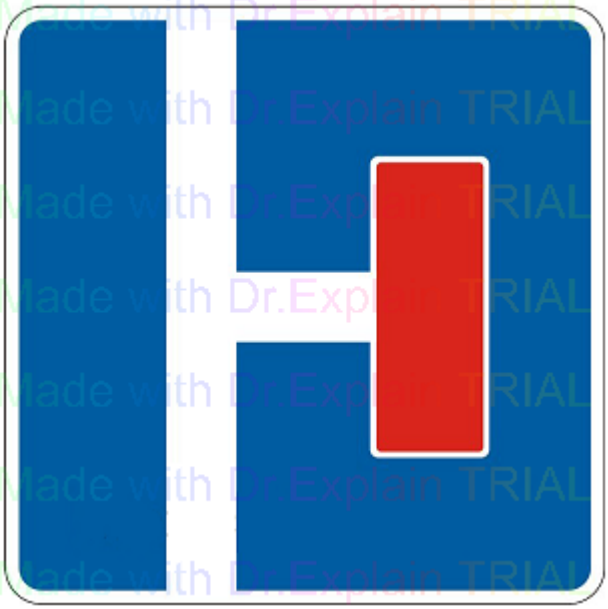


Обнаружение тупиков

Руководство пользователя

Авторы программы: Рылов Александр
соавтор: Крутиков Александр
группа ИВТ-42
руководитель: доцент кафедры
ЭВМ
Караваева Ольга Владимировна

по всем вопросам обращаться по почте yadrodisk@yandex.ru
или по телефону 8-951-349-59-37 (Александр)
8-953-673-04-67 (Алексакндр)

2014

Оглавление

Обнаружение тупиков

Лабораторная работа предназначена для студенов 4-ого курса ВятГУ, направления ИВТ (Факультет Автоматики и Вычислительной Техники) или ВМ(Факультет Автоматики и Вычислительной Техники).

Теоретические сведения

Обнаружение тупиков.

Чтобы распознать тупиковое состояние, необходимо для каждого процесса определить, сможет ли он когда-либо снова развиваться, то есть изменять свои состояния. Так как нас интересует возможность развития процесса, а не сам процесс смены состояния, то достаточно рассмотреть только самые благоприятные изменения состояния.

Очевидно, что незаблокированный процесс (он только что получил ресурс и поэтому не заблокирован) через некоторое время освобождает все свои ресурсы и затем благополучно завершается. Освобождение ранее занятых ресурсов может «разбудить» некоторые ранее заблокированные процессы, которые, в свою очередь, развиваясь, смогут освободить другие ранее занятые ресурсы. Это может продолжаться до тех пор, пока либо не останется незаблокированных процессов, либо какое-то количество процессов все же останется заблокированными. В последнем случае (если существуют заблокированные процессы при завершении указанной последовательности действий) начальное состояние *S* является состоянием тупика, а оставшиеся процессы находятся в тупике в состоянии *S*. В противном случае *S* не есть состояние тупика.

Обнаружение тупиков посредством редукции графа повторно используемых ресурсов.

Наиболее благоприятные условия для незаблокированного процесса P, могут быть представлены редукцией (сокращением) графа повторно используемых ресурсов (см. первый раздел данной главы, описание модели Холта). Редукция графа может быть описана следующим образом:

· Граф повторно используемых ресурсов сокращается процессом P;, который не является ни заблокированной, ни изолированной вершиной, с помощью удаления всех ребер, входящих в вершину P; и выходящих из p|. Эта процедура является эквивалентной приобретению процессом p| неких ресурсов, на которые он ранее выдавал запросы, а затем освобождению всех его ресурсов. Тогда P, становится изолированной вершиной.

· Граф повторно используемых ресурсов несокращаем (не редуцируется), если он не может быть сокращен ни одним процессом.

· Граф ресурсов типа SR является полностью сокращаемым, если существует последовательность сокращений, которая удаляет все дуги графа.

Приведем лемму, которая позволяет предложить алгоритмы обнаружения тупика.

Лемма. Для ресурсов типа SR порядок сокращений несуществен; все последовательности ведут к одному и тому же несокращаемому графу.

Доказательство. Допустим, что лемма неверна. Тогда должно существовать некоторое состояние S, которое сокращается до некоторого несокращаемого состояния S1 с помощью последовательности seq1 и до несокращаемого состояния S2 — с помощью последовательности seq2 так, что S1≠S2 (то есть все процессы в состояниях S1и S2 или заблокированы, или изолированы).

Если сделать такое предположение, то мы приходим к противоречию, которое устраняется только при условии, что S1=S2. Действительно, предположим, что последовательность seq1 состоит из упорядоченного списка процессов (P1, P2, …, Pk). Тогда последовательность seq1 должна содержать процесс P, который не содержится в последовательности seq2. В противном случае S1=S2, потому что редукция графа только удаляет дуги, уже существующие в состоянии S, а если последовательности seq1 и seq2 содержат одно и то же множество процессов (пусть и в различном порядке), то должно быть удалено одно и то же множество дуг. И доказательство по индукции покажет, что $P \neq P_i$ $(i = 1, 2, \ldots k)$ приводит к указанному нами противоречию.

