1 Введение

Доклад основан на статье *Fully dynamic algorithms for chordal graphs and split graphs*. Она вышла в 2008 году, автор — *Louis Ibarra*. Также в секции с невошедшим материалом будет представлено одно утверждение из более новой статьи: *Maintaining Chordal Graphs Dynamically: Improved Upper and Lower Bounds* [2018, Banerjee, Raman, Satti].

Извиняюсь, что мой «пересказ» настолько большой — это из-за того, что статья предполагает много исходных знаний. Непосредственное содержание статьи обсуждается в секции 3, так что внимательно читать можно только её, а остальное просто окинуть взглядом для общего понимания, пропуская доказательства.

1.1 Обозначения

Мы будем пользоваться стандартными графовыми и теоретико-множественными обозначениями. Например, $\mathsf{N}(v)$ — множество соседей вершины v, $\mathsf{N}[v] = \mathsf{N}(v) \cup \{v\}$, $\mathsf{G}[S]$ — подграф, индуцированный подмножеством вершин S.

Также с помощью $\mathfrak{n}$ и $\mathfrak{m}$ будем обозначать количество вершин и рёбер графа соответственно (из контекста всегда должно быть ясно, какого конкретно графа).

1.2 Динамические задачи на графах

Общий сеттинг такой. Дан граф G. Мы можем сделать некоторый препроцессинг. После этого мы должны обрабатывать онлайн запросы, которые концептуально можно разделить на две категории: **update** и **query** — модифицировать граф и посчитать некоторую функцию от текущего графа соответственно. Конечно, от динамического алгоритма ожидается, что он будет обрабатывать запросы быстрее своего статического аналога (это если бы мы после каждого *update* пересчитывали всю нужную информацию для *query* с нуля).

1.3 Хордальность

В нашем случае граф G невзвешенный, неориентированный, без петель и кратных рёбер, и мы хотим динамически проверять его на хордальность.

Определение 1. Граф называется *хордальным*, если любой цикл длины хотя бы 4 имеет *хорду* (ребро между несоседними вершинами на цикле).

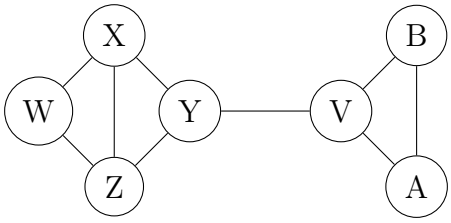

Рис. 1: Пример хордального графа

Наблюдение 1. *Из определения очевидно, что хордальность — наследственное свойство, то есть если граф хордальный, то любой его индуцированный подграф тоже хордальный.*

Таким образом, мы хотим обрабатывать в режиме онлайн следующие запросы:

1. **InsertQuery(u, v)**: является ли граф $G + uv$ хордальным?
2. **DeleteQuery(u, v)**: является ли граф $G - uv$ хордальным?
3. **Insert(u, v)**: если $G + uv$ хордальный, то добавить ребро в граф, иначе проигнорировать этот запрос.
4. **Delete(u, v)**: если $G - uv$ хордальный, то удалить ребро из графа, иначе проигнорировать этот запрос.

Самый быстрый статический алгоритм для проверки графа на хордальность работает за время $O(V + E)$, то есть нам нужно улучшить эту асимптотику.

Базовая версия алгоритма из статьи работает за $O(V)$ для запросов всех видов. Далее автор предлагает оптимизации для *InsertQuery* ($O(\log^2 V)$) и *DeleteQuery* ($O(\sqrt{E})$, то есть эта оптимизация что-то улучшает только для разреженных графов).

В следующей секции мы сформулируем несколько структурных критериев хордальности. Это довольно старые результаты, и в статье они не доказываются, но я включил их в доклад для полноты и замкнутости.

2 Как устроены хордальные графы?

Мы докажем, что хордальность эквивалентна существованию двух объектов: *Perfect Elimination Order* и *Clique Tree*. По факту, в явном виде нам понадобится только *Clique Tree*, а PEO будет нужен лишь внутри доказательств различных утверждений.

