И.Б.Бурдонов.
ПРОБЛЕМА ОТКАТА ПО ДЕРЕВУ ПРИ ОБХОДЕ НЕИЗВЕСТНОГО ОРИЕНТИРОВАННОГО ГРАФА КОНЕЧНЫМ РОБОТОМ.
"Программирование". 2004. No. 6.

Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом

И.Б.Бурдонов
Institute for System Programming of Russian Academy of Sciences (ISPRAS),
B. Communisticheskaya, 25, Moscow, Russia
igor@ispras.ru
http://www.ispras.ru/~RedVerst/

Abstract. Обход ориентированного графа – это маршрут, проходящий через все вершины и дуги графа, причем дуга проходится только в направлении ее ориентации. Обход с любой стартовой вершины существует только для сильно-связных графов. Обход неизвестного графа означает, что топологию графа мы узнаем только в процессе движения по нему, что аналогично задаче обхода лабиринта роботом, находящимся внутри него и не имеющем плана лабиринта. Если робот – это «компьютер общего вида» без ограничения на число его состояний, то известны алгоритмы обхода с оценкой $O(nm)$, где n – число вершин, а m – число дуг графа. Если число состояний ограничено, то робот – это конечный автомат, аналог машины Тьюринга, в которой лента заменена графом, а ее ячейки привязаны к вершинам и дугам графа. Выбор роботом еще не пройденной им дуги, исходящей из текущей вершины, определяется заданным извне порядком исходящих дуг для каждой вершины.

Наилучшие известные алгоритмы обхода для конечного робота основаны на построении выходящего остова графа с корнем в стартовой вершине и обходе его с целью поиска всех еще непройденных дуг. При этом возникает проблема отката (*backtracking*) по дереву: перебор всех вершин дерева в порядке обратном их естественной частичной упорядоченности – от листьев к корню. Поэтому верхняя оценка алгоритмов отличается от оптимальной $O(nm)$ на величину, требуемую для совершения отката по выходящему дереву. Наилучшая известная оценка $O(nm+n^2 loglogn)$ предложена автором в предыдущей статье [1].

В настоящей статье предлагается конечный робот, который выполняет откат по дереву с оценкой $O(n^2 log^*(n))$. Функция log^* определяется как целочисленное решение неравенства $1 \le log_2^{log^*}(n) < 2$, где $log^t = log \circ log \circ \ldots \circ log$ (знак суперпозиции $\circ$ применяется $t-1$ раз) – t-ая композиционная степень логарифма. Для обхода графа соответствующая оценка $O(nm+n^2 log^*(n))$ получается на любом сильно связном графе при некотором (но, к сожалению, не любом) порядке исходящих дуг. Интересно, что такой порядок дуг может быть отмечен символами конечного робота, совершающего обхода графа. Тем самым, можно построить робот, который дважды обходит граф, первый раз с оценкой $O(nm+n^2 loglogn)$, а второй раз с оценкой $O(nm+n^2 log^*(n))$.

1. Введение

Данная статья посвящена той же проблеме, что и предыдущая статья автора [1]: проблеме обхода неизвестного графа конечным роботом. Эта проблема была поставлена М.О. Рабином в 1967 г. [2]. Под обходом понимается маршрут, начинающийся в заданной стартовой вершине и проходящий через каждое ребро графа. Для ориентированного графа каждое ориентированное ребро (дуга) может проходиться только в направлении его ориентации. Предполагается, что граф заранее неизвестен и его топология может выясняться только в процессе движения по дугам. Для существования обхода ориентированного графа с любой стартовой вершины, граф должен быть сильно-связным, то есть, каждая его вершина должна быть достижима из каждой вершины по некоторому маршруту. Конечный робот – это аналог машины Тьюринга, в которой лента заменена графом: ячейка, хранящая символ внешнего алфавита робота, соответствует вершине или дуге графа, а перемещение робота происходит по дуге в направлении ее ориентации. Робот решает проблему обхода, если на любом сильно связном графе с любой стартовой вершиной он останавливается через конечное число шагов и пройденный им маршрут является обходом.

Робот должен каким-то образом указывать, по какой исходящей дуге он перемещается из текущей вершины. Если дуги, исходящие из вершины v перенумерованы $1..d_{out}(v)$, где $d_{out}(v)$ – полустепень исхода вершины v, робот может указывать номер исходящей дуги. Однако, в этом случае робот будет конечным только тогда, когда полустепень исхода вершин ограничена сверху. Это ограничение легко снимается, если добавить ячейку памяти для каждой дуги, исходящей из вершины v, и связать эти ячейки в цикл, который будем называть v-циклом (рис.1). Граф с заданными v-циклами исходящих дуг во всех вершинах v графа будем называть *упорядоченным* графом. Для робота добавляется *внутреннее* перемещение (обозначенное буквой i) на ячейку следующей по v-циклу дуги. При *внешнем* перемещении (обозначенном буквой o) по дуге $(v,v`)$ робот попадает в ячейку первой дуги в $v`$-цикле. Робот указывает, какой переход он делает: внутренний (i) или внешний (o). Тем самым роботу не нужно идентифицировать исходящую дугу $(v,v`)$, по которой он хочет пройти, – это всегда текущая дуга, в ячейке которой он находится.

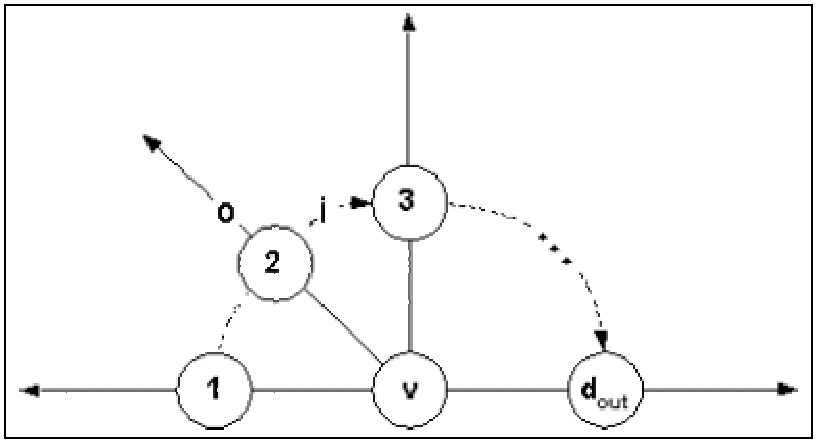

Рис.1: Вершина v и v-цикл исходящих дуг

Мы будем считать, что роботу одновременно доступны для чтения и записи две ячейки: ячейка текущей вершины и ячейка текущей дуги, исходящей из этой вершины. Заметим, что это не является ограничением. Если ячейка вершины v отсутствует, то вместо нее всегда может использоваться ячейка первой дуги v-цикла. Попадая в вершину, робот делает пометку в ячейке первой дуги v-цикла, считывает из нее информацию о вершине и

запоминает в своем состоянии. Для изменения информации о вершине, робот по *v*-циклу перемещается в помеченную ячейку первой дуги и делает запись в нее.

Минимальная длина обхода сильно-связного графа имеет оценку $\Theta(nm)$, где n – число вершин, а m – число дуг графа. В настоящее время наилучшие известные алгоритмы обхода графа конечным роботом основаны на построении двух остовов графа: выходящего и входящего дерева с корнем в стартовой вершине. Выходящее дерево используется для перемещения робота из стартовой вершины в любую вершину и, тем самым, в начало любой еще не пройденной дуги графа. Входящее дерево используется для возвращения в стартовую вершину после прохода хорды выходящего дерева.

После того, как все дуги, исходящие из листовой вершины выходящего дерева, пройдены, эта вершина и ведущая в него дуга дерева удаляются (помечаются специальными терминальными метками). Таким образом выходящее дерево всегда заканчивается только в таких листовых вершинах, из которых исходят еще не пройденные дуги графа. В целом вершины терминируются в порядке обратном их естественной частичной упорядоченности по выходящему дереву: от листьев к корню. Этот процесс называется откатом (*backtracking*) по дереву. Робот останавливается, когда терминируется корень дерева – стартовая вершина.

В зависимости от того, как обходится выходящее дерево, различают алгоритмы поиска в глубину (DFS) и в ширину (BFS). В обоих вариантах верхняя оценка отличается от оптимальной $O(nm)$ на величину, требуемую для совершения полного отката по выходящему дереву. В 1971 г. автор статьи предложил BFS-алгоритм с откатом за $O(n^2 \log n)$ проходов по дугам [3]. В 1993 г. Y.Afek и E.Gafni описали DFS-алгоритм с той же оценкой отката [4]. В предыдущей статье автора оценка улучшена до $O(n^2 \log\log n)$ [1].

Данная статья специально посвящена проблеме отката по дереву. Для возвращения в корень дерева (стартовую вершину) в каждой листовой вершине добавляется хорда, ведущая в корень. Предлагается конечный робот, который выполняет DFS-откат по дереву с оценкой $O(n^2 \log^*(n))$. Функция $\log^*$ определяется как «число логарифмирований» – целочисленное решение неравенства $1 \le \log_2^{\log^*(n)} < 2$, где $\log^t = \log \circ \log \circ \ldots \circ \log$ (знак суперпозиции $\circ$ применяется $t-1$ раз) – t-ая композиционная степень логарифма.

К сожалению, этот результат еще не означает, что можно построить робот для обхода любого сильно связного графа с оценкой $O(nm+n^2 \log^*(n))$. Дело в том, что до завершения обхода робот может оказаться в вершине, из которой нет маршрута, ведущего в стартовую вершину и состоящего только из пройденных дуг. В этом случае в пройденной части графа вместо входящего дерева будет построен лес входящих деревьев, и робот сможет вернуться не в корень выходящего дерева (стартовую вершину), а в корень последнего из таких входящих деревьев. Проблема отката в такой ситуации усложняется и оценка ее решения может превысить $O(n^2 \log^*(n))$. Выделение роботом выходящего дерева и леса входящих деревьев в данном графе при данной стартовой вершине определяется упорядочиванием графа, то есть, порядком дуг в *v*-циклах для всех вершин *v*. Этот порядок задается извне и по сути определяет выбор роботом еще непройденной дуги среди множества непройденных дуг, исходящих из текущей вершины. В завершении статьи показано, что для любого сильно связного (неупорядоченного) графа существует такой порядок дуг в циклах исходящих дуг, при котором в процессе работы робота лес входящих деревьев всегда состоит только из одного дерева с корнем в стартовой вершине. Такой порядок дуг может быть отмечен символами конечного робота, совершающего обхода графа. Тем самым, можно построить робот, который дважды обходит граф, первый раз с оценкой $O(nm+n^2 \log\log n)$, а второй раз с оценкой $O(nm+n^2 \log^*(n))$.

2. Определение робота на графе

Определим граф и робот на графе формально. Мы будем использовать алгебраическую запись для функций, то есть, вместо $f(x)$ писать xf.

