Детерминиран Краен Автомат

Връх в автомата и връзка към друг такъв се представя чрез следната структура:

struct Node

{

bool final = 0;

vector < pair< char , Node<T>\* > > connections;

};

Всеки Node има булева променлива, която показва дали той е финален и също вектор от std::pair, който в себе си държи връзка към друг Node и съответната буква която ги свързва;

В структурата Node са реализирани следните функции:

bool already\_printed(const vector<Node<T>\*> , const Node<T>\*);

* Помощна функция която казва дали даден връх се съдържа във вектор от върхове;

Node<T>\* get\_node(const char);

* Помощна функция връщата върха който е свързан с дадения с бъквата която е подадена като аргумент, ако такъв няма се връща nullptr;

void get\_nodes\_with\_epsi\_or\_quest(vector<pair<char,Node<T>\*>>& , const char);

* Функция която поема предварително дефиниран вектор, и буква. Тази функция добавя към вектора всички съседи които са свързани към дадения или с конкретна буква, или с епсилон преход или с символ за произволен символ. Като към резултата се добавя и с каква буква или символ е свързан;

void add\_node(const char a,const bool fin);

* По подадена буква и булева променлива, тази функция създава нов съсед с дадената буква. Ако такъв съществува, тази функция няма да промени неговото свойство за финалност;

void print(int j,vector<Node<T>\*>printed);

* Функция която приема int и вектор. Функцията работи като за всеки връх гледа дали е финален или не, в зависмост от това се изпечатва различна представа за върха. След като върха бива напечатан, той се добавя към вектор и се подава същата функция на неговите деца. Целта на вектора е да не се случи функцията да изпадне в бездънна рекурсия при печатането, ако има цикъл. Като ако в края има цикъл се принтира –loop, за да го индикира. Int-a служи за отстъп. Колкото по навътре отива функцията, толкова отстъпа ще расте и принтирането ще е по-прегледно;

void print\_dot(vector<Node<T>\*>printed,ofstream& myfile);

* Абсолютно същата идея като горната, само дето всичко се принтира в .dot файл;

bool recognise(string a);

* По подадено начало и string, тази функция казва дали следкато бъде изчетен string-a ще се намира във финално състояние, ако не или ако по пътя се разбира че даден връх не съществува, естествено се връща false;

Node<T>\* build\_cycle\_help(string abc);

* По подаден string тази функция създава нова клонка в автомата, без да създава последната връзка, тя връща предпоследния връх, като ако предварително е бил създаден някой връх, тя няма да промени детермираността на автомата.

void build\_cycle(string abc);

* Тази функция извиква горната, след което получения връх бива зашит към върха от който е извикана дадената функция;

Node<T>\* get\_node(string a);

* По даден string тази функция връща върха който е свързан следвайки символ по символ string-a, ако по пътя се окаже че дадена връзка не съществува, функцията връща nullptr;

void get\_all\_nodes(vector<Node<T>\*>& all);

* Тази функция бива извикван с предварително дефиниран вектор. Тя има за цел да вкара във вектора всеки връх който е негово дете, като е оптимизирана, ако има цикъл да добавя даден връх само веднъж.

void get\_all\_nodes\_special(vector<pair<bool,Node<T>\*>>& all,int);

* Тази функция има за цел да върне всички върхове които са свързани с даден връх като също казва дали даден връх е част от цикъл, int-a служи да каже колко е отдалечен от началото даден връх.

int get\_index(vector<Node<T>\*>a,Node<T>\* b);

* Служи да каже на кой индекс е даден връх, като ако не съществува, функцията връща -1.

Node<T>\* copy\_nodes\_help(vector<Node<T>\*>&all,vector<Node<T>\*>&old);

* Тази функция има за цел да копира висчки върхове с техните връзки и да ги върне като вектор. За всяко дете на върха от който бива извикана се проверява дали е край на цикъл, ако е то копието бива зашито с копието на оригиналния връх от който почва цикъла, ако не то създава се копие на текущия връх, вика се функцията за следвашия връх от оригинала, и се връща създаденото копие.

void copy\_nodes(vector<Node<T>\*>&all,vector<Node<T>\*>&old);

* Главната функция която вика горната след като започне, защото тя няма за цел да връща връх, а само да пълни даден вектор.

bool to\_izraz\_help(vector<string>&,vector<Node<T>\*>,vector<Node<T>\*>&);

* Тази функция когато стигне до връх който е краен добавя към вектор №1 празен стринг и ако има свързани към него върхове се извиква за тях. Така буква по буква се конструират изразите които изграждат даден автомат, само че отзад напред. Вектор №2 служи да не се окаже функцията в цикъл, а вектор №3 е такъв че когато бива извикана същата функция за съсед на конкретен връх, ако този връх е бил в цикъл той ще бъде добавен в вектора, така когато самия връх се види, той ще знае че след него има цикъл и ще сложи скоба преди себе си.

bool already\_printed\_special(const vector<pair< bool , Node<T>\* >>p,const Node<T>\* curr);

* Гледа дали даден връх се среща във вектор от pair< bool , връх >. Тази функция се ползва само от get\_all\_nodes\_special.

int get\_index\_pair(const vector<pair< bool , Node<T>\* >>,Node<T>\* b)

* Връща индекса на който се среща даден връх, ако той го няма, връща -1. Тази функция се ползва само от get\_all\_nodes\_special.

