Слайд 1. Исследование ориентированного графа коллективом двигающихся автоматов.

Слайд 2. Зачем это нужно? (Практическое значение исследования графа автоматами).
Какие графы исследуются и для чего?
Вот здесь перечислены важнейшие.

Теперь почему граф исследуют автоматы?

Во-первых, хочется, чтобы исследование выполнялось автоматически, т.е. с помощью некоторого алгоритма.

Формализацией алгоритма является машина Тьюринга, т.е. автомат, ползающий по ленте.

При исследовании графа автомат будет ползать по графу, это я формализую потом.

На практике, с одной стороны, нужно, чтобы автомат требовал поменьше памяти, а, с другой стороны, обычно размеры, то есть число вершин и рёбер, графа как-то ограничены.

Поэтому мы будем рассматривать и такие автоматы, которые могут быть не конечными на рассматриваемом классе графов, но конечны на любом подклассе графов ограниченного размера.

Автомат, который конечен на всём классе графов, я буду называть *роботом*.

Если автомат не конечный, но граф не помещается в его память, то такой автомат будем называть *полуроботом*. По сути, это то же самое, что локальный алгоритм на графе.

А если граф помещается в память автомата, то это *неограниченный автомат*.

Слайд 3. Обход графа

Обход графа – это построение и проход автоматом маршрута, содержащего все вершины и ребра графа.

Иногда имеет смысл говорить о покрытии ребер графа не одним маршрутом, а набором маршрутов: либо из-за недостаточной связности графа, либо для ускорения времени обхода, если эти маршруты проходятся параллельно несколькими автомтами.

Обычно обход должен начинаться с выделенной начальной вершины графа. Я буду для краткости называть её корнем.

Зачем нужен обход графа при исследовании графа? Я скажу только о двух случаях.

1) Обход лабиринта.
То, что из чего возникло понятие обхода графа, да и сама теория графов. Всё началось с задачи о семи мостах Кёнигсберга, которую решил Эйлер.
Ну, лабиринт является моделью очень многих вещей.

2) Тестирование.
По роду наших занятий для нас наиболее интересно именно это приложение.
Немного упрощая, можно сказать, что в этом случае граф – это граф переходов автоматной или автомато-подобной модели тестируемой системы.
Её начальное состояние – это корень.
Автомат на графе – это тестирующая система.
Много автоматов на графе – это распределённая тестирующая система.
Автомат проходит по дуге графа из текущей вершины – это значит, что в текущем состоянии тестируемой системы выполняется тестовое воздействие на систему, наблюдается результат этого воздействия и наблюдается изменённое состояние системы. Такой граф, как правило, ориентированный.

Слайд 4. Упорядоченный граф

Когда выполняется обход графа, на каждом шаге автомат должен указать ребро, инцидентное текущей вершине, чтобы пройти по этому ребру.

Для этого используется нумерация таких ребер, начиная с 1.

В ориентированном графе ребро проходится только в направлении его ориентации, поэтому оно имеет номер только в вершине, из которой выходит.

В неориентированном графе ребро может проходиться в обоих направлениях, поэтому оно имеет два номера: по одному в каждой из инцидентных ему вершин.

Такой граф называется *упорядоченным*.

В любом случае автомат указывает номер ребра в текущей вершине.

Слайд 5. Структура во время обхода
Посмотрим, какая структура возникает на графе при его обходе.

Непройденные рёбра покрасим в белый цвет.

Если граф неориентированный, мы сами ориентируем пройденное ребро в том направлении, по которому первый раз его прошли.

Если мы в вершине ещё не были, она белая.
Если были, но прошли не по всем инцидентным ей рёбрам, то серая.
А если прошли по всем её рёбрам, то чёрная.

Пройденный граф – подграф, порождённый пройденными рёбрами.

Прямое ребро – это ребро, по которому мы первый раз в пришли в какую-то вершину.
Они порождают прямое дерево – остов пройденного графа, ориентированный от корня.
Остальные пройденные рёбра – *хорды* прямого дерева, они обозначены чёрным пунктиром.

Серое дерево – это поддерево прямого дерева, содержащее корень и все серые вершины, все его листья тоже серые.

Рёбра серого дерева серые, остальные прямые рёбра чёрные.

Слайд 6. Неизвестный граф

Если граф известен, то можно вычислить его обход минимальной длины, а потом пройти по этому маршруту.

Для неизвестного графа так сделать уже нельзя.
Что значит, что граф неизвестен? Неизвестен заранее, до прохода по всем его рёбрам.

На каждом шаге, выбирая очередное ребро, мы знаем только часть графа. Что мы знаем?

Во-первых, пройденный граф, во-вторых, что-то о белых рёбрах, инцидентных серым вершинам. Что и когда мы можем узнавать об этих рёбрах?

Автомат будем называть *неизбыточным*, если он «узнаёт» степень вершины, когда первый раз в неё попадает. Поскольку пройденные рёбра мы знаем, всегда известно и число белых рёбер, инцидентных вершине.

Автомат будем называть *свободным*, если он «узнаёт» о наличии или отсутствии белого ребра только при попытке пройти по нему. Если ребро есть, идём по нему, а если нет, остаёмся на месте.

Свободный автомат работает немного дольше, потому что может оказаться, что ему нужно лишний раз придти в вершину, чтобы убедиться, что она чёрная.

Поскольку автомат не знает, куда ведут белые рёбра, всё равно какое белое ребро он выберет в текущей серой вершине. Можно считать, что он проходит белые рёбра из серой вершины в порядке возрастания их номеров.

В ориентированном графе это означает, что если из вершины выходят k пройденных рёбер, то они имеют номера от 1 до k. В неориентированном графе это не обязательно так, потому что какое-то ребро ab может быть пройдено первый раз не из вершины a, а из вершины b.

Для прохода по ребру ab автомат называет номер ребра в a.

Если граф неориентированный и такое ребро есть, то после прохода по нему автомат узнаёт номер ребра в b.

Слайд 7. Автомат на графе.
Исследование неизвестного графа похоже на машину Тьюринга.

Вершина графа – это ячейка ленты, а движение по ребру – это движение по ленте влево или вправо.

То есть лента обычной машины Тьюринга – это граф специального вида.

Но тут есть проблема для робота.

Дело в том, что номер ребра является выходным символом автомата.

Для того, чтобы автомат был конечным, в неориентированном графе степень вершины, а в ориентированном графе полустепень исхода вершины должны быть ограничены сверху.

Как обойти это ограничение?

Слайд 8. Вершина графа преобразуется в цепочку вершин.
Для этого можно преобразовать вершину ориентированного графа в цепочку вершин, каждая из которых имеет полустепень исхода не больше 2.

- Для неориентированного графа преобразование аналогично.

- Для машины Тьюринга это означает, что ячейка ленты превращается в цепочку ячеек конечной, но неограниченный длины, по которой можно перемещаться в вертикальном направлении. Получается что-то вроде ленты переменной ширины.

Слайд 9. Неограниченный автомат.

Если граф помещается в память автомата, то по мере обхода автомат может запоминать в своей памяти весь пройденный граф.

Для этого достаточно нумеровать вершины в порядке их обнаружения, т.е. когда по белому ребру приходим в белую вершину, и записывать в вершине её номер.

