```
#include <set>
#include <list>
#include <vector>
#include <fstream>
#include <iostream>
#include "d_util.h"
#include <string>

// Written by Eric Wasiolek Copyright 2008 except set operators + * - ==

class neuron{
	private:
		string name;
		string lineage;
		string maintype;
		string subtype1;
		string subtype2;
		set<string> genes;

	public:
		neuron()
		{}
		void setName(string x)
		{
			name = x;
		}
		void setLineage(string x)
```

```
{
	lineage = x;
}
void setMainType(string x)
{
	maintype = x;
}
void setSubType1(string x)
{
	subtype1 = x;
}
void setSubType2(string x)
{
	subtype2 = x;
}
void addGene(string gene)
{
	genes.insert(gene);
}
string getName()
{
	return name;
}
string getLineage()
{
	return lineage;
}
string getMainType()
{
```

```
                return maintype;
        }
        string getSubType1()
        {
                return subtype1;
        }
        string getSubType2()
        {
                return subtype2;
        }
        set<string> getGenes()
        {
                return genes;
        }
};

class syn {
        public:
                string n1;
                string n2;
                int wt;
        };

// SET FUNCTION IMPLEMENTATIONS

template <typename T>
bool operator== (const set<T>& lhs, const set<T>& rhs)
{
        typename set<T>::const_iterator myself = lhs.begin(), other = rhs.begin();
```

```
	// return false if the sets do not have the same size
	if (lhs.size() == rhs.size())
	{
		// compare until encounter end of the sets or
		// find two elements that are not equal
		while (myself != lhs.end() && *myself++ == *other++);

		// if we left the loop before reaching the end
		// of the sets, they are not equal
		if (myself != lhs.end())
			return false;
		else
			return true;
	}
	else
		return false;
}

template <typename T>
set<T> operator+ (const set<T>& lhs, const set<T>& rhs)
{
	// constuct union
	set<T> setUnion;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
```

```
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*lhsIter++);
		else if (*rhsIter < *lhsIter)
			// *rhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*rhsIter++);
		else
		{
			// the two values are equal. insert just one and
			// move both itertors forward
			setUnion.insert(*lhsIter++);
			rhsIter++;
		}

		// flush any remaining items
		if (lhsIter != lhs.end())
			while (lhsIter != lhs.end())
				setUnion.insert(*lhsIter++);
		else if (rhsIter != rhs.end())
			while (rhsIter != rhs.end())
				setUnion.insert(*rhsIter++);

	return setUnion;
}

template <typename T>
```

```
set<T> operator* (const set<T>& lhs, const set<T>& rhs)
{
	// constuct intersection
	set<T> setIntersection;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter is in lhs and not in rhs. move iterator
			// forward
			lhsIter++;
		else if (*rhsIter < *lhsIter)
			// *rhsIter is in rhs and not in lhs. move iterator
			// forward
			rhsIter++;
		else
		{
			// the same value is in both sets. insert one value
			// and move the iterators forward
			setIntersection.insert(*lhsIter);
			lhsIter++;
			rhsIter++;
		}

		return setIntersection;
}
```

```
template <typename T>
set<T> operator- (const set<T>& lhs, const set<T>& rhs)
{
	// construct difference
	set<T> setDifference;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter belongs to lhs but not to rhs. put it in
			// the difference
			setDifference.insert(*lhsIter++);
		else if (*rhsIter < *lhsIter)
			// *rhsIter is in the rhs but not in the lhs. pass
			// over it
			rhsIter++;
		else
		{
			// the same value is in both sets. move the
			// iterators forward
			lhsIter++;
			rhsIter++;
		}

		// flush any remaining items from lhs
```

```
                if (lhsIter != lhs.end())
                        while (lhsIter != lhs.end())
                                setDifference.insert(*lhsIter++);

        return setDifference;
}

set<string> findGeneSet(string neuronname, neuron neuron[], int numneurons)
{
        for(int i=0; i < numneurons; ++i)
