# Краен Недетерминиран Автомат

Димитър Николаев Татарски

ФН 81818

Този документ описва разработката и работата с Краен Недерминиран Автомат реализиран на C++.

## Съдържание

Съдържание

Увод

Преглед на предметната част

Проектиране
- Автоматът
- Командния интерфейс

Реализация и тестване
- Тестване на NFA_CLI
- Тестови примери за програмата

Заключение

Използвана литература

# Увод

**Крайните Недерминирани Автомати** са прости сметачни машини, които разполагат с крайна памет и краен брой състояния.

Целта на този проект е да реализира такъв модел на **C++**. Проекта цели да ползва минимално стандартната библиотека. Реализирани са класовете **`DynamicArray<T>, String`** и структурата **`Pair<T1, T2>`**. Те се ползват, за да се реализира класа **NFA(**крайният недерминиран автомат). Помощни класове на NFA са структурата **Edge** и класа **State**.

**`DynamicArray<T>`** е масив с динамична памет. Той разполага с член функции за манипулиране на елементите си: добавяне на нов елемент, добавяне на нов уникален елемент, премахване на елемент.
Паметта, в която се пази редицата от данни (данните от тип T) бива автоматично презаделяна ако се напълни, или изпразни прекалено много. Ползвателя на **`DynamicArray<T>`** не трябва да се притеснява за заделянето на памет.

**`class String`** Е **`DynamicArray<char>`**. Той разполага с всички публични член функции на **`DynamicArrray<T>`**, но ги разширява с полезни функции за работа със символни низове.

**`struct Pair<T1, T2>`** пази в себе си 2 данни - едната от тип T1, другата от тип T2.

**`struct Edge`** пази в себе си индекс(int) и буква(char). Тази структура моделира “преход” в автомата.

**`class State`** моделира състояние на автомата. Той пази името на състоянието, дали състоянието е начално и дали състоянието е крайно.

# Преглед на предметната част

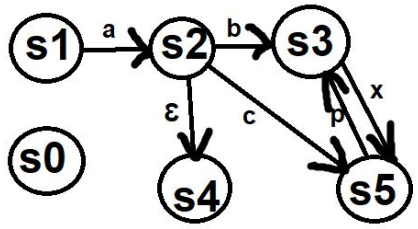
Крайния недетерминиран автомат ще представим като краен ориентиран граф, чиито ребра са именувани със символ.
Ще разполагаме с масив от състоянията(`DynamicArray<State>`). Това ще са върховете на графа.
Ще разполагаме и със списък на съседство(`DynamicArray<DynamicArray<Edge>>`). Така ще моделираме насочените именувани ребра. По този начин ще определяме функцията на преходите на крайния недерминиран автомат. За празни преходи ще се ползва затапващия символ.

Азбуката на автомата ще се определя от множеството от букви по ребрата на графа.

На **Фигура 1** е илюстриран примерен автомат и как class NFA ще го пази в паметта си:

**DynamicArray<State> = [s0,s1, s2, s3, s4, s5]**

```
DynamicArray<DynamicArray<Edge>> = [
  [],
  [Edge(2, 'a')],
  [Edge(3, 'b'), Edge(4, '\0'), Edge(5, 'c')],
  [Edge(5, 'x')],
  [],
  [Edge(3, 'p']
]
```



