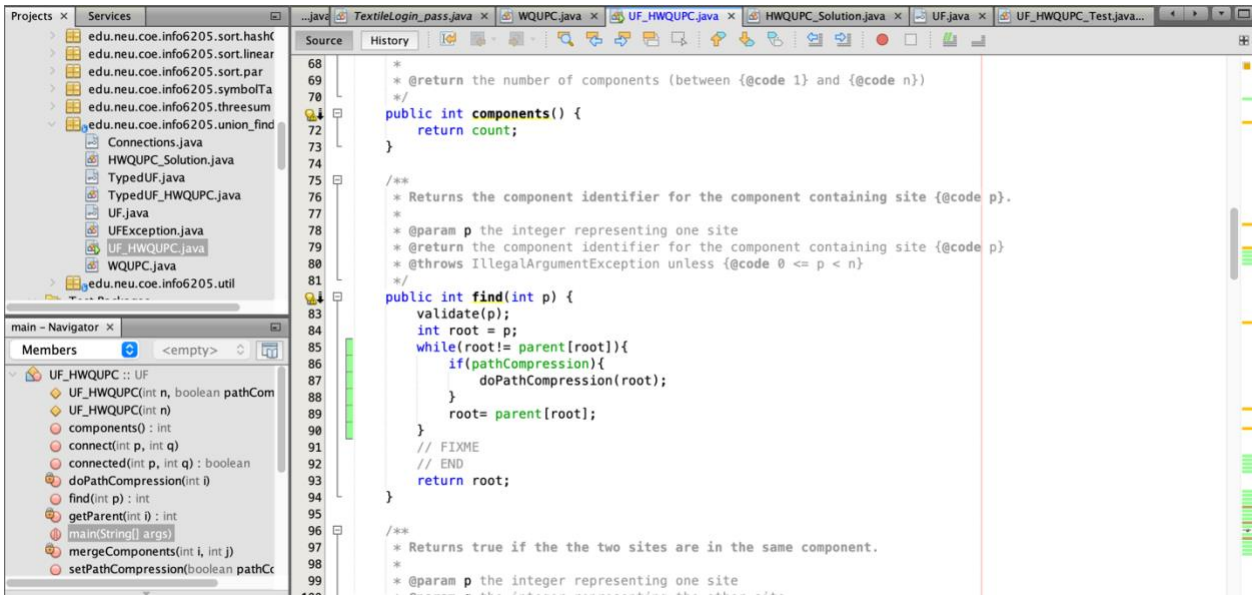
Program Structures and Algorithms
Spring 2022
Assignment No:3

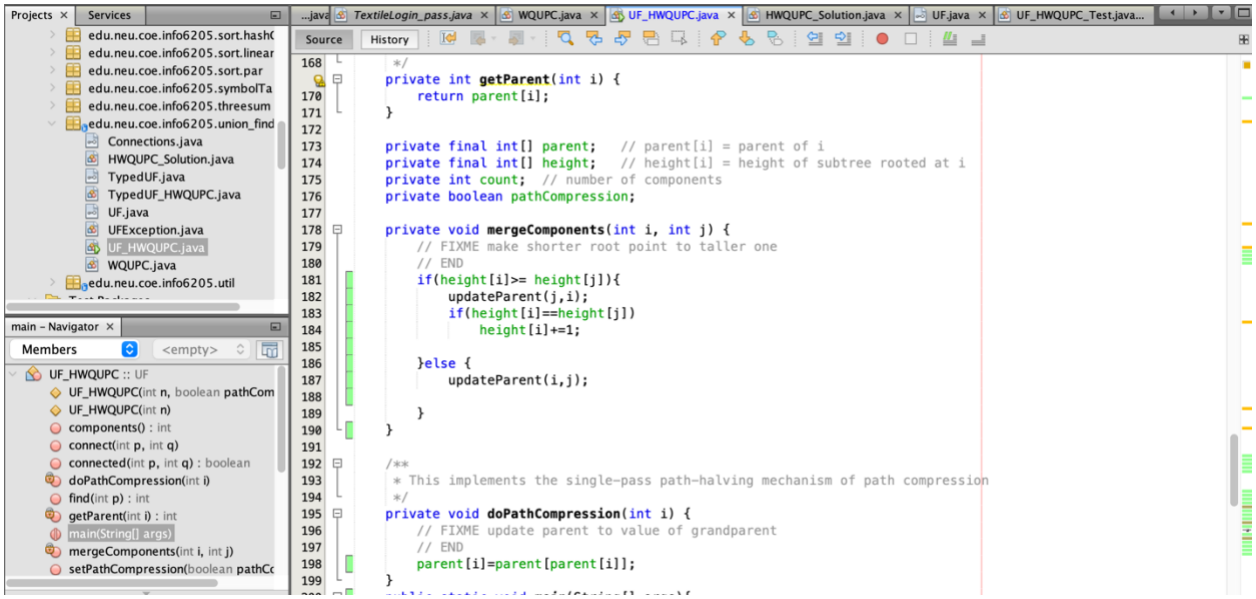
Name: Sangram Vuppala
NU ID: 002958963
Section: 01

