

Transform Expression

Big-O Blue - Lecture 04: Stack & Queue

Tóm tắt đề bài

Tóm tắt đề bài

Cho T biểu thức đại số với các dấu ngoặc

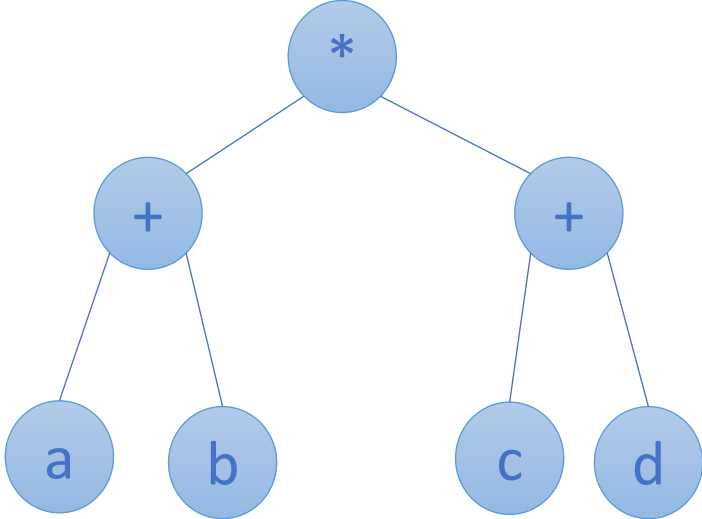
- Các toán tử 2 ngôi: $+$ $-$ $*$ $/$ $^$ và các dấu ngoặc $()$
- Các toán hạng: $a..z$
- Các biểu thức con với các phép toán luôn được đặt trong các dấu ngoặc

Yêu cầu: Tìm ký pháp nghịch đảo Ba Lan (RPN) của mỗi biểu thức.

(Biết rằng mỗi biểu thức chỉ có duy nhất 1 dạng RPN).

→ Không tồn tại biểu thức dạng $(a*b*c)$ (do có 2 RPN là $ab*c*$ và $abc**$)

Tóm tắt đề bài

Mathematical expression	Computer & calculator
$((a+b) * (c+d))$	<div><pre>graph TD; A((*)) --- B((+)); A --- C((+)); B --- D((a)); B --- E((b)); C --- F((c)); C --- G((d))</pre></div> <div>RPN = $ab+cd+*$</div> <ul style="list-style-type: none">✓ Giảm số kí tự của biểu thức✓ Giảm số lần thao tác

Tóm tắt đề bài

Cách chuyển đổi một biểu thức dưới dạng **ký pháp Ba Lan (RPN)** ?

Expression	RPN
$(A + B)$	$AB+$
$(A - B)$	$AB-$
$(A * B)$	$AB*$
(A / B)	$AB/$
$(A ^ B)$	$AB^$

Mô tả Input/Output

Input

- $T \leq 100$
- $|\text{expression}| \leq 400$

Output

T expression dưới dạng RPN

Giải thích ví dụ

Ví dụ 1

$E = (a + (b * c))$
= (a + (**b*c**))
= (**a** + **bc***)
= abc*+

Expression	RPN
(A + B)	AB+
(A - B)	AB-
(A * B)	AB*
(A / B)	AB/
(A ^ B)	AB^

Ví dụ 2

$E = ((a+b)*(z+x))$
= **$((a+b)*(z+x))$**
= $(ab+ * (z+x))$
= **$(ab+ * zx+)$**
= $ab+zx+*$

Expression	RPN
$(A + B)$	$AB+$
$(A - B)$	$AB-$
$(A * B)$	$AB*$
(A / B)	$AB/$
$(A ^ B)$	$AB^$

Ví dụ 3

$E = ((a+t)*((b+(a+c))^(c+d)))$
= **$((a+t)*((b+(a+c))^(c+d)))$**
= $(at+ * ((b+(a+c))^(c+d)))$
= $(at+ * ((b + ac+)^{(c+d)}))$
= $(at+ * (bac++ ^{(c+d)}))$
= $(at+ * (bac++ ^{cd+}))$
= **$(at+ * bac++cd+^)$**
= $at+bac++cd+^*$

Expression	RPN
$(A + B)$	$AB+$
$(A - B)$	$AB-$
$(A * B)$	$AB*$
(A / B)	$AB/$
$(A ^ B)$	$AB^$

Hướng dẫn giải

Nhận xét

Xét trên thứ tự tính toán của biểu thức, ta nhận thấy biểu thức con được tính toán trước bởi cặp ngoặc chứa nó (ngoặc mở và đóng hiện tại gần nó nhất)

→ Nguyên tắc **LIFO**

→ Ta sẽ sử dụng cấu trúc dữ liệu **stack** để giải quyết.

(a	+	(b	*	c))
---	---	---	---	---	---	---	---	---

Nhận xét

Như vậy **stack** sẽ lưu những gì ?

Ta nhận thấy rằng biểu thức $ab+$ khi viết dưới dạng RPN, phép toán được thực hiện bởi hai toán hạng **liền trước** nó → quan tâm thứ tự của các phép toán hơn các toán hạng

→ **stack** để lưu thứ tự của các phép toán

→ toán hạng đưa thẳng vào RPN

(a	+	(b	*	c))
---	---	---	---	---	---	---	---	---

Nhận xét

Hình thức hoạt động của **stack** như thế nào ?

- Gặp toán tử: **push** vào **stack**
- Gặp **dấu ngoặc đóng**: lấy **top** trong **stack** ra và đưa vào kết quả RPN

Tại sao cách thực hiện này lại đúng ?

- Chỉ tồn tại duy nhất một cách biểu diễn RPN
*(Không tồn tại $(a*b*c)$ vì có 2 $RPN = abc**$ hoặc $RPN = ab*c*$)*
- Để đảm bảo tồn tại duy nhất, các biểu thức con **luôn** được đặt trong một cặp ngoặc → Gặp ngoặc đóng ta có thể đưa ra toán tử mà không cần suy xét bên trong có nhiều hơn hai toán tử hay không

Các bước giải

Bước 1: Nhập T và T biểu thức.

Bước 2: Với mỗi biểu thức, khởi tạo một **stack** và biến RPN lưu kết quả cuối cùng.

Bước 3: Duyệt từng ký tự trong biểu thức từ trái sang phải. Nếu ký tự hiện tại là:

- ❑ Các toán hạng (a, b, c,...), ta đưa vào biến RPN.
- ❑ Phép toán, ta đưa chúng vào stack.
- ❑ Dấu ngoặc đóng, ta lấy các phần tử đỉnh của stack ra và thêm vào RPN.

Bước 4: In RPN.

Độ phức tạp: $O(N * T)$ với **N** là chiều dài của biểu thức.

Mã giả

Mô phỏng Thuật toán

(a	+	(b	*	c))
---	---	---	---	---	---	---	---	---

stack

--	--	--	--	--	--	--	--	--

RPN =

Mô phỏng Thuật toán

((a	+	b)	*	(z	+	x))
---	---	---	---	---	---	---	---	---	---	---	---	---

stack

--	--	--	--	--	--	--	--	--	--	--	--	--

RPN =

Mã giả

```
Read T
For tc = 1 → T
  read exp
  stack = []
  rpn = ""
  for I = 0 → len(exp)-1
    if exp[i] in ['+', '-', '*', '/', '^']:
      push exp[i] to stack
    else if exp[i] in ['a'..'z']:
      rpn += exp[i]
    else if exp[i] == ')':
      rpn += top of stack
      pop stack
  print(rpn)
```

Thank you