This document is in compliance with SMDS standard version *3.4*

|  |  |
| --- | --- |
| **Description:** | ServiceStationRAR is a simple program that calculates the minimum number of service stations, given a list of connecting towns, a company must build to have stations located either in a town or directly adjacent to a town with a station. |

**Development Estimates/Actuals**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Est Sz** | **Act Sz** |  | **Strt Date** | **Est Cmplt** | **Act Cmplt** |  | **Est Effrt** | **Act Effrt** |
| Reqs | 10 | 7 |  | 9/06/16 | 9/06/16 | 9/08/16 |  | 3 | 2 |
| Inpt Partition | 20 | 1 |  | 9/20/16 | 9/20/16 | 9/20/16 |  | 2 | 1 |
| Test Cases | 20 | 12 |  | 9/20/16 | 9/20/16 | 9/20/16 |  | 2 | 1.5 |
| Design |  |  |  | 9/26/16 | 9/26/16 | 9/27/16 |  | 2 | 1 |
| # Classes | *1* | 1 |  |  |  |  |  |  |  |
| # Methods | *1* | *2* |  |  |  |  |  |  |  |
| # Dsng Elms | *3* | *3* |  |  |  |  |  |  |  |
| Algor Correct |  |  |  | 10/4/16 | 10/4/16 | 10/18/16 |  | 2 | 3 |
| Implementatn |  |  |  | 10/17/16 | 10/17/16 | 10/18/16 |  | 5 | 14 |
| NCLC |  |  |  |  |  |  |  |  |  |
| Requir. Trace |  |  |  |  |  |  |  |  |  |
| Code Correct |  |  |  | 10/25/16 | 10/25/16 | 10/26/16 |  | 2 | 4 |
| Final Test |  |  |  | 11/15/16 | 11/15/16 | 11/15/16 |  | 2 | 2.5 |
| Inspec |  |  |  | 11/15/16 | 11/15/16 | 11/15/16 |  | 1 | .5 |
| Wrap up |  |  |  | 11/15/16 | 11/15/16 | 11/15/16 |  | 1 | .5 |
|  |  |  |  |  |  | Total Hrs |  | *22 hrs* | *28.75 hrs* |

Estimates/Actuals Comments

* *I anticipate spending more time than necessary to implement solution due to having a novice level of C++ experience.*
* This is also a comment!

**Requirements**

*\*\*Constraints*

1. ServiceStationRAR is developed using the MTM SMDS template version 4.2  
   Validation: Observation & Inspection
2. ServiceStationRAR is coded in C++ and adheres to the *MTM Standard for C++ Source Files*Validation: Observation & Inspection

*\*\*Preconditions*

1. The directory that contains the ServiceStationRAR executable must also contain an input text file(s), which features the number of towns and their respective connections.  
   Validation: Observation & Test

*\*\*Invocation*

1. ServiceStationRAR will be invoked from the Windows command line: *./ServiceStationRAR*Validation: Observation & Test
2. ServiceStationRAR will be supplied an input text file input.txt from which to read in configurations of town clusters.  
   Validation: Guaranteed

*\*\*Input/Ouput*

1. The input consists of up to nine configurations of clusters of towns.  
   Validation: Inspection & Test & Guaranteed
2. Each configuration begins with two positive integers, N and M, separated by a space, where N is the number of towns and M is the number of connections between said towns.  
   Validation: Inspection & Test & Guaranteed
3. M number of connections follows, each connection being on a single line by itself.  
   Validation: Inspection & Test & Guaranteed
4. For each connection, there will be two positive integers separated by a single whitespace. The first integer will be a town and the second integer is the town it connects to.  
   Validation: Inspection & Test & Guaranteed
5. Input is guaranteed to have an EOF after the last line, indicated by a connection of N = 0, M = 0.  
   Validation: Inspection & Test
6. The only output is the minimum number of stations that need to be built so that, for any town, there is a service station in it or in a directly adjacent town.  
   Validation: Inspection & Test

