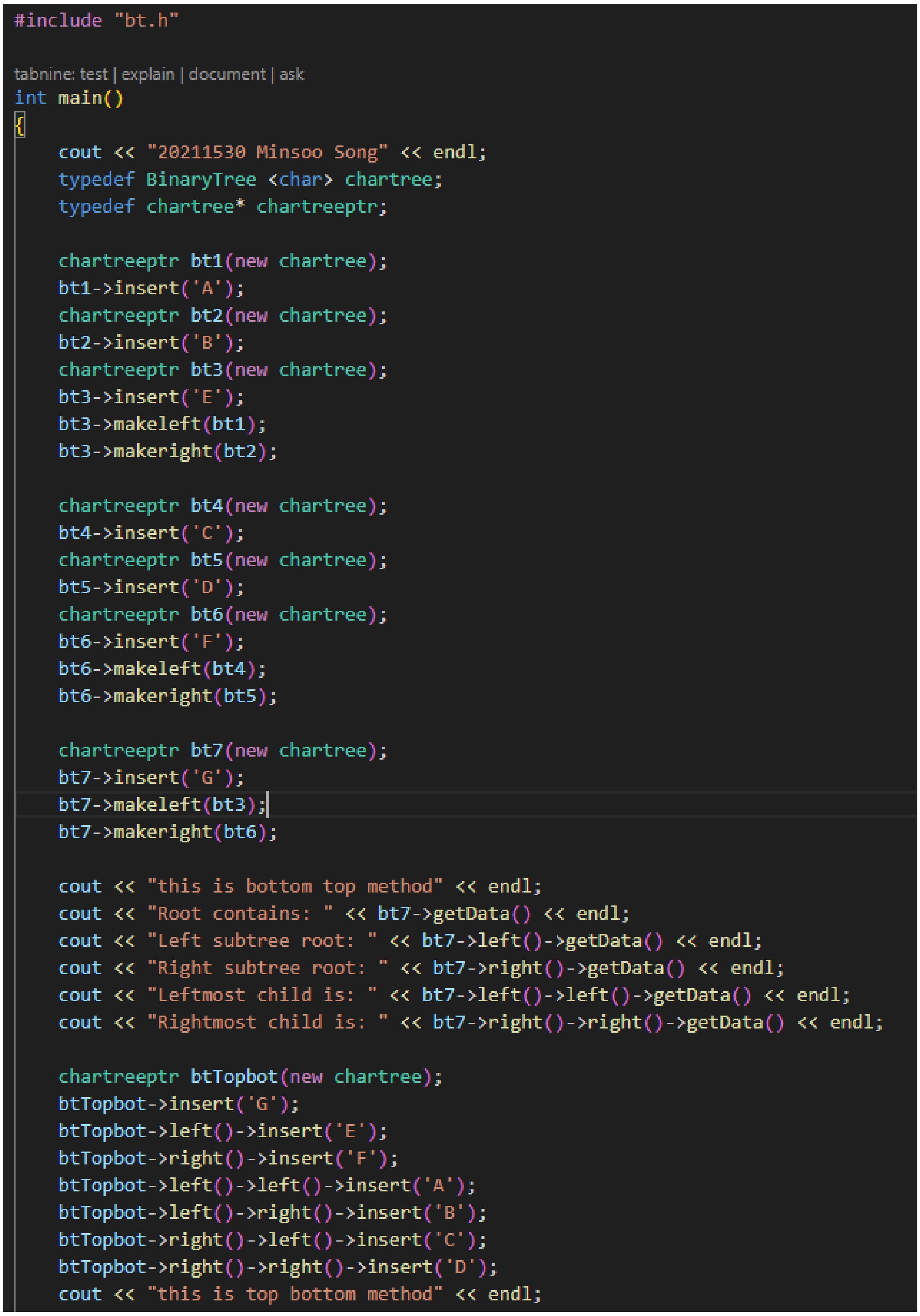
20211530 송민수

tree –24,40

graph –47,61

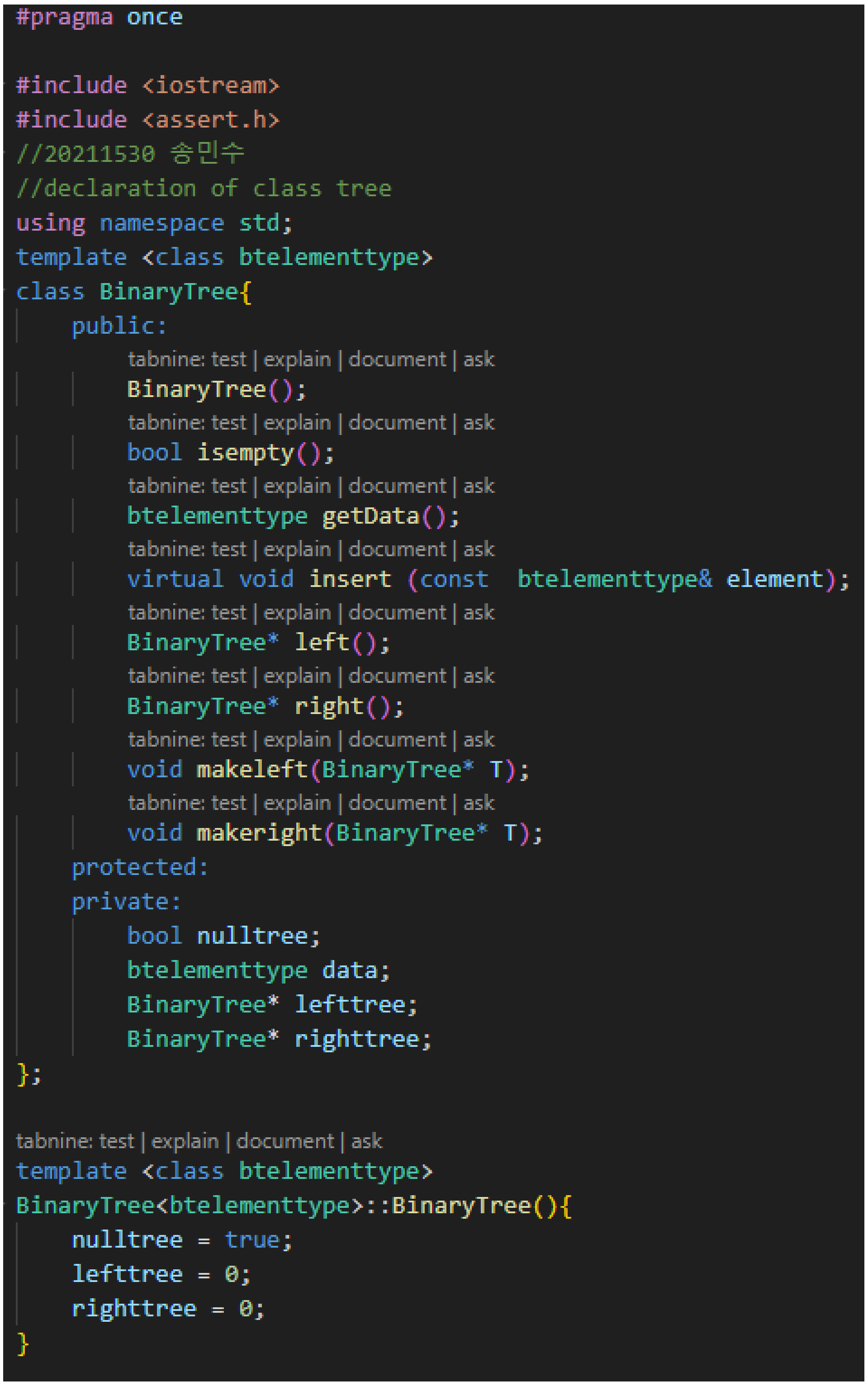
Tree 24

Main

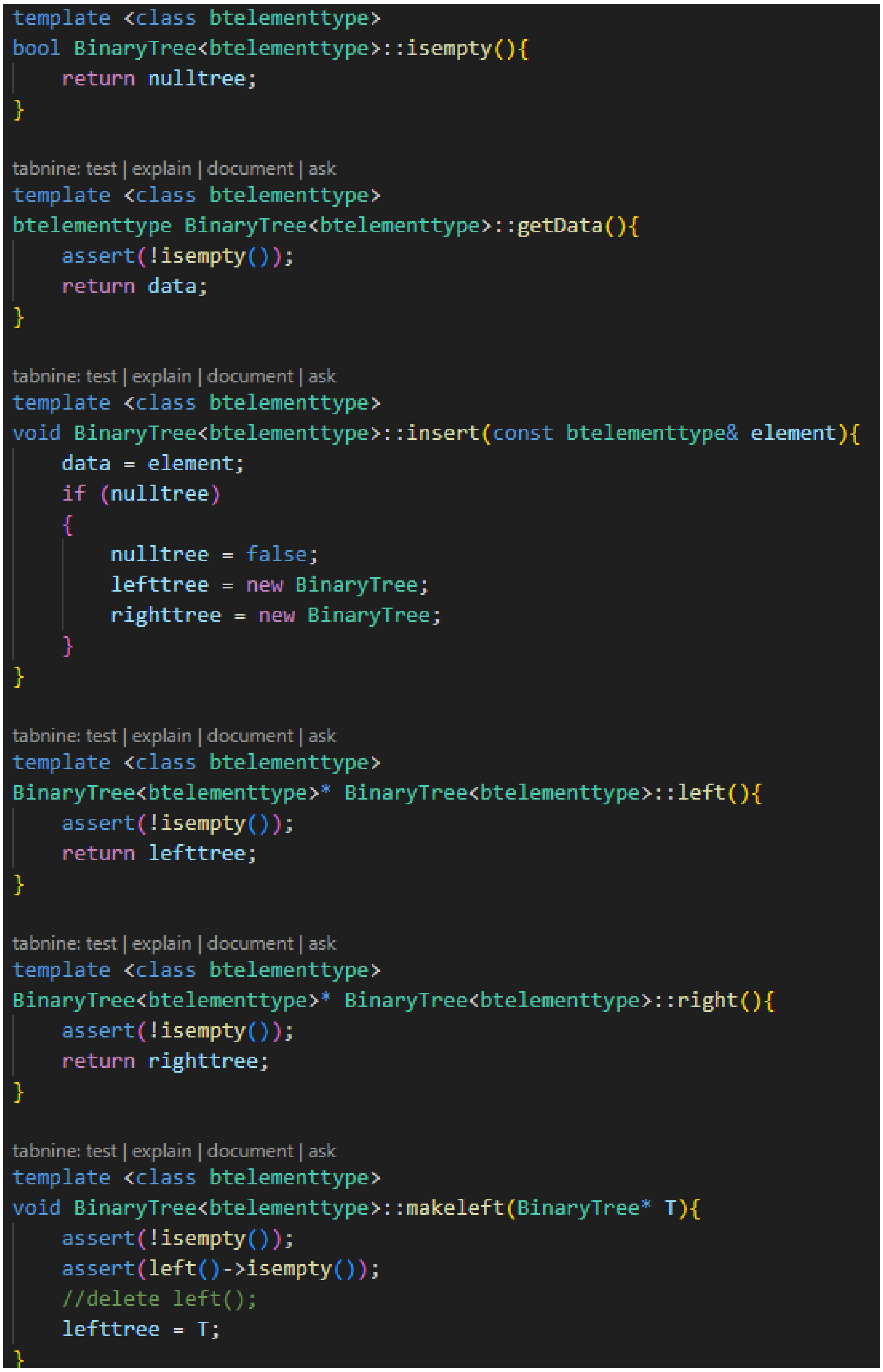


```
#include "bt.h"

tabnine: test | explain | document | ask
int main()
{
    cout << "20211530 Minsoo Song" << endl;
    typedef BinaryTree <char> chartree;
    typedef chartree* chartreeptr;

    chartreeptr bt1(new chartree);
    bt1->insert('A');
    chartreeptr bt2(new chartree);
    bt2->insert('B');
    chartreeptr bt3(new chartree);
    bt3->insert('E');
    bt3->makeleft(bt1);
    bt3->makeright(bt2);

    chartreeptr bt4(new chartree);
    bt4->insert('C');
    chartreeptr bt5(new chartree);
    bt5->insert('D');
    chartreeptr bt6(new chartree);
    bt6->insert('F');
    bt6->makeleft(bt4);
    bt6->makeright(bt5);

    chartreeptr bt7(new chartree);
    bt7->insert('G');
    bt7->makeleft(bt3);
    bt7->makeright(bt6);

    cout << "this is bottom top method" << endl;
    cout << "Root contains: " << bt7->getData() << endl;
    cout << "Left subtree root: " << bt7->left()->getData() << endl;
    cout << "Right subtree root: " << bt7->right()->getData() << endl;
    cout << "Leftmost child is: " << bt7->left()->left()->getData() << endl;
    cout << "Rightmost child is: " << bt7->right()->right()->getData() << endl;

    chartreeptr btTopbot(new chartree);
    btTopbot->insert('G');
    btTopbot->left()->insert('E');
    btTopbot->right()->insert('F');
    btTopbot->left()->left()->insert('A');
    btTopbot->left()->right()->insert('B');
    btTopbot->right()->left()->insert('C');
    btTopbot->right()->right()->insert('D');
    cout << "this is top bottom method" << endl;
```