Ориентированный граф, на котором работает робот, определяется как $G=(V,E,\alpha,\beta,\gamma,\delta,X,\chi)$, где:

- V – множество вершин;
- E – множество дуг (для удобства будем считать, что $E \cap V = \varnothing$);
- $\alpha: E \to V$ – функция, определяющая начальную вершину (начало) дуги;
- $\beta: E \to V$ – функция, определяющая конечную вершину (конец) дуги;
- $\gamma: V \to E$ – функция, определяющая первую дугу в цикле исходящих дуг, с условием:
  - $\forall v \in V \; v\gamma\alpha = v$;
- $\delta: E \to E$ – функция, определяющая следующую вершину в цикле исходящих дуг, с условием:
  - $\forall e \in E \; \exists k = 0..d_{out}(e\alpha)-1 \; e\alpha\gamma\delta^k = e$, где $\delta^k = \delta \circ \delta \circ \ldots \circ \delta$ и знак суперпозиции $\circ$ применяется $k-1$ раз;
- X – множество символов, которые могут храниться в ячейках вершин и дуг;
- $\chi: V \cup E \to X$ – функция, определяющая символы, хранящиеся в ячейках вершин и дуг.

Граф *конечен*, если конечны множества V и E. Вершина v и дуга e *инцидентны*, если $v=e\alpha$ или $v=e\beta$. Дуги e и $e`$ *смежны*, если $e\beta=\alpha e`$. *Маршрут* – это последовательность смежных дуг; *начало* маршрута (*конец* конечного маршрута) – это начало первой (конец последней) дуги маршрута. *[a,b]-маршрут* – конечный маршрут, начинающийся в вершине a и оканчивающийся в вершине b; в этом случае говорят, что вершина b *достижима* из вершины a. *[a,b]*-маршрут *замкнут*, если $a=b$. Маршрут *простой*, если каждая вершина инцидентна не более, чем двум дугам маршрута; *простой путь* – незамкнутый простой маршрут, *простой контур* – замкнутый простой маршрут. Граф *связен*, если для каждой пары вершин, хотя бы одна из них достижима из другой; граф *сильно-связен*, если каждая вершина достижима из каждой вершины. В дальнейшем мы будем рассматривать только конечные сильно-связные графы.

Робот на графе G определяется как $R=(Q,X,T)$, где:

- Q – множество состояний;
- X – множество входных символов, совпадающее с множеством символов графа;
- $T \subseteq Q \times X \times X \times Q \times X \times X \times \{i,o\}$ – множество переходов.

В каждый момент времени робот находится в текущей вершине $v \in V$ на текущей дуге $e \in E$, исходящей из v, то есть, $e\alpha = v$. Робот находится в состоянии $q \in Q$, читает символ вершины vx и символ дуги ex. Переход $(q,vx,ex,q`,x`_v,x`_e,i/o) \in T$ означает, что робот переходит в состояние $q`$, записывает символы в ячейку вершины $vx = x`_v$ и в ячейку дуги $ex = x`_e$. При внутреннем переходе (i) робот остается в той же вершине v, но переходит на следующую дугу в v-цикле $e\delta$. При внешнем переходе (o) робот переходит по дуге e в ее конечную вершину $v\beta$ на первую исходящую из нее дугу $v\beta\gamma$.

Робот конечен, если множества Q и X конечны. Робот детерминирован, если для каждой тройки $(q,vx,ex) \in Q \times X \times X$ существует не более одного перехода $(q,vx,ex,q`,x`_v,x`_e,i/o) \in T$.

Если робот оказывается в состоянии q в текущей вершине v на текущей дуге e и для тройки (q,vx,ex) нет ни одного перехода, будем говорить, что робот *останавливается*. Будем считать, что один символ из X выделен как *начальный* и находится во всех ячейках вершин и дуг в начале работы робота. Текущую вершину в начале работы робота будем называть *стартовой* и обозначать v_1, текущей дугой является первая исходящая дуга $v_1\gamma$. Последовательность внешних переходов, которые делает робот R на графе G с начала работы, очевидно, определяет маршрут в G, начинающийся в стартовой вершине, который мы будем называть пройденным маршрутом. Если робот останавливается, этот маршрут конечен. В данной статье мы будем рассматривать конечные детерминированные роботы.

Алгоритм робота мы будем записывать на языке Си. Возврат из главной функции робота интерпретируется как его остановка. Ячейки вершин и дуг представляются структурами, состоящие из нескольких *полей*. Тем самым, множество символов X – это множество значений таких структур. Как правило, мы будем использовать однобитовые поля, единичные значения которых будем называть *метками*. Начальным символом будем считать структуру с нулевыми значениями всех полей. Текущую вершину будем обозначать через v, а текущую исходящую дугу через e; таким образом, доступ к полю *field* ячейки вершины или дуги осуществляется конструкцией, соответственно, *v.field* или *e.field*. Для внутреннего и внешнего перемещений будем использовать внешние функции *Next* и *Traverse*, изменяющие v и e (рис.2). Мы будем строить ряд все более усложняющихся роботов так, что более сложный робот использует полностью или частично менее сложные роботы. Для этого программы на Си разбиваются на функции с учетом их использования в программах последующих роботов.

```
/* Внутреннее перемещение: e=eδ */
void Next();

/* Внешнее перемещение: v=eβ, e=eβγ */
void Traverse();
```

Рис.2: Внешние функции робота

3. Проблема линейного отката

Граф G, представляющий собой простой контур, будем называть *линейным* графом (рис.3). Последовательность вершин контура $v_1,\ldots,v_n$, для $i=1..n$ $d_{out}(v_i)=1$, для $i=1..n-1$ $v_i\gamma\beta=v_{i+1}$, $v_n\gamma\beta=v_1$. Через $G[i]$ обозначим i-ую дугу графа: для $i=1..n$ $G[i]\alpha=v_i$, для $i=1..n-1$ $G[i]\beta=v_{i+1}$, $G[n]\beta=v_1$.

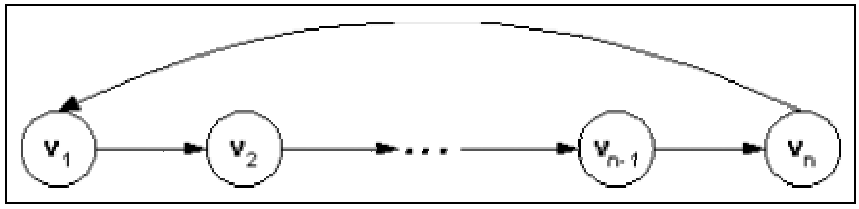

Рис.3: Линейный граф

Поскольку в линейном графе полустепень исхода каждой вершины равна 1, для робота на линейном графе функция *Next* не используется. Будем считать, что в множестве символов робота выделено подмножество *конечных* символов. В записи алгоритмов на Си конечный

символ – это символ, содержащий метку *terminal*. Вершину, помеченную меткой *terminal*, будем называть *терминированной*; а пометку вершины меткой *terminal – терминацией* вершины.

Будем говорить, что робот выполняет *линейный откат*, если он на любом линейном графе останавливается и до остановки терминирует некоторые вершины графа в порядке обратном их естественной упорядоченности $v_{i[m]}, \ldots, v_{i[1]}$, где $i[m]>i[m-1]>\ldots>i[2]>i[1]$. Такой робот будем называть *линейным роботом*. *Проходом* робота по линейному графу будем называть его перемещение от стартовой вершины снова до стартовой вершины или до остановки. Сложность линейного робота мы будем измерять в числе *проходов*. Поскольку при каждом проходе, кроме, быть может, последнего, робот проходит n дуг, длина пройденного маршрута ограничена сверху числом проходов, умноженным на n.

В конечном счете, нас будет интересовать проблема *полного линейного отката*, то есть, отката, при котором робот терминирует все вершины графа, начиная с последней вершины v_n и заканчивая первой вершиной v_1. Робот, решающий эту проблему, будем называть *полным линейным роботом*.

a. Поиск последней вершины за $\Theta(\log n)$ проходов

Поиск последней вершины: робот, начиная с первой вершины линейного графа, должен найти последнюю вершину и закончить работу. Проблема поиска последней вершины – это простейший случай линейного отката, когда терминируется одна последняя вершина. Y.Afek и E.Gafni [1] показали, что проблема поиска последней вершины (названная ими “last in the ring”) решается конечным роботом за $\Theta(\log n)$ проходов. Здесь мы приведем свой вариант доказательства.

Теорема 1: Проблема поиска последней вершины решается конечным роботом за $\Theta(\log n)$ проходов.

Верхняя оценка. Формальное описание робота *R1*, решающего проблему поиска последней вершина за $O(\log n)$ проходов, приведено на рис.5, иллюстрация на рис.4. Вначале все вершины являются кандидатами на последнее место – на первом проходе все они помечаются меткой *candidate*. На каждом следующем проходе робот удаляет половину кандидатов, снимая метки через одну. Остаются те кандидаты, четность которых в последовательности кандидатов совпадает с четностью числа кандидатов и, тем самым, с четностью последней вершины (рис.5). Для этого на каждом проходе робот запоминает четность числа кандидатов и проверяет, сколько осталось кандидатов: один или больше. Когда остается один кандидат – последняя вершина, робот выполняет проход до единственного кандидата и заканчивает работу. Таким образом, робот делает $O(\log n)$ проходов.

Нижняя оценка. Пусть имеется конечный робот, решающий проблему поиска последней вершины. Сначала рассмотрим поведение робота на бесконечном простом пути с последовательностью вершин $v_1, v_2, \ldots$. Будем считать, что на каждом проходе мы помещаем робот в стартовую вершину в некотором произвольно выбранном стартовом состоянии, после чего он двигается по пути, расставляя в вершинах новые символы. Покажем, что перед i-ым проходом робота последовательность S_i символов в вершинах является периодической и может быть представлена в виде $S_i=A_i \hat{} B_i^{\omega}$, где сумма длин предпериода и периода $|A_i|+|B_i| \le |Q|^{i-1}$. Действительно, перед первым проходом во всех вершинах находится начальный символ и $|A_1|=0$, $|B_1|=1$, $|Q|^{1-1}=1$. Далее, по индукции, пусть перед i-ым проходом робота утверждение верно. Рассмотрим первые $|Q|+1$

периодов B_i. Хотя бы в двух из них робот читает первый символ периода $B_i[1]$ в одном и том же состоянии. Пусть первый из двух таких периодов имеет номер j, а второй k: $1 \le j < k \le |Q|+1$. Тогда $|A_{i+1}|=|A_i|+(j-1)|B_i|$ и $|B_{i+1}|=(k-j)|B_i|$. Суммируя, получаем перед следующим $i+1$-ым проходом $|A_{i+1}|+|B_{i+1}| = |A_i|+(k-1)|B_i| \le |A_i|+|Q||B_i| \le |Q|(|A_i|+|B_i|) \le |Q|^i$.

