Zeynep Tufekci (NUID: 001565591)

INFO 6205

Program Structures & Algorithms

Fall 2020

Assignment 4

- **Task and Observations**

Step 1: Benchmark Weighted Quick Union with size vs. Weighted Quick Union with depth

```
private void mergeComponents(int i, int j) {

    if(i==j)//i and j are the roots of p and q
        return;

    if (height[i] < height[j]) {
        updateParent(i,j);
        updateHeight(j,i);
    }
    else {
        updateParent(j,i);
        updateHeight(i,j);
    }
}
private void updateParent(int p, int x) {
    parent[p] = x;
}

private void updateHeight(int p, int x) {
    height[p] += height[x];
}
```

Fig. 1 WQU with depth

```
private void mergeComponents(int i, int j) {

    if(i==j)//i and j are the roots of p and q
        return;

     if (depth[i] < depth[j]) {
         updateParent(i,j);
     } else if (depth[j] < depth[i]) {
         updateParent(j,i);
     } else {
         updateParent(i,j);
         updateDepth(j);
     }


}
private void updateParent(int p, int x) {
    parent[p] = x;
}

private void updateDepth(int p) {
    depth[p]++;
    if (depth[p] > maxDepth) {
        maxDepth = depth[p];
    }
}
```

Fig. 2 WQU with size

Step 2: Benchmark Weighted Quick Union with Path comparison with one pass path halving vs. Weighted Quick Union with Path comparison with two way

```
while (p != root) {
    int newp = parent[p];
    parent[p] = root;
    p = newp;
}
```

Fig. 3 all intermediate fix[WQUPC.java]

```
while (p != root) {//grandparent fix
    parent[p] = parent[parent[p]];
    p = parent[p];
}
```

Fig. 4 grandparent fix

- **Output**

For Step (1) :

I experimented n=100, 100000, 200000 and 500000 items.

Benchmarking size vs depth

a) Performance:

```
While N = 100
Weighted Quick Union with size:
Average time:(with size) 0.51246 ms
Weighted Quick Union with rank/depth:
Average time:(with depth) 0.50823 ms

While N = 100000
Weighted Quick Union with size:
Average time:(with size) 50.3709 ms
Weighted Quick Union with rank/depth:
```

Zeynep Tufekci (NUID: 001565591)

```
Average time:(with depth) 44.61872 ms

While N = 200000
Weighted Quick Union with size:
Average time:(with size) 111.71562 ms
Weighted Quick Union with rank/depth:
Average time:(with depth) 104.65875 ms

While N = 500000
Weighted Quick Union with size:
Average time:(with size) 364.5121 ms
Weighted Quick Union with rank/depth:
Average time:(with depth) 325.06366 ms
```

b) Average Max Depth:

```
Lg(100) : 6.0
While N = 100
Weighted Quick Union with size:
Average max depth 3.3 for size
Weighted Quick Union with rank/depth:
Average max depth 3.5 for depth

Lg(100000) : 16.0
While N = 100000
Weighted Quick Union with size:
Average max depth 7.3 for size
Weighted Quick Union with rank/depth:
Average max depth 7.0 for depth

Lg(200000) : 17.0
While N = 200000
Weighted Quick Union with size:
Average max depth 7.7 for size
Weighted Quick Union with rank/depth:
Average max depth 7.4 for depth

Lg(500000) : 18.0
While N = 500000
Weighted Quick Union with size:
Average max depth 8.2 for size
Weighted Quick Union with rank/depth:
Average max depth 7.7 for depth
```

For Step (2): Benchmarking fixing grandparent vs all intermediate nodes

Performance, Average Depth during construction and Average Depth at the end of construction

I experimented n=100, 100000, 200000 and 500000 items

```
While N = 100
Weighted Quick Union with all intermediate node fix:
Average time:(all intermediate node fix) 0.35069 ms
Average depth during construction: 3.0
Average depth at the end of operation: 1.7
Weighted Quick Union with grandparent node fix:
Average time:(with grandparent node fix) 0.21503 ms
Average depth during construction: 3.1
Average depth at the end of operation: 1.9

While N = 100000
Weighted Quick Union with all intermediate node fix:
Average time:(all intermediate node fix) 37.31866 ms
Average depth during construction: 5.4
Average depth at the end of operation: 1.5
Weighted Quick Union with grandparent node fix:
```

```
Average time:(with grandparent node fix) 39.51255 ms
Average depth during construction: 5.7
Average depth at the end of operation: 1.9

While N = 200000
Weighted Quick Union with all intermediate node fix:
Average time:(all intermediate node fix) 89.17628 ms
Average depth during construction: 5.7
Average depth at the end of operation: 1.6
Weighted Quick Union with grandparent node fix:
Average time:(with grandparent node fix) 78.04129 ms
Average depth during construction: 6.0
Average depth at the end of operation: 1.9

While N = 500000
Weighted Quick Union with all intermediate node fix:
Average time:(all intermediate node fix) 273.94531 ms
Average depth during construction: 6.0
Average depth at the end of operation: 1.8
Weighted Quick Union with grandparent node fix:
Average time:(with grandparent node fix) 263.78659 ms
Average depth during construction: 6.1
Average depth at the end of operation: 1.6
```

- **Relationship conclusion**

For Step 1 :

T1 and T2 are two disjoint trees where size(T2) >= size(T1). X is an element in T1. The depth of x increases by 1, only when T1 is placed below another tree T2. At this situation, the size of the merging tree will be at least double the size of T1 because size(T2) >= size(T1). The tree with x can double at most lg N times until it reaches a total of N. So, we can double up to lg N times and each time, our tree adds one level. To sum up, maximum lg N levels occur.

If the tree is weighted by size

$$h = 1, N \geq 1$$

$$h = 2, N \geq 2$$

$$h = 3, N \geq 2^2$$

$$h = 4, N \geq 2^3$$

$$if\ h = x\ , N \geq 2^x - 1 \text{ so, } h \leq lgN + 1 \text{ where}$$

N: tree size H: tree height

If the tree is weighted by depth the relation between h and N is the same.

So $h \leq lgN + 1$ for weighted by depth. Because of randomness of merging it could be lower than this. But using either size or depth will result in O(lg N) run time and depth for union and find.

For Step 2;

Zeynep Tufekci (NUID: 001565591)

In Path compression, both grandparent fix and intermediate fix are running similar time(performance) the average depth in construction and at the end of the construction is very close. It has O(lg*n) upper bound.
It can be shown (Tarjan, 1984) that with, using path-compression find makes any combination of up to N-1 Union operations and M Find operations have a worst-case time cost of
O (N + M log* N)

- **Evidence to support relationship**

For Step 1;

N		Performance (Time)	Average depth
100000	WQU with size	50.3709 ms	7.3
	WQU with depth	44.61872	7.0
200000	WQU with size	46.68597ms	7.7
	WQU with depth	48.5517ms	7.4
500000	WQU with size	46.68597ms	8.2
	WQU with depth	48.5517ms	7.7

Using either size or depth will result in at most O(log N) run time for union and find.

For Step2;

N		Performance (Time)	Average depth at the end of merging tree	Average depth during constructing tree
100000	WQUPC.java	37.31866 ms	5.4	1.5
	WQUPC alternative with grandparent set	39.51255 ms	5.7	1.9
200000	WQUPC.java	89.17628 ms	5.7	1.6
	WQUPC alternative with grandparent set	78.04129 ms	6.0	1.9
500000	WQUPC.java	273.94531 ms	6.0	1.8
	WQUPC alternative with grandparent set	263.78659 ms	6.1	1.6