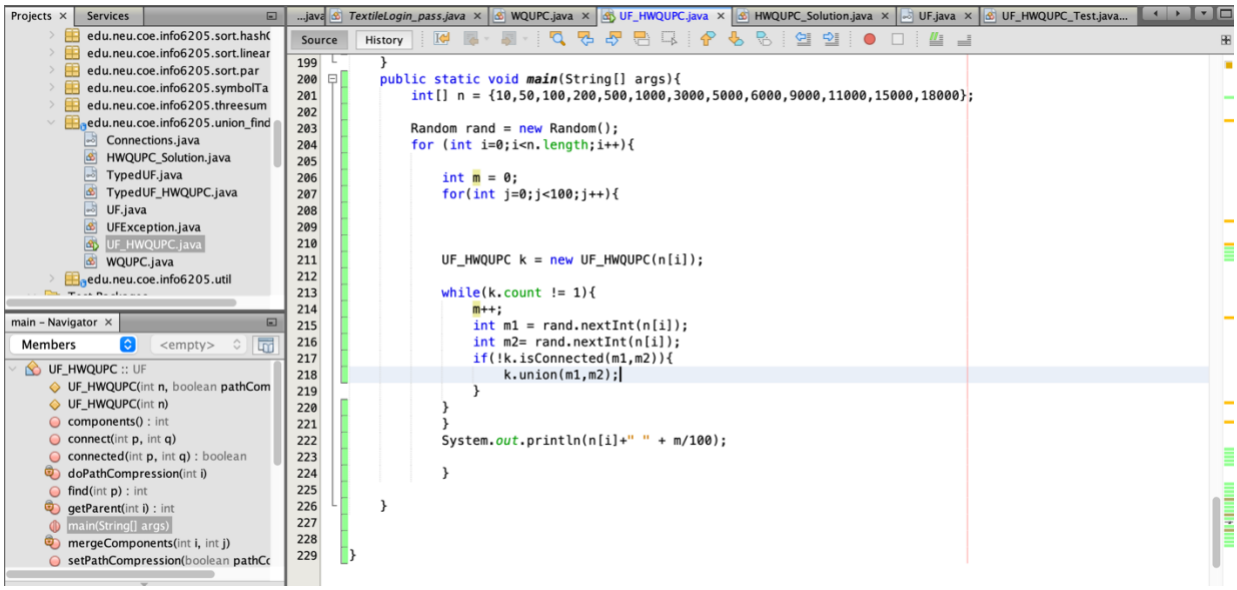
Task:

1. a) Implement height-weighted Quick Union with Path Compression in the class UF_HWQUPC
2. Using your implementation of UF_HWQUPC, develop a UF ("union-find") client that takes an integer value n from the command line to determine the number of "sites." Then generates random pairs of integers between 0 and n-1, calling connected() to determine if they are connected and union() if not. Loop until all sites are connected then print the number of connections generated. Package your program as a static method count() that takes n as the argument and returns the number of connections; and a main() that takes n from the command line, calls count() and prints the returned value. If you prefer, you can create a main program that doesn't require any input and runs the experiment for a fixed set of n values. Show evidence of your run(s).
3. Determine the relationship between the number of objects (*n*) and the number of pairs (*m*)

OUTPUT SCREEN:

