UM-SJTU JOINT INSTITUTE
Data Structures and Algorithms
(VP281)

Programming Assignment

Programming Assignment Five
Graph Algorithms

Name: Pan Chongdan
ID: 516370910121

Date: December 9, 2018

1 Introduction

The programming assignment asks us to implement a graph data structure using the adjacency list representation.

2 Code Appendix

The appendix shows the cpp code for main project and time comparison. The following part is for the main algorithm

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <queue>
#include <map>
#include <vector>
#include "min_heap.h"
using namespace std;
class node{
    public:
    int number,in_degree,D;
    bool updated=0;
    node(int n){
        number=n;
        in_degree=0;
        D=0;
        updated=0;
    };
    map<node*,int>neighbour;
};
struct node_cpr{
    bool operator()(node*a,node*b)const{
        if(b->updated==0)return false;
        else return a->D>b->D;
    }
};
struct edge{
    node *start_node;
    node *end_node;
    int weight;
};
int main(int argc,char*argv[]){

    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    int N,n,MST=0;
    string str;
    cin>>N;
    std::vector<edge*>edges;//edge set
    std::vector<node*>T;
    std::map<int, node*>nodes;//nodes set
```

```
42
43      queue<node*> q;
44      queue<node*>order;
45      while(!cin.eof()){
46          getline(cin,str);//read the input
47          if(str.empty())continue;
48          stringstream input;
49          input.clear();
50          input.str(str);
51          int start,end,weight,old_weight;
52          input>>start>>end>>weight;//read node information
53          edge* newedge=new edge;
54          auto it=nodes.find(start);//it refers to the first node in edge
55          if(it==nodes.end()){//the nodes hasn't been insert
56              node* newnode=new node(start);
57              nodes.insert({start,newnode});
58              newedge->start_node=newnode;
59          }
60          else newedge->start_node=it->second;
61
62          it=nodes.find(end);
63          if(it==nodes.end()){//the nodes hasn't been insert
64              node* newnode=new node(end);
65              nodes.insert({end,newnode});
66              newedge->end_node=newnode;
67              newnode->in_degree++;
68          }
69          else {
70              newedge->end_node=it->second;
71              it->second->in_degree++;
72          }
73          newedge->weight=weight;
74
75          if(newedge->start_node->neighbour.find(newedge->end_node)!=newedge->start_node->neighbour.end(
76              old_weight=newedge->start_node->neighbour.find(newedge->end_node)->second;
77              if(old_weight>weight){
78                  newedge->start_node->neighbour.find(newedge->end_node)->second=weight;
79                  newedge->end_node->neighbour.find(newedge->start_node)->second=weight;
80              }
81          }
```

```
        else {
            newedge->start_node->neighbour.insert({newedge->end_node,weight});//keep the neighbour and
            newedge->end_node->neighbour.insert({newedge->start_node,weight});//
        }
        edges.push_back(newedge);
    }
    //determine DAG through topological sort
    n=nodes.size();//the input nodes number

    for(auto it=nodes.begin();it!=nodes.end();++it)if(it->second->in_degree==0)q.push(it->second);
    while(!q.empty()){
        auto front=q.front();
        q.pop();
        order.push(front);
        for(auto it=edges.begin();it!=edges.end();++it){
            if((*it)->start_node==front&&(*it)->end_node->in_degree!=0){
                (*it)->end_node->in_degree--;
                if((*it)->end_node->in_degree==0)q.push((*it)->end_node);
            }
        }

    }
    if(order.size()==n)cout<<"The graph is a DAG"<<endl;
    else cout<<"The graph is not a DAG"<<endl;


    //calculate the MST weight
    bool flag=1;
    node* small;
    T.push_back(nodes.begin()->second);
    nodes.erase(nodes.begin());//erase the first node from T'
    min_heap<node*,node_cpr>V;//a min heap used to sort the node with smallest D in T'
    V.clear();



    while(!nodes.empty()){  //there is still nodes in T'
        small=NULL;
        for(auto it=nodes.begin();it!=nodes.end();++it){//for every nodes in T'
            for(auto it1=it->second->neighbour.begin();it1!=it->second->neighbour.end();++it1){//trave
                if(nodes.find(it1->first->number)==nodes.end()){//the neighbour is not in T' already
```

