Около DFS: depth-first search

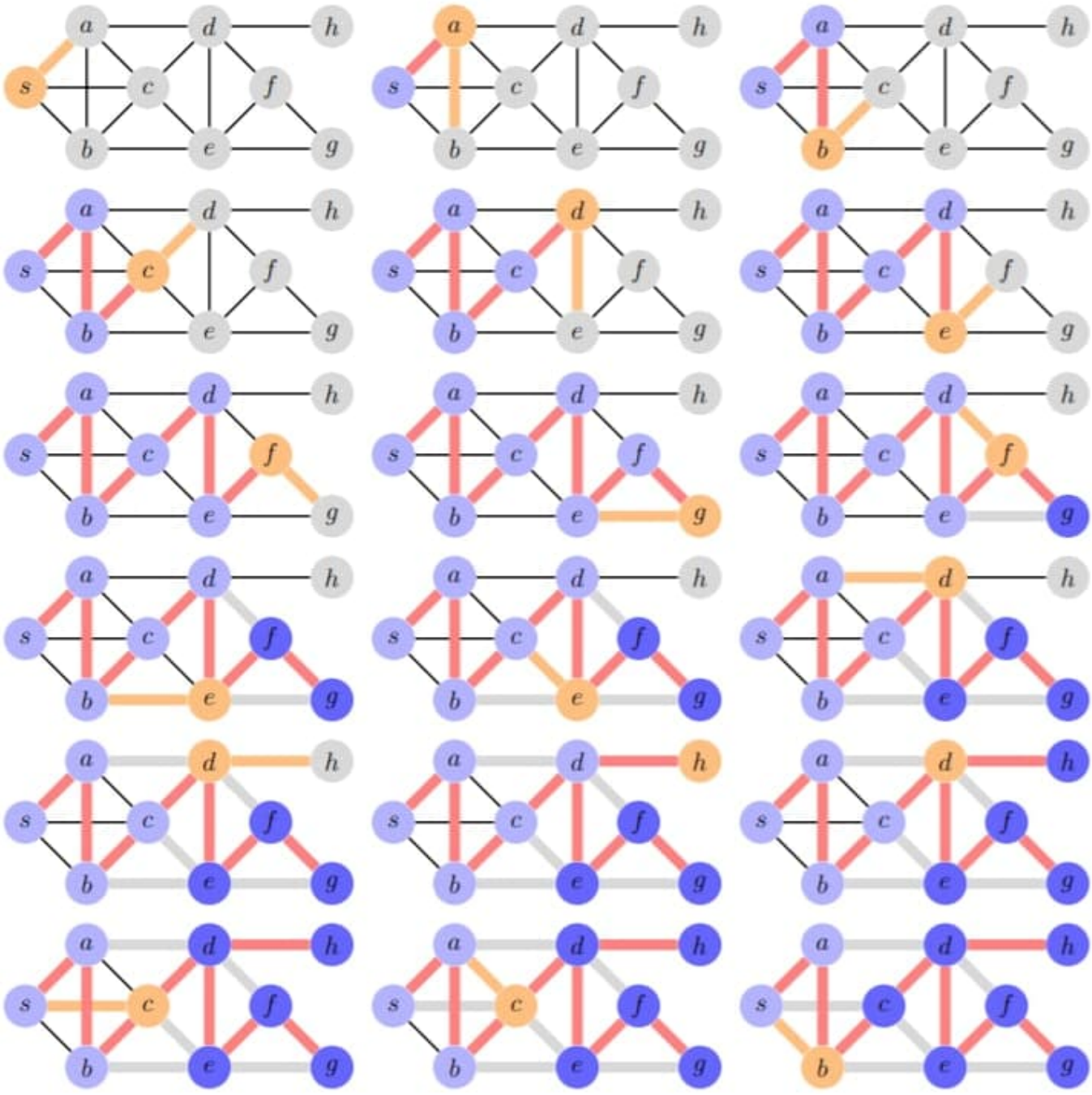
1. Описание алгоритма

Очень часто в задачах на графы (определение связности, наличия пути, цикла итд.) необходимо обойти граф. Один из способов это сделать — обойти его **в глубину** (depth first search): из уже достигнутой вершины v_1 пройти в смежную и еще не посещенную вершину v_2, из нее — в другую смежную и еще не посещенную вершину v_3 и так далее; если вдруг все смежные с v_i вершины посещены, вернуться к v_{i-1} и пройти в смежную и еще не посещенную вершину, отличную от v_i.