2.1 Perfect Elimination Order

Определение 2. *Perfect Elimination Order* (или PEO) для графа G — это такая перестановка его вершин $v_1, v_2, \ldots, v_n$, что $\forall i : (N(v_i) \cap \{v_{i+1}, \ldots, v_n\})$ — клика. Для удобства будем называть вершины из множества $N(v_i) \cap \{v_{i+1}, \ldots, v_n\}$ *правыми соседями* v_i относительно $PEO(G)$.

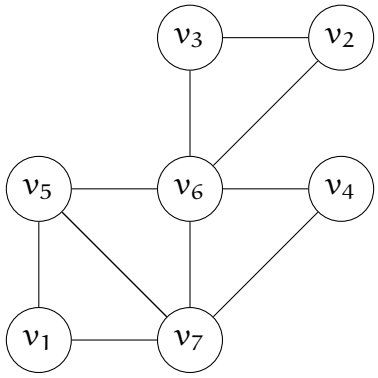

Рис. 2: Пример корректного PEO

Наша цель — показать, что хордальность эквивалентна существованию PEO. Для этого докажем несколько вспомогательных утверждений.

Определение 3. Пусть $S \subseteq V(G)$, $a, b \notin S$. Множество S называется ab-*сепаратором*, если вершины a и b в одной компоненте связности в G, но в разных в $G - S$.

Определение 4. Множество $S \subseteq V(G)$ называется *вершинным сепаратором*, если оно является ab-сепаратором для некоторых $a, b \in V(G)$.

Лемма 1. *Граф G является хордальным, если и только если любой его минимальный по включению вершинный сепаратор является кликой.*

Доказательство. Пусть граф G не является хордальным. Тогда найдётся цикл без хорд $C = c_1, c_2, \ldots, c_k$, где $k \geqslant 4$. Посмотрим на (c_1, c_3)-сепараторы. Заметим, что любой такой сепаратор S обязан содержать вершину c_2, а также c_i для некоторого $i \geqslant 4$. Так как в C нет хорд, то в частности нет и ребра (c_2, c_i), а значит S не является кликой.

В другую сторону: пусть G хордальный. Рассмотрим любой его минимальный по включению вершинный сепаратор S. Пусть он разделяет вершины a и b, обозначим их компоненты в $G - S$ за A и B соответственно. Мы хотим показать, что S клика, для этого рассмотрим произвольные $x, y \in S$ и покажем, что $xy \in E(G)$.

Заметим, что так как S — минимальный по включению сепаратор, то из каждой вершины S есть рёбра в компоненты A и B. Тогда существуют путь x в y, проходящий

только через вершины из множества A. Среди всех таких путей выберем кратчайший, обозначим его за P_A. Аналогично определим путь P_B. Заметим, что $P_A \cup P_B$ — цикл длины хотя бы 4. Так как G хордальный, то на этом цикле есть хорда. Внутри P_A и P_B её быть не может (иначе это не кратчайшие пути), кроме того между A и B нет рёбер, так что единственный вариант — это ребро xy, чего мы и хотели. □

Определение 5. Вершина $v \in V(G)$ называется *симплициальной*, если $N(v)$ — клика.

Лемма 2. *В хордальном графе есть симплициальная вершина. Если граф к тому же не является полным, то найдутся две несмежных симплициальных вершины.*

Доказательство. Индукция по числу вершин. База для $n = 1$ очевидна. Теперь рассмотрим граф G (считаем, что для всех $n < |V(G)|$ утверждение доказано). Если G — полный, то любая вершина симплициальна. Иначе найдутся такие $a, b \in V(G)$, что $ab \notin E(G)$. Пусть S — минимальный по включению ab-сепаратор. Вспомним, что он является кликой. За A и B обозначим компоненты связности графа $G - S$, в которых находятся a и b соответственно.

