

Program Structures & Algorithms

Spring 2022

Assignment No. 3

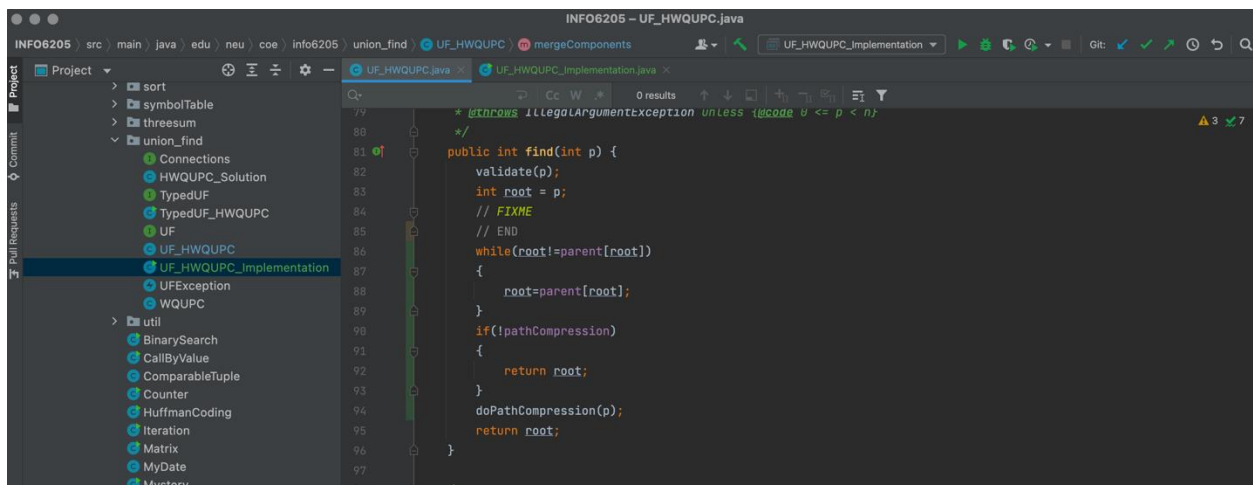
Sravya Ganda (002103774)

Task:

Expectation from the assignment is to implement a quick union with path compression by implementing three methods (find, Merge and doCompression) in UF_HWQUPC.java. There is a set of testcases that are to be passed. Secondly, we need to create random pairs of integers(m) between 0 to n-1 until our component becomes 1. Finally, a relationship must be defined between N and M.

Part 1:

Included the below logic and test cases passed.



```
INFO6205 - UF_HWQUPC.java
INFO6205 > src > main > java > edu > neu > coe > info6205 > union_find > UF_HWQUPC > UF_HWQUPC_Implementation
Project
  > sort
  > symbolTable
  > threesum
  > union_find
    > Connections
    > HWQUPC_Solution
    > TypedUF
    > TypedUF_HWQUPC
    > UF
    > UF_HWQUPC
    > UF_HWQUPC_Implementation
    > UFException
    > WQUPC
  > util
    > BinarySearch
    > CallByValue
    > ComparableTuple
    > Counter
    > HuffmanCoding
    > Iteration
    > Matrix
    > MyDate
    > Mustaru

79 * @throws IllegalArgumentException unless (0 <= p < n)
80 */
81 public int find(int p) {
82     validate(p);
83     int root = p;
84     // FIXME
85     // END
86     while(root != parent[root])
87     {
88         root = parent[root];
89     }
90     if(!pathCompression)
91     {
92         return root;
93     }
94     doPathCompression(p);
95     return root;
96 }
97
```

```

/**
 * This implements the single-pass path-halving mechanism of path compression
 */
private void doPathCompression(int i) {
    // FIXME update parent to value of grandparent
    // END
    parent[i]=parent[parent[i]];
}
}

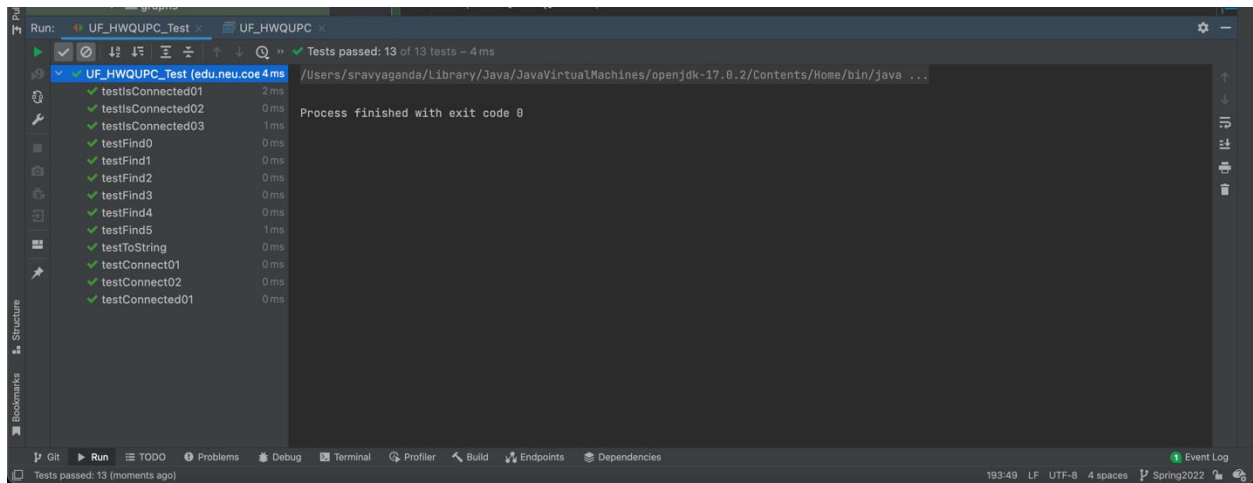
```

```

private void mergeComponents(int i, int j) {
    // FIXME make shorter root point to taller one
    // END
    int rootP=find(i);
    int rootQ=find(j);
    if(rootP == rootQ)
    {
        return;
    }
    if(height[rootP] < height[rootQ]) {
        parent[rootP] = rootQ;
        //height[rootP]++;
    }
    else if(height[rootP] == height[rootQ]){
        parent[rootQ]=rootP;
        height[rootP]++;
    }
    else
    {
        parent[rootQ] = rootP;
    }
}
}

```

Test Cases Screenshot



Task 2:

Created a new java class `UF_HWQUPC_Implementation.java` and implemented three methods (`main`, `count`, `createPairs`) the code screenshot is available below. Executed the random pairs method for 10 times and calculated the average M value for the accuracy.

1. For $N=50000$ then M is approximately 281149
2. For $N=60000$ then M is approximately 344558
3. For $N=70000$ then M is approximately 412777
4. For $N=80000$ then M is approximately 462611

```

public class UF_HWQUPC_Implementation {

    public static void main(String args[])
    {
        Scanner s=new Scanner(System.in);
        int n=s.nextInt();

        int avg=count(n);
        System.out.println("Average pairs for n "+ n + " is "+ avg );
    }

    public static int count(int n)
    {
        Queue<Integer> q=new LinkedList<Integer>();
        for(int i=1;i<=10;i++) {
            q.add(createPairs(n));
        }
        int avg=0;
        while(!q.isEmpty())
        {
            avg+=q.poll();
        }
        return avg/10;
    }
}

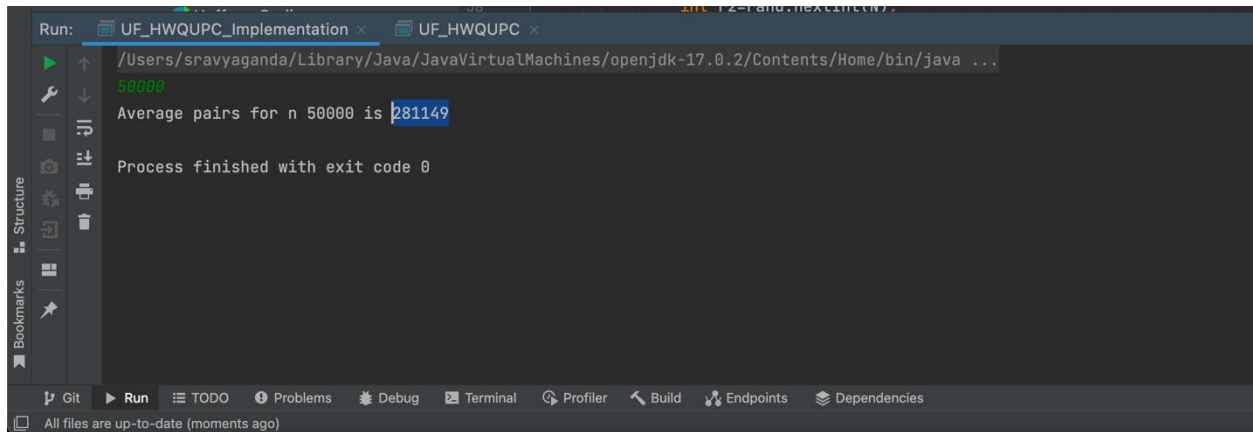
```

```

    public static int createPairs(int N)
    {
        UF_HWQUPC uf=new UF_HWQUPC(N, pathCompression: true);
        Random rand=new Random();
        int m=0;
        while(uf.components()!=1)
        {
            int r1=rand.nextInt(N);
            int r2=rand.nextInt(N);
            uf.connect(r1,r2);
            m++;
        }
        return m;
    }
}

