Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Информационная безопасность» (ИУ8)

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

СЕМЕСТРОВЫЙ ПРОЕКТ

«Реализация алгоритма, который по входному недерминированному конечному автомату строит детерминированный и эмулирует его»

Вариант 11

Руководитель проекта: _____ / Чесноков В.О.
(подпись, дата)

Разработчик проекта: _____ / Кошман А.А.
(подпись, дата)

Москва, 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Конечный автомат - это некоторая абстрактная модель, содержащая конечное число состояний чего-либо. Используется для представления и управления потоком выполнения каких-либо команд.

Конечный автомат подходит для :

- реализации искусственного интеллекта в играх, получая аккуратное решение без написания громоздкого и сложного кода;
- интерфейсов, в которых есть логика прогнозирования отрисовки, основанная на выбранных параметрах. Т.е. там где моделируется поведение, при котором реакция на будущие события зависит от предыдущих событий;
- структурирования приложений : продуманное применение конечных автоматов облегчает организацию и сопровождение, как логики пользовательского интерфейса, так и логики приложения;
- построения системы автоматизации документооборота;
- для улучшения характеристик нейроподобных сетей;

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Основные определения

Конечный автомат (КА) - это компьютерная программа, которая состоит из:

- Элементов маркированного списка;
- Событий, на которые реагирует программа;
- Состояний, в которых программа пребывает между событиями;
- Переходов между состояниями при реагировании на события;
- Действий, выполняемых в процессе переходов;
- Переменных, которые содержат значения, необходимые для выполнения действий между событиями.

Недетерминированный конечный автомат (НКА) - это набор из пяти элементов: $\langle V, Q, Q_0 \in Q, F \subset Q, \delta : Q \times V \rightarrow 2^Q \rangle$, где :

- V - входной алфавит (конечное множество входных символов), из которого формируются входные слова, воспринимаемые конечным автоматом;
- Q - множество состояний автомата;
- Q_0 - множество начальных состояний автомата;
- F - множество конечных состояний автомата;
- δ - функция перехода (ФП), имеющая вид : начальное состояние ФП, символ перехода, конечное состояние ФП.

Недетерминированность автоматов может достигаться тремя способами: либо существует несколько начальных состояний, либо существуют переходы, помеченные пустым символом λ , либо из одного состояния выходит несколько переходов, помеченных одной и той же меткой.

Детерминированный конечный автомат (ДКА) - это набор из пяти элементов : $\langle V, Q, q_0 \in Q, F \subset Q, \delta : Q \times V \rightarrow Q \rangle$, где :

- q_0 – начальное состояние автомата.

Это такой автомат, в котором существует одно начальное состояние, нет дуг с меткой λ и из любого состояния по любому символу возможен переход не более, чем в одно состояние.

Алгоритм детерминизации

Теорема о детерминизации

Для любого конечного автомата может быть построен эквивалентный ему детерминированный конечный автомат.

Преобразование произвольного конечного автомата к эквивалентному детерминированному осуществляется в два этапа: сначала удаляются дуги с меткой λ, затем проводится собственно детерминизация.

1) Удаление λ-переходов (дуг с меткой λ).

Чтобы перейти от исходного конечного автомата $M = (V, Q, q_0 ,F , \delta)$ к эквивалентному конечному автомату $M' = (V, Q' , q_0 , F' , \delta')$ без λ-переходов, достаточно в исходном графе M проделать следующие преобразования :

- Если мощность множества начальных состояний больше 1, то создается новое состояние q_0, из которого создаются переходы с меткой λ в состояния $\subseteq$ Q. Иначе q_0 присваивается единственное состояние из Q.
- Все состояния, кроме начального, в которые заходят только дуги с меткой λ, удаляются; тем самым определяется множество Q′ конечного автомата M′. Понятно, что $Q' \subseteq$ Q. При этом полагаем, что начальное состояние остается прежним.
- Множество дуг конечного автомата M′ и их меток (тем самым и функция переходов M′) определяется так: для любых двух состояний $p, r \in Q'$, $p \to_a r$ имеет место тогда и только тогда, когда $a \in$ V, а в графе M имеет место одно из двух: либо существует дуга из p в r, метка которой содержит символ *a*, либо существует такое состояние q, что $p \Rightarrow {}^{+}_{\lambda} q$ и $q \to_a r$. При этом вершина q может и не принадлежать множеству Q′, т.е. она может и исчезнуть при переходе к автомату M′ (рис. 7.11). Если же $q \in Q'$, то, естественно, в M′ сохранится дуга (q,r) и символ *a* будет одним из символов, принадлежащих метке этой дуги (рис. 7.12).

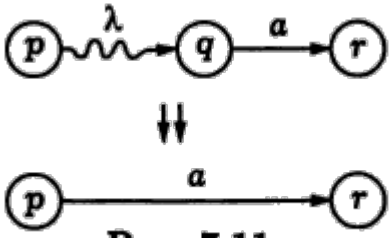


Рис. 7.11

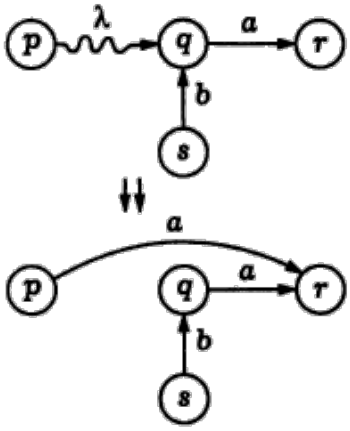


Рис. 7.12

Таким образом, в M′ сохраняются все дуги M , метки которых отличны от λ и которые соединяют пару (вершин) состояний из множества Q′. Кроме этого, для любой тройки состояний p, q, r (не обязательно различных), такой, что p, r ∈ Q′ и существует путь ненулевой длины из p в q , метка которого равна λ (т.е. путь по λ-переходам), а из q в r ведет дуга, метка которой содержит символ *a* входного алфавита, в M′ строится дуга из p в r , метка которой содержит символ *a* (см. рис. 7.11).

- Множество заключительных состояний F′ конечного автомата M′ содержит все состояния q ∈ Q′, т.е. состояния конечного автомата M, для которых имеет место $q \Rightarrow_{\lambda}^{*} q_f$ для некоторого $q_f \in F$ (т.е. либо состояние q само является заключительным состоянием конечного автомата M, либо из него ведет путь ненулевой длины по дугам с меткой λ в одно из заключительных состояний конечного автомата M) (рис. 7.13).

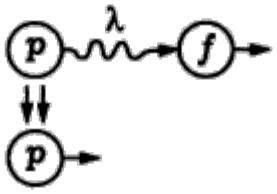


Рис. 7.13

2) Детерминизация.

Пусть M = (Q,V, q_0, F, δ) — конечный автомат без λ - переходов. Построим эквивалентный M детерминированный конечный автомат M′. Этот конечный автомат определяется таким образом, что его множество состояний есть множество всех подмножеств множества состояний конечного автомата M. Это

значит, что каждое отдельное состояние конечного автомата M′ определено как некоторое подмножество множества состояний конечного автомата M. При этом начальным состоянием нового конечного автомата (т.е. M′) является одноэлементное подмножество, содержащее начальное состояние старого конечного автомата (т.е. M), а заключительными состояниями нового конечного автомата являются все такие подмножества Q, которые содержат хотя бы одну заключительную вершину исходного конечного автомата M. Для удобства будем называть состояния конечного автомата M′ состояниями-множествами. Важно понимать, что каждое такое состояние-множество есть отдельное состояние нового конечного автомата, но никак не множество его состояний. В то же время для исходного конечного автомата M это именно множество его состояний. Образно говоря, каждое подмножество состояний автомата M "свертывается" в одно состояние нового конечного автомата (ДКА). Функция переходов нового конечного автомата определена так, что из состояния-множества S по входному символу *a* конечный автомат M′ переходит в состояние-множество, представляющее собой объединение всех множеств состояний НКА, в которые этот НКА переходит по символу *a* из каждого состояния множества S. Таким образом, конечный автомат M′ является детерминированным по построению.

Алгоритм эмуляции

Эмуляция по заданному входному слову происходит в детерминированном автомате.

Изначально ДКА находится в стартовом состоянии q_0. Автомат считывает символы входного слова по очереди. При считывании очередного символа a_i автомат переходит в состояние $\delta(q_t, a_i)$, где q_t — текущее состояние автомата. Процесс продолжается до тех пор, пока не будет достигнут конец входного слова.

