ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

КУРСОВА РАБОТА

Дисциплина: „Формални езици и езикови процесори”

тема: Графична имплементация на евристичния метод на Дейкстра

Изготвил:
Васил Стойчев Стойков
Фак. № 121222094
Група: 40
III курс, КСИ
e-mail: vstoykov@tu-sofia.bg

София, 2024

Въведение

Евристичният алгоритъм да Дейкстра служи за преобразуване на математически изрази от инфиксен към постфиксен запис. Това се осъществява чрез така наречения стек на операторите, като осен него другите елементи са входната лента и изходната лента. Съгласно установени правила в стека се запазват и извеждат операторите от математическия израз, като след извеждане те попадат в изходната лента. Операндите от друга страна се записват директно във изходната лента, като те могат да бъдат както числа, така и параметри. След като входната лента се изпразни, стека също следва да се изпразни отгоре надолу. Когато и входната лента и стека са празни в изходната лента имаме оригиналния инфиксен математически израз записан във постфиксен или обратен полски запис.

История

Едсгер Дейкстра е нидерландски учен, изучавал първо математика и физика, след което приложна физика, като това помогнало той да бъде първият компютърен програмист от Нидерландия. Той е известен главно с двата си алгоритъма: единия служи за намиране на кай-кратkия път в граф, а другият служи за преобразуване на инфиксен математически израз в постфиксен. Този алгоритъм станал известен на публиката за пръв път през ноември 1961 година. Неговото наименование на английски е “shunting yard“ или буквално преведено площадка за маневриране, тъй като работата на алгоритъмът му напомняло на влакова площадка за маневриране.

Принцип на действие

За работата на евристичният алгоритъм са необходими: 1 стек, който да съхранява операторите, 1 опашка, която да служи за изход и 1 масив, лист или друга подобна структура с входни данни. Освен тях е необходим и „речник“, който да съхранява стойности за приоритетите на операторите. Алгоритъмът работи по следния начин - първо се прочита символ (или символи при използване на големи числа) от входната лента и се определя дали той е оператор или операнд. След като алгоритъмът успешно е успял да различил входния символ следва една един от два варианта: ако входния символ е операнд той отива директно в изходната лента, но ако той е оператор трябва да се определи дали той трябва да отиде в стека за оператори или не. Това се случва посредством речника с приоритети за операторите. Тук отново има няколко възможни изхода. Първият възможен изход е когато операторът от входната линия има приоритет по-голям

от този на последния оператор във стека. В този случай операторът се записва в стека и алгоритъмът продължава със следващия символ във входната лента. Ако обаче приоритета на входния оператор е по-малък от този на последния оператор в стека от стека в изходната лента се извеждат всички оператори, докато текущият оператор не срещне оператор с приоритет по-малък от неговия. След което този оператор се записва в стека и алгоритъмът отново продължава със следващия символ от входната лента. Третият вариант е, когато символът от входната лента е със същия приоритет, като последния символ в стека. В този случай последния оператор в стека се извежда в изходната лента, а текущият оператор се записва в стека. Когато данните от входната лента се изчерпат стека се изпразва отгоре надолу в изходната лента.

Специални случай

Ако алгоритъмът трябва да се справя със степенуване трябва да се вземе предвид и последователността на действията, тъй като при множество поредни степенувания редът на действията трябва да се обърне. Методът, по който алгоритъмът се справя с този проблем е лесен за имплементация. Когато инициализираме речника със стойности на приоритетите за различните оператори трябва да се инициализира втори речник с приоритети, като този на степенуването трябва да бъде завишен. Така алгоритъмът разполага с два речника: един за операторите от входната лента и един за операторите от стека. Когато трябва операторите да се сравняват се взимат стойности от съответния речник. Така, когато трябва в стека да се вкарат два или повече последователни знака за степенуване те влизат в стека, без да се извежда каквото и да било. Това гарантира правилната последователност при работа със степени. Освен степените друг специален случай е когато в математическия израз има скоби. Отварящата скоба има максимален приоритет при влизане, което гарантира това, че нищо няма да излезе от стека, и по-нисък приоритет, от този на всички операнди, в стека, което гарантира задържането на всички оператори между скобите в стека. Затварящата скоба има нулев приоритет при влизане и няма инициализиран приоритет във стека, тъй като тя не влиза в стека. Нулевият приоритет гарантира извеждане на всички оператори между скобите във изходната лента. Когато затварящата скоба срещне отваряща те се самоунищожават и алгоритъмът продължава работата си. Освен това има и два специални символа, които се използват при работата на алгоритъма, това са ‘#’ и ‘ý’. Тези символи служат съответно за инициализиране на край на израза и край на стека. Когато тези два оператора се срещат те се самоунищожават също както се самоунищожават и скобите.