```

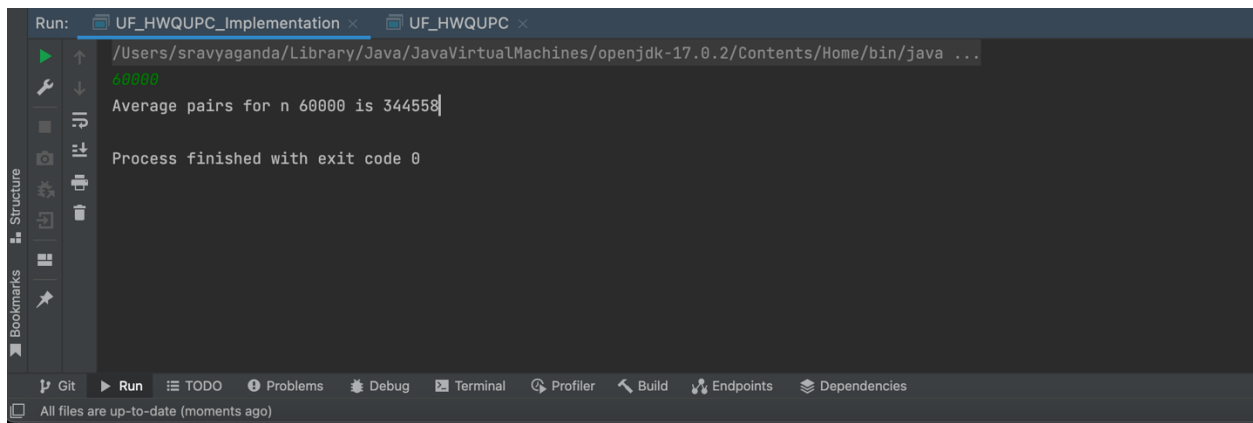
OUTPUT Screenshots



This screenshot shows the output of a Java program in an IDE terminal. The program calculates the average number of pairs for n=50000. The output is as follows:

```
Run: UF_HWQUPC_Implementation x UF_HWQUPC x
/Users/sravyaganda/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java ...
50000
Average pairs for n 50000 is 281149
Process finished with exit code 0
```

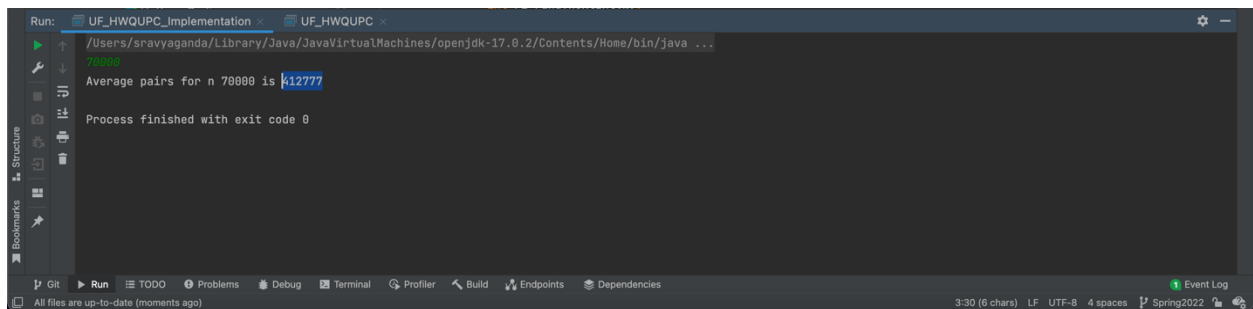
The IDE interface includes a sidebar with 'Structure' and 'Bookmarks' views, and a bottom toolbar with 'Git', 'Run', 'TODO', 'Problems', 'Debug', 'Terminal', 'Profiler', 'Build', 'Endpoints', and 'Dependencies'.



