Stacks and Queues

Hsuan-Tien Lin
Dept. of CSIE, NTU
March 28–29, 2011
What We Have Done

- HW2 Hints
- String Pattern Matching Algorithm (Knuth-M-P)
- Stack: Parenthesis Balancing
- System Stack
- Stack with Dynamically Allocated Array
- Postfix Expression
- Reading Assignment:
  Stack with Fixed C Array
$a/b - c + d \times e - a \times c$

- precedence: `{*, /}` first; `{+, -}` later
- steps
  - $f = a/b \implies ab/$
  - $g = f - c \implies fc- \implies ab/c-$
  - $h = d \times e \implies de*$
  - $i = g + h \implies gh+ \implies ab/c - de*+$
  - $j = a \times c \implies ac*$
  - $\ell = i - j \implies ij- \implies ab/c - de* + ac*-$

**Postfix Notation**

Same operand order, but put “operator” after needed operands—can “operate” immediately when seeing operator—no need to look beyond for precedence
Postfix from Infix (Usual) Notation

- **infix:** \( (((((3 \div 4) - 5) + (6 \times 7)) - (8 \times 9))\)
- **parenthesize:** 
  \[ 3 \div 4 - 5 + 6 \times 7 - 8 \times 9 \]
- for every triple in parentheses, switch orders 
  \( (((((3 \div 4) 5-) (6 \times 7) +) (8 \times 9) -)\)
- remove parentheses 
  \[ 3 \div 4 - 5 + 6 \times 7 - 8 \times 9 \]

**difficult to parenthesize efficiently**
Evaluate Postfix Expressions

\[ \frac{34}{5} - 67 \times +89 \times - \]

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

stored where?

**stack** so closest operands will be considered first!
Postfix Evaluation

for each \textit{token} in the input do

\hspace{1em} if \textit{token} is a number
\hspace{2em} push \textit{token} to the stack

\hspace{1em} else if \textit{token} is an operator
\hspace{2em} sequentially pop operands $a_{t-1}, \cdots, a_0$ from the stack
\hspace{2em} push $\textit{token}(a_0, a_1, a_{t-1})$ to the stack

end if

end for

return the top of stack

matches closely with the definition of postfix notation
One-Pass Algorithm for Infix to Postfix

- at `/`, note 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 `*`, note 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!
a*b-c+d/e*f*g-i+k/l*m END

stack 1 (num):
stack 2 (operator):

output: ab*c-de/f*g*i-k/l/m*+

method 0: "keep count of" what to output
method 1: output to stack 1
method 2: don't use stack 1 and direct output
Stack Solution to Infix-Postfix Translation

```plaintext
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
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

- +

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
Queue

- object: a container that holds some elements
- action: enqueue (to the rear), dequeue (from the front)

- first-in-first-out (FIFO): 買票，印表機
- also very restricted data structure, but also important for computers
Reading Assignment

be sure to go ask the TAs or me if you are still confused
Reading Assignment

be sure to go ask the TAs or me if you are still confused
Comparing Stacks with Queues: A Mazing Problem (Sec. 3.5 and More)

\[ \text{GET-OUT-RECURSIVE}(m, (0, 0)) \]

Getting Out of Maze Recursively

\[
\text{GET-OUT-RECURSIVE}(\text{Maze } m, \text{ Postion } (i, j)) \\
\text{mark } (i, j) \text{ as visited} \\
\text{for each unmarked position } (k, \ell) \text{ from } (i, j) \text{ do} \\
\hspace{1em} \text{if } (k, \ell) \text{ is an exit} \\
\hspace{2em} \text{return TRUE} \\
\hspace{1em} \text{end if} \\
\hspace{1em} \text{if } \text{GET-OUT-RECURSIVE}(m, (k, \ell)) \\
\hspace{2em} \text{return TRUE} \\
\hspace{1em} \text{end if} \\
\text{end for} \\
\text{return FALSE}
\]
Getting Out of Maze by Stack

\textbf{GET-OUT-STACK}(Maze }m\text{, Postion } (i, j)\text{)

\begin{algorithmic}
\State \textbf{while} stack not empty \textbf{do}
\State \hspace{1em} \textbf{if} \ (i, j) \textbf{ is an exit} \textbf{then}
\State \hspace{2em} \textbf{return} TRUE
\State \hspace{1em} \textbf{end if}
\State \hspace{1em} \textbf{push} \ (k, \ell) \textbf{ to stack}
\State \textbf{end for}
\State \textbf{return} FALSE
\end{algorithmic}

- similar result to recursive version, but conceptually different
  - recursive: one path on the system stack
  - stack: many positions-to-be-explored on the user stack
- in textbook: a slightly different version for stack
From Stack to Queue

Getting Out of Maze by Queue

\begin{equation}
\text{GET-OUT-STACK}(\text{Maze } m, \text{ Position } (i, j))
\end{equation}

\begin{algorithm}
\textbf{while} stack not empty \textbf{do}

\begin{align*}
(i, j) &\leftarrow \text{dequeue from queue} \\
\text{mark } (i, j) &\text{ as visited}
\end{align*}

\begin{algorithm}
\textbf{for} each unmarked position \((k, \ell)\) from \((i, j)\) in reverse order \textbf{do}

\begin{align*}
\text{if } (k, \ell) \text{ is an exit} &\quad \text{return } \text{TRUE} \\
\text{enqueue } (k, \ell) \text{ to queue}
\end{align*}

\end{algorithm}

\textbf{end for}

\textbf{end while}

\text{return } \text{FALSE}
\end{algorithm}

- use of stack/queue: store the yet-to-be-explored positions
- stack version: first (lexicographically) way out (explore deeply)
- queue version: shortest way out (explore broadly)
Some Useful Implementations in C++

Standard Template Library (STL)

- container `vector`: dynamically growing dense array
- container adapter `stack`: turning some container to a stack
- container adapter `queue`: turning some container to a queue

```cpp
#include <vector>
#include <stack>
#include <queue>

using namespace std;

vector<int> intarray;
stack<char> charstack;
queue<double> doublequeue;

intarray.resize(20); intarray[3] = 5;
charstack.push('(');
char c = charstack.pop();
doublequeue.push(3.14);

double d = doublequeue.pop();
```