ДКА допускает входное слово , если существует путь из начального состояния в какое-то терминальное, такое что буквы, выписанные с переходов на этом пути по порядку, образуют заданное входное слово. Результат представляется в виде списка состояний, по которому переходил ДКА по заданному слову.

ТРЕБОВАНИЯ К ПРОЕКТУ

- Программа должна быть платформонезависимой, не иметь зависимостей от нестандартных библиотек и выполнена в виде консольного приложения.
- Программа принимает входные данные в виде одного или нескольких текстовых файлов и записывает результат работы в текстовый файл. Имена входных и выходных файлов задаются через аргументы командной строки. Программа неинтерактивная, пользовательский ввод не предусмотрен.
- Программный код должен быть достойного качества, отформатирован и выполнен в едином стиле.
- Реализация алгоритма должна быть инкапсулирована.
- Программа должна быть покрыта тестами. Тесты должны содержать проверку корректности всех основных реализованных алгоритмов. Каждый тест представляет собой два текстовых файла с одинаковым именем, но разным расширением (например, 001.dat и 001.ans) в формате входных и выходных данных соответственно.

ВЫБОР ТЕХНОЛОГИЙ

Выбор языка программирования

Выбор языка программирования очень важная часть в разработке любого проекта. Нужно подобрать именно тот язык, который будет самым оптимальным для конкретной разработки. Именно поэтому чем больше проект, тем больше стек технологий, который в нем используется. Глобальные проекты состоят из огромного количества подпроектов. Для каждого из подпроектов выбирается тот язык, который будет наилучшем образом справляться с задачами, поставленными в нем.

Важными критериями при выборе технологий являются:

- Размер и тип проекта
- Сложность проекта
- Скорость разработки
- Доступные инструменты разработки
- Наличие готовых решений
- Гибкость решения
- Наличие подробной документации
- Требования к нагрузкам
- Требования к безопасности
- Кроссплатформенность
- Возможность интеграции с другими решениями

Так как данная работа планировалась быть не очень объёмной, а время на разработку было ограничено, то стоял вопрос выбора одного языка, а не нескольких.

Всем известно, что языки программирования разделяются по сфере применения. Основными сферами являются веб-разработка, мобильная и игровая разработка. Самыми популярными языками в веб-разработке на данный момент являются: HTML, CSS, JavaScript, Java , Python и PHP . В разработке мобильных приложений на Android: Java, на iOS : Swift и Objective-C. И наконец в разработке игр: C#, C++, JavaScript, Java, Smalltalk.

Рассмотрев достоинства и недостатки каждого из перечисленных языков, я остановилась на выборе C++. Ниже приведено подробное доказательство принятого решения.

C++ - язык общего назначения и задуман для того, чтобы настоящие программисты получили удовольствие от самого процесса программирования. За исключением второстепенных деталей он содержит язык С как подмножество. Язык С расширяется введением гибких и эффективных средств, предназначенных для построения новых типов. Программист структурирует свою задачу, определив новые типы, которые точно соответствуют понятиям предметной области задачи. Такой метод построения программы обычно называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определенных пользователем. С такими объектами можно работать надежно и просто даже в тех случаях, когда их тип нельзя установить на стадии трансляции. Программирование с использованием таких объектов обычно называют объектно-ориентированным. Если этот метод применяется правильно, то программы становятся короче и понятнее, а сопровождение их упрощается.

Достоинства C++ :

- Чрезвычайно мощный язык, содержащий средства создания эффективных программ практически любого назначения.
- Компилируемость со статической типизацией.
- Сочетание высокоуровневых и низкоуровневых средств.
- Реализация ООП.
- Работает максимально быстро.
- Предсказуемое выполнение программ, что является важным для построения систем реального времени.
- Автоматический вызов деструкторов объектов при их уничтожении, причём в порядке, обратном вызову конструкторов. Это упрощает (достаточно объявить переменную) и делает более надёжным освобождение ресурсов (память, файлы, семафоры и т. п.), а также позволяет гарантированно выполнять переходы состояний программы, не обязательно связанные с освобождением ресурсов (например, запись в журнал).
- Пользовательские функции-операторы позволяют кратко и ёмко записывать выражения над пользовательскими типами в естественной алгебраической форме.

- Язык поддерживает понятия физической (const) и логической (mutable) константности. Это делает программу надёжнее, так как позволяет компилятору, например, диагностировать ошибочные попытки изменения значения переменной. Объявление константности даёт программисту, читающему текст программы дополнительное представление о правильном

использовании классов и функций, а также может являться подсказкой для оптимизации. Перегрузка функций-членов по признаку константности позволяет определять изнутри объекта цели вызова метода (константный для чтения, неконстантный для изменения). Объявление mutable позволяет сохранять логическую константность при использовании кэшей и ленивых вычислений.

- Поддерживаются различные стили и технологии программирования, включая традиционное директивное программирование, ООП, обобщённое программирование, метапрограммирование (шаблоны, макросы).
- Используя шаблоны, возможно создавать обобщённые контейнеры и алгоритмы для разных типов данных, а также специализировать и вычислять на этапе компиляции.
- Возможность имитации расширения языка для поддержки парадигм, которые не поддерживаются компиляторами напрямую. Например, библиотека Boost.Bind позволяет связывать аргументы функций.
- Возможность создания встроенных предметно-ориентированных языков программирования. Такой подход использует, например библиотека Boost.Spirit, позволяющая задавать EBNF-грамматику парсеров прямо в коде C++.
- Используя шаблоны и множественное наследование можно имитировать классы-примеси и комбинаторную параметризацию библиотек. Такой подход применён в библиотеке Loki, класс SmartPrt которой позволяет, управляя всего несколькими параметрами времени компиляции, сгенерировать около 300 видов «умных указателей» для управления ресурсами.
- Кроссплатформенность: стандарт языка накладывает минимальные требования на ЭВМ для запуска скомпилированных программ. Для определения реальных свойств системы выполнения в стандартной библиотеке присутствуют соответствующие возможности (например, std::numeric_limits). Доступны компиляторы для большого количества платформ, на языке C++ разрабатывают программы для самых различных платформ и систем.
- Эффективность. Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы. Ни одна из языковых возможностей, приводящая к дополнительным накладным расходам, не является обязательной для использования — при необходимости язык позволяет обеспечить максимальную эффективность программы.
- Имеется возможность работы на низком уровне с памятью, адресами.

- Высокая совместимость с языком Си, позволяющая использовать весь существующий Си-код (код на Си может быть с минимальными переделками скомпилирован компилятором C++; библиотеки, написанные на Си, обычно могут быть вызваны из C++ непосредственно без каких-либо дополнительных затрат, в том числе и на уровне функций обратного вызова, позволяя библиотекам, написанным на Си, вызывать код, написанный на C++).

В совокупности с вышеописанным, можно сделать вывод, что в данной разработки программного обеспечения C++ - оптимальный выбор.

Выбор структуры данных

1) При написании проекта рассматривались три варианта структур данных для хранения функций перехода конечного автомата :
   - **Двумерный динамический массив**, выполняющий функцию таблицы переходов (табличное представление функции *δ)*. Предполагалось, что индексы строк и столбцов будут соответствовать начальному и конечному состояниям функции перехода соответственно. В ячейке на пересечении строки и столбца записывается допустимые входной и выходной символ (или список символов), если существует функция перехода для рассматриваемых состояний.
   - **Красно-черное дерево,** в котором хранились бы пары : состояние, список функций перехода, в которых заданное состояние является начальным.
   - **Хэш – таблица,** в которой хранились бы пары : состояние, функция перехода, в которой заданное состояние является начальным.

В процессе анализа задачи и рассмотрении возможных вариантов представления списка ФП ДКА было выявлено :

- Хранение в двумерном динамическом массиве не рацианально с точки зрения выделенной памяти, так как не каждому состоянию от рассматриваемого состояния будет существовать функция перехода.
- Поиск, вставка и удаление элементов в красно-черном дереве осуществляется за время O(logN), тогда как, при некоторых разумных допущениях, все три операции в хэш-таблицах в среднем выполняются за время O(1). Поэтому хранение в хэш - таблице более рационально, чем в красно-черном дереве.