· P≠P1, так как вершина S может быть редуцирована процессом P1, а состояние S2 должно, следовательно, также содержать процесс P1.

· Пусть P≠Pi, (i = 1, 2,..., j). Однако, поскольку после редукции процессами Pi, (i= 1, 2, ..., j) возможно еще сокращение графа процессом Pj+1, это же самое должно быть справедливо и для последовательности seq2 независимо от порядка следования процессов. То же самое множество ребер графа удаляется с помощью процесса Pi. Таким образом, P≠ Pj+1.

Следовательно, P≠Pi для i = 1, 2,..., k и P не может существовать, а это противоречит нашему предположению, что S1≠S2. Следовательно, S1=S2.

Теорема о тупике. Состояние S есть состояние тупика тогда и только тогда, когда граф повторно используемых ресурсов в состоянии S не является полностью сокращаемым.

Доказательство.

а) Предположим, что состояние S есть состояние тупика и процесс Pi находится в тупике в S. Тогда для всех Sj, таких что S —*—> Sj процесс Pi заблокирован в Sj (по определению). Так как сокращения графа идентичны для серии операций процессов, то конечное несокращаемое состояние в последовательности сокращений должно оставить процесс Pi блокированным. Следовательно, граф не является полностью сокращаемым.

б) Предположим, что S не является полностью сокращаемым. Тогда существует процесс Pi, который остается заблокированным при всех возможных последовательностях операций редукции в соответствие с леммой. Так как любая последовательность операций редукции графа повторно используемых ресурсов, оканчивающаяся несокращаемым состоянием, гарантирует, что все ресурсы типа SR, которые могут когда-либо стать доступными, в действительности освобождены, то процесс Pi навсегда заблокирован и, следовательно, находится в тупике.

Следствие 1. Процесс Pi не находится в тупике тогда и только тогда, когда серия сокращений приводит к состоянию, в котором Pi не заблокирован.

Следствие 2. Если S есть состояние туника (по ресурсам типа SR), то по крайней мере два процесса находятся в тупике в S.

Из теоремы о тупике непосредственно следует и алгоритм обнаружения тупиков. Нужно просто попытаться сократить граф по возможности эффективным способом; если граф полностью не сокращается, то начальное состояние было состоянием тупика для тех процессов, вершины которых остались в несокращенном графе. Рассмотренная нами лемма позволяет удобным образом упорядочивать сокращения.

Граф повторно используемых ресурсов может быть представлен или матрицами, или списками. В обоих случаях экономия памяти может быть достигнута использованием взвешенных ориентированных мультиграфов (слиянием определенных дуг получения или дуг запроса между конкретным ресурсом и данным процессом в одну дугу с соответствующим весом, определяющим количество единиц ресурса).

Рассмотрим вариант с матричным представлением. Поскольку граф двудольный, он представляется двумя матрицами n*m. Одна матрица – матрица распределения D=|| di j||, в которой элемент dij отражает количество единиц Rj ресурса, распределенного процессу Pi, то есть dij=|(Rj,Pi)|, где (Rj,Pi) – дуга между вершинами Rj и Pi, ведущая из Rj в Pi. Вторая матрица – матрица запросов N=|| ni j||, где nij=|(Pi,Rj)|.

В случае использования связанных списков дл отображения той же структуры можно построить две группы списков. Ресурсы, распределенные некоторому процессу Pi, связаны с Pi указателями: Pi®(Rx, dx)®(Ry, dy)®…®(Rz, dz), где Rj – вершина, представляющая ресурс, а dj – вес дуги dj=|(Rj,Pi)|. Подобным образом и ресурсы, запрошенные процессом Pi, связаны вместе.

Аналогичные списки создаются и для ресурсов (списки распределенных и запрошенных ресурсов) Ri®(Pu, du)®(Pv, nv)®…®(Pw, nw), где (Pj,Ri)|.

Для обоих представлений удобно также иметь одномерный массив доступных единиц ресурсов (r1,r2,…rm), где ri указывает число доступных (нераспределенных) единиц ресурса Ri , т.е. ri=|Ri|-S |(Ri,Pk)|

Простой метод прямого обнаружения тупика заключается в просмотре по порядку списка (или матрицы) запросов, причем, где возможно, производятся сокращения дуг графа до тех пор, пока нельзя будет сделать более ни одного сокращения. При этом самая плохая ситуация возникает, когда процессы упорядочены в некоторой последовательности P1,P2, ... , Pn, а единственно возможным порядком сокращений является обратная последовательность, то есть Pn, Pn-1, ... ,P2,P1, а также в случае, когда процесс запрашивает все m ресурсов. Тогда число проверок процессов равно

n + (n-1) + ... + 1 = n*(n+l)/2,

причем каждая проверка требует испытания m ресурсов. Таким образом, время выполнения такого алгоритма в наихудшем случае пропорционально m*n2.