*\*\*Postconditons*

1. None

*\*\*Testing*

1. ServiceStationRAR 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. ServiceStationRAR will be tested using a random test generator against another student’s version. Any discrepancies will be explained.  
   Validation: Inspection

*\*\*Inspection*

1. An inspection on all completed sections will be performed on a later date.

A final inspection will be done before delivering program.  
Validation: Observation

*\*\*Algorithm Correctness Argument*

1. An algorithm correctness argument is included in this SMDS.  
   Validation: Observation

*\*\*Code Correctness Argument*

1. A code correctness argument is included in this SMDS.  
   Validation: Observation

**Sample Runs**

Input: 6 5 Output: 2  
 1 3  
 3 2  
 3 4  
 2 6  
 5 6  
 0 0

Input: 7 6 Output: 3  
 1 4  
 1 2  
 3 5  
 2 6  
 5 6  
 6 7  
 0 0

Input: 6 5 Output: 1  
 1 3  
 2 3  
 4 3  
 5 3  
 6 3  
 0 0

Input: 6 9 Output: 2  
 1 2  
 1 3  
 1 6  
 2 4  
 2 6  
 3 4  
 3 5  
 3 6  
 4 6  
 0 0

Input: 6 5 Output: 2  
 1 2  
 2 3  
 3 4  
 4 5  
 5 6  
 0 0

Input: 6 6 Output: 3  
 1 2  
 2 5  
 3 2  
 3 4  
 3 5  
 5 6  
 0 0

**Input Space Partitioning**

1. A single test case (configuration) exists.
   1. There is only one cluster.
      1. numTowns = 3 (triad)
         1. numPairs is 0.
         2. numPairs is 1.
         3. numPairs is > 1 & <= 3.
         4. numPairs is > 3.
      2. numTowns > 3 & < 35
         1. numPairs is 0.
         2. numPairs is 1.
         3. numPairs is > 1 & <= numTowns.
         4. numPairs is > numTowns.
      3. numTowns = 35
         1. numPairs is 0.
         2. numPairs is 1.
         3. numPairs is > 1 & <= numTowns.
         4. numPairs is > numTowns.
   2. There is more than one cluster.  
      Several sets of clusters that are variations of 1.1 orders.
2. Two to nine test cases exist.

Several test cases from 1

**Test Cases**

|  |  |  |
| --- | --- | --- |
| **Input Space Partition Summary** | **Input file for each test case** | **Output file for this test case** |
| 1.1.1.1  Number of clusters = 1  numTowns = 3  numPairs = 0 | ***testIn***  ***3 0***  ***0 0*** | ***testOut***  ***3*** |
| 1.1.1.2  Number of clusters = 1  numTowns = 3  numPairs = 1 | ***testIn***  ***3 1***  ***1 2***  ***0 0*** | ***testOut***  ***2*** |
| 1.1.1.3  Number of clusters = 1  numTowns = 3  numPairs = 2 | ***testIn***  ***3 2***  ***1 2***  ***2 3***  ***0 0*** | ***testOut***  ***1*** |
| 1.1.1.4  Number of clusters = 1  numTowns = 3  numPairs = 4 | ***testIn***  ***3 2***  ***1 2***  ***2 3 3 1 1 3***  ***0 0*** | ***testOut***  ***1*** |
| 1.1.2.1  Number of clusters = 1  numTowns = 7  numPairs = 0 | ***testIn***  ***7 0***  ***0 0*** | ***testOut***  ***7*** |
| 1.1.2.2  Number of clusters = 1  numTowns = 7  numPairs = 1 | ***testIn***  ***7 1***  ***1 2***  ***0 0*** | ***testOut***  ***6*** |
| 1.1.2.3  Number of clusters = 1  numTowns = 7  numPairs = 2 | ***testIn***  ***7 2***  ***1 2***  ***3 4***  ***0 0*** | ***testOut***  ***5*** |
| 1.1.2.4  Number of clusters = 1  numTowns = 7  numPairs = 8 | ***testIn***  ***7 2***  ***1 2***  ***2 3***  ***3 4***  ***4 5***  ***5 6 6 7 7 1 0 0*** | ***testOut***  ***2*** |
| 1.1.3.1  Number of clusters = 1  numTowns = 35  numPairs = 0 | ***testIn***  ***35 0***  ***0 0*** | ***testOut***  ***35*** |
| 1.1.3.2  Number of clusters = 1  numTowns = 35  numPairs = 1 | ***testIn***  ***35 1***  ***1 2 0 0*** | ***testOut***  ***34*** |
| 1.1.3.3  Number of clusters = 1  numTowns = 35  numPairs = 2 | ***testIn***  ***35 2***  ***1 2***  ***3 4***  ***0 0*** | ***testOut***  ***33*** |
| 1.1.3.4  Number of clusters = 1  numTowns = 35  numPairs = 36 | ***testIn***  ***35 36***  ***1 2***  ***1 3 3 4***  ***2 4 4 5***  ***2 5 6 9 7 8 7 9 9 10 8 10 11 12 11 14 14 15 12 13 12 15 15 13 16 19 17 19 17 18 19 20 18 20 24 23 22 24 22 23 22 25 25 22 27 29 27 30 26 29 30 28 32 35 35 33 34 33 34 31***  ***0 0*** | ***testOut***  ***14*** |