В стандартной библиотеки C++ существует две реализации хэш-таблицы : std::unordered_map<состояние, ФП> и std::unordered_multimap<состояние, ФП>. Было принято решение использовать вторую представленную реализацию, так как она поддерживает хранение информации с эквивалентыми ключами.

2) Для хранения множеств начальных (конечных) состояний, а также состояний ДКА был выбран std::set<состояние>, так как :
   - в нем могут хранится только уникальные состояния. Это является востребованным в данной ситуации, так как пользователь может случайно внести несколько раз в список конечных (начальных) состояний одно и то же состояние.
   - операции поиска и вставки в среднем работают за O(logN).

3) Для хранения выходного слова был выбран std::list< состояние >, так как данная структура не сортирует добавленные элементы, а в рассматриваемом объекте важен порядок добавления.

ПЛАН РЕШЕНИЯ ЗАДАЧИ

1) Заданный входной файл с исходным недетерминированным автоматом и словом для эмуляции считывается конструктором объекта NonDeterAutomaton;
2) Создаётся объект DeterAutomaton;
3) Для объекта DeterAutomaton вызывается метод детерминизации, аргументом которого является NonDeterAutomaton (алгоритм детерминизации описан в разделе «Логика программы»);
4) В объекте DeterAutomaton происходит процесс эмуляции по входному слову;
5) В заданный выходной файл записывается детерминированный автомат и результат эмуляции (алгоритм записи описан в разделе «Формат входных и выходных файлов»);

ФОРМАТ ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ

Программа принимает входные данные в виде одного или нескольких текстовых файлов и записывает результат работы в текстовый файл. Имена входных и выходных файлов задаются через аргументы командной строки.

Алгоритм чтения из файла :

1 - ая строка : количество состояний автомата
2 - ая строка : начальные состояния автомата в формате « 0;1;…; », где 0 – название состояния q_0.
3 - ья строка : список конечных состояний автомата в формате « 0;1;…; »
4 - ая строка : список функций перехода автомата в формате « 0,*a*,1; … ; » , где 0 - начальное состояние ФП, *a* - символ из входного алфавита V, 1 – конечное состояние ФП.
5 - ая строка : заданное входное слово « absacsbv » , где a, b, c, s, v ∈ V.

Алгоритм записи в файл :

1 - ая строка : количество состояний автомата
2 - ая строка : начальное состояние автомата « {0,1,…}; »
3 - ья строка : список конечных состояний автомата в формате « {0,1, … }; {5,6, … }; {4}; … ; » , где {0,1, … } и {4} – терминальные состояния ДКА.
4 - ая строка : список функций перехода автомата в формате «{0,1, … },*a*,{5,6, … }; … ; » , где {0,1, … } - начальное состояние ФП, *a* - символ из входного алфавита V, {5,6, … } – конечное состояние ФП.
5 - ая строка : accept – если входное слово допустимо для ДКА, not accept – если недопустимо. Далее писок состояний соответствующих входному слову в формате «{0,1};{4};{1,2,3};{5,7};{6};».

ИНСТРУКЦИЯ ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММЫ

Необходимо создать текстовый файл в папке cmake-build-debug, находящейся в папке проекта, и, пользуясь разделом «Формат входных и выходных данных», записать в него исходные данные НКА. Далее запустить программу через командную строку, задав имена созданного файла и файла для записи через аргументы командной строки. В результате работы программы в заданный файл для записи будут занесены данные ДКА и результат его эмуляции.

ЛОГИКА ПРОГРАММЫ

Задача детерминизации и эмуляции НКА решена при помощи 4 классов :

1)

```
//Функция перехода для недетерминированного автомата
class Transition
{
    state Out;   //начальное состояние ФП
    char Letter; //символ перехода
    state In;    //конечное состояние ФП
```

2)

```
//Функция перехода для детерминированного автомата
class FuncTransition
{
    set<state> Out;  //множество начальных состояний ФП
    char Letter;    //символ перехода
    set<state> In;   //множество конечных состояний ФП
```

3)

```
//Недетерминированный конечный автомат
class NonDeterAutomaton
{
    unordered_multimap<state, Transition> Data; //список функций перехода
    set<state> Begin;    //множество начальных состояний
    set<state> Final;    //множество конечных состояний
    int Num;              //количество состояний
    string InputWord;     //входное слово
```

4)

```
//Детерминированный конечный автомат
class DeterAutomaton
{
    unordered_multimap<set<state>, FuncTransition> Data; //список функций перехода
    set<state> Begin;              //начальное состояние
    set<set<state>> Final;       //множество конечных состояний
    int Num;                      //количество состояний

    string InputWord;             //входное слово для эмуляции ДКА
    list<set<state>> OutputList;  //результат эмуляции в виде списка  состояний ДКА
    bool Accept;                  //переменная, отвечающая за то, является ли входное
                                  //слово допустимым для ДКА
```

Ключевые действия :

1) `NonDeterAutomaton NKA;` - создание объекта НКА
2) `fromFile(NKA, file);` - инициализация НКА при помощи заданного входного файла
3) `NKA.eraseExtra();` - удаление из НКА «лишних» состояний, а т.е. состояний, к которым нельзя добраться из множества начальных состояний НКА, а также по которым нельзя перейти в множество конечных состояний НКА.
    - `NKA.wayToBegin();` - проверка существования для каждого состояния НКА пути от начального состояния
    - `NKA.wayToFinal();` - проверка существования для каждого состояния НКА пути до конечного состояния
    - `NKA.eraseFromBeginFinal();` - удаление “лишних” состояний из множеств начальных и конечных состояний
    - `Data.erase(…);` - удаление ФП , в которых учавствуют “лишние” состояния

4) `DeterAutomaton DKA;` - создание объекта ДКА
5) `DKA.determinization(NKA);` - инициализация ДКА (а следовательно и детерминизация) при помощи объекта НКА :

    - `DKA.InputWord = NKA.getInputWord();` - инициализация поля входного слова
    - `DKA.uniqueWays({NKA.Begin}, …)` – рекурсивный метод установки уникальных «путей» из какого-либо состояния ДКА
    - `DKA.reductionFinal({NKA.Final}, …)` – установка множества конечных состояний ДКА в соответствии с множеством конечных состояний НКА
    - `DKA.emulation(InputWord);` - эмуляция полученного ДКА по входному слову
6) `toFile(DKA, file);` - запись в заданный файл данных ДКА и результата эмуляции

ОЦЕНКА СЛОЖНОСТИ

Условные обозначения

- $N_н$ - количество всех состояний НКА, $N_н \geq 2$;
- $N_д$ - количество всех состояний ДКА, $N_д \in [2,\ \sum_{i=1}^{N_н} C_{N_н}^{i}\ ]$;
- V - мощность входного алфавита;
- B - количество начальных состояний НКА, $B \in [1, N_н]$;
- $F_н$ - количество конечных состояний НКА, $F_н \in [1, N_н]$;
- $F_д$ - количество конечных состояний ДКА, $F_д \in [1, N_д]$;
- $D_н$ - количество функций перехода НКА, $D_н \in [1, V \times N_н^2]$,
  так как каждое состояние НКА (всего $N_н$) может иметь количество функций перехода в каждое из $N_н$ состояний не больше, чем существует символов в алфавите (V);
- $D_д$ – количество функций перехода ДКА, $D_д \in [1, V \times N_д]$,
  так как каждое состояние ДКА (всего $N_д$) может иметь количество функций перехода не больше, чем существует символов в алфавите (V);
- K – мощность состояния ДКА, $K \in [1, N_н]$;
- S – длина входного слова.

