Недетерминиран краен автомат

Божидар Димитров

1 Увод

1.1 Описание и идея на проекта

Проектът представлява програма с конзолен интерфейс. В нея потребителят може да получи информация във връзка с краен недетерминиран автомат - дали езикът, който разпознава, е празен, дали дума принадлежи на него и т.н. За всяка операция са използвани отделни методи, което прави кода по-практичен и дава възможност части от него да бъдат използвани и в други програми.

1.2 Цел и задачи на разработката

Проектът дава възможност на потребителя да извършва основните операции с недетерминирани крайни автомати, като освен това може да проверява дали езикът на автомата съдържа конкретна дума, дали е празен, дали е безкраен. Програмата позволява работа с файлове - записване на автомат във файл, прочитане на автомат от файл и като цяло основните операции - отваряне, писане, записване - както на същото място, така и на друго, затваряне.

1.3 Структура на документацията

Първо ще Ви запозная с понятия и алгоритми, които ще използвах в процеса на изпълнение на задачата. След това ще представя основни трудности, с които се сблъсках, както и моята интерпретация на проблема, заедно с идеите ми за неговото решаване. Ще разгледам и части от решението, които представят съществена част от цялостния код. Накрая ще обобщя постигнатите резултати и ще дам насоки за усъвършенстване.

2 Преглед на предметната област

2.1 Основни понятия, концепции и алгоритми, които ще бъдат използвани

Дефиниция (Недетерминиран краен автомат) — петорка $A = (\Sigma, Q, \delta, s, F)$, където:

- Σ е крайно множество от символи, което ще наричаме *азбука*;
- Q е крайно множество от състояния;
- $\delta \colon Q \times \Sigma \cup \{\varepsilon\} \to 2^Q$ е *функция на прехода*;

- $s \in Q$ е *начално състояние*;
- $F \subseteq Q$ е множество от *финалния състояния*.

Дефиниция (Етикетиран, ориентиран мултиграф) — това е тройка (Σ, V, E), където:

- Σ е множество от символи(етикети);
- V е множество от *върхове*;
- $E \subseteq V \times V \times (\Sigma \cup \{\varepsilon\})$ е множество от *ребра*.

Дефиниция (Път, цикъл, примка) — *Път* наричаме множество от върхове $v_1, v_2, \ldots, v_n$ такова, че (v_i, v_{i+1}, c_i) е ребро за всяко $i \in [n-1]$ за някои $c_i \in \Sigma \cup \{\varepsilon\}$. *Цикъл* наричаме път, в който първият и последният връх съвпадат. *Примка* наричаме цикъл, съставен само от едно ребро.

Дефиниция (Граф на автомат) — за всеки автомат $A = (\Sigma, Q, \delta, s, F)$, построяваме граф $G(A) = (\Sigma, V, E)$, като:

- $V \equiv Q$;
- $(v_1, v_2, c) \in E \iff \delta(v_1, c) \ni v_2$.

Дефиниция (Език на автомат) — нека A е автомат и $G(A)$ е неговият граф. Казваме, че дума $w \in \Sigma^*$ се *разпознава* от A тогава и само тогава, когато съществува път $s, v_1, \ldots, v_n$, където $v_n \in F$, като w се получава от последователно записване на символите върху ребрата на пътя. *Език* на автомат наричаме

$$L(A) = \{w \in \Sigma^* : w \text{ се разпознава от } A\}$$

Използвани алгоритми:

- **Breadth First Search(BFS)** – това е обхождане на граф в широчина. При фиксиран връх v алгоритъмът ни дава информация за всеки връх u дали има път, който започва във v и завършва в u.
- **Reverse Polish Notation**(обратен полски запис) – решава алгебричен израз, отчитайки приоритета на операциите. В случая на калкулатор става дума за числа, събиране, умножение и степенуване, а в нашия случай – за автомати, обединение, конкатенация, звезда на Клини.
- **Rabin-Scott Powerset Construction** – това е алгоритъм, позволяващ ни да трансформираме недетерминиран автомат в детерминиран; в новия автомат множеството от състояния представлява степенното множество на множеството от финални състояние в началния автомат.
- За командата recognize използваме двумерен масив used[vertices][word.size()+1], като used[v][pos] ни дава информация дали има път от 0 до v, който записва префикса от първите pos символа на word. Стойностите попълваме рекурсивно.

2.2 Дефиниране на проблеми и сложност на поставената задача

Проблемите са основните неща, които ни интересуват за автоматите - кои думи се разпознават от тях, дали са краен брой. Сложността идва основно от алгоритмите, с които трябва да се справим. Освен горепосочените, използвахме двумерен BFS в търсене на отговор на въпроса дали дадена дума принадлежи на езика на автомат. Разбира се, използвахме и основните алгоритми за регулярни операции с автомати - обединение, конкатенация, звезда на Клини.

2.3 Подходи, методи за решаване на поставените проблемите

За решаването на проблема използваме структура Edge, която съдържа в себе си две цели числа - краищата на реброто, и един символ, както и клас Machine, в който отразяваме основните характеристики на автоматите, както и методите. Работим с графа на автомата, като него дефираме чрез:

- string name – това е идентификаторът на автомата;
- vector<vector<Edge> > – списъците на съседите на всеки връх;
- vector<Edge> – всички ребра на графа;
- int vertices – броя на върховете в графа;
- vector<int> final – списък с върховете, отговарящи на финалните състояния.

