CSCE 629 - 602 Analysis of Algorithms

September 28, 2016

Homework III **Rishabh Yadav**
Prof. Anxiao (Andrew) Jiang UIN: 425009824

1A. 17.1-2 Solution

The worst case of $k-$bit counter with DECREMENT operation occurs when there is *one* 1 and followed by $(k-1)$ $0s$ if we DECREMENT such a counter state we will get *one* 0 and followed by $(k-1)$ $1s$, thereby flipping all the $k-$bits,

FOR EXAMPLE:

8→	1	0	0	0
7→	0	1	1	1

when decremented
flips all the 4- bits, ($k-$bits)

Now let us suppose the next operation is an INCREMENT which changes counter from 7 to 8.

FOR EXAMPLE:

7→	0	1	1	1
8→	1	0	0	0

when incremented
again flips all the 4- bits, ($k-$bits)

In the worst case we alternate between one DECREMENT and one INCREMENT operation. This will flip all the $k-$ bits with every operation. For n such operations we will flip $k-$bits each time. Hence the cost of n such operations is $\Theta(nk)$.

2A. 17.2-1 Solution

We assign the following amortized cost for each operation, (*This is the accounting method*)

Operation	*Amortized cost*
PUSH	3
POP	1
MULTI-POP	1
COPY	0

- PUSH:- It take 1 credit to push the element on to the stack. It stores 1 credit for the pop operation that may occur in future and 1 credit for the copying/cloning of this element. hence 3 credits total.
- POP:- This operation utilizes 1 credit stored from the PUSH operation. It stores 1 credit for the copying operation. Hence 1 credit.
- MULTIPOP:- This operation utilizes 1 credit stored from the PUSH operation. It stores 1 credit for the copying operation. Hence 1 credit.
- COPY:- This operation uses the credit in the *account* from other operations. Hence 0 credits.

Since amortized cost for each operation is constant i.e. $O(1)$. The total cost of n stack operations is $O(n)$.

3A. 22.2-7 Solution

Idea

We represent every professional wrestlers who are *babyfaces* with B and *heels* with H. Also,
We will represent the given problem as a graph with following characteristics,

- Each professional wrestlers is represented as a vertex in the graph.
- Ever rivalry, r, between two professional wrestlers, u, v is represented with and edge between u and v.

Now according to the constraints given in the problem, We observe/deduce the following

- There can only be edges between a vertex B and a vertex H.
- Alternatively, there cannot exists edge between a vertex B & another vertex B OR a vertex H & another vertex H.

We can reduce this problem to check whether a graph is bipartite or a graph coloring problem with only two colors, (say red and blue).

The algorithm proposed here runs a Breadth first search (BFS) and tries to color each node alternatively with two colors (red and blue). If at any point we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 colors and hence is not Bipartite).

Pseudo Code

1. int color[n]
2. for i = 0 → n
   (a) color[i] = "NOT-COLORED"
3. color[s] = "RED"
4. queue q = {}
5. Set B = {}, Set R = {s}
6. q.enqueue(s)
7. while q is not empty// Run while there are vertices in queue (Essentially BFS)
   (a) int u = q.dequeue()
   (b) for v = 0 → n // Find all non-colored adjacent vertices
      i. if (edge exists between u,v and color[v] == "NOT-COLORED")
         A. if color[u] == "RED"
            then color[v] = "BLUE" and B.push(v)

else color[v] = "RED" and R.push(v)// Assign alternate color to this adjacent v of u

B. q.push(v)

ii. else if (edge between u,v and color[u] == color[v])

iii. return false

8. return true, sets R,B.

Proof of correctness

Since we know that BFS is exhaustive i.e. visits every node whilst examining every edge. Since at every iteration we check for the neighbor's color, we can say that whenever there are two adjacent vertex with same color code our algorithm exits. If we reach the end that means we have not found a pair of adjacent vertices with same color code and all the vertex have been visited, we return the two sets, which are essentially the sets of 'professional wrestlers'.

Time Complexity

Algorithm proposed above visits every node exactly once and examined every edge, Hence the time complexity is $O(n + r)$, where n is the number of node and r is the number of edges.

Time Complexity = $O(n + r)$

References

Introduction to Algorithms by T. Cormen, C. Leiserson, R. Rivest, C. Stein