Programming Design, Spring 2014
Homework 7

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

Submission. To submit your work, please upload the following file to the online grading system at http://lckung.im.ntu.edu.tw/PD/.

1. Your .cpp file for Problems 1 and 2.

Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***8:00am, April 21, 2014***. Please answer in either English or Chinese.

Problem 0

(0 point) Please read Chapters 9 and 10 of the textbook.[1] In any case, I strongly suggest you to read the textbook thoroughly before you start to do this homework. For structures and type definitions, please read the slides.

Problem 1

Note. The program you write for this homework will be used, modified, extended for future homework. Please try your best to write a good program!

(100 points) Do you still remember that you were selling apples? In Homework 2, we discussed about forecasting future demands based on historical sales data. With the forecasting system, your business is surprisingly successful. Now you regularly get orders from some loyal customers living in the neighborhood, and you are going to make decisions about the route for delivering apples to them. Again, as an Information Management major student, you will build an information system to facilitate decision making.

Figure 1 is an abstract map of your neighborhood. Your store is at location 1 and two customers Eren and Mikisa live at locations 4 and 5. Each day, Eren orders 10 apples and Mikasa orders 15 apples. These quantities are labeled as -10 and -15 at the two locations. To satisfy their demands, you supply 25 apples from your store. That is why you see a label 25 at location 1. Locations 2 and 3 are just some intersections of roads in your neighborhood. As no one orders apples at this two locations, their labels are 0. In summary, these labels for locations are the supply quantities (where $-x$ means a demand of x units).

Among locations, there are roads. Between two locations there may be two roads in two directions (like those between locations 1 and 2) or just one road in one direction (like the one between locations 2 and 5). Of course, it is also possible for two locations to have no direct connection. On each road, there is a label representing the number of minutes required for traveling through that road. For example, it takes 4 minutes to move from location 1 to location 2. Your question is simple: How to move from location 1, where your store locates, to locations 4 and 5 and then back to location 1 *as fast as possible*?

If you are patient enough, enumerating all possible routes is not impossible. But you are ambitious and will not be satisfied with only two customers in such a small neighborhood. What if one day you have ten customers distributing on a map with fifty locations and five hundred roads? It is clear that a *decision support system* is required to expand your business.

Starting from this homework, we will build such a system. We will start from the following task: Writing a class that represents a map. Such a class will be our foundation for further functionalities.

[1] The textbook is *C++ How to Program: Late Objects Version* by Deitel and Deitel, seventh edition.

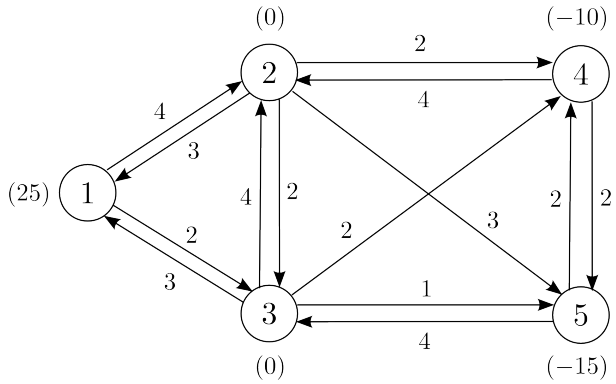

Figure 1: A network

Basics of graphs

(100 basic points and 20 bonus points) The problem that we want to solve can be best formulated with *graph theory*. Mathematically, a *network* (graph) has *nodes* (vertices) and *arcs* (edges/links). In general, arcs may be *directed*, which means an arc between nodes u and v may be either from u to v or from v to u. For an *undirected* arc between two nodes, one may travel in either direction. For an arc from u to v, it will be expressed as (u, v). A *path* (route) from node s to node t is a set of arcs

$$(s, v_1), (v_1, v_2), ..., (v_{k-1}, v_k), \text{ and } (v_k, t)$$

such that s and t are connected. In this case, s is called the *source* and t is called the *destination* of the path. A *cycle* is a path whose destination node is the source node. A path is a *simple path* if it is not a cycle. A network is an *acyclic network* if it contains no cycle. An arc may have *weights* for representing a distance, a cost, etc. A node may also have weights for representing some attributes.

