стр. 6 To clear an edge e = (u, v), a searcher must traverse the edge from one end-point u to the other end-point v. Так что очищение здесь — немного другое.

Поиск, который описывается в статье, называется рёберным поиском. Я рекомендую просмотреть раздел 2 внимательней, чтобы точно понимать, что все верно

несколько другой смысл: наложить условие связности — то же, что запретить третье, и допустить первое условие только на первом шаге или и т.д.

Это https://ru.wikipedia.org/wiki/Декомпозиция_графа_на_ветви

Используется термин "ширина декомпозиции T"

Для branchwidth используется термин "ширина ветвления"

Краткий реферат по статье «Connected Graph Searching»

Связный поиск на графе.

Возможные действия:

1. поставить преследователя на вершину,
2. передвинуть преследователя в соседнюю вершину,
3. убрать преследователя с вершины.

К классической постановке задачи добавляется условие связности: на каждом шаге множество очищеных ребер должно быть связно. Кроме того, можно выделить задачи, в которых недопускается третье действие, а первое допускается только на первом шаге или, если мы очищаем вершину, инцидентную уже очищеному ребру.

Цена связности

Цена связности — насколько больше нужно преследователей в случае связного поиска, по сравнению с несвязным.

$s(G)$ — вершинно-поисковое число графа G, $cs(G)$ — аналогичная характеристика, но в случае связного поиска, а $mcs(G)$ — в случае монотонного связного поиска.

Теорема 1 Для любого связного графа с n вершинами верно, что

$$\frac{cs(G)}{s(G)} = O(logn).$$

Branch-разложение (branch decomposition) графа G — это дерево T такое, что все его внутрение вершины имеют степень 3, и существует взаимно-однозначное соответствие между его листьями и ребрами G.

Если удалить ребро e из дерева T, то мы получим два дерева $T_1^{(e)}$ и $T_2^{(e)}$. Тогда *e-cut* определяется как пара $\{E_1^{(e)}, E_2^{(e)}\}$, где $E_i^{(e)} \subset E$ — это множество листьев $T_i^{(e)}$ для $i = 1, 2$.

Вес T определяется как

$$\omega(T) = \max_{e \in ET} \delta(E_1^{(e)}).$$

Branch-вес G:

$bw(G) = \min_T \omega(T)$ (минимум берется по всем branch-разложениям графа G).

Branch-разложение T графа G *связно*, если для любого ребра e из T множества $E_1^{(e)}$ и $E_2^{(e)}$ формируют связные подграфы графа G.

Квартет в branch-разложении T — это упорядоченный набор (A_1, A_2, B_1, B_2) из четырех попарно непересекающихся поддеревьев дерева T таких, что

1. существует ребро $e = \{x, y\} \in ET$ такое, что корни a_1 и a_2 деревьев A_1 и A_2 являются соседними для вершины x в T, а корни b_1 и b_2 деревьев B_1 и B_2 являются соседними для вершины y в T;
2. $\partial(A_1, B_1) \neq \emptyset,\ \partial(A_2, B_2) \neq \emptyset$;
3. $\partial(A_1, A_2) = \emptyset$.

$\partial(A, B) = \{v \in VT |\ \exists e_1 \in EA,\ \exists e_2 \in EB:\ v \in e_1,\ v \in e_2\}$

Алгоритм Make-it-Connected

```
input: branch-разложение T для 2-edge-connected графа веса k;
output: связное branch-разложение T' веса ≤ k;
begin
  S := T;
  while существует квартет (A1, A2, B1, B2) в S do
    заменяем (A1, A2, B1, B2) в S на (A1, B1, A2, B2) и получаем S';
    if (A1, A2, B2, B1) не квартет в S then S := S'
    else
      заменяем (A1, A2, B1, B2) в S на (A1, B2, A2, B1) и получаем S'';
      if ω(S') ≤ ω(S'') then S := S' else S := S'';
    endif
  endwhile
  T' := S;
end
```

Лемма 1 Пусть T — branch-разложение 2-edge-connected графа G веса k. Тогда алгоритм Make-it-Connected строит связное branch-разложение T' веса $\leq k$ за время $O(m^3)$.