Более эффективный алгоритм может быть получен за счет хранения некоторой дополнительной информации о запросах. Для каждой вершины процесса P, определяется так называемый счетчик ожиданий wi отображающий количество ресурсов (не число единиц ресурса), которые в какое-то время вызывают блокировку процесса. Кроме этого, можно сохранять для каждого ресурса запросы, упорядоченные по размеру (числу единиц ресурса). Тогда следующий алгоритм сокращений, записанный на псевдокоде, имеет максимальное время выполнения, пропорциональное m*n.

For all PÎL do Begin
For all RjÎ|(Rj,P)|>0 do Begin
rj:=rj+|(Rj,P)|;
For all PiÎ 0<|(Pi,Rj)| rj do Begin

wi:=wi -1;
If wi=0 then L:=LÈ{Pi}
End
End
End
DeadLock:=Not(L={P1,P2,...,Pn});
For all

Здесь L — это текущий список процессов, которые могут выполнять редукцию графа. Можно сказать, что L := {Pj | wi=0}. Программа выбирает процесс P из списка L, процесс P сокращает граф, увеличивая число доступных единиц rj всех ресурсов Rj, распределенных процессу P, обновляет счетчик ожидания wj каждого процесса Pj, который сможет удовлетворить свой запрос на частный ресурс Rj, и добавляет Pj к L, если счетчик ожидания становится нулевым.

Методы обнаружения тупика по наличию замкнутой цепочки запросов.

Структура графа обеспечивает простое необходимое (но не достаточное) условие для тупика. Для любого графа $G = <X, E>$ и вершины $x Î X$ пусть $P(x)$ обозначает множество вершин, достижимых из вершины x, то есть

$P(x) = \{ y \mid (x. y) Î E \} U \{ z \mid (y, z) Î E \& y Î P(x) \}$.

Можно сказать, что в ориентированном графе *потомством вершины x,* которое мы обозначаем как $P(x)$, называется множество всех вершин, в которые ведут пути из x.

Тогда если существует некоторая вершина $x Î X : x Î P(x)$, то в графе G имеется цикл.

Теорема 1. Цикл в графе повторно используемых ресурсов является необходимым условием тупика.

Для доказательства этой теоремы можно воспользоваться следующим свойством ориентированных графов: если ориентированный граф не содержит цикла, то существует линейное упорядочение вершин, такое, что если существует путь от вершины i к вершине j, то i появляется перед j в этом упорядочении.

*Теорема 2.*Если $S>$не является состоянием тупика и $S \xrightarrow{pi} S_T$, гдеST есть состояние тупика в том и только в том случае, когда операция процесса Pi есть запрос и Pi находится в тупике в $S T$.

Это следует понимать таким образом, что тупик может быть вызван только при запросе, который не удовлетворен немедленно. Учитывая эту теорему, можно сделать вывод, что проверка на тупиковое состояние может быть выполнена более эффективно, если она проводится непрерывно, то есть по мере развития процессов. Тогда надо применять редукцию графа только после запроса от некоторого процесса Pi и на любой стадии необходимо сначала попытаться сократить с помощью процесса Pi. Если процесс Pi смог провести сокращение графа, то никакие дальнейшие сокращения не являются необходимыми.

Ограничения, накладываемые на распределители, па число ресурсов, запрошенных одновременно, и количество единиц ресурсов, приводят к более простым условиям для тупика.

Пучок(или узел) в ориентированном графе $G = <X, E>$ — это подмножество вершин $ZÎ X$, таких что $V x Î Z, P(x) = Z$, то есть потомством каждой вершины из Z является само множество Z. Каждая вершина в узле достижима из каждой другой вершины этого узла, и узел есть максимальное подмножество с этим свойством. Поясним сказанное рис. 12.

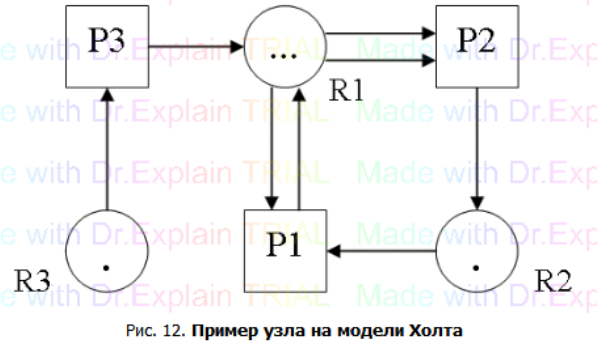


Рис. 12. **Пример узла на модели Холта**

Следует заметить, что наличие цикла — это необходимое, но не достаточное условие для узла. Так, на рис. 13 изображены два подграфа. Подграф *а* представляет собой пучок (узел), тогда как подграф *б*

представляет собой цикл, но узлом не является. В узел должны входить дуги, но они не должны из него выходить.

