djList \*listA = new AdjList();  
 listA->addV(0);  
 listA->addV(1);  
 listA->addE(0,1);  
 listA->getAllE(0);  
 listA->addV(2);  
 listA->addE(0,2);  
 listA->getAllE(0);  
 if(listA->checkE(0,2))  
 cout << "(0,2) exists" << endl;  
 if(listA->checkE(0,3))  
 cout << "(0,3) exists" << endl;  
 listA->addV(3);  
 listA->addE(1,3);  
 if(listA->checkE(1,3))  
 cout << "(1,3) exists" << endl;  
 listA->removeE(1,3);  
 if(listA->checkE(1,3))  
 cout << "(1,3) exists" << endl;  
 listA->removeV(3);  
  
 //tests for adjacency matrix implementation  
  
 AdjMatrix \*matA = new AdjMatrix();  
 matA->addV();  
 matA->addV();  
 matA->addE(0,1);  
 matA->addV();  
 matA->addE(0,2);  
 matA->getAllE(0);  
 matA->removeE(0,2);  
 matA->getAllE(0);  
 matA->removeV(1);  
 matA->getAllE(0);  
 matA->addE(0,1);  
 if(matA->checkE(0,1))  
 cout << "(0,1) exists" << endl;  
  
  
 //test cases for DFS  
  
 cout << "Cycle" << endl;  
 //test cases for adjacency list  
 AdjList\* listT = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 9; i++)  
 {  
 listT->addE(i,i+1);  
 }  
 listT->addE(9,0);  
 auto start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 99; i++)  
 {  
 listT->addE(i,i+1);  
 }  
 listT->addE(99,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 999; i++)  
 {  
 listT->addE(i,i+1);  
 }  
 listT->addE(999,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 9999; i++)  
 {  
 listT->addE(i,i+1);  
 }  
 listT->addE(9999,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
  
 cout << "Clique" << endl;  
 //test cases for clique  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
 cout << "Cycle" << endl;  
 //Test Cases for BFS  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 9; i++)  
 {  
 listT->addE(i,i+1);  
 }  
 listT->addE(9,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 99; i++)  
 {  
 listT->addE(i,i+1);  
 }  
 listT->addE(99,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 999; i++)  
 {  
 listT->addE(i,i+1);  
 }  
 listT->addE(999,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 9999; i++)  
 {  
 listT->addE(i,i+1);  
 }  
 listT->addE(9999,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
  
 cout << "Clique" << endl;  
 //test cases for clique  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
  
  
  
  
  
 cout << "Random-25%" << endl;  
 //test cases for BFS  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 int randN = rand() % 100;  
 if(randN <= 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 int randN = rand() % 100;  
 if(randN <= 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN <= 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN <= 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
 //tests cases for BFS  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 int randN = rand() % 100;  
 if(randN <= 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 int randN = rand() % 100;  
 if(randN <= 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN <= 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN <= 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
 cout << "Random-75%" << endl;  
 //test cases for BFS  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 int randN = rand() % 100;  
 if(randN > 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 int randN = rand() % 100;  
 if(randN > 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN > 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN > 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->DFS(0);  
 cout << "Time taken to run DFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
 //tests cases for BFS  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 int randN = rand() % 100;  
 if(randN > 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 int randN = rand() % 100;  
 if(randN > 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN > 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete listT;  
 //test cases for adjacency list  
 listT = new AdjList();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 listT->addV(i);  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN > 25)  
 listT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listT->BFS(0);  
 cout << "Time taken to run BFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
  
  
  
 //Tests for Adjacency Matrix  
  
 //Tests for DFS  
  
 cout << "Matrix" << endl;  
 cout << "Cycle" << endl;  
 //test cases for adjacency Matrix  
 AdjMatrix\* MatrixT = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 9; i++)  
 {  
 MatrixT->addE(i,i+1);  
 }  
 MatrixT->addE(9,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 99; i++)  
 {  
 MatrixT->addE(i,i+1);  
 }  
 MatrixT->addE(99,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 999; i++)  
 {  
 MatrixT->addE(i,i+1);  
 }  
 MatrixT->addE(999,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 9999; i++)  
 {  
 MatrixT->addE(i,i+1);  
 }  
 MatrixT->addE(9999,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
  