                if(neuronname == neuron[i].getName())
                        return neuron[i].getGenes();
}

int min(neuron n[], int numneurons)
{
        int min = 100;
        for(int i = 0; i < numneurons; ++i)
                if(n[i].getLineage().length() < min)
                        min = n[i].getLineage().length();
        return min;
}

int max(neuron n[], int numneurons)
{
        int max = 0;
        for(int i = 0; i < numneurons; ++i)
                if(n[i].getLineage().length() > max)
                        max = n[i].getLineage().length();
```

```
	return max;
}

bool find(vector<string> n, string x)
{
	int vsize = n.size();
	for(int i = 0; i < vsize; ++i)
		if(n[i] == x)
			return true;
		else if(i == vsize-1)
			return false;
}

// ofstream outfile;
// outfile.open("devstagesgenes");

template< typename Iterator>
void writeContainerFile(Iterator first, Iterator last, const string& separator, ofstream& output)
{
	Iterator iter = first;
	while(iter != last)
	{
		output << *iter << separator;
		iter++;
	}
}


int main(int argc, char *argv[])
```

```
{
	// cout << "Entered main" << endl;
	ifstream neurons;
	string inputneuron, outputneuron;
	ofstream outfile;

	if(argc != 3)
	{
		cout << "Usage: develstages graphname outputfile " << endl;
		return 0;
	}
	else
	{
		neurons.open(argv[1]);
		if(!neurons)
		{
			cout << "Could not open graph file!" << endl;
			return 0;
		}
		else
			cout << "Opened graph file " << argv[1] << " successfully." << endl;
		outfile.open(argv[2]);
		if(!outfile)
		{
			cout << "Could not open output file!" << endl;
			return 0;
		}
		else
			cout << "Opened output file " << argv[2] << " successfully." << endl;
```

```
}



int numneurons = 0;
string gene;
vector<neuron> vneurons(numneurons);
neurons >> numneurons;
string temp;
neuron n[numneurons];
int counter = 0;
for(int i = 0; i < numneurons; ++i)
{
        neurons >> temp;
        n[i].setName(temp);
        ++counter;
        neurons >> temp;
        n[i].setLineage(temp);
        ++counter;
        neurons >> temp;
        n[i].setMainType(temp);
        ++counter;
        neurons >> temp;
        n[i].setSubType1(temp);
        ++counter;
        neurons >> temp;
        n[i].setSubType2(temp);
        ++counter;
        while(1)
```

```
                {
                        neurons >> gene;
                                if(gene == "end")
                                        break;
                        n[i].addGene(gene);
                        ++counter;
                }
        }

int maxstring = max(n, numneurons);
cout << "Diag:  maxstring: " << maxstring << endl;
int minstring = min(n, numneurons);
cout << "Diag: minstring: " << minstring << endl;
int numfiles = maxstring - minstring + 1;

int filenum = 0;

int stage = 1;

set<string> already_genes;

for(int length=minstring; length <= maxstring; ++length)
{
        vector<string> vneur;
        vector<string> vnewneur;
        vector<string> vs;
        vector<string> vm;
        vector<string> vi;
        vector<string> vcs;
```

```
vector<string> vcm;
vector<string> vci;

int stageedges = 0;
outfile << endl << endl << "Development Stage: " << stage << endl << endl;

// Find all neurons of lineage string length length or less

int i = 0;
for(i = 0; i < numneurons; ++i)
        if(n[i].getLineage().length() <= length)
        {
                vneur.push_back(n[i].getName());
                if(n[i].getMainType() == "SN")
                        vs.push_back(n[i].getName());
                else if(n[i].getMainType() == "MN")
                        vm.push_back(n[i].getName());
                else if(n[i].getMainType() == "IN")
                        vi.push_back(n[i].getName());
        }


if(stage > 1)
{

// set<string> newly_activated_genes;

for(int i = 0; i < numneurons; ++i)
{
```

```
        if(n[i].getLineage().length() == length)
                vnewneur.push_back(n[i].getName());
        if((n[i].getLineage().length() == length) && (n[i].getMainType() == "SN"))