Тогда при проходе любого ребра мы можем узнать номера обеих инцидентных ему вершин.

На самом деле номер вершины – это единственное, что нужно хранить в самой вершине.

Всё остальное можно хранить в памяти автомата, индексируя номером вершины.

Если вершины пронумерованы заранее и их номера в них записаны, то такой граф будем называть *нумерованным*.

Тогда автомат будет только читать из вершины её номер и не будет в неё писать.

Слайд 10. Что-то вроде плана доклада.
Прежде всего, граф может быть ориентированным или неориентированным. От этого существенно зависят как алгоритмы обхода, так и оценки их сложности.

Во-вторых, граф известный или неизвестный.
Известный граф нам нужен для того, чтобы оценить минимальную длину обхода.

А для неизвестного графа мы как раз и будем рассматривать различные алгоритмы обхода.
Они могут быть неизбыточными или свободными, в зависимости от способа общения с графом.
А по размеру их памяти они делятся на роботы, полароботы и неограниченные автоматы.

Также бывает интересно различать первый обход графа и повторный, при котором можно использовать пометки на графе, оставленные автоматом при первом проходе.

Далее несколько автоматов на графе. Зачем нужно несколько автоматов?

Это либо решает проблему времени, уменьшая время обхода, либо решает проблему ограниченности памяти компьютера за счёт распределённого обхода.

Под конец я расскажу об обходе недетерминированного графа.

Слайд 11. Неориентированный граф. Длина обхода (1)
Сначала рассмотрим неориентированный граф.

Обход такого графа существует тогда и только тогда, когда граф связен.

Минимальная длина обхода не более, чем *2m-1*, где *m* число рёбер.
Это обход известного графа неограниченным автоматом.

Действительно, если каждое ребро удвоить, то граф превращается в эйлеров граф (степени всех вершин чётные). Его эйлеров цикл соответствует обходу исходного графа длиной *2m*. Но последнее ребро можно не проходить, поскольку это был бы его повторный проход.

Пример достижимости оценки внизу слайда.

Слайд 12. Неориентированный граф. Длина обхода (2)
Пусть *S* множество ребёр, которые нужно удвоить, чтобы получился эйлеров граф. Пусть *r* – минимальное число рёбер в *S*.

Тогда минимальная длина обхода равна *m+r*, если корень не инцидентен рёбрам из *S*, или на единицу меньше в противном случае. В первом случае проходится эйлеров цикл, а во втором – эйлеров путь с началом в корне.

На слайде справа маленькие примеры этих двух случаев.

Как найти *S* смотри Слайд 47.

В *S* нет циклов, поскольку любой цикл можно удалить без ущерба для чётности всех вершин. Поэтому, если *n* – число вершин в графе, то минимальная длина обхода равна *m+n-2*.

Пример достижимости этой оценки внизу слайда.

Слайд 13. Неизвестный граф. Алгоритм Тэрри. Свободный робот.

Для обхода неизвестных неориентированных графов имеется алгоритм Тэрри.

Он реализуется свободным роботом, если степень вершины ограничена, или полуроботом для любых графов.

Это DFS-алгоритм, т.е. обход в глубину.

В DFS-алгоритме серое дерево всегда состоит только из одного серого пути.

Каждое ребро автомат проходит 2 раза.

Как работает алгоритм Тэрри? Два основных правила:

Первое: после прохода по хорде немедленно возвращаемся по ней назад.

Второе: если идти некуда (т.е. текущая вершина чёрная), откат по серому пути до серой вершины. Если это невозможно, то есть корень чёрный, то конец обхода.

Можно заметить, что петли тоже проходятся по два раза. Этого можно было бы не делать, если распознавать петли.

Вторая оптимизация такая: откат по серому пути, все вершины которого, начиная с корня, стали чёрными, можно не делать, так как в этом случае все вершины и рёбра уже чёрные.

Формальное описание алгоритма Тэрри смотри: Слайд 48.

Слайд 14. Оптимизация алгоритма Тэрри.
Вот здесь в таблице сведены две оптимизации алгоритма, о которых я говорил, в зависимости от типа автомата.

Эту таблицу я привожу для того, чтобы продемонстрировать различие между типами автоматов, а ещё для того, чтобы показать полезность повторных обходов графа.

Если автомат полуробот, он легко распознаёт петли, запоминая номер вершины, из которой выходит по ребру, и сравнивая с номером вершины, куда попал. Они совпадают только для петли.

Робот так не может.
Но он может распознать петлю, если она имеет не два разных номера в вершине, а один. Когда проходим по белому ребру, которое ведёт в другую вершину, то в ней это ребро ещё белое. А если мы видим, что ребро чёрное, значит прошли петлю.

Другой вариант для робота: во время обхода размечать граф, т.е. оставлять пометки в его вершинах, так, чтобы при последующем обходе распознавать петли. Мы знаем, как это сделать за два обхода, т.е. петли будут распознаваться, начиная с третьего обхода.

Если все вершины серого пути чёрные, то это путь по рёбрам с максимальными номерами. Его может распознать неизбыточный робот. Он может разметить такой путь, ставя пометку в корне и в конце каждого ребра, которое ведёт из вершины с пометкой и имеет максимальный номер. Как только вершина с пометкой становится чёрной, конец обхода.

Свободный автомат, даже если он неограниченный, вообще не может быть уверен, что вершина серого пути чёрная, пока не попробовал выйти из неё по белому ребру, а для этого ему нужно сделать откат по серому пути до этой вершины. Поэтому свободный автомат такую оптимизацию сделать не может.

Зато свободный робот при первом обходе может так разметить граф, что при повторном обходе не будет отката по всей самой правой ветви прямого дерева от листа до корня.

Оптимизации с повторным обходом по петле и самому правому пути дерева смотри: Слайд 49.

Слайд 15.BFS-алгоритм. Свободный робот.
Теперь посмотрим BFS-алгоритм, т.е. алгоритм обхода в ширину.

Он тоже реализуется свободным роботом, если степень вершины ограничена, или полуроботом для любых графов.

Чем он отличается от алгоритма Тэрри?

Он проходит хорды по одному разу.

Пройдя по хорде, робот не возвращается, а снова пытается идти по белым рёбрам.

Из-за этого в сером дереве может вырасти новая ветвь.

Когда текущая вершина становится чёрной, ищем серую вершину. Двигаться будем только по серому дереву.

Если мы в листе дерева, то сначала откатывается по дереву, т.е. идём по нему *вниз* к корню, либо до серой вершины, либо до развилки.
Рёбра, которые проходим, делаем чёрными.
После развилки идём *вверх* до серой вершины.

Если текущая чёрная вершина не листовая, сразу идём *вверх* до серой вершины.

Конец обхода, когда в сером дереве остался один корень и он почернел.

Длина обхода $m+\boldsymbol{O}(n^2)$.
Эта оценка достижима.

См. доказательство Слайд 50.

Слайд 16. Свободный «жадный» алгоритм Дейкстра).
Отличие этого алгоритма от обхода в ширину в том, как делается поиск серой вершины.
Вместо движения по серому дереву вниз, а потом вверх, автомат по всему пройденному графу сначала вычисляет кратчайший путь до серой вершины, а потом идёт по этому пути.