 cout << "Clique" << endl;  
 //test cases for clique  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
 cout << "Cycle" << endl;  
 //Test Cases for BFS  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 9; i++)  
 {  
 MatrixT->addE(i,i+1);  
 }  
 MatrixT->addE(9,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 99; i++)  
 {  
 MatrixT->addE(i,i+1);  
 }  
 MatrixT->addE(99,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 999; i++)  
 {  
 MatrixT->addE(i,i+1);  
 }  
 MatrixT->addE(999,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 9999; i++)  
 {  
 MatrixT->addE(i,i+1);  
 }  
 MatrixT->addE(9999,0);  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
  
 cout << "Clique" << endl;  
 //test cases for clique  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
 cout << "Random-25%" << endl;  
 //test cases for random-25%  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 int randN = rand() % 100;  
 if(randN>=25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 int randN = rand() % 100;  
 if(randN>=25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN>=25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN>=25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
 //test cases for random-25%  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 int randN = rand() % 100;  
 if(randN>=25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 int randN = rand() % 100;  
 if(randN>=25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN>=25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN>=25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
  
 cout << "Random-75%" << endl;  
 //test cases for random-25%  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 int randN = rand() % 100;  
 if(randN < 25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 int randN = rand() % 100;  
 if(randN<25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN<25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN<25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->DFS(0);  
 cout << "Time taken to run DFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
 //test cases for random-75%  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 10; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 10;j++)  
 {  
 int randN = rand() % 100;  
 if(randN<25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 10 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 100  
 for(int i = 0; i < 100; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 100; i++)  
 {  
 for(int j = 0; j < 100;j++)  
 {  
 int randN = rand() % 100;  
 if(randN<25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 100 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 1000  
 for(int i = 0; i < 1000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 1000; i++)  
 {  
 for(int j = 0; j < 1000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN<25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 1000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
 //Computer could not complete 10,000  
 /\*  
 delete MatrixT;  
 //test cases for adjacency Matrix  
 MatrixT = new AdjMatrix();  
 //cycle of 10000  
 for(int i = 0; i < 10000; i++)  
 {  
 MatrixT->addV();  
 }  
 for(int i = 0; i < 10000; i++)  
 {  
 for(int j = 0; j < 10000;j++)  
 {  
 int randN = rand() % 100;  
 if(randN<25)  
 MatrixT->addE(i,j);  
 }  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 MatrixT->BFS(0);  
 cout << "Time taken to run BFS on 10000 vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 \*/  
  
  
  
 fstream fin;  
 fin.open("Program5/Stan.txt",ios::in);  
 string line;  
 int item;  
 getline(fin,line);  
 getline(fin,line);  
 getline(fin,line);  
 getline(fin,line);  
 vector<int> d;  
 cout << "Stanford Data Testing" << endl;  
 //test cases for adjacency list  
 AdjList\* listS = new AdjList();  
 //cycle of 10  
 for(int i = 0; i < 6301; i++)  
 {  
 listS->addV(i);  
 }  
 for(int i = 0; i < 20777; i++)  
 {  
 getline(fin,line);  
 istringstream s(line);  
 while(s >> item)  
 d.push\_back(item);  
 listS->addE(d[0],d[1]);  
 d.clear();  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 listS->DFS(0);  
 cout << "Time taken to run DFS on Stanford Data vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 start\_time = chrono::high\_resolution\_clock::now();  
 listS->BFS(0);  
 cout << "Time taken to run BFS on Stanford Data vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 fin.close();  
  
  
 fin.open("Program5/Stan.txt",ios::in);  
 getline(fin,line);  
 getline(fin,line);  
 getline(fin,line);  
 getline(fin,line);  
 cout << "Stanford Data Testing for Adjacency Matrix" << endl;  
 //test cases for adjacency list  
 AdjMatrix\* matrixS = new AdjMatrix();  
 //cycle of 10  
 for(int i = 0; i < 6301; i++)  
 {  
 matrixS->addV();  
 }  
 for(int i = 0; i < 20777; i++)  
 {  
 getline(fin,line);  
 istringstream s(line);  
 while(s >> item)  
 d.push\_back(item);  
 matrixS->addE(d[0],d[1]);  
 d.clear();  
 }  
 start\_time = chrono::high\_resolution\_clock::now();  
 matrixS->DFS(0);  
 cout << "Time taken to run DFS on Stanford Data vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 start\_time = chrono::high\_resolution\_clock::now();  
 matrixS->BFS(0);  
 cout << "Time taken to run BFS on Stanford Data vertices: " << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
  
}

The beginning section shows the testing of each function on both implementations. The rest shows the extensive testing of the DFS and BFS algorithms. This also test both the addV and addE functions.