Data Structures and Algorithms (資料結構與演算法)

Lecture 6: Stack

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

motivation

Visual Intuition of Stack

figure by Mx. Granger, licensed under CC0 1.0 via Wikimedia Commons

last-in-first-out (LIFO)

- stack of chairs
- stack of plates
- elevator

stack: a restricted data structure,
but important for computer science

The Three Stack Operations

figure by Mx. Granger, licensed under CC0 1.0 via Wikimedia Commons

```
PEEP(S)
    // GET usually named PEEP
  // return top element of S
```

```
PUSH(S, data)
    // INSERT usually named PUSH
  // put data onto top of S
```

```
POP(S)
    // REMOVE usually named POP
  // remove and return top element of S
```

sometimes other utility functions like SIZE() or ISEMPTY()

Parentheses Balancing

C

```c
int main(){
  printf("Hello␣World");
  return 0;
}
```

—(), {}, "", . . . need pairing

LISP

```lisp
(pow
  (* (+ 3 5)
     2)
  4)
```

—() needs pairing

how can we check parentheses balancing?

Stack Solution to Parentheses Balancing

any ')' should match last unmatched '(' (LIFO)

'(': Push ')': Pop

Parentheses Balancing Algorithm

```
$S$ = empty stack
for each $c$ in input
    if $c$ is '('
        Push($S$, $c$)
    else // $c$ is ')'
        if not IsEmpty($S$)
            Pop($S$)
        else
            return False
return IsEmpty($S$)
```

many more sophisticated use in compiler design

System Stack

- function call: compute with a new scratch paper
- old (original) scratch paper: temporarily not used; will be first to return to
- system stack: stack of scratch papers (stack frames), each containing
  - local variables (including parameters): to be used for calculating within this function
  - previous frame (return) pointer: to be used when escaping from this function

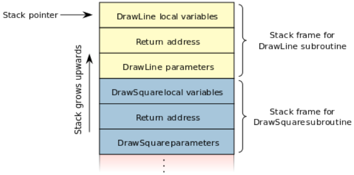
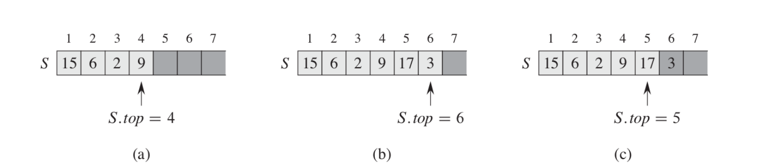


figure by Cameron McCormack, licensed under CC-BY-SA 2.5 via Wikimedia Commons

some related issues: security attack?

implementation

Stacks Implemented on Array



(Textbook Figure 10.1)

(a) stack with 4 elements

(b) after PUSH(S, 17) and PUSH(S, 3)

(c) after POP(S) which returns 3

```
PUSH($S$, $data$)
  $S.top = S.top + 1$
  $S.arr[S.top] = data$
```

```
POP($S$)
  $S.top = S.top - 1$
  return $S.arr[S.top + 1]$
```

usually: consecutive array with *S.top*
at 'tail' of array for $O(1)$ operations

Stacks Implemented on Linked List

if singly-linked list, top at head or tail?
which would you choose?

application: postfix evaluation

Stack for Expression Evaluation

$$a/b - c + d * e - a * c$$

- precedence: $\{*, /\}$ first; $\{+, -\}$ later
- steps
  - $f = a/b$
  - $g = f - c$
  - $h = d * e$
  - $i = g + h$
  - $j = a * c$
  - $\ell = i - j$

$$ab/c - de * +ac * -$$

Postfix Notation

same operand order, but put “operator” **after** needed operands
—can “operate” immediately when seeing operator
—no need to look beyond for precedence

Evaluate Postfix Expressions

$$3 * 4 - (5 + 6) * 7 + 8 * 9 \Longrightarrow 3\ 4\ *\ 5\ 6\ +\ 7\ *\ -\ 8\ 9\ *\ +$$

- how to evaluate? left-to-right, “operate” when see operator
- 3, 4, * ⇒ 12
- 12, 5, 6, + ⇒ 12, 11
- 12, 11, 7, * ⇒ 12, 77
- 12, 77, - ⇒ -65
- -65, 8, 9, * ⇒ -65, 72
- -65, 72, + ⇒ 7

stored where?
stack so closest operands will be considered first!

Stack Solution to Postfix Evaluation

Postfix Evaluation

```
S = empty stack
for each token in input
    if token is a number
        PUSH(S, token)
    elseif token is an operator
        b = POP(S)
        a = POP(S)
        PUSH(S, token(a, b))
return POP(S)
```

$3\ 4 * 5\ 6 + 7 * -\ 8\ 9 * +$

- 3, 4, * ⇒ 12
- 12, 5, 6, + ⇒ 12, 11
- 12, 11, 7, * ⇒ 12, 77
- 12, 77, - ⇒ -65
- -65, 8, 9, * ⇒ -65, 72
- -65, 72, + ⇒ 7

matches closely with the definition of postfix notation

application: expression parsing

Postfix from Infix (Usual) Notation

- infix:

$$3 \;/\; 4 \;-\; 5 \;+\; 6 \;*\; 7 \;-\; 8 \;*\; 9$$

- parenthesize:

$$3 \;/\; 4 \;-\; 5 \;+\; 6 \;*\; 7 \;-\; 8 \;*\; 9$$

- for every triple in parentheses, switch orders

- remove parentheses

need multi-passes if using computers

One-Pass Algorithm for Infix to Postfix

infix ⇒ postfix efficiently?

- at /, not sure of what to do (need later operands) so **store**

$$a/b - c + d * e - a * c$$

- at -, know that a / b can be a b / because - is of lower precedence

$$a/b-c + d * e - a * c$$

- at +, know that ? - c can be ? c - because + is of same precedence but {-, +} is left-associative

$$a/b - c+d * e - a * c$$

- at *, not sure of what to do (need later operands) so **store**

$$a/b - c + d*e - a * c$$

stored where? **stack** so closest operators will be considered first!

Stack Solution to Infix-Postfix Translation

```
$S_2$ = empty stack
for each token in input
    if token is a number
        output token
    elseif token is an operator
        while not Is-Empty($S_2$) and Peep($S_2$) is higher/same precedence
            output Pop($S_2$)
        Push($S_2$, token)
```

- here: infix to postfix with operator stack S_2
  —closest operators will be considered first
- recall: postfix evaluation with operand stack S
  —closest operands will be considered first
- mixing the two algorithms (say, use two stacks): simple calculator

Some More Hints on Infix-Postfix Translation

```
$S_2$ = empty stack
for each token in input
    if token is a number
        output token
    elseif token is an operator
        while not Is-Empty($S_2$) and Peep($S_2$) is higher/same precedence
            output Pop($S_2$)
        Push($S_2$, token)
```

- for left associativity and binary operators
    - right associativity? same precedence needs to wait
    - unary/trinary operator? same
- parentheses? highest priority
    - at '(', cannot pop anything from stack
      —like seeing '*' while having '+' on the stack
    - at ')', can pop until '(' —like parentheses matching

Summary

Lecture 6: Stack

- motivation

  temporary storage with LIFO
- implementation

  $O(1)$ **push/pop from tail of array**
- application: postfix evaluation

  stack as temporary storage of partial results
- application: expression parsing

  stack as temporary storage of waiting operands