Выберем последовательность стартовых состояний робота на бесконечном пути такую же, какая получается при его работе на конечном линейном графе. Тогда в начале i-го прохода последовательность S_i* символов в вершинах линейного графа будет начальным отрезком последовательности S_i на бесконечном пути. Если на k-ом проходе робот заканчивает работу в последней вершине, то для последовательности S_{k+1}, сумма длин предпериода и периода не меньше n. Получаем $n \le |A_{k+1}|+|B_{k+1}| \le |Q|^k$, то есть, $k=\Omega(\log n)$. □

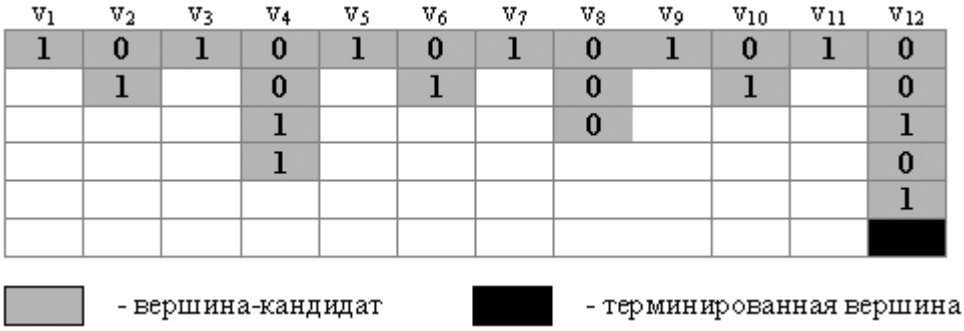

Рис.4: Иллюстрация к поиску последней вершины за $O(\log n)$ проходов

```
struct vertex { /* структура символа вершины */
    unsugned start: 1 = 0; /* стартовая вершина */
    unsigned candidate: 1 = 0;
};
struct vertex v; /* текущая вершина */

void R1() {
    unsigned counter = 0; /* = 0, 1 или 2 */
    unsigned parity = 0;

    /* первый проход */

    v.start = 1;
    do {
        v.candidate = 1; parity ^= 1; if (counter < 2) counter++;
        Traverse();
    } while (!v.start);

    /* средние проходы */

    while (counter > 1) {
        unsigned candidate_parity = 0;
        unsigned new_parity = 0;
        counter = 0;
        do {
```

```
            if (v.candidate) {
                candidate_parity ^= 1;
                if  (candidate_parity != parity) v.candidate = 0;
                else { new_ parity ^= 1; if (counter < 2)
                counter++; }
            }
            Traverse();
        } while (!v.start);
        parity = new_parity;
    }

    /* конечный проход */

    while (!v.candidate) Traverse();
    v.candidate = 0;
}
```

Рис.5: Робот *R1*. Поиск последней вершины за *O(logn)* проходов

b. Фильтрующий робот

Пусть в линейном графе G выделено некоторое подмножество вершин $U \subseteq V$, содержащее стартовую вершину. Тогда для произвольного робота R на этом графе можно построить *фильтрующий робот* $R\langle U\rangle$, который на U-вершинах ведет себя также как робот R, а $V \setminus U$-вершины просто пропускает. Для этого достаточно в программе робота R функцию *Traverse* заменить на функцию *Traverse*$\langle U\rangle$, которая осуществляет фильтрацию вершин по принадлежности к множеству U:

```
void Traverse<U>() { Traverse(); while (v∉U) Traverse(); }
```

Поведение робота $R\langle U\rangle$ на графе G, очевидно, эквивалентно поведению робота R на графе $G\langle U\rangle$, получающемся из G удалением всех вершин из $V \setminus U$ и слиянием двух дуг инцидентных каждой удаляемой вершине. Если робот R останавливается на любом линейном графе и выполняет $T(|V|)$ проходов, то фильтрующий робот $R\langle U\rangle$ также останавливается и выполняет $T(|U|)$ проходов. Если робот R выполняет поиск последней вершины, то робот $R\langle U\rangle$ находит последнюю вершину в множестве U за то же число проходов, за которое робот R находит последнюю вершину в графе $G\langle U\rangle$. Если робот R выполняет полный линейный откат, то робот $R\langle U\rangle$ выполняет на графе G частичный (не полный) линейный откат, терминируя все вершины множества U, что эквивалентно полному линейному откату робота R на графе $G\langle U\rangle$.

Для задания множества U мы будем использовать ниже два способа.

Первый способ. Определим в структуре символа вершины метку *filter*. Множество U – это множество *filter*-вершин. Фильтрация реализуется так:

```
void Traverse_filter() { Traverse(); while (!v.filter)
Traverse(); }
```

Второй способ. Определим в структуре символа вершины метку начала диапазона *range*. Множество *U* – это множество всех вершин между началом диапазона включительно и первой терминированной или стартовой вершиной не включительно. Фильтрация реализуется так:

```
void Traverse_range()
{ Traverse(); if (v.terminal || v.start) while (!v.range)
Traverse(); }
```

Фильтрующий робот с фильтрующей функцией *Traverse_...* будем обозначать *R<Traverse_...>*.

c. Полный линейный откат за *O(nlogn)* проходов

На основе робота *R1* мы построим теперь первый полный линейный робот *R2*.

Теорема 2: Существует конечный робот *R2*, решающий проблему полного линейного отката за *O(nlogn)* проходов.

Формальное описание робота *R2* приведено на рис.6. Для терминации последней нетерминированной вершины вызывается фильтрующий робот *R1<Traverse_range>*, в качестве метки *range* используется метка *start*. Робот *R2* останавливается, когда терминируется стартовая вершина. Обозначим через *k(i)* число проходов робота *R1* на линейном графе с числом вершин *i*. Тогда $k(i) \le C\log_2(i)$ при $i>N$, где *C* и *N* некоторые константы. При $n>N$ число проходов робота *R2* не превосходит $\Sigma\{k(i)|i=1..N\} + \Sigma\{C\log_2(i)|i=N+1..n\} \le \Sigma\{k(i)|i=1..N\} + Cn\log_2 n = O(n\log n)$. □

```
struct vertex { /* структура символа вершины */
    unsugned terminal: 1 = 0; /* метка конечного символа */
    unsugned start: 1 = 0; /* стартовая вершина */
    unsigned candidate: 1 = 0;
};
struct vertex v; /* текущая вершина */

void Traverse_range() /* фильтрующая функция Traverse для робота R1 */
{ Traverse(); if (v.terminal) while (!v.start) Traverse(); }

void R2() {
    do {
        R1<Traverse_range>(); v.terminal = 1;
        while (!v.start) Traverse(); /* возврат в стартовую вершину */
    } while (!v.terminal);
}
```

Рис.6: Робот *R2* на базе робота *R1*. Полный линейный откат за *O(nlogn)* проходов

d. Полный линейный откат за *O(n)* проходов при бесконечном числе символов робота

Теорема 3: Существует робот *R3* с бесконечным числом символов и конечным числом состояний, решающий проблему полного линейного отката за *O(n)* проходов.

Работа робота *R3* состоит из двух этапов (иллюстрация на рис.7, формальное описание на рис.8).

Этап прибавления единицы. На первом проходе первая вершина нумеруется номером *1*. Каждый следующий проход увеличивает на единицу номера нумерованных вершин, ставит номер *1* в первую ненумерованную вершину и проверяет, остались ли ненумерованные вершины. Этап заканчивается, когда все вершины пронумерованы номерами *n,n–1,…,2,1*.

Этап вычитания единицы. На каждом проходе робот уменьшает номера вершин на единицу. Как только в какой-то вершине номер становится равным нулю, эта вершина терминируется. Этап заканчивается, когда терминируется стартовая вершина.

На каждом этапе робот выполняет не более *n* проходов, поэтому общее число проходов не больше *2n*. □

Этап прибавления единицы

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
1											
2	1										
3	2	1									
4	3	2	1								
5	4	3	2	1							
6	5	4	3	2	1						
7	6	5	4	3	2	1					
8	7	6	5	4	3	2	1				
9	8	7	6	5	4	3	2	1			
10	9	8	7	6	5	4	3	2	1		
11	10	9	8	7	6	5	4	3	2	1	
12	11	10	9	8	7	6	5	4	3	2	1

Этап вычитания единицы

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
11	10	9	8	7	6	5	4	3	2	1	■
10	9	8	7	6	5	4	3	2	1	■	■
9	8	7	6	5	4	3	2	1	■	■	■
8	7	6	5	4	3	2	1	■	■	■	■
7	6	5	4	3	2	1	■	■	■	■	■
6	5	4	3	2	1	■	■	■	■	■	■
5	4	3	2	1	■	■	■	■	■	■	■
4	3	2	1	■	■	■	■	■	■	■	■
3	2	1	■	■	■	■	■	■	■	■	■
2	1	■	■	■	■	■	■	■	■	■	■
1	■	■	■	■	■	■	■	■	■	■	■
■	■	■	■	■	■	■	■	■	■	■	■

■ - терминированная вершина

Рис.7: Иллюстрация к полному линейному откату за *O(n)* проходов при бесконечном числе символов

```
struct vertex { /* структура символа вершины */
    unsigned terminal: 1 = 0; /* метка конечного символа */
    unsigned logstart: 1 = 0; /* стартовая вершина */
    unsigned number = 0; /* бесконечное множество возможных значений */
};
struct vertex v; /* текущая вершина */
```

```
#define GO_TO_LOGSTART while (!v.logstart) Traverse();

void Plus_pass() { /* проход прибавления единицы */
    while (v.number){ v.number++; Traverse();  }
    v.number = 1; Traverse();
}

void Minus_pass() { /* проход вычитания единицы */
    while (v.number > 1) { v.number--; Traverse(); }
    v.number = 0;
}

void R3() {
    unsigned counter = 1;
    v.logstart = 1;
    do { /* этап прибавления единицы */
        Plus_pass(); if (v.logstart) counter = 0; GO_TO_LOGSTART
    } while(counter);
    do { /* этап вычитания единицы */
        Minus_pass(); v.terminal = 1; GO_TO_LOGSTART
    } while (!v.terminal);
}
```

Рис.8: Робот *R3*. Полный линейный откат за *O(n)* проходов при бесконечном числе символов

e. Логарифмический линейный откат за *O(n)* проходов

Линейный откат будем называть *логарифмическим*, если в последовательности терминируемых вершин $v_{i[m]}, \ldots, v_{i[1]}$ расстояние между соседними терминируемыми вершинами, а также между крайними терминируемыми вершинами и крайними вершинами графа ограничено сверху логарифмом от числа вершин: $n–i[m]<[\log_2 n]+1$, $i[j]–i[j–1]<[\log_2 n]+1$ для $m \geq j>1$, $i[1]–1<[\log_2 n]+1$. Робот, решающий эту проблему, будем называть *логарифмическим роботом*.

