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
  - $f = a/b \Longrightarrow ab/$
  - $g = f - c \Longrightarrow fc- \Longrightarrow ab/c-$
  - $h = d * e \Longrightarrow de*$
  - $i = g + h \Longrightarrow gh+ \Longrightarrow ab/c - de*+$
  - $j = a * c \Longrightarrow ac*$
  - $\ell = i - j \Longrightarrow ij- \Longrightarrow ab/c - de*+ac*-$

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

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

- 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

One-Pass Algorithm for Infix to Postfix

infix $\Rightarrow$ postfix efficiently?

- 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/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

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

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

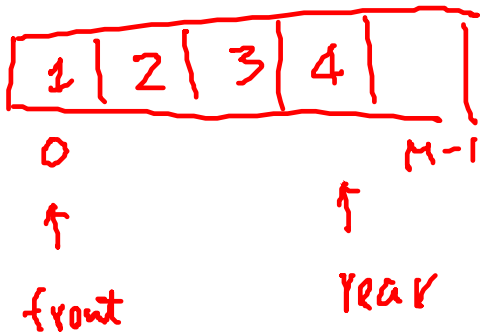
Queues (Sec. 3.3): Abstract Data Type

Queue

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

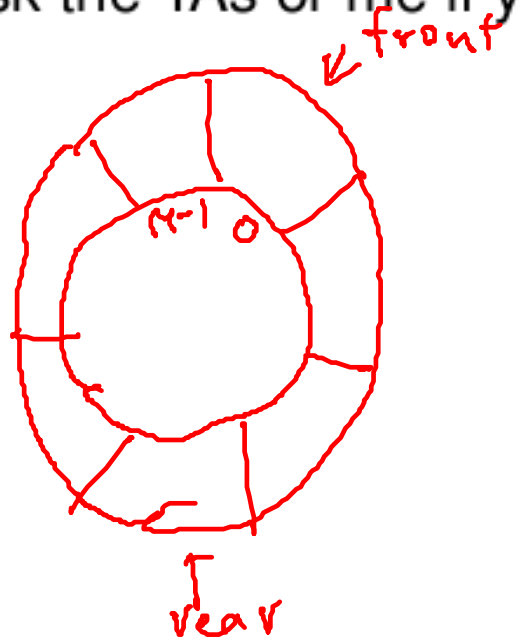
- first-in-first-out (FIFO): 買票，印表機
- also very restricted data structure, but also important for computers

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 array

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, Postion (i, j))
  mark (i, j) as visited
  for each unmarked position (k, ℓ) from (i, j) do
    if (k, ℓ) is an exit
      return TRUE
    end if
    if GET-OUT-RECURSIVE(m, (k, ℓ))
      return TRUE
    end if
  end for
  return FALSE
```

From Recursive to Stack

Getting Out of Maze by Stack

GET-OUT-STACK(Maze m, Postion (i, j))

```
while stack not empty do
    (i, j) ← pop from stack
    mark (i, j) as visited
    for each unmarked position (k, ℓ) from (i, j) in reverse order do
        if (k, ℓ) is an exit
            return TRUE
        end if
        push (k, ℓ) 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

GET-OUT-~~STACK~~(Maze m, Postion (i, j))

```
while ~~stack~~ Q not empty do
    (i, j) ← dequeue from queue
    mark (i, j) as visited
    for each unmarked position (k, ℓ) from (i, j) in reverse order do
        if (k, ℓ) is an exit
            return TRUE
        end if
        enqueue (k, ℓ) to queue
    end for
end while
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

```
1  #include <vector>
2  #include <stack>
3  #include <queue>
4  using namespace std;
5  vector<int> intarray;
6  stack<char> charstack;
7  queue<double> doublequeue;
8  intarray.resize(20); intarray[3] = 5;
9  charstack.push_back('(');
10 char c = charstack.pop_back();
11 doublequeue.push_back(3.14);
12 double d = doublequeue.pop_front();
```

std::vector