Докажем, что в графе $G[A \cup S]$ найдётся симплициальная вершина $v \in A$. Если $G[A \cup S]$ полный, то это очевидно, иначе по предположению индукции есть две симплициальные вершины $u, v \in (A \cup S) : uv \notin E(G[A \cup S])$. Так как S — клика, то одна из вершин обязана лежать в A.

Если $v \in A$ симплициальна в $G[A \cup S]$, то она симплициальна и в G, так как $N_G(v) \subseteq A \cup S$. Аналогичным образом можно показать, что найдётся симплициальная вершина $w \in B$. Таким образом, мы нашли пару несмежных (так как между A и B нет рёбер) симплициальных вершин в G. □

Теорема 1. *Граф G является хордальным* $\iff \exists PEO(G)$.

Доказательство. Если граф хордальный, то найдётся симплициальная вершина $v \in V(G)$. Скажем, что $v_1 = v$. Оставшийся порядок построим индуктивно (так можно, потому что $G - v$ тоже хордальный). Такой PEO нам подойдёт просто из определения симплициальности.

Пусть нашлось $PEO(G) = v_1, \ldots, v_n$. Докажем от противного что G хордальный. Если G не хордальный, то есть цикл $C = c_1, c_2, \ldots, c_k$ без хорд, где $k \geqslant 4$. Посмотрим на вершину этого цикла, которая идёт раньше остальных в $PEO(G)$, пусть это вершина c_i. Тогда, так как нет хорды (c_{i-1}, c_{i+1}), то правые соседи c_i в PEO не образуют клику, противоречие. □

2.2 Clique Tree

Изучим второй объект, который нам понадобится. *Clique Tree* — это дерево, вершинами которого являются максимальные по включению клики графа G (обозначим множество

таких клик за $\mathcal{K}_G$). Кроме того, это дерево должно удовлетворять следующему условию

$$\forall K_1, K_2 \in \mathcal{K}_G, K \in \mathtt{Path}(K_1, K_2) : K_1 \cap K_2 \subseteq K$$

Здесь $\mathtt{Path}(K_1, K_2)$ — это множество клик, лежащих на пути между K_1 и K_2 в дереве. Это условие также называется *intersection property*.

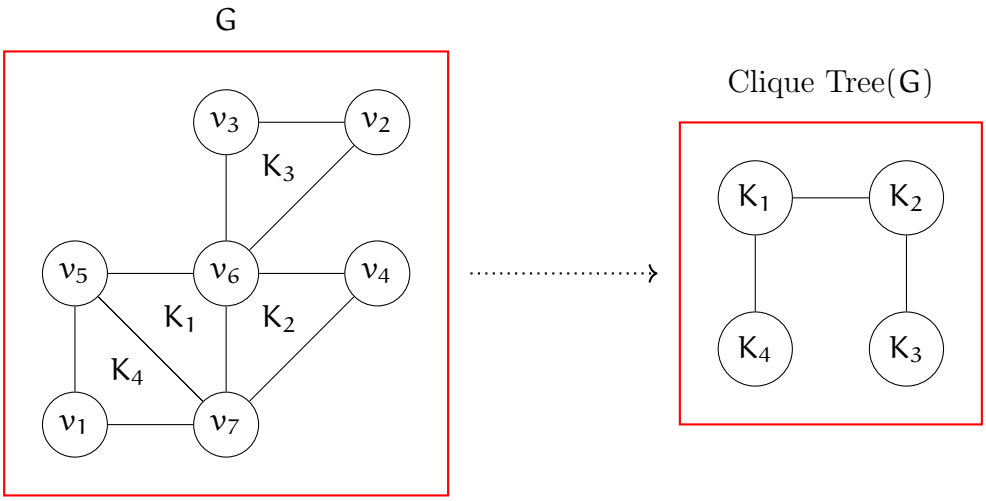

Рис. 3: Пример корректного Clique Tree

Прежде всего, оценим размер Clique Tree.

Лемма 3. *Пусть* G *— хордальный граф. Тогда* $|\mathcal{K}_G| \leqslant n$.

