This document is in compliance with SMDS standard version *3.5*

|  |  |
| --- | --- |
| **Description:** | Calculates the possibility of having a BiColorable (bipartite) set of nodes when they are connected through a number of edges through searching for two adjacent nodes with the same color. |

**Development Estimates/Actuals**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Est Sz** | **Act Sz** |  | **Strt Date** | **Est Cmplt** | **Act Cmplt** |  | **Est Effrt** | **Act Effrt** |
| Reqs | 14 | 17 |  | 9/1/2017 | 9/1/2017 | 9/4/2017 |  | 2:00 | 1:30 |
| Leaf Inpt Part | 20 | 13 |  | 9/21/17 | 9/21/17 | 9/21/17 |  | 0:30 | 0:30 |
| Test Cases | 15 | 12 |  | 9/21/17 | 9/21/17 | 9/21/17 |  | 0:30 | 0:30 |
| Design |  |  |  | 9/21/17 | 9/21/17 | 9/21/17 |  | 1:00 | 1:00 |
| # Classes | 1 | 1 |  |  |  |  |  |  |  |
| # Methods | 4 | 2 |  |  |  |  |  |  |  |
| # Dsng Elms | 10 | 12 |  |  |  |  |  |  |  |
| Implementatn |  |  |  | 10/1/17 | 10/5/17 | 10/5/17 |  | 2:00 | 5:30 |
| NCLC | 5 | 5 |  |  |  |  |  |  |  |
| Correctness |  |  |  | 10/24/17 | 10/26/17 | 11/1/17 |  | 1:00 | 2:00 |
| Rand Test |  |  |  | 10/24/17 | 10/26/17 | 11/1/17 |  | 3:00 | 8:00 |
| Final Test |  |  |  | 12/5/17 | 12/5/17 | 12/5/17 |  | 2:00 | 2:00 |
| Wrap up |  |  |  | 12/5/17 | 12/5/17 | 12/5/17 |  | 0:30 | 0:30 |
|  |  |  |  |  |  | Total Effrt: |  | 12.5 hrs | 21 hrs |

Estimates/Actuals Comments

* See accompanying log: smdsForBiColorableTskLog*.*xlsx

**Requirements**

The priority of these requirements is imperative.

*\*\*Constraints*

1. BiColorableFNA is coded in C++  
   Validation: Review and Test(Build).
2. The source file for BiColorableFNA is in compliance with *MTM C++ Source File Standard v4.0*.  
   Validation: Review
3. The SMDS is conforms to version 2.4 of the standard

*\*\*Preconditions*

1. An input file exists in a place where it can be accessed when this program is invoked.  
   Validation: Test

*\*\*Invocation*

1. Launch application execution file.  
   Validation: Test

*\*\*Input/Ouput*

1. Input begins with a positive integer, nmbrNodes, indicating the number of nodes in the file.  
   Validation: Test
2. Second input is a positive integer, nmbrEdges, indicating the number of edges in the file.

Validation: Test

1. Two integers, the first node in line l, firstNode and the second node in line l, secondNode separated with a single space.

Validation: Test

1. Last input, endProcess, indicates the end of inputs. EOF when endProcess is equal to zero.

Validation: Test

1. firstNodes and secondNode values are restricted such that 0 <= firstNode < nmbrNodes and 0 <= secondNode < nmbrNodes.  
   Validation: Guaranteed
2. The restrictions on the input values are 1 < nmbrNodes < 200.  
   1 < nmbrEdges <= nmbrNodes.  
   Validation: Guaranteed
3. Output prints out “BICOLORABLE” or “NOT BICOLORABLE”. No other options available for output.  
   Validation: Test and Review

*\*\*Postconditons*

1. The R04 input file is unchanged.  
   Validation: Review and Test

*\*\*Testing*

1. BiColorableFNA will be tested using repeatable test scripts and files. Statement and branch coverage will be determined. Any statements or branches not executed will be explained.   
   Validation: Inspection
2. BiColorableFNA will be tested using a random test generator against another student’s version. Any discrepancies will be explained.  
   Validation: Inspection

*\*\*Reviews/Inspections*

1. A series of peer reviews will be held on all sections and a report make of each review  
   Validation: Observation

*\*\*Correctness Arguments*

1. A correctness arguments will be supplied for both the algorithm and the code.  
   Validation: Reviews

Sample Runs

*Sample Run 1:*

If *stdin* is:

3

3

0 1

1 2

2 0

9

8

0 1

0 2

0 3

0 4

0 5

0 6

0 7

0 8

0

Then *stdout* should be:

NOT BICOLORABLE.

BICOLORABLE.

*Sample Run 2:*

If *stdin* is:

4

4

0 1

1 2

2 3

3 4

0

Then *stdout* should be:

BICOLORABLE.

