IT 328 – Programming Assignment 1

3SAT <=p Clique, Independent Sets <=p Clique

9/23/15

Andrew Hinh ULID: ahinh

Brandon Burton ULID: bmburt1

Unix Login: bmburt1

Unix Directory: ~/IT328/…

..

To find the maximum independent set, we first reduced the independent set problem into a clique-problem. This is acceptable because it is proven that the solution in calculating the maximum independent set is the same level of difficulty as finding the solution to a clique. Because a clique problem can be solved in polynomial time, we reduced the maximum independent set problem to a clique problem to be solved more efficiently.

In our code, we first had to convert the graphs from graphs.txt into a new file that contains the same graphs but inversed. We use the inverse of the independent set graphs for the clique because the relationship between the two is that for any given graph G, the maximum independent set of G is the equivalent as the maximum clique of the inverse of G, or G’. The hardest part was just understanding the concept and trying to figure out which data structure to modify the graphs in. Once you understood what a maximum set was, a clique, and understanding the relationship between the two; reducing the problem went by a lot easier.

Code Output:

**FindAssignment.java**

G1 ( 2, 1) [0, 1] (size=2, 0ms)

G2 ( 2, 1) [0, 1] (size=2, 0ms)

G3 ( 4, 5) [0, 1, 2] (size=3, 1ms)

G4 ( 4, 5) [0, 1, 3] (size=3, 0ms)

G5 ( 6, 13) [0, 1, 2, 5] (size=4, 3ms)

G6 ( 6, 10) [1, 2, 3, 4] (size=4, 1ms)

Source code:

**Graph.java**

**import** java.util.\*;

/\*\*

\* This program was created by Andrew Hinh and Brandon Burton for IT 328

\* **@author** ahinh,bburt

\*

\*/

**public** **class** Graph {

ArrayList<ArrayList<Integer>> vertexMap;

// ArrayList<ArrayList<Integer>> candidateList;

// Stack<Integer> currentClique = new Stack<Integer>();

//remember the adjacency list

**private** **byte**[][] edgeMatrix;

//the size of the the matrix (n)

**private** **int** vertexCount = 0;

**private** **int** maxCliqueSize = 0;

**private** **int** edgeCount = 0;

**private** Set<Integer> maximalClique;

**public** Graph(**int** size) {

edgeMatrix = **new** **byte**[size][size];

// candidateList = new ArrayList<ArrayList<Integer>>();

vertexMap = **new** ArrayList<ArrayList<Integer>>();

vertexCount = size;

maximalClique = **new** HashSet<Integer>();

}

**public** **void** addVertex(**int** row, String connectedList) {

String[] split = connectedList.split(" ");

// ArrayList<Integer> arr = new ArrayList<Integer>();

**try** {

**for** (**int** i = 0; i < vertexCount; i++)

{

**byte** val = Byte.*parseByte*(split[i]);

edgeMatrix[row][i] = val;

**if** (val == 1 && row != i)

edgeCount+=1;

}

} **catch** (NumberFormatException nfe) {

System.***out***.println("Error");

}

// vertexMap.add(arr);

//edgeList.add(adjacencyList);

}

/\*\*

\* This function will calculate the maximum clique

\*/

**public** String calculateMaximumClique() {

Calendar cal = Calendar.*getInstance*();

**long** timeBefore = System.*currentTimeMillis*();

Stack<Integer> s = **new** Stack<Integer>();

**for** (**int** i = 0; i < vertexCount; i++)

{

s.push(i);

}

recursiveFind(**new** Stack<Integer>(), s, **new** HashSet<Integer>());

**long** timeAfter = System.*currentTimeMillis*();

//System.out.println("The maximal clique is " + maximalClique);

StringBuilder sb = **new** StringBuilder();

sb.append("( "+vertexCount+", "+(edgeCount / 2)+") ");

sb.append(maximalClique);

sb.append(" (size=" + maximalClique.size()+", ");

sb.append((timeAfter - timeBefore) + "ms)");

**return** sb.toString();

}

/\*\*

\*

\* **@param** potentialStack

\* **@param** remainderStack

\* **@param** skipStack

\* **@return**

\*/

**public** **synchronized** **void** recursiveFind(Stack<Integer> potentialStack, Stack<Integer> remainderStack, Set<Integer> skipStack) {

// uncomment this to see the calculation

// StringBuilder s = new StringBuilder();

// s.append(" P: ");

// s.append(potentialStack);

// s.append(" R: ");

// s.append(remainderStack);

// s.append(" S: ");

// s.append(skipStack);

// System.out.println(s.toString());

**if** (remainderStack.size() == 0 && skipStack.size() == 0)

{

HashSet<Integer> clique = **new** HashSet<Integer>();

clique.addAll(potentialStack);

**if** (clique.size() > maximalClique.size()) {

// System.out.println("The highest clique is now " + clique + "\n");

maximalClique = clique;

}

}

**for** (Integer pivotVertex : remainderStack)

{

//remainderStack.remove(pivotVertex);

//

// if () {

//

// }

// else {

Stack<Integer> newPotentialStack = **new** Stack<Integer>();

Stack<Integer> newRemainderStack = **new** Stack<Integer>();

Set<Integer> newSkipSet = **new** HashSet<Integer>();

newPotentialStack.addAll(potentialStack);

newPotentialStack.add(pivotVertex);

newRemainderStack.addAll(getNeighbors(remainderStack, pivotVertex));

newRemainderStack.remove(pivotVertex);

newSkipSet.addAll(getNeighbors(skipStack, pivotVertex));

//newSkipNodes.add(pivotVertex);

recursiveFind(newPotentialStack, newRemainderStack, newSkipSet);

skipStack.add(pivotVertex);