Експериментална част

Нека разгледаме пример, където използваме алгоритъма на Дейкстра за преобразуване на инфиксен математически израз в постфиксен.

Инфиксен запис: a + b ^ c * d / (e + f * g)

```
Step #0

Input:
[a][+][b][^][c][*][d][/][(][e][+][f][*][g][)]

Stack:

| |
|_|

Output:
[][][][][][][][][][][][]
```

Фиг. 1

```
Step #1

Input:
[+][b][^][c][*][d][/][(][e][+][f][*][g][)]

Stack:

| |
|_|

Output:
[a][][][][][][][][][][][]
```

Фиг. 2

```
Step #2

Input:
[b][^][c][*][d][/][(][e][+][f][*][g][)]

Stack:

| |
|+|
|_|

Output:
[a][][][][][][][][][][][]
```

Фиг. 3

```
Step #3

Input:
[^][c][*][d][/][(][e][+][f][*][g][)]

Stack:

| |
|+|
|_|

Output:
[a][b][][][][][][][][][][]
```

Фиг. 4

```
Step #4

Input:
[c][*][d][/][()[e][+][f][*][g][])]

Stack:

| |
|^|
|+|
|_|

Output:
[a][b][][][][][][][][][][][]
```

Фиг. 5

```
Step #5

Input:
[*][d][/][()[e][+][f][*][g][])]

Stack:

| |
|^|
|+|
|_|

Output:
[a][b][c][][][][][][][][][][]
```

Фиг. 6

```
Step #6

Input:
[d][/][()[e][+][f][*][g][])]

Stack:

| |
|*|
|+|
|_|

Output:
[a][b][c][^][][][][][][][][][]
```

Фиг. 7

```
Step #7

Input:
[/][()[e][+][f][*][g][])]

Stack:

| |
|*|
|+|
|_|

Output:
[a][b][c][^][d][][][][][][][][]
```

Фиг. 8

```
Step #8

Input:
[()[e][+][f][*][g][])]

Stack:

| |
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][][][][][][][]
```

Фиг. 9

```
Step #9

Input:
[e][+][f][*][g][])]

Stack:

| |
|(|
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][][][][][][][]
```

Фиг. 10

```
Step #10

Input:
[+][f][*][g][])]

Stack:

| |
|(|
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][e][][][][][][]
```

Фиг. 11

```
Step #11

Input:
[f][*][g][])]

Stack:

| |
|+|
|(|
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][e][][][][][][]
```

Фиг. 12

```
Step #12

Input:
[*][g][])]

Stack:

| |
|+|
|(|
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][e][f][][][][][]
```

Фиг. 13

```
Step #13

Input:
[g][)]

Stack:

| |
|*|
|+|
|(|
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][e][f][][][][][]
```

Фиг. 14

```
Step #14

Input:
[)]

Stack:

| |
|*|
|+|
|(|
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][e][f][g][][][][]
```

Фиг. 15

```
Step #15

Input:
[)]

Stack:

| |
|+|
|(|
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][e][f][g][*][][][]
```

Фиг. 16

```
Step #16

Input:
[)]

Stack:

| |
|(|
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][e][f][g][*][+][][]
```

Фиг. 17

```
Step #17

Input:


Stack:

| |
|/|
|+|
|_|

Output:
[a][b][c][^][d][*][e][f][g][*][+][][]
```

Фиг. 18

```
Step #18

Input:


Stack:

| |
|+|
|_|

Output:
[a][b][c][^][d][*][e][f][g][*][+][/][]
```

Фиг. 19

```
Step #19

Input:


Stack:

| |
|_|

Output:
[a][b][c][^][d][*][e][f][g][*][+][/][+]
```