Доказательство. Индукция по количеству вершин в G. Если он хордальный, то есть симплициальная вершина $v \in V(G)$. По предположению индукции выполнено $|\mathcal{K}_{G-v}| \leqslant n-1$. Заметим, что v содержится ровно в одной клике из $\mathcal{K}_G$ (так как $N[v]$ — клика). Тогда $|\mathcal{K}_G| \leqslant 1 + |\mathcal{K}_{G-v}| \leqslant n$. □

Теперь докажем, что хордальность эквивалентна наличию Clique Tree.

Теорема 2. *Граф* G *хордальный* $\iff \exists$*Clique Tree*(G)

Доказательство. Докажем индукцией по количеству вершин, что у любого хордального графа есть Clique Tree. База для $n = 1$ очевидна. Теперь рассмотрим хордальный граф G. В нём есть симплициальная вершина $v \in V(G)$. Заметим, что она попадёт ровно в одну максимальную по включению клику (это клика $K := N[v]$). По предположению индукции для графа $G - v$ есть Clique Tree T'. Рассмотрим два случая:

1. $K' := (K - v) \in \mathcal{K}_{G-v}$. Получим T из T', заменив K' на K. Так как вершины v нет в других кликах, *intersection property* сохранилось.

2. $K' \notin \mathcal{K}_{G-v}$. Тогда $\exists P \in \mathcal{K}_{G-v} : K' \subseteq P$. Получим T из T', подвесив K к вершине P. Тогда *intersection property* сохранилось, так как $\forall R \in K_G : K \cap R \subseteq P \cap R$, а для пары (P, R) свойство выполнено, потому что она есть в T'.

В обратную сторону тоже будем доказывать по индукции. Пусть для графа G существует Clique Tree (обозначим его за T). Хотим показать, что G хордальный. Рассмотрим лист L дерева T и его предка P. Так как клика L является максимальной по включению в графе G, то найдётся вершина $v \in L \setminus P$. Заметим, что L — это единственная клика в дереве, которая содержит v (иначе сломается *intersection property*). Тогда v — симплициальная в G (каждое ребро, инцидентное v, должно содержаться в какой-то клике, а здесь они все в L).

Построим Clique Tree (обозначим его за T') для графа $G - v$. Если $L' := (L - v) \in \mathcal{K}_{G-v}$, то T' получается из T заменой L на L', иначе T' получается из T удалением L.

Тогда по предположению индукции $G - v$ хордальный, значит для него есть PEO. Получим из него PEO для графа G, вставив вершину v в начало (она симплициальная, поэтому корректность сохранилась). Тогда граф G тоже хордальный. □

Определение 6. Пусть T — некоторое дерево, вершинами которого являются элементы $\mathcal{K}_G$. Для вершины $v \in V(G)$ скажем, что $\mathcal{K}_G(v) := \{K \in \mathcal{K}_G \mid v \in K\}$. Тогда будем говорить, что T обладает *induced subtree property*, если $\forall v \in V(G) : \mathcal{K}_G(v)$ образует связный подграф в T.

Утверждение 1. *Наличие induced subtree property эквивалентно наличию intersection property.*

Доказательство. Пусть для некоторого $v \in V(G)$ подграф $\mathcal{K}_G(v)$ не связен в T. Тогда $\exists K_1, K_2 \in \mathcal{K}_G(v), K \in \mathtt{Path}(K_1, K_2) : v \notin K$. Тогда $K_1 \cap K_2 \not\subseteq K$, а значит нарушилось *intersection property*.

Пусть $\exists K_1, K_2 \in \mathcal{K}_G, K \in \mathtt{Path}(K_1, K_2) : K_1 \cap K_2 \not\subseteq K$. Тогда $\exists v \in (K_1 \cap K_2) \setminus K$. Заметим, что подграф $\mathcal{K}_G(v)$ не является связным, то есть нарушилось *induced subtree property*. □

Определение 7. Для графа G определим *weighted clique intersection graph* W_G, как взвешенный граф на множестве вершин $\mathcal{K}_G$, где $\forall K_1, K_2 \in \mathcal{K}_G : w(K_1, K_2) = |K_1 \cap K_2|$.