Design

Significant Data items/Structures

* const vector<int>& townsPR
* int connectsTo
* int nmbrConnects
* int ndxTownsPR
* int nmbrTownsPR
* int nmbrTownsUsedPR
* int thisTown
* int& minCoveredPtrPR
* multimap<int, int, greater<int> > connections
* const vector<unsigned long long>& adjMatrixPtrPR
* const vector<unsigned long long>& coverMatrixPtrPR
* const unsigned long long allCoveredPR
* unsigned long long coveredPR

Solution Analysis

To find the correct number of service stations, the algorithm first eliminates any redundant connections. The first service station is placed at the town with the most number of roads connected to it. The towns can then be partitioned into triads based on whether or not a town is adjacent to a service town. More service stations may be then added into the center of these triads if they meet the necessary conditions.

Note: SS stands for ServiceStation

Algorithm

main(){

A00 **Define** nmbrTownsPR, nmbrConnects;

A01 While( cin << nmbrTownsPR << nmbrConnects){

A02 **If**( !nmbrTownsPR && !nmbrConnects)

A03 Break;

//If

A04 For(ndx from 0 to nmbrConnects){

A05 **Define** thisTown, connectsTo;

A06 **Read** thisTown, connectsTo;

A07 **Decrement** thisTown, connectsTo;

A08 **Set** adjMatrixPtrPR [thisTown] = connectsTo;

A09 **Set** adjMatrixPtrPR [connectsTo] = thisTown;

}//For

A09 **Define** connections**;**

A10 **For**(ndx from 0 to nmbrTownsPR){

A11 **Define** nmbrConnections = 0;

A12 **For**(ndy from 0 to nmbrTownsPR){

A13 **If**(adjMatrixPtrPR[ndx] &

A14 **Convert**<unsigned long long>(1) << ndy)

A15 **Increment** nmbrConnections;

//If

}//For

A16 Set connections(pair<int, int> (nmbrConnections, ndx));

}//For

A17 **Define** townsPR[nmbrTownsPR];

A18 **Define** iterator ndx = townsPR.begin();

A19 **For**(**Define** iterator ndy; **from** connections.begin() **to** connections.end()){

A20 **Set** \*ndx++ = ndy -> second;

}//For

A21 **Define** coverMatrixPtrPR(nmbrTownsPR);

A22 **Define** isCovered = 0;

A23 **For**(**Define** ndxTownsPR = nmbrTownsPR - 1; **from** ndxTownsPR **to** 0 ){

A24 **Define** towns = townsPR[ndxTownsPR];

A25 **Set** isCovered |= adjMatrixPtrPR[towns] |

(static\_cast<unsigned long long>(1) << towns);

