CSCI 204 Stack ADT Workshop

Fall 2017

Xiannong Meng

1. We discussed the algorithm to determine if a string is a palindrome using a stack. Write the function is_palindrome(s) using a stack. You may use the help function given below to pre-process the input string to remove all non-alphabet characters. Test your program with the following strings.

   “aa”, “b”, “aac”, “aabb”, “abba”, “A Toyota’s a Toyota.”, “Civic”, “Tell a ballet.”

   ```
   def preprocess(s):
     """ Remove all non-alphabet and turn every one to lower case """
     s = s.lower()
     new_s = ''
     for c in s:
       if c >= 'a' and c <= 'z':
         new_s += c
     return new_s
   ```

2. Write a function eval_postfix(e) to compute the value of a postfix expression using a stack. The parameter e is an algebraic expression in the form of a string. We can assume each value in the expression is a single digit, and the operators are limited to +, -, *, and /. For example 23* should result in 6, 643+* should result in 42, and 62/44+- should result in -5. For simplicity you may assume the original values in the expression string are non-negative. However once the evaluation starts the values on the stack can be any integers.

   The general algorithm is as follows. You may assume all expressions are well formed (valid). Test your program using the examples given in the problem and pick some extra ones by yourself.

   ```
   while expression not exhausted yet:
       read next token
       if it is a variable:
           push it onto the stack
       else:   # an operator ‘op’
           right = pop()
           left = pop()
           result =  left op right   # you should write a function for doing the arithmetic operation
           push the result back onto the stack
   the value at the top of the stack is the result of the expression
   ```

3. Airline scheduling problem (HPAir from Carrano’s *Data Abstraction and Problem Solving in C++* 3rd edition.
   Given
   i. A set of cities that HPAir serves

ii. Pairs of city names, each pair represents the origin and destination of one flight
iii. Pair of cities names each of which represents a request to fly from an origin to a destination

Find whether or not a path exists between two cities requested by a passenger

General idea

1. Start from the origin city
2. Find a city that is connected to the current city
3. If the next city is the destination, we are done. Report a route has been found
4. Otherwise go from this city and repeat step 2
5. If we visited all cities and no route is found, we declare the failure

Algorithm

```
Find_route(inOrigCity, inDestCity)
  create a new stack
  mark all cities un-visited;

  myStack.push(inOrigCity);
  mark inOrigCity visited;

  while (!myStack.isEmpty() and
         inDestCity != myStack.peek())
        if (no flights exist from the city on the top
              of the stack to un-visited cities):
            myStack.pop()    # backtrack
        else:
           select a unvisited destination city C from the city
                 at the top of the stack
           myStack.push(C)
           mark C as visited


  if myStack.isEmpty():
    return False
  else:
    return True   # the stack should contain the route
```