Теорема 4: Существует конечный робот *R4*, решающий проблему логарифмического линейного отката за *O(n)* проходов.

Робот эмулирует работу робота *R3*. Идея заключается в том, что для записи номера *i* используется его представление B_i в виде двоичного позиционного кода, записываемого поразрядно в последовательность вершин. Этот код есть последовательность из двоичных чисел *0* и *1* длиной $[\log_2 i]+1$, причем младший разряд соответствует началу последовательности, а старший разряд содержит *1* и соответствует концу последовательности. Например, для номеров $i=7,8$ их двоичные представления $B_7=111$, $B_8=0001$. Вершина, содержащая младший разряд, помечается меткой *low*. Для логарифмического отката терминируются не все вершины B_i, а первая из них – вершина младшего разряда (исключение составляет код из двух разрядов – терминируются обе вершины). Формальное описание робота приведено на рис.10, иллюстрация на рис.9.

Прибавление единицы к номеру *i* робот производит поразрядно, используя стандартный алгоритм двоичного сложения: *0+1=1*, *1+1=0* и перенос *1* в следующий разряд. Перенос запоминается в состоянии робота. Эта процедура начинается тогда, когда робот находится в вершине младшего разряда номера. Если не происходит переноса *1* из старшего разряда номера, длина двоичного кода не изменяется, и робот прибавляет единицу к следующему номеру. В противном случае, для номера вида $i=2^s-1$ прибавление единицы вызывает перенос из старшего разряда номера и увеличение длины двоичного кода на *1*. Робот помечает меткой *shift* следующую вершину и переносит все разряды всех последующих номеров без изменения на одну вершину вперед. В следующем проходе прибавление единицы начинается с метки *shift*. Если происходит перенос из старшего разряда последнего номера, меткой *shift* помечается стартовая вершина. В целом, последовательность номеров имеет вид $k,k-1,\ldots,r+1,[r,]r-1,\ldots,2,1$, где один номер $k>r\geq 1$ может отсутствовать (очевидно, отсутствующий номер имеет вид $r=2^s-1$). Последний номер всегда равен *1* ($r>1$) или *2* ($r=1$). В функции *Plus_pass* робот выполняет один-два прохода. Если еще ни одна вершина не пронумерована, стартовая вершина становится младшим и единственным разрядом единственного номера *1*. В противном случае сначала определяется последний номер. Если он равен *1*, выполняется прибавление единиц ко всем номерам до первого переноса из старшего разряда. Если последний номер равен *2*, робот находит его и помещает за ним номер *1*. Каждый вызов функции *Plus_pass* добавляет к числу нумерованных вершин ровно одну вершину.

Вычитание единицы из номера робот также производит поразрядно, используя стандартный алгоритм двоичного вычитания: *1–1=0*, *0–1=1* и заем *1* из следующего разряда. Заем запоминается в состоянии робота. Заметим, что на этом этапе двоичный код номера *i* может содержать больше чем $[\log_2 i]+1$ разрядов, то есть, старшие разряды могут быть нулевыми. На этом этапе робот терминирует вершины младших разрядов номеров (за исключением случая двух вершин, когда терминируется последняя вершина). В функции *Minus_pass*, робот сначала, учитывая только нетерминированные вершины, за один проход вычисляет число вершин (*1,2* или больше), число номеров (*1* или больше), а также последний и предпоследний номера. Если имеется одна или две вершины, терминируется последняя вершина. Если имеется больше двух вершин, но только один номер, терминируется стартовая вершина. В остальных случаях выполняется один или два прохода вычитания единицы, пока последний номер не станет равным нулю. Далее за один проход ищется предпоследний номер и терминируется следующая за ним вершина – вершина младшего разряда последнего номера. Этап заканчивается, когда терминируется стартовая вершина.

При каждом вызове функции *Plus_pass* (1-2 прохода) одна вершина нумеруется, поэтому этап прибавления единицы выполняется не более чем за *2n* проходов. На втором этапе одна вершина терминируется не более чем за 4 прохода. Поскольку количество номеров, очевидно, не больше *n*, этот этап выполняется не более чем за *4n* проходов. Суммарно получается *O(n)* проходов. Длина двоичного кода каждого из номеров $k,\ldots,1$, очевидно, не превосходит длины двоичного кода номера $k\leq n$, а длина двоичного кода номера *n* равна $[\log_2 n]+1$. Таким образом, робот выполняет логарифмический линейный откат. □

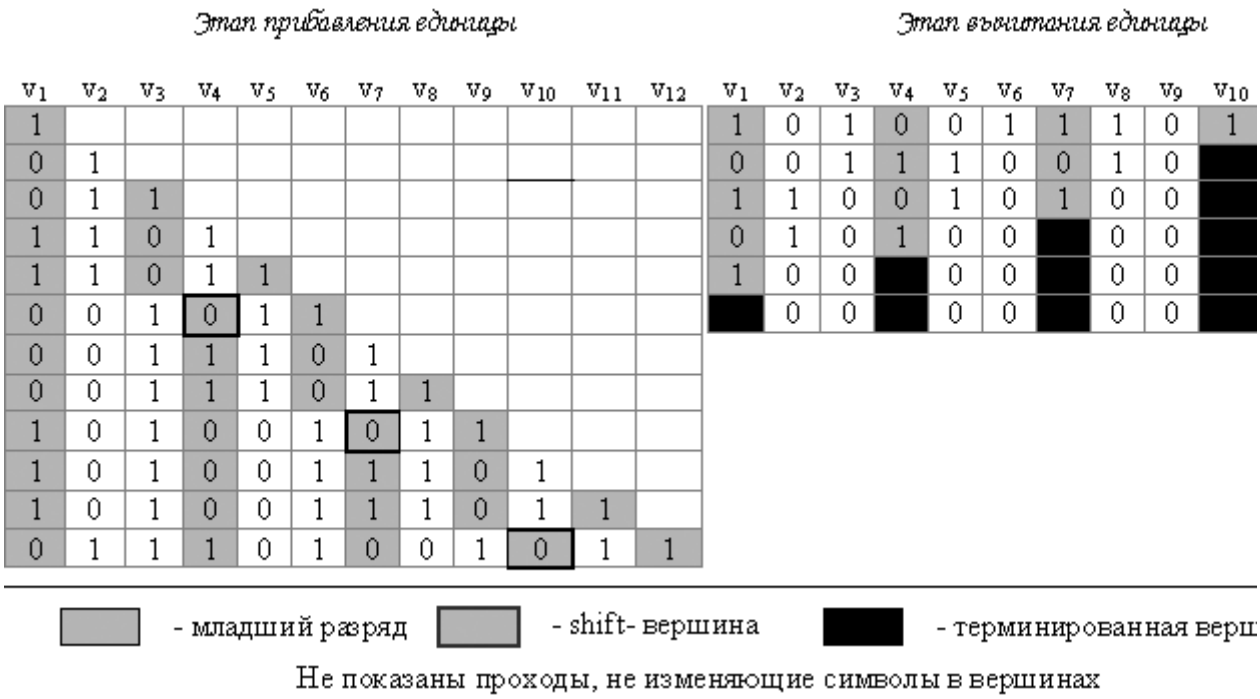

Рис.9: Иллюстрация к логарифмическому линейному откату за *O(n)* проходов

```
struct vertex {  /* структура символа вершины */
    unsigned terminal: 1 = 0;  /* метка конечного символа */
    unsigned logstart: 1 = 0;  /* стартовая вершина */
    unsigned number: 1 = 0;  /* признак разряда номера */
    unsigned low: 1 = 0;  /* признак младшего разряда номера */
    unsigned bit: 1 = 0;  /* содержимое разряда номера */
    unsigned shift: 1 = 0;  /* отсюда продолжается прибавление единицы после
    сдвига */
};
struct vertex v;  /* текущая вершина */

#define END_OF_NUMBER v.low || !v.number || v.logstart ||
v.terminal
#define END_OF_LAST_NUMBER !v.number || v.logstart || v.terminal

unsigned vertex_counter = 0;  /* = 1,2,3  число вершин */

void Number_calculation() {  /* вычисление номера и числа вершин */
    unsigned bit_position = 0;  /* = 0,1,2 */
    unsigned number = 0;  /* = 0,1,2,3,4  */
    do {
        switch (bit_position) {  /* вычисление номера */
            case 0: bit_position++; number = v.bit; break;
            case 1: bit_position++; number += 2*v.bit; break;
            default: if (v.bit) number = 4;
        }
        if (vertex_counter < 3) vertex_counter++;
```

```
        Traverse();
    } while (!END_OF_NUMBER);
    return number;
}

void Shift_pass(low) unsigned low; { /* сдвиг номеров на одну вершину */
    struct vertex prev = {0,0,1,0,1,0}; /* разряд, содержащий 1 */
    struct vertex curr;
    prev.low = low;
    while (v.number)
        { curr = v; v = prev; prev = curr; Traverse(); }
    v = prev; Traverse();
}

char * Plus_pass() { /* Один-два прохода прибавления единицы */
    unsigned carry = 1;
    unsigned last;
    unsigned shift = 0; /* =1 при переносе из старшего разряда непоследнего
    номера */
    unsigned end; /* =1, если все вершины пронумерованы */

    if (!v.number) { /* нет нумерованных вершин */
        Shift_pass(1); /* новый номер 1 */
    else { /* есть нумерованные вершины */
        do { /* вычисление последнего номера */
            last = Number_calculation();
        } while (!END_OF_LAST_NUMBER);
        if (last == 2) { /* последний номер 2 */
            Shift_pass(1); /* новый номер 1 */
        }
        else { /* последний номер 1 */
            GO_TO_LOGSTART
            while (!v.shift) Traverse(); /* поиск shift-вершины */
            v.shift = 0;
            while (1){ /* цикл прибавления единиц */
                do { /* прибавление единицы к одному номеру */
                    v.bit ^= carry; carry &= !v.bit; Traverse();
                } while (!END_OF_NUMBER);
                if (carry) { /* перенос из старшего разряда */
                    if (!END_OF_LAST_NUMBER) { shift = 1; v.shift
                    = 1; } Shift_pass(0); break;
                }
            }
        }
    }
    while (!v.logstart) /* GO_TO_LOGSTART с поиском ненумерованных
    вершин */
    { if (!v.number) end = 0; Traverse(); }
    if (!shift) v.shift = 1; /* нет переноса из старшего разряда непоследнего
    номера */
```

```
    if (end) return "все вершины пронумерованы";
    else return "остались непронумерованные вершины";
}

char * Minus_pass() { /* От одного до четырех проходов вычитания единицы */
    unsigned num_counter = 0; /* = 1,2 число номеров */
    unsigned last = 0; /* = 1,2,3,4 последний номер */
    unsigned next_to_last; /* = 1,2,3,4 предпоследний номер */
    unsigned steal = 1;
    do { /* Проход вычисления числа вершин, числа номеров и последних двух
    номеров */
        next_to_last = last; /* предпоследний номер */
        last = Number_calculation();
        if (num_counter < 2) num_counter++; /* число номеров */
    } while (!END_OF_LAST_NUMBER);
    GO_TO_LOGSTART
    if (vertex_counter == 1) return "одна вершина";
    if (vertex_counter == 2) /* две вершины, номер 2 */
    { Traverse(); return "больше одной вершины"; }
    if (num_counter == 1) /* младший разряд единственного номера */
        return "больше одной вершины";
    next_to_last -= last;
    while (last > 0) { /* От одного до двух проходов вычитания единицы */
        do {
            do { /* вычитание единицы из одного номера */
                v.bit ^= steal; steal &= v.bit; Traverse();
            } while (!END_OF_NUMBER);
        } while (!END_OF_LAST_NUMBER);
        GO_TO_LOGSTART
        last--;
    }
    /* Один проход поиска предпоследнего номера */
    while (Number_calculation() != next_to_last) ;
    return "больше одной вершины"; /* младший разряд последнего номера */
}

void Log_Terminal() { v.terminal = 1; }

void R4() {
    v.logstart = 1;
    /* этап прибавления единицы */
    while (Plus_pass() == "остались непронумерованные вершины") ;
    do { /* этап вычитания единицы */
        Minus_pass(); Log_Terminal(); GO_TO_LOGSTART
    while (!v.terminal);
}
```