Длина обхода $m+O(n^2)$.

Эта оценка дана для обхода неограниченным автоматом.

Робот и полуробот не могут вычислять кратчайший путь, не перемещаясь по хордам, поэтому для них такой алгоритм не имеет смысла.

Слайд 17. Обход неориентированного графа. СВОДКА.
На этом слайде в таблице представлена сводка результатов по обходу неориентированного графа, о которых я рассказывал, кроме оптимизаций для алгоритма Тэрри.
Что хочется отметить?

Понятно, что обход неизвестного графа длиннее, чем известного.
Однако в терминах числа рёбер разница всего лишь на единицу для свободного автомата при первом проходе.

Если задано ещё и число вершин, то алгоритм поиска в ширину и жадный алгоритм лучше алгоритма Тэрри при достаточно большом числе рёбер по сравнению с числом вершин: вместо второго m у нас будет $O(n^2)$.

Для жадного алгоритма мы не знаем его точной оценки сверху.
Может быть, там должно стоять $O(n)$ вместо $O(n^2)$, но это только гипотеза. Если кто знает, скажите

Все алгоритмы могут быть реализованы роботом, но для жадного алгоритма это не имеет смысла.

Слайд 18. Ориентированный граф.
Теперь перейдём к ориентированным графам. Ориентированное ребро я буду называть, как это принято, дугой. Дугу можно проходить только в направлении её ориентации.

Для существования обхода требуется сильно-связность графа.

Более точно: граф должен быть 1-го рода – цепочка компонентов сильной связности и разделяющих дуг, начальная вершина – в первом компоненте.

- Если граф неизвестен, то он должен быть графом 2-го рода: простой путь, ведущий в последний компонент сильной связности.

- Пройденный подграф является графом 1-го рода, а в конце обхода, конечно, совпадает со всем графом.

- И ещё: в ориентированном графе вершина чёрная, если пройдены все выходящие из неё дуги (не обязательно пройдены все входящие).

Слайд 19. Минимальная длина обхода.
Минимальный обход имеет длину $O(nm)$, где n – число вершин, m – число дуг.
Пример достижимости оценки.
Здесь из вершины n в начальную вершину ведёт много кратных дуг.

Слайд 20. Ограниченная полустепень исхода вершин.
На этом слайде пример достижимости оценки для ограниченной полустепени исхода.

Это ограничение может быть меньше, чем число кратных дуг на предыдущем слайде.

Поэтому несколько последних вершин образуют сбалансированное по высоте дерево с ограниченной полустепенью исхода, из листьев которого дуги ведут уже в начальную вершину.

Вообще же при ограниченной полустепени исхода оценка становится $O(n^2)$, т.к. $m \le sn$.

Слайд 21. Неизвестный граф. DFS-автомат. Полуробот.
DFS-алгоритм может применяться и в случае ориентированного графа. Но тут есть проблема с откатом, потому что мы не можем вернуться по дуге в обратном направлении. Вместо этого мы должны пройти некоторый ориентированный путь из конца дуги в её начало.

Для этого делаются две вещи.

Во-первых, серый путь отмечается: каждая его дуга отмечена в её начале.

Во-вторых, строится лес обратных деревьев: по одному на каждый компонент сильной связности пройденного графа.

В каждом компоненте обратное дерево является остовом, ориентированным к корню компонента.
В этот корень входит разделяющая дуга из предыдущего компонента, в первом компоненте это корень графа.

Обратная дуга отмечена в её начале. На слайде обратные дуги красные.

- Для отката у нас всегда есть цикл, состоящий из двух путей: путь по обратным дугам до корня компонента и серый путь от корня до текущей вершины. Вот по этому циклу мы и ищем нужную вершину.

При откате по хорде ищем начало хорды.
По ходу дела корректируем лес обратных деревьев.

- При откате по дереву ищем последнюю серую вершину на сером пути.

Затем двигаемся дальше по серому пути, перекрашивая все рёбра в чёрные, а потом снова идём по циклу до найденной вершины.

Но это если автомат знает, куда ему идти, т.е. знает начало хорды или последнюю серую вершину на сером пути.

Если автомат полуробот, он может это определить, так что на слайде приведена оценка длины обхода полуроботом.

Как она получается? Каждый откат требует прохода двух путей. Поскольку длина пути меньше числа вершин n, а откатов не больше числа дуг m, получаем оценку $O(nm)$.

Слайд 22. BFS-полуробот. Жадный неограниченный.
В BFS-алгоритме серое дерево состоит не из одного пути, оно может сильно ветвиться.

После прохода по хорде полуробот двигается не в начало хорды, а в такую серую вершину, что путь по серому дереву от корня компонента до этой вершины не содержит других серых вершин.

Длина обхода $O(mn)$. Эта оценка достижима.
Память автомата $O(\log n)$.

Жадный неограниченный автомат не использует ни серое дерево, ни прямое или обратное деревья.

Оценка длины обхода для всех треёх алгоритмов такая же, как для минимального обхода. Но это в худшем случае.

А в других случаях разница может оказаться весьма существенной.

Есть пример, где минимальная длина имеет порядок числа дуг, а DFS- и BFS-алгоритмы проходят $\Omega(nm)$ дуг. Смотри Слайд 51.

Однако для жадного алгоритма это неизвестно.
Есть гипотеза, что отличие имеет порядок всего лишь $O(n)$.

Если автомат неограниченный, то, очевидно, при любом алгоритме обхода повторный обход можно сделать как минимальный: на первом обходе собирается вся информация о графе, потом по ней вычисляется минимальный обход и он проходится автоматом.

Слайд 23. Робот. Проблема отката.
Задачу обхода ориентированного графа роботом впервые поставил Рабин в устной лекции в 1967 году.

Со временем выяснилось, что основная проблема для робота – это проблема отката, то есть возврата из конца дуги в её начало.

Для этого используется цикл дуг, в который входит эта дуга.

Только проходя такой цикл, робот может скорректировать обратные деревья при откате по хорде и удалить из серого дерева путь от ставшего чёрным листа до серой вершины или развилки дерева.

Но как найти начальную вершину дуги?

Робот не может просто пройти по циклу в эту вершину, поскольку она никак специально не помечена.

- Самый простой способ отката предполагает проход по циклу дуг столько раз, какова длина цикла.

В 1970-ом году я придумал робот, основанный на DFS-алгоритме и использующий такой простой откат.

Суммарно на откаты дополнительно тратилось n^3 проходов дуг.

В следующем году я переделал робот на BFS-алгоритм, что дало суммарное время отката $n^2 logn$.

Слайд 24. Робот. Логарифмический откат.
В 1993-ем году появилась работа Афека и Гафни, в которой для DFS-алгоритма применялся логарифмический откат.

Все вершины в цикле помечаются, а потом, с учётом четности их числа пометки удаляются через одну и вычисляется чётность числа оставшихся пометок.

Это дало такую же оценку $n^2 logn$.

В 2003-ем году я «скрестил» BFS-алгоритм с логарифмическим откатом и получилась оценка $n^2 loglogn$.

Время отката можно сократить, если синими могут становиться не все вершины цикла, а только *значимые*.
Что такое значимая вершина?
Единственное требование: искомая вершина значимая.

