```cpp
#include "graph.h"
#include <iostream>
#include <cstdlib>
#include <ctime>

int main(){
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    srand(time(NULL));
    int size;
    std::cin >> size;
    graph a(size);
    while (!std::cin.eof()){
        int s,t,w;
        std::cin >> s >> t >> w;
        a.addedge(s,t,w);
    }
    if (a.dag()) std::cout << "The graph is a DAG" << std::endl;
    else std::cout << "The graph is not a DAG" << std::endl;
    int minimum = a.prim();
    if (minimum==-1) std::cout << "No MST exists!" << std::endl;
    else std::cout << "The total weight of MST is " << minimum << std::endl;
    return 0;
}
```

main.cpp

```cpp
#ifndef GRAPH_H
#define GRAPH_H
#include <vector>
#include <queue>
#include <utility>
#include <climits>
#include <cstdlib>
#include <iostream>

typedef struct _edge_{
    int s,t,w;
}edge;

struct edgecomp{
    bool operator()(const edge &lhs,const edge &rhs){
        return lhs.w>rhs.w;
    }
};

class graph{
    int size;
    std::vector<std::vector<std::pair<int,int>>> adj;
    std::vector<std::vector<std::pair<int,int>>> convadj;
    std::vector<int> innode;
    public:
        graph(int size):size(size),adj(size,std::vector<std::pair<int, int>>()),
            convadj(size,std::vector<std::pair<int, int>>()),innode(size,0){}
        void addedge(int s,int t,int w);
        bool dag();
        int prim();
```

```
};
#endif
```

graph.h

```
#include "graph.h"
#include <set>

void graph::addedge(int s,int t,int w){
    this->adj[s].emplace_back(t,w);
    this->convadj[t].emplace_back(s,w);
    this->innode[t]++;
}

bool graph::dag(){
    std::queue<int> q1,q2;
    for (int i=0;i<this->size;i++){
        if (this->innode[i]==0) q1.push(i);
    }
    while (!q1.empty()){
        int nownode = q1.front();
        q1.pop();
        q2.push(nownode);
        for (auto t: this->adj[nownode]){
            if (this->innode[t.first]!=0){
                this->innode[t.first]--;
                if (this->innode[t.first]==0) q1.push(t.first);
            }
        }
    }
    if (q2.size()==size) return true;
    return false;
}

struct primcmp{
    bool operator()(const std::pair<int, int> &lhs,const std::pair<int, int> &rhs){
        if (lhs.second==rhs.second) return lhs.first< rhs.first;
        return lhs.second<rhs.second;
    }
};

int graph::prim(){
    std::vector<int> d(this->size,INT_MAX);
    std::set<std::pair<int, int>,primcmp> dset;
    d[rand()%this->size] = 0;
    int total = 0;
    for (int i=0;i<this->size;i++){
        dset.emplace(i,d[i]);
    }
    while (!dset.empty()){
        int dnum = dset.begin()->first;
        //std::cerr << dset.size() << " " << dnum << " " << d[dnum] << std::endl;
        if (d[dnum]==INT_MAX) return -1;
        dset.erase(dset.begin());
        total+=d[dnum];
        for (auto edg:this->adj[dnum]){
            int dst = edg.first;
            int wg = edg.second;
```

```
54              auto where = dset.find(std::pair<int,int>(dst,d[dst]));
55              if ((where!=dset.end())&&(d[dst]>wg)){
56                  //std::cerr << "Find " << dst << std::endl;
57                  dset.erase(where);
58                  d[dst]=wg;
59                  dset.emplace(dst,d[dst]);
60              }
61          }
62          for (auto edg:this->convadj[dnum]){
63              int dst = edg.first;
64              int wg = edg.second;
65              auto where = dset.find(std::pair<int,int>(dst,d[dst]));
66              if ((where!=dset.end())&&(d[dst]>wg)){
67                  //std::cerr << "Find " << dst << std::endl;
68                  dset.erase(where);
69                  d[dst]=wg;
70                  dset.emplace(dst,d[dst]);
71              }
72          }
73      }
74      return total;
75  }
```

graph.cpp