Student Name: **Hongyao Tao (001067209)**

INFO 6205
Program Structures & Algorithms
Spring 2021

Assignment No.4

- **Implement depth-weighted Quick Union with 2 loop Path Compression**

```java
public void union2(int p, int q) {
    int rootP = find(p);
    int rootQ = find(q);
    if (rootP == rootQ) return;
    // make smaller root point to larger one
    if (depth[rootP] < depth[rootQ]) {
        parent[rootP] = rootQ;
    }else if (depth[rootP] > depth[rootQ]) {
        parent[rootQ]=rootP;
    }else {
        parent[rootQ] = rootP;
        depth[rootP] += 1;
    }
    count--;
}
```

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

    return root;
}
```

Figure 1. The implement of WQUPC class

- **Implement size-weighted Quick Union with 2 loop Path Compression**

```java
public void union(int p, int q) {
    int rootP = find(p);
    int rootQ = find(q);
    if (rootP == rootQ) return;
    // make smaller root point to larger one
    if (size[rootP] < size[rootQ]) {
        parent[rootP] = rootQ;
        size[rootQ] += size[rootP];
    } else {
        parent[rootQ] = rootP;
        size[rootP] += size[rootQ];
    }
    count--;
}
```

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

    return root;
}
```

Figure 2. The implement of WQUPC class

- **Implement size-weighted Quick Union with 2 loop Path Compression**

```java
private void mergeComponents(int i, int j) {
    int iroot=find(i);
    int jroot=find(j);
    if(iroot==jroot) return;
    int iRootHeight=height[iroot];
    int jRootHeight=height[jroot];
    if(iRootHeight>=jRootHeight){
        updateParent(jroot,iroot);
        updateHeight(iroot,jroot);
    }else {
        updateParent(iroot,jroot);
        updateHeight(jroot,iroot);
    }
    count--;

}
```

```java
private void doPathCompression(int i) {
    // TO BE IMPLEMENTED update parent to value of grandparent
    int grandparent=getParent(getParent(i));
    updateParent(i,grandparent);
}
```

```java
public int find(int p) {
    validate(p);

    int currentNode=p;
    int parent=getParent(currentNode);
    while (parent!=currentNode) {
        if(pathCompression){
            doPathCompression(currentNode);
        }
        currentNode=getParent(currentNode);
        parent=getParent(currentNode);
    }
    return parent;
}
```

Figure 3. The implement of UF_HWQUPC class

- **Implement of benchmark**

```java
@Test
public void benchmark() {
    int initialN = 500;
    writeToCSV( fileName: "CSV/Assignment4/randomPaires.csv",  line: "M,N,Time");
    int times=10;
    for (int i = 0; i < times; i++) {
        initialN *= 2;
        String description = "Random generator";
        UFPairHelper ufPairHelper = new UFPairHelper(initialN);
        WQUPC wqupc = new WQUPC(initialN);
        Supplier<List<List<Integer>>> supplier = () -> ufPairHelper.randomIntegePairs();
        Benchmark<List<List<Integer>>> benchmark = new Benchmark_Timer<>(
                description: description + " for " + initialN + " Integers",
                fPre: null,
                wqupc::benchMarkUnion,
                fPost: null
        );
        double average = benchmark.runFromSupplier(supplier,  m: 100);
        writeToCSV( fileName: "CSV/Assignment4/randomPaires.csv",  line: ufPairHelper.getPairsSize()+","+initialN + "," + average);
        System.out.println("Average milionSecond :" + average+", Pairs size :"+ufPairHelper.getPairsSize()+", N size :"+ initialN);
    }
}
```

```java
@Test
public void benchmark2() {
    int initialN = 500;
    writeToCSV( fileName: "CSV/Assignment4/alter_randomPaires.csv",  line: "M,N,Time");
    int times=10;
    for (int i = 0; i < times; i++) {
        initialN *= 2;
        String description = "Random generator";
        UFPairHelper ufPairHelper = new UFPairHelper(initialN);
        WQUPC wqupc = new WQUPC(initialN);
        Supplier<List<List<Integer>>> supplier = () -> ufPairHelper.randomIntegePairs();
        Benchmark<List<List<Integer>>> benchmark = new Benchmark_Timer<>(
                description: description + " for " + initialN + " Integers",
                fPre: null,
                wqupc::benchMarkUnion2,
                fPost: null
        );
        double average = benchmark.runFromSupplier(supplier,  m: 100);
        writeToCSV( fileName: "CSV/Assignment4/alter_randomPaires.csv",  line: ufPairHelper.getPairsSize()+","+initialN + "," + average);
        System.out.println("Average milionSecond :" + average+", Pairs size :"+ufPairHelper.getPairsSize()+", N size :"+ initialN);
    }
}
```

```java
@Test
public void benchmark3() {
    int initialN = 500;
    writeToCSV( fileName: "CSV/Assignment4/weight_randomPaires.csv",  line: "M,N,Time");
    int times=10;
    for (int i = 0; i < times; i++) {
        initialN *= 2;
        String description = "Random generator";
        UFPairHelper ufPairHelper = new UFPairHelper(initialN);
        UF_HWQUPC wqupc = new UF_HWQUPC(initialN,  pathCompression: true);
        Supplier<List<List<Integer>>> supplier = () -> ufPairHelper.randomIntegePairs();
        Benchmark<List<List<Integer>>> benchmark = new Benchmark_Timer<>(
                description: description + " for " + initialN + " Integers",
                fPre: null,
                wqupc::benchMarkUnionMerge,
                fPost: null
        );
        double average = benchmark.runFromSupplier(supplier,  m: 100);
        writeToCSV( fileName: "CSV/Assignment4/weight_randomPaires.csv",  line: ufPairHelper.getPairsSize()+","+initialN + "," + average);
        System.out.println("Average milionSecond :" + average+", Pairs size :"+ufPairHelper.getPairsSize()+", N size :"+ initialN);
    }
}
```

