SOURCE CODE

```
// William Kelley
// Algorithms - Assignment 1
// Graphs
// Source for Help: A lot of stackoverflow, nothing in particular was used but
// definitely pulled a lot of references and similarities from.

#include <iostream>
#include <cstdlib>
#include <string>
#include <cstring>
#include <iomanip>
#include <vector>
#include <list>
#include <algorithm>
#include <fstream>
#include <sstream>

std::ifstream infile;
std::ofstream outfile;

using namespace std;

template <class T>
class Graph {
public:
	Graph() {};
	~Graph<T>() {};
	//virtual bool adjacent(T x, T y) { return false; };
	//virtual vector<T> neighbors(T x) {};
	virtual void addNode() { cout << "Parent class 'addNode()'" << endl; };
	virtual void deleteNode() { cout << "Parent class 'deleteNode()'" << endl; };
	virtual void addEdge() { cout << "Parent class 'addEdge()'" << endl; };
	virtual void deleteEdge() { cout << "Parent class 'deleteEdge()'" << endl; };
};

template <class T>
class AdjacencyMatrix : public Graph<T> {
private:
	std::vector<T> adjmatrix;
public:
	AdjacencyMatrix() {};
	~AdjacencyMatrix() {};
	std::vector<T> getGraph() const;
	void inputGraph(string s);
	void printGraph() const;
	void printGraph(vector<T>) const;
	bool adjacent(T x, T y);
	std::vector<T> neighbors(T x);
	void addNode(T x);
	void deleteNode(T x);
	void addEdge(T x, T y);
	void deleteEdge(T x, T y);
};

template <class T>
class AdjacencyList : public Graph<T> {
private:
	std::list<T> adjlist;
public:
	AdjacencyList() {};
	~AdjacencyList() {};
	std::list<T> getGraph() const;
	void inputGraph(string s);
	void printGraph() const;
	void printGraph(vector<T>) const;
	bool adjacent(T x, T y);
	std::vector<T> neighbors(T x);
	void addNode(T x);
	void deleteNode(T x);
	void addEdge(T x, T y);
	void deleteEdge(T x, T y);
};

template<class T>
std::vector<T> AdjacencyMatrix<T>::getGraph() const
{
	return adjmatrix;
}

template<class T>
void AdjacencyMatrix<T>::inputGraph(string s)
{
	string input;
	T array[20];

	infile.open(s);
	while (std::getline(infile, input))
	{
		T x;
		replace(input.begin(), input.end(), ',', ' ');
		replace(input.begin(), input.end(), ':', ' ');
		input.erase(std::remove(input.begin(), input.end(), ' '), input.end());
		for (auto i = 0; i < input.length(); ++i)
		{
```

```
            array[i] = input[i] - '0';
        }
        x = array[0];
        for (auto j = 0; j < input.length(); ++j)
        {
            addEdge(x, array[j]);
        }
    }
    infile.close();
}

template<class T>
void AdjacencyMatrix<T>::printGraph() const
{
    cout << endl;
    for (auto i = adjmatrix.begin(); i != adjmatrix.end(); ++i)
    {
        std::cout << (*i / 10) << "->" << (*i % 10) << '\n';
    }
    cout << endl;
}

template<class T>
void AdjacencyMatrix<T>::printGraph(vector<T> x) const
{
    for (auto i = x.begin(); i != x.end(); ++i)
    {
        std::cout << (*i / 10) << "->" << (*i % 10) << '\n';
    }
    cout << endl;
}

template<class T>
bool AdjacencyMatrix<T>::adjacent(T x, T y)
{
    return false;
}

template<class T>
std::vector<T> AdjacencyMatrix<T>::neighbors(T x)
{
    std::vector<T> returnVector;
    std::vector<T> currentVector = getGraph();
    T singleDigit = NULL;
    T nodeId = (x * 10);
    T max = nodeId + 9;
    for (nodeId; nodeId <= max; ++nodeId)
    {
