計算機程式設計 HW10report

103061223 李俊穎

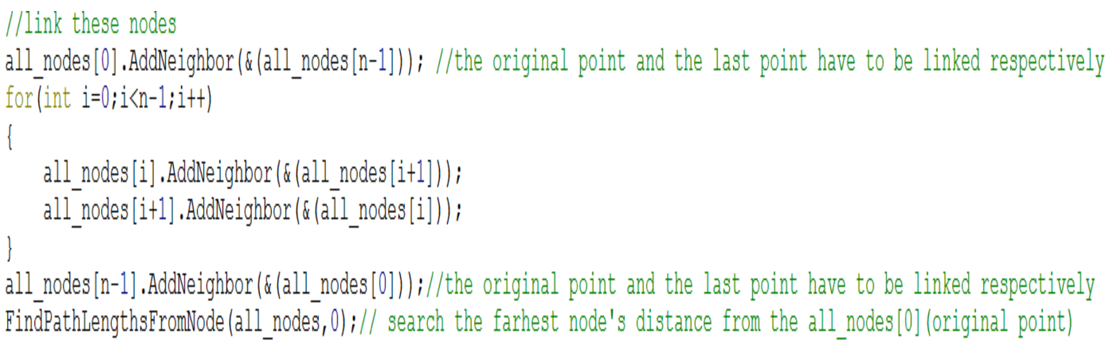
1. Introduction of the problem and your solution

The homework 10 is a “Small World Network” that we have to investigate the number of links to reach neighbor nodes which are connected randomly. First, we have to generate how many nodes we have in this program. The nodes just like the countries or villages, and they all linked by the road each sides. This homework’s problem is that when these nodes are connected in a ring, how we to calculate the distances from a start node to the farthest node.

Because this time most of the codes have already been given, we just need to translate the hints in the function which given by instructor. Besides, we need to produce the one pair of nodes and add it into the ring at a time. My solution is consider all the condition pairs of nodes. After generating them, I store them into a class, and then push_back the class in a vector named path. So I can add the pair of nodes randomly in the all_nodes neighbor until it finish the for loop. The following is my code:

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

In the first step, I link all the nodes to be a ring:



```
//link these nodes
all_nodes[0].AddNeighbor(&(all_nodes[n-1])); //the original point and the last point have to be linked respectively
for(int i=0;i<n-1;i++)
{
    all_nodes[i].AddNeighbor(&(all_nodes[i+1]));
    all_nodes[i+1].AddNeighbor(&(all_nodes[i]));
}
all_nodes[n-1].AddNeighbor(&(all_nodes[0]));//the original point and the last point have to be linked respectively
FindPathLengthsFromNode(all_nodes,0);// search the farhest node's distance from the all_nodes[0](original point)
```

The original point and last point have to link respectively because if we put them in the for loop, the vector of the all_nodes will overflow.
Next, I link all the paths between two nodes in the ring:

```
//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
{
    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;
}
```

