Графы можно рассматривать на **четырех уровнях** (для каждого из них есть характерные задачи):

1. вершин/узлов/нод – предсказание характеристик вершин (например, классификация пользователей, категоризация товаров)
2. ребер/связей – восстановление пропущенных связей (дополнение графов знаний, рекомендация друзей)
3. подграфов/сообществ – кластеризациия (поиск сообществ в социальных сетях)
4. графа целиком – классификация графов (предсказание свойств молекул).

Еще задачи: генерация графов, моделирование их развития/эволюции (например, симуляция движения частиц).

Вершины обозначают N (или V, $\mathcal{V}$), ребра E ($\mathcal{E}$), весь граф $G(N, E)$. Что именно считать вершинами и, особеннно, ребрами в каждом конкретном случае бывает неочевидно, и от этого решения зависит успех моделироваания.

Графы бывают **ненаправленными** (например, сети друзей или коллег) и **направленными** (телефонные звонки, подписки в Твиттере). Отдельно выделяют **двудольные** (биграфы, bipartite, могут быть и multipartite) – в них вершины делятся на два непересекающихся множества и каждая может быть связана ребрами только с вершинами из другого множества (пример – пользователи-и-фильмы, здесь связь означает «смотрел», или ученые-и-статьи, здесь связь – «написал, был одним из авторов»). Из них можно получить «проекции» – графы, содержащие вершины только одного множества, где ребрами связаны те, кто был соединен с одними и теми же вершинами другого множества (например, пользователи, смотревшие один и тот же фильм, или соавторы статьи). Также бывают графы **с петлями** (self-edges, или self-loops) и такие, в которых каждая пара вершин может соединяться несколькими ребрами (**multigraph**). Графы бывают **связными** (из любой вершины можно по ребрам дойти до любой другой) и **несвязными** (тогда можно рассматривать каждую компоненту связности отдельно). Если у ребер есть веса (например, частота контактов в социальной сети), граф называется **взвешенным**.

Еще разновидность – **графы знаний**, у них ребра имеют дополнительный параметр – значение связи (например, «является автором», «родился в» и т.д.). У ребер могут быть следующие свойства:

1. симметричность – $(u, \tau, v) \in E \Leftrightarrow (v, \tau, u) \in E$ (пример: если A – «друг» B, значит, и B – «друг» A)
2. антисимметричность – $(u, \tau, v) \in E \Rightarrow (v, \tau, u) \notin E$ (если Пушкин – «автор» Онегина, значит, Онегин – не «автор» Пушкина)
3. инверсия – $(u, \tau_1, v) \in E \Rightarrow (v, \tau_2, u) \in E$ (если команда A «обыграла» команду B, значит, команда B «проигала» (связь с обратным смыслом) команде A)
4. композиция – $(u, \tau_1, y) \in E \wedge (y, \tau_2, v) \in E \Rightarrow (u, \tau_3, v) \in E$ (если A «старше» B, а B «старше» C, то A «старше» C)
5. 1-to-N (преподаватель «вел предмет» у нескольких студентов, и эти студенты в векторном пространстве не сливаются в одного)

Граф можно изобразить:

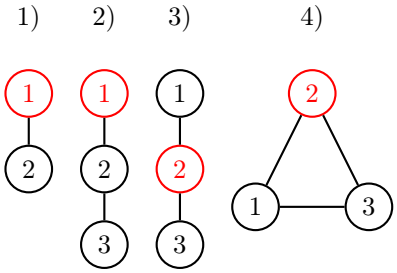
1. **графически**;
2. с помощью **матрицы смежности** (adjacency matrix) размера N*N. Каждый элемент такой матрицы – либо 0 (нет ребра), либо 1 (есть ребро) либо вес ребра (если есть). Для ненаправленного графа матрица симметричная, для направленного – нет. В реальной жизни матрицы очень разреженные, что создает проблемы с хранением данных для больших графов;
3. как **список ребер** – список пар «вершина+связанная с ней вершина». Это более эффективный, по сравнению с матрицами, способ хранения, но с такими списками неудобно работать, например,

считать степени вершин;

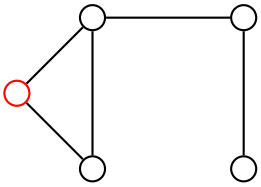
4. как **список смежности** (adjacency list), он сопоставляет каждой вершине сразу весь список связанных с ней вершин. Это тоже эффективно с точки зрения памяти плюс удобнее обрабатывать.