- BEST – Лучший случай
- WORST – Худший случай
- BMW – Лучший/средний/худший случай

Алгоритмы НКА

- `setData(…)`
  - `наличие цикла – O(`$D_н$`)`
  - `Data.insert(…) – O(1)`
  -

`BEST – O(1)`

`WORST – O(`$V \times N_н^2$`)`

- `eraseFromBeginFinal(…)`
  - `Begin.find() – O(log B)`
  - `Begin.erase(…) – O(1)`
  - `Final.find() – O(log F)`

- Final.erase(…) – O(1)

BEST – O(log B + log F) = O(1)
WORST – O(log B + log F) = O(log $N_н$ + log $N_н$) = O(log $N_н$)

- wayToBegin()
  - set.insert() – O(1)
  - Data.equal_range(…) – O(1)
  - наличие цикла – O($\frac{D_н}{N_н}$)
  - set.find() – O(log $N_н$)
  - наличие рекурсии – O($N_н$)

BEST – O($\frac{D_н}{N_н} * \log N_н * N_н$) = O($\log N_н$)
WORST – O($D_н * \log N_н$) = O($V \times N_н^2 * \log N_н$)

- wayToFinal()
  - set.find() – O(log $N_н$)
  - Data.equal_range(…) – O(1)
  - наличие цикла – O($\frac{D_н}{N_н}$)
  - наличие рекурсии – O($N_н$)

BEST - O(log $N_н$ + $\frac{D_н}{N_н} * N_н$) = O(log $N_н$)
WORST - O(log $N_н + D_н$) = O(log $N_н + V \times N_н^2$) = O($V \times N_н^2$)

- eraseExtra()
  - наличие цикла – O(B)
  - wayToBegin()
    BEST – O($\log N_н$)
    WORST – O($V \times N_н^2 * \log N_н$)
  - наличие цикла – O($D_н$)
  - наличие цикла – O(F)
  - wayToFinal()
    BEST - O(log $N_н$)
    WORST - O($V \times N_н^2$)
  - Data.insert(…) – O(1)
  - наличие цикла – O($D_н$)
  - set.find() – O(log $N_н$)
  - eraseFromBeginFinal(…)
    BEST – O(1)
    WORST – O($N_н$)

- наличие цикла – O($D_н$)
- Data.erase() – O(1)

BEST – O(B * $\log N_н$ + $D_н$ + F * $\log N_н$ + $D_н$ + $D_н$) = O(B * $\log N_н$ + $D_н$ + F * $\log N_н$) = O(log $N_н$)
WORST - O(B * $\log N_н$ + $D_н$ + F * $\log N_н$) = O($N_н$ * $\log N_н$ + $V \times N_н^2$) = = O($V \times N_н^2$)

Алгоритмы ДКА

- reductionFinal(list<int> & m)
  - наличие цикла – O($F_н$)
  - наличие цикла – O($N_н$)
  - set.find() – O(log $N_н$)
  - Final.insert(…) – O(1)

BEST – O($F_н$ * $N_н$ * log $N_н$) = O($N_н$ * log $N_н$)
WORST - O($F_н$ * $N_н$ * log $N_н$) = O($N_н^2$ * log $N_н$)

- uniqueWays(…)
  - наличие цикла – O(K)
  - Data.equal_range(…) - O(1)
  - наличие цикла - O($\frac{D_н}{N_н}$)
  - Begin.find() – O(log B)
  - наличие цикла - O($\frac{D_н}{N_н}$)
  - Data.insert(…) - O(1)
  - set.find() – O(log $N_д$)
  - Data.count() – O(1)
  - наличие рекурсии – O($N_д$)

BEST – O(K * (1 + $\frac{D_н}{N_н}$ * log B) + $\frac{D_н}{N_н}$ * (1 + log $N_д$ + 1)) * $N_д$) =
= O($N_д$ * (K * $\frac{D_н}{N_н}$ * log B + $\frac{D_н}{N_н}$ * log $N_д$)) = O ($\frac{N_д}{N_н}$) = O ($\frac{\sum_{i=1}^{N_н} C_{N_н}^i}{N_н}$)
WORST – O($N_д$ *(K * $\frac{D_н}{N_н}$ * log B + $\frac{D_н}{N_н}$ * log $N_д$)) =
= O ($\sum_{i=1}^{N_н} C_{N_н}^i$ *(($N_н$ * $V \times N_н$ * log $N_н$ + $V \times N_н$ * log $\sum_{i=1}^{N_н} C_{N_н}^i$))
= O ($\sum_{i=1}^{N_н} C_{N_н}^i$ * $V \times N_н^2$ * log $N_н$)

- emulation()
    - InputWord.empty() – O(1)
    - Final.find() – O(log $F_д$)
    - наличие цикла – O($N_д$)
    - Data.equal_range(…) – O(1)
    - наличие цикла - O($\frac{D_д}{N_д}$)
    - push_back(…) – O(1)
    - Final.find() – O(log $F_д$)

BEST – O(log $F_д$ + $N_д$ * $\frac{D_д}{N_д}$ * log $F_д$) = O($D_д$ * log $F_д$) = O(1)

WORST - O($D_д$ * log $F_д$) = O(V * $\sum_{i=1}^{N_н} C_{N_н}^i$ * log $\sum_{i=1}^{N_н} C_{N_н}^i$)

- determinization(…)
    - uniqueWays(…)

      BEST – O ($\frac{\sum_{i=1}^{N_н} C_{N_н}^i}{N_н}$)

      WORST - O ($\sum_{i=1}^{N_н} C_{N_н}^i$ * V × $N_н^2$ * log $N_н$)
    - reductionFinal(…)

      BEST – O($N_н$ * log $N_н$)

      WORST - O($N_н^2$ * log $N_н$)
    - emulation()

      BEST – O(1)

      WORST - O(V * $\sum_{i=1}^{N_н} C_{N_н}^i$ * log $\sum_{i=1}^{N_н} C_{N_н}^i$)

BEST – O ($\frac{\sum_{i=1}^{N_н} C_{N_н}^i}{N_н}$)

WORST - O ($\sum_{i=1}^{N_н} C_{N_н}^i$ * V × $N_н^2$ * log $N_н$)

ОЦЕНКА ИСПОЛЬЗОВАНИЯ ПАМЯТИ

Алгоритмы НКА

- setData(…)
  BMW - O(1)

- eraseFromBeginFinal(…)
  - set::iterator – O(1)
  - Begin.find(…) – на месте
  - Begin.erase(…) – на месте
  - Final.find(…) – на месте
  - Final.erase(…) – на месте

  BMW - O(1)

- wayToBegin()
  - наличие рекурсии – $O(D_н)$
  - unordered_multimap::iterator – O(1)

  BEST - O(1)
  WORST - $O(D_н) = O(V \times N_н^2)$

- wayToFinal()
  - наличие рекурсии - $D_н$
  - set.find()
    BEST – O(1)
    WORST - $O(\log N_н)$
  - unordered_multimap::iterator – O(1)
  - set::iterator – O(1)

  BEST – O(1)
  WORST – $O(\log N_н \times V \times N_н^2)$

- eraseExtra()
  - map - $O(N_н)$
  - wayToBegin()
    BEST - O(1)
    WORST - $O(D_н) = O(V \times N_н^2)$

- unordered_multimap - $O(D_н)$
- wayToFinal()
  BEST – O(1)
  WORST – $O(\log N_н \times V \times N_н^2)$
- list<iterator> - O(1)
  BEST – O(1)
  WORST - $O(D_н)$
- eraseFromBeginFinal(…) – O(1)

BEST - $= O(N_н)$
MEDIAN - $= O(\log N_н \times V \times N_н^2)$

Алгоритмы ДКА

- reductionFinal(…)
  - set::iterator - O(1)
  - find(…) – на месте
  - Final.insert(…) – на месте

BMW - O(1)

- uniqueWays(…)
  - tryFinal(…) – O(1)
  - unordered_multimap<char, list<int>> - $O(K * \frac{D_д}{N_д})$
  - set::iterator - O(1)
  - Begin.find(…) – на месте
  - Data.insert(…) – на месте
  - наличие рекурсии – $O(N_д)$

BEST – $O(N_д * K * \frac{D_д}{N_д}) = O(1)$
WORST $O(N_д * K * \frac{D_д}{N_д}) = O(N_н * V * \sum_{i=1}^{N_н} C_{N_н}^{i})$

- emulation()
  - bool – O(1)
  - list<int> - O(K)
  - unordered_multimap::iterator – O(1)
  - list.push_back(…) – на месте
  - Final.find(…) – на месте

BEST – O(1)
WORST – O($N_н$)

- determinization(…)
  - set<set<…>> - O(K * $N_д$)
  - uniqueWays(…)
    BEST - O(1)
    WORST - O($N_н * V * \sum_{i=1}^{N_н} C_{N_н}^{i}$)
  - reductionFinal(…)– O(1)
  - emulation()
    BEST – O(1)
    WORST – O($N_н$)

