This document is in compliance with SMDS standard version 4.2

|  |  |
| --- | --- |
| **Description:** | Determines number of service stations necessary to conveniently service customers in a number of towns.  See “Programming Challenges”P by Skiena & Revilla, problem 8.6.4 |

**Development Estimates/Actuals**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Est Sz** | **Act Sz** |  | **Strt Date** | **Est Cmplt** | **Act Cmplt** |  | **Est Effrt** | **Act Effrt** |
| Reqs | 10 | 14 |  | 9/7 | 9/8 | 9/8 |  | 1.5 hrs | 1.42 hrs |
| Inpt Partition | 12 | 27 |  | 9/19 | 9/20 | 9/20 |  | 2.5 hrs | 2.08 hrs |
| Test Cases | 24 | 13 |  | 9/20 | 9/20 | 9/20 |  | 1 hr | 0.66 hrs |
| Design | 30 |  |  | 9/26 | 9/27 | 9/27 |  | 2.5 hrs | 8.17 hrs |
| # Classes |  |  |  |  |  |  |  |  |  |
| # Methods |  |  |  |  |  |  |  |  |  |
| # Dsng Elms |  |  |  |  |  |  |  |  |  |
| Algor Correct |  |  |  | 10/3 | 10/4 | 10/4 |  | 2 hrs | 1.43 hrs |
| Implementatn |  |  |  | 10/17 | 10/18 |  |  | 2 hrs | 21.5 hrs |
| NCLC |  |  |  |  |  |  |  |  |  |
| Requir. Trace |  |  |  |  |  |  |  |  |  |
| Code Correct |  |  |  | 10/18 | 10/18 |  |  | 2 hrs | 1 hr |
| Final Test |  |  |  |  |  |  |  | 2 hrs | 2 hrs |
| Inspec |  |  |  |  |  |  |  | 2 hrs | 5 hrs |
| Wrap up |  |  |  |  |  |  |  | 2 hrs | 2 hrs |
|  |  |  |  |  |  | Total Hrs |  | 19.5 hrs | 45.5 hrs |

Estimates/Actuals Comments

* Estimated an hour and a half to go through problem requirements. With help of formatting from examples it should not take too long.
* Test cases were smaller than expected since for (2) and (3) the partitions are repetitions of (1)
* Verifying the algorithm covered cases and that I had grabbed the correct variables for structure took slightly longer than expected.
* Estimate for design is off since I needed to rework algorithm. Algorithm trace found issues in class exercise.
* I ran out of time to fix the code and update the algorithm to the newest version. This will require more time over the estimate, and will need to be turned in after the initial deadline of 10/18/16.
* Switching to backtracking proved to be extremely time consuming. The problem space allows for some edge cases that can be harder to get rid of while still trying to have an exhaustive search. I was not able to rewrite the SMDS documentation on the newer version of the code, but it has been included as a reference of the hours of work I did.
* After many hours of working on the algorithm and implementation I was able to get it accepted by UVA.

**Requirements**

The priority of these requirements is imperative.

*\*\*Constraints*

1. ServiceStationBTL is coded in C++.  
   Validation: Inspection and Test (Build)
2. The source file for ServiceStationBTL is in compliance with *MTM C++ Source File Standard v4.0*.  
   Validation: Inspection
3. The SMDS conforms to version 4.2 of the standard.

*\*\*Preconditions*

None

*\*\*Invocation*

1. Launch application execution file.  
   Validation: Test

*\*\*Input/Ouput*

1. Input begins with a positive integer, nmbrTowns, followed by a second positive integer, nmbrPairs. nmbrTowns should specify the number of towns being considered, while nmbrPairs specifies the number of connections between pairs of towns.Validation: Test
2. nmbrPairs lines of town pairs follows the first line.  
   Validation: Test
3. Each line of town pairs lists two towns by number, townNum, separated by a space.  
   Validation: Inspection
4. The final line of input should be “0 0”.  
   Validation: Inspection
5. The input restriction for the first line is 3<= nmbrTowns <= 35.  
   Validation: Test
6. For the following lines all town numbers for the pairs should be 1<= townNum <= nmbrTowns.  
   Validation: Test and Inspection
7. Output to *stdout* should consist of a single positive integer indicating the least number of   
   stations to have a station either in or directly connected to each town.  
   Validation: Test and Inspection