Откат по хорде делается тогда, когда мы первый раз по ней прошли, поэтому значимыми можно считать только такие вершины на цикле, которые являются началом однократно пройденной дуги цикла.
Это даёт суммарную оценку откатов по хордам порядка nm.

Так что второе слагаемое в оценке длины обхода – это только суммарная оценка откатов по дереву.
При откате по дереву мы ищем серую вершину или развилку дерева, вот они и будут считаться значимыми.

Слайд 25. Робот. Откат по дереву.
Формально задача отката по дереву звучит так.

Пусть в дереве, ориентированном от корня, добавлены дуги, ведущие из листьев в корень.
Получается сильно связный граф.
Нужно пометить все вершины дерева в порядке от листьев к корню.

Слайд 26. Робот. Цифровой откат.

В том же 2003-ем году я придумал способ отката по дереву, который назвал цифровым. Я не буду его рассказывать, это заняло бы слишком много времени. А его оценка – это $n^2 log^* n$, где log^* – это число логарифмирований, которые превращают аргумент в число меньше 2. Я не нашёл специального названия для такой функции и обозначил как логарифм со звёздочкой.

К сожалению, это не значит, что теперь второе слагаемое в оценке длины обхода становится таким же маленьким. Дело в том, что пройденный граф, как я уже говорил, – это граф 1-го рода, поэтому в нём не один остов, ориентированный к корню, а лес деревьев по одному в каждом компоненте сильной связности. Из-за этого длина обхода в общем случае не меняет свой порядок при цифровом откате.

Логарифм со звёздочкой при обходе получается только тогда, когда граф специально упорядочить, то есть не при произвольной нумерации дуг, исходящих из вершины. Зато такое упорядочивание можно сделать при первом обходе роботом, так что оценка длины повторного обхода уже будет со звёздочкой.

Слайд 27. Два робота.

Сейчас мы постепенно перейдём к коллективу автоматов. Для начала посмотрим, как могут обходить граф два робота.

Предполагается, что оба робота могут читать и писать в вершины графа, а, кроме того, могут «переговариваться» друг с другом, то есть обмениваться сообщениями по независимой от графа сети связи.

Идея алгоритма в том, что роботы идут по графу синхронно, но с расстоянием в одну дугу. Поэтому, когда первый робот узнаёт, что нужно делать откат, т.е. возвращаться на одну дугу назад, в начале этой дуги как раз стоит второй робот. Достаточно сообщить ему, чтобы он пометил вершину, в которой находится, и идти до помеченной вершины. Подробнее см. Слайд 52.

В результате время обхода становится по порядку минимальным $O(nm)$.

Слайд 28. Коллектив свободных неограниченных автоматов. Обход известного графа.

Теперь рассмотрим общую задачу обхода графа коллективом свободных неограниченных автоматов.

Автоматы генерируются в корне графа, а дальше они могут двигаться по дугам и обмениваться между собой сообщениями.
Автомат, который генерирует автоматы, будем называть генератором, а остальные – движками.
Число генерируемых автоматов не ограничено.
Требуется, чтобы по каждой дуге прошёл хотя бы один автомат.
Это значит, что мы покрываем дуги графа не одним, а несколькими маршрутами.
Мы не требуем сильной связности графа, будем покрывать дуги графа достижимости.
Автоматы неограниченные.

Поскольку, кроме прохода дуг, у нас есть ещё обмен сообщениями, оценивать будем время обхода, считая, что пересылка сообщения и проход по дуге занимают не больше 1 такта.

Сначала рассмотрим случай известного графа.
Автоматы не пишут в вершины, и не читают из них.
Граф известен генератору и все нужные пометки он может делать в своей памяти.
Генератор сообщает каждому движку, куда ему идти, в виде последовательности номеров дуг.

Если за 1 такт генерируется 1 движок, то время обхода на самом плохом графе равно m, а на самом хорошем графе равно корень из m.

Если за 1 такт генератор может сгенерировать неограниченное, но конечное, число автоматов, то время обхода на самом плохом графе равно $D+1$, а на самом хорошем графе равно D.

Слайд 29. Коллектив свободных неограниченных автоматов. Обход неизвестного графа.

Если граф неизвестен, то в памяти генератора хранится только его пройденный подграф.

Когда движок попадает в новую вершину, ему надо знать о ней только одно: её номер.

Для простоты будем считать, что граф нумерованный.

Генератор на каждом такте сообщает движку, по какой дуге ему сейчас идти, а в ответ движок сообщает номер вершины в конце этой дуги и ждёт дальнейших указаний.

В первом случае оценка не меняется по порядку.

Генератор направляет движок по белой дуге, а если её нет, то по любой дуге серого дерева. Вместо отката движок останавливается.

Хорошо бы посмотреть отличие от известного графа.

Во втором случае оценка меняется, теперь это D^2 или m, если m меньше.

Легко доказать, что это наилучшая по порядку оценка для неизвестного графа, т.е. алгоритм не может работать быстрее.

Как он работает?

На каждом такте генерируем столько движков, сколько не хватает до числа серых вершин, или ничего не генерируем, если движков достаточно. Три дополнительных слайда:

Слайд 30. Обход ориентированного графа. СВОДКА

На этом слайде сводка результатов по ориентированным графам.

Что хочется отметить?

Во-первых, если автоматов много, это существенно уменьшает время обхода.

Во-вторых, в отличие от неориентированного случая здесь разница между роботом и полуроботом весьма существенная.

Правда, никто не доказал, что оценка для робота не может быть улучшена до минимальной оценки $O(nm)$.

Знаком вопроса отмечены остальные нерешённые задачи.

Слайд 31. Практический пример.
А сейчас один практический пример работы, которая подтолкнула нас к исследованию поведения коллектива автоматов на графе.
Наши коллеги тестировали модели цифровой аппаратуры с помощью сети компьютеров.
В примере на слайде показан размер графа и время его тестирования с различным числом компьютеров.

Здесь не было генератора, движки статически распределялись по компьютерам и каждый из них общался только с соседями в соответствии с топологией связей. Использовался DFS-алгоритм обхода графа.

Это распределённый обход графа с целью уменьшения времени обхода. Ну, как это время уменьшалось, видно в таблице.

В этом примере со временем всё хорошо, но граф, хотя он и большой, всё же помещался в памяти одного компьютера, т.е. в памяти автомата.

Слайд 32. Проблема слишком больших графов.
А что делать, если граф не помещается в память автомата?

Когда мы рассматривали обход графа коллективом свободных неограниченных автоматов, выяснилось, что память вершин нам, в общем-то, не нужна, достаточно, чтобы граф был нумерованным.

Сохраняя эти предположения, будем считать, что, если граф не помещается в память одного автомата, то есть этот автомат полуробот, то граф всё же должен помещаться в суммарную память нескольких полуроботов, число которых неограничено.

На самом деле, это важно при тестировании.
В этом случае никакой памяти вершин нет, поскольку вершина – это состояние тестируемой системы.
Его можно наблюдать, но в него нельзя писать, можно только перейти в другое состояние в результате тестового воздействия.

Слайд 33. Автоматы-регуляторы.