Рис.10: Робот *R4* на базе робота *R3*. Логарифмический линейный откат за *O(n)* проходов

f. Комбинация логарифмического и полного линейных роботов

Лемма 1: Пусть $S(k) = \Sigma\{L(i)|i=1..k\} - L(r)$, где $[\log_2 k]+1 \geq L(k) \geq 1$, для $i<k$ $L(i) = [\log_2 i]+1$ и $k>r\geq 1$. Если $S(k) = n$, то при достаточно большом $n > N_2$ имеет место $L(i) \leq \log_2 n$ для $i=1..k$.

Действительно, поскольку $L(k) \geq 1$, $L(i)=[\log_2 i]+1 \geq \log_2 i$ и $\log_2(k-1)+2 \geq [\log_2(k-1)]+1 \geq L(k-1) \geq L(r)$, имеем $n = S(k) \geq 1+\Sigma\{\log_2 i|i=1..k-1\} - \log_2(k-1) - 2 = \log_2(k-1)! - \log_2(k-1) - 1 = \log_2((k-2)!/2)$.
Поскольку логарифм факториала растет быстрее, чем линейная функция, при достаточно большом k ($k\geq 49$) имеем $\log_2((k-2)!/2) \geq 4k$. Отсюда $\log_2 n \geq \log_2 k+2$.
Для $i=1..k$ имеем $L(i) \leq [\log_2 k]+1 \leq \log_2 k+2$.
Таким образом, при достаточно большом k имеем $\log_2 n \geq \log_2 k+2 \geq L(i)$ для $i=1..k$.

С другой стороны, поскольку $L(k) \leq [\log_2 k]+1$, $L(i)=[\log_2 i]+1 \leq \log_2 i+1$ и $L(r) \geq L(1) = 1$, имеем $n = S(k) \leq \log_2 k+1+\Sigma\{\log_2 i+1|i=1..k-1\} - 1 = k-1+\log_2 k!$.
Поскольку функция $k-1+\log_2 k!$ монотонно возрастает, при достаточно большом $n > N_2$ ($N_2=256$) число k также будет достаточно большим ($k>49$) и, следовательно, $\log_2 n \geq L(i)$ для $i=1..k$. □

Лемма 2: Пусть функция $T(n)$ монотонно неубывающая и, начиная с некоторого $n>N$, $L(i) \leq \log_2 n$, $\Sigma\{L(i)\} \leq n$. Тогда $\Sigma\{L(i)T(L(i))\} \leq nT(\log_2 n)$ при $n>N$.

Действительно, поскольку функция T монотонно неубывающая, $\Sigma\{L(i)T(L(i))\} \leq \Sigma\{L(i)T(\log_2 n)\} = \Sigma\{L(i)\}T(\log_2 n) \leq nT(\log_2 n)$. □

Теорема 5: Пусть конечный робот R решает проблему полного линейного отката за $O(nT(n))$ проходов, где $T(n)$ монотонно неубывающая положительная функция. Тогда можно построить конечный робот $R5(R)$, который решает проблему полного линейного отката за $O(nT(\log_2 n))$.

Робот $R5(R)$ строится на базе робота $R4$, который осуществляет логарифмический линейный откат за $O(n)$ проходов. Модификация заключается в изменении функции *Log_Terminal*. Каждый раз, когда робот $R4$ терминирует вершину младшего разряда номера i, робот $R5(R)$ вместо этого ставит в вершину метку начала диапазона *range* и вызывает фильтрующий робот $R<Traverse\_range>$, который выполняет полный линейный откат в диапазоне, соответствующем всем разрядам номера i. После этого метка *range* удаляется.

В программе робота на рис.11 приведены только структура символа вершины, функция *Log_Terminal*, которой модифицированный робот $R4$ отличается от немодифицированного, и фильтрующая функция *Traverse_range*, используемая фильтрующим роботом как описано выше (3.2). Заметим, что при композиции $R5(R)$ в структуру символа подставляются поля роботов $R4$ и R. Возможные коллизии имен разрешаются систематическим переименованием полей.

Число проходов, когда робот $R5(R)$ работает как робот $R4$, очевидно, такое же, как у робота $R4$, то есть, $P_{R4}(n)=O(n)$.

В конце этапа прибавления единицы робота $R4$ последовательность номеров имеет вид $k,k-1,...,r+1,[r,]r-1,...,2,1$, где один номер $k>r\geq 1$ может отсутствовать. Длина диапазона

$L(i) = [\log_2 i]+1$. Сумма длин диапазонов $S(k) = \Sigma\{L(i)|i=1..k\} - L(r) = n$. Таким образом, условия Леммы 1 выполнены и поэтому при $n > N_2$ длины всех диапазонов $L(i) \le \log_2 n$. Фильтрующий робот *R<Traverse_range>* работает в каждом диапазоне, соответствующем разрядам номера i. Число проходов робота в i-ом диапазоне $P_R(i) \le CL(i)T(L(i))$, если $i>N$, где C и N некоторые константы. Поскольку функция T монотонно не убывает, для $i>N$ выполнены также условия Леммы 2 и поэтому суммарное число проходов фильтрующего робота $\Sigma P_R(i) \le \Sigma\{P_R(i)|i=1..N\}+\Sigma\{CL(i)T(L(i))|i=N..k\} \le \Sigma\{P_R(i)|i=1..N\}+CnT(\log_2 n)$ при $n > N_2$. Поскольку функция T монотонно не убывает и положительна, имеем оценку $\Sigma P_R(i) = O(nT(\log_2 n))$.

Таким образом, общее число проходов робота *R5(R)* $P_{R5}(n) = P_{R4}(n) + \Sigma P_R(i) = O(n)+O(nT(\log_2 n))$. Поскольку функция T монотонно не убывает и положительна, имеем оценку $P_{R5}(n) = O(nT(\log_2 n))$. □

```
struct vertex { /* структура символа вершины */
    unsigned terminal: 1 = 0; /* общее поле роботов R4 и R */
    unsigned logstart: 1 = 0; /* поле робота R4 */
    unsigned number: 1 = 0; /* поле робота R4 */
    unsigned low: 1 = 0; /* поле робота R4 */
    unsigned bit: 1 = 0; /* поле робота R4 */
    unsigned shift: 1 = 0; /* поле робота R4 */
    unsigned range: 1 = 0; /* метка начала диапазона */
    . . . /* поля робота R, кроме поля terminal */
};
struct vertex v; /* текущая вершина */

void Traverse_range() { /* фильтрующая функция Traverse для робота R */
    Traverse(); if (v.terminal || v.logstart) while (!v.range)
    Traverse();
}

void Log_Terminal(){ /* откат внутри номера */
    v.range = 1;
    R<Traverse_range>(); /* вызов робота R в диапазоне */
    v.range = 0;
}
```

Рис.11: Робот *R5(R)*. Комбинация логарифмического (*R4*) и полного (*R*) линейных роботов

g. Композиционная степень логарифмического робота и полный линейный откат за $O(n\log_2^t n)$ проходов для любого фиксированного $t \ge 1$

Композиционной степенью логарифма назовем функцию $\log_2^t = \log_2 \circ \log_2 \circ \ldots \circ \log_2$, где знак суперпозиции ∘ применяется $t-1$ раз.

Теорема 6: Для любого целого $t \ge 1$ существует конечный робот *R6(t)*, который решает проблему полного линейного отката за $O(n\log_2^t n)$.

Доказательство теоремы будем вести индукцией по *t*. Согласно Теореме 2, для *t=1* имеем робот *R6(1)=R2* с числом проходов $O(n\log_2 n)$. Пусть утверждение теоремы верно для *t*, то есть, существует робот *R6(t)* с числом проходов $O(n\log_2^t n)$. Докажем утверждение для *t+1*. Обозначая $T(n)=\log_2^t n$, применим теорему 5, используя в роботе *R5(R)* в качестве робота *R* робот *R6(t)*. Мы получим робот *R6(t+1)* с числом проходов $O(nT(\log_2 n)) = O(n\log_2^t \log_2 n) = O(n\log_2^{t+1} n)$, что и требовалось доказать. □

h. Полный линейный откат за *O(nlog*(n))* проходов

Число проходов робота *R6(t)* имеет оценку сверху $O(n\log_2^t n)$. При достаточном увеличении *t* можно получить $\log_2^t n = O(1)$, то есть, можно применять композицию логарифмических роботов до тех пор, пока длина получающихся отрезков не станет меньше некоторой константы. На таких малых отрезках можно уже применять алгоритм отката с числом проходов, ограниченным сверху также некоторой константой. В результате мы должны получить робот с числом проходов порядка *nlog*(n)*, где функция *log** дает необходимое число логарифмирований и определяется как целочисленное решение неравенства $1 \le \log_2^{\log^*(n)} < 2$ (основание логарифмирования для *log** мы не указываем, предполагая его равным *2*).

Однако, остается проблема конечного числа состояний и символов робота. Каждая композиция роботов увеличивает число состояний и символов, которые становятся функциями от *log*(n)* и, тем самым, от *n*. Этого можно избежать, если избавиться от рекурсии при композиции роботов, заменив ее итерацией. При этом оказывается возможным не увеличивать порядок числа проходов.

Теорема 7: Существует конечный робот *R7*, который решает проблему полного линейного отката за число проходов *O(nlog*(n))*.