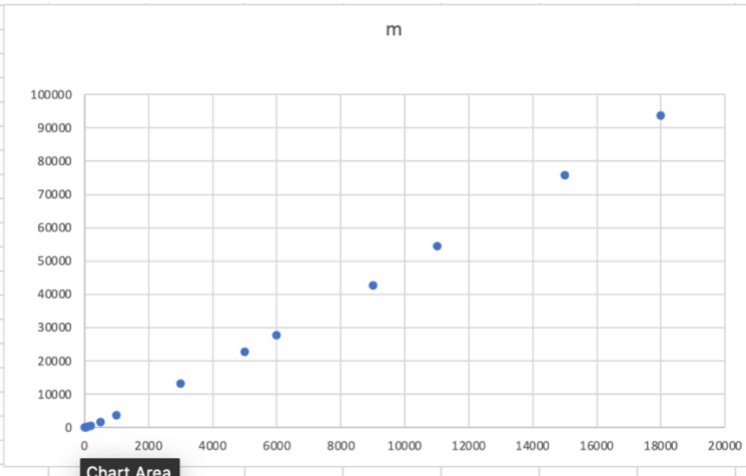



Projects
Services
edu.neu.coe.info6205.sort.hash(
edu.neu.coe.info6205.sort.linear
edu.neu.coe.info6205.sort.par
edu.neu.coe.info6205.symbolTa
edu.neu.coe.info6205.threesum
edu.neu.coe.info6205.union_find
Connections.java
HWQUPC_Solution.java
TypedUF.java
TypedUF_HWQUPC.java
UF.java
UFException.java
UF_HWQUPC.java
WQUPC.java
edu.neu.coe.info6205.util
main – Navigator
Members
<empty>
UF_HWQUPC :: UF
UF_HWQUPC(int n, boolean pathCom
UF_HWQUPC(int n)
components() : int
connect(int p, int q)
connected(int p, int q) : boolean
doPathCompression(int i)
find(int p) : int
getParent(int i) : int
main(String[] args)
mergeComponents(int i, int j)
setPathCompression(boolean pathCc
...java
TextileLogin_pass.java
WQUPC.java
UF_HWQUPC.java
HWQUPC_Solution.java
UF.java
UF_HWQUPC_Test.java...
Source
History
168 */
169 private int getParent(int i) {
170 return parent[i];
171 }
172
173 private final int[] parent; // parent[i] = parent of i
174 private final int[] height; // height[i] = height of subtree rooted at i
175 private int count; // number of components
176 private boolean pathCompression;
177
178 private void mergeComponents(int i, int j) {
179 // FIXME make shorter root point to taller one
180 // END
181 if(height[i]>= height[j]){
182 updateParent(j,i);
183 if(height[i]==height[j])
184 height[i]+=1;
185
186 }else {
187 updateParent(i,j);
188
189 }
190 }
191
192 /**
193 * This implements the single-pass path-halving mechanism of path compression
194 */
195 private void doPathCompression(int i) {
196 // FIXME update parent to value of grandparent
197 // END
198 parent[i]=parent[parent[i]];
199 }


Projects
Services
edu.neu.coe.info6205.sort.hash(
edu.neu.coe.info6205.sort.linear
edu.neu.coe.info6205.sort.par
edu.neu.coe.info6205.symbolTa
edu.neu.coe.info6205.threesum
edu.neu.coe.info6205.union_find
Connections.java
HWQUPC_Solution.java
TypedUF.java
TypedUF_HWQUPC.java
UF.java
UFException.java
UF_HWQUPC.java
WQUPC.java
edu.neu.coe.info6205.util
main – Navigator
Members
<empty>
UF_HWQUPC :: UF
UF_HWQUPC(int n, boolean pathCom
UF_HWQUPC(int n)
components() : int
connect(int p, int q)
connected(int p, int q) : boolean
doPathCompression(int i)
find(int p) : int
getParent(int i) : int
main(String[] args)
mergeComponents(int i, int j)
setPathCompression(boolean pathCc
...java
TextileLogin_pass.java
WQUPC.java
UF_HWQUPC.java
HWQUPC_Solution.java
UF.java
UF_HWQUPC_Test.java...
Source
History
199 }
200 public static void main(String[] args){
201 int[] n = {10,50,100,200,500,1000,3000,5000,6000,9000,11000,15000,18000};
202
203 Random rand = new Random();
204 for (int i=0;i<n.length;i++){
205
206 int m = 0;
207 for(int j=0;j<100;j++){
208
209
210
211 UF_HWQUPC k = new UF_HWQUPC(n[i]);
212
213 while(k.count != 1){
214 m++;
215 int m1 = rand.nextInt(n[i]);
216 int m2= rand.nextInt(n[i]);
217 if(!k.isConnected(m1,m2)){
218 k.union(m1,m2);
219 }
220 }
221 }
222 System.out.println(n[i]+" " + m/100);
223
224 }
225
226 }
227
228
229 }

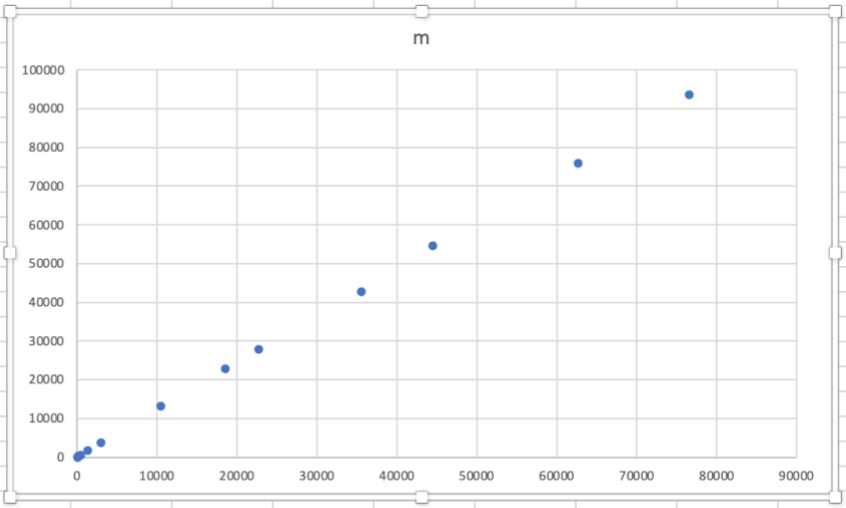
GRAPHS AND EVIDENCE:
Graph between N and m : with different values of N and avg m value for 100 experiments

