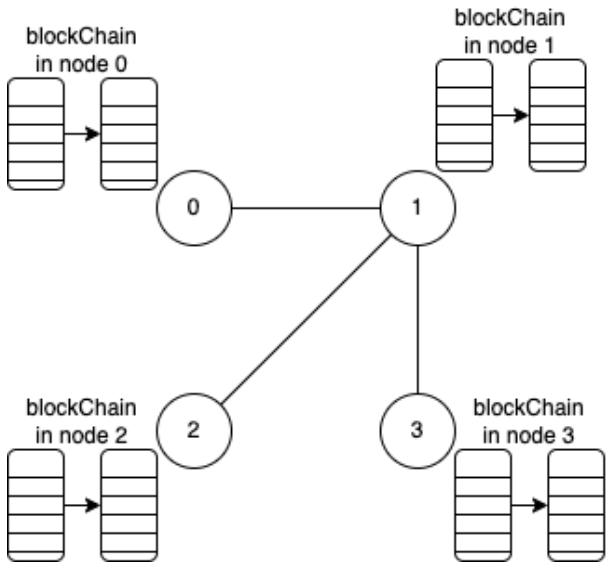
Data Structures – Spring 2023
Project 4
Due: April 3rd 2023 at 11:59 PM

Objective:
The goal of this project is to create a network of BlockChains using the given class structures. This is an extension of project 3.

Description:
In this project, we have an additional class called blockNetwork that has four fields:

- numNodes – maintains the number of nodes in the network
- allNodes – an array/vector of blockchain objects, one for each node
- u – this is a node number representing one end of the edge connecting two nodes
- v – this is a node number representing the other end of the edge connecting two nodes

This means that node u is connected to node v. In the input file, you are given a list of these u-v values that represent the connection between node u and node v. Here is a visualization of the network (from input1.txt).



In this project, on top of all the information you had already stored in project 3, you will also have to keep track of the fromValue and toValue for each transaction.

These two (fromValue and toValue) are essentially how much of coins each person ID (either fromID or toID) have. This is not a part of the input transaction. If the person is already existent in the blockchain in this node, then you need to add to the last value that the person has (after that specific transaction has taken place). If the person is not existent in the blockchain in this

node, then initialize the value of this person to have 100 (100 here represents the number of cyrpto coins). You need not search across nodes in the network. Assume that all the person IDs are unique to a specific node.

Input explanation:

The input file given is in the following format:

```
4               <— number of nodes
10              <— number of transactions per block
27              <— total number of transactions

3               <— number of edges

0 1             <— edge connecting node u (0) and node v (1)
1 2
1 3


0 10001 111 222 50 0042:01012021
0 10002 115 111 80 0142:01012021
0 10003 114 111 20 0245:01022021
0 10004 222 111 10 0143:01022021
0 10005 222 113 10 0144:01022021
0 10006 212 115 70 0244:01022021
0 10007 111 222 40 0245:01022021
0 10008 114 115 70 0334:01022021
0 10009 212 114 70 0445:01032021
0 10010 212 115 70 0555:01032021
0 10011 111 222 50 0142:01032021
0 10012 113 111 80 0242:01042021

1 10013 321 333 20 0345:01042021
1 10014 555 321 10 1143:01042021
1 10015 789 555 10 2144:01042021

2 10016 999 987 80 0242:01042022
2 10017 987 654 20 0345:01042022
2 10018 552 654 10 1143:01042022
2 10019 666 552 10 2144:01042022

3 10020 999 987 80 0242:01042023
3 10021 987 654 20 0345:01042023
3 10022 553 654 10 1143:01042023
3 10023 666 553 10 2144:01042023
3 10024 999 987 80 0242:01042023
3 10025 987 654 20 0345:01042023
```

3 10026 553 654 10 1143:01042023
3 10027 666 553 10 2144:01042023
The input file starts with the number of nodes. Then followed by the number of transactions per block, the total number of transactions, then the edge list. After which you have all the transactions.

Each transaction has the transaction ID, followed by the fromID, toID, amount to be transferred, and finally the time stamp as from project 3. But at the beginning of each transaction, you have the node into which you are going to insert into this transaction.

You may also assume that there will always be enough funds to be transferred from the fromValue to the toValue. But you would have to find out how much the formID and toID person previously had and need to calculate on top of that. Here is an example:

0 10001 111 222 **50** 0042:01012021
- 111 initialized with 100
- 222 initialized with 100
- After transaction takes place, 111 will have 50 and 222 will have 150. The value that 111 and 222 has before the transaction is stored as the fromValue and toValue.

0 10002 115 111 80 0142:01012021
- 115 initialized with 100
- After transaction takes place, 111 will have 130 and 115 will have 20.

0 10003 114 111 20 0245:01022021
- 114 initialized with 100
- After transaction takes place, 114 will have 80 and 111 will have 150.

To get you all started, a **boiler plate file** is given as well. This file is also given alongside this project. Keep up with the FAQ discussion for the project as more assumptions will be updated there. Sample output file will be released soon as well.

Redirected input:

Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. To use redirected input in Visual Studio environment (on windows), follow these steps: After you have opened or created a new project, on the menu go to project, project properties, expand configuration properties until you see Debugging, on the right you will see a set of options, and in the command arguments type <"input filename". The < sign is for redirected input and the input filename is the name of the input file (including the path if not in the working directory).

If you use macOS or linux (or windows using powershell), you may use the terminal to compile and run your program using g++. This is the compiler that we use in the autograder on GradeScope (you may need to install g++ on your computer). Once you installed g++, you may compile your program using the following command:
g++ project4.cpp -o p4

You may then run the program with the redirected input using the following command:

./p4 < input1.txt

For powershell,
Get-Content input1.txt | ./p4

Make sure your program and your input file are in the same folder.

You may also use Google CoLab as an alternative and a sample file is given along with the project. This is an ipynb file which has to be opened in Google CoLab.

A simple program that reads the input file (one number) and displays everything that was read is given below.

```
#include<iostream>
using namespace std;

int main()
{
  int num = 0;
  cout << "Hello World" << endl;
  cin >> num;
  cout << num;
  return 0;
}
```

Sample output file for corresponding input files will be released. The input1.txt file given is a simple input file that will help you understand the project, more complicated ones will be released later and used for grading.

Submission:

Submission will be through GradeScope. Your program will be autograded with test cases and then hand graded to check for documentation and if you have followed the specifications given. You may submit how many ever times to check if your program passes the test cases provided. Test cases will be released at the beginning and later other test cases will be released while grading. For the autograder to work, the program you upload must be named as **project4.cpp**. This will be the only file you will submit. Make sure it is well commented.

Constraints:

1. In this project, the only header you will use the header files that are given to you in the boiler plate code. Using other or excess header files will be subject to a heavy grade penalty for the project.
2. None of the projects is a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.
    - Please review academic integrity policies in the modules.

How to ask for help:

1. Check FAQ discussion board of the project first.
2. Email the TAs with your precise questions.
3. Upload your well commented and clean program to CodePost.
    a. This is a website which is used to share code with your TA only.
    b. You will upload your program and I can view it and comment on it.
    c. Here is the invite link to our class for the summer.
    d. Once you join the class on CodePost, you can upload your program to the Project 1 assignment.

        Note: Your program will **not** be auto graded at CodePost, this is just for when you want to ask a question and a place where I can look at your program and comment on it. CodePost is great for live feedback. GradeScope is the place where you should submit and where your program will be autograded.