*\*\*Postconditons*

None

*\*\*Testing*

1. ServiceStationBTL will be tested using test scripts and files.  
   Validation: Inspection
2. ServiceStationBTL will be randomly tested against another student’s version.Validation: Inspection

*\*\*Inspection*

1. When complete an inspection will be held on all sections, including the SMDS.

Validation: Observation

*\*\*Algorithm Correctness Argument*

1. A correctness algorithm will be provided for both the algorithm and the code.Validation: Inspection

Sample Runs

*Sample Run 1:*

If *stdin* is:

8 12

1 2

1 6

1 8

2 3

2 6

3 4

3 5

4 5

4 7

5 6

6 7

6 8

0 0

Then *stdout* should be:

2

*Sample Run 2:*

If *stdin* is:

6 6

1 2

1 6

2 3

3 4

4 5

5 6

0 0

Then *stdout* should be:

3

*Sample Run 3:*

If *stdin* is:

6 10

1 2

1 4

1 5

1 6

2 3

2 4

2 5

3 4

4 5

5 6

0 0

Then *stdout* should be:

2

*Sample Run 4:*

If *stdin* is:

7 12

1 2

1 6

1 7

2 3

2 7

3 4

3 7

4 5

4 7

5 6

5 7

6 7

0 0

Then *stdout* should be:

1

*Sample Run 5:*

If *stdin* is:

8 8

1 2

1 8

2 3

3 4

4 5

5 6

6 7

7 8

0 0

Then *stdout* should be:

3

*Sample Run 6:*

If *stdin* is:

8 9

1 3

1 6

2 4

2 7

3 5

4 6

5 6

6 8

7 8

0 0

Then *stdout* should be:

3

*Sample Run 7:*

If *stdin* is:

8 8

1 8

2 7

2 8

3 4

3 5

3 7

4 6

5 6

0 0

Then *stdout* should be:

3

**Input Space Partitioning**

1. One set of town combinations is in the input
   1. nmbrTowns is 3
      1. nmbrPairs is 0
      2. nmbrPairs is 1
      3. nmbrPairs is greater than 1
   2. nmbrTowns is greater than 3 but less than 35
      1. nmbrPairs is 0
      2. nmbrPairs is 1
      3. nmbrPairs is greater than 1
   3. nmbrTowns is 35
      1. nmbrPairs is 0
      2. nmbrPairs is 1
      3. nmbrPairs is greater than 1
2. Two sets of town combinations are in the input

*Several pairs of town combinations that are variations on (1)*

1. More than two town combinations but no more than 10 combinations exist in the input

*Several sets of town combinations that are variations on (1)*

**Test Cases**

|  |  |  |
| --- | --- | --- |
| **Input Space Partition Summary** | **Input file for each test case** | **Output file for this test case** |
| 1.1.1  One town combination  numbrTowns is 3  nmbrPairs is 0 | *ssT01in*:  3 0  0 0 | *ssT01out*:  3 |
| 1.1.2  One town combination  numbrTowns is 3  nmbrPairs is 1 | *ssT02in*:  3 1  1 2  0 0 | *ssT02out*:  2 |
| 1.1.3  One town combination  numbrTowns is 3  nmbrPairs > 1 | *ssT03in*:  3 2  1 2  1 3  0 0 | *ssT03out*:  1 |
| 1.2.1  One town combination  3 < numbrTowns < 35  nmbrPairs is 0 | *ssT04in*:  4 0  0 0 | *ssT04out*:  4 |
| 1.2.2  One town combination  3 < numbrTowns < 35  nmbrPairs is 1 | *ssT05in*:  4 1  1 4  0 0 | *ssT05out*:  3 |
| 1.2.3  One town combination  3 < numbrTowns < 35  nmbrPairs > 1 | *ssT06in*:  4 3  1 2  2 3  3 4  0 0 | *ssT06out*:  2 |
| 1.3.1  One town combination  numbrTowns is 35  nmbrPairs is 0 | *ssT07in*:  35 0  0 0 | *ssT07out*:  35 |
| 1.3.2  One town combination  numbrTowns is 35  *nmbrPairs* is 1 | *ssT08in*:  35 1  1 35  0 0 | *ssT08out*:  34 |
| 1.3.3  One town combination  numbrTowns is 35  nmbrPairs > 1 | *ssT09in*:  35 11  1 2  1 4  1 7  2 3  3 5  3 6  4 5  4 7  5 7  6 8  7 8  0 0 | *ssT09out*:  29 |
| 2  Two town combinations | *ssT10in*:  3 2  1 2  1 3  35 11  1 2  1 4  1 7  2 3  3 5  3 6  4 5  4 7  5 7  6 8  7 8  0 0 | *ssT10out*:  1  29 |
| 2  Two town combinations | *ssT11in*:  4 0  4 3  1 2  2 3  3 4  0 0 | *ssT11out*:  4  2 |
| 3  Between 3 and 10 town combinations | *ssT12in*:  35 1  1 35  3 2  1 2  1 3  35 11  1 2  1 4  1 7  2 3  3 5  3 6  4 5  4 7  5 7  6 8  7 8  0 0 | *ssT12out*:  34  1  29 |
| 3  Between 3 and 10 town combinations | *ssT13in*:  3 2  1 2  1 3  4 0  4 1  1 4  0 0 | *ssT13out*:  1  4  3 |

