Assigment 6 DMA

Mikkel Luc Carlsen

7. december 2020

1 Sets is represented using Arrays. The representative for each set is is also the value of all other members in the set. Therefor the 3 functions Init, Find and Union will look as the following:

```
Init(n)
    A=array(n)
    for k = 0 to n-1
        A[k] = k
    return A
```

Init creates an array A with with length n containing n elements, where each element is the only instance in it's own set.

```
Find(i)
    return A[i]
```

Find returns the representative of i.

```
Union(i,j)
    a1 = Find(i)
    a2 = Find(j)
    if a1 ≠ a2 then
        for k=0 to A.length - 1
            if find(k) = a2
                A.[k] = a1
            else
                ()
    else
        ()
```

Union uses find to get the representative for each of the two values inserted into Union, then it compares the two representatives and if they are not equal, then it changes the representive for all j to i's representative.

2 Init(9)
The Array looks as following:
A=[0, 1, 2, 3, 4, 5, 6, 7, 8]

Union(7,3)
The Array now looks as following and the affected elements have been marked with red color:
A=[0, 1, 2, 7, 4, 5, 6, 7, 8]

Union(8,0)
The Array now looks as following and the affected elements have been marked with red color:
A=[8, 1, 2, 7, 4, 5, 6, 7, 8]

Union(5,0)
The Array now looks as following and the affected elements have been marked with red color:
A=[5, 1, 2, 7, 4, 5, 6, 7, 5]

Union(4,5)
The Array now looks as following and the affected elements have been marked with red color:
A=[4, 1, 2, 7, 4, 4, 6, 7, 4]

Union(2,7)
The Array now looks as following and the affected elements have been marked with red color:
A=[4, 1, 2, 2, 4, 4, 6, 2, 4]

Union(0,6)
The Array now looks as following and the affected elements have been marked with red color:
A=[4, 1, 2, 2, 4, 4, 4, 2, 4]

Union(5,8)
The Array now looks as following and the affected elements have been marked with red color:
A=[4, 1, 2, 2, 4, 4, 4, 2, 4]

Union(1,4)
The Array now looks as following and the affected elements have been marked with red color:
A=[1, 1, 2, 2, 1, 1, 1, 2, 1]

3 The run time of Init(n) is $\Theta(n)$, since Init runs through the whole list and sets each element's value.

The run time of Find(i) is $\Theta(1)$, since Find just returns the representative.

The run time of Union(i,j) is $\Theta(n)$, since if Find(i) $\neq$ Find(j) then Union runs through all elements in the list and checks if they are equal to Find(j) and then changes those that are equal to Find(j) to be Find(i).

4 Effectively my version of Init(n) doesn't differ from the one under "Hurtig forening". The only difference is that in my function i show that i create an array and return the array, whereas this is implicit in the one under "Hurtig forening".

The differnce between my Find(i) and the one under "Hurtig forening" is that my function doesn't need a while-loop to run up to the "first" element in the set to find the representative. My function can just return the representative of whatever is on that index, since the representative of each set is also the only value that can be in the set.

The differnce between my Union(i,j) and the one under "Hurtig forening" is that when mine enters the if statement, then it has to run through the whole list and compare the elements in the list to make sure it has changed the whole set to have the new representative. This is also the reason that my Union takes O(n) while the one under "Hurtig forening" takes O(d) time. Meaning that my Union is unaffacted of how many elements there is to be united, since it will always traverse the whole array. While on the other hand the one under "Hurtig forening" is affected by that amount of elements in each set, since it unites all elements in the two sets. So to use the run time as a function of the amount of elements in the two sets there is to be united, then it could be expressed as $\Theta(n)$ where n is the total amount of elements in the two sets, for the one under "Hurtig forening". But it still does this much faster, since it just binds the root of one of the trees to the root of the other tree.

5 In CLRS the running time of UNION is told to be $\Theta(n)$ on average. In CLRS UNION works by setting the tail of the first set, to point at the head of the second set and then setting the tail of the first set to now be the tail of the second set. More over it also changes the pointers of all elements in the second set, that points to the set itself, to now point at the first set. The overall difference between UNION in ClRS and my own could shortly be said to be, that CLRS has to update pointers for the elements and for the set itself, whilst mine does not. To use the run-time as a function of the amount of elements in the two sets that is being united, then it would also be expressed as $\Theta(n)$, where n is the total amount of elements in the two sets.

6
```
Weighted-Union-Heuristic(i,j)
    a_1 = Find(i)
    a_2 = Find(j)
    if a_1 ≠ a_2 then
        if (size[a_1] > size[a_2]) then
            size[a_1] = size[a_1] + size[a_2]
            for k=0 to A.length - 1
                if find(k) = a_2 then
                    A.[k] = a_1
                else
                    ()
        else
            size[a_2] = size[a_2] + size[a_1]
            for k=0 to A.length - 1
                if find(k) = a_1 then
                    A.[k] = a_2
                else
                    ()
    else
        ()
```

The Weighted-Union-Heuristic as a difference also changes a value "size", which is the size of each set. Weighted-Union-Heuristic changes the size of a set to be its current size plus the size of the other set. It also compares the size of the two sets, before changing the representative for one of the sets. This has no effect for the running, since the function will still only enter one of the for-loops, Hence the run time is still $\Theta(n)$.