Формальное описание робота *R7* приведено на рис.13, иллюстрация – на рис.12. Также как робот *R6(t)* робот *R7* строится как система вложенных фильтрующих по диапазонам логарифмических роботов *R4*. Самый внешний диапазон – диапазон уровня *1* – это весь путь от первой до последней вершины. После открытия диапазона уровня *i* выполняется этап прибавления единицы. Затем вызывается функция *Minus_pass* (от одного до четырех проходов). Эта функция сообщает, сколько вершин в диапазоне: одна или больше. Если в диапазоне более одной вершины, открывается новый вложенный диапазон уровня *i+1*, содержащий вершины всех разрядов последнего номера уровня *i*. (В диапазоне длины 2 открывается вложенный единичный диапазон, содержащий последнюю вершину.) Диапазон открывается меткой *logstart*, метка *logstart* объемлющего диапазона удаляется. Исключение составляет случай, когда оба диапазона начинаются с одной вершины. Дополнительно начала всех действующих диапазонов помечены меткой *filter*. При открытии вложенного диапазона в его вершинах устанавливается начальный символ (кроме меток *logstart* и *filter*). Если самый внутренний диапазон содержит единственную вершину (о чем сообщает функция *Minus_pass*), он удаляется, вершина терминируется, снимаются метки *logstart* и *filter*. После этого происходит возврат к объемлющему диапазону. Для этого с помощью фильтрующего робота *R1<Traverse_filter>* ищется последняя *filter*-вершина, то есть, начало объемлющего диапазона и помечается меткой *logstart*.

Оценим число проходов робота. Сначала рассмотрим проходы, выполняемые роботами *R4* (как совокупность функций *Plus_pass* и *Minus_pass*). По Теореме 4, в диапазоне длиной *m* число проходов этого робота $P_{R4}(m) \le Cm$, при $m>N$, где *C* и *N* константы. Подберем *C*

достаточно большое, чтобы неравенство $P_{R4}(m) \le Cm$ выполнялось при любом натуральном m. Самый внешний робот $R4_1$ выполняет не более Cn проходов и размещает в вершинах номера $k,k-1,\ldots,r+1,[r,]r-1,\ldots,2,1$, где один номер $k>r\ge 1$ может отсутствовать. Если $n>1$, для каждого из этих номеров $k_j>1$ создается диапазон из $L(k_j) = [\log_2 k_j]+1$ вершин, на котором работает вложенный робот $R4_2$, выполняя не более $CL(i)$ проходов. В силу аддитивности линейной функции, суммарно робот $R4_2$ также выполняет не более Cn проходов. То же самое относится ко всем вложенным роботам $R4_j$. Таким образом, для всех вложенных роботов $R4$ имеем суммарное число проходов $P^{\Sigma}_{R4}(n) \le Cnt(n)$, где $t(n)$ число уровней вложенности. Согласно Лемме 1, каждый уровень вложенности уменьшает размер диапазона $m>N_2$ до $\log_2 m$. Тем самым, $t(n) \le N_2 + \log^*(n) = O(\log^*(n))$ и $P^{\Sigma}_{R4}(n) = O(n\log^*(n))$.

При открытии каждого диапазона (кроме самого внешнего) выполняется один проход для разметки диапазона начальным символом. После этапа прибавления единицы в конце каждого диапазона из m вершин оказывается номер *1* или *2*. В диапазоне из *2*-х вершин, в свою очередь, выделяется вложенный единичный диапазон, содержащий последнюю вершину. Единичные диапазоны тупиковые (не содержат вложенных диапазонов). Таким образом, вершина может быть концом максимум трех диапазонов (из m, *2-х* и *1-ой* вершины). Поэтому общее число диапазонов всех уровней равно $O(n)$ и, значит, на разметку диапазонов начальным символом суммарно тратится $P_{mark} = O(n)$ проходов.

Остается оценить суммарное число проходов $P^{\Sigma}_{R1}(n)$ фильтрующего робота *R1<Traverse_filter>*. Для поиска последней *filter*-вершины тратится $P_{R1}(t)=O(\log_2 t)$ проходов, где t – число имеющихся *filter*-вершин. Каждый такой поиск выполняется при удалении ставшего единичным диапазона. Тем самым, число поисков не превосходит общего числа диапазонов $O(n)$. Поскольку $t = O(\log^*(n))$, имеем $P^{\Sigma}_{R1}(n) = O(n)O(\log_2\log^*(n)) = O(n\log_2\log^*(n)) = O(n(\log^*(n)-1)) = O(n\log^*(n))$.

В целом, робот *R7* делает число проходов $P_7(n) = P^{\Sigma}_{R4}(n) + P_{mark}(n) + P^{\Sigma}_{R1}(n) = O(n\log^*(n)) + O(n) + O(n\log^*(n)) = O(n\log^*(n))$ проходов. □

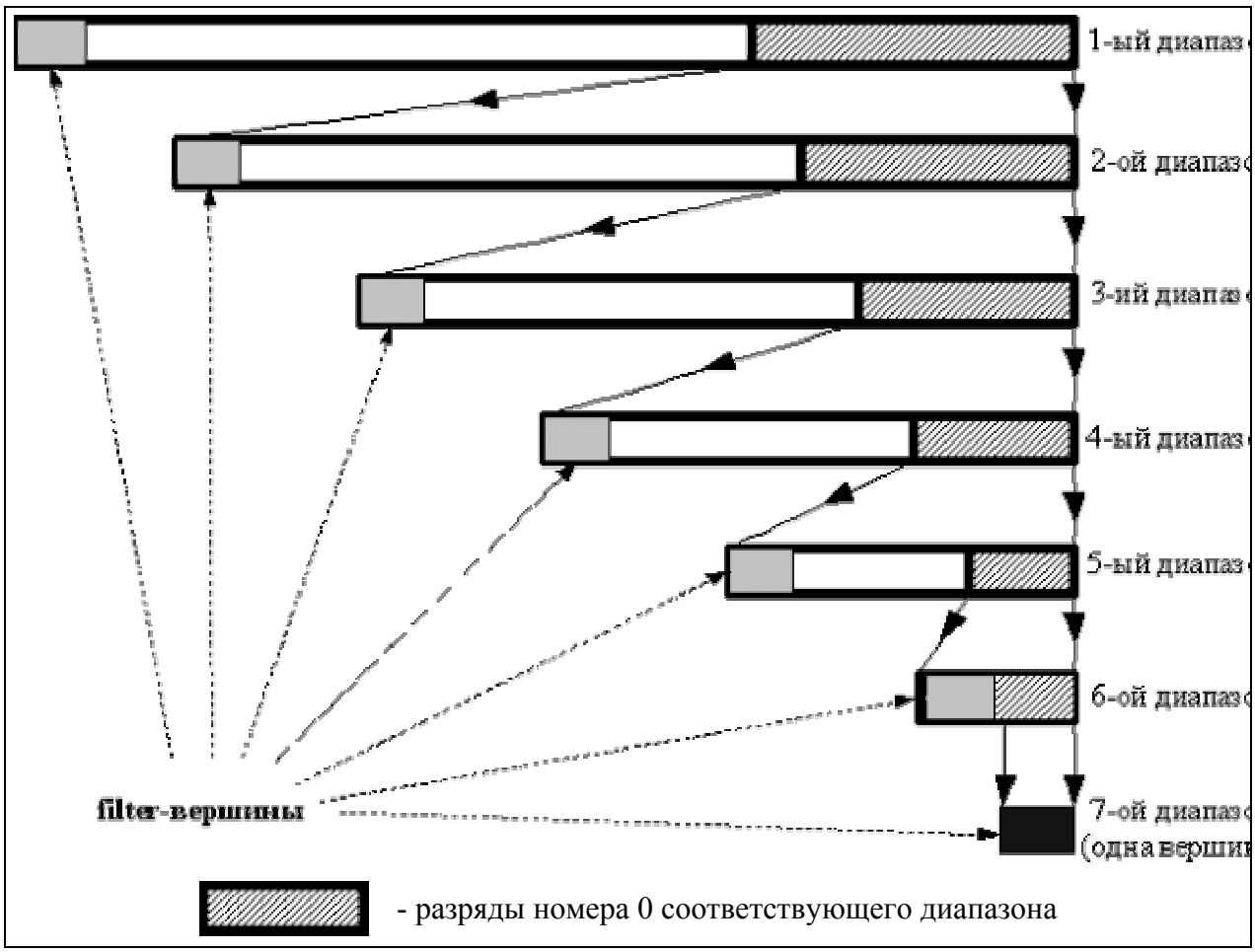

Рис.12: Иллюстрация к полному линейному откату за *O(nlog*(n))* проходов

```
struct vertex { /* структура символа вершины */
    unsigned filter: 1 = 0; /* фильтрующее поле для функции Traverse_filter */
    /* поля робота R4 */
    unsigned terminal: 1 = 0; /* метка конечного символа */
    unsigned logstart: 1 = 0; /* метка начала диапазона */
    unsigned number: 1 = 0; /* признак разряда номера */
    unsigned low: 1 = 0; /* признак младшего разряда номера */
    unsigned bit: 1 = 0; /* содержимое разряда номера */
    unsigned shift: 1 = 0; /* отсюда продолжается прибавление единицы после
    сдвига */
    /* поля робота R1 */
    unsigned start: 1 = 0; /* стартовая вершина */
    unsigned candidate: 1 = 0;
};
struct vertex v; /* текущая вершина */

void Traverse_filter() { /* фильтрующая функция Traverse для робота R1 */
{ Traverse(); while (!v.filter) Traverse(); }

void Traverse_range() /* фильтрующая функция Traverse для робота R4 */
{ Traverse(); if (v.terminal || v.start) GO_TO_LOGSTART }
```

```
char * Minus_pass7() { /* вычитание единицы */
    unsigned new_range_start;
    if (Minus_pass<Traverse_range>() == "больше одной вершины") {
        /* открытие нового вложенного диапазона */
        if (!v.logstart) /* новое начало вложенного диапазона */
        { new_range_start = 1; v.logstart = 1; v.filter = 1; }
        else new_range_start = 0; /* старое начало вложенного диапазона */
        do { /* установка начального символа в диапазоне */
            v.number = 0; v.low = 0; v.bit = 0; v.shift = 0;
            Traverse();
        } while (!v.terminal && !v.start);
        GO_TO_LOGSTART
        if (new_range_start) /* переход на новое начало вложенного диапазона
        */
            { v.logstart = 0; GO_TO_LOGSTART }
        return "больше одной вершины";
    }
    else return "одна вершина"
}

void R7() {
    v.start = 1; v.logstart = 1;
    while (1) {
        /* этап прибавления единицы в диапазоне */
        while (Plus_pass<Traverse_range>()
        == "остались непронумерованные вершины") ;
        while (Minus_pass7() == "одна вершина") { /* вычитание единицы
        */
            v.terminal = 1; /* терминация вершины */
            v.logstart = 0; v.filter = 0; /* удаление диапазона из одной
            вершины */
            if (v.start) return; /* стартовая вершина терминирована */
            R1<Traverse_filter>(); /* поиск начала объемлющего диапазона
            */
            v.logstart = 1; /* переход на объемлющий диапазон */
        } /* в диапазоне больше одной вершины */
    }
}
```

Рис.13: Робот *R7* на базе логарифмического робота (*R4*) и робота поиска последней вершины (R_1).
Полный линейный откат за *O(nlog*(n))* проходов

4. Откат по зацикленному дереву

Лесом называют граф без замкнутых маршрутов; *деревом* называют связный лес; любой лес есть объединение деревьев. *Корневым деревом* называют дерево с выделенной вершиной – корнем дерева. *Выходящим деревом* называется корневое дерево, в котором

любая вершина достижима из корня. Соответственно, *входящим деревом* называется корневое дерево, в котором из любой вершины достижим корень. *Суграфом* называют подграф графа с тем же множеством вершин. Суграф, являющийся деревом, *остовом*. *Хордой* называют дугу графа, не принадлежащую выделенному в графе остову.