// }

}

}

/\*\*

\* This method returns a data structure that holds

\* **@param** workingStack

\* **@param** vertex

\* **@return**

\*/

**private** Stack<Integer> getNeighbors(Stack<Integer> workingStack, **int** vertex) {

Stack<Integer> stack = **new** Stack<Integer>();

**for** (**int** i = 0; i < workingStack.size(); i++)

{

**int** j = workingStack.get(i);

**if** (edgeMatrix[vertex][j] == 1) {

stack.add(j);

}

}

**return** stack;

}

/\*\*

\*

\* **@param** workingSet

\* **@param** vertex

\* **@return**

\*/

**private** Stack<Integer> getNeighbors(Set<Integer> workingSet, **int** vertex) {

Stack<Integer> stack = **new** Stack<Integer>();

Integer[] candidates = workingSet.toArray(**new** Integer[workingSet.size()]);

**for** (**int** i = 0; i < workingSet.size(); i++)

{

**if** (edgeMatrix[vertex][i] == 1) {

stack.add(i);

}

}

**return** stack;

}

**public** String toString() {

//return "This graph has " + vertexCount + " vertices and " + edgeIndex.size() + " edges\n";

//return "GX ( " + vertexCount + ", " + edgeIndex.size() + ") size=" + 0 + "\n";

StringBuilder b = **new** StringBuilder();

**for** (**int** i = 0; i < vertexCount; i++)

{

**for** (**int** j = 0; j < vertexCount; j++)

{

b.append(edgeMatrix[j][i]+",");

}

b.append("\n");

}

**return** b.toString();

}

**public** **int** getSize() {

**return** vertexCount;

}

}

**Main.java**

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.util.ArrayList;

/\*\*

\* This was made by Andrew Hinh and Brandon Burton for IT 328

\* **@author** ahinh, bburt

\*

\*/

**public** **class** Main {

**public** **static** **void** main(String args[])

{

ArrayList<Graph> graphList;

//File file = new File("H:/IT328/Assignment1/graphs.txt");

File file = **new** File("graphsDense.txt");

FileInputStream fis = **null**;

BufferedReader reader = **null**;

graphList = **new** ArrayList<Graph>();

**try** {

fis = **new** FileInputStream(file);

reader = **new** BufferedReader(**new** InputStreamReader(fis));

String line = reader.readLine();

**while** (line != **null**) {

**int** graphSize = Integer.*valueOf*(line);

Graph g = **new** Graph(graphSize);

**for** (**int** i = 0; i < graphSize; i++)

{

g.addVertex(i, reader.readLine());

}

line = reader.readLine();

**if** (graphSize > 0)

{

**if** (graphSize < 8)

System.***out***.println("G"+(graphList.size() + 1) + " " + g.calculateMaximumClique());

//now that the graph is populated correctly and work is done, add it to the list

graphList.add(g);

}

}

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if** (fis != **null**)

reader.close();

fis.close();

} **catch** (IOException ex) {

ex.printStackTrace();

}

}

}

}

**FindAssignment.java**

**import** java.io.\*;

**import** java.util.\*;

/\*\*

\* This program was created by Andrew Hinh and Brandon Burton for IT 328

\* **@author** ahinh,bburt

\*

\*/

**public** **class** FindAssignment {

**public** **static** **void** main(String args[])

{

ArrayList<Graph> graphList;

//File file = new File("H:/IT328/Assignment1/graphs.txt");

//File file = new File("graphs.txt");

FileInputStream fis = **null**;

BufferedReader reader = **null**;

**int** numVertices;

**int**[][] graph;

//Convert graphs.txt to new file and read in new file

**try** {

Scanner scan = **new** Scanner(**new** File("graphs.txt"));

FileWriter fw = **new** FileWriter(**new** File("inverseGraphs.txt"));

**while**(scan.hasNextLine())

{

numVertices = scan.nextInt();

**if**(numVertices==0)

{

**break**;

}

graph = **new** **int**[numVertices][numVertices];

**for**(**int** row = 0; row < numVertices; row++)

{

**for**(**int** col = 0; col < numVertices; col++)

{

graph[row][col] = scan.nextInt();

}

}

//rewrite graph to InverseGraphs.txt

fw.write(numVertices+"\r\n");

**for**(**int** row = 0; row < numVertices; row++)

{

**for**(**int** col = 0; col < numVertices; col++)

{

**if**(row == col)

fw.write(1 + " ");

**else** **if** (graph[row][col] == 1)

fw.write(0 + " ");

**else**

fw.write(1 + " ");

}

fw.write("\r\n");

}

}

fw.close();

} **catch** (FileNotFoundException e2) {

// **TODO** Auto-generated catch block

e2.printStackTrace();

}

**catch** (IOException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

graphList = **new** ArrayList<Graph>();

**try** {

fis = **new** FileInputStream(**new** File("InverseGraphs.txt"));

reader = **new** BufferedReader(**new** InputStreamReader(fis));

String line = reader.readLine();

**while** (line != **null**) {

**int** graphSize = Integer.*valueOf*(line);

Graph g = **new** Graph(graphSize);

**for** (**int** i = 0; i < graphSize; i++)

{

g.addVertex(i, reader.readLine());

}

line = reader.readLine();

**if** (graphSize > 0)

{

**if** (graphSize < 8)

System.***out***.println("G"+(graphList.size() + 1) + " " + g.calculateMaximumClique());

//now that the graph is populated correctly and work is done, add it to the list

graphList.add(g);

}

}

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if** (fis != **null**)

reader.close();

fis.close();

} **catch** (IOException ex) {

ex.printStackTrace();

}

}

}

}