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
Stack for Expression Evaluation (Sec. 3.6)

\[
a/b - c + d * e - a * c
\]

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

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

difficult to parenthesize efficiently
Evaluate Postfix Expressions

\[
\frac{34}{5} - 67 * 89 * -
\]

- 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)
- \(37.75, 72, - \Rightarrow \ldots\)

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
One-Pass Algorithm for Infix to Postfix

infix ⇒ postfix efficiently?

- at /, note sure of what to do (need later operands) so store
  \[ \frac{a}{b} - c + d \times e - a \times c \]

- at -, know that a / b can be a b / because - is of lower precedence
  \[ \frac{a}{b} - c + d \times e - a \times c \]

- at +, know that ? - c can be ? c - because + is of same precedence but \{-, +\} is left-associative
  \[ \frac{a}{b} - c + d \times e - a \times c \]

- at *, note sure of what to do (need later operands) so store
  \[ \frac{a}{b} - c + d \times e - a \times c \]

stored where? stack so closest operators will be considered first!
a*b/d+c-d/e+f*g*h END

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

output: ab*d/c+de/-fg*h*+

method 0: "keep count of" what to output
method 1: output to stack 1
method 2: don't use stack 1 and direct output
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
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
Queues (Sec. 3.3): Abstract Data Type

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
Circular Queues with Fixed C Array (Part of Sec. 3.3)

Reading Assignment

be sure to go ask the TAs or me if you are still confused
Circular Queues with Dynamic Array (Sec. 3.4)

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)

GET-OUT-RECURSIVE($m$, (0, 0))

Getting Out of Maze Recursively

GET-OUT-RECURSIVE(Maze $m$, Position $(i, j)$)

mark $(i, j)$ as visited

for each unmarked position $(k, \ell)$ from $(i, j)$ do

if $(k, \ell)$ is an exit

return TRUE

end if

if GET-OUT-RECURSIVE($m$, $(k, \ell)$)

return TRUE

end if

end for

return FALSE
From Recursive to Stack

Getting Out of Maze by Stack

GET-OUT-STACK(Maze \( m \), Position \((i, j)\))

while stack not empty do
  \((i, j) \leftarrow \) pop from stack
  mark \((i, j)\) as visited
  for each unmarked position \((k, \ell)\) from \((i, j)\) in reverse order do
    if \((k, \ell)\) is an exit
      return TRUE
    end if
    push \((k, \ell)\) to stack
  end for
end while
return FALSE

- 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

\[
\text{GET-OUT-STACK}(\text{Maze } m, \text{ Position } (i, j))
\]
\[
\text{while stack not empty do}
\]
\[
(i, j) \leftarrow \text{dequeue from queue}
\]
\[
\text{mark } (i, j) \text{ as visited}
\]
\[
\text{for each unmarked position } (k, \ell) \text{ from } (i, j) \text{ in reverse order do}
\]
\[
\text{if } (k, \ell) \text{ is an exit}
\]
\[
\text{return TRUE}
\]
\[
\text{end if}
\]
\[
\text{enqueue } (k, \ell) \text{ to queue}
\]
\[
\text{end for}
\]
\[
\text{end while}
\]
\[
\text{return FALSE}
\]

- 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_back('(');
char c = charstack.pop_back();
doublequeue.push_back(3.14);
double d = doublequeue.pop_front();
```