Теорема 3. *Пусть G хордальный граф, T — некоторое дерево на множестве вершин $\mathcal{K}_G$. Тогда T удовлетворяет intersection property, если и только если T является максимальным остовным деревом графа W_G.*

Доказательство. Не обсуждалось в докладе. □

3 Динамическая поддержка Clique Tree

Мы наконец-то переходим к непосредственному содержанию статьи :) Мы будем поддерживать Clique Tree в явном виде, перестраивая его при добавлении/удалении рёбер.

Также для каждой клики из дерева мы будем поддерживать хэш-множество из вершин, принадлежащих ей. Кроме того, для каждой пары соседних клик в дереве будем хранить размер их пересечения.

3.1 Delete Query

Теорема 4. *Пусть* G *— хордальный граф,* $T := \mathit{Clique\ Tree}(G)$*,* $uv \in E(G)$*. Тогда* $G - uv$ *хордальный* $\iff$ uv *содержится ровно в одной клике из* $\mathcal{K}_G$*.*

Доказательство. Пусть $G - uv$ хордальный. Предположим, что при этом $\exists K_1, K_2 \in \mathcal{K}_G : uv \in K_1, K_2$. Из-за *intersection property* можно считать, что K_1 и K_2 соседние в T. Так как это максимальные по включению клики, то найдутся вершины $a \in K_1 \setminus K_2$ и $b \in K_2 \setminus K_1$.

Заметим, что $ab \notin E(G)$. Это можно понять следующим образом: пусть при удалении ребра (K_1, K_2) дерево T распадается на компоненты A и B, при этом $K_1 \in A$, $K_2 \in B$. Тогда $\forall K \in A : b \notin K$, иначе нарушится *induced subtree property* для вершины b, так как $b \in K_2 \setminus K_1$. По аналогичной причине $\forall K \in B : a \notin K$. Отсюда следует, что $\nexists K \in \mathcal{K}_G : ab \in K$.

Тогда (a, u, b, v) — это цикл без хорды в графе $G - uv$, значит этот граф не хордальный, противоречие.

Теперь докажем в другую сторону — пусть мы знаем, что ребро uv содержится ровно в одной клике из $\mathcal{K}_G$, хотим понять, что тогда граф $G - uv$ хордальный. Если это не так, то в нём найдётся цикл C длины хотя бы 4 без хорды. Заметим, что в графе G ребро uv является единственной хордой для C. Кроме того, должно быть выполнено $|C| = 4$, иначе G тоже был бы не хордальным (ребро uv разбивает C на два цикла без хорд, тогда если $|C| > 4$, то один из них тоже имеет длину хотя бы 4). Тогда $C = (a, u, b, v)$ для некоторых $a, b \in V$, при этом $ab \notin E(G)$, так как у C нет хорд.

В графе G тройки вершин $A := \{a, u, v\}$ и $B := \{b, u, v\}$ образуют клики. Тогда $\exists K_1, K_2 \in \mathcal{K}_G : A \subseteq K_1, B \subseteq K_2$. При этом $K_1 \neq K_2$, так как $ab \notin E(G)$. Тогда ребро uv содержится в двух кликах из $\mathcal{K}_G$, противоречие.

□

Тогда базовая реализация **Delete Query** работает очень просто: достаточно пройтись по всем кликам из дерева и посчитать, сколько клик содержат ребро uv. Так как для хордального графа G выполнено $|\mathcal{K}_G| \leqslant n$, то данная операция работает за $\mathcal{O}(n)$.

3.2 Delete

Пусть G хордальный, $T :=$ Cique Tree(G), $uv \in E(G)$. Проверим, является ли хордальным $G - uv$. Если нет, то проигнорируем запрос. Иначе мы знаем, что $\exists!\, K \in \mathcal{K}_G : uv \in K$.