Figure 4. The implement of BenchmarkUF class

- **Unit test result**

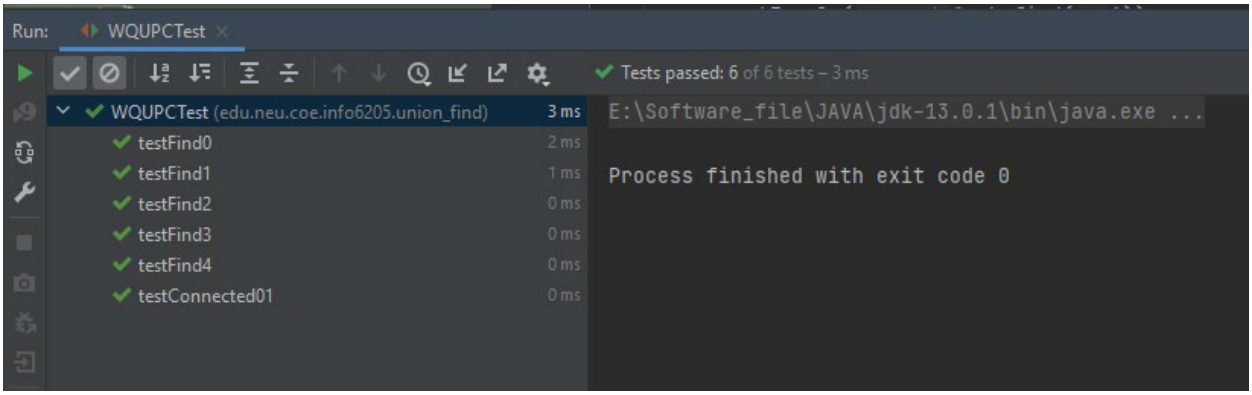

Figure 5. The result of unit test of WQUPC class

- **Comparison among different implementation**

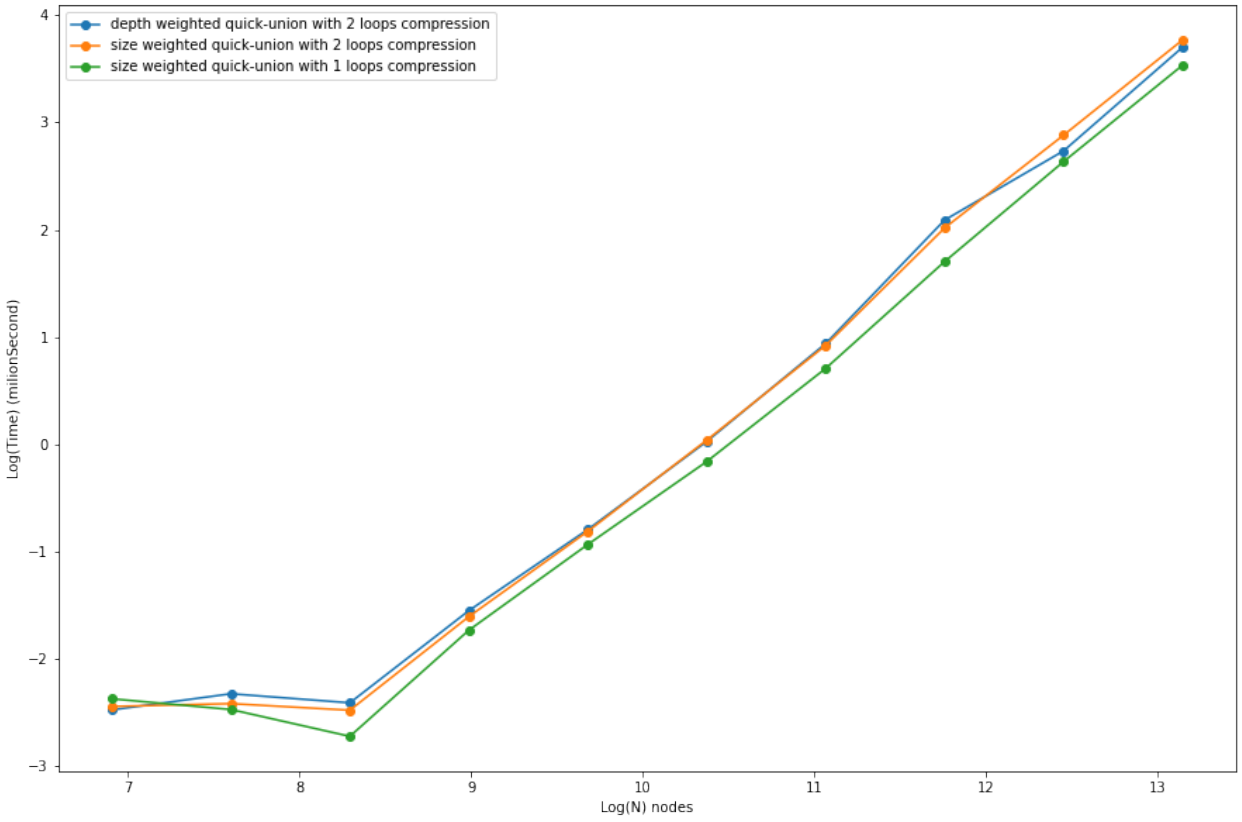

Figure 5. The line graph of Time and N under different QU implements

- **Evidence to support the conclusion:**

A	B	C
M	N	Time
846	1000	0.0843
1860	2000	0.097985
4085	4000	0.090048
8971	8000	0.212519
19704	16000	0.452509
43273	32000	1.02772
95037	64000	2.559924
208719	128000	8.089478
458385	256000	15.41076
1006700	512000	40.4607

M	N	Time
846	1000	0.087046
1860	2000	0.089396
4085	4000	0.084052
8971	8000	0.200992
19704	16000	0.444919
43273	32000	1.040843
95037	64000	2.51599
208719	128000	7.516121
458385	256000	17.8394
1006700	512000	43.31208

M	N	Time
846	1000	0.093453
1860	2000	0.084611
4085	4000	0.065893
8971	8000	0.17708
19704	16000	0.393679
43273	32000	0.852603
95037	64000	2.030659
208719	128000	5.493248
458385	256000	13.94291
1006700	512000	34.22699

Figure 4. The data of examination(Depth weighted quick-union with 2 loops compression, Size weighted quick-union with 2 loops compression and size weighted quick-union with 1 loops compression)

- **Conclusion**

According to Figure 5, the benchmark time of size weighted quick-union with 1 loop path compression algorithm always is below the orange line and blue line. It implies that the time complexity of the one-loop path compress algorithm is better than the time complex of two loops path compress algorithm. From the comparison between the orange line and blue line, the blue line almost same as the orange line, which demonstrates that using depth for weighted quick union is unnecessary since it has same time cost of using size.