            for (auto i = currentVector.begin(); i != currentVector.end(); ++i)
            {
                  if (*i == nodeId)
                  {
                        returnVector.push_back(*i);
                  }
                  else
                  {
                        void;
                  }
            }
      }
      cout << endl;

      return returnVector;
}

template<class T>
void AdjacencyMatrix<T>::addNode(T x)
{
      T point = x + (x * 10);          // this allows for the user to combine the
points into a single value to be stored
      adjmatrix.erase(std::remove(adjmatrix.begin(), adjmatrix.end(), point),
adjmatrix.end());
      adjmatrix.push_back(point);
}

template<class T>
void AdjacencyMatrix<T>::deleteNode(T x)
{
      T point = (x * 10);
      for (T i = 0; i <= 9; ++i)
      {
            adjmatrix.erase(std::remove(adjmatrix.begin(), adjmatrix.end(),
point+i), adjmatrix.end());
      }
}

template<class T>
void AdjacencyMatrix<T>::addEdge(T x, T y)
{
      T point = (x * 10) + y;
      adjmatrix.erase(std::remove(adjmatrix.begin(), adjmatrix.end(), point),
adjmatrix.end());   //deleting edge if it exists and re-adding just for ease
      adjmatrix.push_back(point);
}

template<class T>
```

```
void AdjacencyMatrix<T>::deleteEdge(T x, T y)
{
	T point = (x * 10) + y;
	std::vector<int>::iterator it;
	it = find(adjmatrix.begin(), adjmatrix.end(), point);
	if (it != adjmatrix.end())
	{
		cout << "Edge found, deleteing edge." << endl;
		adjmatrix.erase(std::remove(adjmatrix.begin(), adjmatrix.end(), point),
adjmatrix.end());
	}
	else
	{
		cout << "Unable to locate edge." << endl;
	}
}

template<class T>
std::list<T> AdjacencyList<T>::getGraph() const
{
	return adjlist;
}

template<class T>
void AdjacencyList<T>::inputGraph(string s)
{
	string input;
	T array[20];

	infile.open(s);
	while (std::getline(infile, input))
	{
		T x;
		replace(input.begin(), input.end(), ',', ' ');
		replace(input.begin(), input.end(), ':', ' ');
		input.erase(std::remove(input.begin(), input.end(), ' '), input.end());
		for (auto i = 0; i < input.length(); ++i)
		{
			array[i] = input[i] - '0';
		}
		x = array[0];
		for (auto j = 0; j < input.length(); ++j)
		{
			addEdge(x, array[j]);
		}
	}
	infile.close();
}
```

```
template<class T>
void AdjacencyList<T>::printGraph() const
{
    cout << endl;

    for (auto i = adjlist.begin(); i != adjlist.end(); ++i)
    {
        std::cout << (*i / 10) << "->" << (*i % 10) << '\n';
    }
    cout << endl;
}

template<class T>
void AdjacencyList<T>::printGraph(vector<T> x) const
{
    for (auto i = x.begin(); i != x.end(); ++i)
    {
        std::cout << (*i/10) << "->" << (*i%10) << '\n';
    }
    cout << endl;
}

template<class T>
bool AdjacencyList<T>::adjacent(T x, T y)
{
    return false;
}

template<class T>
std::vector<T> AdjacencyList<T>::neighbors(T x)
{
    std::vector<T> returnVector;
    std::list<T> currentList = getGraph();
    T singleDigit = NULL;
    T nodeId = (x * 10);
    T max = nodeId + 9;
    for (nodeId; nodeId <= max; ++nodeId)
    {
        for (auto i = currentList.begin(); i != currentList.end(); ++i)
        {
            if (*i == nodeId)
            {
                returnVector.push_back(*i);
            }
            else
            {
                void;
```

```
			}
		}
	}
	cout << endl;

	return returnVector;
}