Как выглядит множество $\mathcal{K}_{G-uv}$? Заметим, что оно отличается от $\mathcal{K}_G$ тем, что клика K удалилась, а также возможно добавились $K_u := K - v$ и $K_v := K - u$.

Сначала разберёмся со случаем, когда $K_u, K_v \in \mathcal{K}_{G-uv}$. Разобьём соседей вершины K в дереве T на три множества: $N_u := \{K' \in N_T(K) \mid u \in K'\}$, $N_v := \{K' \in N_T(K) \mid v \in K'\}$ и $N_{\overline{uv}} := \{K' \in N_T(K) \mid u, v \notin K'\}$ (каждый сосед K попал ровно в одно множество, так как ребро uv содержалось только в клике K). Тогда преобразуем дерево T следующим образом: удалим K, добавим K_u и K_v, проведём между ними ребро. Кроме того, N_u и $N_{\overline{uv}}$ подвесим к K_u, а N_v к K_v. Можно показать, что *induced subtree property* сохранилось, значит мы получили корректное Clique Tree.

Пусть, например, оказалось, что $K_u \notin \mathcal{K}_{G-uv}$. Тогда $\exists K \in \mathcal{K}_{G-uv} : K_u \subseteq K$. Заметим, что в силу *intersection property* можно считать, что $K \in N_u$. Тогда после того, как мы применили перестроение, описанное в предыдущем абзаце, можно просто стянуть ребро (K_u, K). Аналогично поступим, если $K_v \notin \mathcal{K}_{G-uv}$.

Ясно, что перестроить T, посчитать хэш-множества для добавленных вершин и размеры пересечений для новых пар соседних клик можно за $\mathcal{O}(n)$.

3.3 Insert Query

Теорема 5. *Пусть G — хордальный граф, $uv \notin E(G)$. Тогда $G + uv$ хордальный, если и только если существует такое $T :=$ Clique Tree(G), что $(K_u, K_v) \in E(T)$ для некоторых $K_u, K_v \in \mathcal{K}_G$ таких, что $u \in K_u, v \in K_v$. Иными словами, нужно, чтобы нашлось дерево, в котором соседствуют клики, содержащие вершины u и v.*

Доказательство. Пусть мы знаем, что $G + uv$ хордальный, хотим показать, что для G найдётся Clique Tree с нужным свойством. Пусть $T' :=$ Clique Tree$(G + uv)$. Перестроим T' в соответствии с процедурой удаления ребра uv, описанной в секции 3.2. Клики, содержащие вершины u и v окажутся соседними (это K_u и K_v, либо другая пара, получившаяся после стягиваний рёбер).

В другую сторону: пусть нашлось $T :=$ Clique Tree(G) такое, что $(K_u, K_v) \in E(T)$ для некоторых K_u, K_v таких, что $u \in K_u, v \in K_v$. Тогда заметим, что $S := K_u \cap K_v$ является uv-сепаратором: пусть после удаления ребра (K_u, K_v) дерево T распалось на компоненты X и Y, при этом $K_u \in X$, $K_v \in Y$. Пусть $A := \bigcup\limits_{K \in X} K$, $B := \bigcup\limits_{K \in Y} K$. Заметим, что $A \cap B = S$ (если в пересечении есть какая-то ещё вершина, нарушится *induced subtree property*, так как она не лежит либо в K_u, либо в K_v). Тогда в $G - S$ множества A и B разделены, а значит разделены и вершины u и v.

Предположим, что $G + uv$ не является хордальным, тогда в нём есть цикл C длины хотя бы 4. Так как G был хордальным, то ребро uv обязано лежать на цикле C. То есть $C = (u, v, a_1, \ldots, a_k)$, где $k \geqslant 2$. Так как S разделяет u и v в G, то $\exists a_i \in S$. Но тогда одно из рёбер (u, a_i) и (v, a_i) является хордой (эти рёбра есть, так как они содержатся в кликах K_u и K_v соответственно). Противоречие. □

Мы получили какой-то критерий, но пока не очень понятно, как его проверять, ведь критерий говорит о существовании подходящего Clique Tree, а мы поддерживаем только одно конкретное. Следующая теорема разрешит этот вопрос.

