

数据结构与算法 I 实验报告

实验 4：区块链 (3)

龚舒凯 2022202790 应用经济-数据科学实验班

<https://github.com/GONGSHUKAI>

2023 年 11 月 18 日

区块链 (3): 迷你区块链系统

1 需求分析

问题描述: 在区块链 (1) 中, 我们从文件中读取数据, 构造了由一组区块构成的链表。本实验要求在此基础上, 实现一个迷你区块链系统, 多个节点之间通过“随机”共识来维护一条一致的链, 完成消息“收发”处理, 以及查询功能。具体而言, 该迷你区块链系统中有两个**区块链节点**和一个**终端客户节点**:

1. **区块链节点 (server):** 每个区块链节点维护一个区块链, 除此之外, 区块链节点中还有“客户消息队列”和“区块消息队列”, 可以根据终端客户发来的客户消息或另一个节点发来的区块消息进行响应, 执行交易/查询功能。
2. **终端客户 (client):** 终端客户每隔一定时间随机向另外两个程序发送交易请求或查询请求。交易数据从交易数据集中生成。每条请求将“插入”到对应节点的“客户消息队列”尾部。

基本要求:

1. 多个节点能正常通信, 并能维护一个一致的区块链。在两个节点分别按顺序展示链表所有区块, 比较是否一致。
2. 支持按照 height 或 hash 查找一个区块; 或按照交易 id 查询一个交易; 以及按顺序展示当前链表中所有区块的 height 和 hash 值。

输出形式: 打开 client 和 server1、server2 三个程序, 每个程序都有自己的输出, 输出内容包括:

1. **client:** 在命令行中输入了客户要求后 (transaction 或 inquiry), 显示客户要求的内容, 以及消息是否发送成功。
2. **server:** 如果是客户的交易请求, 则显示消息是否处理成功。如果是查询请求, 则显示该区块链节点下的查询信息 (height/hash/txid)。

