1.3 Dijkstra's Two-Stack Algorithm

click to begin demo
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
\begin{align*}
\text{infix expression:} & \quad (1 + (2 + 3) \times (4 \times 5)) \\
\text{(fully parenthesized)}: & \quad (1 + (2 + 3) \times (4 \times 5))
\end{align*}
\]
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

(value stack)     (operator stack)

(1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
\begin{align*}
( & 1 + ( ( 2 + 3 ) \times ( 4 \times 5 ) ) ) \\
\end{align*}
\]
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

(1 + ( (2 + 3) * (4 * 5) ) )
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

value stack
operator stack
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
(1 + (2 + 3) * (4 * 5))
\]
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```plaintext
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

- Value stack
- Operator stack

- 1
- +
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

Dijkstra's two-stack algorithm:

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

Value stack: 1

Operator stack: +
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
(1 + (2 + 3) \times (4 \times 5))
\]

value stack | operator stack
---|---
2 | +
1 | +

\[
(1 + (2 + 3) \times 4 \times 5)
\]
Dijkstra's two-stack algorithm

**Value**: push onto the value stack.

**Operator**: push onto the operator stack.

**Left parenthesis**: ignore.

**Right parenthesis**: pop operator and two values; push the result of applying that operator to those values onto the operand stack.
Dijkstra’s two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

```
value stack   operator stack

1   +   3 + 2
```

17
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
(1 + ( (2 + 3) \times (4 \times 5))) = 5
\]
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
\begin{align*}
\text{value stack} & \quad \text{operator stack} \\
\hline
1 & \quad \hphantom{5} \\
5 & \quad \ast \\
\ast & \quad \hphantom{5}
\end{align*}
\]

\[
(1 + (2 + 3)) \ast (4 \ast 5)
\]
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
(1 + (2 + 3)) \times (4 \times 5)
\]
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
(1 + (2 + 3) * (4 * 5))
```

```
value stack   operator stack

4
5
1
+
*     
```

25
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

Value stack
```
1
4
5
```
Operator stack
```
+  
*  
*  
+  
```
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
(1 + ( (2 + 3) \times (4 \times 5) ) )
\]
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

value stack | operator stack
---|---
5 | +
4 | *
5 | *
1 | +
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
  ( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

value stack          operator stack
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
\begin{array}{c}
5 \\
1 \\
\end{array} \quad \begin{array}{c}
* \\
+ \\
\end{array}
\]

value stack \quad operator stack

\[
(1 + (2 + 3) \times (4 \times 5)) = 20
\]
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

![Diagram of value and operator stacks]
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[(1 + (2 + 3) \times (4 \times 5))\]

value stack | operator stack
--- | ---
1 | 
+ | 20 | * | 5

(1 + (2 + 3) \times (4 \times 5))}
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
(1 + (2 + 3) \times (4 \times 5)) \rightarrow 35
\]

\[
\begin{array}{cc}
\text{value stack} & \text{operator stack} \\
1 & + \\
20 & \ast & 5 & = & 100
\end{array}
\]
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

(value stack) \( \begin{array}{ccc} ( & 1 & + & ( & ( & 2 & + & 3 & ) & * & ( & 4 & * & 5 & ) & ) & ) \end{array} \)

(operator stack) \( \begin{array}{ccc} 100 & + & 1 \end{array} \)
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

```
100 + 1 = 101
```
**Dijkstra's two-stack algorithm**

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

```
( ( 1 + ( ( 2 + 3 ) ) * ( 4 * 5 ) ) )
```

value stack       operator stack

101
Dijkstra's two-stack algorithm

**Value:** push onto the value stack.

**Operator:** push onto the operator stack.

**Left parenthesis:** ignore.

**Right parenthesis:** pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
(1 + (2 + 3) * (4 * 5))
\]

value stack          operator stack
Dijkstra's two-stack algorithm

Value: push onto the value stack.
Operator: push onto the operator stack.
Left parenthesis: ignore.
Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

\[
(1 + (2 + 3) * (4 * 5))
\]

result

101