BEST – O(K * $N_д$ + 1 + 1) = O(1)
WORST - O(K * $N_д$ + $N_н * V * \sum_{i=1}^{N_н} C_{N_н}^{i}$) =
= O($N_н$ * $\sum_{i=1}^{N_н} C_{N_н}^{i}$ + $N_н * V * \sum_{i=1}^{N_н} C_{N_н}^{i}$) = O($N_н * V * \sum_{i=1}^{N_н} C_{N_н}^{i}$)

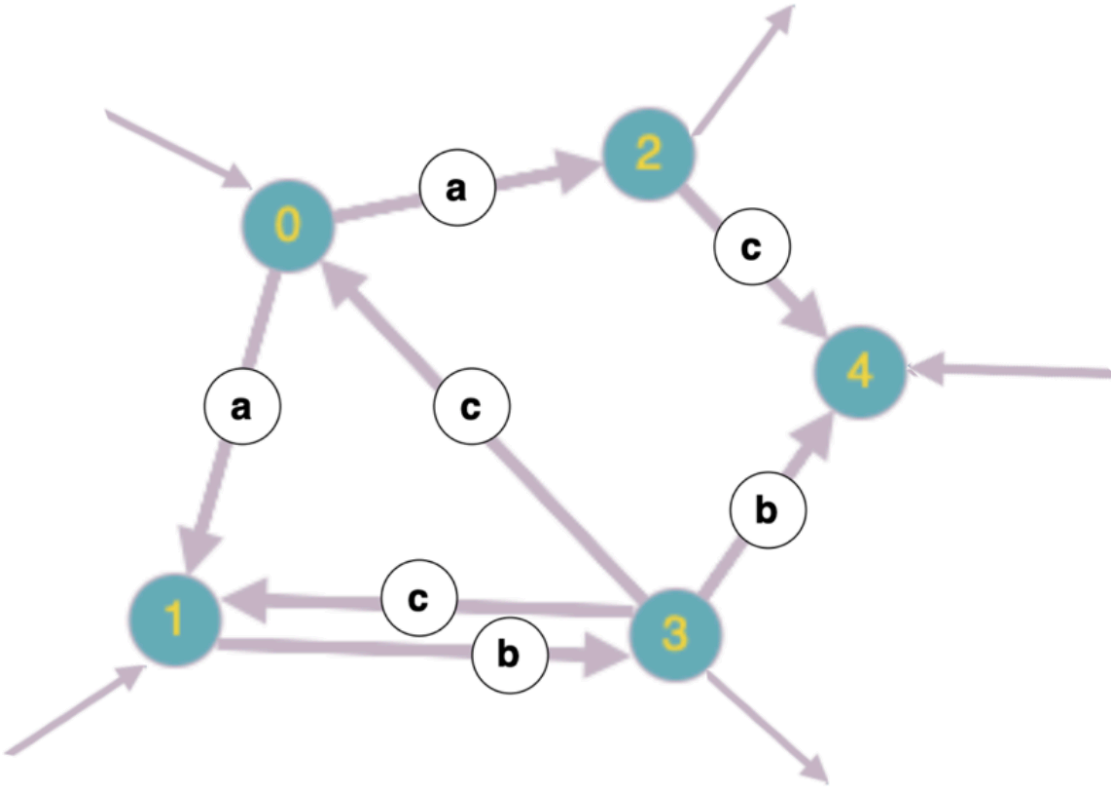
ОПИСАНИЕ ТЕСТОВ

1) test1

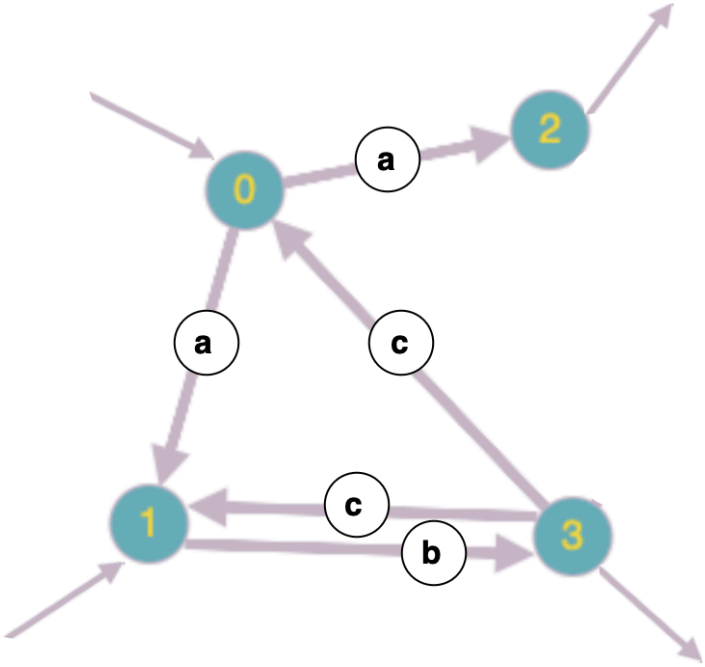
- Запись в файле test1.txt исходных данных НКА (см. раздел «Формат входных и выходных данных») :

```
5
0;1;4;
2;3;
0,a,2;0,a,1;1,c,3;3,c,0;3,b,1;2,c,4;3,b,4;
aaaacbaccaa
```

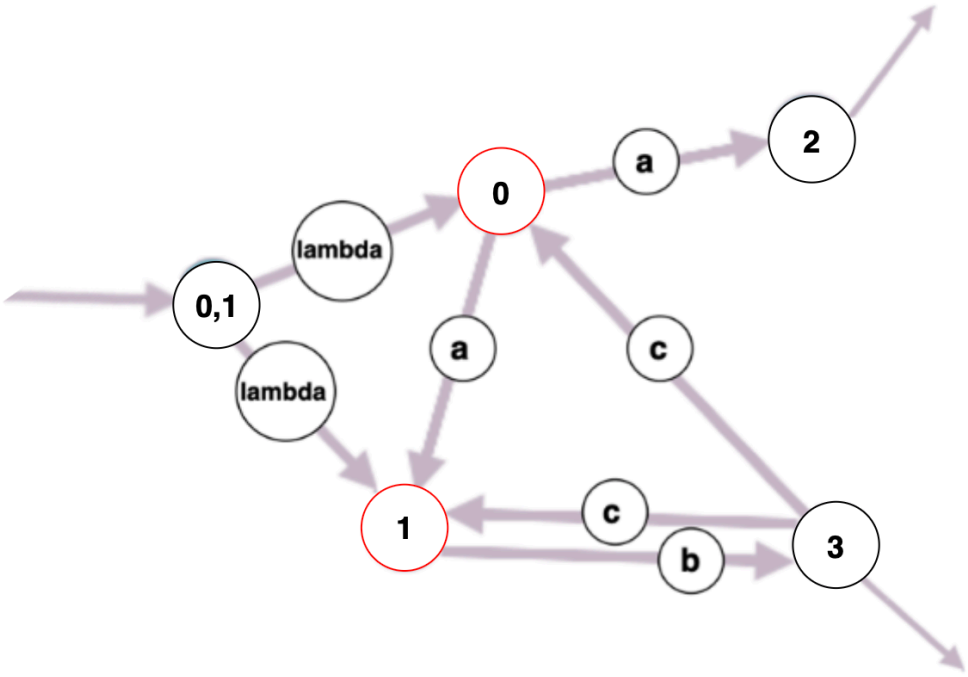
- Графическое представление заданного НКА