Теорема 6. *Пусть* G *хордальный граф,* $T := \mathit{Clique\ Tree}(G)$, $uv \notin E(G)$. *Также пусть* X *и* Y *— две ближайшие клики в дереве* T *такие, что* $u \in X$, $v \in Y$.

Тогда $\exists T' := \mathit{Clique\ Tree}(G), (K_u, K_v) \in E(T') : u \in K_u, v \in K_v \iff \min_{e \in Path(X,Y)} w(e) = |X \cap Y|$, *где вес ребра* $e = (K_1, K_2)$ *определяется, как* $|K_1 \cap K_2|$.

Доказательство. Если на пути между X и Y нашлось такое ребро $e = (K_1, K_2)$, что $|K_1 \cap K_2| = |X \cap Y|$, то $T' := (T - e + XY)$ — подходящее нам Clique Tree (оно является Clique Tree, так как совпадает по весу с T, а значит тоже является максимальным остовным деревом для графа W_G).

В другую сторону: пусть нашлось Clique Tree T' с нужными нам свойствами, то есть для некоторых K_u, K_v выполнено $u \in K_u, v \in K_v, (K_u, K_v) \in E(T')$. Посмотрим на K_u и K_v в дереве T. Понятно, что $Path_T(X, Y) \subseteq Path_T(K_u, K_v)$ (из-за *induced subtree property* для вершин u и v). Тогда $\forall e \in Path_T(X, Y) : |K_u \cap K_v| \leqslant |X \cap Y| \leqslant w(e)$ (из-за *intersection property*).

Теперь посмотрим на X и Y в дереве T'. Пусть при удалении ребра (K_u, K_v) дерево T' распадётся на две компоненты A и B, при этом $X \in A, Y \in B$ (они окажутся в разных компонентах из-за *induced subtree property* для вершин u и v). Тогда среди рёбер из пути $Path_T(X, Y)$ найдётся такое ребро e, которое соединяет компоненты A и B. Так как оно не попало в T', а T' является максимальным остовным деревом графа W_G, выполнено $w(e) \leqslant w(K_u, K_v) = |K_u \cap K_v|$. Тогда в совокупности с предыдущим абзацем мы получили, что $\exists e \in Path_T(X, Y) : w(e) \leqslant |K_u \cap K_v| \leqslant |X \cap Y| \leqslant w(e) \Longrightarrow w(e) = |X \cap Y|$, что и требовалось. □

3.4 Insert

Пусть G хордальный, $T := \text{Clique Tree}(G)$, $uv \notin E(G)$. Проверим, является ли хордальным $G + uv$. Будем считать, что в T есть пара соседних клик (K_u, K_v), где $u \in K_u, v \in K_v$ (В доказательстве теоремы 6 описано, как этого добиться).

Как поменяется $\mathcal{K}_{G+uv}$ по сравнению с $\mathcal{K}_G$? Во-первых, должны добавиться какие-то клики, содержащие ребро uv. Заметим, что это будет единственная клика $K := \{u, v\} \cup S$ (K клика, так как $S = K_u \cap K_v$, при этом она максимальная по включению, так как S это uv-сепаратор, в частности $N(u) \cap N(v) \subseteq S$). Несложно проверить, что от добавления S перестать быть максимальными по включению могли быть только клики K_u и K_v.

В итоге перестраивать дерево T мы будем следующим образом: добавим клику K, проведём из неё рёбра в K_u и K_v. Если $K_u \subseteq K$, стянем ребро (K_u, K). Аналогично с K_v. Можно показать, что после таких операций *subtree induced property* сохранилось. Опять же, все вещи, которые мы должны поддерживать, можно пересчитать за $\mathcal{O}(n)$.