**Design**

Significant Data Items/Structures

--------------------------------------------

SS is used as a shorthand for ServiceStation

Ndx is used as a shorthand for index

int nmbrTowns

int nmbrPairs

int townNum

int[][] townMatrix

TownCluster //Class to hold townArray as object.

int[] townArray //used to signify which towns are in the

//cluster. These towns are denoted by a 1 in

//the array

int nmbrTownsInCluster

townCluster[] clusters //Keep track of all clusters found.

int nmbrClusters

int ssCnt

int MAX\_TOWNS = 35

int townA, townB

char[] townList

//Constants to be used for clarity. These give names to placeholder values used in

//townList

char ssTown = ‘S’ //Town has service station

char ssAdjTown = ‘A’ //Town next to service station

char townInCluster = ‘C’ //Town has been placed in a cluster

char notProcessedTown = ‘N’ //Town is not yet in cluster, but needs one

char empty = ‘E’ //Town does not exist. Array could be longer than

// towns given

Solution Analysis

-----------------------

Each town configuration must be looked at independently. In each town configuration, we want to create groupings of towns that are connected. Each separate group in the town configuration can be solved separately and added up. We use a mixture of recursive backtracking and a greedy algorithm looking for the maximum number of connections that have not been serviced.

Algorithm

-------------

A00 **Define:**

const int MAX\_TOWNS = 35;

const char ssTown = 'S';

const char ssAdjTown = 'A';

const char townInCluster = 'C';

const char notProcessedTown = 'N';

const char empty = 'E';

const int MAX\_SEARCH = 3;

Town

int neighbors[MAX\_TOWNS + 1];

int nmbrNeighbors = 0;

char state = empty;

int townNum;

TownCluster

int townArray[MAX\_TOWNS];

int nmbrTownsInCluster;

int nmbrTowns, nmbrPairs, ssCnt, townA;

int townB, nmbrTownsRemaining, nmbrClusters;

Town townList[MAX\_TOWNS + 1];

String returnString;

A01 **Read** nmbrTowns and nmbrPairs from stdin;

A02 **While**(nmbrTowns is not zero or nmbrPairs is not zero){

A03 **Set** nmbrTownsRemaining to nmbrTowns;

A04 **Set** ssCnt to 0;

A05 **Set** nmbrClusters to 0;

A06 **For**(townNdx iterates from 0 to MAX\_TOWNS){

A07 **Set** townList[townNdx].townNum to townNdx;

A08 **If**(townNdx is greater than 0 an townNdx less than or equal to

nmbrTowns){

A09 **Set** townList[townNdx].state to notProcessedTown;

}//**If**

A10 **Else**{

A11 **Set** townList[townNdx].state to empty;

}//**Else**

A12 **For**(neighborNdx iterates from 0 to townList[townNdx].nmbrNeighbors){

A13 **Set** townList[townNdx].neighbors[neighborNdx] to 0;

}//**For**

A14 **Set** townList[townNdx].nmbrNeighbors to 0;

}//**For**

A15 **For** (pairNdx: iterates from one to nmbrPairs){

A16 **Read** townA and townB from stdin;

A17 **If**(townA is not equal to townB and findNeighbor(townA,

townB) is false){

A18 **Set** townList[townA].neighbors[townList[townA]

.nmbrNeighbors] to townB;;

A19 **Increment** townList[townA].nmbrNeighbors by one;

A20 **Set** townList[townB].neighbors[townList[townB]

.nmbrNeighbors] to townA;;

A21 **Increment** townList[townA].nmbrNeighbors by one;

}// **If**

}// **For**

//Set stations on all disconnected towns.