```
                    if(it1->second<it->second->D||it->second->updated==0){
                        it->second->D=it1->second;//update D
                        it->second->updated=1;
                        V.enqueue(it->second);//put the updated value in min heap
                    }
                    it->second->neighbour.erase(it1);
                }
            }
        }
        while(!V.empty()){
            small=V.dequeue_min();// find the node with smallest D(v)
            if(nodes.find(small->number)!=nodes.end())break;//the node is still in T', so we can use i
            else small=NULL;
        }
        if(V.empty()&&small==NULL){
            flag=0;
            break;
        }
        MST=MST+small->D;//calculate the MST
        nodes.erase(nodes.find(small->number));//put the node out of T' and put it in T
        T.push_back(small);

    }
    if(flag&&n>=N)cout<<"The total weight of MST is "<<MST<<endl;
    else cout<<"No MST exists!"<<endl;

    //release the memory
    for(auto it=T.begin();it!=T.end();++it)delete (*it);
    for(auto it=nodes.begin();it!=nodes.end();++it)delete it->second;
    for(auto it=edges.begin();it!=edges.end();++it)delete (*it);
    return 0;
}
```

The following is for min heap.

```cpp
#ifndef MIN_HEAP_H
#define MIN_HEAP_H

#include <algorithm>

// OVERVIEW: A specialized version of the 'heap' ADT implemented as a binary
//           heap.
template<typename TYPE, typename COMP = std::less<TYPE> >
class min_heap{
public:
    typedef int size_type;

    min_heap(COMP comp = COMP());
    virtual void clear();
    virtual void percolate_up(int id);

    virtual void percolate_down(int id);

    virtual void enqueue(const TYPE &val);



    virtual TYPE dequeue_min();


    virtual const TYPE &get_min() const;


    virtual size_type size() const;


    virtual bool empty() const;

private:
  // Note: This vector *must* be used in your heap implementation.
  std::vector<TYPE> data;
  // Note: compare is a functor object
  COMP compare;

private:
  // Add any additional member functions or data you require here.
```

```
42  };
43
44  template<typename TYPE, typename COMP>
45  min_heap<TYPE, COMP> :: min_heap(COMP comp) {
46      compare = comp;
47      data.push_back(TYPE());
48  }
49
50  template<typename TYPE, typename COMP>
51  void min_heap<TYPE, COMP> :: clear() {
52      data.clear();
53  }
54
55
56  template<typename TYPE, typename COMP>
57  int min_heap<TYPE, COMP> :: size() const {
58      return data.size()-1;
59  }
60
61  template<typename TYPE, typename COMP>
62  bool min_heap<TYPE, COMP> :: empty() const {
63
64      return size()<=0;
65  }
66
67  template<typename TYPE, typename COMP>
68  void min_heap<TYPE, COMP> :: percolate_up(int id) {
69      while(id>1&&compare(data[id/2],data[id])){
70          std::swap(data[id/2],data[id]);
71          id=id/2;
72      }
73  }
74
75  template<typename TYPE, typename COMP>
76  void min_heap<TYPE, COMP> :: percolate_down(int id) {
77      for(auto j=2*id;j<=size();j=2*id){
78          if(j<size()&&compare(data[j],data[j+1]))j++;
79          if(!compare(data[id],data[j]))break;
80          std::swap(data[id],data[j]);
81          id=j;
82      }
```

The following is for unsorted heap.

```
}



template<typename TYPE, typename COMP>
void min_heap<TYPE, COMP> :: enqueue(const TYPE &val) {
    if(data.empty())data.push_back(val);
    data.push_back(val);
    percolate_up(size());
}

template<typename TYPE, typename COMP>
TYPE min_heap<TYPE, COMP> :: dequeue_min() {
    if(this->empty())return data[0];

    std::swap(data[1],data[size()]);
    auto item=data[size()];
    data.pop_back();
    percolate_down(1);

    return item;
}

template<typename TYPE, typename COMP>
const TYPE &min_heap<TYPE, COMP> :: get_min() const {
    // Fill in the body.
    if(this->empty())return data[0];
    else return data[1];
}




#endif //min_heap_H
```