**Input Space Partitioning**

1. nmbrNodes: 2

Note: Nodes are BiColorable by default

1. nmbrNodes: 3
   1. nmbrEdges < 3
      1. nodes are centric (ex: 0-1, 0-2)
      2. nodes are cyclic (ex: 0-1, 1-2)
   2. nmbrEdges = 3
2. nmbrNodes > 3
   1. nmbrNodes > nmbrEdges

Note: Nodes are BiColorable

* 1. nmbrNodes <= nmbrEdges
     1. Nodes form centric group(s) (ex: 0-1, 0-2, 0-3)
     2. Nodes form cyclic group(s) (ex: 0-1, 1-2, 2-3)
        1. At least one cyclic group is odd
        2. All cyclic groups are even

**Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input Space Partition Number** | **Partition Description** | **Input file or Scenario ref.** | **Output** | **Case or file #** |
| 1  nmbrNodes:2 |  | *2*  *1*  *0 1*  *0* | BICOLORABLE | 01 |
| 2.1.1  nmbrNodes:3  nmbrEdges:2 |  | 3  2  0 1  0 2  0 | BICOLORABLE | 02 |
| 2.1.2  nmbrNodes:3  nmbrEdges:2 |  | 3  2  0 1  1 2  0 | BICOLORABLE | 03 |
| 2.2  nmbrNodes:3  nmbrEdges:3 |  | 3  3  0 1  1 2  2 0  0 | NOT BICOLORABLE | 04 |
| 3.1  nmbrNodes:4  nmbrEdges:3 |  | 4  3  0 1  0 2  0 3  0 4  0 | BICOLORABLE | 05 |
| 3.1  nmbrNodes:199  nmbrEdges:100 |  | 199  100  0 1  1 2  .  .  198 135  0 | BICOLORABLE | 06 |
| 3.1  nmbrNodes:8  nmbrEdges:8 |  | 8  8  0 1  1 2  2 3  3 4  4 5  5 6  6 7  7 0  0 | BICOLORABLE | 07 |
| 3.2.1  nmbrNodes:5  nmbrEdges:5 |  | 5  5  0 1  0 2  0 3  0 4  0 5  0 | BICOLORABLE | 07 |
| 3.2.1  nmbrNodes:100  nmbrEdges:100 |  | 100  100  0 1  0 2  0 3  .  .  20 93  20 94  20 95  0 | BICOLORABLE | 08 |
| 3.2.2.1  nmbrNodes:5  nmbrEdges:6 |  | 5  6  0 1  0 2  0 3  0 4  0 5  5 4  0 | NOT BICOLORABLE | 09 |
| 3.2.2.1  nmbrNodes:9  nmbrEdges:15 |  | 9  15  0 1  0 5  1 5  1 3  1 4  2 0  4 5  4 7  7 8  0 8  2 5  8 2  8 4  7 8  0 4  0 | NOT BICOLORABLE | 10 |
| 3.2.2.1  nmbrNodes:151  nmbrEdges:121 |  | 151  121  0 1  0 2  0 3  2 3  .  .  148 149  148 150  148 151  151 150  0 | NOT BICOLORABLE | 11 |
| 3.2.2.2  nmbrNodes:6  nmbrEdges:7 |  | 6  6  0 1  1 2  2 3  3 4  4 5  5 6  6 0  0 | BICOLORABLE | 12 |
| 3.2.2.2  nmbrNodes:199  nmbrEdges:151 |  | 199  151  0 1  1 2  2 3  3 0  .  .  196 197  197 198  198 199  199 196  0 | BICOLORABLE | 13 |

**Design**

*Solution Principle*

The input data can be held as follow:

int nodeColorArray[] // An array to hold nodes colors  
vector<int> adjacencyList[] // A vector to hold adjacent nodes  
int nmbrNodes // Number of nodes - user input  
int nmbrEdges // Number of edges - user input  
int firstNode // First node of two connected nodes - user input  
int secondNode // Second node of two connected nodes - user input

A Breadth first search will run on nodeColorArray[]. This algorithm will divide the searching procedures into layers. Each layer is associated with a node and an adjacency vector of adjacent nodes. Nodes without colors are assigned colors on nodeColorArray[]and colors are checked from the same array. adjacencyList[]is created as an array of vectors to hold adjacent nodes with vector index representing the node number. Before trying to change the node color, its color is checked with the parent node. If it is found to be the same then:

stdout BICOLORABLE as isBiColorable() returns true

Otherwise, the loop ends without finding any two adjacent nodes with the same color and hence:

stdout NOT BICOLORABLE as isBiColorable() returns false

Algorithm

main()

A01 **Define**:

const int MAX\_NODES = 198;

bool nodesArray[MAX\_NODES];

A03 **While** (true) {

A02 **Read** nmbrNodes from stdin;

A04 **if** (nmbrNodes is equal to 0) {

A05 **Exit;**

}

A06 **Read** nmbrEdges from stdin;