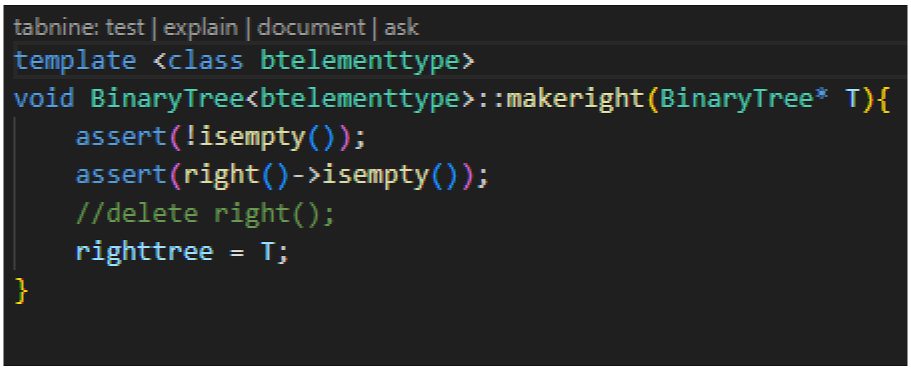
Bt.h



```cpp
#pragma once

#include <iostream>
#include <assert.h>
//20211530 송민수
//declaration of class tree
using namespace std;
template <class btelementtype>
class BinaryTree{
    public:
        tabnine: test | explain | document | ask
        BinaryTree();
        tabnine: test | explain | document | ask
        bool isempty();
        tabnine: test | explain | document | ask
        btelementtype getData();
        tabnine: test | explain | document | ask
        virtual void insert (const  btelementtype& element);
        tabnine: test | explain | document | ask
        BinaryTree* left();
        tabnine: test | explain | document | ask
        BinaryTree* right();
        tabnine: test | explain | document | ask
        void makeleft(BinaryTree* T);
        tabnine: test | explain | document | ask
        void makeright(BinaryTree* T);
    protected:
    private:
        bool nulltree;
        btelementtype data;
        BinaryTree* lefttree;
        BinaryTree* righttree;
};

tabnine: test | explain | document | ask
template <class btelementtype>
BinaryTree<btelementtype>::BinaryTree(){
    nulltree = true;
    lefttree = 0;
    righttree = 0;
}
```

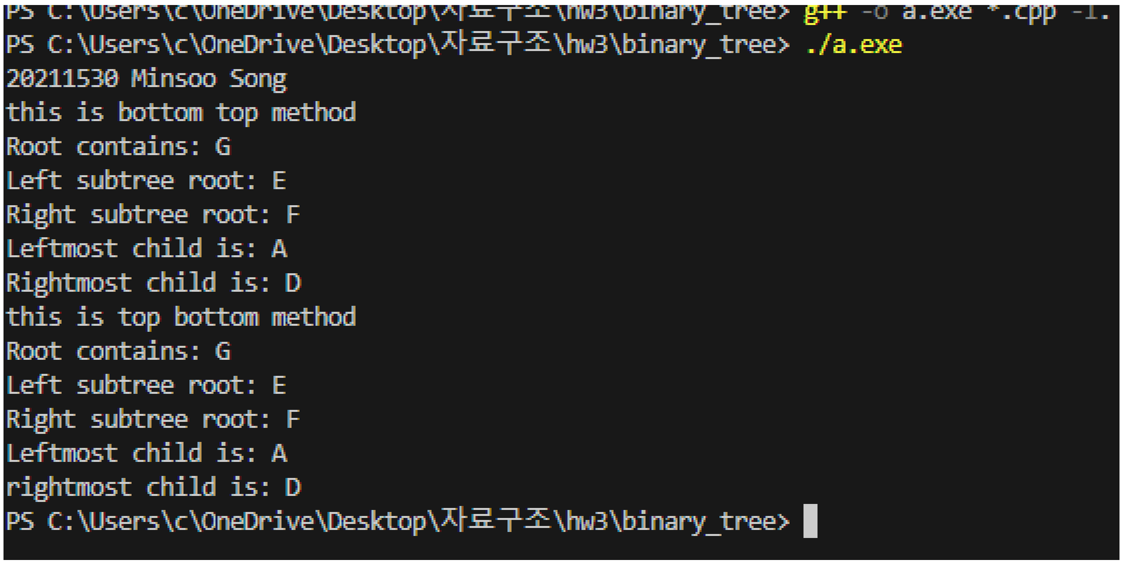


```
template <class btelementtype>
bool BinaryTree<btelementtype>::isempty(){
    return nulltree;
}

tabnine: test | explain | document | ask
template <class btelementtype>
btelementtype BinaryTree<btelementtype>::getData(){
    assert(!isempty());
    return data;
}

tabnine: test | explain | document | ask
template <class btelementtype>
void BinaryTree<btelementtype>::insert(const btelementtype& element){
    data = element;
    if (nulltree)
    {
        nulltree = false;
        lefttree = new BinaryTree;
        righttree = new BinaryTree;
    }
}

tabnine: test | explain | document | ask
template <class btelementtype>
BinaryTree<btelementtype>* BinaryTree<btelementtype>::left(){
    assert(!isempty());
    return lefttree;
}

tabnine: test | explain | document | ask
template <class btelementtype>
BinaryTree<btelementtype>* BinaryTree<btelementtype>::right(){
    assert(!isempty());
    return righttree;
}

tabnine: test | explain | document | ask
template <class btelementtype>
void BinaryTree<btelementtype>::makeleft(BinaryTree* T){
    assert(!isempty());
    assert(left()->isempty());
    //delete left();
    lefttree = T;
}
```



```
tabnine: test | explain | document | ask
template <class btelementtype>
void BinaryTree<btelementtype>::makeright(BinaryTree* T){
    assert(!isempty());
    assert(right()->isempty());
    //delete right();
    righttree = T;
}
```

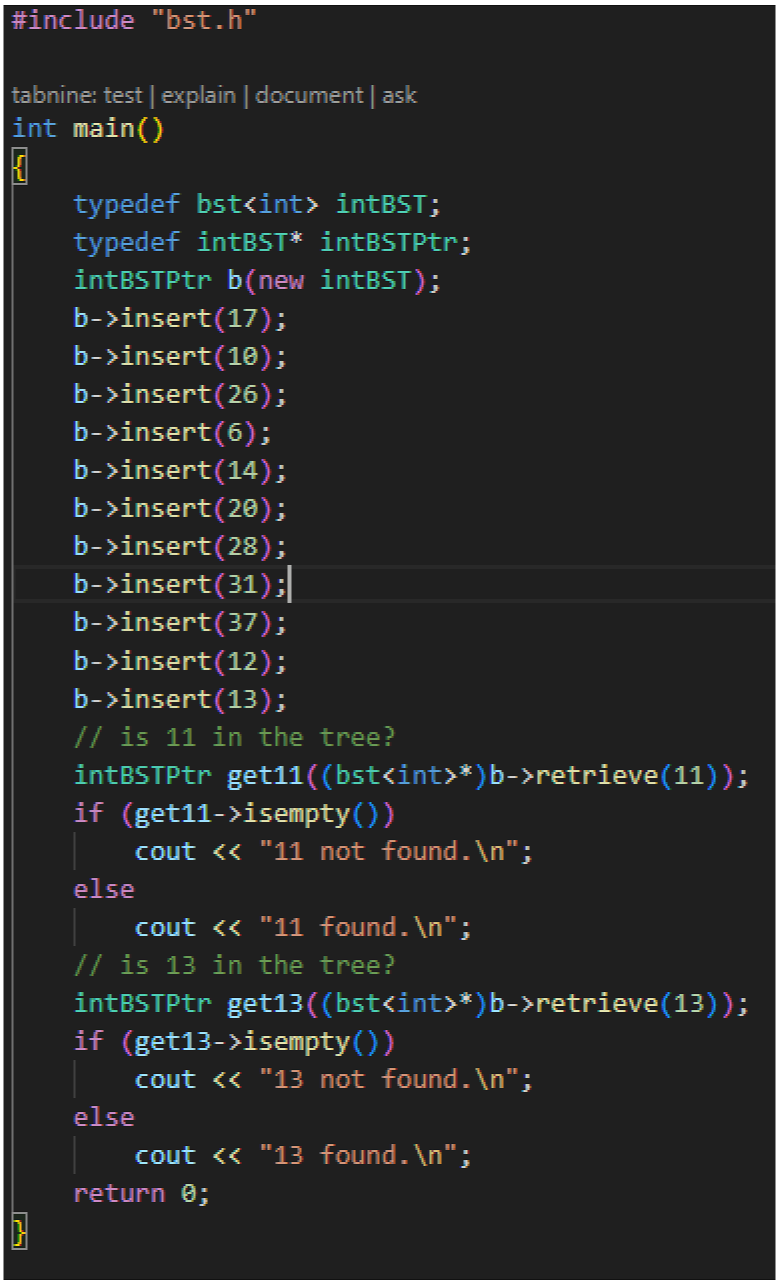
Output



```
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\binary_tree> g++ -o a.exe *.cpp -I.
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\binary_tree> ./a.exe
20211530 Minsoo Song
this is bottom top method
Root contains: G
Left subtree root: E
Right subtree root: F
Leftmost child is: A
Rightmost child is: D
this is top bottom method
Root contains: G
Left subtree root: E
Right subtree root: F
Leftmost child is: A
rightmost child is: D
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\binary_tree>
```