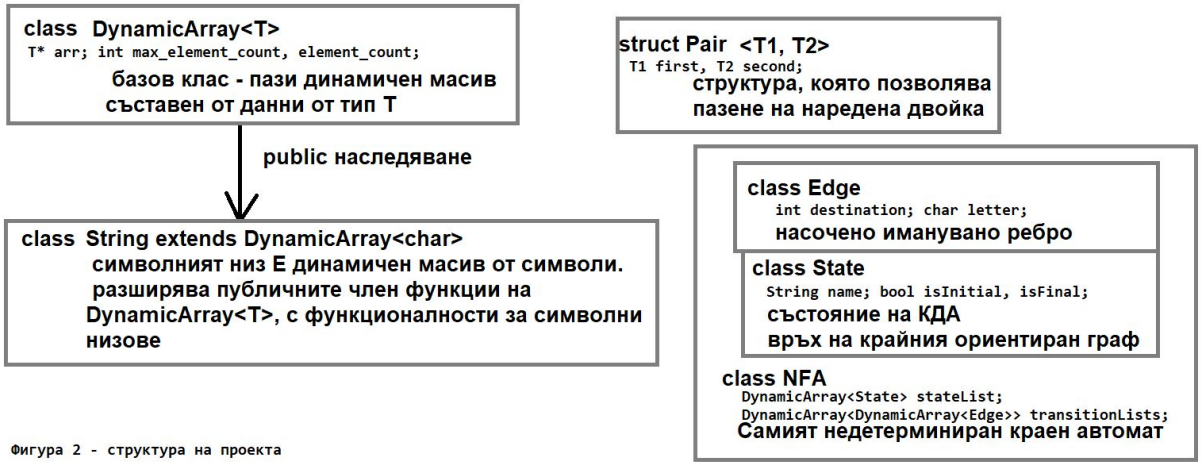

**Фигура 1**

Върху ориентирания граф ще бъдат прилагани няколко алгоритъма. Те са алгоритъм за принадлежност на дума, алгоритъм за детерминизация, за обединение, за сечение, за допълнение, за намиране на резултат от функцията на преходите и още няколко алгоритъма.

# Проектиране

## Автоматът

Фигура 2 показва структурата на класовете и структурите в проекта, без командния интерфейс. Той ще бъде описан в отделна глава.




Фигура 2 - структура на проекта

Удачно е публичното наследяване на DynamicArrar<char> от String, защото всички публични член функции и данни ползвани от DynamicArray<char> са приложими върху String.

`NFA` разполага с конструктор, който прави автомат от регулярен израз.
`NFA(const String& regex).`
Този конструктор строи автомат от регулярен израз в следния формат.

```
R = {д у м а} | (R) | (R)* | (R).(R) | (R)+(R)
```

## Командния интерфейс

Реализиран чрез `class NFA_CLI`. Пази в себе си `DynamicArray<NFA>`.
Работи като филтър за низове. Има единствена публична функция `readCommand`.
`String readCommand(const String& other);`

# Реализация и тестване

Не бяха разработвани unit tests успоредно с разработката на проекта. Това усложни реализацията на някои класове - пишейки един клас, трябваше да се прехвърля в друг и да премахвам моя грешка там.

Тестването на NFA_CLI може да се случва директно. Инициализира се чрез единствения си конструктор и се извиква read_command с различни команди.

Идентификаторите на автоматите са всъщност индекса им в списъка от автомати на NFA_CLI.

## Тестване на `NFA_CLI`

Следния код тества `NFA_CLI`. Той работи, докато не бъде подадена командата exit.

```
NFA_CLI cli1;
String result;
while (!(result == (String)"exiting...")) {
      char command[100];
      cin.getline(command, 100);
      result = cli1.readCommand(String(command));
      cout << result << endl;
}
```

## Тестови примери за програмата

| **Вход:**<br>reg aaa<br>reg bbb<br>union 0 1<br>print 2<br>recognize 2 aaabbb<br>recognize 2 aaa<br>recognize 2 bbb | **Изход:**<br>0<br>1<br>2<br>Описание на автомат с индекс 2<br>0<br>1<br>1 |
|---|---|
| **Вход:**<br>reg ((a+b))*<br>print 0 | **Изход:**<br>0<br>Описание на автомат с индекс 0 |

# Заключение

Има много насоки, по които все още може да се развива. Списък включващ, някои от тези насоки е:

1) Абстракция на класа NFA_CLI. Поддръжка на повече команди.
2) Още алгоритми, които работят върху Краен Недерминиран Автомат.

# Използвана литература

Използвал съм знания от курса по Езици, Автомати и Извислимост.