A07 **Populate** nodeColorArray[] by assigning all values to zero  
A08 **Populate** adjacencyList[] by reading from stdin  
A09 **If** (isBiColorable(nodeColorArray[], nmbrNodes) {  
A10 **Write** BICOLORABE. to stdout;  
 }//If  
A11 **Else** {  
 **Write** NOT BICOLORABE. to stdout;  
 }//While processing all graphs in input file  
A13 **Return** to invoker;

int  
oppositeColor(int colorCode) {  
/\*\* ----------------------------------------------------------------------------  
 \* DESCRIPTION - oppositeColor  
 \* Return 1 (blue) if input is 2 (red) and vice versa.  
 \*  
 \* REQUIREMENTS  
 \* R01 On entry colorNode is guaranteed to be either 1 or 2.  
 \* R02 On entry the return value is guaranteed to be either 1 or 2.  
 \*  
 \* SAMPLE INVOCATION  
 \* colorCode = 1  
 \* return 2  
 \* colorCode = 2  
 \* return 1  
 \*  
 \* DESIGN  
 \* Significant variables & Solution Principles  
 \* -------------------------------------------  
 \* int colorCode //input and output parameter  
 \* Return 1 if colorCode is equal to 2 OR  
 \* Return 2 if colorCode is equal to 1  
 \*  
 \* Algorithm  
 \* ---------  
 \* A01 if (colorCode == 1)  
 \* return 2;  
 \* A02 if (colorCode == 2)  
 \* return 1;  
 \* -----------------------------------------------------------------------------\*/

bool  
isBicolorable(int colorArray[], int nmbrNodes) {  
/\*\* ----------------------------------------------------------------------------  
 \* DESCRIPTION - isBicolorable  
 \* Return true if nodes are bicolorable or false if nodes are not bicolorable.  
 \*  
 \* REQUIREMENTS  
 \* R01 On entry colorArray[] is an array of nodes colors  
 \* R02 Returns true if the graph represented by colorArray[] is bi-colorable or  
 \* false otherwise  
 \*  
 \* SAMPLE INVOCATION  
 \* See main() Sample Runs  
 \*  
 \* DESIGN  
 \* Significant variables & Solution Principles  
 \* -------------------------------------------  
 \* int colorArray[], nmbrNodes //input parameter  
 \* Return false if two adjacent nodes have the same color  
 \* Return true otherwise  
 \*  
 \* Algorithm  
 \* ---------  
 \* A01 Set colorArray[0] = ‘0’;  
 \* A02 For (i from 0 though nmbrNodes) {  
 \* A03 For (j from 0 through adjacencyList.size()) {  
 \* A04 if (colorArray[adjacencyList <j>] == ‘0’) {  
 \* A05 Set colorArray[adjacencyList <j>] = oppositeColor(colorArray[i]);  
 \* }//if  
 \* A06 elseif (colorArray[adjacencyList <j>] == colorArray[i]) {  
 \* return false;  
 \* }//elseif  
 \* A07 else {  
 \* continue;  
 \* }//else  
 \* }//for  
 \* }//for  
 \* return true;  
 \* -----------------------------------------------------------------------------\*/

Algorithm Correctness Argument

CR01 By A01 definition of "Define".  
CR02 By A02 and A03 and A04 definition of "Read".  
CR03 By A05 and A06 cause program termination if input is zero.  
CR04 By A07 definition of "Populate"  
CR05 By A08 definition of "Read".  
CR06 By A09 and A10 and A11 definition of "Display".

**Code**

See attached listing.

**Random Test Generation**

RandomGraphsFNA.cpp

bicolorableFNA.sh

**Random Test Results**

|  |  |
| --- | --- |
| **File Name:** | **Result** |
| 1: randomBiColorOCLin1.txt | NOT BICOLORABLE. |
| 2: randomBiColorOCLin10.txt | NOT BICOLORABLE. |
| 3: randomBiColorOCLin11.txt | NOT BICOLORABLE. |
| 4: randomBiColorOCLin12.txt | NOT BICOLORABLE. |
| 5: randomBiColorOCLin13.txt | NOT BICOLORABLE. |
| 6: randomBiColorOCLin14.txt | NOT BICOLORABLE. |
| 7: randomBiColorOCLin15.txt | NOT BICOLORABLE. |
| 8: randomBiColorOCLin2.txt | NOT BICOLORABLE. |
| 9: randomBiColorOCLin3.txt | NOT BICOLORABLE. |
| 10: randomBiColorOCLin4.txt | NOT BICOLORABLE. |
| 11: randomBiColorOCLin5.txt | NOT BICOLORABLE. |
| 12: randomBiColorOCLin6.txt | NOT BICOLORABLE. |
| 13: randomBiColorOCLin7.txt | NOT BICOLORABLE. |
| 14: randomBiColorOCLin8.txt | NOT BICOLORABLE. |
| 15: randomBiColorOCLin9.txt | NOT BICOLORABLE. |

**Test Report**

See homework 18.

**Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| *Version* | *Date* | *Author* | *Comment* |
| 1.0 | 9/4/17 | Andrew Fakhry | Description and requirements |
| 2.0 | 9/14/17 | Andrew Fakhry | Input space partitioning |
| 3.0 | 9/21/17 | Andrew Fakhry | Algorithm |
| 4.0 | 10/17/17 | Andrew Fakhry | Correctness argument |
| 5.0 | 10/26/17 | Andrew Fakhry | Coding |
| 6.0 | 11/7/17 | Andrew Fakhry | Random tests |
| 7.0 | 12/5/17 | Andrew Fakhry | Unit testing and coverage |