Зацикленным деревом будем называть граф T^*, который получается из выходящего дерева T добавлением хорд, ведущих из листьев в корень r. Стартовой вершиной будем считать корень дерева. Число вершин зацикленного дерева будем обозначать n. *Развилкой* будем называть вершину дерева с полустепенью исхода больше 1. Листовые вершины, очевидно, не являются развилками.

Робот на зацикленном дереве использует не только внешнюю функцию *Traverse*, но и функцию *Next*. Будем говорить, что робот выполняет *откат по дереву*, если он для любого выходящего дерева T останавливается на зацикленном дереве T^* и до остановки терминирует все вершины дерева в порядке обратном их естественной частичной упорядоченности, то есть, от листьев к корню. *Проходом* робота по дереву будем называть его перемещение от корня до листа и далее по хорде снова в корень, или перемещение от корня до остановки. Сложность отката по дереву мы будем измерять в числе *проходов*. При каждом проходе, кроме, быть может, последнего, робот проходит простой контур, состоящий из простого пути от корня до листа и возвращающей в корень хорды. Длина такого контура не превосходит n и поэтому длина пройденного маршрута ограничена сверху числом проходов, умноженным на n.

Теорема 8: Существует конечный робот *R8*, который решает проблему отката по дереву за число проходов $O(n\log^*(n))$.

Формальное описание робота *R8* приведено на рис.15, иллюстрация – на рис.14. Робот *R8* является модификацией робота *R7*, которая заключается в следующем. Прежде всего, робот выделяет в дереве один контур (линейный подграф) P_i, который будем называть *активным контуром*, его дуги помечаются меткой *active* и называются *активными дугами*. Кроме того, для каждой пройденной вершины v первая исходящая дуга v-цикла помечается меткой *first*. Самый первый активный контур будет состоять из таких первых дуг – это контур P_1.

На активном контуре P_i робот работает аналогично роботу *R7*. Для этого вместо внешней функции *Traverse* используется функция прохода по активному контуру *Traverse_active*. Откат выполняется не до стартовой вершины (корень дерева), а до ближайшей к листу развилки u. Тогда вместо того, чтобы терминировать вершину u, робот проверяет, не является ли исходящая из u активная дуга последней в u-цикле. Если да, то вершина терминируется и откат продолжается. В противном случае, активной становится следующая дуга e. Активный контур после вершины u продолжается по дуге e, а далее по первым исходящим дугам $e\beta\gamma$, $e\beta\gamma\beta\gamma$, $e\beta\gamma\beta\gamma\beta\gamma$,… до новой листовой вершины и заканчивается хордой. Контур P_i заменяется на контур P_{i+1}; эти контуры имеют общий простой $[r,u]$-путь P^1_{i+1}. Таким образом, робот перебирает активные контуры $P_1,P_2,\ldots$ – все пути от корня до листовых вершин, реализуя тем самым поиск по дереву в глубину и останавливается, когда терминируется корень дерева.

Смена активного контура в развилке u прерывает этап вычитания единицы в последнем (по вложенности) диапазоне, концом которого является u, и продолжает этап прибавления единицы в первом (самом внешнем) диапазоне. Все диапазоны, кроме первого, располагаются в конечном отрезке нумерованных вершинах активного пути – в вершинах последнего нулевого номера внешнего диапазона. Прибавление единицы сдвигает этот

конечный отрезок до тех пор, пока остаются ненумерованные вершина в новом активном пути. После этого происходит возврат к этапу вычитания единицы в последнем диапазоне.

Теперь оценим число проходов робота. Для этого заметим, что робот *R8* отличается от робота *R7* следующим. Прибавление и вычитание единицы в первом диапазоне выполняется не последовательно (сначала прибавление, а потом вычитание), а поочередно. Прибавление единицы выполняется до тех пор, пока на текущем активном пути остаются ненумерованные вершины. Далее выполняется вычитание единицы до тех пор, пока не происходит смена активного пути, после чего снова прибавляется единица. И так далее. Кроме того, прибавление единицы не затрагивает конечный отрезок нумерованных вершин (соответствующих разрядам номера *0* первого уровня), просто сдвигая его по активному пути. При смене активного пути прерывается работа во вложенных диапазонах, но она продолжается с прерванного места после завершения прибавления единицы в первом диапазоне. Понятно, что эти модификации сохраняют оценку числа проходов каждого уровня, которая остается равной *O(n)*. Поскольку диапазоны всех текущих уровней, по-прежнему, образуют строго вложенную структуру, число уровней, по-прежнему, ограничено *O(log*(n))*. Это дает ту же оценку *O(nlog*(n))* как для числа проходов на всех уровнях, так и для числа проходов, которое тратится на смену уровней, прежде всего, на поиск объемлющего диапазона при закрытии единичного диапазона (функция *R1<Traverse_filter>*). Тем самым, сохраняется та же оценка общего числа проходов *O(nlog*(n))*. □

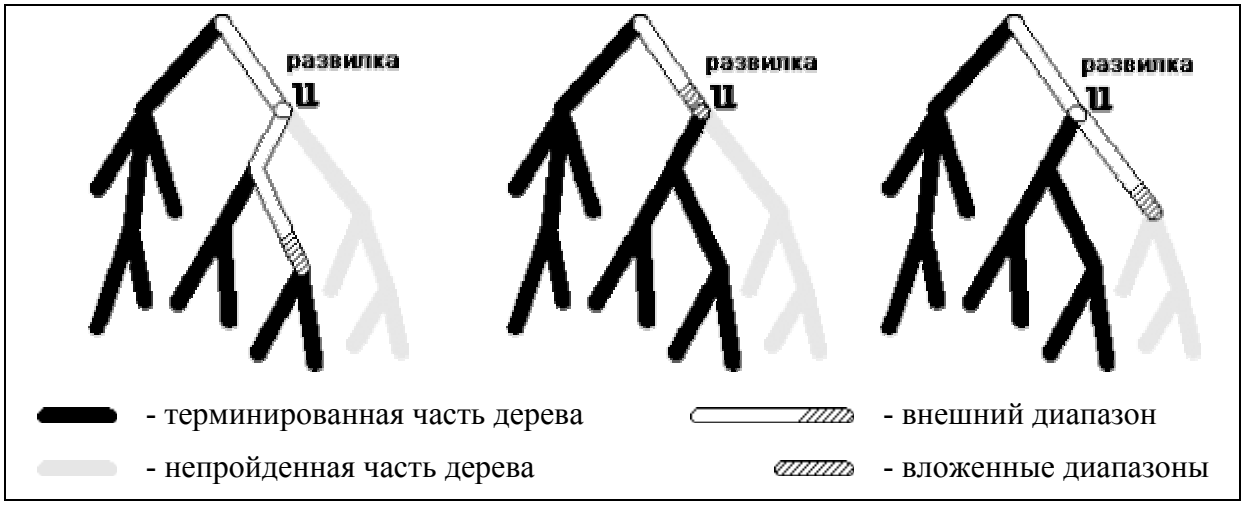

Рис.14: Иллюстрация к откату по дереву за *O(nlog*(n))* проходов

```
struct vertex { /* структура символа вершины */
    unsigned filter: 1 = 0; /* фильтрующее поле для функции Traverse_filter */
    /* поля робота R4 */
    unsigned terminal: 1 = 0; /* метка конечного символа */
    unsigned logstart: 1 = 0; /* метка начала диапазона */
    unsigned lastlogstart: 1 = 0; /* метка начала последнего диапазона */
    unsigned number: 1 = 0; /* признак разряда номера */
    unsigned low: 1 = 0; /* признак младшего разряда номера */
    unsigned bit: 1 = 0; /* содержимое разряда номера */
    unsigned shift: 1 = 0; /* отсюда продолжается прибавление единицы после
    сдвига */
    /* поля робота R1 */
```

```
    unsigned start: 1 = 0; /* стартовая вершина */
    unsigned candidate: 1 = 0;
};
struct vertex v; /* текущая вершина */

struct arc { /* структура символа дуги */
    unsigned first: 1 = 0; /* признак дуги vγ – первой дуги в v-цикле дуг */
    unsigned active: 1 = 0; /* признак активной дуги */
};
struct arc e; /* текущая дуга */

void Traverse_active() { /* функция Traverse для прохода по активному
контуру */
    if (!e.first) { e.first = 1; e.active = 1; }
    while (!e.active) Next(); Traverse();
}

void Traverse_filter() /* фильтрующая функция Traverse для робота R1 */
{ Traverse_active(); while (!v.filter) Traverse_active(); }

#define GO_TO_LOGSTART while (!v.logstart) Traverse_active();

void Traverse_range() { /* фильтрующая функция Traverse для робота R4 */
    Traverse_active(); if (v.terminal || v.start) GO_TO_LOGSTART
}

#define END_OF_NUMBER v.low || !v.number || v.filter ||
v.terminal
#define END_OF_LAST_NUMBER !v.number || v.filter || v.terminal

char * Minus_pass8() { /* вычитание единицы */
    unsigned new_range_start;
    if (Minus_pass<Traverse_range>() == "больше одной вершины") {
        /* открытие нового вложенного диапазона */
        if (!v.logstart) /* новое начало вложенного диапазона */
        { new_range_start = 1; v.logstart = 1; v.filter = 1; }
        else new_range_start = 0; /* старое начало вложенного диапазона */
        do { /* установка начального символа в диапазоне */
            v.number = 0; v.low = 0; v.bit = 0; v.shift = 0;
            Traverse_active();
        } while (!v.terminal && !v.start);
        GO_TO_LOGSTART
        if (new_range_start) /* переход на новое начало вложенного диапазона
        */
            { v.logstart = 0; GO_TO_LOGSTART }
        return "больше одной вершины";
    }
    else return "одна вершина"
}

void R8() {
```

```
    v.start = 1; v.logstart = 1;
    while (1) {
        /* этап прибавления единицы в диапазоне */
        while (Plus_pass<Traverse_range>()
        == "остались непронумерованные вершины") ;
        while (Minus_pass8() == "одна вершина") {
            /* изменение активной дуги */
            while (!e.active) Next(); e.active = 0; Next();
            e.active = 1;
            if (!e.first) { /* новая активная дуга */
                while (!e.first) Next();
                /* переход на первый диапазон */
                v.logstart = 0; v.lastlogstart = 1;
                while (!v.start) Traverse_active();
                v.logstart = 1;
                /* этап прибавления единицы в первом диапазоне */
                while (Plus_pass<Traverse_range>()
                == "остались непронумерованные вершины") ;
                /* переход на последний диапазон */
                v.logstart = 0;
                while (!lastlogstart) Traverse_active();
                v.logstart = 1; v.lastlogstart = 0;
            }
            else { /* все исходящие дуги уже были активными */
                v.terminal = 1; /* терминация вершины */
                v.logstart = 0; v.filter = 0; /* удаление единичного
                диапазона */
                if (v.start) return; /* стартовая вершина терминирована */
                R1<Traverse_filter>(); /* поиск начала объемлющего
                диапазона */
                v.logstart = 1; /* переход на объемлющий диапазон */
        } /* в диапазоне больше одной вершины – переход на этап прибавления
        единицы */
    }
}
```