Я расскажу алгоритм, при котором пройденный граф хранится в суммарной памяти автоматов, которые мы назвали регуляторами.

Регулятор не взаимодействует с графом и хранит информацию о локальном окружении одной вершины пройденного графа:

- номер вершины,
- адрес *родителя*, то есть регулятора начала входящей прямой дуги,
- для каждой выходящей прямой дуги – адрес *потомка*, то есть регулятора конца дуги.

Регулятор управляет перемещением движков через свою вершину.

Слайды и текст в старой версии смотри: Слайд 56- Слайд 60.

Слайд 34. Алгоритм обхода коллективом полуроботов (1).

1. **Откуда берутся регуляторы?**

Регулятор корня – он же генератор.

Для другой вершины регулятор создаёт движок, первым попавший в вершину (по белой дуге в белую вершину).

2. **Как движок узнаёт, что он первый?**

Для этого делается поиск регулятора по номеру вершины.

Регуляторы связаны в список по их адресам.

По списку посылается сообщение с номером вершины и адресом движка.

Если регулятор не найден, то движок в эту вершину попал первым.

3. **Как регулятор управляет движением движков?**

Движок спрашивает у регулятора текущей вершины, ***куда идти***?

Регулятор отвечает: ***иди по дуге*** номер такой-то.

Дуги перебираются в порядке возрастания номеров: 1,2,3, и так далее.

Если такой дуги не оказалось, движок сообщает об этом регулятору и тот теперь знает число выходящих дуг.

Далее регулятор перебирает номера по циклу.

Этого достаточно, чтобы обойти граф.

Слайд 35. Алгоритм обхода коллективом полуроботов (2).

4. Как узнать о конце обхода?

Как и BFS-алгоритме у нас есть прямое дерево и его серое поддерево.
Отличие лишь в том, что один автомат сам должен после хорды пытаться строить новую ветвь дерева, а когда автоматов много, то эти ветви создаются ими параллельно.

Вместо отката – сообщение ***конец***, посылаемое по дуге в обратном направлении, то есть из автомата в конце дуги регулятору начала дуги.
При откате по хорде отправитель – движок, прошедший по хорде.
При откате по прямой дуге отправитель – регулятор конца дуги.
Регулятор-получатель отмечает дугу как *законченную*.
Законченная дуга, по сути, то же, что чёрная дуга.

Как движок узнаёт, что он прошёл по хорде?
Это происходит, когда движок не первым пришёл в вершину, т.е. при опросе регуляторов регулятор найден.

Сам регулятор посылает ***конец*** родителю, когда все выходящие дуги стали законченными.
Если это регулятор корня, то конец обхода.

При такой работе, если не принять никаких дополнительных мер, то время обхода экспоненциально.
Это видно на примере. Дуга вниз имеет номер 1, а дуга направо номер 2. Над дугой направо написан номер движка в порядке генерации, который первым проходит эту дугу. Из этого примера видно, как вредно ходить по законченным дугам.

5. Как уменьшить время обхода?

Регулятору не надо посылать движки по законченным дугам. Они блокируются до конца обхода.

Слайд 36. Алгоритм обхода коллективом полуроботов (3).

6. **Как уменьшить число одновременно существующих движков?**

Нужно уничтожать движок, если он:
a) прошёл по хорде, или
b) прошёл по прямой дуге, а дальше ему некуда идти, потому что все выходящие дуги оказались законченными.

Тогда каждый движок проходит путь длиной $O(n)$, после чего выполняет опрос за время $O(n)$ и становится регулятором или уничтожается.
Значит, движок живёт $O(n)$ тактов.

Если за такт генерируется не более одного движка, число одновременно существующих движков $O(n)$.

Слайд 37. Алгоритм обхода коллективом полуроботов (4).

7. **Как уменьшить число генерируемых движков?**

Если не тормозить генерацию движков, то их будет нагенерировано столько, сколько длится обход.

Для торможения реализуется старт-стопный механизм перемещения движков по прямой дуге.
Регулятор направляет движок по прямой дуге только после получения от регулятора конца дуги сообщения ***запрос*** движка.
Дуга временно блокируется до получения следующего ***запроса***, который будет отправлен, когда движок пойдёт дальше.

Если все выходящие дуги заблокированы, то приходящий движок либо уничтожается, если все дуги закончены, либо ожидает разблокировки. Запрос не посылается, поэтому в вершине не более одного ждущего движка.

Если все выходящие из корня дуги блокированные, генератор приостанавливает генерацию движков.

Общее число движков будет равно $O(m)$.

Слайд 38. Оценки.
Для оценки будем считать, что проход по дуге и пересылка сообщения выполняются за 1 такт.
Также за 1 такт генерируется не более 1 движка.
Адрес автомата – это просто его номер.

Время обхода порядка $m+nD$.
Второе слагаемое возникает из-за опроса регуляторов: время одного опроса не больше числа регуляторов, которое не больше числа вершин n, а на прямом пути от корня до листа длиной не более D опросы регуляторов происходят строго последовательно в порядке от корня к листу.

В таблице приведены размеры сообщения и памяти полуробота.
И, для сравнения, те же размеры для неограниченных автоматов.

Мы видим, что размер сообщения остаётся тем же. Там два основных параметра: адрес автомата и номер дуги.

Если полустепень исхода вершины не ограничена, то выигрыша по памяти автомата не получается. Но этого и следовало ожидать, поскольку в этом случае локализация информации о графе не может дать выигрыша: полустепень исхода может достигать числа дуг в графе.
Если адрес автомата не используется повторно, то получается даже проигрыш, поскольку число автоматов тоже может достигать числа дуг в графе.

Для ограниченной полустепени исхода число дуг столько же по порядку, сколько вершин. Получается выигрыш в n раз.

Если граф получен с помощью преобразования вершины в цепочку вершин, когда получается полустепень исхода 2, число вершин преобразованного графа по порядку равно числу дуг исходного графа. И получается выигрыш в m раз.

Здесь мы имеем пример классического правила: если хотите сэкономить память, придётся пожертвовать временем.
Смотри Слайд 53.

Слайд 39. Справедливый недетерминизм.

Под конец я обещал рассказать про обход недетерминированного графа.

Упорядоченный граф будем называть *недетерминированным*, если одним номером дуги может быть помечена не одна, а несколько выходящих дуг.

Δ-дугой будем называть множество дуг с одним номером и одним началом.

Будем считать, что недетерминированный выбор дуги в Δ-дуге задаётся недетерминированной функцией выбора, которая для каждой Δ-дуги недетерминированным образом возвращает дугу из этой Δ-дуги.

Недетерминизм справедливый, если для любой Δ-дуги в любой бесконечной последовательности значений функции выбора каждая дуга из этой Δ-дуги встречается бесконечное число раз.

Гипотеза о справедливом недетерминизме графа эквивалентна гипотезе о глобальном тестировании: при бесконечном числе прогонов теста будут получены все возможные варианты поведения тестируемой системы.

Слайд 40. Справедливый недетерминизм. Коллектив полуроботов (1).

