Stack

Hsuan-Tien Lin

Dept. of CSIE, NTU

March 23, 2021

What We Have Done

- asympototic notation for complexity and its usage
- stack: motivation, parenthesis balancing

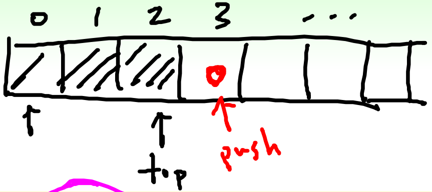
Intuition

Stack

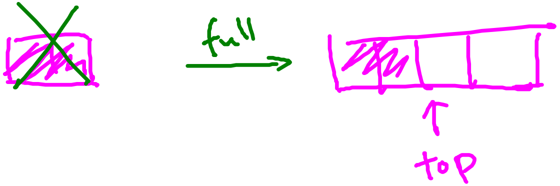
mimic: “pile of documents” on your desk

Implementation

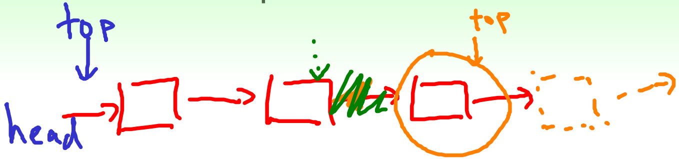
Stacks Implemented on Array



usually: (growable) consecutive array and push/pop at `end-of-array`



Stacks Implemented on Linked List



usually: singly linked list and push/pop at `head`

```
typedef struct{
    int data;
    struct node* next;
} node;
```

Application: Expression Evaluation

Stack for Expression Evaluation

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

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

in-fix expression

中序

$*(d, e)$ prefix

$d, e, *$ postfix

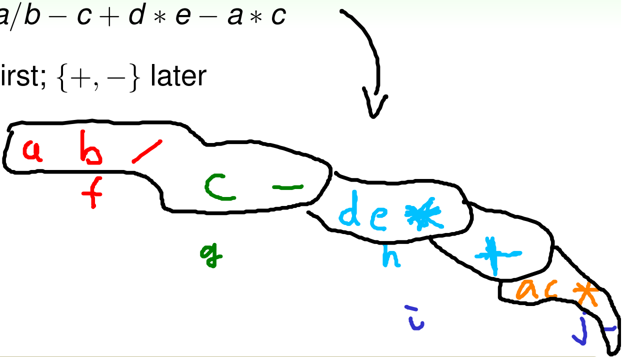
Postfix Notation

same operand order, but put “operator” after needed operands

Stack for Expression Evaluation

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

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



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

$$34/5 - 67 * +89 * -$$

- how to evaluate? left-to-right, “operate” when see operator
- 3, 4, / $\Rightarrow$ 0.75
- 0.75, 5, - $\Rightarrow$ -4.25
- -4.25, 6, 7, * $\Rightarrow$ -4.25, 42 (note: -4.25 stored for latter use)
- -4.25, 42, + $\Rightarrow$ 37.75
- 37.75, 8, 9, * $\Rightarrow$ 37.75, 72 (note: 37.75 stored for latter use)
- 37.75, 72, - $\Rightarrow$ ...

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

Stack Solution to Postfix Evaluation

Postfix Evaluation

```
for each token in the input do
    if token is a number
        push token to the stack
    else if token is an operator
        sequentially pop operands $a_{t-1}, \cdots, a_0$ from the stack
        push token$(a_0, a_1, a_{t-1})$ to the stack
    end if
end for
return the top of stack
```

matches closely with the definition of postfix notation

Application: Expression Parsing

One-Pass Algorithm for Infix to Postfix

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

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

a b

One-Pass Algorithm for Infix to Postfix

- 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

ab /

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

~~/~~

$-$

One-Pass Algorithm for Infix to Postfix

- 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

ab / c - d e * + a c * -

$$a/b - c + d \boxed{*} e \boxed{-} a * c$$

[illegible]

[illegible]

One-Pass Algorithm for Infix to Postfix

- 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

```
for each token in the input do
    if token is a number
        output token
    else if token is an operator
        while top of stack is of higher (or same) precedence do
            pop and output top of stack
        end while
        push token to the stack
    end if
end for
```

- here: infix to postfix with operator stack
  —closest operators will be considered first
- recall: postfix evaluation with operand stack
  —closest operands will be considered first
- mixing the two algorithms (say, use two stacks): simple calculator (i.e. HW1, Problem 4)

Some More Hints on Infix-Postfix Translation

```
for each token in the input do
   if token is a number
      output token
   else if token is an operator
      while top of stack is of higher (or same) precedence do
         pop and output top of stack
      end while
      push token to the stack
   end if
end for
```

- parentheses? higest priority
    - at '(', cannot pop anything from stack
      —like seeing '*' while having '+' on the stack
    - at ')', can pop until '(' —like parentheses matching