In our problem, nodes are locations and arcs are roads. For simplicity, we will only consider directed arcs each with one weight and nodes each with one weight. All weights will be integers. For each arc, its weight means the traveling time; for each node, its weight means the supply quantity. Our objective is to find a cycle that passes through a subset of nodes (including our store and all customers) with the minimized total traveling time. In this homework, we will only build a class that stores a network. Finding the minimum-time cycle will be left to future homework.[2]

The class

If you already have a concrete idea about building a class for networks, you may simply skip this section. Otherwise, you may want to take a look at the suggestions below. First, two structures, `Node` and `Arc`, should be defined:

```
class Node
{
friend class Network;
private:
  int id;
  int weight;
};
class Arc
{
friend class Network;
```

[2]The class that we will build is used to store data. In other words, it is some kind of *data structure*. More data structures will be introduced and compared in the course "Data Structures". Finding the best route is an optimization problem. All kinds of optimization problems for business operations will be introduced and discussed in the course "Operations Research".

```
private:
  int nodeFrom;
  int nodeTo;
  int weight;
};
```

We then define the following class:

```
class Network
{
private:
  int n;
  int m;
  int nMax;
  int mMax;
  Node* node;
  Arc* arc;
public:
  Network();
  Network(int nMax, int mMax);
  ~Network();
  bool addNode(Node v);
  bool addArc(Arc e);
  bool removeNode(int nodeID);
  bool removeArc(int arcID);
  bool removeArc(int nodeFrom, int nodeTo);
};
```

The first two instance variables `n` and `m` are the number of nodes and arcs, respectively. The fifth and sixth instance variables `node` and `arc` are dynamic arrays that store lists of nodes and arcs, respectively. The reason for using dynamic arrays instead of static arrays is clear: We do not know how large the network will be. If the default constructor is used to create the network, these two arrays are empty (so `node` and `arc` point to `NULL`). On the contrary, if the constructor `Network(int nMax, int mMax)` is used, the lengths of the two dynamic arrays should be set to `nMax` and `mMax`, respectively.

To add things into the initially empty network, one invokes the two functions `addNode(Node v)` and `addArc(Arc e)` to enlarge the network. When one tries to add a node or an arc, first one should check whether that node or arc already exist (by checking `id`, `nodeFrom`, and `nodeTo`). If the node/arc already exists, the function returns `false` and do nothing else. If the node/arc does not exist, one should then check whether there is still an empty space in the dynamic array. If yes, use it; if no, one should *elongate* the dynamic array to, e.g., `nMax * 2` or `mMax * 2`. In other words, every time when the array is full and one more node/arc should be added, the array is elongated to be twice longer.

Sometimes a road may be closed or even a location may be obsoleted. Therefore, we need functions to remove nodes and arcs. When a node is asked to be removed, the function `removeNode(int nodeID)` is invoked. Again, one should first check whether the node really exists. If so, one should remove that node AND all the arcs using it as an endpoint. To remove an arc, `removeArc(int arcID)` or `removeArc(int nodeFrom, int nodeTo)` should be used. Again, one needs to check whether the arc really exist. When one removes an arc, the two endpoints should remain in the network in any case. All these functions should return `false` if nothing is removed. If a removal makes the dynamic array unnecessarily long, it is your discretion whether to make it shorter by releasing some memory spaces. Finally, we need a destructor to release all spaces dynamically allocated.

Please note that the above class definitions and descriptions are just suggestions. You do not need to follow these suggestions. Even if you want to follow, you should make modifications when it helps.

Input/output formats

In this homework, you just create a class for future use. To test whether your class really works, we design a task for you to complete: Given a sequence of nodes $(v_1, v_2, ..., v_n)$, determine whether there is a path from v_1 to v_n that really go through all given nodes in the given order. Below we describe the input/output formats in details.

