Homework3_432001358

February 15, 2022

0.1 22 SPRING CSCE 629 600: ANALYSIS OF ALGORITHMS - Homework 3

0.1.1 Name: Rohan Chaudhury

0.1.2 UIN: 432001358

0.1.3 Question (1) Textbook page 456, Exercise 17.1-3.

Suppose we perform a sequence of n operations on a data structure in which the i th operation costs i if i is an exact power of 2, and 1 otherwise. Use aggregate analysis to determine the amortized cost per operation.

0.1.4 Answer:

Given, that cost of ith operation is i if i is an exact power of 2 or 1 otherwise.

If c(i) is the cost of the i th operation, then:

c(i) = i, when i is an exact power of 2

c(i) = 1, otherwise

So, the cost of n number of operations would be:

$$\sum_{i=1}^{n} c(i) = \sum_{i=1}^{\lceil log\ n \rceil} 2^i + \sum_{where\ i \leq n\ and\ i\ is\ not\ a\ power\ of\ 2} 1$$

$$\leq \sum_{i=1}^{\lceil log\ n \rceil} 2^i + n$$

$$= 2^{(1+\lceil log\ n \rceil)} - 1 + n\ \leq (2n-1) + n$$

$$\leq 3n$$

$$\in O(n)$$

In order to find the average cost we have to divide by n, so:

The average cost = 3n/n = 3, which is O(1) operation.

Hence, using aggregate analysis we find that the amortized cost per operation is O(1).

0.1.5 Question (2) Textbook page 602, Exercise 22.2-7.

There are two types of professional wrestlers: "babyfaces" ("good guys") and "heels" ("bad guys"). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an O(n+r)-time algorithm that determines whether it is possible to designate some of the wrestlers as babyfaces and the remainder as heels such that each rivalry is between a babyface and a heel. If it is possible to perform such a designation, your algorithm should produce it.

0.1.6 Answer :

Main idea of the algorithm

The main idea of the algorithm is to use a graph where the n number of vertices of the graph will represent the n number of professional wrestlers and the r number of edges of the graph will represent the r pairs of rivalries between the wrestlers. Initially, we will assign the first wrestler to the "babyfaces" type. Then we will perform breadth-first search (BFS) to visit all the n vertices. As we have assigned the first wrestler to the "babyfaces" type, we will assign all its immediate neighbors to the other "heels" type and we will continue this procedure of alternate type assignment for all the other connecting vertices. Now during this process if we find that while assigning a vertex to either the "babyfaces" or "heels" type, one of its immediate neighbors has already been assigned to the same type ("babyfaces" or "heels") as this current vertex then we will return that the required kind of designation is not possible. Otherwise we will return that the required kind of designation is possible after visiting all the n vertices of the graph and also return the corresponding type ("babyfaces" or "heels") assignments of the vertices.

For our pseudo code we will denote "babyfaces" with value of 1 and "heels" with value of 2.

Pseudocode for the BFS algorithm: BFS(n,r)

1 -> #Let N[1…n] be the list of n professional wrestlers

2 -> #Let R[1…r] be the list of r pairs of wrestlers for which there are rivalries

3 -> #Let Adj[1…n] be the adjacency list of all the n vertices of the graph which we will construct from the list of r pairs of wrestlers

4 -> Adj=[[ ] for i in range(n)]

5 -> for (a,b) in R:

6 -> Adj[a].append(b)

7 -> Adj[b].append(a)

8 -> #For our pseudo code we will denote "babyfaces" type with value of 1 and "heels" type with value of 2.

9 -> #Let type[1…n] be an integer array to store the types of all the n vertices. Intially we will assign 0 to the entire array

```
10 -> type=[0]*n
11 -> #Let us now take a queue for the purpose of performing BFS on our graph
12 -> que=[]
13 -> for i=1 to n:
14 -> # if no type is assigned, then assign type with value 1
15 ->    if (type[i]==0):
16 ->       que.append(i)
17 ->       type[i]=1
18 ->       while (length(que)!=0):
19 ->            #popping and storing the first value of queue in curr
20 ->            curr=que.remove(0)
21 ->            #Looping over the vertices connected to the curr node
22 ->            for j in Adj[curr]:
23 ->             #checking if type of node curr and j are the same or not. If same, then we return False and terminate the program
24 ->              if (type[j]==type[curr]):
25 ->                 print (“The required designation is not possible”)
26 ->                 return False
27 ->             #if node j has no assigned type then we assign the type opposite to that of node curr to node j
28 ->             if (type[j]==0):
29 ->                 if (type[curr]==1):
30 ->                    type[j]=2
31 ->                 else:
32 ->                    type[j]=1
33 ->                 #appending the node j to que
34 ->                 que.append(j)
35 -> #If outside the first for loop which is here, then all the nodes are visited and proper designations have been assigned
36 -> print (“It is possible to perform the required designation”)
37 -> # returning the designated types array named “type” where 1 denotes the type “babyfaces” and 2 denotes the type “heels” for the corresponding vertex of that index
38 -> return type
```

Proof of correctness:

We are visiting all the vertices and comparing the type of the connecting vertices with type of this current vertex. If they are of the same type then we are saying that it is not possible to perform the required designation. Otherwise, if the neighboring vertex have no type assigned then we are assigning the type opposite to the current vertex to the neighboring vertex. In this way if we can successfully visit all the vertices and assign a type to them then we can say that the required designation can be performed, otherwise not. This is the proof of correctness of the algorithm.

Time complexity:

Given n is the number of vertices of the graph which represents the n number of professional wrestlers and the r is the number of edges of the graph which represents the r pairs of rivalries between the wrestlers:

1. Time complexity of enqueue and dequeue operations is O(1)
2. Time complexity of visiting n vertices is O(n) and time complexity of visiting r edges using the adjacency list is O(2r) (which is ultimately O(r)) as each edge can be traversed a maximum of 2 times
3. Time complexity of performing the BFS is O(n+r)

Hence the overall time complexity of the algorithm is O(n+r) which is as required.