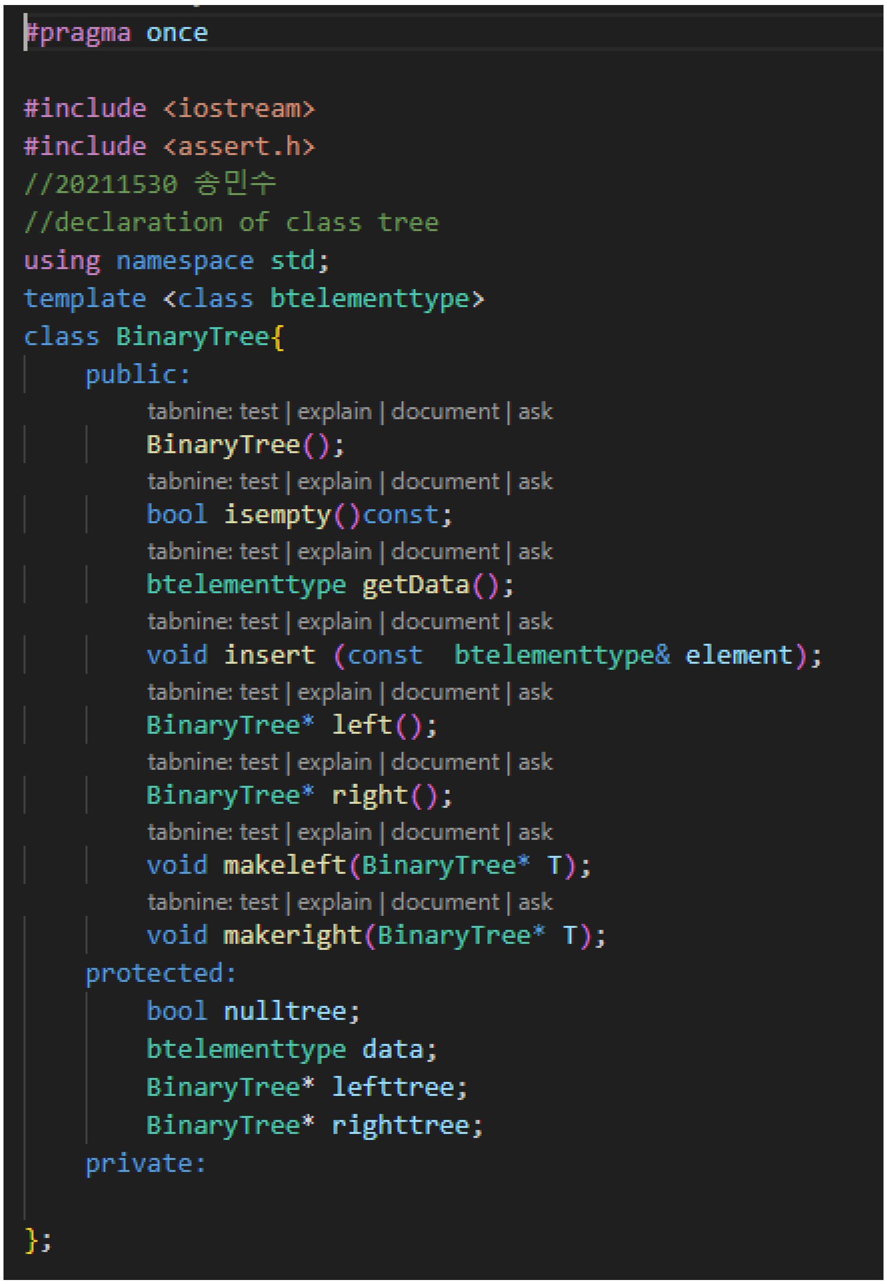
Tree 40

Main



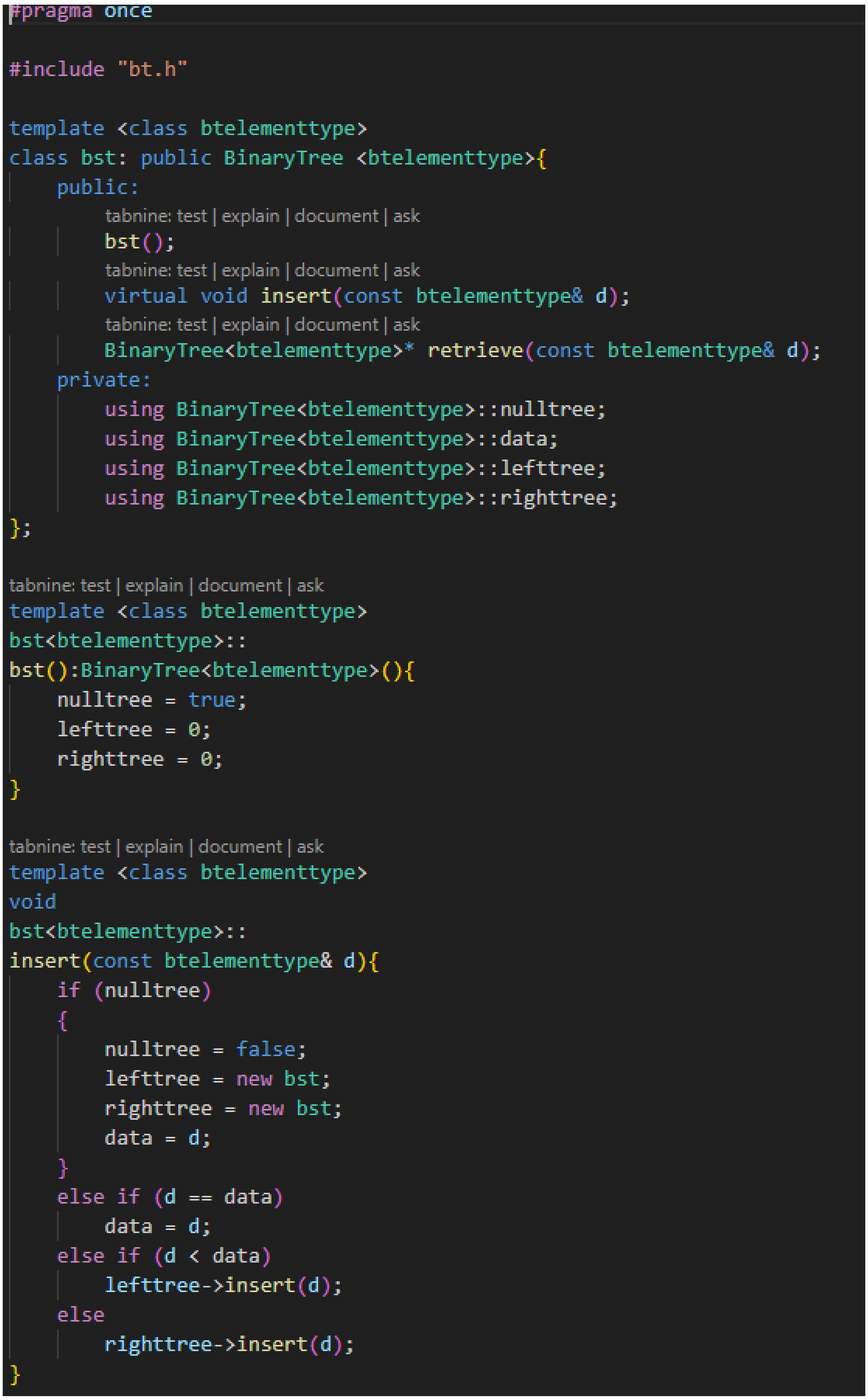
```
#include "bst.h"

tabnine: test | explain | document | ask
int main()
{
    typedef bst<int> intBST;
    typedef intBST* intBSTPtr;
    intBSTPtr b(new intBST);
    b->insert(17);
    b->insert(10);
    b->insert(26);
    b->insert(6);
    b->insert(14);
    b->insert(20);
    b->insert(28);
    b->insert(31);
    b->insert(37);
    b->insert(12);
    b->insert(13);
    // is 11 in the tree?
    intBSTPtr get11((bst<int>*)b->retrieve(11));
    if (get11->isempty())
        cout << "11 not found.\n";
    else
        cout << "11 found.\n";
    // is 13 in the tree?
    intBSTPtr get13((bst<int>*)b->retrieve(13));
    if (get13->isempty())
        cout << "13 not found.\n";
    else
        cout << "13 found.\n";
    return 0;
}
```

Bt.h

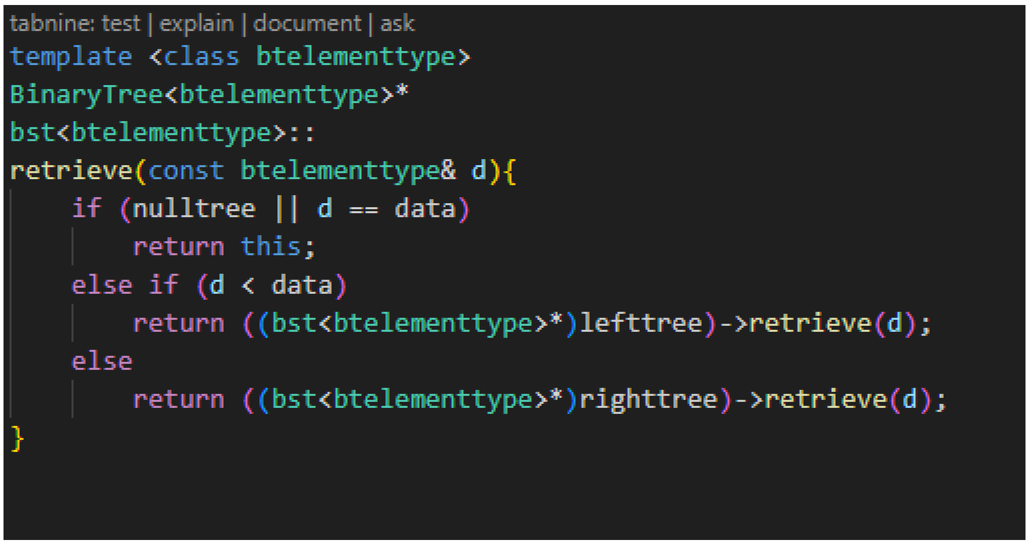


```cpp
#pragma once

#include <iostream>
#include <assert.h>
//20211530 송민수
//declaration of class tree
using namespace std;
template <class btelementtype>
class BinaryTree{
    public:
        tabnine: test | explain | document | ask
        BinaryTree();
        tabnine: test | explain | document | ask
        bool isempty()const;
        tabnine: test | explain | document | ask
        btelementtype getData();
        tabnine: test | explain | document | ask
        void insert (const  btelementtype& element);
        tabnine: test | explain | document | ask
        BinaryTree* left();
        tabnine: test | explain | document | ask
        BinaryTree* right();
        tabnine: test | explain | document | ask
        void makeleft(BinaryTree* T);
        tabnine: test | explain | document | ask
        void makeright(BinaryTree* T);
    protected:
        bool nulltree;
        btelementtype data;
        BinaryTree* lefttree;
        BinaryTree* righttree;
    private:

};
```