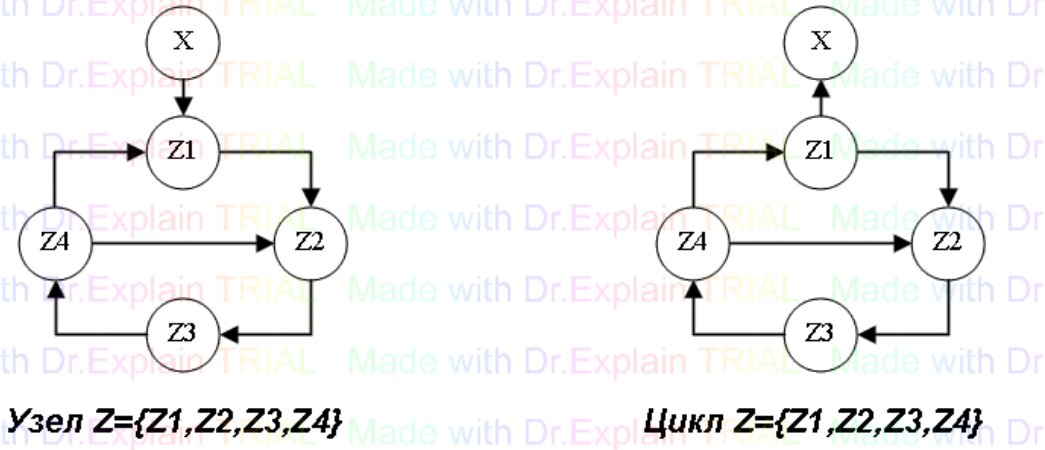


Рис. 13 **Узел и цикл в ориентированном графе**

Если состояние системы таково, что удовлетворены все запросы, которые могут быть удовлетворены, то существует простое достаточное условие существования тупика. Эта ситуация возникает, если распределители ресурсов не откладывают запросы, которые могут быть удовлетворены, а выполняют их по возможности немедленно (большинство распределителей следует этой дисциплине).

Состояние называется *фиксированным,* если каждый процесс, выдавший запрос, заблокирован.

Теорема 3. Если состояние системы фиксированное (все процессы, имеющие запросы, удовлетворены), то наличие узла в соответствующем графе повторно используемых ресурсов является достаточным условием тупика.

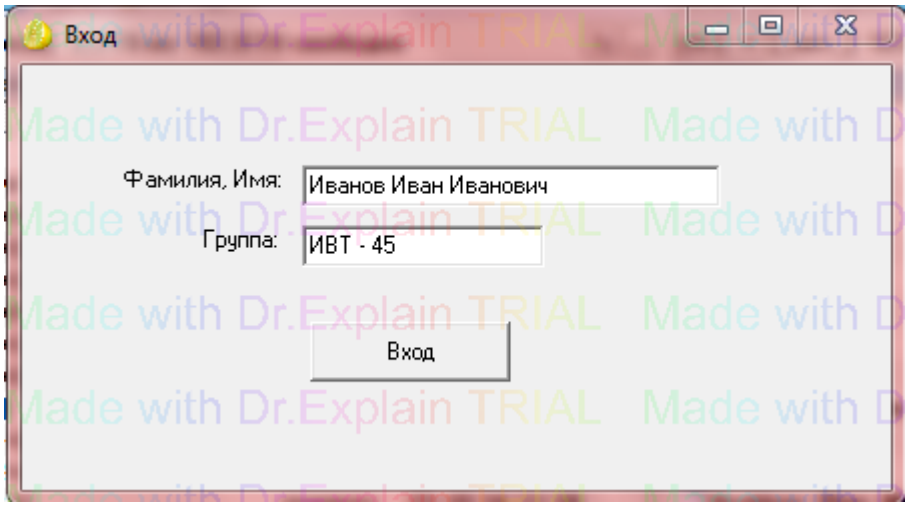
Теорема 4. Граф повторно используемых ресурсов с единичной емкостью указывает на состояние тупика тогда и только тогда, когда он содержит цикл.

Выполнение лабораторной работы

Не забудьте показать результат преподавателю.