template<class T>
void AdjacencyList<T>::addNode(T x)
{
	T point = x + (x * 10);		// this allows for the user to combine the
points into a single value to be stored
	adjlist.erase(std::remove(adjlist.begin(), adjlist.end(), point),
adjlist.end());
	adjlist.push_back(point);
}

template<class T>
void AdjacencyList<T>::deleteNode(T x)
{
	T point = (x * 10);
	for (T i = 0; i <= 9; ++i)
	{
		adjlist.erase(std::remove(adjlist.begin(), adjlist.end(), point + i),
adjlist.end());
	}
}

template<class T>
void AdjacencyList<T>::addEdge(T x, T y)
{
	T point = (x * 10) + y;
	adjlist.erase(std::remove(adjlist.begin(), adjlist.end(), point),
adjlist.end());	//deleting edge if it exists and re-adding just for ease
	adjlist.push_back(point);
}

template<class T>
void AdjacencyList<T>::deleteEdge(T x, T y)
{
	T point = (x * 10) + y;
	std::list<int>::iterator it;
	it = find(adjlist.begin(), adjlist.end(), point);
	if (it != adjlist.end())
	{
		cout << "Edge found, deleteing edge." << endl;
		adjlist.erase(std::remove(adjlist.begin(), adjlist.end(), point), 
adjlist.end());
	}
	else
	{
		cout << "Unable to locate edge." << endl;
	}
}


int main()
{
	AdjacencyMatrix<int> matrix;
	AdjacencyList<int> list;

	int inputX;
	int inputY;

	string fileName = "text.txt";

	cout << "Note to testers: only accepts single digit values.\n\n";

	cout << "Input File and Print for Matrix\n";

	matrix.inputGraph(fileName);
	matrix.printGraph();

	cout << "Input File and Print for List\n";

	list.inputGraph(fileName);
	list.printGraph();

	cout << "Add Node to Matrix(enter 0 to exit): \n";
	cin >> inputX;
	while (inputX != 0) {
		matrix.addNode(inputX);
		cout << "\nEnter another node to enter or enter 0 to exit\n";
		cin >> inputX;
	}

	matrix.printGraph();

	cout << "Delete Node from Matrix(enter 0 to exit): \n";
	cin >> inputX;
	while (inputX != 0) {
		matrix.deleteNode(inputX);
		cout << "\nEnter another node to delete or enter 0 to exit\n";
		cin >> inputX;
```

```
        }

        matrix.printGraph();

        cout << "Add Edge to Matrix(enter 0 to exit): x,y with values separated by
space\n";
        cin >> inputX >> inputY;
        while (inputX != 0) {
                matrix.addNode(inputX);     // by my theory if there's a node, then it
relates to itself as well
                matrix.addEdge(inputX, inputY);
                cout << "\nEnter another edge to enter or enter 0 0 to exit\n";
                cin >> inputX >> inputY;
        }

        matrix.printGraph();

        cout << "Delete Edge from Matrix(enter 0 to exit): x,y with values separated
by space\n";
        cin >> inputX >> inputY;
        while (inputX != 0) {
                matrix.deleteEdge(inputX, inputY);
                cout << "\nEnter another edge to delete or enter 0 0 to exit\n";
                cin >> inputX >> inputY;
        }

        matrix.printGraph();

        cout << "What Node would you like to know the neighbors of? (enter node or 0
to exit)\n";
        cin >> inputX;
        while (inputX != 0) {
                matrix.printGraph(matrix.neighbors(inputX));
                cout << "\nEnter another node to find out neighbors or enter 0 to
exit\n";
                cin >> inputX;
        }

        cout << "\n\nSince my functions are virtually the same for either, I've only
displayed the Matrix\n";

        system("PAUSE");
}
```