                vcs.push_back(n[i].getName());
        else if((n[i].getLineage().length() == length) && (n[i].getMainType() == "MN"))
                vcm.push_back(n[i].getName());
        else if((n[i].getLineage().length() == length) && (n[i].getMainType() == "IN"))
                vci.push_back(n[i].getName());
}
}

if(!vnewneur.empty())
{
        outfile << endl << "New neurons at this stage: " << endl;
        writeContainerFile(vnewneur.begin(), vnewneur.end(), " ", outfile);
        outfile << endl << "New sensory neurons at this stage: " << endl;
        writeContainerFile(vcs.begin(), vcs.end(), " ", outfile);

        outfile << endl << "New motor neurons at this stage: " << endl;
        writeContainerFile(vcm.begin(), vcm.end(), " ", outfile);

        outfile << endl << "New inter-neurons at this stage: " << endl;
        writeContainerFile(vci.begin(), vci.end(), " ", outfile);
}
        outfile << endl << "All Sensory neurons at this stage: " << endl;
        writeContainerFile(vs.begin(), vs.end(), " ", outfile);

        outfile << endl << "All Motor neurons at this stage: " << endl;
        writeContainerFile(vm.begin(), vm.end(), " ", outfile);
```

```
        outfile << endl << "All Inter-neurons at this stage: " << endl;
        writeContainerFile(vi.begin(), vi.end(), " ", outfile);

outfile << endl << "Genes at this development stage: " << endl;
vector<set<string> > vs_stage;
vector<set<string> > vs_sensory;
vector<set<string> > vs_motor;
vector<set<string> > vs_inter;
vector<set<string> > vs_new_sensory;
vector<set<string> > vs_new_motor;
vector<set<string> > vs_new_inter;
set<string> newly_activated_genes;

int vneursize = vneur.size();
for(int i = 0; i < vneursize; ++i)
        vs_stage.push_back(findGeneSet(vneur[i], n, numneurons));
int vssize = vs.size();
for(int i = 0; i < vssize; ++i)
        vs_sensory.push_back(findGeneSet(vs[i], n, numneurons));
int vmsize = vm.size();
for(int i = 0; i < vmsize; ++i)
        vs_motor.push_back(findGeneSet(vm[i], n, numneurons));
int visize = vi.size();
for(int i = 0; i < visize; ++i)
        vs_inter.push_back(findGeneSet(vi[i], n, numneurons));
set<string>::iterator fsiter;
set<string>::iterator esiter;
set<string> us_stage;
```

```
	us_stage = vs_stage[0];
	for(int i = 0; i < vs_stage.size(); ++i)
		us_stage = us_stage + vs_stage[i];
	if(stage == 1)
		already_genes = us_stage;
	fsiter = us_stage.begin();
	esiter = us_stage.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
	if(us_stage.empty())
		cout << "There are no genes at this development stage." << endl;
cout << "Made it to 4" << endl;

	set<string> us_sensory;
	set<string> us_motor;
	set<string> us_inter;

	if(!vs_sensory.empty())
	{
	us_sensory = vs_sensory[0];
	for(int i = 0; i < vs_sensory.size(); ++i)
		us_sensory = us_sensory + vs_sensory[i];
	outfile << endl << "Sensory Neuron Genes at this stage: " << endl;
	fsiter = us_sensory.begin();
	esiter = us_sensory.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
	}
	else
		outfile << "There are no sensory neuron genes at this stage." << endl;
```

```
if(!vs_motor.empty())
{
us_motor = vs_motor[0];
for(int i = 0; i < vs_motor.size(); ++i)
        us_motor = us_motor + vs_motor[i];
outfile << endl << "Motor Neuron Genes at this stage: " << endl;
fsiter = us_motor.begin();
esiter = us_motor.end();
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
        outfile << "There are no motor neuron genes at this stage." << endl;

if(!vs_inter.empty())
{
us_inter = vs_inter[0];
for(int i = 0; i < vs_inter.size(); ++i)
        us_inter = us_inter + vs_inter[i];
fsiter = us_inter.begin();
esiter = us_inter.end();
outfile << endl << "Inter-neuron genes at this stage: " << endl;
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
        outfile << "There are no inter-neuron genes at this stage." << endl;

outfile << endl << "Sensory-specific genes: " << endl;
set<string> ss;
ss = us_sensory - us_motor - us_inter;
```

```
if(!ss.empty())