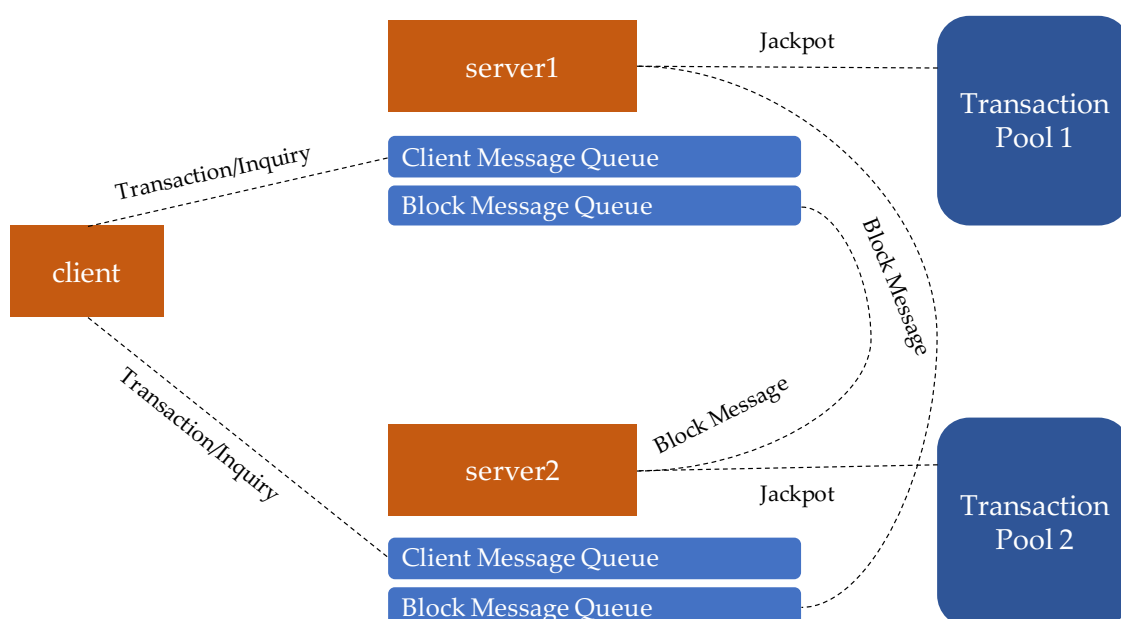


图 1: 迷你区块链系统示意图

2 具体实现

2.1 区块链结构的实现

在区块链 (1) 实验中，我们已经完成了区块链中各组成部分的定义 (block, transaction, input, output)、数据集的读取 (I/O stream) 和按 height/txid 查找区块/交易的功能。在本实验中，我们将区块链 (1) 的代码封装为头文件 Block_Chain.h，方便后续在 server 和 client 的设计中直接调用。

2.2 三个节点的通信

在本实验中，通信是通过文件的读写来实现的。每个节点都维护两个文件夹：clientMessage(客户消息) 和 blockMessage(区块消息)。

- 终端客户向节点 i 发送“客户消息”时，在节点 i 的 clientMessage 文件夹下写文件“`clientMessage.txt`”，文件内容为客户消息 (transaction 或 inquiry)。
- 节点 i 向节点 j 发送“区块消息”时，在节点 j 的 blockMessage 文件夹下写文件“`blockMessage.txt`”，文件内容为区块消息 (区块链中的一个区块)。
- 节点 i 在接受客户消息时，需从自己的 clientMessage 文件夹下读文件“`clientMessage.txt`”，将其还原为一个 transaction 类型或 inquiry 类型的对象。
- 节点 i 在接受区块消息时，需从自己的 blockMessage 文件夹下读文件“`blockMessage.txt`”，将其还原为一个 block 类型的对象。

终端客户和两个区块链节点的主函数都是无限循环运行的 (`while(1)`)，通过 `exec` 方式同时运行三个程序，并在终端客户处输入指令，即可实现三个节点的通信。需要设置三个节点的通信速度上限 (例如：间隔 5 秒发送一条消息)，否则会导致磁盘容量溢出。

2.3 终端客户 (client) 的设计

终端客户只有两个功能：发送交易请求和发送查询请求：

1. 首先，使用者在终端输入查询类型 (是 transaction 还是 inquiry) 和查询内容 (交易数据集中的交易 id 或 height/hash 值)。
2. 然后，程序用随机数生成的方式随机选取一个节点向其发送消息

随机数生成调用了 `#include <random>`，以运行程序的计算机为随机数种子，生成一个服从均匀分布 $U[0, 1]$ 的随机数。代码实现如下：

```
1  int getRandom(int start, int end){
2      random_device rd;
3      mt19937 gen(rd());
4      uniform_int_distribution<> dis(start, end);
5      return dis(gen);
6  }
```

根据生成的随机数访问对应的区块链节点。

如果客户发送的是交易请求，那么在终端输入要发送的交易的 txid。在数据集中找到这一交易后，将交易信息clientMessage1.txt(数字以此类推) 写入区块链节点的clientMessage文件夹下

```

1  void sendTransaction(int fileNumber, int server, block *firstblock, block *endblock){
2      cout << "Visit Server " << server << endl;
3      string fileName = "block_chain_server" + to_string(server) + "/clientMessage/clientMessage" + to_
4      ofstream outputfile;
5      outputfile.open(fileName);
6
7      if (outputfile.is_open()){
8          string client_txid;
9          cout << "Input txid: ";
10         cin >> client_txid;
11
12         block *p = firstblock;
13         int find = 0; //找到交易信息则 find = 1, 否则 find = 0
14         int i = 0;
15         int j = 0;
16         int k = 0;
17         while (p != endblock){
18             while (p->transactions[i].txid != ""){
19                 if (p->transactions[i].txid == client_txid){
20                     outputfile << "Transaction Request" << "\n";
21                     outputfile << "transaction" << i << "info" << "\n";
22                     outputfile << p->height << "\n";
23                     outputfile << p->transactions[i].txid << "\n";
24                     outputfile << p->transactions[i].input_count << "\n";
25                     outputfile << p->transactions[i].output_count << "\n";
26                     outputfile << p->transactions[i].is_coinbase << "\n";
27                     while (p->transactions[i].inputs[j].scriptSig != ""){
28                         outputfile << "\n";
29                         outputfile << "input" << j << "info" << "\n";
30                         outputfile << p->transactions[i].inputs[j].pre_block << "\n";
31                         outputfile << p->transactions[i].inputs[j].prevTxID << "\n";
32                         outputfile << p->transactions[i].inputs[j].prevTxOutIndex << "\n";
33                         outputfile << p->transactions[i].inputs[j].scriptSig << "\n";
34                         j++;
35                     }
36                     while (p->transactions[i].outputs[k].script != ""){
37                         outputfile << "\n";
38                         outputfile << "output" << k << "info" << "\n";

