# 計算機程式設計 HW11report

103061223 李俊穎

## 1. Introduction of the problem and your solution

The homework 11 is a “Small World Network with Information”. This time we need to add the student ID in the HW 10 code, and get the department from the ID to classify these students. Also, we have to find out the closest links between a student and another student neighbor which selected randomly and calculate how many trees do we need to find out all the students in the same department.

Because this time most of the codes have already been given, I just need to use a for loop to run all the students with a judge function. The following is my solution:

## 2. Implementation details ,Additional features of your program (data structure, flows and algorithms)

In the first step, I link all the students in random:

I link all the paths between two students:

```
//Randomly choose one pair of nodes and add links at a time
//first:generate all the possible ligatures in the ring
for(int i=1;i<n-2;i++) //start from node 1, link to node 3,node 4...(because the neighbor nodes have been linked),then change to nodes 2
{
    for(int j=i+2;j<n;j++)
    {
        path1 line;
        line.setpair_1(i);  line.setpair_2(j);  //store them into a class
        path.push_back(line); //store them into a vector
    }
}
for(int i=2;i<n-1;i++)  //start from node 0,link to node2,3...until n-1 nodes(node 1,node n have already been linked)
{
    path1 line;
    line.setpair_1(0);  line.setpair_2(i);
    path.push_back(line);
}
for(int i=0;i<100;i++)   //scatter sequence in the vector<path1> path
{
    a=rand()%path.size();
    b=rand()%path.size();
    while(a==b)   //if choose the same element, do it again.
    {
        a=rand()%path.size();
        b=rand()%path.size();
    }
    path1 temp;
    temp=path[a];
    path[a]=path[b];
    path[b]=temp;
}
for(int i=0;i<all_students.size()*0.5;i++) //take the path nodes out,and point them into the all_nodes to set the neighbor
{
    int c,d;
    c=path[i].getpair_1();
    d=path[i].getpair_2();
    all_students[c].AddNeighbor(&(all_students[d]));
    all_students[d].AddNeighbor(&(all_students[c]));
    //search the farhest node's distance from the all_nodes[0](original point)
}
```

After producing all the links, store them into the vector with class type, and randomize them.

What this step does is add one pair of students, and in this part I print all the students' ID:

```
for(int i=0;i<all_students.size();i++)  //print the Student ID and their neighbor StudentID
{
    cout<<"the student "<<all_students[i].getStudentID()<<" has the neighbor";
    for(int j=0;j<all_students[i].{ vector<Student> all_students(n) }
    {
        cout<<static_cast<Student*>(all_students[i].GetNeighbor(j))->getStudentID()<<" ";
    }
    cout<<endl;
}
```

Second, I ask the user to choose which department do they want to search, and then use a for loop to scan all the students. If the student's department is correct, the FindPathLengthsFromNode() will generate a tree from that student, and then tree++.

```
cout<<"which department would you like to search:"<<endl;
cin>>department;
for(int i=0;i<all_students.size();i++)//count trees for the department which is entered
{
    if(all_students[i].getDepartment()==department && all_students[i].getLevel()==0)
    {
        FindPathLengthsFromNode(all_students,i);
        tree++;
    }
}
```

From the above we can calculate how many trees do we need to find out all the students in the same department.

The following is the function FindPathLengthsFromNode(), I pass the vector all_students by reference, so the data can be rewrite back to the class.

```
void FindPathLengthsFromNode(vector<Student> &all_students, int i){
    vector<Node*> currentShell; //Nodes to be processed now
    vector<Node*> nextShell; //Store nodes to be processed next
    int distance=0;
    all_students[i].setLevel(1);
    currentShell.push_back(&(all_students[i])); //Starting node
    while(currentShell.size()!=0){
        nextShell.clear();
        distance++;
        for(int i=0;i<currentShell.size();i++)//each node in currentShell
        {
            for(int j=0;j<currentShell[i]->getSize();j++)//each neighbor of node
            {
                if(currentShell[i]->GetNeighbor(j)->getID()!=0&&currentShell[i]->GetNeighbor(j)->getLevel()==0)
                {   //level of neighbor is not assigned
                    nextShell.push_back(currentShell[i]->GetNeighbor(j));
                    currentShell[i]->GetNeighbor(j)->setLevel(distance);
                }
            }
        }
        currentShell=nextShell;
    }
}
```

The extra thing I add is setting the start student's level as 1 before the while loop, so every time I can check if this level of neighbor be assigned. If so, set the level with distance. By the way, because I set the original student's level be 1, so in the

following codes:

```
    for(int i=0;i<all_students.size();i++) //reset all the nodes have initial level=0
    {
        for(int j=0;j<all_students[i].getSize();j++)
        {
            all_students[i].GetNeighbor(j)->setLevel(0);
        }
    }
    cout<<"enter two students to find closet link:"<<endl;
    cin>>Student1>>Student2;
    FindPathLengthsFromNode(all_students,Student1);// use student1 to generate a tree
    if(all_students[Student2].getLevel()==0) //if the tree generates from student 1 doesn't include student 2,then
    {
        cout<<"there in the different tree:"<<endl;
    }
    else  //if they are in the same tree, to substract their level substract is their closet distance
    {
        cout<<"there distance is "<<all_students[Student2].getLevel()-all_students[Student1].getLevel()+1<<endl;
        //because I set the start student level=1 previously,so when the function FindPathLengthsFromNode
        //executes, the student1's level will be 1,however, it's actually level 0.That's why I have to add 1 after
    }
    return 0;
}
```

When I cout the closet distance, I have to add one back because the real level for the tree's original student is level 0. If I just substract their level, the distance will reduce one. Then the result will be wrong.