{
fsiter = ss.begin();
esiter = ss.end();
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
        outfile << endl << "There are no sensory-specific genes." << endl;

outfile << endl << "Motor-specific genes: " << endl;
set<string> ms;
ms = us_motor - us_sensory - us_inter;
if(!ms.empty())
{
fsiter = ms.begin();
esiter = ms.end();
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
        outfile << endl << "There are no motor-specific genes." << endl;

outfile << endl << "Inter-neuron specific genes:" << endl;
set<string> is;
is = us_inter - us_sensory - us_motor;
if(!is.empty())
{
fsiter = is.begin();
esiter = is.end();
writeContainerFile(fsiter, esiter, " ", outfile);
```

```
	}
	else
		outfile << endl << "There are no interneuron specific genes." << endl;


	outfile << endl << "Genes common to sensory neurons, motor neurons, and interneurons at this stage: " << endl;
	set<string> common;
	common = us_inter * us_sensory * us_motor;
	if(!common.empty())
	{
	fsiter = common.begin();
	esiter = common.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
	}
	else
		outfile << endl << "There are no genes common to sensory neurons, motor neurons, and interneurons at this stage." << endl;



	if(stage > 1)
	{


	if(!vnewneur.empty())
	{
	outfile << endl << "New genes at this stage: " << endl;
      int vnewneursize = vnewneur.size();
	vector<set<string> > vs_new_stage;
	for(int i = 0; i < vnewneursize; ++i)
		vs_new_stage.push_back(findGeneSet(vnewneur[i], n, numneurons));
```

```
set<string> us_new_stage;
us_new_stage = vs_new_stage[0];
for(int i = 0; i < vs_new_stage.size(); ++i)
        us_new_stage = us_new_stage + vs_new_stage[i];
newly_activated_genes = us_new_stage - already_genes;
fsiter = us_new_stage.begin();
esiter = us_new_stage.end();
writeContainerFile(fsiter, esiter, " ", outfile);
if(us_new_stage.empty())
        outfile << "There are no new genes at this development stage." << endl;

if(!newly_activated_genes.empty())
{
        outfile << endl << "Newly activated genes this stage: " << endl;
        fsiter = newly_activated_genes.begin();
        esiter = newly_activated_genes.end();
        writeContainerFile(fsiter, esiter, " ", outfile);
}
else
        outfile << endl << "There are no newly activated genes this stage." << endl;



set<string> us_new_sensory;
set<string> us_new_motor;
set<string> us_new_inter;

if(!vcs.empty())
```

```
{
int vcssize = vcs.size();
for(int i = 0; i < vcssize; ++i)
        vs_new_sensory.push_back(findGeneSet(vcs[i], n, numneurons));
// set<string> us_new_sensory;
us_new_sensory = vs_new_sensory[0];
for(int i = 0; i < vs_new_sensory.size(); ++i)
        us_new_sensory = us_new_sensory + vs_new_sensory[i];
fsiter = us_new_sensory.begin();
esiter = us_new_sensory.end();
outfile << endl << "New sensory neuron genes at this stage: " << endl;
writeContainerFile(fsiter, esiter, " ", outfile);
}

if(!vcm.empty())
{
int vcmsize = vcm.size();
for(int i = 0; i < vcmsize; ++i)
        vs_new_motor.push_back(findGeneSet(vcm[i], n, numneurons));
// set<string> us_new_motor;
us_new_motor = vs_new_motor[0];
for(int i = 0; i < vs_new_motor.size(); ++i)
        us_new_motor = us_new_motor + vs_new_motor[i];
fsiter = us_new_motor.begin();
esiter = us_new_motor.end();
outfile << endl << "New motor neuron genes at this stage: " << endl;
writeContainerFile(fsiter, esiter, " ", outfile);
}
```

```
if(!vci.empty())
{
int vcisize = vci.size();
for(int i = 0; i < vcisize; ++i)
        vs_new_inter.push_back(findGeneSet(vci[i], n, numneurons));
// set<string> us_new_inter;
us_new_inter = vs_new_inter[0];
for(int i = 0; i < vs_new_inter.size(); ++i)
        us_new_inter = us_new_inter + vs_new_inter[i];
fsiter = us_new_inter.begin();
esiter = us_new_inter.end();
outfile << endl << "New interneuron genes at this stage: " << endl;
writeContainerFile(fsiter, esiter, " ", outfile);
}

outfile << endl << "New sensory-specific genes: " << endl;
set<string> us_new_spec_sensory;