Класа Автомат, чрез който се създава даден такъв има следната реализация:

class Automa

{

private:

Node<T>\* start = nullptr;

};

В себе си той държи само Noda-a който е началото на автомата.

В него са реализирани следните функции за работа с него:

void add\_node(string path,char concetion,bool fin);

* По подаден път фунцкията отива в дадения връх и създава нов кото е свързан с него с буквата connection и се дефинира дали той е финален чрез fin. Ако такъв връх е съществувал, той няма да бъде променен. Ако по пътя даден връх не съществува той няма да бъде създаден. Тази функция има специален случай ако start e nullptr и path e “”, то тогава бива създавано началото на автомата, като fin определя дали то е финално (или Автомат : [] или Автомат : [f] )

void print();

* Извиква print функцията за върха start, като предварително е дефинирал вектор който да спомага с циклите. Int-a за print функцията бива зададен с 0, защото в началото отстъпи няма.

bool recognise(string a);

* Проверява дали дадена дума принадлежи на автомата, като извиква recognise() за върха start.

void add\_cycle(string path,string cycle);

* По подадена дума за начало (path) функцията отива на дадения връх, след което създава цикъл който започва и свършва в него и е с път (cycle). Като ако думата path не се разпознава, този цикъл няма да бъде създаден.
* Пример: Авт []-a->[] ; add\_cycle(“ab”,”a”) -> Нищо няма да създаде, защото няма такъв път.

void print\_dot(string name);

* По подадено име за файл тази функция създава файл (.dot) с това име и го пълни с това което генерира print\_dot функцията дефинирана за върхове. След това се създава и готов файл с име <name>\_converter.cmd, който ако бъде отворен дадения файл ще бъде конвертиран към <name>.pdf. Използва се Graphviz за целта.

Node<T>\* node\_copy() const;

* Копира върховете на даден автомат и връща първия такъв (началото на автомата-копие).

Automa<T>& operator+(const Automa& t);

* Оператор + на ав1 и ав2 създава нов автомат, който има в началото @ преход към началата на ав1 и ав2.

void concat(Automa& t1,Automa& t2);

* Тази функция взима всички върхове на t1 и към всеки финален прави @ преход към началото на t2.

int size();

* Връща броя на всички върхове в автомата. Тя вика get\_all\_nodes() за началото и накрая връща размера на вектора.

Automa& operator=(const Automa& t);

* Нормален оператор за присвояване ☺

Node<T>\* copy\_automa(Automa& other);

* Функция копираща цял автомати връщаща неговото начало.

void copy(Automa& other);

* Функция която копира даден автомат към този към който е извикана функцията;

void to\_automa(string izraz);

* Функция приемаща регулярен израз ( string ) който бива преобразен до автомат.

void to\_automa\_help(string izraz,Node<T>\* lst,vector<Node<T>\*>&);

* Главната функция за този проект. Тя приема израз, връх от който да се строи и вектор държащ построените върхове. Идеята е взимайки израз да го разбие на под изрази. След което ако в даден подизраз има скоби той бива съжо допълнително подразбит. Целта е всеки израз да бъде максимално разбит за да може по-лесно строене на автомат. Автомата не поддържа сечение.

void read(string file\_name);

* Приема име на файл и изкарва на конзолата редовете които се приемта от автомата за който е извикана тази функция. В файла нов ред без нищо в него се счита за ‘@’.

void completion();

* Прави допълнение на автомата. Извиква get\_all\_nodes() и след това всяко финално става нефинално и обратно.

void delete\_automa();

* Извиква get\_all\_nodes() след което изтрива всички върхове и прави start = nullptr,стига да е имало нещо за начало за триене.

void to\_izraz();

* Извиква to\_izraz\_help() за start върха, след което принтира всички изрази които биват генерирани, като празен израз се записва като ‘@’.

void check\_language();

* Проверява дали езика на даден автомат е празен, краен, пълен или неопределен (безкраен, но не пълен).

За автомата специално е дефиниран << оператора като,

Automa t;

t << ”abc”;

би казало дали думата “abc” се разпознава от автомата.

При стартиране на програмата, въведете символа ‘?’ от клавиатурата, за да видите възможните опции за работа с автомата.