Пусть граф конечный и справедливо недетерминированный.
Будем рассматривать неизбыточные полуроботы

Нам будут нужны два предположения:

1) В дельта-дуге нет кратных дуг, то есть дуги одной Δ-дуги можно различить по их конечным вершинам.
2) В каждой вершине известно, кроме её номера, число Δ-дуг и число дуг в каждой Δ-дуге.

Проблема обхода недетерминированного графа в том, что регулятор посылает движок по одной дуге, а он проходит другую дугу с тем же номером, то есть в той же Δ-дуге.

Слайд 41. Справедливый недетерминизм. Коллектив полуроботов (2).

Попробуем приспособить к недетерминированному графу алгоритм обхода детерминированного графа с помощью движков и регуляторов, о котором я рассказывал.

Первое решение, которое приходит на ум, простое: пусть движок сообщит регулятору, куда он попал.
А регулятор начала дуги смотрит, туда ли он направлял движок. Если не туда, движок убивается, а если туда, ему разрешают продолжить работу.

Однако это решение неудачное.
Например, Δ-дуга состоит из двух незаблокированных прямых дуг: *a* и *b*. Регулятор посылает первый движок по дуге *a*, а второй – по дуге *b*. Но из-за недетерминизма всё получается наоборот: первый проходит дугу *b*, а второй – дугу *a*. Регулятор убивает оба движка. А на следующем шаге всё повторяется. Недетерминизм справедливый: он честно перебирает дуги в Δ-дуге: *a*, *b*, *a*, *b* и так далее. Но регулятор убивает все движки.

Слайд 42. Справедливый недетерминизм. Коллектив полуроботов (3).

Второе решение подсказывает этот же пример. Нам ведь всё равно, какую дугу пройдёт движок *a* или *b*: они обе незаблокированные.
Так что регулятор должен вести себя просто умнее: если движок прошёл одну из незаблокированных дуг, то всё в порядке.
Только при проходе заблокированной дуги движок уничтожается.

Это решение правильное, но неэффективное. Дело в том, что движку приходится опрашивать регуляторы после прохода каждой дуги.

Третье решение использует приём, аналогичный тому, что использовался в детерминированном случае. Там движок не опрашивал регуляторы, если проходил по прямой дуге, так как регулятор её начала уже знал регулятор конца дуги.
Однако в недетерминированном случае движок может пройти по другой дуге.
С другой стороны, движок всегда знает номер вершины, в которую попал.
Если бы регулятор для каждой прямой выходящей дуги хранил не только регулятор её конца, но и номер вершины, он мог бы подсказать движку по номеру вершины адрес её регулятора.
Это всё делается обменом сообщениями между движком и регулятором.
В этом случае поиск регулятора выполняется, как и в детерминированном случае, только после прохода белой дуги.

Слайд 43. Справедливый недетерминизм. Коллектив полуроботов. Оценки.

Как оценить такой алгоритм?

Размер сообщений и памяти автомата увеличиваются, но имеют тот же порядок, что в детерминированном случае.

А время обхода, конечно, зависит от недетерминированной функции выбора.

Если она на самом деле детерминирована, то получаем ту же по порядку оценку $m+nD$.

Пример другой функции справедливого выбора – это t-недетерминизм, который для фиксированного числа t гарантирует, что за t попыток пройти по Δ-дуге будут пройдены все её дуги. Но здесь получается оценка, по крайней мере, экспоненциальная. Пример внизу слайда: зловредная функция выбора из t попыток пройти по Δ-дуге только последний раз направляет движок по дуге направо.

Для практического применения нужно использовать какие-то дополнительные ограничения.
Например, не все дуги недетерминированные. Есть k детерминированных дуг и по ним можно попасть из корня в любую вершину.
Тогда сначала обходим граф по детерминированным дугам. а потом, используя прямое дерево, посылаем движки для прохода недетерминированных дуг.
Каждый такой движок проходит детерминированный путь длины порядка n, потом недетерминированную Δ-дугу, после чего уничтожается.
Тогда t-недетерминированная Δ-дуга с точки зрения обхода эквивалентна t детерминированным дугам.
Получается уже полиномиальная оценка $\boldsymbol{O}(k+ t(m-k)+nD)$.

Слайд 44. Недетерминированный граф. Δ-обход.
Другой подход к недетерминированному графу – это модификация самой цели обхода: вместо того, чтобы проходить по всем дугам, требуется пройти по всем Δ-дугам, то есть хотя бы по одной дуге в каждой Δ-дуге.

Это имеет практический смысл при тестировании: в каждом состоянии системы мы пробуем подать на неё каждое тестовое воздействие хотя бы по одному разу.

Δ-маршрут = множество маршрутов с одной начальной вершиной, которое «ветвится» по всем дугам каждой проходимой Δ-дуги.

Δ-обход = Δ-маршрут, проходящий по всем Δ-дугам.
Если при движении по графу мы называем номер следующей Δ-дуги в соответствии с Δ-обходом, то мы гарантированно пройдём какой-то маршрут из Δ-обхода, то есть пройдём по всем Δ-дугам.

Граф *сильно Δ-связен*, если для каждой пары вершин *a* и *b* существует Δ-маршрут, все маршруты которого начинаются в *a* и заканчиваются в *b*.

Δ-обход существует, если граф сильно Δ-связен.

Необходимое условие включает ещё, как и в детерминированном случае соответствующим образом определённые недетерминированные графы 1-го и 2-го рода. Но они будут немного другие, я о них не буду говорить. Во всяком случае для существования Δ-обхода, начиная с любой начальной вершины, необходима именно сильно Δ-связность.

Слайд 45. Алгоритм Δ-обхода.

Существует алгоритм Δ-обхода неизвестного сильно Δ-связного графа одним свободным полуроботом, который может читать и писать в вершины графа.

Рассказывать алгоритм я не буду. Он не простой и очень отличается от тех алгоритмов, о которых я рассказывал. На слайде приведены только оценки алгоритма.

n – число вершин, m – число Δ-дуг, s – ограничение сверху на число Δ-дуг из одной вершины.

Длина Δ-обхода $\boldsymbol{O}(nm)$.
Память вершины и автомата $\boldsymbol{O}(s\log n)$.

Если нельзя писать в вершину графа, то информация, которая должна храниться в вершине, может храниться в автомате-регуляторе, который создаётся для вершины. Однако в этом случае требуется поиск регулятора, что увеличивает время обхода в n раз, поскольку граф недетерминированный и поиск регулятора придётся делать после прохода каждой дуги.

Нерешённая задача: Δ-обход коллективом автоматов.

Слайд 46. Спасибо за внимание !

Слайд 47. Неориентированный граф. Длина обхода
Как найти S ? Это минимальное паросочетание в полном графе, вершины которого – все вершины с нечётной степенью, а вес ребра – длина минимального пути между вершинами.

Слайд 48. Алгоритм Тэрри.
Для обхода неориентированных графов имеется хорошо известный простенький алгоритм Тэрри.

Он строит обход длиной *2m* (каждое ребро проходится 2 раза).

По сути, это свободный DFS-алгоритм, т.е. алгоритм обхода в глубину, для неориентированного графа.

Как работает алгоритм Тэрри?
Ребро прямого дерева запоминается в вершине, в которую мы попадаем, когда первый раз проходим по этому ребру.
Вначале прямое дерево и текущий путь содержат только корень и не содержат рёбер. Текущая вершина – корень. Помечаем корень как старый.