The input will consist of several lines of English characters and integers. In each line, first there is one or two English characters representing the task to be done for this line. Following integers then provide information regarding that task. Two consecutive items are always separated by a white space. The five tasks are:

- (Adding a node) When a line starts with `AN` followed by two integer v and w, one should try to add a node whose id is v and weight is w into the network. The node should be added if and only if no existing node uses v as its id.
- (Adding an arc) When a line starts with `AA` followed by three integer u, v, and w, one should try to add an arc from node u to node v with weight w into the network. The arc should be added if and only if no existing arc goes from u to v.
- (Removing a node) When a line starts with `RN` followed by one integer v, one should try to remove the node whose id is v from the network. Once the node is removed, all arcs having it as an endpoint should be removed. Nothing should happen to the network if and only if no node uses v as its id.
- (Removing an arc) When a line starts with `RA` followed by two integers u and v, one should try to remove the arc going from u to v from the network. Nothing should happen to the network if and only if no such arc exists.
- (Is there a path?) When a line starts with `P` followed by a sequence of integers v_1, v_2, ..., and v_n, one should check whether a path $(v_1, v_2, ..., v_n)$ exists. If yes, one should output two integers, the total length of this path followed by the sum of supply quantities of all nodes on this path. Of course, a white space is used to separate the two output numbers. If no such path exist, simply output one integer 0. In the given path, a node or an arc may be passed by more than once. In this case, a node's supply quantity should be counted only once but an arc's distance should be counted as many times as it is passed by.

Only node weights may be nonpositive. All other integers are all positive. For the first four tasks, nothing should be output.

As an example, if "PDSp14_hw07_input.txt" is the input file, "PDSp14_hw07_output.txt" contains what should be output by your program. From `AN 1 25` to `AA 5 4 2`, we construct the network in Figure 1. Then four lines of `AN` and `AA` should fail as those nodes and arcs already exist. Then three `P` tasks should be performed:

- For `P 1 2 3 4`, the output is `8 15`.
- For `P 1 2 4 5 3 1 2`, the output is `19 0`. Please note that the distance between nodes 1 and 2 are counted twice but the supply quantity at node 1 is counted only once.
- For `P 4 3 2`, the output is `0` because there is no arc from node 4 to node 3.

Then there are five `RN` and `RA` tasks. The first three should be executed while the last two should not. After the removals, the network becomes the one in Figure 2. It is then unsurprising that the same set of `P` tasks now all get 0 as the output.

Grading criteria

- 70 points are given based on the correctness of your output. Each correct line of output gives you two points. The input will be organized in the following way:

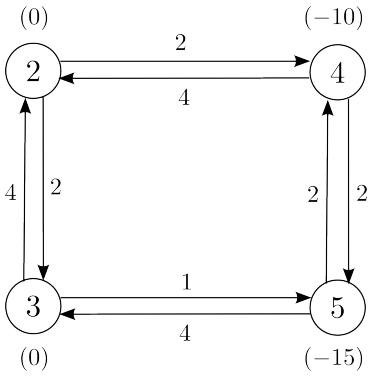

Figure 2: A network

  - In the first part, there are only `AN` and `AA` tasks. Then ten `P` tasks will be assigned to test your functions for adding nodes and arcs. In this part, there will be no conflict when adding an item.
  - In the second part, there are still only `AN` and `AA` tasks. However, some of these tasks should fail as nodes and arcs may already exist. Then ten `P` tasks will be assigned.
  - In the third part, all tasks are possible. Then fifteen `P` tasks will be assigned.

- 30 points will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.

(Bonus) Problem 2

(20 points) While the testing data for Problem 1 will be of just a moderate size, we also prepare another set of testing data which contains a really huge network and many many addition and removal tasks. Ten `P` tasks will be embedded somewhere in the input data, each gives you two points.

You do not need to write a new program for Problem 2. As long as your program is scalable and efficient enough for a huge network, you earn the twenty bonus points. If you cannot, do not feel disappointed. Try to look for some techniques that we have not taught you to improve your program. Or simply wait for a few weeks until we teach you something useful.