A26 **Set** coverMatrixPtrPR[ndxTownsPR] = isCovered;

}//For

A27 **Define** allCoveredPR =

(static\_cast<unsigned long long>(1) << nmbrTownsPR) - 1;

A28 **Define** coveredPR = 0;

A29 **Define** minCoveredPtrPR = nmbrTownsPR;

A30 **Call** dominatingSet(townsPR, nmbrTownsPR, 0, 0, adjMatrixPtrPR, coverMatrixPtrPR, allCoveredPR, coveredPR, minCoveredPtrPR);

}//while

}//main()

A31 dominatingSet(){

A32 **If**(minCoveredPtrPR <= nmbrTownsUsedPR + 1 || ndxTownsPR == nmbrTowns)

A33 return;

//If

A34 **If**(coveredPR | coverMatrixPtrPR[ndxTownsPR]) != allCoveredPR)

A35 return;

//If

A36 **Define & Set** crrntTownNdx = townsPR[ndxTownsPR];

A37 **Define & Set** crrntTown = static\_cast<unsigned long long>(1) << crrntTownNdx;

A38 **Define & Set** whichIsCoveredPR = coveredPR | adjMatrixPtrPR[crrntTownNdx] | crrntTown;

A39 **If**(whichIsCoveredPR == allCoveredPR){

A40 **Set** minCoveredPtrPR = nmbrTownsUsedPR + 1;

A41 return;

//If

A42 **Call** dominatingSet(townsPR, nmbrTownsPR, ndxTownsPR + 1, nmbrTownsUsedPR + 1, adjMatrixPtrPR, coverMatrixPtrPR, allCoveredPR, whichIsCoveredPR, minCoveredPtrPR);

A43 **Call** dominatingSet(townsPR, nmbrTownsPR, ndxTownsPR + 1, nmbrTownsUsedPR + 1, adjMatrixPtrPR, coverMatrixPtrPR, allCoveredPR, coveredPR, minCoveredPtrPR);

}//dominatingSet

Algorithm Correctness Argument  
Note: R01 – R06 are not applicable in correctness argument.

* CR07 By While loop which will only continue as long as there exists an N and M to read in; in A01 and A02
* CR08 By The For Loop which populates adjMatricPtrPR using the number of connections ; in A04
* CR09 By A05 defines thisTown and connectsTo, A06 then reads in thisTown and connectsTo.
* CR10 By If statement If(!nmbrTowns && !nmbrConnects) checks to make sure both variables are not zero. If they are zero, a break statement is issued and the while loop ends.
* CR11 By A32 – A43

dominatingSet works by checking to see first if the minimal coverage needed to service a set of towns matches the current number of towns designated as service stations. If this case is met, then the minimal number of service stations has been found.  
  
Otherwise, dominatingSet does a bitwise or comparison between coveredPR and coverMatrixPtrPR[] at ndxTownsPR and sees if they are not equal to allCoveredPR. If this case is met, the minimal number of service stations has been found.  
  
Otherwise, dominatingSet checks if whichIsCoveredPR equals allCoveredPR. If this case is met, minCoveredPtrPR is set to nmbrTownsUsedPR + 1;   
  
If neither of the first two if statements is triggered, dominatingSet will recursively call on itself twice to reduce the problem by feeding covered and allCovered into the parameters.

**Code**

Refer to the ServiceStationsRAR.cpp file located in this directory to examine code.

Complies with MTM Standard for C++ Source Code (version 3.2)

**Requirements Trace**

*Good software engineering practice documents the code elements that implement requirements that are validated by test or inspection. This can be done using a  
Inspectable Requirements → Code Elements Trace* Table*.*

*For an Inspectable Requirement → Code Elements* Trace *use one of the following table:*

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Function(s)** | **Code Element** |
| R07 | main() | L02 |
| R08 | main() | L03 |
| R09 | main() | L07 – L09 |
| R10 | main() | L04 |
| R11 | dominatingSet() | L01 – L14 |

**Code Correctness Arguments**

Code Correctness Argument for function main():