CONSOLE PRINT OUT

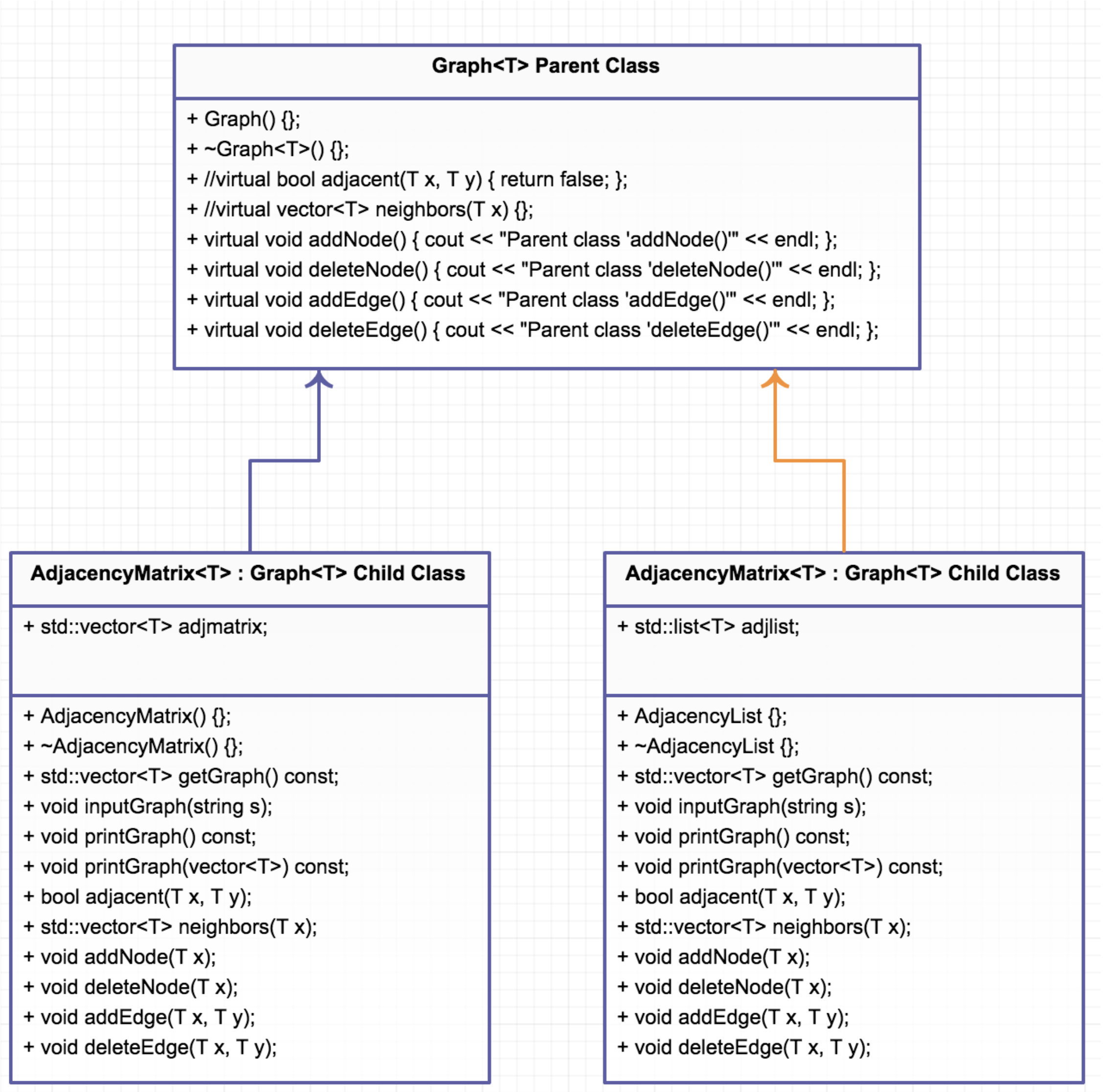
Note to testers: only accepts single digit values.

```
Input File and Print for Matrix

1->1
1->2
1->3
1->4
2->2
2->4
2->5
3->3
3->1
3->5
4->4
4->1

Input File and Print for List

1->1
1->2
1->3
1->4
2->2
2->4
2->5
3->3
3->1
3->5
4->4
4->1

Add Node to Matrix(enter 0 to exit):
5

Enter another node to enter or enter 0 to exit
6

Enter another node to enter or enter 0 to exit
0

1->1
1->2
1->3
1->4
2->2
2->4
2->5
3->3
```

```
3->1
3->5
4->4
4->1
5->5
6->6

Delete Node from Matrix(enter 0 to exit):
1

Enter another node to delete or enter 0 to exit
5

Enter another node to delete or enter 0 to exit
6

Enter another node to delete or enter 0 to exit
0

2->2
2->4
2->5
3->3
3->1
3->5
4->4
4->1

Add Edge to Matrix(enter 0 to exit): x,y with values separated by space
5 9

Enter another edge to enter or enter 0 0 to exit
6 2

Enter another edge to enter or enter 0 0 to exit
0 0

2->2
2->4
2->5
3->3
3->1
3->5
4->4
4->1
5->5
5->9
6->6
6->2
```

```
Delete Edge from Matrix(enter 0 to exit): x,y with values separated by space
2 4
Edge found, deleteing edge.

Enter another edge to delete or enter 0 0 to exit
6 2
Edge found, deleteing edge.

Enter another edge to delete or enter 0 0 to exit
0 0

2->2
2->5
3->3
3->1
3->5
4->4
4->1
5->5
5->9
6->6

What Node would you like to know the neighbors of? (enter node or 0 to exit)
1



Enter another node to find out neighbors or enter 0 to exit
2

2->2
2->5


Enter another node to find out neighbors or enter 0 to exit
3

3->1
3->3
3->5


Enter another node to find out neighbors or enter 0 to exit
0


Since my functions are virtually the same for either, I've only displayed the Matrix
Press any key to continue . . .
```