A22 **For**(townNdx iterates from 1 to nmbrTowns){

A23 **If**(townList[townNdx].nmbrNeighbors == 0){

A24 **Set** townList[townNdx].state to ssTown;

A25 **Increment** ssCnt by one;

A26 **Decrement** nmbrTownsRemaining by one;

}//**If**

}//**For**

A27 **Construct** clusters[MAX\_TOWNS + 1];

//Creates clusters

A28 **For** (townNdx iterates from 1 to nmbrTowns) {

A29 **If** (townList[townNdx].state == notProcessedTown) {

A30 **Construct** townCluster;

A31 **Set** townCluster.nmbrTownsInCluster to 0;

A32 **Set** townCluster.townArray

[townCluster.nmbrTownsInCluster]

to townList[townNdx].townNum;

A33 **Set** townList[townNdx].state to townInCluster;

A34 **Increment** townCluster.nmbrTownsInCluster by one;

A35 **For** (searchNdx iterates from 0 to

townList[townNdx].nmbrNeighbors) {

A36 **If**(townList[townList[townNdx]

.neighbors[searchNdx]].state

== notProcessedTown) {

A37 **Set** townCluster.townArray

[townCluster.nmbrTownsInCluster]

to townList[townList[townNdx].

neighbors[searchNdx]].townNum;

A38 **Set** townList[townList[townNdx]

.neighbors[searchNdx]].

state to townInCluster;

A39 **Increment** townCluster.nmbrTownsInCluster

by one;

A40 **Set** townCluster to

addLinkedTownsToCluster(townList,

townList[townList[townNdx]

.neighbors[searchNdx]].townNum,

townCluster);

}//**If**

}//**For**

A41 **For**(townNdx iterates from 0 to

townCluster.nmbrTownsInCluster){

A42 **Set** townList[townCluster

.townArray[townNdx]]

.state to townInCluster;

}//**For**

A43 **Set** clusters[nmbrClusters] to townCluster;

A44 **Increment** nmbrClusters by one;

}//**If**

}//**For**

A45 **For**(clusterNdx iterates from 0 to nmbrClusters){

A46 **Set** threshold to greedyAlgorithm(townList,

clusters[clusterNdx], nmbrTowns, 0,

clusters[clusterNdx].nmbrTownsInCluster,

clusters[clusterNdx].nmbrTownsInCluster);

A47 **Increment** ssCnt by recursiveSolver(townList,

clusters[clusterNdx], nmbrTowns, 0,

clusters[clusterNdx].nmbrTownsInCluster,

threshold, 0);

}//**For**

A48 **Display** ssCnt;

A49 **Read** nmbrTowns and nmbrPairs from stdin

}//**While** nmbrTowns is not zero

A50 **Return**;

int recursiveSolver(Town townList[], TownCluster cluster, int nmbrTowns,

int ssCnt, int nmbrTownsRemaining, int threshold, int lowestTown){

A51 **If**(nmbrTownsRemaining is equal to 0 or ssCnt is equal to threshold){

A52 **If**(ssCnt is less than threshold){

A53 **return** ssCnt;

}//**If**

A54 **Else**{

A55 **return** threshold;

}//**Else**

}//**If**

A56 **Else If**(ssCnt is greater than threshold or threshold is equal to 1){

A57 **return** threshold;

}//**Else If**

A58 **Else**{

A59 **Set** tempCnt to ssCnt;

A60 **For**(townIdx iterates from lowestTown to cluster.nmbrTownsInCluster){

A61 **Set** tempCnt to ssCnt;

A62 **Set** tempRemaining to nmbrTownsRemaining;

A63 **If**(townList[cluster.townArray[townIdx]].state != ssTown){

A64 **Increment** tempCnt by one;

A65 **Construct** tempTowns[MAX\_TOWNS + 1];

A66 **For**(copyNdx iterates from 0 to nmbrTowns){

A67 **Set** tempTowns[copyNdx] to townList[copyNdx];