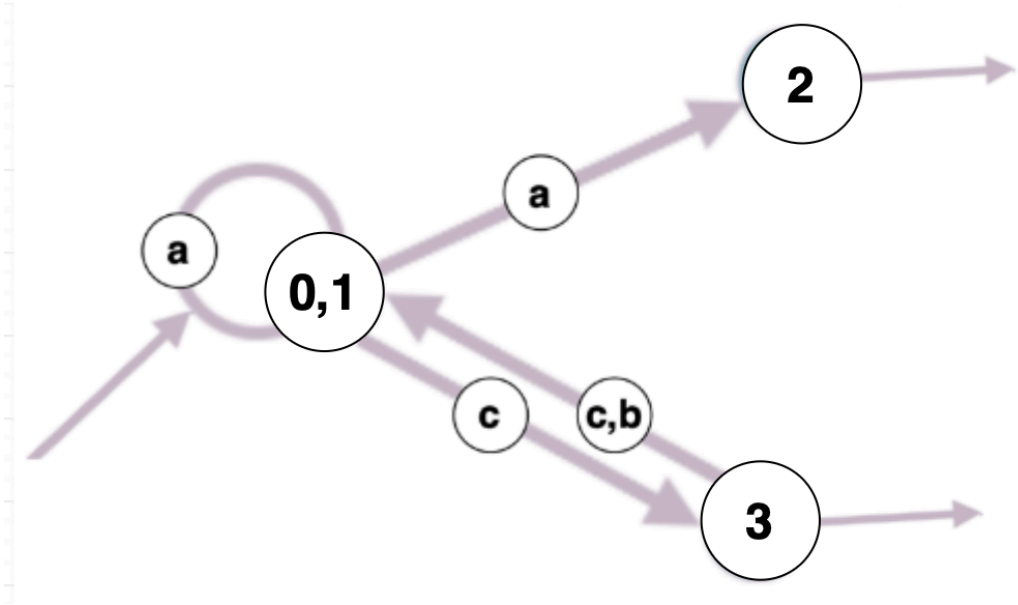
- Удаление из исходного НКА состояния 4, так оно является терминальным : в него можно попасть из других состояний и оно является начальным, но из него не существует выходных ФП :



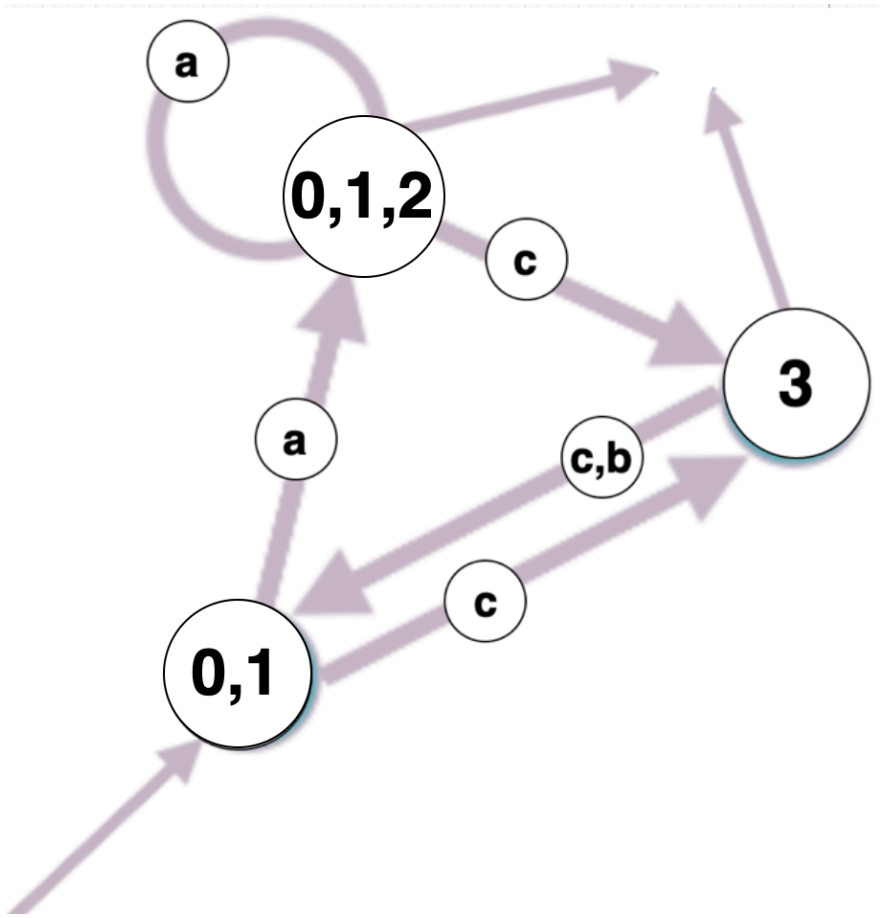
- Создание единственного начального состояния и соединение его с множеством начальных состояний λ – переходами (множество начальных состояний помечено красным цветом) :



- Удаление λ – переходов :



- Финальный этап детерминизации конечного автомата - установка уникальных «путей» из каждого состояния КА :



- Запись в файле test1.dat данных ДКА и результата эмуляции :

```
3
{0,1};
{3};{0,1,2};
{0,1,2},c,{3};{0,1,2},a,{0,1,2};{3},b,{0,1};{3},c,{0,1};{0,1},c,{3};{0,1},a,{0,1,2};
accept {0,1};{0,1,2};{0,1,2};{0,1,2};{0,1,2};{3};{0,1};{0,1,2};{3};{0,1};{0,1,2};{0,1,2};
```

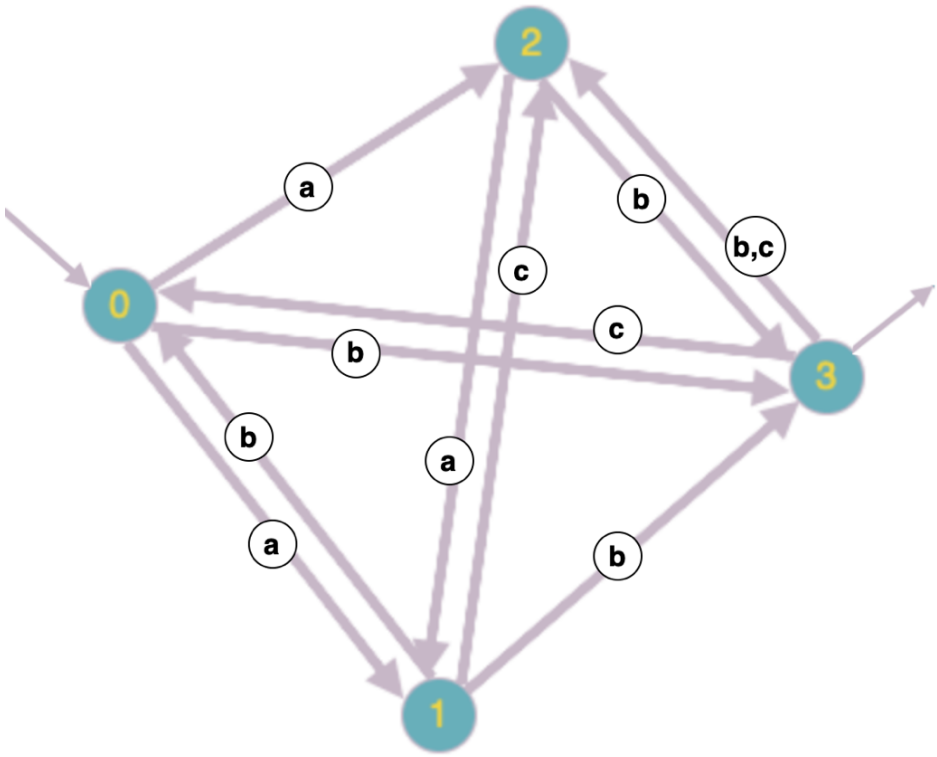
2) test2

- Запись в файле test2.txt исходных данных НКА :