Положение вершины в графе определяет ее характеристики:

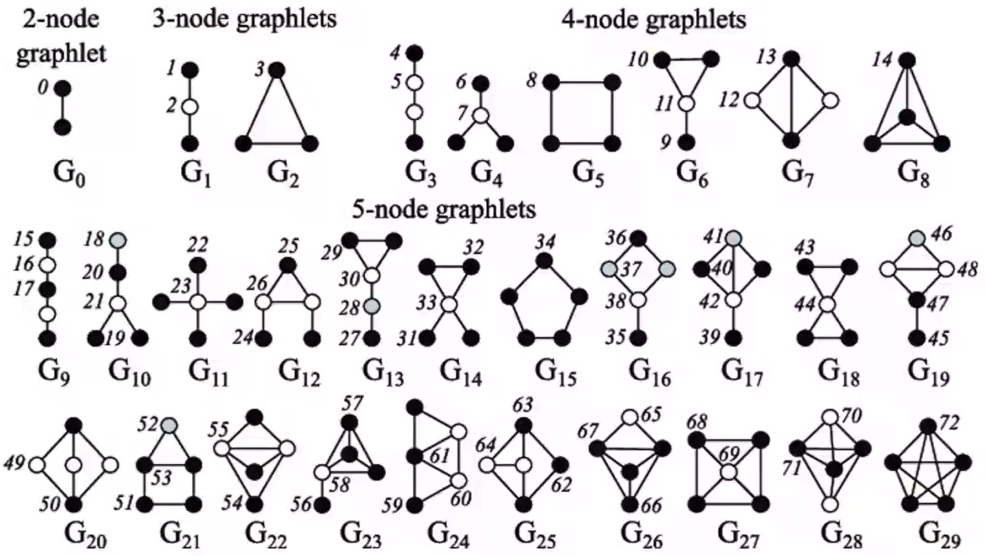
1. **степень** (degree) – количество вершин, с которыми она связана ребрами (например, $k_i = 4$), в неправленном различают in-degree и out-degree (просто degree будет суммой этих двух); в матрице смежности можно найти как сумму по строке или столбцу. Все вершины, связанные с данной, котируются одинаково (просто считать, это хорошо, но много информации теряется, это плохо)
2. меры центральности – нужны, чтобы точнее оценить «важность» вершины. Их много разных:
   - **центральность с помощью собственных значений** (Eigenvector Centrality) – сумма таких же центральностей всех соседей. Здесь важность вершины зависит от важности тех вершин, с которыми она соединена. Задается рекурсивно: $e_u = \frac{1}{\lambda} \sum_{v \in V} A[u, v] e_v$, где λ – константа (или $\lambda c = Ac$, привет, теорема Фробениуса – Перрона)
   - **центральность с помощью соседства** (Betweenness Centrality) – сумма долей кратчайших путей (для каждой пары вершин), которые проходят через заданную вершину: $c_v = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$
   - **центральность с помощью близости** (Closeness Centrality) – единица, деленная на сумму длин кратчайших путей до всех вершин, кроме заданной: $c_v = 1 / \sum_{u \neq v} d(u, v)$
3. **коэффициент кластеризации** – степень соединенности соседей вершины, доля существующих ребер между соседями вершины от числа всех потенциально возможных: $c_u = \frac{2|e_{ij} : v_i, v_j \in N(u), e_{ij} \in E|}{k_u(k_u - 1)}$, где $N(u)$ – множество соседей вершины u, k_u – количество этих соседей
4. **графлеты** – сохраняют информацию о локальной топологии вершины. Графлет – подграф из 2, 3 и т.д. вершин, связанных уникальным образом (rooted connected non-isomorphic subgraphs). Все подграфы, которые можно получить поворотом, отражением и т.д. – это один и тот же графлет. Но если мы изменим положение в нем заданной вершины, получим другой графлет. Важно, что когда мы ищем графлеты в графе, мы не можем игнорировать существующие ребра. Посчитав графлеты, мы получим представление вершины в виде вектора (Graphlet degree vector, GDV), элементы которого – 1 (такой графлет с участием этой вершины есть) или 0 (графлета нет) либо число (если мы хотим учитывать количество графлетов каждого типа). Порядок графлетов в векторе в разных источниках может отличаться, главное – договориться внутри проекта.
   Графлеты из 2 и 3 вершин:



Вектор для красной вершины: [2, 2, 0, 1])



Для справки – графлеты из 2-5 вершин:



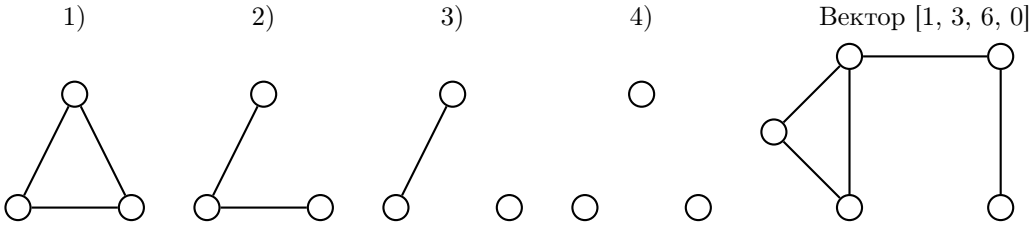
5. Плюс можно задать сколько угодно характеристик, связанных с предметной областью (цвет, тип, класс, ранг, знак и т.д.).

Ребрам часто приписывают **вес**, но они, как и вершины, могут иметь специические для предметной области характеристики.

Для двух вершин можно посчитать **длину кратчайшего пути**, метрики **локального соседства**: 1) **количество общих соседей** ($S[u,v] = |N(u) \cap N(v)|$), 2) коэффициент **Жаккара** ($S_{Jaccard}[u,v] = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$), 3) индекс **Адамика-Адара** ($S_{AA}[v_1,v_2] = \sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{log(d_u)}$). Проблема со всеми ними - если вершины не имеют общих соседей, все три метрики будут равны нулю. Для учета **глобального соседства** существует индекс **Каца** (количество всех путей между двумя вершинами: $S_{Katz}[u,v] = \sum_{i=1}^{\infty} \beta^i A^i[u,v]$, где β – параметр, штрафующий за длину пути, A - матрица смежности, воведенная в степень, так что A^n показывает количества путей длиной n между каждыми двумя вершинами). Можно посчитать матрицу индексов (для всех пар вершин сразу): $S_{Katz} = \sum_{i=0}^{\infty} (I - \beta A)^{-1} - I$