Както личи, върховете са целите числа от 0 до vertices−1, като считаме, че началното състояние $s = 0$, $\varepsilon = \_$, $\emptyset = /$.
За работата с файлове използваме клас File_Container. Използаме методи от динамичното програмиране, рекурсия.

2.4 Потребителски изисквания и качествени изисквания

За работа с програмата е нужен единствено достъп до компютър. Всички команди се появяват при нужда от помощ посредством командата <help>. След въвеждането ѝ потребителят получава инструкции за начина на използване на останалите команди, които трябва да бъдат спазвани, за да може приложението да работи.

3 Проектиране

3.1 Обща архитектура – ООП дизайн

Както споменахме, имаме една структура Edge и два класа – File_Container и Machine. Освен методите, директно отговарящи за конкретно изпълнение на команда, имаме и такива, които са спомагателни. Някои от тях са:

- void fillMatrix() – попълва списъците на съседите;
- vector<bool> BFS(int v) – намира достижимите върхове от v, като връща вектор с vertices на брой елемента, като BFS[i] е 1, ако i е достижим от v, и нула иначе.
- vector<bool> _closure(int v) – намира т.нар. празна обвивка на v, т.е. кои върхове са достижими от v посредством единствено ε-преходи;
- void changeStart(int start) – променя началното състояние от 0 на start;
- Machine copy() – създава копие на автомата.

3.2 Диаграми

```
Machine create_by_word(string t)
{
    if(t=="/") return Machine("", {}, 1, {});
    Machine res("", {}, t.size()+1, {(int)t.size()});
    for(int j=0; j<t.size(); j++)
    {
        res.getEdges().push_back({j, j+1, t[j]});
    }
    return res;
}
```

Тук е представена една от ключовите стъпки в генерарирането на автомат от регулярен израз – това е така нареченото дъно на рекурсията, когато регулярният израз е просто една дума.

```
string Concat(Machine& m1, Machine& m2, vector<Machine>& m)
{
    Machine m4 = m2.copy();
    m4.shift(m1.getVertices());
    int v = m1.getVertices() + m4.getVertices();
    vector<Edge> e;
    for(Edge E:m1.getEdges())
    {
        e.push_back(E);
    }
    for(Edge E:m4.getEdges())
    {
        e.push_back(E);
    }
    vector<int> f;
    for(int F:m1.getFinal())
    {
        e.push_back({F,m1.getVertices(),'_'});
    }
    for(int F:m4.getFinal())
    {
        f.push_back(F);
    }
    string n = "(" + m1.getName() + "_._" + m4.getName() + ")";
    m.push_back(Machine(n,e,v,f));
    return n;
}
```

```
Machine star()//звезда на Клини
{
    fillMatrix();
    Machine ans = this->copy();
    for(int i:final)
    {
        ans.edges.push_back({i,0,'_'});
    }
    ans.vertices++;
    ans.final.push_back(ans.vertices-1);
    ans.edges.push_back({0,ans.vertices-1,'_'});
    ans.name = name + "*";
    return ans;
}
```

Тук представяме още две важни конструкции – конкатенация и звезда на Клини.

4 Реализация, тестване

Кодът на някои от методите/функциите бе пренаписван няколко пъти, за да може да работи правилно във всички случаи. Например в метода isFinite() отначало бях пропуснал да добавя изискване символът на реброто да не е празният символ. Отделно, първоначално бях пропуснал частта с премахването на епсилон-преходите в трансформатора. Два текстови файла с примерни автомати са прикачени в хранилището. Отделно програмата позволява най-разнообразни тестове.

5 Заключение

5.1 Обобщение на изпълнението на началните цели

Не успях да изпълня задачата с рисуването, както и напълно работещ мутатор – не успях да реализирам премахването на ε-преходите, въпреки че имам следната идея: за да ги премахнем, то за всеки връх v разглеждаме неговата празна обвивка, което възможно благодарение на метода _closure, приложен върху v. В новия граф имаме, че (i, j, s) е ребро тогава и само тогава, когато съществуват върхове x и y такива, че:

- _closure$(i)[x]$ = true;
- _closure$(y)[j]$ = true;
- $(x, y, s) \in$ edges.

Началното състояние остава 0, а новите финални са тези i, които в своята празна обвивка съдържат някое от старите финални състояния.
Въпреки тези пропуски смятам, че се справих с всички останали проблеми – обединение, конкатенация, разпознаване на дума от автомат, които бяха същевременно интересни и трудоемки.

5.2 Насоки за бъдещо развитие и усъвършенстване

Смятам, че би било интересно да се добави метод, който по даден автомат генерира регулярен израз, отговарящ на разпознавания от автомата език. Също така задачата с рисуването остава в графата с бъдещите подобрения на тази програма. Също би било интересно след изобразяването на самия автомат при подаване на дума, ако тя се разпознава от него, то чрез визуализация да се показва пътя, съответстващ на нейното извеждане.