EXAMEN PARCIAL I - SOLUCIÓN

Docente: M.S. Franci Suni Lopez

16/10/2020

1. Explique cómo implementar dos pilas en un solo array, de tal manera que ninguna pila se desborde a menos que el número total de elementos en ambas pilas juntas sea n. Las operaciones PUSH y POP deben ejecutarse en tiempo O(1).

   Dadas las pilas T y R. Inicialmente, vamos a establecer T.top = 0 y R.top = n + 1.

   Básicamente, la pila T usa la primera parte del array y la pila R usa la última parte del array. En la pila T, el top es el elemento más a la derecha de T. En la pila R, el top es el elemento más a la izquierda de R. Los algoritmos PUSH y POP se definen a continuación:

Algorithm 1 PUSH(S,x)

```
1:  if S == T then
2:      if T.top + 1 == R.top then
3:          error "overflow"
4:      else
5:          T.top = T.top + 1
6:          T[T.top] = x
7:      end if
8:  end if
9:  if S == R then
10:     if R.top − 1 == T.top then
11:         error "overflow"
12:     else
13:         R.top = R.top − 1
14:         T[T.top] = x
15:     end if
16: end if
```

Algorithm 2 POP(S)

```
if S == T then
    if T.top == 0 then
        error "underflow"
    else
        T.top = T.top − 1.
        return T[T.top + 1]
    end if
end if
if S == R then
    if R.top == n + 1 then
        error "underflow"
    else
        R.top = R.top + 1.
        return R[R.top − 1]
    end if
end if
```

2. Diseñe un algoritmo no recursivo con un tiempo de ejecución $\Theta(n)$ que invierta una lista simplemente enlazada de n elementos. El algoritmo no debe utilizar más que un almacenamiento constante más allá del necesario para la lista en sí.

   Para invertir la lista, usamos dos variables adicionales que nos va a permitir reasignar los punteros a medida que se recorre la lista. De esta forma, el nodo k que apuntaba a k+1, con la reasignación de los punteros el nodo k+1 apuntara a k. A continuación, se describe el algoritmo para invertir la lista:

Algorithm 9 REVERSE(L)

```
a = L.head.next
b = L.head
while a ≠ NIL do
    tmp = a.next
    a.next = b
    b = a
    a = tmp
end while
L.head = b
```

3. La complejidad del algoritmo A es 2^{n+1}, y del algoritmo B es 2^{2n}. ¿Por lo tanto, la complejidad del algoritmo A es $O(2^n)$? y ¿del algoritmo B es $O(2^n)$? Justifique su respuesta.

   Usando la definición de Notación O, tenemos que para el algoritmo A:
   $2^{n+1} \geq 2^1 \cdot 2^n$ para todo $n \geq 0$, entonces definimos $n_0=0$ y $c = 2^1$. Por lo tanto $2^{n+1}=O(2^n)$.
   Sin embargo, para el caso del algoritmo B, 2^{2n} no es $O(2^n)$. Para que cumpla la premisa, debe de existir un n_0 y una constante c tal que $n \geq n_0$ que implica $2^n \cdot 2^n = 2^{2n} \leq c2^{2n}$, entonces $2^n \leq c$ para $n \geq n_0$.
   Lo cual es claramente imposible ya que c tiene que ser una constante.

4. Diseñe un algoritmo que dada una lista simplemente enlazada: x_1 -> x_2 -> ... -> x_2n. La cual contiene 2n nodos. Reordene los nodos de tal manera que estén de la siguiente forma x_1 -> x_2n -> x_2 -> x_2n-1 -> x_3 -> ....

   Realizar un conteo de elementos en la lista y luego dividir la lista por la mitad, generando dos sublistas A: x_1, x_2, ..., x_n y B: x_n+1, ..., x_2n. Luego invertir el orden de los nodos de la lista B. Finalmente, fusionamos A y B, intercalando los elementos de ambas listas para obtener el resultado deseado.

5. Suponga que hay dos listas simplemente enlazadas que se cruzan en algún punto y a partir de ahí se convierten en una lista enlazada única. Se conocen los punteros de cabeza o de inicio de ambas listas, pero no se conoce el nodo de intersección. Además, se desconoce el número de nodos en cada una de las listas antes de que se crucen y puede ser diferente en cada lista. List1 puede tener n nodos antes de que alcance el punto de intersección, y List2 puede tener m nodos antes de alcanzar el punto de intersección

donde m y n pueden ser m = n, m <n o m> n. Proporcione un algoritmo para encontrar el punto de fusión.

Para encontrar el punto de fusión, comparamos cada puntero al nodo de la primera lista con cualquier otro puntero de nodo en la segunda lista por lo que los punteros de nodo coincidentes nos llevarán al nodo de intersección. Pero, la complejidad del tiempo en este caso será O (mn), que será alta.