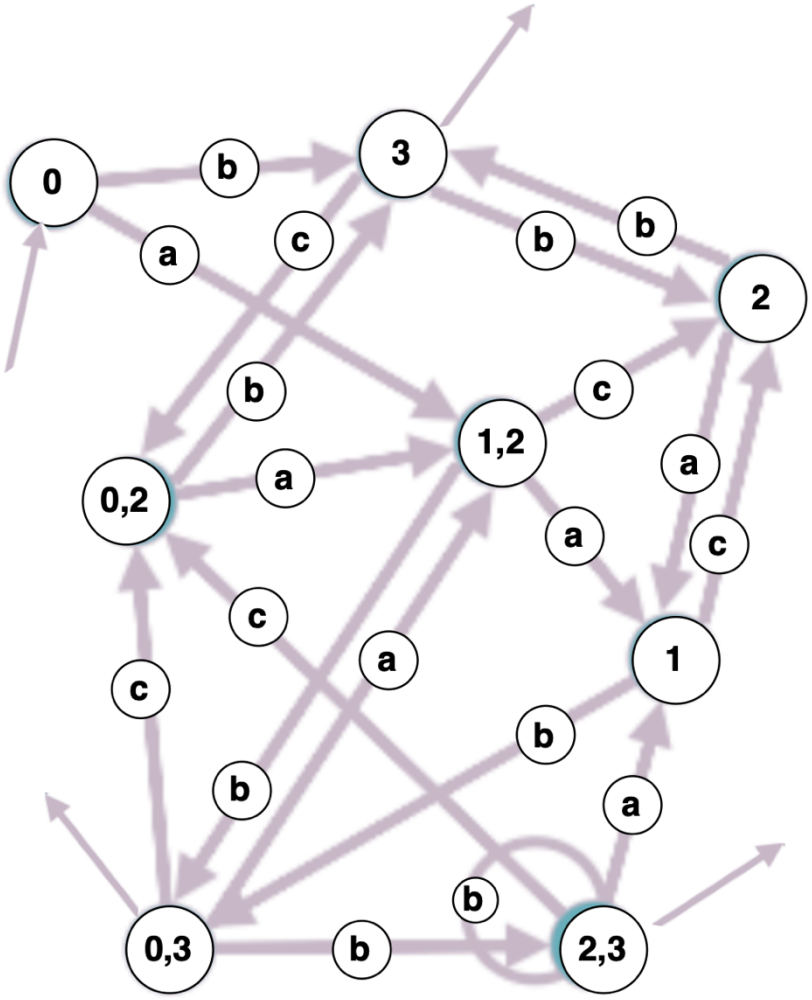
```
4
0;
3;
0,a,2;0,a,1;0,b,3;1,b,0;1,c,2;1,b,3;2,a,1;2,a,1;2,a,1;2,b,3;3,c,2;3,b,2;3,c,0;
aacbc
```

ФП «2,a,1» написана несколько раз для подтверждения того, что такого рода ошибки не влияют на работу программы, так как они не изменяют логики детерминизации.

- Графическое представление заданного НКА



- Так как в заданном НКА нет терминальных состояний и мощность множества начальных состояний равна 1, то можно сразу переходить к установки уникальных «путей» из каждого состояния :



- Запись в файле test2.dat данных ДКА и результата эмуляции для теста test2 :

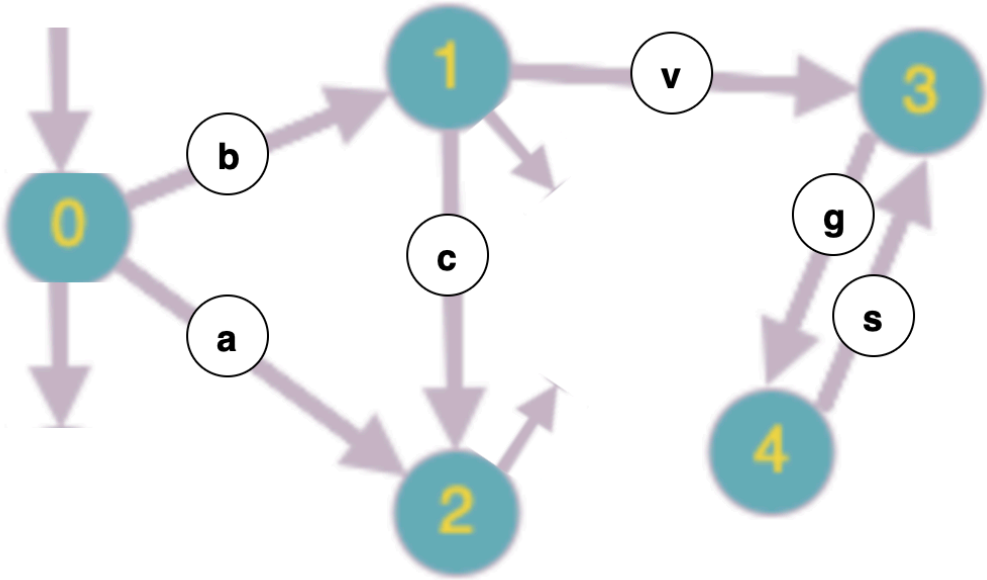
```
8
{0};
{3};{0,3};{2,3};
{2,3},c,{0,2};{2,3},b,{2,3};{2,3},a,{1};{0,2},b,{3};{0,2},a,{1,2};{0,3},c,{0,2};{0,3},b,
{2,3};{0,3},a,{1,2};{2},b,{3};{2},a,{1};{1},c,{2};{1},b,{0,3};{0},b,{3};{0},a,{1,2};{3},b,
{2};{3},c,{0,2};{1,2},a,{1};{1,2},c,{2};{1,2},b,{0,3};
not accept {0};{1,2};{1};{2};{3};{0,2};
```

3) test3

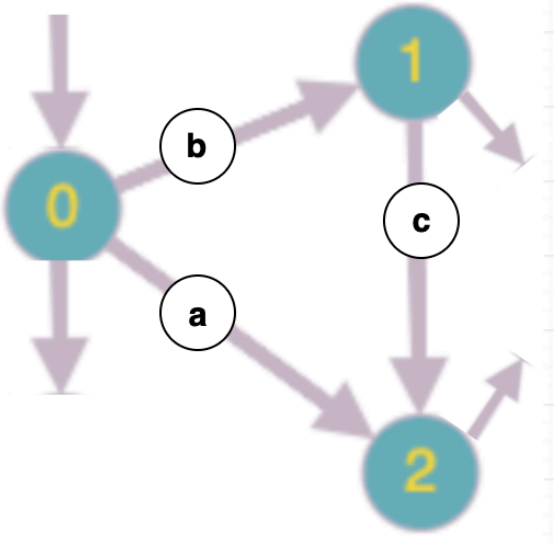
- Запись в файле test3.txt исходных данных НКА :

```
5
0;
0;1;2;
0,b,1;1,c,2;0,a,2;4,s,3;3,g,4;1,v,3;
bc
```

- Графическое представление заданного НКА :



- Заданный автомат является уже детерминированным, но в нем присутствуют 2 терминальных состояния (3, 4), которые необходимо удалить :



- Запись в файле test3.dat данных ДКА и результата эмуляции для теста test3 :

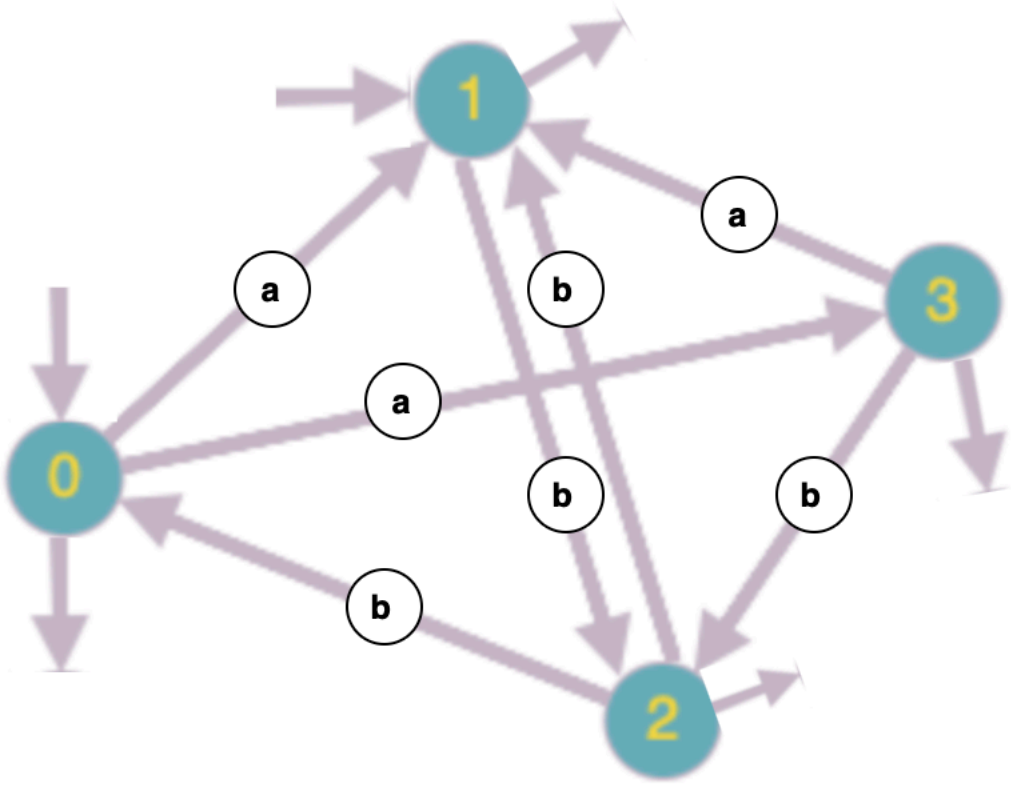
```
3
{0};
{0};{1};{2};
{0},a,{2};{0},b,{1};{1},c,{2};
accept {0};{1};{2};
```