Bst.h

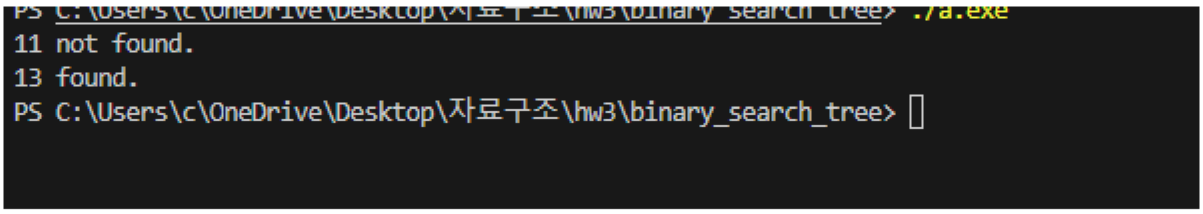


```
#pragma once

#include "bt.h"

template <class btelementtype>
class bst: public BinaryTree <btelementtype>{
    public:
        tabnine: test | explain | document | ask
        bst();
        tabnine: test | explain | document | ask
        virtual void insert(const btelementtype& d);
        tabnine: test | explain | document | ask
        BinaryTree<btelementtype>* retrieve(const btelementtype& d);
    private:
        using BinaryTree<btelementtype>::nulltree;
        using BinaryTree<btelementtype>::data;
        using BinaryTree<btelementtype>::lefttree;
        using BinaryTree<btelementtype>::righttree;
};

tabnine: test | explain | document | ask
template <class btelementtype>
bst<btelementtype>::
bst():BinaryTree<btelementtype>(){
    nulltree = true;
    lefttree = 0;
    righttree = 0;
}

tabnine: test | explain | document | ask
template <class btelementtype>
void
bst<btelementtype>::
insert(const btelementtype& d){
    if (nulltree)
    {
        nulltree = false;
        lefttree = new bst;
        righttree = new bst;
        data = d;
    }
    else if (d == data)
        data = d;
    else if (d < data)
        lefttree->insert(d);
    else
        righttree->insert(d);
}
```



```
tabnine: test | explain | document | ask
template <class btelementtype>
BinaryTree<btelementtype>*
bst<btelementtype>::
retrieve(const btelementtype& d){
    if (nulltree || d == data)
        return this;
    else if (d < data)
        return ((bst<btelementtype>*)lefttree)->retrieve(d);
    else
        return ((bst<btelementtype>*)righttree)->retrieve(d);
}
```

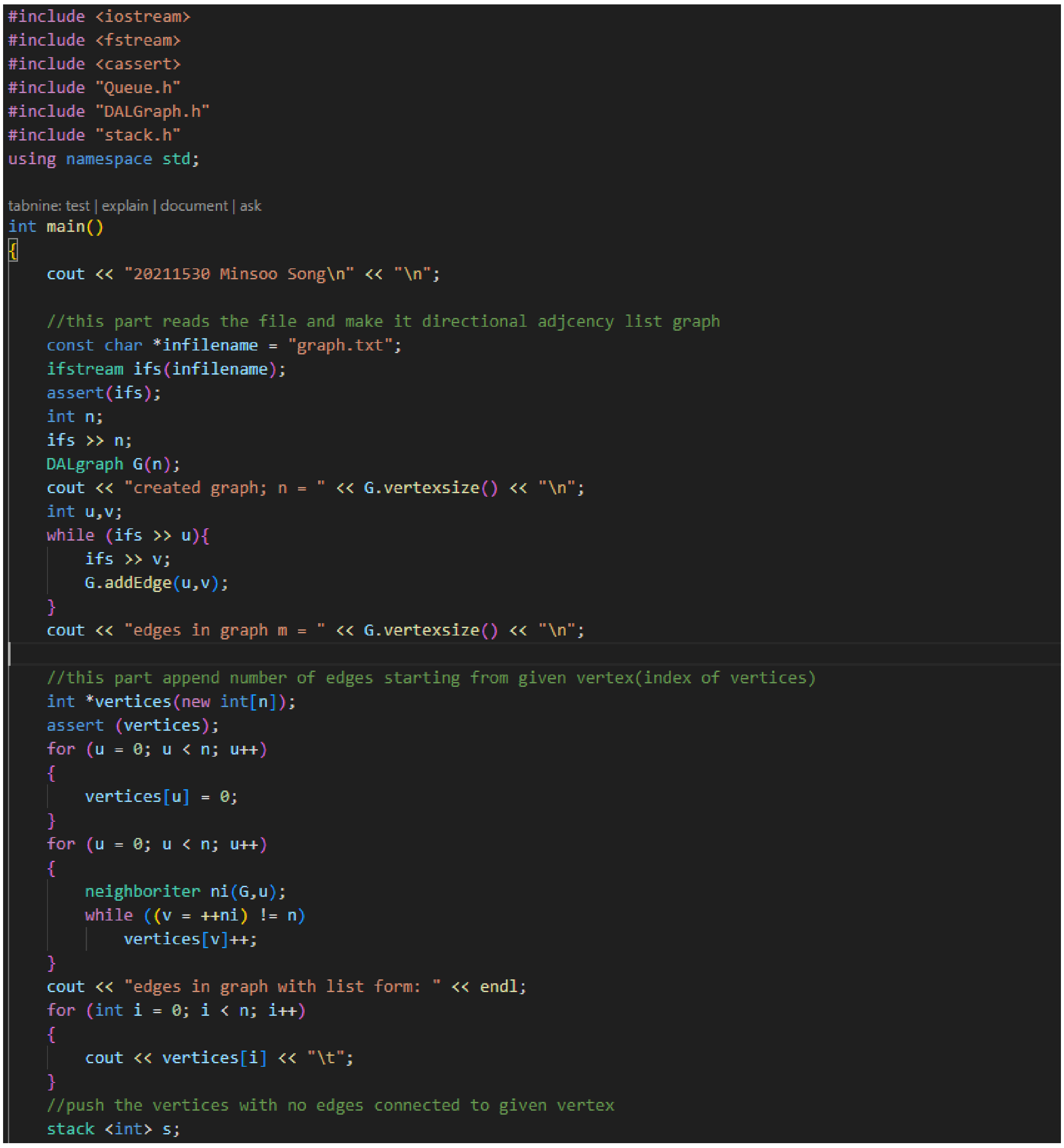
Output

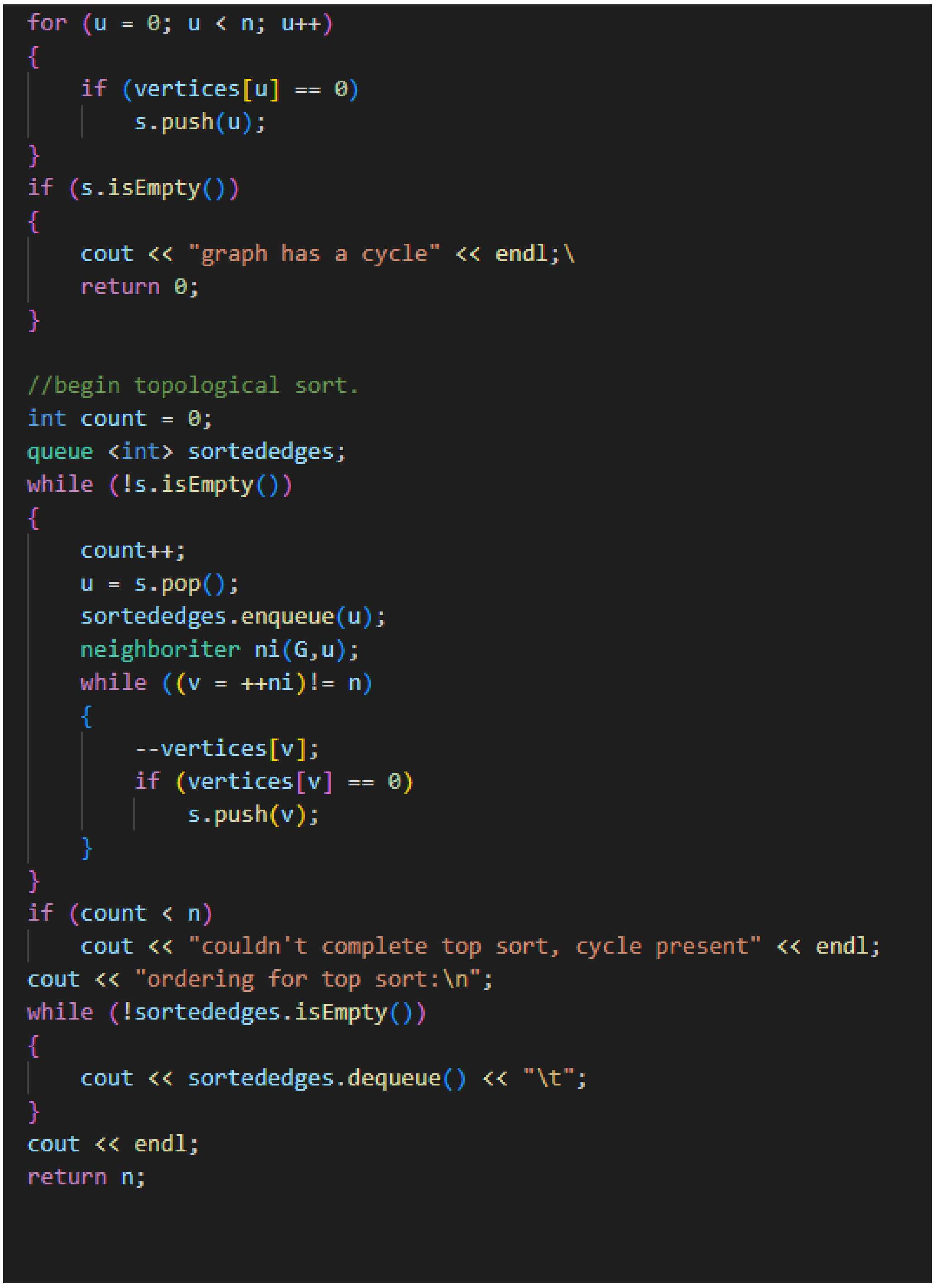


```
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\binary_search_tree> ./a.exe
11 not found.
13 found.
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\binary_search_tree> 
```