На вход подается граф в виде списка смежности. В реализации ниже мы строим дерево предшествования p (формально $p : V \to V$ сопоставляет вершине v вершину $p(v)$, из которой v была впервые посещена) и $nr : V \to \mathbb{N}$ функцию, сопоставляющую вершине v ее порядковый номер при посещении (если $nr(v) = k$, то до v было посещено $k - 1$ вершин). В принципе нам необязательно вычислять $p(\cdot)$ и $nr(\cdot)$ — это полезные мелочи, которые пригодятся нам для приложений DFS.

```
1:  procedure DFS(G = (V, E), s)
2:      for v ∈ V do
3:          nr(v) ← 0, p(v) ← 0
4:      end for
5:      for e ∈ E do
6:          u(e) ← false
7:      end for
8:      i ← 1, v ← s, nr(s) ← 1
9:      while v ≠ s или ∃w ∈ A_s u(sw) = false do
10:         while ∃w ∈ A_v : u(vw) = false do
11:             выбрать w ∈ A_v : u(vw) = false, u(vw) = true
12:             if nr(w) = 0 then
13:                 p(w) ← v, i ← i + 1, nr(w) = i, v ← w
14:             end if
15:         end while
16:         v ← p(v)
17:     end while
18:     return p, nr
19: end procedure
```

Пример работы DFS. На картинке ниже пример работы для данного графа со следующим порядком вершин: $s, a, b, c, d, e, f, g, h$. Рыжим цветом помечена текущая вершина и проверяемое ребро. Посещенные вершины отмечены светло-синим цветом. Ребра $(p(v), v)$ покрашены в красный, посещенные ребра — в серый. На рисунке пропущены кадры с проверкой уже посещенных ребер.



Заметьте, что он работает рекурсивно — для каждой новой вершины словно снова запускается свой dfs. Будем надеяться, что теперь вы поняли, что такое поиск в глубину (DFS). Так что перейдем к некоторым фактам про него.

2. Свойства DFS

1. Дан связный граф G. Покажите, что полученное DFS дерево содержит все вершины графа.
2. Докажите, что каждое ребро графа при DFS используется по одному разу в каждом направлении.
3. Докажите, что можно подобрать такую константу k, что DFS использует не более чем $k \cdot (E + V)$ операций для любого графа. Иными словами, сложность алгоритма не более чем $O(V + E)$.
4. Предложите алгоритм определения связности графа с помощью DFS.

3. Топологическая сортировка

На самом деле, можно запустить DFS и для ориентированных графов — правда, теперь идти по ребру в алгоритме можно будет только в разрешенном направлении.

Обратимся к более специфическим задачам, чем просто определение связности графа. Для следующих задач придумайте алгоритм, использующий поиск в глубину, решающий эти задачи.

5. Дан ориентированный граф с V вершинами и E рёбрами. Требуется перенумеровать его вершины таким образом, чтобы каждое ребро вело из вершины с меньшим номером в вершину с большим. Топологическая сортировка может быть не единственной или не существовать вовсе.
6. Есть n переменных, значения которых нам неизвестны. Известно лишь про некоторые пары переменных, что одна переменная меньше другой. Требуется проверить, не противоречивы ли эти неравенства, и если нет, выдать переменные в порядке их возрастания (если решений несколько — выдать любое).

4. Поиск точек сочленения

Определение. Вершина называется *точкой сочленения*, если при удалении ее из графа увеличивается количество компонент связности.

Определение. Будем называть ребро (ориентированное) *древесным*, если оно имеет вид $p(v)v$ в терминах DFS.