Лемма 2 Пусть T — branch-разложение графа G веса k. Тогда можно найти увеличивающийся связный $(k \log_2 m)$-клубок $X_0, \ldots, X_m$ в G за время $O(m^3)$.

k-клубок $X_0, \ldots, X_r$ в графе G *связен*, если подграф сформированый из ребер множества X_i связен для $i = 1, \ldots, r$.

В итоге, можно проследить цепочку рассуждений про декомпозицию на ветви -> связную декомпозицию -> монотонные связные клубки -> монотонную связную стратегию на графе. И без доказательств эту цепочку явно выделить и рассказать.

Посмотрите ещё стр. 13 последний абзац.

Связный поиск на дереве.

Теорема 2 Если дерево T не линейно, то для него выполняется следующее неравенство
$$s(T) \leq cs(T) \leq 2s(T) - 2,$$
и это неравенство нельзя улучшить.

Лемма 3 $cs(T) = mcs(T)$ для любого дерева.

Во втором разделе все до гусениц , действительно, можно опустить. Кроме формулировки Теоремы 2, конечно.

На стр. 16: Our second step for the proof of Theorem 2 is to prove that k-caterpillars are exactly the graphs that can be connectedly cleared with at most k + 1 searchers.

Так что именно про гусениц хорошо бы рассказать, и привести примеры гусениц для малых k.

Далее, на той же странице 16 параграф, который начинается с “Given a tree T and two vertices v,w…” вводит два важных понятия, в терминах которых формулируется Лемма 5, это основной результат для доказательства Теоремы 2.

Попробуйте разобрать небольшой кусочек на стр. 18 “Proof of Theorem 2.” Он основывается на Лемме 5 и понимании D_k и B_k.

Я бы сказала, что начиная с пункта 4.2 можно ничего не разбирать и не рассказывать. В целом, конечно, на ваше усмотрение, если что-то очень нравится, конечно, озвучивайте.

Итого:
Перечитать аккуратно второй раздел — его нужно описать и рассказать подробно, потому что это не та же задача, что у нас на лекциях.

Третий раздел — составить цепочку утверждений, ведущих к связной монотонной стратегии, и все определения, с ней (цепочкой) связанные.

Четвертый — попробовать разобраться в обозначениях, Лемме 5 и кусочке доказательства Теоремы 2.

k-гусеница определяется рекурсивно:

1. 0-гусеница — это путь,
2. для $k \geq 1$, граф G является k-гусеницей, если это дерево, содержащее путь P (*spine*) такой, что компоненты связности $G - VP$ являюся $(k-1)$-гусеницами.

Теорема 3 Сущесвует алгоритм, котрый за линейное время вычисляет вершинно-поисковое число дерева T и монотонную связную выигрывающую программу на нем.

$cs_x(T) = \min_Z \max_i |Z_i|$, где минимум берется по всем монотонным связным выигрывающим программам Z таким, что $Z_1 = x$.

$$cs(T) = \min_{x \in VT} cs_x(T).$$

$$cs^+_x(T) = \max\{1, cs_x(T)\}.$$

$T[y]$ — поддерево дерева T с корнем в y, состоящее из y и всех его потомков.

Лемма Пусть $y_1, \ldots, y_d$ $(d \geq 1)$ — потомки вершины y в дереве T.

1. Если $d = 1$, то $cs^+_y(T[y]) = cs^+_{y_1}(T[y_1])$.
2. Если $d > 1$, то упорядочив y_i так, что $cs^+_{y_i}(T[y_i]) \geq cs^+_{y_{i+1}}(T[y_{i+1}])$ для $1 \leq i < d$, мы получим

$$cs^+_y(T[y]) = \max\{cs^+_{y_1}(T[y_1]), cs^+_{y_2}(T[y_2]) + 1\}.$$