4) test4

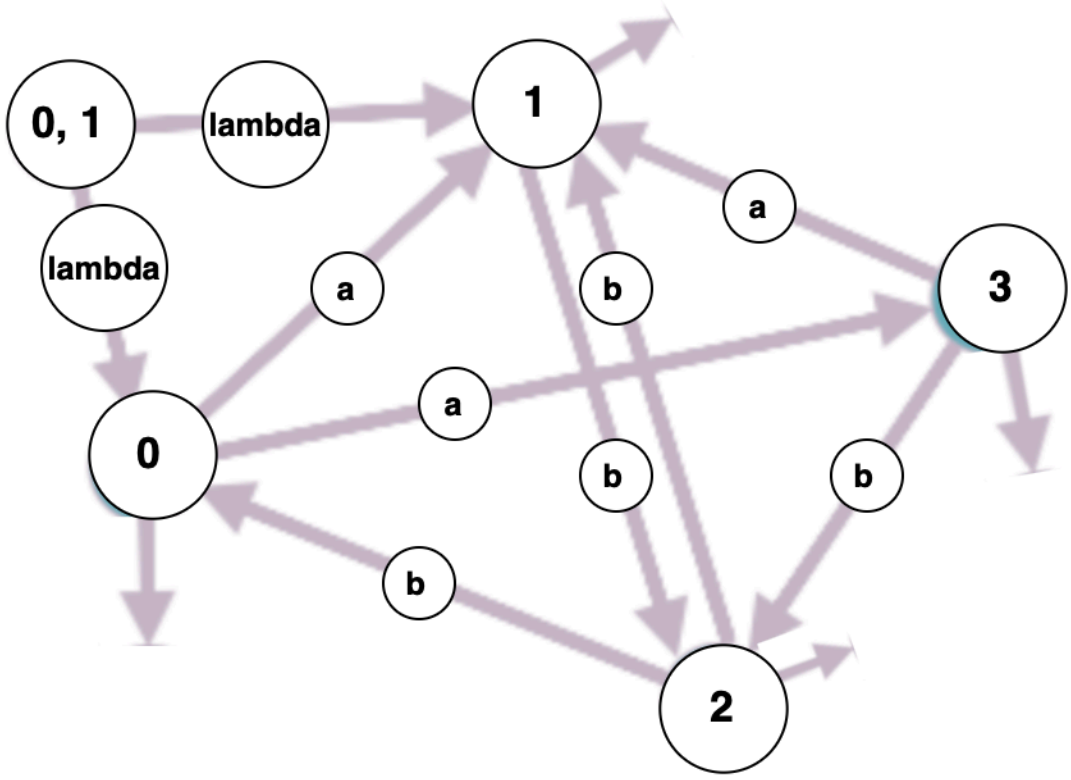
- Запись в файле test4.txt исходных данных НКА :

```
4
0;1;
0;1;2;3;
0,a,1;0,a,3;1,b,2;2,b,1;2,b,0;3,a,1;3,b,2;
abba
```

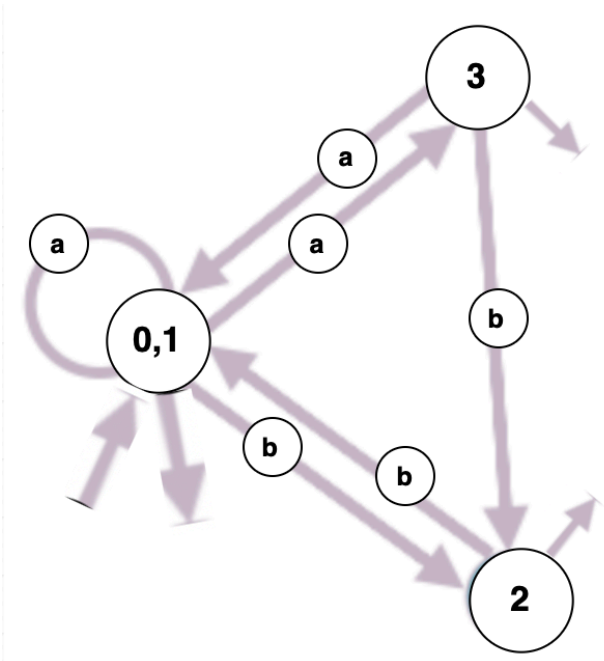
- Графическое представление заданного НКА :



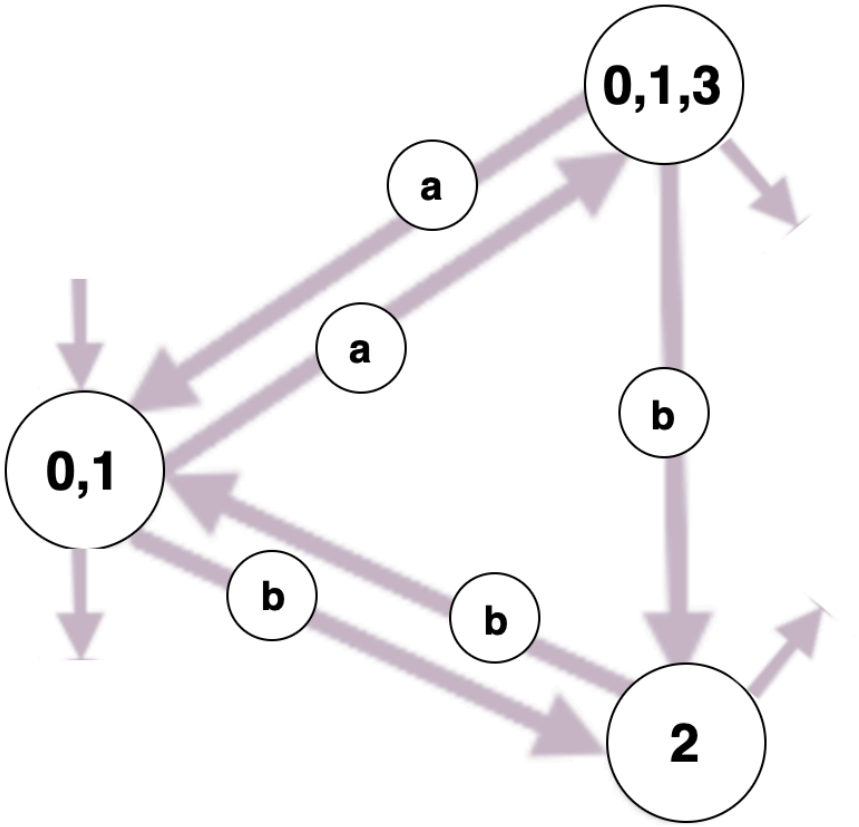
- Так как в заданном НКА 2 начальных состояния, их следует объединить в новое состояние, которое будет единственным начальным. И соединить новое начальное состояние с множеством начальных состояний НКА λ – переходами :



- Удаление λ – переходов :



- Финальный этап детерминизации конечного автомата - установка уникальных «путей» из каждого состояния КА. Так как в заданном НКА все состояния были конечными, в ДКА все состояния тоже конечные.



- Запись в файле test4.dat данных ДКА и результата эмуляции для теста test4 :

```
3
{0,1};
{0,1};{0,1,3};{2};
{2},b,{0,1};{0,1},b,{2};{0,1},a,{0,1,3};{0,1,3},b,{2};{0,1,3},a,{0,1,3};
accept {0,1};{0,1,3};{2};{0,1};{0,1,3};
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) https://ru.bmstu.wiki/Конечный_автомат
2) http://mathhelpplanet.com/static.php?p=determinizatsiya-konechnykh-avtomatov
3) https://neerc.ifmo.ru/wiki/index.php?title=Недетерминированные_конечные_автоматы
4) https://neerc.ifmo.ru/wiki/index.php?title=Детерминированные_конечные_автоматы
5) https://ru.wikipedia.org/wiki/Конечный_автомат#Детерминированность
6) https://msdn.microsoft.com/ru-ru/library/bb982522.aspx
7) http://graphonline.ru