}//**For**

A68 **For** (checkNdx iterates from 0 to

tempTowns[cluster.townArray[townIdx]]

.nmbrNeighbors) {

A69 **Set** townChk to

tempTowns[cluster.townArray[townIdx]];

A70 **If** (tempTowns[townChk.neighbors[checkNdx]].state

Is equal to townInCluster) {

A71 **Decrement** tempRemaining by one;

A72 **Set**

tempTowns[townChk.neighbors[checkNdx]].state to ssAdjTown;

}//**If**

}//**For** each town directly connected to townNum

A73 **If** (tempTowns[cluster.townArray[townIdx]].state

is not equal to ssAdjTown) {

A74 **Decrement** tempRemaining by one;

}//**If**

A75 **Set** tempTowns[cluster.townArray[townIdx]].state to

ssTown;

A76 **Set** tempCnt to greedyAlgorithm(tempTowns, cluster,

nmbrTowns, tempCnt, tempRemaining, threshold);

A77 **If**(tempCnt is less than threshold){

A78 **Set** threshold to tempCnt;

}//**If**

A79 **If**(lowestTown is less than MAX\_SEARCH){

A80 **Set** tempCnt to recursiveSolver(tempTowns,

cluster, nmbrTowns, ssCnt+1, tempRemaining,

threshold, townIdx +1);

A81 **If**(tempCnt is less than threshold){

A82 **Set** threshold to tempCnt;

}//**If**

}//**If**

}//**If**

}//**For**

}//**Else**

A83 **return** threshold;

}//recursiveSolver

int greedyAlgorithm(Town townList[], TownCluster cluster, int nmbrTowns,

int ssCnt, int nmbrTownsRemaining, int threshold){

A84 **If**(nmbrTownsRemaining is equal to 0 or ssCnt is equal to threshold){

A85 **return** ssCnt;

}//**If**

A86 **Else**{

A87 **Set** maxConnections, townNum, and maxDeadEnds to 0;

A88 **Construct** tempTowns[MAX\_TOWNS + 1];

A89 **For**(copyNdx iterates from 0 to nmbrTowns){

A90 tempTowns[copyNdx] = townList[copyNdx];

}//**For**

A91 **For** (townNdx iterates from 0 to cluster.nmbrTownsInCluster) {

A92 **Set** connectionNmbr, nmbrDeadEnds to 0;

A93 **If** (tempTowns[cluster.townArray[townNdx]].state is not equal to

ssTown) {

A94 **Set** townChk to tempTowns[cluster.townArray[townNdx]];

A95 **For** (connNdx iterates from 0 to townChk.nmbrNeighbors){

A96 **If** (tempTowns[townChk.neighbors[connNdx]].state

is equal to townInCluster) {

A97 **Increment** connectionNmbr by 1;

A98 **Set** isDeadEnd = true;

A99 **For**(deadCheckNdx iterates from 0 to

tempTowns[townChk.neighbors[connNdx]]

.nmbrNeighbors){

A100 **If**(tempTowns[townChk.neighbors

[connNdx]].townNum is not equal to

townChk.townNum){

A101 **Set** isDeadEnd to false;

A102 **break**;

}//**If**

}//**For**

A103 **If**(isDeadEnd is true){

A104 **Increment** nmbrDeadEnds by one;

A105 **Increment** connectionNmbr by one;

}//**If**

}//**If**

}//**For**

A106 **If** (townChk.state is not equal to ssAdjTown) {

A107 **Increment** connectionNmbr by one;

}//**If**

A108 **If** (connectionNmbr is greater than maxConnections

or (townNum is equal to 0 && townChk.state is not equal to ssAdjTown) or (connectionNmbr is equal to maxConnections

and (townChk.state is equal to ssAdjTown

or nmbrDeadEnds is greater than maxDeadEnds))) {

A109 **Set** townNum to cluster.townArray[townNdx];

A110 **Set** maxConnections to connectionNmbr;

A111 **Set** maxDeadEnds to nmbrDeadEnds;

}//**If**

}//**If**

}//**For**

A112 **For** (checkNdx iterates from 0 to tempTowns[townNum].nmbrNeighbors;

) {

A113 **If** (tempTowns[tempTowns[townNum].neighbors[checkNdx]].state

is equal to townInCluster) {

A114 **Decrement** nmbrTownsRemaining by one;