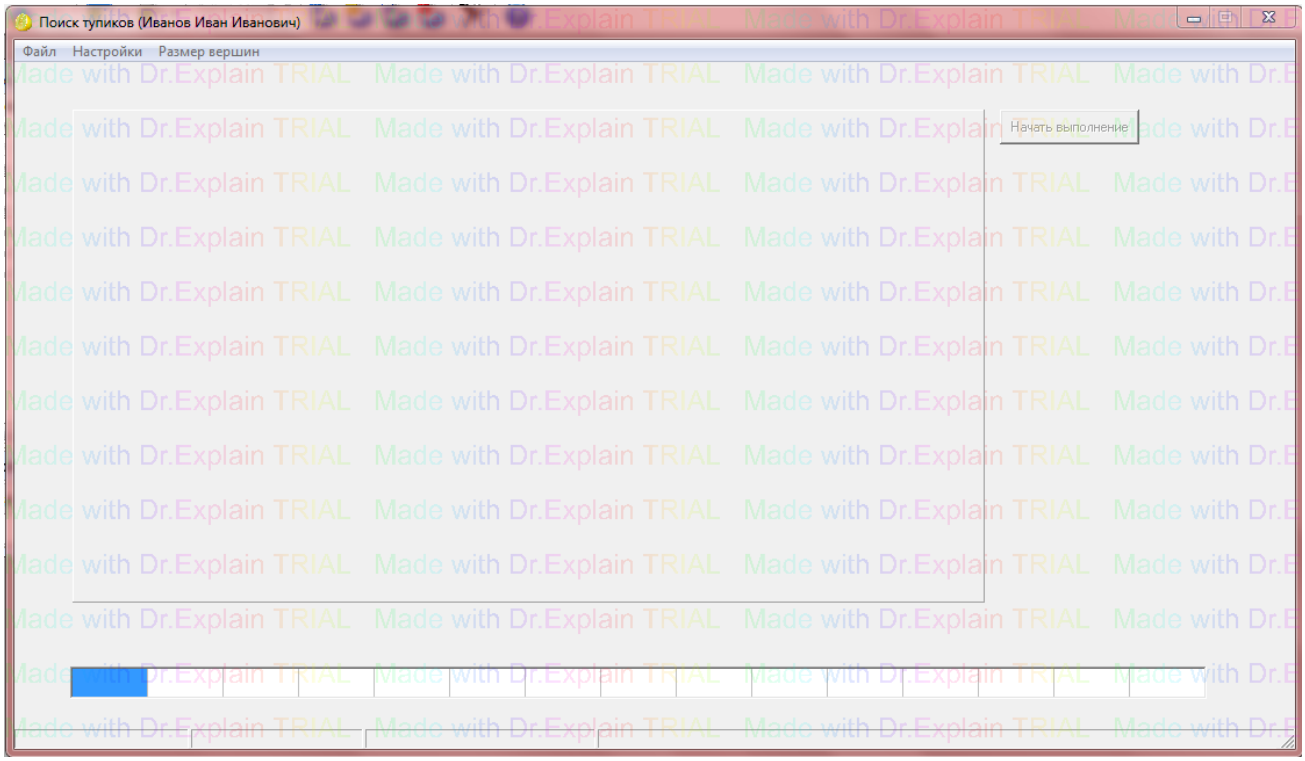
Описание интерфейса

Окно входа в программу.

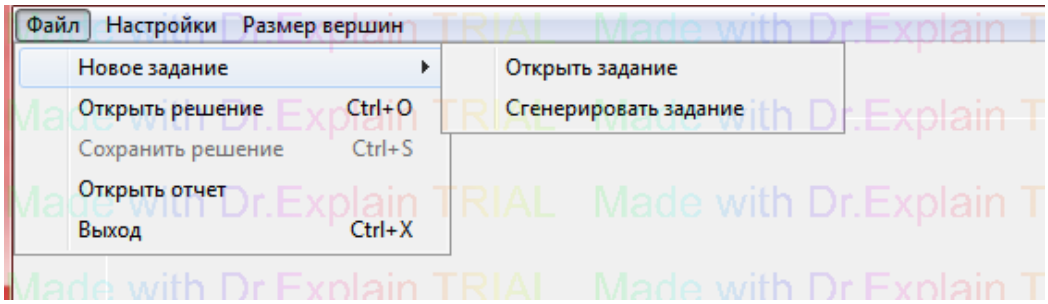


Для входа в программу введите СВОЕ (т.к. оно отображается в ОТЧЕТЕ) ФИО и группу. Нажмите кнопку "Вход".

Окно программы.

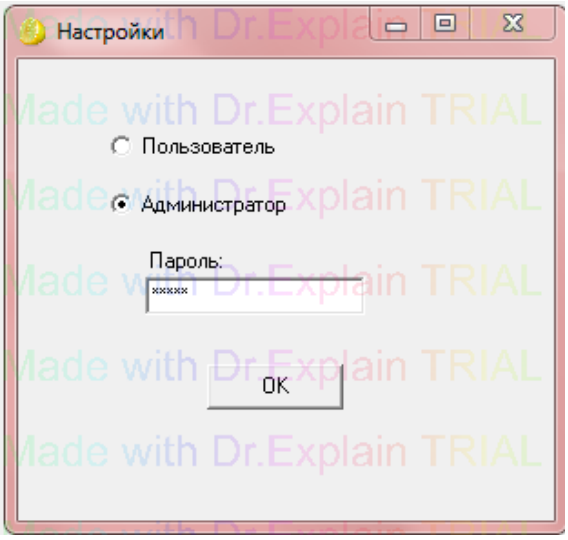


Окно программы представляет собой стандартное окно Windows. Для начала работы нажмите **Файл->Новое задание->Сгенерировать задание** либо по указанию преподавателя **Файл->Новое задание->Открыть задания.**