CA01 by (L03)

CA02 by (L04)

CA03 by (L05)

CA04 by (L07)

CA05 by (L08)

CA06 by (L09)

CA07 by (L10, L11)

CA08 by (L12)

CA09 by (L13)

CA10 by (L15)

CA11 by (L16)

CA12 by (L17)

CA13 by (L18)

CA14 by (L19)

CA15 by (L20)

CA16 by (L21)

CA17 by (L22)

CA18 by (L24)

CA19 by (L25)

CA20 by (L26)

CA21 by (L27)

CA22 by (L28)

CA23 by (L29)

CA24 by (L30)

CA25 by (L31)

CA26 by (L32)

CA27 by (L33)

CA28 by (L34)

CA29 by (L35)

CA30 by (L36)

Code Correctness Argument for function dominatingSet():

CA32 by (L01)

CA33 by (L02)

CA34 by (L04)

CA35 by (L05)

CA36 by (L06)

CA37 by (L07)

CA38 by (L08)

CA39 by (L09)

CA40 by (L011)

CA41 by (L12)

CA42 by (L13)

CA43 by (L14)

**Test Directories, Files, and Scripts/Scenarios**

The test script is called: *ssRandomTestRAR.sh*

The input/output files are formatted as: *ssIn01.txt* & *ssOut01.txt*

The test result files is called: myRandomTestResults.txt

The test can be run by accessing the test directory on katie: /students/areid/esof322/hw9

Then using the command: *bash ssRandomTestRAR.sh*

**Random Test Generation**

Random tests were generated using the SSrandomTest.cpp file included in deliverables.

**Performance Test Procedure**

No performance tests were performed.

**Test Report**

+++ServiceStationRAR Test Run+++

Testing the file ServiceStationRAR.cpp

Input File: ssIn01.txt Output File: ssOut01.txt

Input File: ssIn02.txt Output File: ssOut02.txt

Input File: ssIn03.txt Output File: ssOut03.txt

Input File: ssIn04.txt Output File: ssOut04.txt

Input File: ssIn05.txt Output File: ssOut05.txt

Input File: ssIn06.txt Output File: ssOut06.txt

Input File: ssIn07.txt Output File: ssOut07.txt

Input File: ssIn08.txt Output File: ssOut08.txt

Input File: ssIn09.txt Output File: ssOut09.txt

All test cases passed.

*---ServiceStationRAR Test Run---*

**Inspection Report(s)**

*Any Inspection Notice(s), Inspection Defect List(s), and Inspection Report(s) should be included in the deliverables.*

**Deliverables**

* ServiceStationRAR.cpp
* SSrandomTest.cpp
* ssRandomTest.sh
* myRandomTestResults.txt
* otherRandomTestResults.txt
* 4 reasonably complex random test files
* Finalized SMDS (this thing!)

**Version History**

Most real world programs will undergo some modification during their lifetime. The initial program version and any subsequent modifications should be documented here.

|  |  |  |  |
| --- | --- | --- | --- |
| *Version* | *Date* | *Author* | *Comment* |
| 1.0 | 8/30/16 | Alex Reid | HW1 – Requirements created |
| 2.0 | 9/08/16 | Alex Reid | HW2 – Input Space Partition |
| 3.0 | 9/20/16 | Alex Reid | HW3 – Sample Runs |
| 4.0 | 9/27/16 | Alex Reid | HW4 – Design |
| 5.0 | 10/04/16 | Alex Reid | HW5 – Algo Correctness Argument |
| 6.0 | 10/11/16 | Alex Reid | HW6 – Implementation |
| 7.0 | 10/18/16 | Alex Reid | HW7 – Requirements Trace |
| 8.0 | 10/25/16 | Alex Reid | HW8 – Code Correctness Argument |
| 9.0 | 11/1/16 | Alex Reid | HW9 - Testing |
| 10.0 | 11/15/16 | Alex Reid & Max Austin | HW10 – Random testing, compared with Max Austin’s .cpp file: ServiceStationAMD.cpp |