Graph 47 topological sort

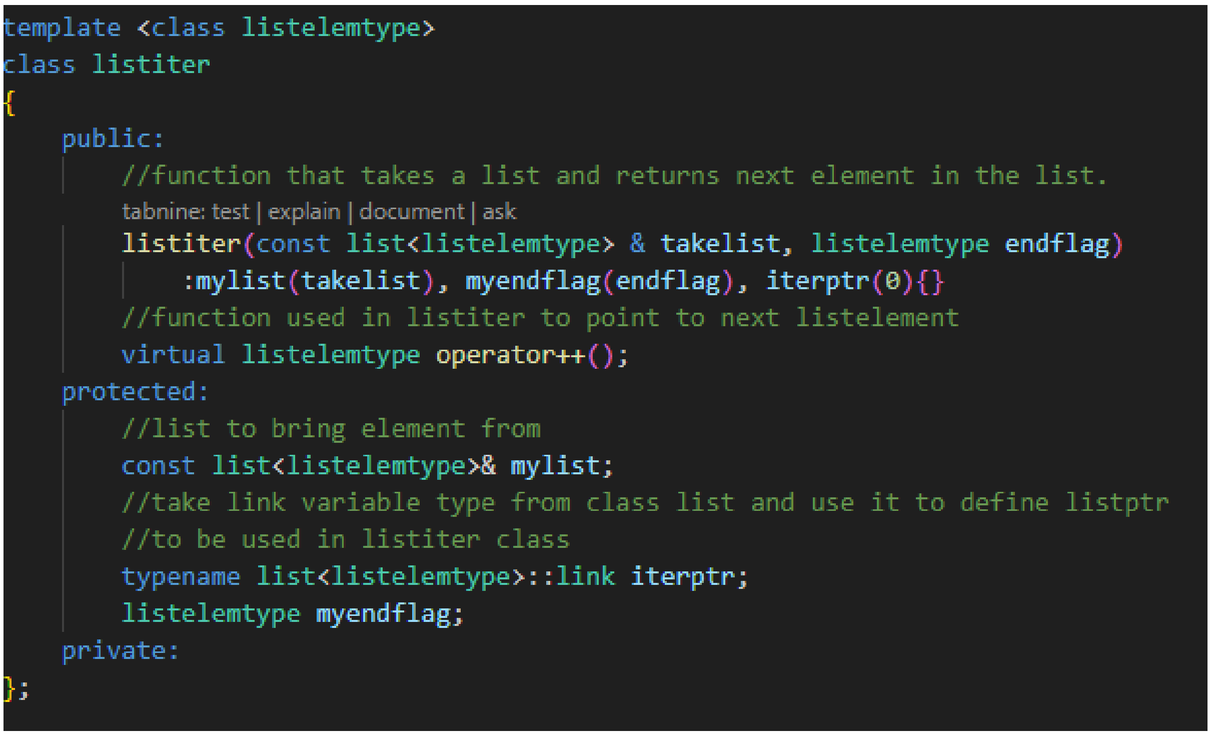
Main

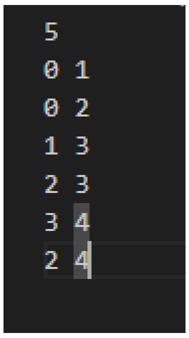

```
#include <iostream>
#include <fstream>
#include <cassert>
#include "Queue.h"
#include "DALGraph.h"
#include "stack.h"
using namespace std;

tabnine: test | explain | document | ask
int main()
{
    cout << "20211530 Minsoo Song\n" << "\n";

    //this part reads the file and make it directional adjcency list graph
    const char *infilename = "graph.txt";
    ifstream ifs(infilename);
    assert(ifs);
    int n;
    ifs >> n;
    DALgraph G(n);
    cout << "created graph; n = " << G.vertexsize() << "\n";
    int u,v;
    while (ifs >> u){
        ifs >> v;
        G.addEdge(u,v);
    }
    cout << "edges in graph m = " << G.vertexsize() << "\n";

    //this part append number of edges starting from given vertex(index of vertices)
    int *vertices(new int[n]);
    assert (vertices);
    for (u = 0; u < n; u++)
    {
        vertices[u] = 0;
    }
    for (u = 0; u < n; u++)
    {
        neighboriter ni(G,u);
        while ((v = ++ni) != n)
            vertices[v]++;
    }
    cout << "edges in graph with list form: " << endl;
    for (int i = 0; i < n; i++)
    {
        cout << vertices[i] << "\t";
    }
    //push the vertices with no edges connected to given vertex
    stack <int> s;
```



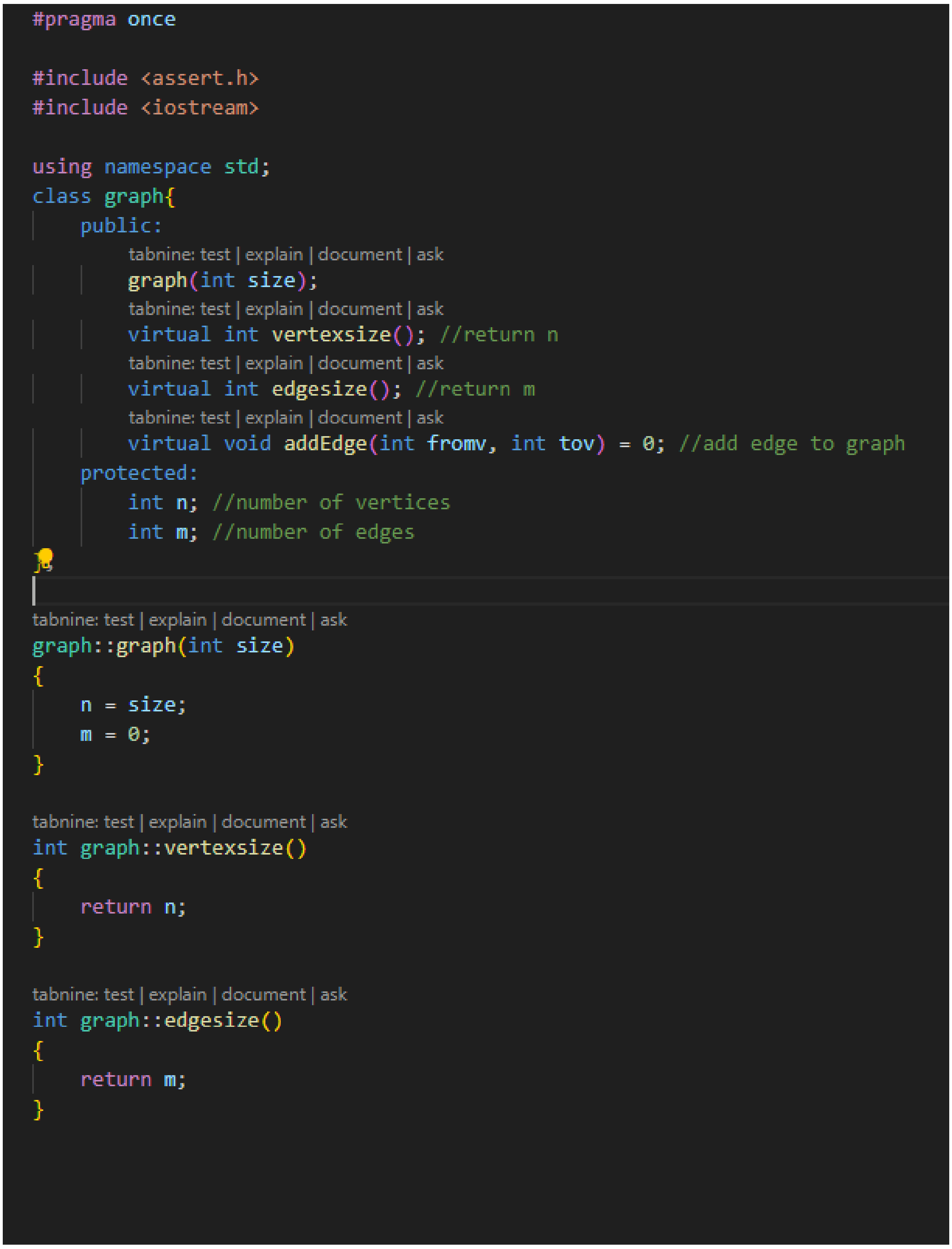
```
for (u = 0; u < n; u++)
{
    if (vertices[u] == 0)
        s.push(u);
}
if (s.isEmpty())
{
    cout << "graph has a cycle" << endl;\
    return 0;
}

//begin topological sort.
int count = 0;
queue <int> sortededges;
while (!s.isEmpty())
{
    count++;
    u = s.pop();
    sortededges.enqueue(u);
    neighboriter ni(G,u);
    while ((v = ++ni)!= n)
    {
        --vertices[v];
        if (vertices[v] == 0)
            s.push(v);
    }
}
if (count < n)
    cout << "couldn't complete top sort, cycle present" << endl;
cout << "ordering for top sort:\n";
while (!sortededges.isEmpty())
{
    cout << sortededges.dequeue() << "\t";
}
cout << endl;
return n;
```

List.h


```cpp
template <class listelemtype>
class listiter
{
    public:
        //function that takes a list and returns next element in the list.
        tabnine: test | explain | document | ask
        listiter(const list<listelemtype> & takelist, listelemtype endflag)
            :mylist(takelist), myendflag(endflag), iterptr(0){}
        //function used in listiter to point to next listelement
        virtual listelemtype operator++();
    protected:
        //list to bring element from
        const list<listelemtype>& mylist;
        //take link variable type from class list and use it to define listptr
        //to be used in listiter class
        typename list<listelemtype>::link iterptr;
        listelemtype myendflag;
    private:
};
```
Graph.txt