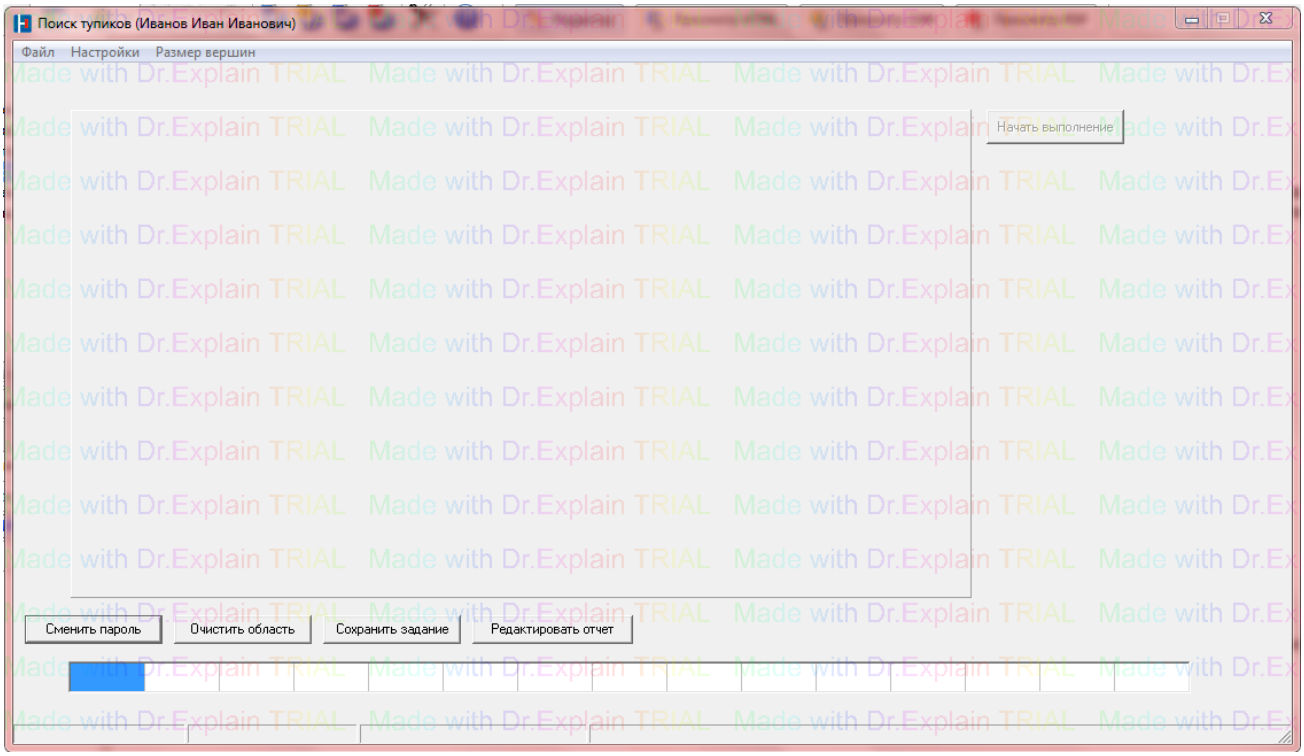


Для загрузки решения **Файл->Открыть решение**, либо комбинация клавиш **Ctrl+O**. Соответственно Сохранить решение можно **Файл->Сохранить решение**, либо комбинация клавиш **Ctrl+S**. Отчет сохраняется в любом случае, при завершениии программы. Отчет можно открыть **Файл->Открыть отчет.** Выйти из программы можно **Файл->Выход**, либо комбинация клавиш **Ctrl+X**.

Окно настроек.

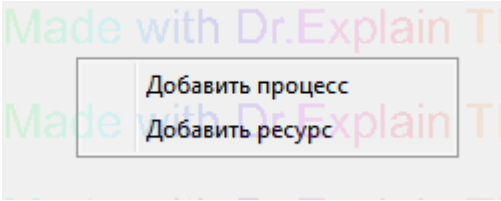


Преподаватель может войти в режиме администратора, введя пароль. По умолчанию программа работает в режиме пользователя (см. рисунок выше).

