгоритмов применяют также оценку снизу и точный порядок величины.

Опр. Если существует константа C: $\varphi \geq C \cdot h$, то φ имеет оценку снизу $\Omega(h)$.

Опр. Если существуют константы C_1 и C_2: $C_1 \cdot h \leq \varphi \leq C_2 \cdot h$, то φ имеет точный порядок $\Theta(h)$.

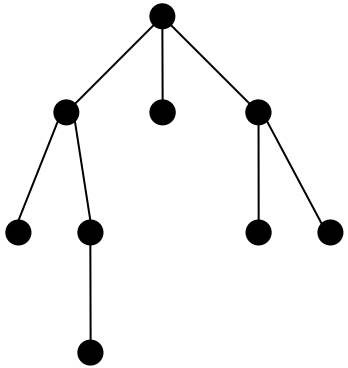
Опр. Массовая алгоритмическая проблема – проблема решения задачи «в общем виде», т.е. когда требуется найти алгоритм для бесконечной серии однотипных задач.

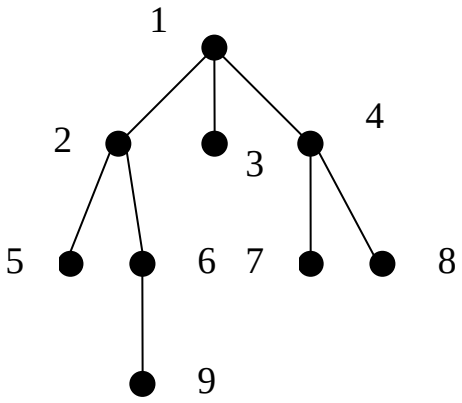
Опр. Уточнением понятия алгоритма называют сужение понятия, ограничивающее области данных и описание шагов.

§2. Поиск в ширину. Поиск в глубину.

Два способа обхода вершин в графе.

Идея поиска в ширину в дереве в стандартном изображении.




1
2
3
4
5
6
7
8
9

Опр. Очередью называется линейный массив $[\, a_1 a_2 \ldots a_k \,]$, в который можно добавлять элементы в «хвост» и извлекать элемент из начала.

Результат добавления – $[\, a_1 a_2 \ldots a_k a_{k+1} \,]$.

Результат извлечения – a_1 и $[\, a_2 \ldots a_k \,]$.

«Первым пришел – первым ушел»

Алгоритм поиска в ширину (Breadth-first search).
Вход – связный граф (V, E).

0. Создать пустую очередь L.

1. Выбрать любую вершину v_1.

Занести в L вершину v_1.
2. Извлечь из очереди L вершину u и пометить ее «развернутая».
Занести в L все неразвернутые вершины, смежные вершине u, не находящиеся в очереди.

3.... Повторить шаг 2, если L не пусто.
В противном случае – остановиться.

Выход – упорядоченная последовательность развернутых вершин, образующая дерево.

Пример. Поиск в ширину в графе Петерсона.

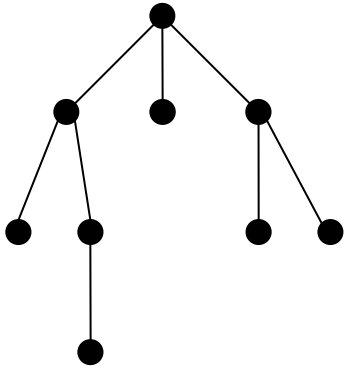
Сложность по памяти и сложность по времени имеет порядок $O(n + m)$, где $n = |V|$, $m = |E|$.

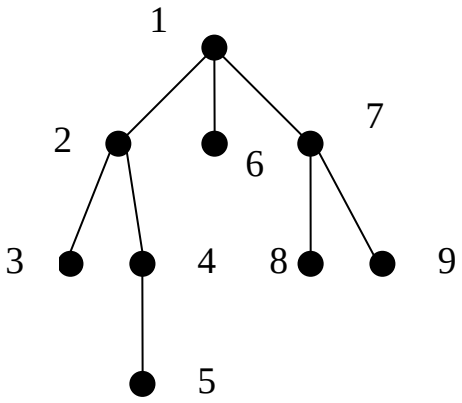
Применение обхода в ширину для поиска кратчайшей цепи между двумя вершинами v_1 и v_2 в связном графе.

Начинаем поиск в ширину с вершины v_1.

Заканчиваем алгоритм, когда v_2 становится развернутой.

Идея поиска в глубину в дереве в стандартном изображении.




1
2
6
7
3
4
8
9
5

Алгоритм поиска в глубину (depth-first search).
Вход – связный граф (V, E).

0. Покрасить все вершины в белый цвет.

1. Выбрать любую вершину v_1. Вызвать подпрограмму DFS(v_1).

Описание подпрограммы DFS(v).
1. Перекрасить вершину v в серый цвет.
2. Для каждой белой вершины w, смежной v, вызвать DFS(w).
3. Перекрасить вершину v в черный цвет.

Выход – упорядоченная последовательность черных вершин, образующая дерево.

Сложность по времени имеет $O(n + m)$,
где $n = |V|$, $m = |E|$.

Сложность по памяти при рекурсивном вызове процедуры DFS пропорциональна глубине рекурсии. В худшем случае, глубина равна n.
Данные при каждом вызове процедуры DFS требуют памяти $O(n)$, Следовательно, при использовании рекурсивного вызова сложность по памяти имеет порядок $O\left(n^2\right)$.

Если использовать в алгоритме специальную структуру, называемую «стек», то сложность по памяти имеет $O(n)$.

Опр. Стеком (магазином) называется линейный массив $\left[a_k a_{k-1} \ldots a_1 \right]$, в который можно добавлять элементы в начало и извлекать элемент из начала.

Результат добавления – $\left[a_{k+1} a_k \ldots a_1 \right]$.

Результат извлечения – a_k и $\left[a_{k-1} \ldots a_1 \right]$.

«Первым пришел – последним ушел»

Применение поиска в глубину для проверки двудольности графа.

Выберем вершину v_1, поместим ее в множество V_1. $V_2 = \varnothing$.

Выполним поиск в глубину, начиная с v_1. Поместим каждую вершину *w*, смежную *v*, в долю V_1 или V_2, не равную доле, которой принадлежит *v*.

Если вершина *w* уже посещалась ранее, то проверяем, что *w* и *v* принадлежат разным долям.

Если *w* и *v* принадлежат одной доле, то алгоритм останавливается с результатом «Граф не является двудольным».

В противном случае результатом алгоритма будет разбиение вершин графа на две доли V_1 и V_2.