A115 **Set** tempTowns[tempTowns[townNum].neighbors[checkNdx]]

.state to ssAdjTown;

}//**If**

}//**For** each town directly connected to townNum

A116 **If** (tempTowns[townNum].state is not equal ssAdjTown) {

A117 **Decrement** nmbrTownsRemaining by one;

}//**If**

A118 **Increment** ssCnt by one;

A119 **Set** tempTowns[townNum].state to ssTown;

A120 **return** greedyAlgorithm(tempTowns, cluster, nmbrTowns,

ssCnt, nmbrTownsRemaining, threshold);

}//**Else**

}//greedyAlgorithm

bool findNeighbor(Town townList[], int townA, int townB) {

bool found = false;

A121 **For** (neighborNum iterates from 0 to townList[townA].nmbrNeighbors) {

A122 **If** (townList[townA].neighbors[neighborNum] is equal to townB) {

A123 **Set** found to true;

A124 **break**;

}//**If**

}//**For**

A125 **return** found;

}//findNeighbor

static TownCluster addLinkedTownsToCluster(Town townList[],

int townNdx, TownCluster cluster) {

A126 **For** (linkNdx iterates from 0 to townList[townNdx].nmbrNeighbors) {

A127 **If** (townList[townList[townNdx].neighbors[linkNdx]].state

is equal to notProcessedTown) {

A128 **Set** cluster.townArray[cluster.nmbrTownsInCluster]

to townList[townList[townNdx]

.neighbors[linkNdx]].townNum;

A129 **Set** townList[townList[townNdx].neighbors[linkNdx]].state to

townInCluster;

A130 **Increment** cluster.nmbrTownsInCluster by one;

A131 **Set** cluster to addLinkedTownsToCluster(townList,

townList[townList[townNdx].neighbors[linkNdx]].townNum,

cluster);

}//**If**

}//**For**

A132 **return** cluster;

}//addLinkedTownsToCluster

//end ServiceStationv2BTL

Algorithm Correctness Argument

----------------------------------------------

1. This does not apply to algorithm correctness.
2. This does not apply to algorithm correctness.
3. This does not apply to algorithm correctness.
4. This does not apply to algorithm correctness.
5. nmbrTowns and nmbrPairs read in on A01 and checked on A02
6. Read in by A15
7. By reading each number separately into townA and townB we use the space

seperator

1. A02 stops at a line of “0 0” (without quotes)
2. townMatrix defined at A00 can hold up to 35 towns
3. A17 tries to match values read in for each town into the town matrix
4. A48 writes the output value.

**Requirements Trace**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Function(s)** | **Code Element** |
| R05 | main | L01 and L02 |
| R06 | main | L15 |
| R07 | main | L15 |
| R08 | main | L02 |
| R09 | main | L00 |
| R10 | main | L17 |
| R11 | main | L48 |

**Code Correctness Arguments**

Code Correctness Argument for function main:

1. Reads in both the nmbrTowns and nmbrPairs from input.
2. Ensures that nmbrTowns and nmbrPairs are not both 0 which is end of input.
3. Sets a tracking variable to nmbrTowns.
4. Sets ssCnt to 0.
5. Sets nmbrClusters to 0.
6. Variable townNdx does iterate from zero to MAX\_TOWNS.
7. The townNum of the town with index townNdx is set to the townNdx.
8. Checks that townNdx is between 1 and nmbrTowns inclusively.
9. The state of the town with index townNdx is set to notProcessedTown.
10. Else catches all towns that do not fit criteria of CA08.
11. State of towns that fall into CA10 are set to empty.
12. neighborNdx does iterate from 0 to the number of neighbors that town has.
13. Each neighbor entry for the town is set to 0.
14. The nmbrNeighbors attribute of the town is set to 0.
15. pairNdx iterates from one to nmbrPairs
16. The pair of towns is read in from stdin and set into variables townA and townB
17. townA is found to be not equal to townB (self-loop) and the towns are not already neighbors. (see   
     CA121 through CA125).
18. This adds townB as a neighbor of townA.
19. The nmbrNeighbors attribute of townA is incremented by one.
20. townA is added as a neighbor of townB.
21. the nmbrNeighbors attribute of townB is incremented by one.
22. townNdx iterates from 1 to nmbrTowns in a for loop.
23. the nmbrNeighbors attribute of town with index townNdx is checked to see if it is 0.
24. The town with index townNdx has its state set to ssTown.
25. ssCnt is incremented by one.
26. nmbrTownsRemaining is decremented by one.
27. The array clusters of type TownCluster is created with size MAX\_TOWNS+1.
28. townNdx iterates from 1 to nmbrTowns.
29. Town with index townNdx is checked to have a state of notProcessedTown.
30. New variable townCluster is created with type TownCluster.
31. The attribute nmbrTownsInCluster of townCluster is set to 0.
32. The first town of townCluster is set to be the town with townNdx.
33. The town with index townNdx has its state set to townInCluster.
34. the nmbrTownsInCluster attribute is incremented.
35. searchNdx iterates from 0 to nmbrNeighbors.
36. The state of the town that matches townList[townList[townNdx].neighbors[searchNdx]] is checked   
     to have the state notProcessedTown.
37. The town we are currently looking at is added to the cluster.
38. The town we are currently looking at has its state updated to be townInCluster.
39. The nmbrTownsInCluster attribute is incremented by one.
40. The town cluster uses the addLinkedTownsToCluster to find more towns that are linked   
     recursively. (See CA126 – CA133 for proof of the cluster creation)
41. townNdx iterates from 0 to nmbrTownsInCluster .
42. Each town in the cluster is set to be townInCluster for the corresponding iterations.
43. The townCluster is added to the array clusters.
44. nmbrClusters is incremented by one.
45. clusterNdx iterates from 0 to nmbrClusters.
46. The value of threshold is set to be the returned value of the greedyAlgorithm call. (CA84-CA120)
47. The value of ssCnt is incremented by the returned value of the recursiveSolver call. (CA51-CA83)
48. The value of ssCnt is displayed by adding it to a string and then displaying all values required for   
     the problem.
49. New values for nmbrTowns and nmbrPairs are read in from stdin.
50. The value 0 is returned as the function exits.

Code Correctness Argument for function recursiveSolver:

1. Statement checks that nmbrTownsRemaining is 0 or that ssCnt is equal to the   
    threshold.
2. Statement checks that ssCnt is less than the threshold.
3. ssCnt is returned.
4. The else catches all conditions outsid of CA52.
5. The value threshold is returned.
6. Statement checks if ssCnt is greater than threshold or if threshold is equal to 1.
7. threshold is returned.
8. Values outside of CA51 and CA 56 are caught.
9. tempCnt is set to the value of ssCnt.
10. townIdx iterates from lowestTown to the nmbrTownsInCluster
11. tempCnt is reset back to ssCnt.
12. tempRemaining is set to nmbrTownsRemaining.
13. Statement checks if town with index townIdx in the cluster does not already have a service   
     station.
14. Increment the tempCnt value by one.
15. New array is constructed (creating the new array allows us to not mess with valid data as we   
     test.)
16. copyNdx iterates from 0 to nmbrTowns.
17. Town with index copyNdx is copied to the temporary array.
18. checkNdx iterates from 0 to the number of neighbors the town has.
19. townChk shorthand variable is set to the town we are looking at.
20. Check if the neighbor of our current town has not been serviced yet.
21. Decrements tempRemaining by one.
22. The town’s state is set to be ssAdjTown.
23. Check if the town we have placed a service station at already had been adjacent.
24. If it has not then decrement tempRemaining by one.
25. Current town has state set to ssTown.
26. tempCnt has value set to the return value of greedyAlgorithm (CA84-CA120).
27. Checks if the tempCnt value is less than the current threshold.
28. The threshold is set to the value of tempCnt.
29. Check if lowestTown is less than MAX\_SEARCH.
30. tempCnt is set to the return value of recursiveSolver.
31. Check if the new value of tempCnt is less than the current threshold.
32. Set the threshold to be tempCnt.
33. Returns the value we found as a threshold.