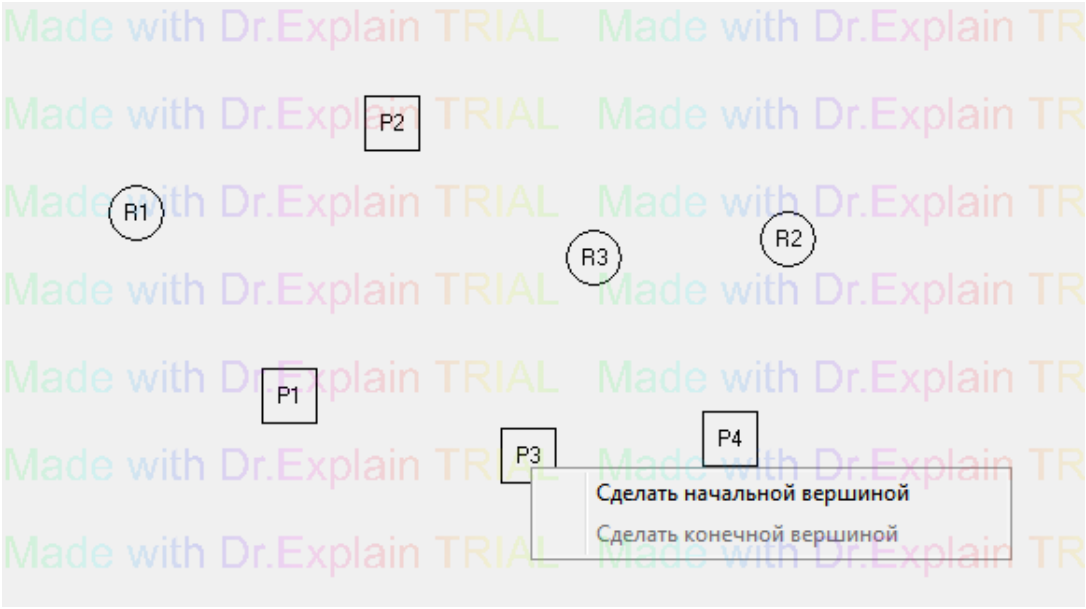


В режиме администратора преподаватель может изменить пароль, очистить область (рисунка), редактировать отчет и сожранять задание.

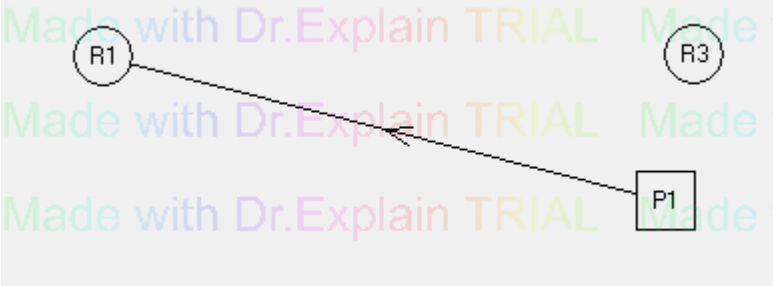
Для создания задания необходимо нарисовать граф (клие ПКМ на панели).



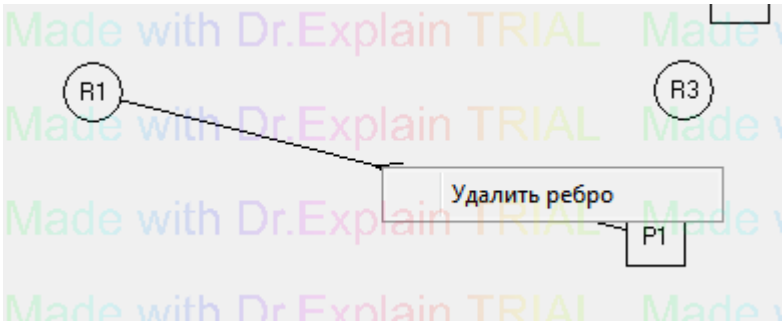
Сделать добавленную вершину начальной или конечной можно аналогично (клик ПКМ по вершине).



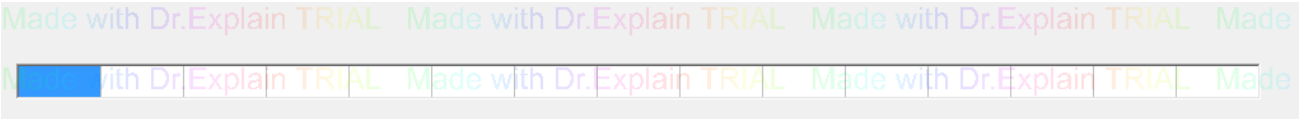
Задав начальную и конечную вершину создается связь в которой стрелка от процесса указывает на запрос процессом данного ресурса, а стрелка от ресурса указывает на принадлежность ресурса к данному процессу.



Для удаление ненужной связи необходимо клик ПКМ по связи. Далее Удалить ребро.



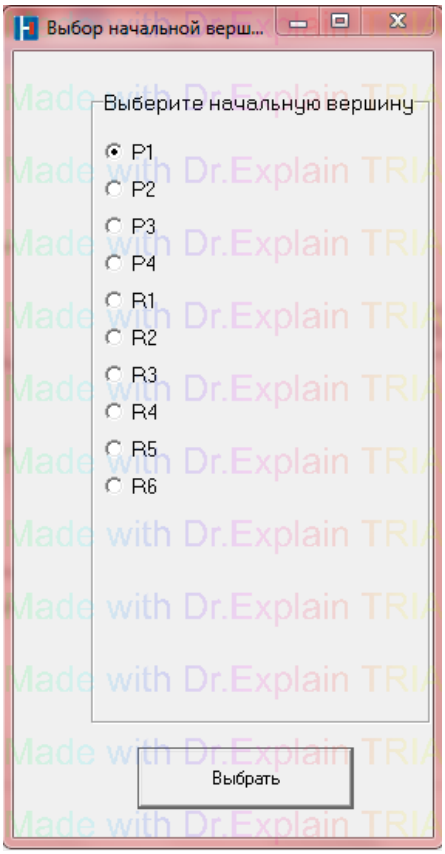
Для работы в режиме пользователя (студента). Необходимо пользоваться списком пройденных вершин (подробнее в разделе порядок выполнения лабораторной работы).