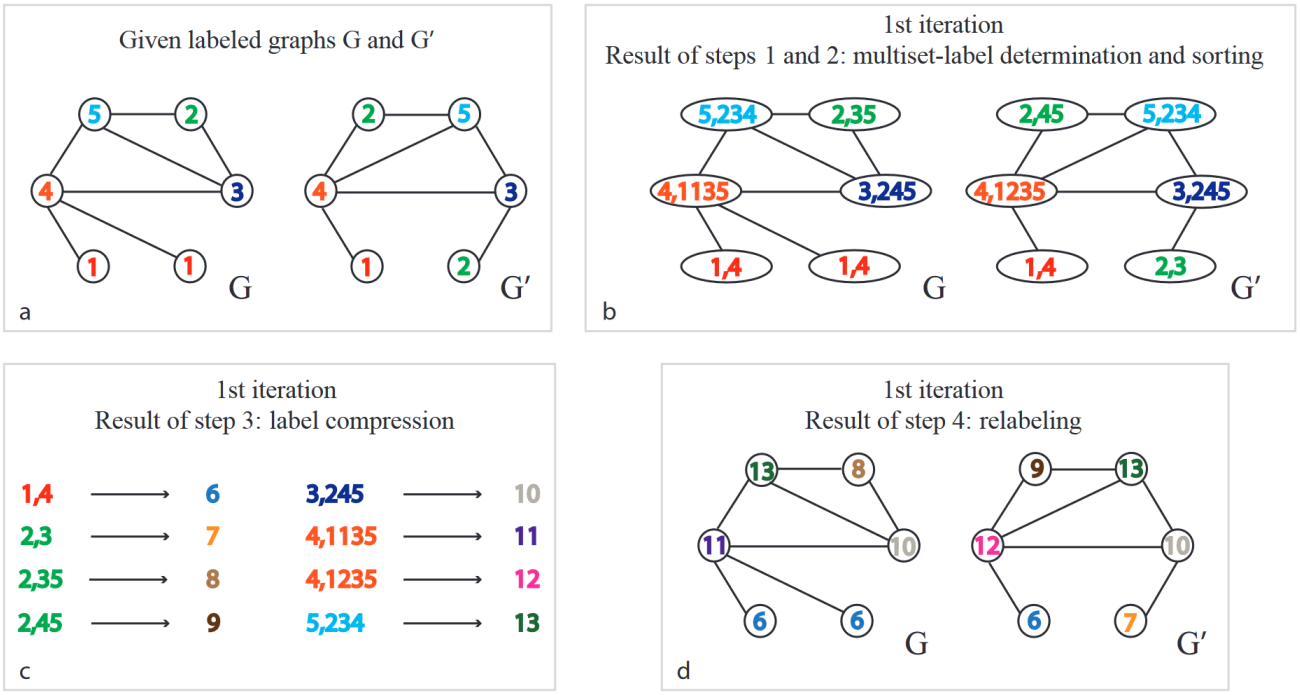
Для всего **графа** можно посчитать агрегированные значения (например, среднюю степень $\overline{k} = \langle k \rangle = \frac{1}{N} \sum_{i=1}^{N} k_i = \frac{2E}{N}$), а можно использовать ядра (в общем случае это скалярное произведение вектров двух графов, позволяющее оценить их схожесть):

- **графлетовое** ядро (как bag of words, только графлетов). Графлеты здесь упрощенные – нет корневой (заданной) вершины и мы разрешаем вершинам внутри графлетов быть несоединенными. $K(G,G') = f_G^T f_{G'}$. С нормализацией $K(G,G') = h_G^T h_{G'}$ ($h_G = \frac{f_G}{Sum(f_G)}$). Проблема с графлетовыми ядрами – дороговизна вычислений (n^k для графлетов размером k и графа из n вершин, проверка на изоморфизм - NP-hard).
  Пример ядра-3:



- **Weisfeiler-Lehman graph kernels**: 1) назначаем каждой вершине метку, например степень, 2) собираем в мультисет метки вершины и всех ее соседей, 3) хэшируем, 4) назначаем вершине получившееся значение, повторяем 2)-4), пока не надоест. При этом все промежуточные метки

мы храним – итоговый вектор будет состоять из количества меток каждого цвета (например, [3,1,5,0]). Все это намного считать куда проще, чем графлеты (линейно от числа ребер).



Эмбеддинги – вектора, неким образом собирающие информацию о положении вершины в графе. Энкодер – функция, которая проецирует вершины в эмбеддинговое пространство ($ENC : V \to \mathbb{R}^d$), декодер – наоборот, реконструирует вершину и ее связи (например, по представлению двух вершин прогнозирует, есть ли между ними связь - $DEC : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$). Цель – получить эмбеддинговое пространство, в котором похожие (что бы это ни значило) вершины будут находиться рядом, и такое представление вершин, чтобы пропустив их через энкодер и декодер мы бы получили граф, максимально похожий на исходный.