```

```

39         outputfile << p->transactions[i].outputs[k].txid << "\n";
40         outputfile << p->transactions[i].outputs[k].index << "\n";
41         outputfile << p->transactions[i].outputs[k].value << "\n";
42         outputfile << p->transactions[i].outputs[k].script << "\n";
43         k++;
44     }
45     find = 1; //找到这条交易记录
46     break;
47 }
48 i++;
49 }
50 if (find == 1) break;
51 else{
52     i = 0;
53     p = p->next;
54 }
55 }
56 if (find == 1){
57     cout << "Message Successfully Sent!" << endl;
58     outputfile.close();
59 }
60 else{
61     cout << "Transaction Not Found!" << endl;
62 }
63 }
64 else{
65     cout << "Unable to open file";
66 }
67 }

```

如果客户发送的是查询请求,那么在终端输入查询类型(根据 height/hash/txid 查询),函数调用Block_Chain.cpp中的查询函数查询区块链节点中的信息。

```

1 void sendInquiry(int fileNumber, int server, int category, string content){
2     cout << "Visit Server " << server << endl;
3     string fileName = "block_chain_server" + to_string(server) + "/clientMessage/clientMessage" + to_
4     ofstream outputfile;
5     outputfile.open(fileName);
6     if (category == 1){ //根据 height 查询
7         outputfile << "height" << "\n";
8         outputfile << content << "\n";
9         cout << "Message Successfully Sent!" << endl;
10        outputfile.close();

```

```

11     }
12     else if (category == 2){//根据 hash 查询
13         outputfile << "hash" << "\n";
14         outputfile << content << "\n";
15         cout << "Message Successfully Sent!" << endl;
16         outputfile.close();
17     }
18     else{//根据 txid 查询
19         outputfile << "txid" << "\n";
20         outputfile << content << "\n";
21         cout << "Message Successfully Sent!" << endl;
22         outputfile.close();
23     }
24 }

```

2.4 区块链节点 (server) 的设计

区块链节点需要完成三大功能

1. 从交易池中抽取交易组成区块并发送区块消息。
2. 接受区块消息并转化为区块。
3. 接受客户消息并执行对应的操作，具体而言：
 - (a) 客户消息是交易，则考察是否能将交易加入本区块链节点的交易池。
 - (b) 客户消息是查询，则从本区块链节点维护的区块链中查询信息。

区块链节点的算法设计可以参考如下伪码：

```

1  while(1){
2      if(中奖){ //中奖调用一个随机数来实现，中奖几率在 0.01-0.1 之间
3          从“交易池”中取出 n (>=1)个交易，组成一个区块 newBLK。
4          newBLK 的 prevHash 等于本节点区块链表最后一个区块 lastBLK 的 hash 值。
5          newBLK 的 hash 值可以采用一个随机函数来生成；
6          height 值为 lastBLK 的 height+1；
7          merkleRoot 和 nonce 都为空。newBLK 中的交易集合由上述 n 个交易构成。
8          将 newBLK 插入本节点的区块链表末尾。
9          将 newBLK 以某个格式组成字符串“发送”给另一个区块链节点的“区块消息队列”。
10     }
11     else{ //没有中奖
12         if(“区块消息队列”不为空){
13             从“区块消息队列”头部取出一个消息(内容为区块)
14             判断该区块是否与已有区块冲突(即是否存在一个区块，与新区块的 preHash 相同。)
15             if(冲突) 丢弃该区块；