Определение Определим функцию $L : V \to \mathbb{N}$ следующим образом: пусть B_v — множество вершин, в которые можно попасть из v по пути, где все ребра, кроме, возможно, последнего — древесные, тогда $L(v) = min_{u \in B_v} nr(u)$.

Осознайте это определение!

7. Вычислите L для всех вершин из картинки на второй странице («пример работы DFS»).

8. Пусть G – связный граф, s — стартовая вершина DFS.
(а) Докажите, что если вершина $u \neq s$ является точкой сочленения, то существует древесное ребро $e = uv$, удовлетворяющее $L(v) \geqslant nr(u)$, где L — функция, определенная выше.
(б) Докажите, что если существует древесное ребро $e = uv$, где $u \neq s$, удовлетворяющее $L(v) \geqslant nr(u)$, то вершина u является точкой сочленения. **(в)** Докажите, что вершина s — точка сочленения тогда и только тогда, когда лежит хотя бы на двух древесных ребрах.

5. Поиск компонент сильной связности

Еще одно применение DFS — поиск компонент сильной связности для ориентированного графа. Для этого немного промодифицируем алгоритм — будем запоминать в Nr порядок вершин, в котором они были *полностью просмотрены*.

```
1:  procedure DFSM(G = (V, E), s)
2:      for v ∈ V do
3:          nr(v) ← 0, p(v) ← 0
4:      end for
5:      for e ∈ E do
6:          u(e) ← false
7:      end for
8:      i ← 1, v ← s, nr(s) ← 1, Nr(s) ← |V|
9:      while v ≠ s или ∃w ∈ A_s u(sw) = false do
10:         while ∃w ∈ A_v : u(vw) = false do
11:             выбрать w ∈ A_v : u(vw) = false, u(vw) = true
12:             if nr(w) = 0 then
13:                 p(w) ← v, i ← i + 1, nr(w) = i, v ← w
14:             end if
15:         end while
16:         j ← j + 1, Nr(v) ← j, v ← p(v)
17:     end while
18:     return p, nr, Nr
19: end procedure
```

9. **Теорема.** Докажите, что алгоритм ниже действительно находит компоненты сильной связности.

```
1:  procedure KOSARAJU-SHARIR(G = (V, E), s)
2:      p, nr, Nr = DFSM(G = (V, E), s)
3:      H = (V*, E*) — граф, полученный из G обращением ориентации
4:      while V* ≠ ∅ do
5:          взять u ∈ V* такую, что Nr(u) максимально
6:          k ← k + 1
7:          nr~, p~ = DFS(H, u)
8:          C_k ← {v ∈ V* | nr~(v) ≠ 0}
9:          удалить из H вершины C_k и смежные с C_k ребра
10:     end while
11:     return C_1, ...
12: end procedure
```

10. Дано n булевых переменных $x_1, x_2, \ldots x_n$. Дана конъюнкция (логическое «И») выражений в скобках, каждое из которых представляет собой дизъюнкцию ((логическое «ИЛИ») ровно двух литералов (переменная или (логическое «НЕ» + переменная). Например, это выражение могло выглядеть так:

$$(x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_3 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2) \wedge \ldots$$

Известно, что скобок было K штук.

Требуется определить, есть ли такие значения переменных, что полученное выражение принимает значение 1 (правда). Напомним, что булевы переменные могут принимать значения 1 (правда) или 0 (ложь).

Докажите, что существует алгоритм, который решает эту задачу «за полиномиальное от K время» — докажите, что существует такой фиксированный многочлен $P(x)$, что количество операций при любых входных данных не больше, чем $P(K)$.

Что это было?
На самом деле эта задача про логическое выражение даже имеет свое название — $2SAT$. А если задача решается за полиномиальное время, то говорят, что она *принадлежит классу P*.

Итого. DFS — один из классических алгоритмов на графах и используется, конечно, еще во многих задачах, даже на первый взгляд не связанных с графами. Похожие свойства имеет еще один алгоритм на графах — BFS (поиск в ширину), похожий на подвешивание за вершину.