N	m
10	16
50	112
100	263
200	603
500	1739
1000	3685
3000	13235
5000	22841
6000	27796
9000	42729
11000	54601
15000	75826
18000	93675



Graph between NlnN and m : with different values of N and avg m value for 100 experiments

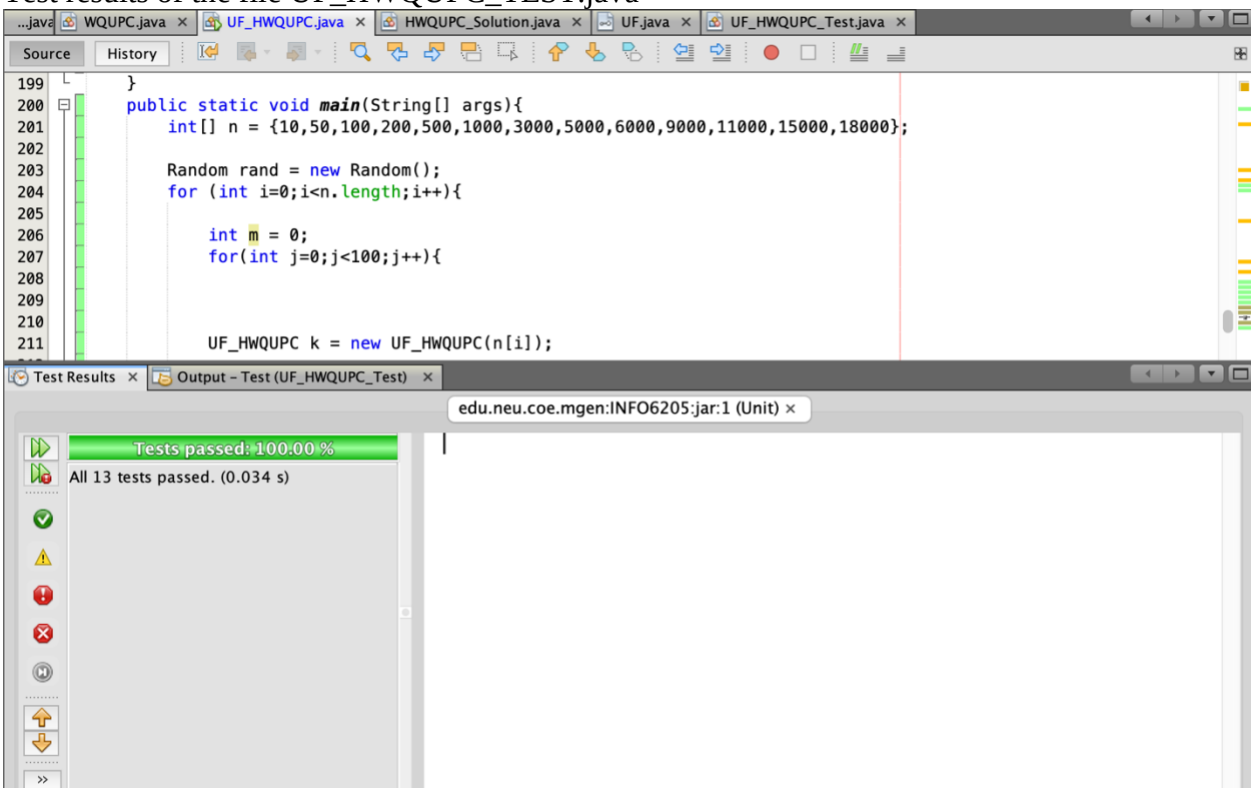
NlnN	m
10	16
84.9485002	112
200	263
460.205999	603
1349.485	1739
3000	3685
10431.3638	13235
18494.85	22841
22668.9075	27796
35588.1826	42729
44455.3195	54601
62641.3689	75826
76594.9051	93675



RELATIONSHIP & CONCLUSION:
We can conclude from the above graph that, NlnN is linearly proportional to m which implies NlnN/m = k(constant).

UNIT TEST RESULTS:

Test results of the file UF_HWQUPC_TEST.java



Test results of the file WQUPCTest.java