Когда проходим новое ребро *ab*, помечаем его как старое в *a* и в *b*.

Шаг алгоритма:
1. Пытаемся идти из текущей вершины *a* по новому ребру *e*.
1.1. Прошли по новому ребру *e*, которое ведёт в *b*.
1.1.1. Если *b* новая, помечаем *e* как прямое в *b*. *a:=b,* ↑1.
1.1.2. Если *b* старая, пытаемся идти по новому ребру.
1.1.2.1. Если нового ребра нет (*b* завершённая), то возвращаемся в *a* по *e* *Откат по хорде*. ↑1.
1.1.2.2. Прошли по новому ребру в вершину *c*. *a:=b*, *b:=c*, ↑1.1.2.
Новая ветвь дерева.
1.2. Если нового ребра нет (*a* завершённая), то:
1.2.1. Если *a* не корень, идём из *a* по прямому ребру (в сторону корня) в *c*. *Откат по дереву*. *a:=c*, ↑1.
1.2.2. Если *a* корень, то *конец обхода*.

Почему каждое ребро проходится 2 раза?
Первый раз ребро проходится, когда мы по нему идём как по новому ребру.
Если это хорда, то второй раз мы её проходим сразу после первого раза при откате по хорде.
Если это прямое ребро, то второй раз мы его проходим при откате по дереву.

Слайд 49. Оптимизация алгоритма Тэрри.

Петля. У петли два номера. Робот. Третий обход: При первом обходе размечаем все хорды. Это можно сделать, т.к. по каждой хорде идём два раза подряд. Первый раз она помечается как хорда в конце, а второй раз в начале. При втором обходе перед тем как идти по белой хорде, помечаем вершину. Если пришли в вершину с пометкой, то это петля, отмечаем эту дугу. В любом случае, поскольку мы возвращаемся по хорде, снимаем пометку в вершине. Поэтому, начиная с третьего обхода петли проходятся по одному разу.

Самая правая ветвь прямого дерева. Свободный робот. Повторный обход.
Если в первом обходе разметить все хорды и запомнить в вершине её степень, то во втором проходе можно самую правую ветвь прямого дерева не проходить дважды. В каждой вершине сначала проходим хорды, каждую в обоих направлениях, а потом остальные рёбра. Для самой правой ветви дерева это будет означать, что в каждой его вершине самое правое выходящее прямое ребро проходится в последнюю очередь. Робот помнит, что идёт по таким дугам и вместо отката конец обхода.

Как неизбыточный робот может в DFS не проходить второй раз последнее ребро.
Когда бываем в корне, запоминаем, есть в корне белые рёбра или уже нет.
Когда выходим из корня по белому ребру, которое становится прямым, т.е. не по петле, то это ребро помечаем в его конце.
Если надо делать откат по помеченному ребру, то делаем его только, если в корне есть белые рёбра. И снимаем пометку. А если в корне белых рёбер нет, то конец обхода.
Когда из корня идём по последнему белому ребру и оно оказалось петлёй, то конец обхода.

Как узнать, что главный путь – самая правая ветвь в дереве?

С самого начала корень помечаем как *красный*.
Если из красной вершины, пройдя все хорды, идём по самому правому ребру, то его конец тоже помечаем как красный.
Из красной вершины не делаем откат по дереву, а заканчиваем обход.

Старый вариант:
Прямое ребро, выходящее из вершины может быть помечено как *красное*.
Вершина может быть помечена признаком – *кружочком*.
Состояния автомата дублируются: оригинал-состояние и дубль-состояние. Кроме того, вводится новое главное начальное состояние, не имеющее оригинала и дубля.

В начале работы, когда автомат находится в корне в главном начальном состоянии.

1. Автомат в главном начальном состоянии.
   Если в корне есть красное ребро, то автомат переходит в дубль старого начального состояния. А если нет, то в оригинал старого начального состояния.
2. Автомат находится в оригинальном состоянии.
   Автомат работает как обычно, за исключением следующего случая: если автомат проходит прямое ребро $a \to b$ в обратном направлении (к корню), т.е. из вершины b в вершину a, то после прохода этого ребра автомат проверяет, нет ли в вершине a кружочка.
   Если в вершине a нет кружочка, то автомат отмечает это ребро как красное в вершине a. И при этом, если какое-то другое ребро в a было помечено как красное, то эта пометка снимается.
   Если в вершине a есть кружочек, то автомат переходит в дублирующее состояние.
3. Автомат находится в дубль-состоянии. Тогда он находится на самой правой ветви прямого дерева.
   3.1.В вершине есть красное ребро.
      3.1.1. Есть белое ребро с номером меньше, чем красное ребро.
      Автомат ставит кружок в вершине и переходит в оригинальное состояние, а потом идёт по ребру.
      3.1.2. Все ребра с номером меньше, чем красное ребро, чёрные.
      Автомат пытается идти по ребру с номером следующим после номера красного ребра. Это будет просто попытка пройти по белому ребру.
         3.1.2.1. Такое ребро есть.
         После его прохода автомат тут же возвращается обратно. Это проход по хорде, имеющей номер больше, чем красное ребро, в дублирующем состоянии.
         3.1.2.2. Такого ребра нет.
         Автомат снимает кружок с вершины, если он там был, и идёт по красному ребру, оставаясь в дублирующем состоянии.
   3.2.В вершине нет красного ребра.
      Тогда автомат находится в листовой вершине самого правого пути прямого дерева. Автомат пытается пройти по белому ребру.
      3.2.1. Такое ребро есть.
      Тогда это хорда, и автомат возвращается назад, оставаясь в дублирующем состоянии.
      3.2.2. Такого ребра нет.
      Конец обхода.

Слайд 50. BFS-алгоритм. Доказательство оценки.
Автомат пройдёт по каждой хорде ровно 1 раз.
Далее по каждому прямому ребру, число которых равно $n-1$, придётся пройти 4 раза:
1) первый проход по белому ребру, 2) возврат в начало ребра, чтобы пометить его как прямое в его начале, 3) повторить проход вперёд, 4) при откате по ребру, который для каждого прямого ребра выполняется 1 раз, поскольку после этого ребро удаляется из серого дерева.
Кроме этого, выполняются поиски серых вершин. Такой поиск начинается в двух случаях: 1) после прохода по белой хорде в вершину, которая из-за этого становится чёрной, если эта вершина не листовая; 2) после отката по прямому ребру, если попали в чёрную развилку. Число первых случаев не более n (вершина только один раз становится чёрной), число вторых случаев тоже не больше n (откат по прямому ребру делается один раз).
Каждый поиск – это проход простого пути (без самопересечения), т.е. имеет длину не более n.
Тем самым, длина обхода не больше $m-(n-1)+4(n-1)+n^2 = m+\boldsymbol{O}(n^2)$.

Эта оценка достижима на графе, который представлен на слайде.

Слайд 51. Неизвестный граф. DFS- и BFS-автоматы. Не робот. Оценки.
Пример на слайде – общий для DFS и BFS.