This screenshot shows the output of a Java program in an IDE terminal. The program calculates the average number of pairs for n=60000. The output is as follows:

```
Run: UF_HWQUPC_Implementation x UF_HWQUPC x
/Users/sravyaganda/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java ...
60000
Average pairs for n 60000 is 344558
Process finished with exit code 0
```

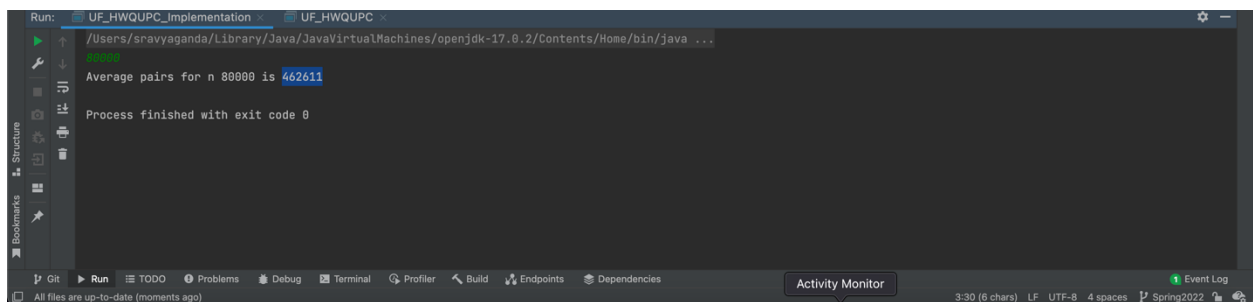
The IDE interface includes a sidebar with 'Structure' and 'Bookmarks' views, and a bottom toolbar with 'Git', 'Run', 'TODO', 'Problems', 'Debug', 'Terminal', 'Profiler', 'Build', 'Endpoints', and 'Dependencies'.



This screenshot shows the output of a Java program in an IDE terminal. The program calculates the average number of pairs for n=70000. The output is as follows:

```
Run: UF_HWQUPC_Implementation x UF_HWQUPC x
/Users/sravyaganda/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java ...
70000
Average pairs for n 70000 is 412777
Process finished with exit code 0
```

The IDE interface includes a sidebar with 'Structure' and 'Bookmarks' views, and a bottom toolbar with 'Git', 'Run', 'TODO', 'Problems', 'Debug', 'Terminal', 'Profiler', 'Build', 'Endpoints', and 'Dependencies'. An 'Event Log' icon is visible on the right side of the toolbar.



This screenshot shows the output of a Java program in an IDE terminal. The program calculates the average number of pairs for n=80000. The output is as follows:

```
Run: UF_HWQUPC_Implementation x UF_HWQUPC x
/Users/sravyaganda/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java ...
80000
Average pairs for n 80000 is 462611
Process finished with exit code 0
```

The IDE interface includes a sidebar with 'Structure' and 'Bookmarks' views, and a bottom toolbar with 'Git', 'Run', 'TODO', 'Problems', 'Debug', 'Terminal', 'Profiler', 'Build', 'Endpoints', and 'Dependencies'. An 'Activity Monitor' button is visible on the right side of the toolbar.

Task 3:

With different large values of N and from observations from the below table M is approximately N to the power of 1.16

N	$N^{1.16}$	M
50000	282361	281149
60000	348863	344558
70000	417170	412777
80000	487062	462611

$$M \sim N^{1.16}$$

We can deduce this relationship to

$$M = (N * \ln(N))/2$$