UML DIAGRAM



Graph<T> Parent Class

+ Graph() {};
+ ~Graph<T>() {};
+ //virtual bool adjacent(T x, T y) { return false; };
+ //virtual vector<T> neighbors(T x) {};
+ virtual void addNode() { cout << "Parent class 'addNode()'" << endl; };
+ virtual void deleteNode() { cout << "Parent class 'deleteNode()'" << endl; };
+ virtual void addEdge() { cout << "Parent class 'addEdge()'" << endl; };
+ virtual void deleteEdge() { cout << "Parent class 'deleteEdge()'" << endl; };

AdjacencyMatrix<T> : Graph<T> Child Class

+ std::vector<T> adjmatrix;

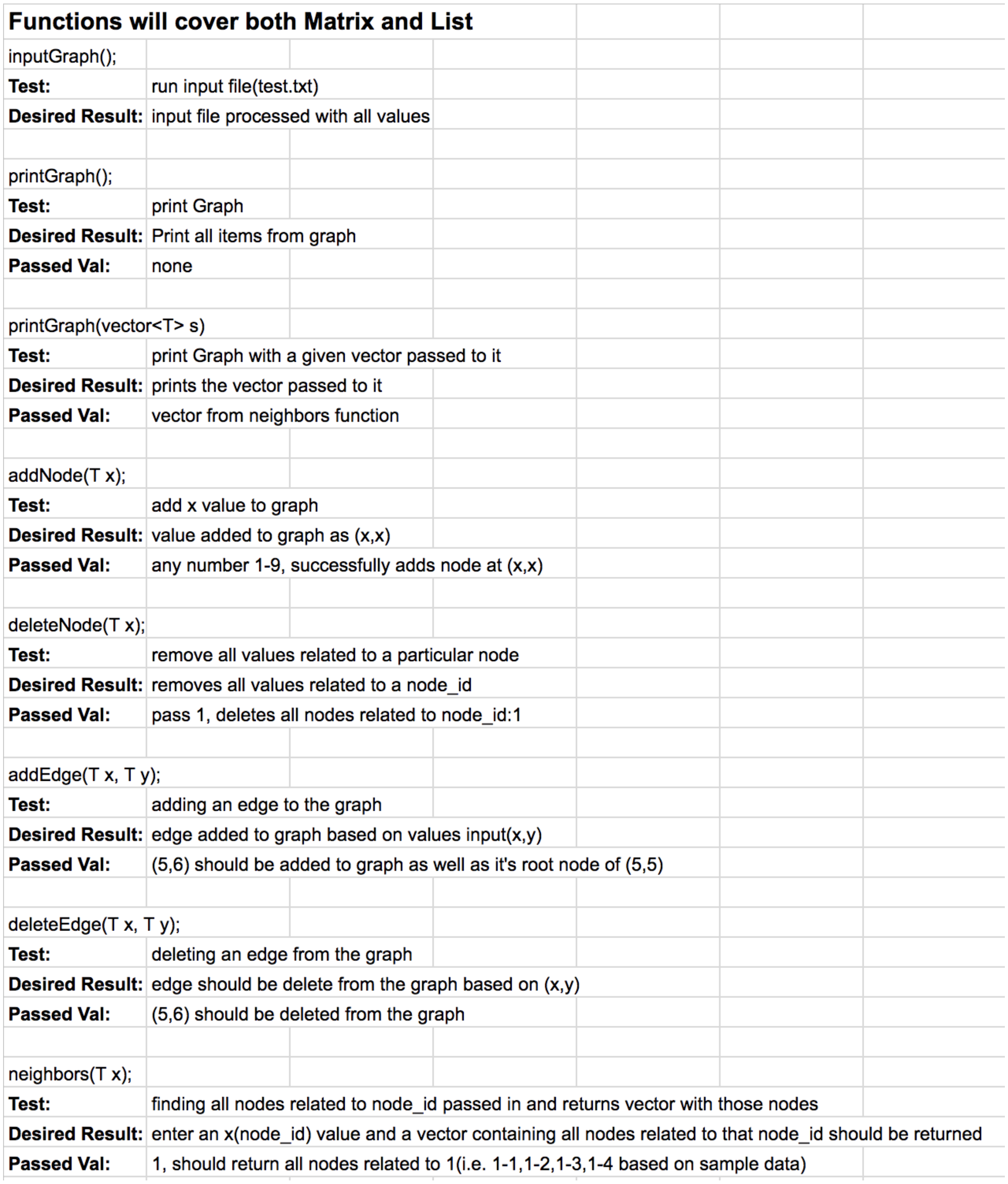
+ AdjacencyMatrix() {};
+ ~AdjacencyMatrix() {};
+ std::vector<T> getGraph() const;
+ void inputGraph(string s);
+ void printGraph() const;
+ void printGraph(vector<T>) const;
+ bool adjacent(T x, T y);
+ std::vector<T> neighbors(T x);
+ void addNode(T x);
+ void deleteNode(T x);
+ void addEdge(T x, T y);
+ void deleteEdge(T x, T y);