Рис.15: Робот *R8*. Откат по дереву за *O(nlog*(n))* проходов

5. Обход сильно-связного графа

Компонентом сильной связности (далее просто *компонентом*) называют максимальный сильно-связный подграф, то есть, сильно-связный подграф, не являющийся подграфом никакого другого сильно-связного подграфа. Для вершины *v* через *K(v)* будем обозначать компонент, которому она принадлежит. *Графом 1-го рода* будем называть граф с линейным порядком компонентов, в котором из каждого непоследнего компонента исходит ровно одна дуга и она ведет в следующий компонент (рис.16); эти дуги будем называть *связующими*.

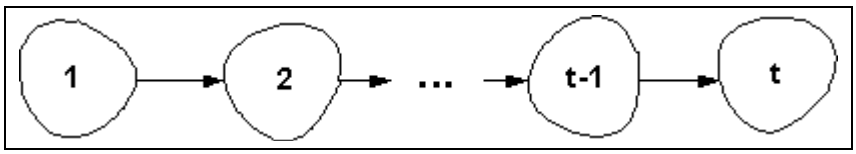

Рис.16: Граф 1-го рода

В графе 1-го рода с любой выделенной в первом компоненте стартовой вершиной всегда существует выходящий остов T_{out} с колрнем в стартовой вершине, очевидно, содержащий все связующие дуги, и входящий остовный лес F_{in} (лес входящих деревьев, являющийся суграфом), состоящий из входящих остовов компонентов, корни которых – концы связующих дуг (для первого компонента – стартовая вершина). Дуги выходящего дерева будем называть *out*-дугами, а дуги входящих деревьев – *in*-дугами. Корень входящего дерева леса F_{in} мы будем также называть корнем компонента K, остовом которого это дерево является, и обозначать $r(K)$.

Пройденным графом маршрута будем называть подграф, состоящий из дуг маршрута и инцидентных им вершин. Очевидно, что пройденный граф является графом 1-го рода.

Известные DFS- и BFS-роботы для обхода сильно-связного графа используют два алгоритма: 1) *алгоритм построения* выходящего остова T_{out} и входящего остовного леса F_{in}, 2) *алгоритм отката* по выходящему остову T_{out}. Дерево T_{out} обходится методом поиска в глубину (DFS) или в ширину (BFS).

При DFS-алгоритме [4] в дереве T_{out} выделяется один активный $[v_1,v_A]$-*out*-путь от корня (стартовой вершины) v_1 до нетерминированной вершины v_A. Робот выполняет *поиск непройденной дуги*: начиная с корня последнего компонента $r(K(v_A))$, двигается по активному пути до его конечной вершины v_A и проходит непройденную дугу e, исходящую из вершины v_A ($\alpha e = v_A$). Если дуга e оказалась хордой, робот выполняет *возвращение по хорде*: используя $[\beta e, r(K(v_A))]$-*in*-путь в лесе F_{in}, ведущий в корень последнего компонента, и $[r(K(v_A)),v_A]$-отрезок активного пути, возвращается в начало хорды, вершину v_A. В противном случае дуга e становится новой *out*-дугой, то есть, добавляется к дереву T_{out}, а ее конец βe становится корневой (и пока единственной) вершиной последнего компонента пройденного графа. Когда все дуги, исходящие из конечной вершины v_A активного пути, пройдены, начинает работать алгоритм отката, задача которого – терминировать вершину v_A, тем самым, сократив активный путь на одну дугу, и вернуться в корень $r(K(v_A))$ последнего компонента.

BFS-алгоритм [1,3] отличается тем, что при *поиске непройденной дуги* робот меняет сам активный путь. Для этого при каждой вершине v дерева T_{out} отмечается одна исходящая из нее активная дуга: робот делает активной следующую по v-циклу дугу и идет по ней. Активный $[v_1,v_A]$-*out*-путь – это путь из активных дуг, начинающийся в стартовой вершине.

Проблема отката отличается от рассмотренной в предыдущем разделе в трех отношениях. 1) Дерево T_{out} не задано с самого начала, а строится в процессе работа первого алгоритма. 2) Для зацикливания дерева вместо хорд, сразу возвращающих из листьев в корень, используются *in*-пути леса F_{in}. 3) Эти *in*-пути возвращают не в стартовую вершину, а в корень последнего компонента.

Если бы не было последнего (3-го) отличия, то есть, пройденный граф *в момент начала отката* всегда состоял бы только из одного компонента (был сильно связен), то для отката в DFS-алгоритме можно было бы предложить модификацию робота *R8*,

учитывающую первые два отличия. 1) Откат выполняется только на одну терминируемую вершину v_A, после чего снова работает алгоритм построения деревьев до тех пор, пока все дуги, исходящие из новой конечной вершины активного пути, не становятся пройденными. 2) Для возвращения в корень последнего компонента из конечной вершины v_A активного пути вместо хорды, ведущей в стартовую вершину v_1, используется $[v_A, v_1]$-*in*-путь.

Наличие нескольких компонентов $K_1,K_2,...,K_t$ в пройденном графе (3-е отличие) усложняет проблему отката. Если применять модификацию робота *R8*, то в каждом из компонентов K_i у нас может сформироваться своя система вложенных диапазонов. Впоследствии при проходе хорды, ведущей из конца активного пути (из компонента K_t) в непоследний компонент K_i, произойдет слияние нескольких компонентов $K_i,K_{i+1},...,K_t$. Робот окажется в корне компонента K_i и ему нужно либо слить диапазоны компонентов $K_i,K_{i+1},...,K_t$ в один диапазон, либо искать начало последнего диапазона – корень последнего компонента K_t. Оба эти решения могут существенно увеличить для отката оценку $O(n^2 log^*(n))$.

Число компонентов в пройденном графе и сама конфигурация выходящего дерева и леса входящих деревьев зависят не только от заданного (неупорядоченного) графа, но и от его упорядочивания, то есть, порядка дуг в *v*-циклах для всех вершин *v*. Этот порядок задается извне. Он определяет выбор роботом еще непройденной дуги среди множества непройденных дуг, исходящих из текущей вершины: робот выбирает первую в *v*-цикле непройденную дугу. Мы покажем, что для любого сильно связного графа существует такой порядок дуг, при котором в пройденном графе всегда будет только один компонент. Более строго: в пройденном графе каждый компонент, кроме, быть может, первого, состоит из одной вершины, инцидентной только связующим дугам. Это означает, что после прохода хорды все компоненты склеиваются в один. Тем более, у нас имеется только один компонент в момент начала отката. Мы покажем также, что такой порядок дуг может быть построен конечным роботом.

Для доказательства этого утверждения рассмотрим произвольный входящий остов T_{in} графа. Если дуги остова T_{in} сделать первыми в *v*-циклах всех вершин, кроме корня, а остальные исходящие дуги разместить в *v*-циклах в произвольном порядке, то мы получим искомый порядок дуг. Действительно, допустим обратное: в какой-то момент времени в пройденном графе имеется не первый компонент *K*, состоящий не из одной вершины. Тогда компонент *K* содержит некоторую дугу e_1 и, поскольку это не первый компонент, вершина αe не стартовая, то есть, из нее исходит дуга остова T_{in} – первая в αe-цикле дуга e^1_1. Эта дуга e^1_1 либо совпадает с дугой e_1, либо должна быть пройдена раньше нее и, следовательно, не может быть связующей дугой, то есть, она также принадлежит *K*. Поскольку компонент сильно связен, он содержит дугу e_2, исходящую из βe^1_1. Повторяя эти рассуждения далее, получаем бесконечную последовательность первых дуг e^1_1, e^1_2,..., принадлежащих компоненту *K*. Поскольку число различных дуг в графе конечно, мы имеем замкнутый маршрут из первых дуг, чего быть не может, поскольку это дуги дерева.

Такой порядок дуг легко задать, пометив дуги остова T_{in}. Это может сделать робот, который обходит граф по DFS-алгоритму [4] или BFS-алгоритму [1,3]. Каждый из этих роботов строит лес входящих деревьев F_{in}, помечая его дуги как *in*-дуги, и этот лес в конце обхода как раз и состоит из одного остова T_{in}.

Таким образом, можно построить робот, который дважды обходит граф, первый раз с оценкой $O(nm+n^2 loglogn)$, а второй раз с оценкой $O(nm+n^2 log^*(n))$.

6. Заключение

Общая проблема обхода неизвестного сильно-связного ориентированного графа конечным роботом до сих пор остается нерешенной. Наилучший результат – алгоритм однократного обхода с оценкой $O(nm+n^2 loglogn)$, предложенной автором в предыдущей статье [1], и алгоритм повторного обхода с оценкой $O(nm+n^2 log^*(n))$, предложенный в настоящей статье. Точная оценка для этой проблемы остается неизвестной (минимум из верхних оценок алгоритмов по всем возможным алгоритмам обхода). Более того, хотя представляется сомнительным, чтобы конечный робот мог обходить граф за $\Omega(nm)$, тем не менее, это также не доказано.

Литература

1. И.Б.Бурдонов. Обход неизвестного ориентированного графа конечным роботом. "Программирование", 2004, №4.
2. M.O. Rabin, Maze Threading Automata. An unpublished lecture presented at MIT and UC Berkeley, 1967.
3. И.Б.Бурдонов. Изучение поведения автоматов на графах. Дипломная работа, МГУ им. М.В.Ломоносова, механико-математический факультет, 1971 г.
4. Y. Afek and E. Gafni, Distributed Algorithms for Unidirectional Networks, SIAM J. Comput., Vol. 23, No. 6, 1994, pp. 1152-1178.