

**Relevant Code:**

set<string> friends(const graph<string>& network, const string& name)

{

set<string> persons;

size\_t i;

int source = 0;

for (i = 0; i < network.size(); i++)

{

if (network.operator[](i) == name)

{

source = i;

break;

}

}

for (i = 0; i < network.size(); ++i)

{

if (i == source)

{

continue;

}

else

{

if (network.is\_edge(source, i))

{

persons.insert(network.operator[](i));

}

}

}

return persons;

}

set<string> common\_friends(const graph<string>& network, const string& name1, const string& name2)

{

set<string> common;

set<string> friends1;

set<string> friends2;

friends1 = friends(network, name1);

friends2 = friends(network, name2);

for (set<string>::iterator iter = friends1.begin(); iter != friends1.end(); ++iter)

{

for (set<string>::iterator iter2 = friends2.begin(); iter2 != friends2.end(); ++iter2)

{

if (\*iter == \*iter2)

{

common.insert(\*iter);

break;

}

}

}

return common;

}

set<string> friends\_of\_friends(const graph<string>& network, const string& my\_name)

{

set<string> friendlies;

set<string> f\_o\_f;

set<string> fluff;

friendlies = friends(network, my\_name);

for (set<string>::iterator iter2 = friendlies.begin(); iter2 != friendlies.end(); ++iter2)

{

fluff = friends(network, \*iter2);

for (set<string>::iterator iter3 = fluff.begin(); iter3 != fluff.end(); ++iter3)

{

f\_o\_f.insert(\*iter3);

}

}

return f\_o\_f;

}

void add\_friend(graph<string>& network, const string& my\_name, const string& new\_friend)

{

int source = -1;

int source2 = -1;

for (int i = 0; i < network.size(); i++)

{

if (source >= 0 && source2 >= 0)

{

break;

}

if (network.operator[](i) == my\_name)

{

source = i;

}

if (network.operator[](i) == new\_friend)

{

source2 = i;

}

}

network.add\_edge(source, source2);

}

void remove\_friend(graph<string>& network, const string& my\_name, const string& ex\_friend)

{

int source = -1;

int source2 = -1;

for (int i = 0; i < network.size(); i++)

{

if (source >= 0 && source2 >= 0)

{

break;

}

if (network.operator[](i) == my\_name)

{

source = i;

}

if (network.operator[](i) == ex\_friend)

{

source2 = i;

}

}

network.remove\_edge(source, source2);

}

bool is\_friend(const graph<string>& network, const string& my\_name, const string& maybe\_friend)

{

int source = -1;

int source2 = -1;

for (int i = 0; i < network.size(); i++)

{

if (source >= 0 && source2 >= 0)

{

break;

}

if (network.operator[](i) == my\_name)

{

source = i;

}

if (network.operator[](i) == maybe\_friend)

{

source2 = i;

}

}

return network.is\_edge(source, source2);

}

**Main Function:**

int main()

{

graph<string> \* world = create\_graph("A7\_test.txt");

set<string> test;

test = friends(\*world, "Bob");

cout << "Bob's Friends:" << endl;

for (set<string>::iterator iter2 = test.begin(); iter2 != test.end(); ++iter2)

{

cout << \*iter2 << " ";

}

cout << endl;

test = friends\_of\_friends(\*world, "Barbara");

cout << "Friends of Barbara's Friends:" << endl;

for (set<string>::iterator iter2 = test.begin(); iter2 != test.end(); ++iter2)

{

cout << \*iter2 << " ";

}

cout << endl;

test = common\_friends(\*world, "Barbara", "Bob");

cout << "Common Friends of Barbara and Bob:" << endl;

for (set<string>::iterator iter2 = test.begin(); iter2 != test.end(); ++iter2)

{

cout << \*iter2 << " ";

}

cout << endl;

add\_friend(\*world, "Barbara", "Bob");

add\_friend(\*world, "Bob", "Barbara");

cout << "There seems to be a new relationship between Bob and Barbara. Are they friends?:" << endl;

if (is\_friend(\*world, "Bob", "Barbara") == true)

cout << "Yes" << endl;

else

cout << "No" << endl;

remove\_friend(\*world, "Steve", "Bob");

add\_friend(\*world, "Steve", "Sally");

test = friends(\*world, "Steve");

cout << "Steve doesn't seem to like how Bob is now friends with his ex, but he has found a new friend. Who may the be?:" << endl;

for (set<string>::iterator iter2 = test.begin(); iter2 != test.end(); ++iter2)

{

cout << \*iter2 << " ";

}

cout << endl;

system("PAUSE");

return 0;

}