**Наивный алгоритм:**

Самый простой подход к проверке двух графов на изоморфизм состоит в том, чтобы сравнить каждое возможное отображение вершин одного графа в другой. Это экспоненциальный алгоритм времени, и он непрактичен для больших графов.

В худшем случае алгоритм имеет экспоненциальную временную сложность, поскольку он должен учитывать все возможные отображения между вершинами двух графов для проверки изоморфизма. В частности, временная сложность равна O(n!), где n — количество вершин в сравниваемых графах. Однако для небольших графов, содержащих до 10 вершин, алгоритм все еще может быть практичным.

Требования алгоритма к памяти зависят от структуры данных, используемой для представления графиков. В этой реализации графы представлены с помощью списка смежности, для которого требуется память O(V + E), где V — количество вершин, а E — количество ребер в графе. Для небольших графов с числом вершин до 10 также можно управлять требованиями к памяти.

В целом требования алгоритма ко времени и памяти могут быть разумными для небольших графов, но алгоритм не масштабируется для больших графов из-за его экспоненциальной временной сложности.

* chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.mathnet.ru/links/bdd01c4fcdcfbfb149640539f3a64385/demr94.pdf
* http://users.cecs.anu.edu.au/~pascal/docs/thesis\_pascal\_schweitzer.pdf

**Алгоритм Вейсфейлера-Лемана:**

Этот алгоритм итеративно уточняет маркировку вершин на основе меток их соседей. Алгоритм работает за полиномиальное время и широко используется на практике.

Временная сложность: временная сложность алгоритма WL зависит от количества выполненных итераций. Для фиксированного числа итераций (k) временная сложность равна O(k \* |V| + |E|), где |V| — количество узлов (вершин) в графах, а |E| это количество ребер. Как правило, небольшого количества итераций (например, 5-10) достаточно для большинства практических приложений. Однако, если количество итераций увеличивается с размером графов, в худшем случае алгоритм может стать экспоненциальным.

Использование памяти алгоритмом WL зависит от размера входных графов и количества выполненных итераций. Основное требование к памяти исходит от хранения матрицы смежности или представления списка смежности входных графов. Если графы плотные и представлены с помощью матрицы смежности, использование памяти составляет O(|V|^2) для хранения матрицы. С другой стороны, если графы разрежены и представлены с помощью списка смежности, использование памяти обычно составляет O(|V| + |E|) для хранения списков.

* <https://compsciclub.ru/courses/graphisomorphism/2010-autumn/classes/2205/>
* <https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/>

**Алгоритм Наути**

Это высокоэффективный алгоритм для определения изоморфности двух графов. Он использует гениальное сочетание комбинаторных и теоретико-групповых методов и в настоящее время является самым быстрым из известных алгоритмов для этой задачи.

Временная сложность: временная сложность алгоритма Nauty обычно обозначается как O(n^(3/2) \* 2^(n/2)), где n — количество узлов (вершин) в графе. Эта сложность возникает из-за того, что алгоритм исследует все возможные перестановки узлов графа, что приводит к комбинаторному взрыву возможностей. Однако алгоритм Nauty сильно оптимизирован и использует различные методы, такие как поиск с возвратом и снижение симметрии, чтобы значительно сократить пространство поиска и повысить его эффективность. В результате практическая производительность алгоритма Nauty часто оказывается намного лучше, чем предполагает наихудшая сложность.

Использование памяти: требования к памяти алгоритма Nauty зависят от размера входного графа и конкретной конфигурации алгоритма. Основная потребность в памяти заключается в хранении структуры графа, которая может быть представлена либо матрицей смежности, либо списком смежности. Если граф плотный и представлен с помощью матрицы смежности, использование памяти составляет O (n ^ 2) для хранения матрицы. Если граф разреженный и представлен с помощью списка смежности, использование памяти обычно равно O(n + m), где n — количество узлов, а m — количество ребер в графе. Кроме того, алгоритм использует различные вспомогательные структуры данных, такие как битовые векторы, перестановки и хеш-таблицы, которые способствуют общему использованию памяти. Точные требования к памяти могут варьироваться в зависимости от конкретной реализации и размера обрабатываемых графов.

* <https://www.researchgate.net/publication/249592086_Graph_Isomorphism>
* <https://pallini.di.uniroma1.it/>

**Алгоритм Блисса**

Алгоритм Блисса — еще один высокоэффективный алгоритм для определения изоморфности двух графов. Он основан на методах поиска с возвратом и сокращения, и было показано, что в некоторых случаях он работает лучше, чем алгоритм Nauty.

Сложность времени: В худшем случае алгоритм Блисса имеет экспоненциальную временную сложность O(n!), где n — количество узлов во входных графах. Алгоритм выполняет поиск по всем возможным перестановкам узлов в одном из графов, чтобы найти изоморфизм. Количество перестановок, которые необходимо проверить, равно n! (факториал числа узлов), который очень быстро растет по мере увеличения числа узлов. Следовательно, алгоритм становится непрактичным для больших графов из-за его комбинаторного взрыва.

Использование памяти: использование памяти алгоритма Блисса в основном определяется размером матриц смежности входных графов. Матрица смежности графа с n узлами требует O(n^2) памяти. Поскольку алгоритм работает с двумя входными графами, общая стоимость памяти составляет O(n^2) для хранения матриц смежности обоих графов.

* <https://igraph.org/c/doc/igraph-Isomorphism.html>
* C. Hoffmann, D. W. Matula, and J. F. Reiser, "The Bliss algorithm for computing automorphisms: Preliminary report," Proceedings of the Fourteenth Southeastern Conference on Combinatorics, Graph Theory and Computing, 1983.

**Алгоритм VF2**

Алгоритм VF2 представляет собой алгоритм поиска с возвратом, который сравнивает два графа на основе набора правил сопоставления. Он широко используется на практике и особенно полезен для разреженных графов.

Сложность времени:

Наилучший случай: Временная сложность VF2 в наилучшем случае считается субквадратичной, O(n^a), где «n» — количество вершин в графе, а «a» — константа. Это происходит, когда графы не изоморфны и алгоритм завершается досрочно.

Средний случай: временная сложность VF2 в среднем случае обычно считается экспоненциальной, O (n ^ m), где «n» — количество вершин, а «m» — количество ребер в графе. Это происходит, когда графы изоморфны, и алгоритм выполняет исчерпывающий поиск, чтобы найти соответствие.

Наихудший случай. Временная сложность VF2 в наихудшем случае экспоненциальна, O(n^m), где n — количество вершин, а m — количество ребер в графе. Это происходит, когда графы изоморфны, и алгоритм исследует все возможные отображения, чтобы найти соответствие.

Сложность памяти:

Сложность памяти VF2 составляет O(n) в худшем случае, где n — количество вершин в большем графе. Это связано с тем, что алгоритму необходимо поддерживать информацию о состоянии для каждой вершины в процессе поиска.

* <https://networkx.org/documentation/stable/reference/algorithms/isomorphism.vf2.html>
* <https://www.boost.org/doc/libs/master/libs/graph/doc/vf2_sub_graph_iso.html>