4 Невошедшее в доклад

То, что я хотел рассказать, но не успел. Вряд ли это вообще кто-то будет читать, так что опишу идеи довольно высокоуровнево.

4.1 Оптимизации

4.1.1 Ускорение Insert Query

Будем хранить наше Clique Tree в Link-Cut Tree, а также дополнительно для каждой вершины хранить любую клику, в которой она содержится. Пусть нам нужно ответить на запрос добавления для ребра (u, v) и мы знаем, что $u \in K_u, v \in K_v$. Во-первых, нам нужно найти ближайшие X и Y такие, что $u \in X, v \in Y$. Для этого подвесим дерево за вершину K_u, теперь путь из K_v ведёт в корень. Сделаем два бинпоиска:

1. самая высокая вершина на этом пути, которая содержит v (это будет Y)
2. самая низкая вершина на этом пути, которая содержит u (это будет X)

Теперь нужно проверить, что $|X \cap Y| = \min_{e \in Path(X,Y)} w(e)$. Минимум на пути мы можем поддерживать в линк-кате. Размеры пересечений для всех пар клик авторы предлагают преподсчитать просто за сколько-нибудь (за $O(n^3)$ или чуть умнее за $O(nm)$, используя PEO), а потом пересчитывать за $O(n)$ на запрос. Не буду описывать, как конкретно это делается, потому что это просто какой-то разбор случаев, но основная идея в том, что мы добавляем $O(1)$ клик, которые мало отличаются от каких-то из присутствовавших до этого, так что размер их пересечения с любой другой кликой мы можем понять за $O(1)$.

Стоит отметить, что такая оптимизация ухудшает время работы операции **Delete** до $O(n \log n)$ на запрос, так как после неё в дереве может поменяться $O(n)$ рёбер.

4.1.2 Ускорение Delete Query

Пусть нам нужно ответить на запрос удаления для ребра (u, v). Другими словами, нам нужно понять, правда ли, что $|\mathcal{K}_G(u) \cap \mathcal{K}_G(v)| = 1$.

Можно показать, что $\sum_{K \in \mathcal{K}_G} |K| \leqslant 2m$ (такой же индукцией, как для размера $\mathcal{K}_G$). Тогда применим корневую декомпозицию: $\lfloor \sqrt{m} \rfloor$ самых больших клик будем обрабатывать отдельно (то есть при каждом запросе просто проходиться по ним и проверять, правда ли, что ребро uv там лежит). А для остальных будем хранить табличку $\mathrm{common}(u, v)$ — количество «маленьких» клик, в которых содержится ребро uv. Можно заметить, что пересчёт при добавлении/удалении рёбер делается за $O(m)$: действительно, добавляется и удаляется $O(1)$ клик. Если добавилась или удалилась «маленькая» клика, мы проходим по всем парам её вершин (их $O(m)$) и обновляем значение common для них.

Таким образом, мы ускорили **Delete Query** до $\mathcal{O}(\sqrt{m})$, но испортили операции **Insert** и **Delete** до $\mathcal{O}(m)$.

4.2 Восстановление PEO по Clique Tree

Пусть мы знаем $T := \text{Clique Tree}(G)$, а также для каждого ребра дерева (K_1, K_2) мы знаем $K_1 \setminus K_2$ и $K_2 \setminus K_1$. Как тогда восстановить $\text{PEO}(G)$?

Запустим обход в глубину из произвольной вершины дерева. Строим PEO с конца, поддерживая его корректным в каждый момент времени.

Пусть мы начинаем обход из клики K, положим в PEO все вершины $v \in K$ в любом порядке.

Пусть сейчас мы находимся в некоторой клике K_1 и хотим перейти по ребру $K_1 \to K_2$. Заметим, что вершины из множества $K_2 \setminus K_1$ не встречались в посещённых ранее кликах (в K_1 их нет по определению, а в предыдущих кликах из-за *induced subtree property*). Тогда добавим их в начало PEO в любом порядке, оно сохранит корректность.