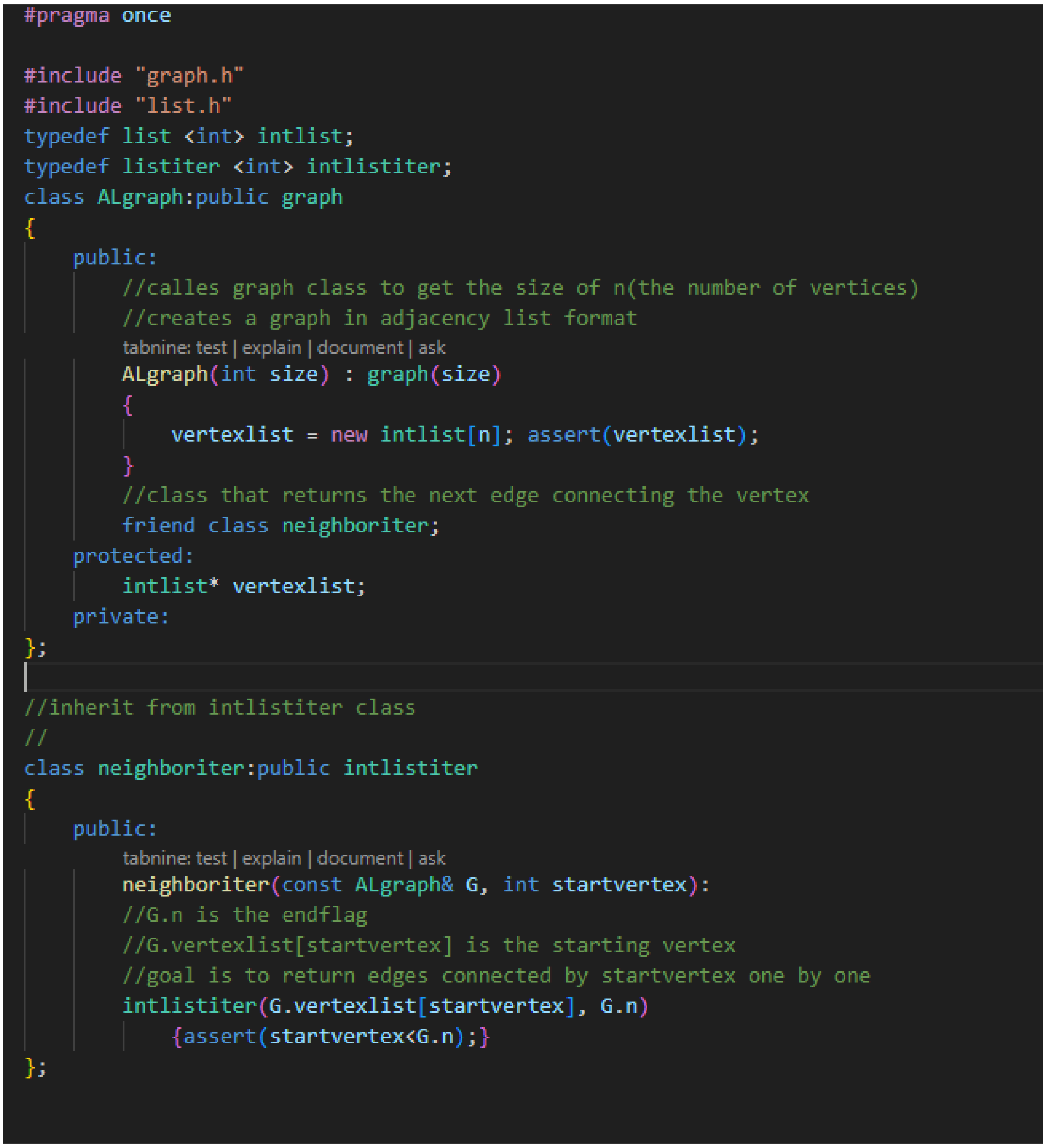

```
5
0 1
0 2
1 3
2 3
3 4
2 4
```

Graph.h

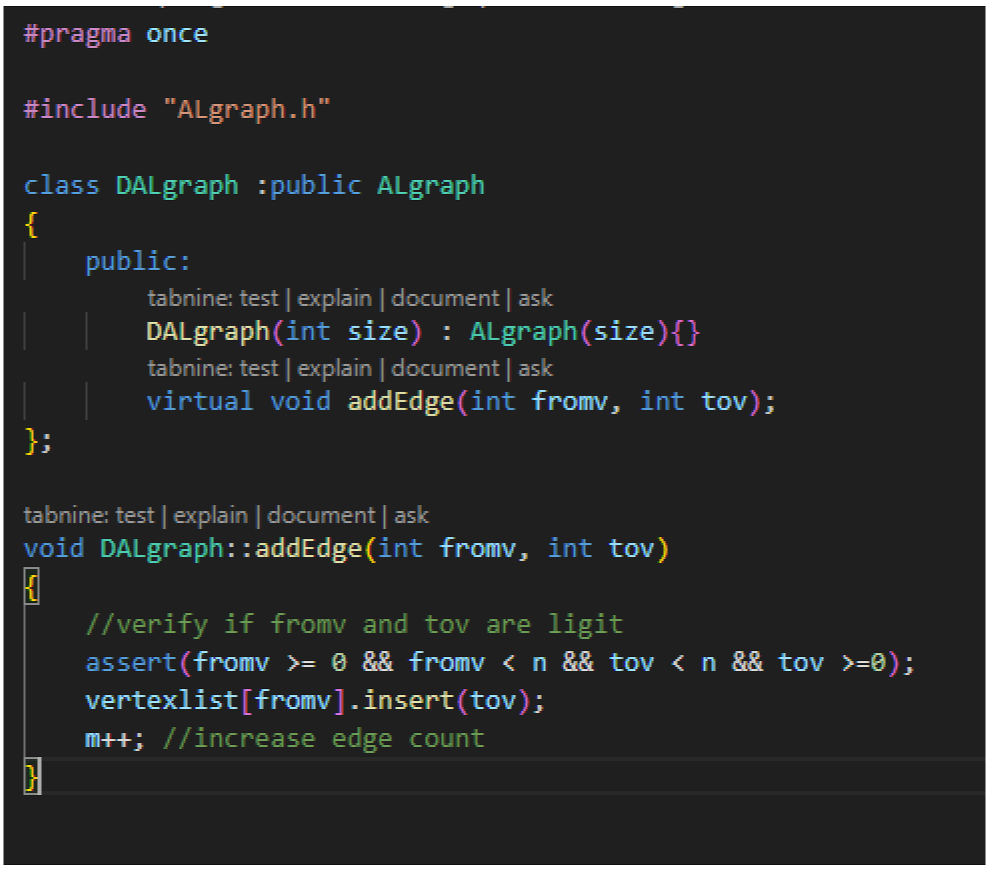


```
#pragma once

#include <assert.h>
#include <iostream>

using namespace std;
class graph{
    public:
        tabnine: test | explain | document | ask
        graph(int size);
        tabnine: test | explain | document | ask
        virtual int vertexsize(); //return n
        tabnine: test | explain | document | ask
        virtual int edgesize(); //return m
        tabnine: test | explain | document | ask
        virtual void addEdge(int fromv, int tov) = 0; //add edge to graph
    protected:
        int n; //number of vertices
        int m; //number of edges
};

tabnine: test | explain | document | ask
graph::graph(int size)
{
    n = size;
    m = 0;
}

tabnine: test | explain | document | ask
int graph::vertexsize()
{
    return n;
}

tabnine: test | explain | document | ask
int graph::edgesize()
{
    return m;
}
```

ALgraph.h

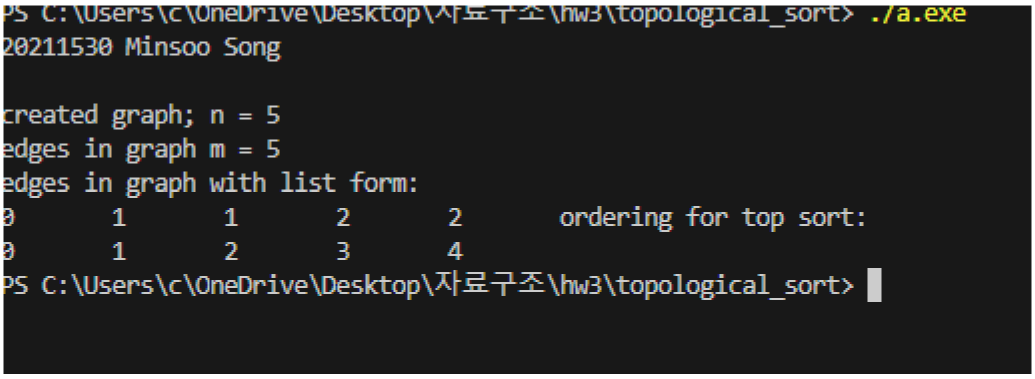

```
#pragma once

#include "graph.h"
#include "list.h"
typedef list <int> intlist;
typedef listiter <int> intlistiter;
class ALgraph:public graph
{
    public:
        //calles graph class to get the size of n(the number of vertices)
        //creates a graph in adjacency list format
        tabnine: test | explain | document | ask
        ALgraph(int size) : graph(size)
        {
            vertexlist = new intlist[n]; assert(vertexlist);
        }
        //class that returns the next edge connecting the vertex
        friend class neighboriter;
    protected:
        intlist* vertexlist;
    private:
};

//inherit from intlistiter class
//
class neighboriter:public intlistiter
{
    public:
        tabnine: test | explain | document | ask
        neighboriter(const ALgraph& G, int startvertex):
        //G.n is the endflag
        //G.vertexlist[startvertex] is the starting vertex
        //goal is to return edges connected by startvertex one by one
        intlistiter(G.vertexlist[startvertex], G.n)
            {assert(startvertex<G.n);}
};
```

DALgraph.h



```cpp
#pragma once

#include "ALgraph.h"

class DALgraph :public ALgraph
{
    public:
        tabnine: test | explain | document | ask
        DALgraph(int size) : ALgraph(size){}
        tabnine: test | explain | document | ask
        virtual void addEdge(int fromv, int tov);
};

tabnine: test | explain | document | ask
void DALgraph::addEdge(int fromv, int tov)
{
    //verify if fromv and tov are ligit
    assert(fromv >= 0 && fromv < n && tov < n && tov >=0);
    vertexlist[fromv].insert(tov);
    m++; //increase edge count
}
```

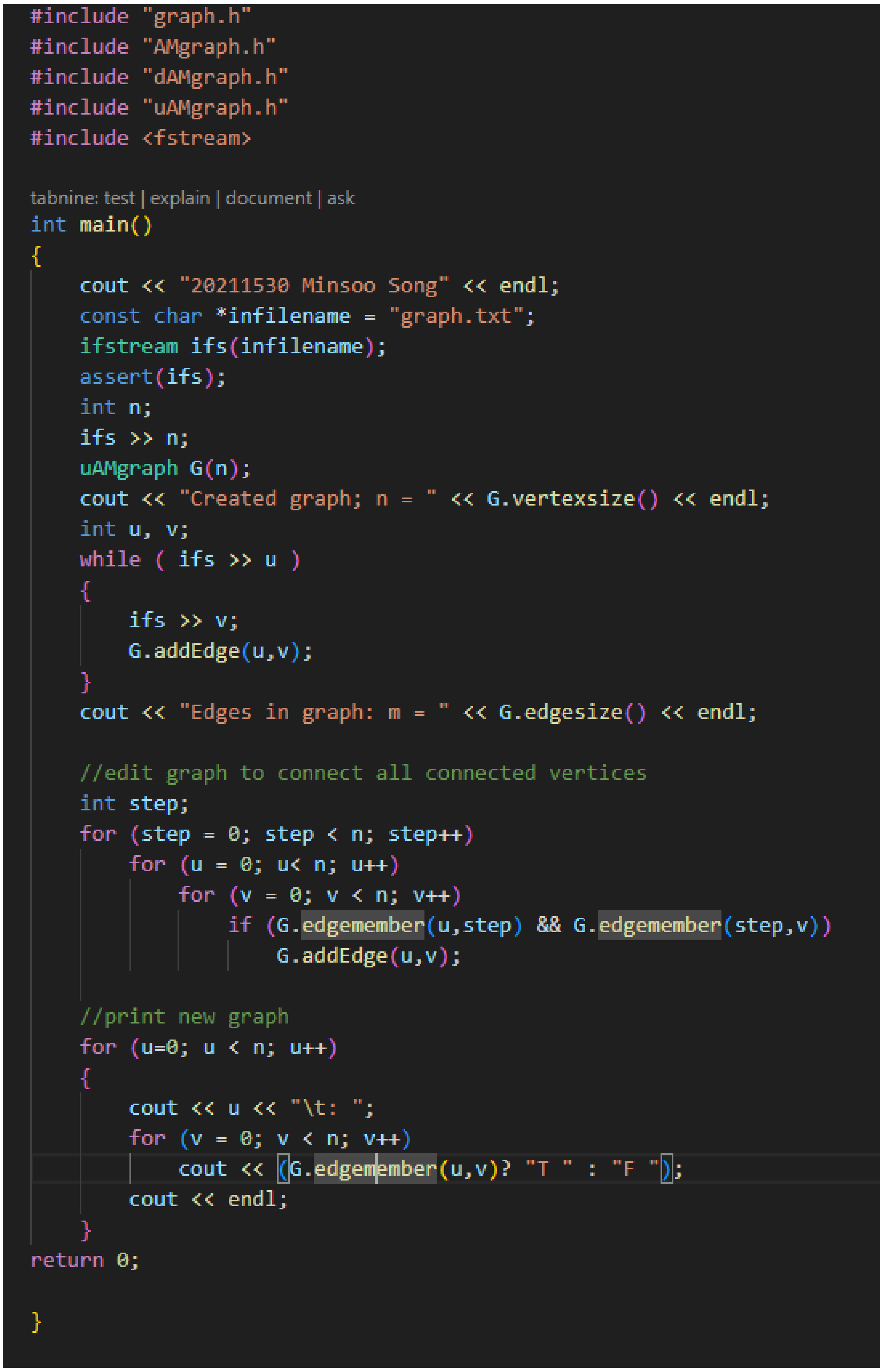
Output



```
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\topological_sort> ./a.exe
20211530 Minsoo Song

created graph; n = 5
edges in graph m = 5
edges in graph with list form:
0       1       1       2       2       ordering for top sort:
0       1       2       3       4
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\topological_sort>
```