Code Correctness Argument for function greedyAlgorithm:

1. Checks if nmbrTownsRemaining is equal to 0 or if ssCnt is equal to threshold.
2. Returns ssCnt.
3. If CA84 was not correct continue with next branch.
4. Sets value of maxConnections, townNum, and maxDeadEnds to 0.
5. Creates new array tempTowns.
6. copyNdx iterates from 0 to nmbrTowns.
7. Value of town with index copyNdx is copied to the temporary array.
8. copyNdx iterates from 0 to nmbrTownsInCluster for current cluster.
9. connectionNmbr and nmbrDeadEnds are both set to 0.
10. Check if the state is not already set to be ssTown.
11. Creates variable with name townChk to hold a town.
12. connNdx iterates from 0 to townChk.nmbrNeighbors.
13. Checks if the current neighbor being considered has not been serviced.
14. Increment connectionNmbr by 1.
15. Sets isDeadEnd to true.
16. deadCheckNdx iterates from 0 the number of neighbors the current neighbor has.
17. Check if the neighbor listed is the same as townChk.
18. Set the value of isDeadEnd to false.
19. Breaks the loop.
20. Check that isDeadEnd is true.
21. nmbrDeadEnds is incremented by one.
22. connectionNmbr is incremented by one.
23. Check if the town set to townChk has state ssAdjTwon
24. connectionNmbr is incremented by one.
25. Checks if the town being considered is a better candidate for the greedy algorithm.
26. Sets townNum to the current town being considered.
27. maxConnections is set to be connnectionNmbr.
28. maxDeadEnds is set to be nmbrDeadEnds.
29. checkNdx iterates from 0 to the number of neighbors the town has.
30. Check if town considered by checkNdx has not been serviced.
31. nmbrTownsRemaining is decremented by one.
32. State of neighbor is set to be ssAdjTown.
33. Check if the state is not already set to be ssAdjTown.
34. nmbrTownsRemaining is decremented by one.
35. ssCnt is incremented by one.
36. State of current town is updated to be ssTown.
37. Recursive call to greedyAlgorithm is returned.

Code Correctness Argument for function findNeighbor:

1. neighborNum is iterated from 0 to the nmbrNeighbors for townA.
2. Check if the neighbor with index neighborNum is equal to townB.
3. found is set to true.
4. The loop is broken.
5. The value of found is returned.

Code Correctness Argument for function addLinkedTownsInCluster:

1. linkNdx is iterated from 0 to nmbrNeighbors for town with index townNdx.
2. Check if the current neighbor has a state of notProcessedTown.
3. Adds the town to the cluster.
4. Sets the state of the town to townInCluster.
5. nmbrTownsInCluster is incremented by one.
6. Recursively calls addLinkedTownsInCluster to add more towns connected to elements of the   
    cluster.
7. The updated cluster is returned.

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

For documentation on testing script please read the comments at the top of serviceStationBTLtestRun.sh.

**Performance Test Procedure**

No performance tests were performed.

**Inspection Report(s)**

The reports of a design inspection are filed with the V&V manager.

**Deliverables**

Main Directory

* serviceStationBTL.cpp
* serviceStationSMDSv8BTL.docx
* serviceStationBTLtestRslts.txt
* serviceStationBTLtestRun.sh
* smdsForServiceStationTskLogv8BTL.xlsx
* testGenerator (Directory)
  + SSrandomTestBTL
  + 9 input files named ssBTLXXin.txt where XX is the two digit number of the test file
* unit\_tests (Directory)
  + 15 input files named ssInXX.txt where XX is the two digit number of the test file
  + 15 expected output files named ssOutXX.txt where XX is the two digit number of the test file
* myRandomTestResults.txt
* otherRandomTestResults.txt

**Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| *Version* | *Date* | *Author* | *Comment* |
| 1.0 | 9/8/2016 | Trevor Brooks | Initial SMDS with requirements |
| 1.1 | 9/20/2016 | Trevor Brooks | Added input partitioning and test cases. |
| 1.2 | 9/27/2016 | Trevor Brooks | Initial algorithm and design sections. |
| 1.3 | 10/4/2016 | Trevor Brooks | Added algorithm correctness |
| 1.4 | 10/18/2016 | Trevor Brooks | Started implementation, minor revisions to design section. |
| 1.5 | 10/25/2016 | Trevor Brooks | Worked on “Test Directories, Files, Scripts/Scenarios” reflecting the test shell script |
| 1.6 | 11/1/2016 | Trevor Brooks | Reworked SMDS Design and Correctness sections to reflect working implementation with a backtracking algorithm. |
| 1.7 | 11/15/16 | Trevor Brooks | Added Inspection Reports, Performance Test Procedure, Deliverables, and Version History sections. |