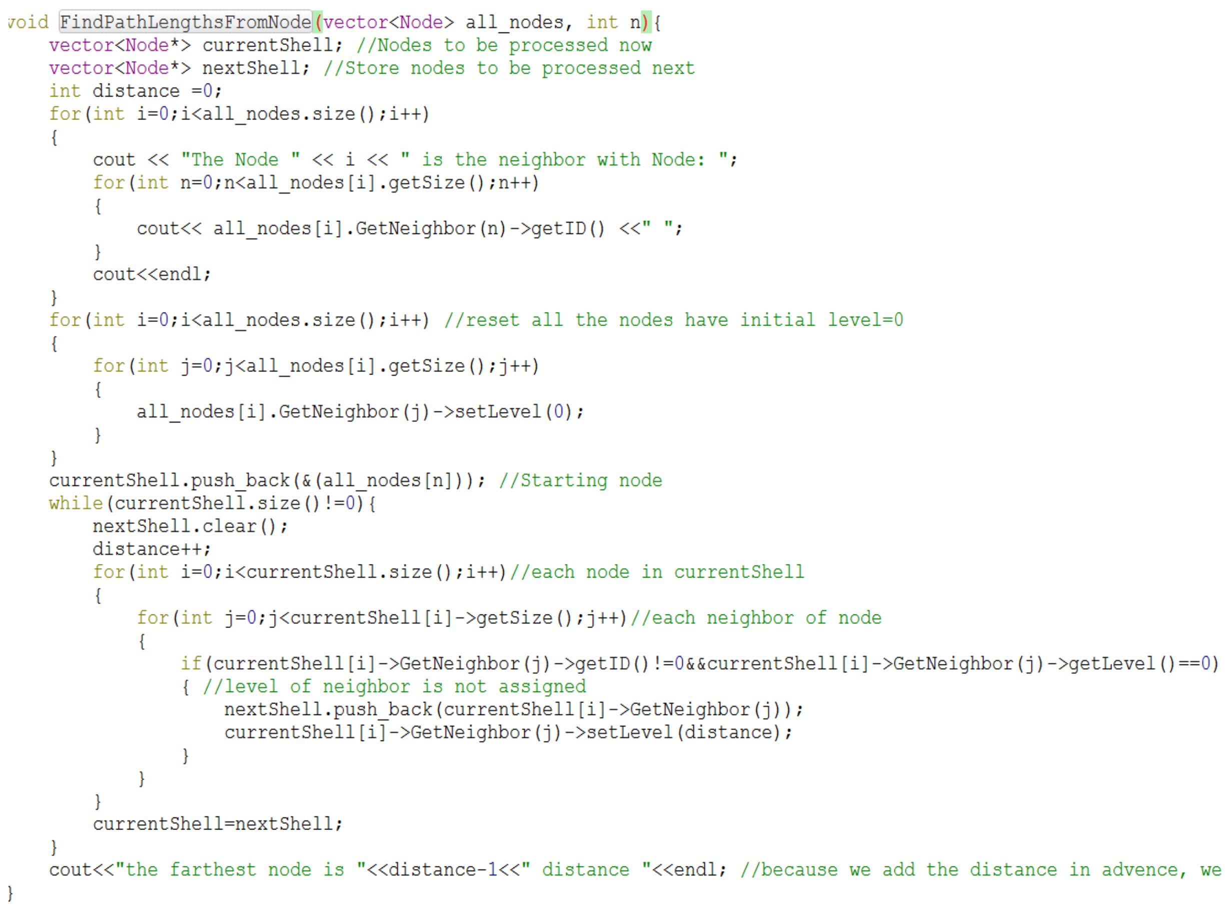
After producing all the paths, store them into the vector with class type, and randomize them.

Third step is as following:

```
for(int i=0;i<path.size();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_nodes[c].AddNeighbor(&(all_nodes[d]));
    all_nodes[d].AddNeighbor(&(all_nodes[c]));
    FindPathLengthsFromNode(all_nodes,0);//search the farhest node's distance from the all_nodes[0](original point)
}
return 0;
```

What this step does is add one pair of nodes at a time, and find the farthest path from nodes 0. After running the path.size() times, all the nodes have been linked with each other, so the farthest distance is 1.

The following is the function FindPathLengthsFromNode():



```cpp
void FindPathLengthsFromNode(vector<Node> all_nodes, int n){
    vector<Node*> currentShell; //Nodes to be processed now
    vector<Node*> nextShell; //Store nodes to be processed next
    int distance =0;
    for(int i=0;i<all_nodes.size();i++)
    {
        cout << "The Node " << i << " is the neighbor with Node: ";
        for(int n=0;n<all_nodes[i].getSize();n++)
        {
            cout<< all_nodes[i].GetNeighbor(n)->getID() <<" ";
        }
        cout<<endl;
    }
    for(int i=0;i<all_nodes.size();i++) //reset all the nodes have initial level=0
    {
        for(int j=0;j<all_nodes[i].getSize();j++)
        {
            all_nodes[i].GetNeighbor(j)->setLevel(0);
        }
    }
    currentShell.push_back(&(all_nodes[n])); //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;
    }
    cout<<"the farthest node is "<<distance-1<<" distance "<<endl; //because we add the distance in advence, we
}
```

The extra thing I add is seting all the nodes' level as 0 before the while loop, so every time I can check if this level of neighbor be assigned. If so, set the level with distance and output it, that's all.