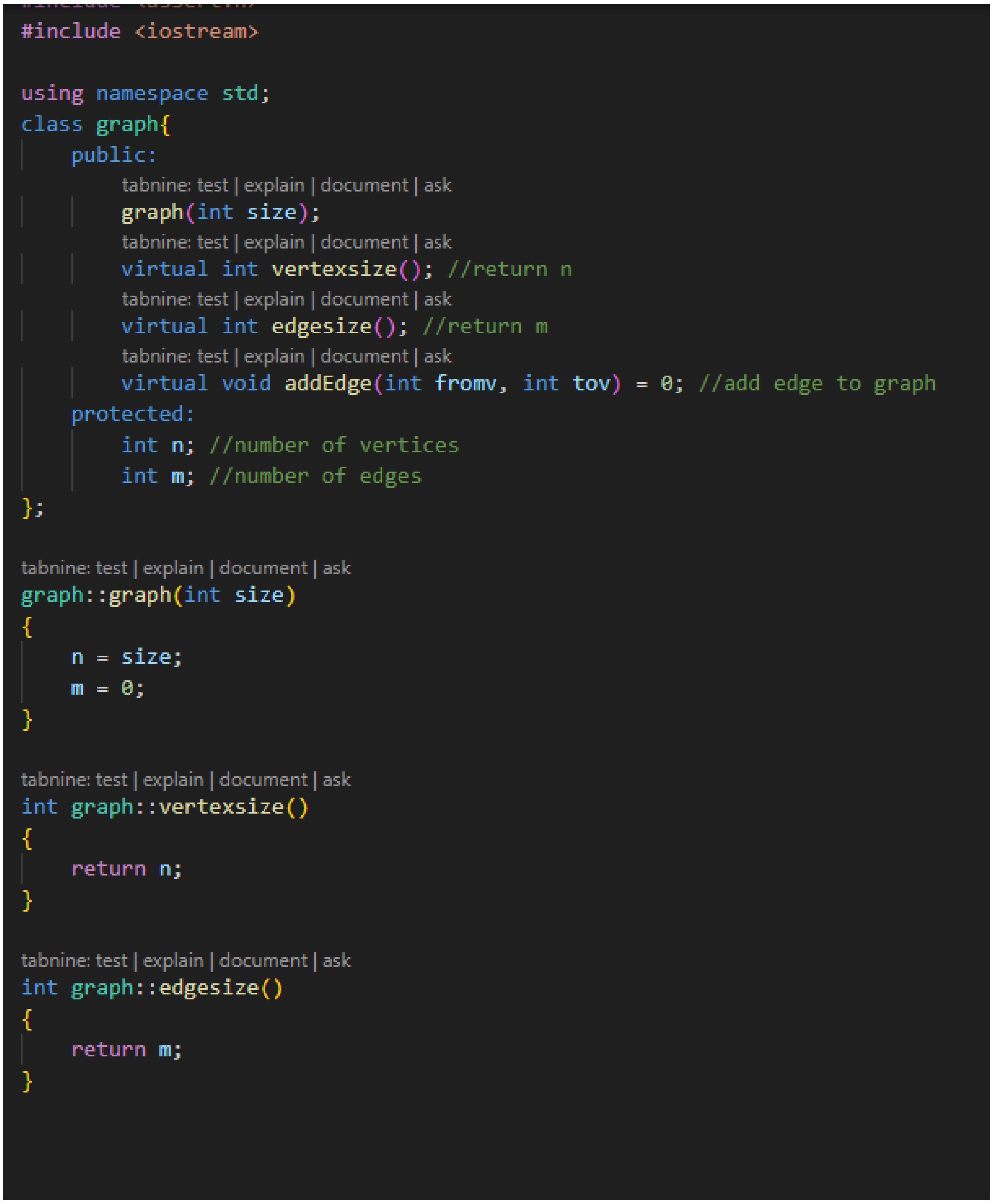
Graph 61 transitive closure

Main



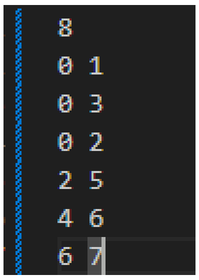
```cpp
#include "graph.h"
#include "AMgraph.h"
#include "dAMgraph.h"
#include "uAMgraph.h"
#include <fstream>

tabnine: test | explain | document | ask
int main()
{
    cout << "20211530 Minsoo Song" << endl;
    const char *infilename = "graph.txt";
    ifstream ifs(infilename);
    assert(ifs);
    int n;
    ifs >> n;
    uAMgraph G(n);
    cout << "Created graph; n = " << G.vertexsize() << endl;
    int u, v;
    while ( ifs >> u )
    {
        ifs >> v;
        G.addEdge(u,v);
    }
    cout << "Edges in graph: m = " << G.edgesize() << endl;

    //edit graph to connect all connected vertices
    int step;
    for (step = 0; step < n; step++)
        for (u = 0; u< n; u++)
            for (v = 0; v < n; v++)
                if (G.edgemember(u,step) && G.edgemember(step,v))
                    G.addEdge(u,v);

    //print new graph
    for (u=0; u < n; u++)
    {
        cout << u << "\t: ";
        for (v = 0; v < n; v++)
            cout << (G.edgemember(u,v)? "T " : "F ");
        cout << endl;
    }
return 0;

}
```

Graph.h



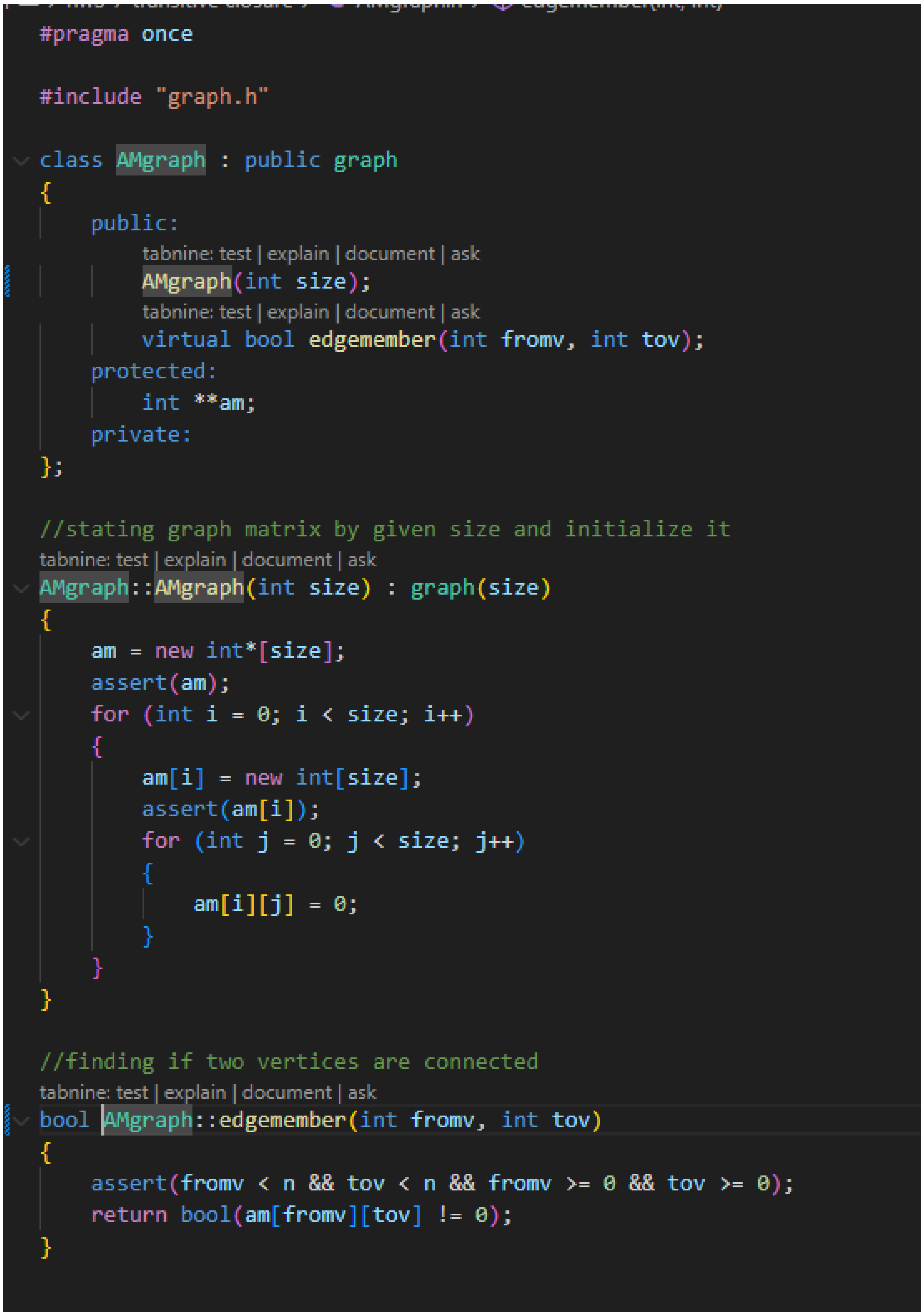
```cpp
#include <iostream>

using namespace std;
class graph{
    public:
        tabnine: test | explain | document | ask
        graph(int size);
        tabnine: test | explain | document | ask
        virtual int vertexsize(); //return n
        tabnine: test | explain | document | ask
        virtual int edgesize(); //return m
        tabnine: test | explain | document | ask
        virtual void addEdge(int fromv, int tov) = 0; //add edge to graph
    protected:
        int n; //number of vertices
        int m; //number of edges
};

tabnine: test | explain | document | ask
graph::graph(int size)
{
    n = size;
    m = 0;
}

tabnine: test | explain | document | ask
int graph::vertexsize()
{
    return n;
}

tabnine: test | explain | document | ask
int graph::edgesize()
{
    return m;
}
```