Эмбеддинги вершин можно строить по-разному:

1. на основе факторизации матриц смежности:
   - Laplacian eigenmaps (постарее). $DEC(z_u, z_v) = ||z_u - z_v||_2^2$, $L = \sum_{(u,v)\in D} DEC(z_u, z_v) * S[u, v]$, где z_u – эмбеддинг вершины, $S[u, v]$ – функция похожести вершин
   - inner-product methods (попродвинутее). $DEC(z_u, z_v) = z_u^T z_v$, $L = \sum_{(u,v)\in D} ||DEC(z_u, z_v) - S[u, v]||_2^2$, $L \approx ||ZZ^T - S||_2^2$
2. с помощью вероятностных подходов – случайные блуждания. Суть – смоделировать много-много проходов по графу, во время которых на каждой вершине мы с некоторой вероятностью выбираем один из путей, с некоторой – возращаемся назад, переносимся в исходную точку или завершаем проход (правила могут быть разными). Это позволяет ухватить особенности структуры графа. $L = \sum_{(u,v)\in D} -log(DEC(z_u, z_v))$. Варианты: Node2Vec, Struct2Vec.
   Все эти методы рассматривают вершины поодиночке, не используют признаки вершин и не умеют работать с новыми вершинами.

Эмбеддинги для графа целиком. Можно взять среднее от вершин – $Z_G = \sum_{v\in G} Z_v$. Можно добавить вершину, связанную со всеми вершинами (под)графа, и рассчитать эмбеддинг для нее. Можно использовать анонимные случайные блуждания (отличаются от обычных тем, что маршруты типа 122

и 133 считаются одним – мы зацикливаемся в соседней вершине, все равно в какой).

PageRank – алгоритм, применяющийся при ранжировании страниц в поиске. Веб можно представить в виде графа: вершины – страницы, ребра – ссылки. Граф очень большой, направленный, с петлями, каждые две вершины могут соединяться ребрами в обоих направлениях одновременно. Как определить важность страницы? Подход PageRank – страница важна, если на нее много ссылаются (исходящие ссылки нас интересуют меньше), при этом не все ссылающиеся равны (ценность ссылки зависит от важности ссылающейся страницы и количества исходящих из нее ссылок). Проблема этого определения в его рекурсивности: $r_i = \sum_{i \to j} \frac{r_i}{d_i}$. **Можно решить в матричном виде**, через собственное разложение, но для миллиардов страниц получается тяжело ($O(n^3)$). Плюс возникают проблемы, если в графе есть тупики (страницы, которые ни на кого не ссылаются) или изолированные компоненты (подграфы, не связанные с основной частью графа). Первая решается добавлением виртуальных ребер, направленных из этих тупиковых вершин во все остальные с равными весами, вторая – добавлением damping factor d: $M = (1-d)M + \frac{d}{n}J_n$, где n – количество узлов, J_n – матрицы единиц (в результате получается Google-матрица).

Лучше сработает итеративный процесс со случайными блужданиями – ищем вероятность того, что человек окажется на данной странице, случайно переходя по ссылкам. Можно начать с вектора из неких рандомных значений (например, $1/n$) и умножать его на Google-матрицу, значения постепенно сойдутся (сложность – $O(n^2)$, сводится к $O(n)$).

Предсказание связей. Задачу можно сформулировать двумя способами: 1) нам дали граф, в котором потеряны некоторые ребра, и нам нужно их восстановить, 2) дан граф со всеми ребрами, существующими на момент t, нам надо предсказать, какие ребра появятся к моменту t+1 (например, рекомендация друзей, соавторов и т.д.). Общий порядок действий во время обучения: для каждой пары вершин считаем некий скор (например, число общих друзей), ранжируем пары по нему, для top-n говорим, что ребро есть, проверяем, есть ли/появились ли они на самом деле. Скор может считаться на основе метрик для двух вершин (основанных на расстоянии или учете локального соседства) или всего графа (глобальное соседство) (см. выше метрики для двух вершин и графа).

Encoder-decoder. Энкодер принимает вершину и возвращает эмбеддинг ($ENC(v) = z_v$), декодер – принимает эмбеддинги и возвращает степень похожести ($DEC(z_v^T z_u) \approx similarity(u, v)$). Простейший способ обучать энкодер – собрать эмбеддинги в матрице размером «размерность эмбеддинга на число вершин» (embedding lookup). Проблема – слишком много параметров для обучения. Примеры: DeepWalk, node2vec