Фиг. 20

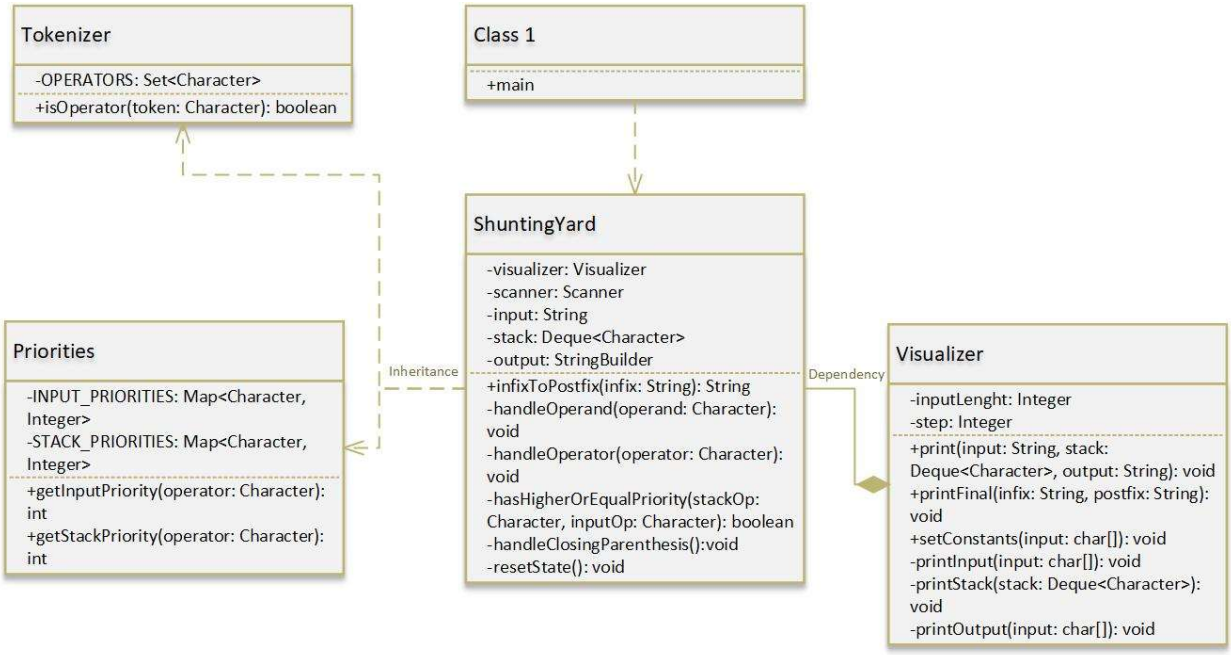
Постфиксен запис: a b c ^ d * e f g * + / +

Имплементация

Линк към github: https://github.com/Matrix2121/Dijkstra

Имплементацията е реализирана чрез 5 класа включвайки главния. Езика за програмиране, който е използван, е Java.

UML клас диаграма



Основен клас

Основният клас, в който се извършва цялата логика. Той е съставен от static полетата “visualizer” от тип “Visualizer”, “scanner” от тип “Scanner”, и “input”, “stack” и “output”, които съхраняват моментното състояние на алгоритъма. Освен тях класа разполага с един главен метод, които разпределя задачите според това, какъв е следващия токен, който трябва да се обработи. Той модифицира “input”, премахвайки празните места във String–а и превръщайки го в масив от символи, който после се обработват от един от трите метода: “handleOperand”, “handleOperator” или “handleClosingParenthesis”. “hasHigherOrEqualPriority” е помощен метод на “handleOperator”, който определя дали оператора трябва да влезе в стека или не. “resetState” се извиква винаги в началото на infixToPostfix, за да изчисти данни, които са останали от предното преобразуване.

Помощни класове

Програмата използва три помощни класа. Първия помощен клас се нарича “Tokenizer”. Той съдържа Set с възможните оператори и единствен метод, която проверява дали подаденият и токен е оператор. Ако съответният токен не бъде намерен в Set-а с оператори, то той е операнд. Втория помощен клас е “Priorities”. Той съдържа две структури от данни от вида Map, като и двете са съставени от Character за ключ и Integer за стойност. Тези речници съдържат стойностите на приоритет на операторите, като в единия се съхраняват приоритетите за входните оператори, а в другия приоритетите за операторите от стека. Освен тези структури от данни, класа разполага и с 2 метода, които приемат символ за входни данни, и връщат число, като и двете търсят съответния символ в съответните речници. Третия помощен клас носи името “Visualizer”. Той се използва за визуализиране на данните на конзолата. Съдържа две полета “inputLengt” и “step”. Първото поле се задава веднъж при инстанцирането на класа, а второто поле се използва като брояч на стъпките, които е извършил алгоритъма от главното си тяло. Освен това съдържа няколко метода, като от тях една се използва за визуализиране на стъпките на конзолата, една за извеждане на крайното състояние на математическия израз в своя постфиксен запис, една се използва за задаване на полетата и три се използват като помощни методи за визуализация по отделно на входящата лента, стека и изходящата лента, като те се сглобяват във главния метод “print”. Той приема като параметри моментните състояния на входната лента, стека и изходящата лента и ги предава на съответните методи.

Заключение

Установените правила за работа на алгоритъма гарантират неговата ефикасност и предвидимост при използването му. Той може точно да превърне един инфиксен математически израз в постфиксен.