Graph.txt



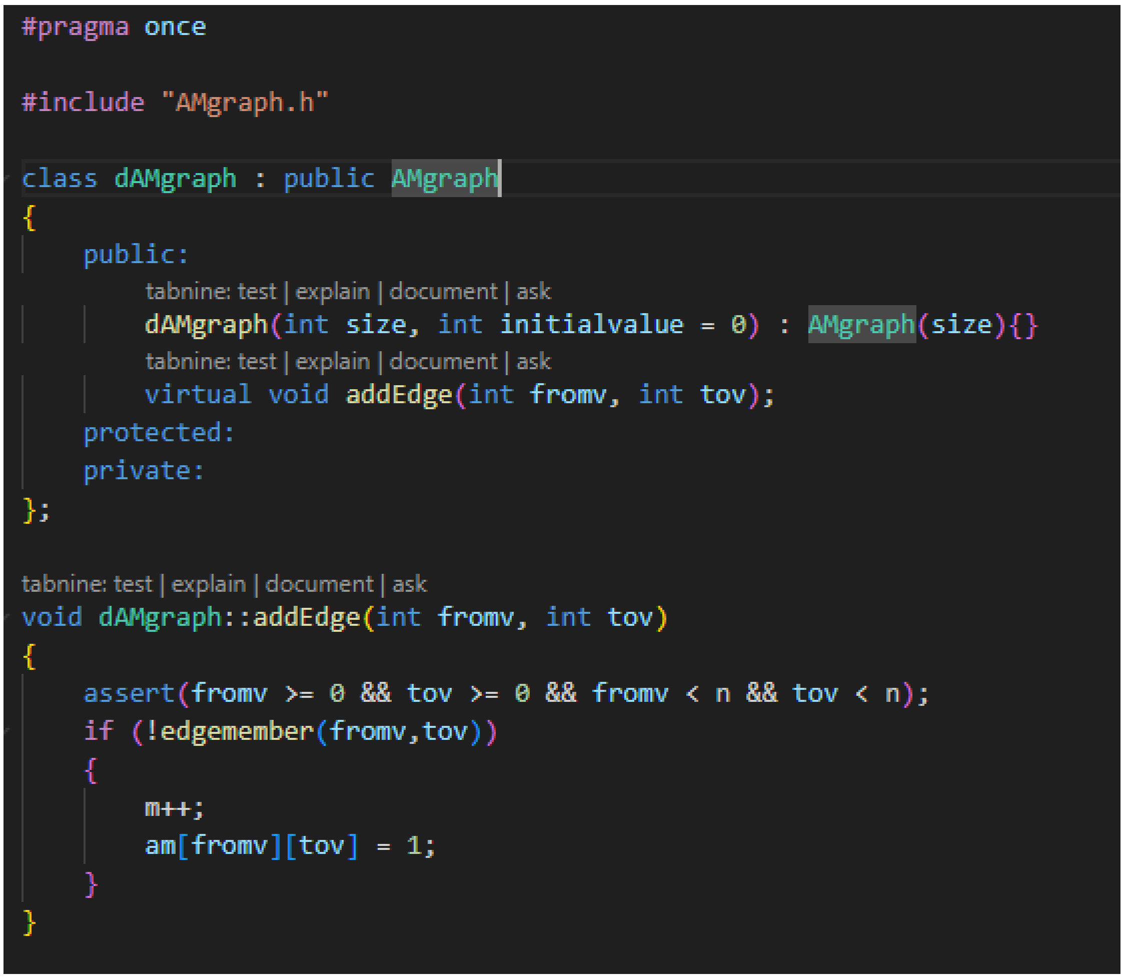
```
8
0 1
0 3
0 2
2 5
4 6
6 7
```

AMgraph.h



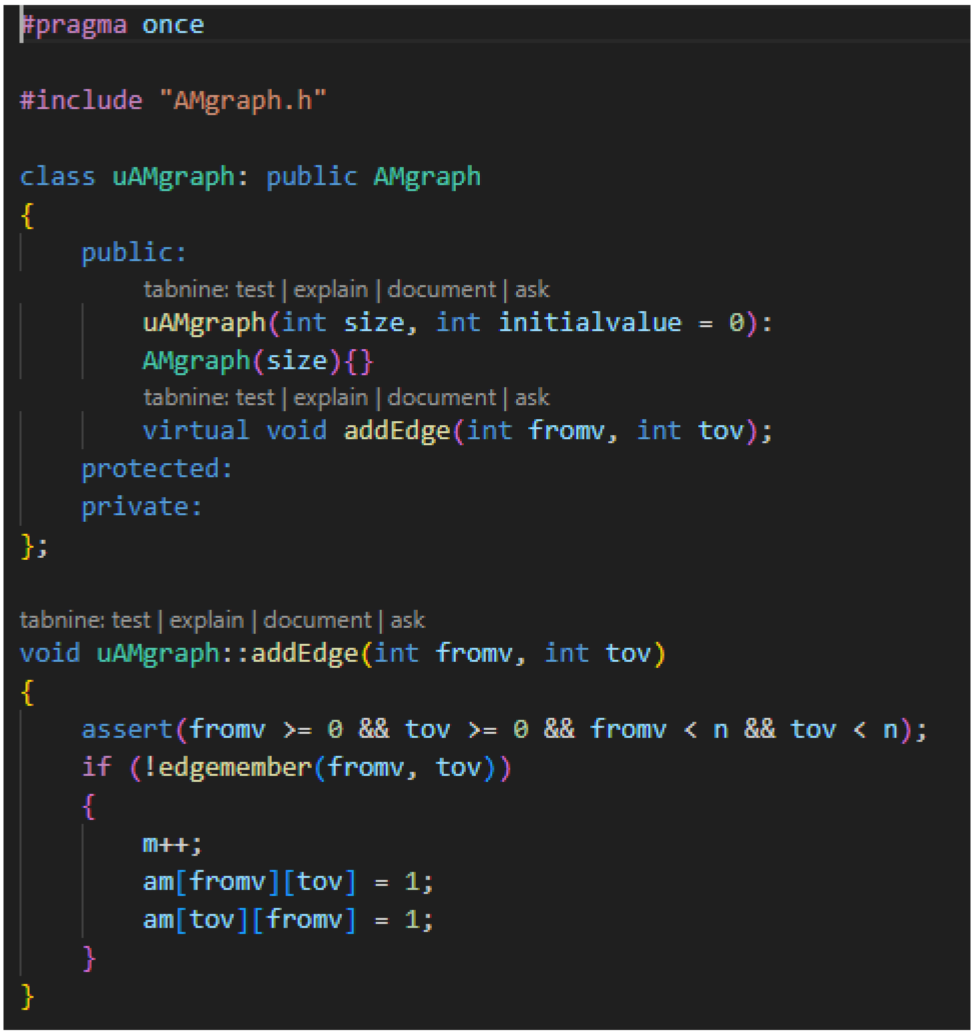
```cpp
#pragma once

#include "graph.h"

class AMgraph : public graph
{
    public:
        tabnine: test | explain | document | ask
        AMgraph(int size);
        tabnine: test | explain | document | ask
        virtual bool edgemember(int fromv, int tov);
    protected:
        int **am;
    private:
};

//stating graph matrix by given size and initialize it
tabnine: test | explain | document | ask
AMgraph::AMgraph(int size) : graph(size)
{
    am = new int*[size];
    assert(am);
    for (int i = 0; i < size; i++)
    {
        am[i] = new int[size];
        assert(am[i]);
        for (int j = 0; j < size; j++)
        {
            am[i][j] = 0;
        }
    }
}

//finding if two vertices are connected
tabnine: test | explain | document | ask
bool AMgraph::edgemember(int fromv, int tov)
{
    assert(fromv < n && tov < n && fromv >= 0 && tov >= 0);
    return bool(am[fromv][tov] != 0);
}
```

dAMgraph.h



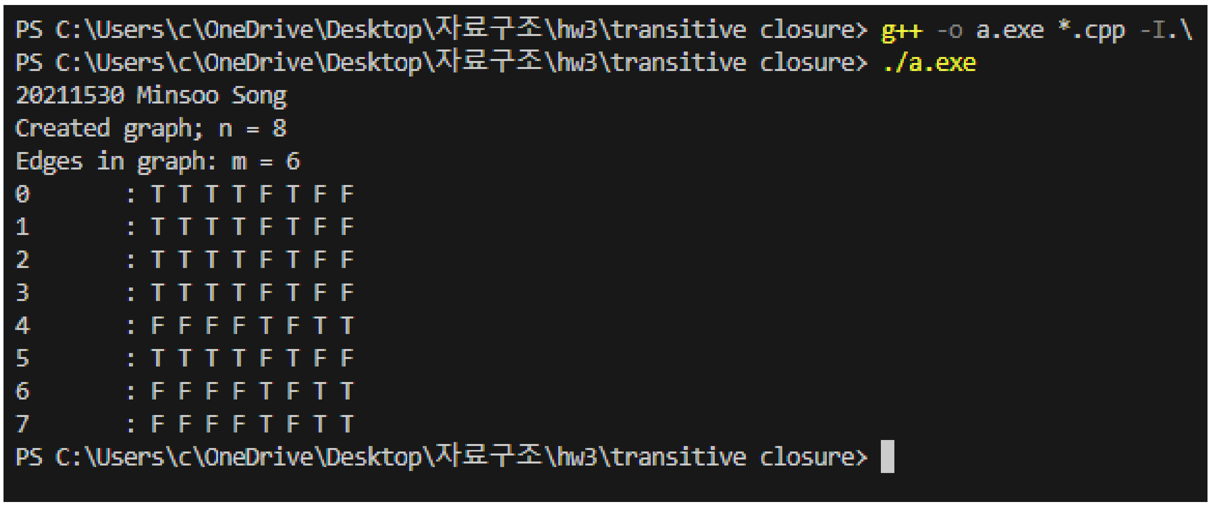
```
#pragma once

#include "AMgraph.h"

class dAMgraph : public AMgraph
{
    public:
        tabnine: test | explain | document | ask
        dAMgraph(int size, int initialvalue = 0) : AMgraph(size){}
        tabnine: test | explain | document | ask
        virtual void addEdge(int fromv, int tov);
    protected:
    private:
};

tabnine: test | explain | document | ask
void dAMgraph::addEdge(int fromv, int tov)
{
    assert(fromv >= 0 && tov >= 0 && fromv < n && tov < n);
    if (!edgemember(fromv,tov))
    {
        m++;
        am[fromv][tov] = 1;
    }
}
```

uAMgraph.h



```cpp
#pragma once

#include "AMgraph.h"

class uAMgraph: public AMgraph
{
    public:
        tabnine: test | explain | document | ask
        uAMgraph(int size, int initialvalue = 0):
        AMgraph(size){}
        tabnine: test | explain | document | ask
        virtual void addEdge(int fromv, int tov);
    protected:
    private:
};

tabnine: test | explain | document | ask
void uAMgraph::addEdge(int fromv, int tov)
{
    assert(fromv >= 0 && tov >= 0 && fromv < n && tov < n);
    if (!edgemember(fromv, tov))
    {
        m++;
        am[fromv][tov] = 1;
        am[tov][fromv] = 1;
    }
}
```

output



```
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\transitive closure> g++ -o a.exe *.cpp -I.\
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\transitive closure> ./a.exe
20211530 Minsoo Song
Created graph; n = 8
Edges in graph: m = 6
0       : T T T T F T F F
1       : T T T T F T F F
2       : T T T T F T F F
3       : T T T T F T F F
4       : F F F F T F T T
5       : T T T T F T F F
6       : F F F F T F T T
7       : F F F F T F T T
PS C:\Users\c\OneDrive\Desktop\자료구조\hw3\transitive closure>
```