Слайд 52. Два робота. Описание алгоритма.
Если конечных автоматов несколько, каждый из них может читать пометки в вершинах, оставленные другими автоматами, и обмениваться с ними сообщениями. Для двух автоматов оценка равна уже $\boldsymbol{\Theta}(nm)$.
Сначала оба автомата в корне. Один идёт по белой дуге, а второй остаётся в корне и ставит в нём *крестик*. Если пройдена петля, то первый об этом узнаёт и идёт по другой дуге. Если это не петля, то теперь они оба на сером пути, первый в его конце – вершине, а второй в предыдущей вершине с крестиком.
Если вершина серая, первый автомат идёт по белой дуге, а второй стоит и ждёт сообщения от первого. В сообщении указывается цвет вершины, куда попал первый.

Если первый автомат попал в белую вершину, второй автомат стирает крестик, продвигается по серому пути на 1 дугу и ставит крестик.

Если первый автомат попал в серую или чёрную вершину (прошёл по хорде), второй автомат остаётся на месте. Далее оба автомата двигаются синхронно по циклу отката, т.е. по обратному пути до серого пути, а далее по серому пути. Это синхронное движение происходит так: первый автомат проходит по дуге и посылает сообщение второму автомату, который тоже проходит по дуге и посылает ответное сообщение, после чего первый автомат может идти

дальше. Они двигаются до тех пор, пока первый автомат не дойдёт до вершины с крестиком. Тогда первый автомат стирает крестик, посылает сообщение второму автомату и тот ставит крестик.

Если конец серого пути стал чёрным, то второй автомат ставит крестик в своей вершине. Далее оба автомата двигаются синхронно по циклу отката, т.е. по обратному пути до серого пути, а далее по серому пути. До тех пор, пока первый автомат не дойдёт до вершины с крестиком.

Если серый путь нулевой длины, т.е. состоит только из корня, то крестик будет стоять в корне, первый автомат дойдёт до корня. Если корень чёрный, конец обхода. Если корень серый, первый автомат идёт по белой дуге, а второй автомат заходит в корень.

Слайд 53. Автоматы с неограниченным числом состояний. Обход неизвестного графа.

1 движок за 1 такт.

A – число автоматов.

Генератор направляет движки так: каждый движок идёт от корня к какой-нибудь серой вершине, а потом по белым дугам, после чего уничтожается. Движок проходит маршрут $O(n)$. Значит, одновременно существующих $O(n)$. А всего $O(m)$.

Генератор хранит граф: список дуг, включая белые из серых вершин, с разделителем между вершинами, для каждой дуги - номер её конца и признак белая или нет: *mlogn*.

Движок хранит: адрес генератора, номер вершины начала дуги, номер дуги: logA+logn+logs.

Сообщение от движка генератору: адрес генератора, адрес движка, номер начала дуги, номер дуги, номер вершины: 2logA+2logn+logs.

Сообщение от генератора движку: номер дуги: *logs*.

Ограниченная полустепень	нет	нет	да	да
Повторный адрес	нет	да	нет	да
Память генератора	mlogn	mlogn	nlogn	nlogn
Память движка	logm	logm	logn	logn
Сообщение генератору	logm	logm	logn	logn
Сообщение движку	logm	logm	1	1

Регуляторы

Память автомата: ограниченное число k адресов автоматов, номер вершины, номер дуги, число выходящих дуг, для каждой выходящей дуги – адрес автомата: klogA+logn+logs+logs+slogA.
Сообщение: ограниченное число k адресов автоматов, номер вершины, число выходящих дуг или номер дуги: klogA+logn+logs.

Ограниченная полустепень	нет	нет	да	да
Повторный адрес	нет	да	нет	да
Память автомата	mlogm	mlogn	logn	logn
Сообщение	logm	logm	logn	logn

Преобразование вершины в цепочку вершин: для преобразованного графа в терминах исходного. Нужно брать столбцы для ограниченной полустепени, но вместо *n* читать *m*.

Слайд 54. Автоматы с неограниченным числом состояний. Обход неизвестного графа.
Много движков за 1 такт. Нижняя оценка.

Слайд 55. Автоматы с неограниченным числом состояний. Обход неизвестного графа.
Много движков за 1 такт. Верхняя оценка.

Слайд 56. Проход по белой дуге в не белую вершину.
Красным, как обычно, отмечены дуги главного дерева. Регулятор знает адреса регуляторов, находящихся на других концах этих дуг.
Движок, попадая в вершину, спрашивает у её регулятора, куда ему идти.
▸ 1. Регулятор, прежде всего, посылает запрос на новый движок вниз по главному дереву через цепочку регуляторов до корня, в котором сидит генератор движков.
Регулятор направляет движки по белым дугам, а если таких нет, то – по красным.
Регулятор начала красной дуги не посылает по ней движок до тех пор, пока не получит запрос от регулятора конца дуги. Тем самым, реализуется старт-стопная дисциплина перемещения движков по дугам.
▸ 2. Регулятор отвечает движку, куда ему идти, указывая номер выходящей дуги.

- 3. Если движок проходит белую дугу, он узнаёт номер вершины, в которую попал.
- 4. Теперь ему нужно узнать, это белая вершина или нет, то есть имеется ли у вершины регулятор. Для этого выполняется опрос регуляторов, которые связаны в список по их адресам в сети связи.
- 5. Если регулятор найден, то это значит, что движок прошёл хорду.
- 6. Об этом регулятору начала дуги посылается сообщение ***конец***.
- 7. Регулятор больше не будет посылать движки по этой хорде.
- 8. Движок самоуничтожается.

Слайд 57. Проход по белой дуге в белую вершину.

- 1. Если регулятор не найден, то вершина белая, а дуга должна стать новой прямой дугой.
- 2. Движок сообщает об этом регулятору начала дуги в сообщении ***запрос***, дуга становится красной.
- 3. Движок становится регулятором новой вершины.
- 4. Регулятор посылает ещё один запрос, потому что ему нужен движок для очередной белой дуги.

Слайд 58. Отсутствующая дуга (вершина становится чёрной).

Что делать, если белой дуги больше нет?

- 1. Движок сообщает об этом регулятору,
- 2. а сам уничтожается.

Запрос не посылается. После этого регулятор будет только ретранслировать запросы, приходящие сверху от регуляторов концов выходящих красных дуг.

Слайд 59. Проход вверх по главному дереву.

Когда вершина стала чёрной, из неё уже не выходят белые дуги, и регулятор направляет движки только по красным дугам.

- 1. Движку можно теперь сообщить не только номер дуги, но и адрес регулятора её конца. Так что опрос регуляторов не нужен.
- 2. Следующий движок по этой дуге не будет направляться, пока от регулятора её конца не придёт запрос.

Слайд 60. Откат по главному дереву.

Поскольку сообщения запрос полностью регулируют направление движков по дугам, откат по главному дереву не нужен для обхода. Однако он нужен для того, чтобы узнать, что обход закончился.

При откате регулятор начала красной дуги получает сообщение ***конец***, и эта дуга перестаёт быть красной, т.е. удаляется из главного дерева.

- 1. Когда чёрная вершина становится листовой вершиной главного дерева, регулятор вершины посылает сообщение ***конец*** регулятору начала входящей прямой дуги. Конец обхода, когда откат невозможен, то есть в чёрном корне нет красных дуг.