AdjacencyMatrix<T> : Graph<T> Child Class

+ std::list<T> adjlist;

+ AdjacencyList {};
+ ~AdjacencyList {};
+ std::vector<T> getGraph() const;
+ void inputGraph(string s);
+ void printGraph() const;
+ void printGraph(vector<T>) const;
+ bool adjacent(T x, T y);
+ std::vector<T> neighbors(T x);
+ void addNode(T x);
+ void deleteNode(T x);
+ void addEdge(T x, T y);
+ void deleteEdge(T x, T y);

UNIT TESTING



Functions will cover both Matrix and List	
inputGraph();	
Test:	run input file(test.txt)
Desired Result:	input file processed with all values
printGraph();	
Test:	print Graph
Desired Result:	Print all items from graph
Passed Val:	none
printGraph(vector<T> s)	
Test:	print Graph with a given vector passed to it
Desired Result:	prints the vector passed to it
Passed Val:	vector from neighbors function
addNode(T x);	
Test:	add x value to graph
Desired Result:	value added to graph as (x,x)
Passed Val:	any number 1-9, successfully adds node at (x,x)
deleteNode(T x);	
Test:	remove all values related to a particular node
Desired Result:	removes all values related to a node_id
Passed Val:	pass 1, deletes all nodes related to node_id:1
addEdge(T x, T y);	
Test:	adding an edge to the graph
Desired Result:	edge added to graph based on values input(x,y)
Passed Val:	(5,6) should be added to graph as well as it's root node of (5,5)
deleteEdge(T x, T y);	
Test:	deleting an edge from the graph
Desired Result:	edge should be delete from the graph based on (x,y)
Passed Val:	(5,6) should be deleted from the graph
neighbors(T x);	
Test:	finding all nodes related to node_id passed in and returns vector with those nodes
Desired Result:	enter an x(node_id) value and a vector containing all nodes related to that node_id should be returned
Passed Val:	1, should return all nodes related to 1(i.e. 1-1,1-2,1-3,1-4 based on sample data)