us_new_spec_sensory = us_new_sensory - us_new_motor - us_new_inter;
if(!us_new_spec_sensory.empty())
{
        fsiter = us_new_spec_sensory.begin();
        esiter = us_new_spec_sensory.end();
        writeContainerFile(fsiter, esiter, " ", outfile);
}

outfile << endl << "New motor-specific genes: " << endl;
set<string> us_new_spec_motor;
us_new_spec_motor = us_new_motor - us_new_sensory - us_new_inter;
if(!us_new_spec_motor.empty())
```

```
{
	fsiter = us_new_spec_motor.begin();
	esiter = us_new_spec_motor.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
}

outfile << endl << "New interneuron-specific genes: " << endl;
set<string> us_new_spec_inter;
us_new_spec_inter = us_new_inter - us_new_sensory - us_new_motor;
if(!us_new_spec_inter.empty())
{
	fsiter = us_new_spec_inter.begin();
	esiter = us_new_spec_inter.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
}

outfile << endl << "Genes common to new neurons at this stage: " << endl;
set<string> new_common;
new_common = us_new_sensory * us_new_motor * us_new_inter;
if(!new_common.empty());
{
	fsiter = new_common.begin();
	esiter = new_common.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
}

already_genes = already_genes + us_new_stage;

}
```

```
}

// write out neuron list
vneursize = vneur.size();
cout << "Diag:  vneursize is now: " << vneursize << endl;
outfile << endl << vneursize << endl;
for(int i=0; i < vneursize; ++i)
        outfile << vneur[i] << endl;



vector<syn> postsyn;
vector<syn> presyn;
vector<syn> newconnection;
vector<syn> oldconnection;
syn vsyn;

// file rewind
neurons.clear();
neurons.seekg(0);
string temp;
for(int i = 0; i < counter+numneurons+1; ++i)
        neurons >> temp;
cout << "Diag temp: " << temp << endl;
int numedges = 0;
string neuron1, neuron2;
int weight = 1;
neurons >> numedges;
cout << "Diag: numedges is: " << numedges << endl;
for(int i = 0; i < numedges; ++i)
```

```
{
        neurons >> neuron1;
        neurons >> neuron2;
        neurons >> weight;
        if(find(vneur, neuron1) && find(vneur, neuron2))
        {
                outfile << neuron1 << " " << neuron2 << " " << weight << endl;
                ++stageedges;
        }

        if(stage > 1)
        {
        if((find(vnewneur, neuron1) == true) && (find(vnewneur, neuron2) == false) &&
(find(vneur, neuron2) == true))
        {
                vsyn.n1 = neuron1;
                vsyn.n2 = neuron2;
                vsyn.wt = weight;
                presyn.push_back(vsyn);
        }
        else if ((find(vnewneur, neuron1) == false) && (find(vneur, neuron1) == true) &&
(find(vnewneur, neuron2) == true))
        {
                vsyn.n1 = neuron1;
                vsyn.n2 = neuron2;
                vsyn.wt = weight;
                postsyn.push_back(vsyn);
        }
        else if ((find(vnewneur, neuron1) == true) && (find(vnewneur, neuron2) == true))
```

```
		{
			vsyn.n1 = neuron1;
			vsyn.n2 = neuron2;
			vsyn.wt = weight;
			newconnection.push_back(vsyn);
		}
		else if ((find(vnewneur, neuron1) == false) && (find(vnewneur, neuron2) == false))
		{
			vsyn.n1 = neuron1;
			vsyn.n2 = neuron2;
			vsyn.wt = weight;
			oldconnection.push_back(vsyn);
		}
		}
}
outfile << stageedges;

if(stage > 1)
{
outfile << endl << endl << "Nature of new synapses: " << endl << endl;
outfile << "Pre-synaptic: " << presyn.size() << endl;
		for(int i = 0; i < presyn.size(); ++i)
			outfile << presyn[i].n1 << " " << presyn[i].n2 << " " << presyn[i].wt << ", ";
outfile << endl << "Post-synaptic: " << postsyn.size() << endl;
		for(int i = 0; i < postsyn.size(); ++i)
			outfile << postsyn[i].n1 << " " << postsyn[i].n2 << " " << postsyn[i].wt << ", ";
outfile << endl << "New connections: " << newconnection.size() << endl;
		for(int i = 0; i < newconnection.size(); ++i)
```

```
                                outfile << newconnection[i].n1 << " " << newconnection[i].n2 << " " <<
newconnection[i].wt << ", ";
        outfile << endl << endl;
        }
        ++stage;
}

outfile.close();

        return 0;
}
```