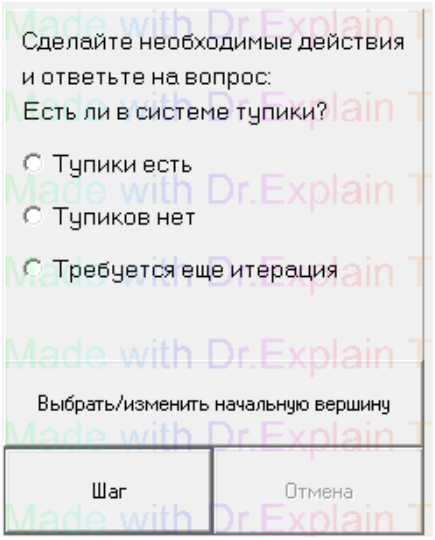


Порядок выполнения лабораторной работы

Порядок выполнения (перед тем как начинать выполнение лабораторной работы ознакомьтесь с разделом Описание интерфейса).

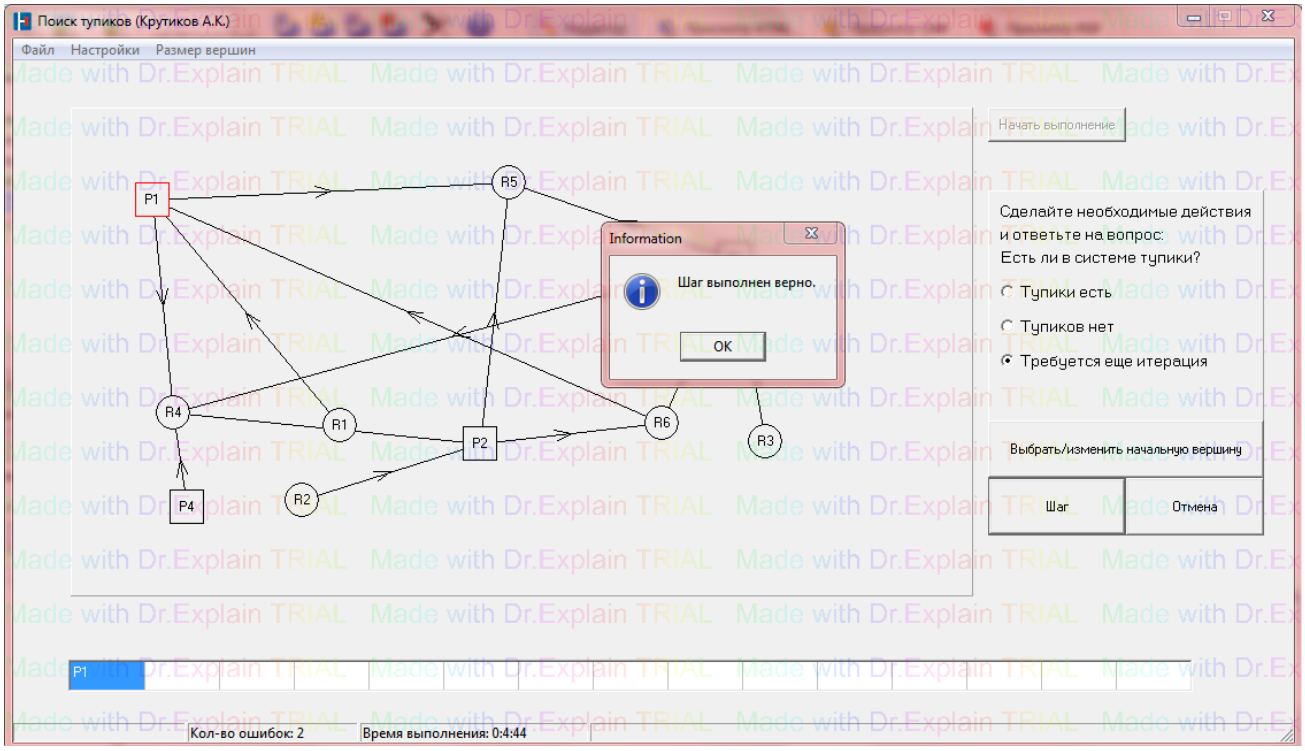
Для начала выполнения программы необходимо выбрать или сгенерировать задания (в зависимости от желания преподавателя).
Чтобы начать выполнение лабораторной работы нажмите начать задание. После начала выполнения для поиска тупика необходимо выбрать вершину и указать требуемые действия. Не забывайте что за один шаг выполняется только одно из действий (выбрать/изменить начальную вершину, пройти по ребру, откатиться по ребру)





Не забывайте добавить вершину в список (по алгоритму см. теоретические сведения). Чтобы добавить вершину в список необходимо кликнуть на вершине ПКМ (Добавить в список или Удалить из списка).

\



Внизу окна вы можете видеть свои ошибки(а именно их количества .
Размер вершин можно поменять в пунке меню Размер Вершин.