```

```

16         else{
17             将该区块插入到本节点区块链表末尾;
18             从“交易池”中删除该区块中包含的交易;
19         }
20     }
21     else{ // “区块消息队列” 为空
22         从“客户消息队列”头部取出一个消息 MSG;
23         if(MSG 是交易){
24             if(“交易池”不包含该交易) 将该交易加入“交易池”;
25             else 丢弃该交易;
26         }
27         else if (MSG 是查询){
28             在本节点维护的区块链表中执行查询;
29             将查询结果输出在屏幕上;
30         }
31     }
32 }
33 Sleep(5 seconds); //休眠 5 秒
34 }

```

两个区块链节点是完全对称的，这里我们只考察区块链节点 server1 的功能设计。

2.4.1 中奖随机数与随机 hash 生成

区块链节点的设计中涉及到两个需要用到随机数/随机字符串生成的地方：分别为“中奖”判定和 newBLK 的 hash 值生成。

在这里,我们取中奖概率为 0.05,以运行程序的计算机为随机数种子生成一个 1 到 100 的均匀分布 $U[0, 100]$ 。取到其中 5 个特定的数判定为“中奖”。随机 hash 的生成是类似的，我们用一个初始串 str 来随机生成一个 65 位的 hash 字符串。代码实现如下：

```

1  bool isWinner(){//假设中奖概率为 0.05
2      random_device rd;
3      mt19937 gen(rd());
4      uniform_int_distribution<> dis(1, 100);
5      vector<int> winningNumbers = {20, 40, 60, 80, 100};
6      int randomValue = dis(gen);
7      return std::find(winningNumbers.begin(), winningNumbers.end(), randomValue) != winningNumbers.end();
8  }
9  string randomStringGenerator(){
10     string str = "3cc8c69766e26f4ec5b4672e6224cd81c75577674f3cce8c9bb9731a2bb0bd6a";
11     random_device rd;
12     mt19937 gen(rd());
13     uniform_int_distribution<> dis(0, str.length()-1);

```

```

14     string randomString = "";
15     for (int i = 0; i < 64; i++){
16         randomString += str[dis(gen)];
17     }
18     return randomString;
19 }

```

2.4.2 消息队列的实现

区块链节点维护了两个文件夹 clientMessage 和 blockMessage，我们要将其中的客户消息/区块消息按发送时间点的先后整理进入两个队列 clientMessageQueue 和 blockMessageQueue，方便客户消息与区块消息的调取。

遍历文件夹信息需要用到 `#include <filesystem>` 下的 `filesystem::directory_iterator` 类，将文件夹中所有以 txt 结尾的文件加入队列。需要注意，由于 `filesystem::directory_iterator` 按照文件名的 ASCII 码从高到低读取文件名，而最先进入文件夹的消息 ASCII 值是最低的，因此我们需要将读取后的文件名逆序排列，然后加入队列。

此外，由于文件夹中不断有新的 txt 文件被写入，而有些未被读取的文件仍滞留在文件夹中。为了不重复将某些消息加入队列，这里设置了一个有序字典 `map <string, bool> &processedFiles` 用于储存已经入队的文件。在入队列时应检查文件名是否已存在于 `processedFiles` 中。

```

1  queue <string> getTxtFileNames(const string& folderPath, map <string, bool>& processedFiles) {
2      queue <string> fileQueue;
3      vector<string> sortedFileNames; // 用于存储已排序的文件名
4      for (const auto& entry : std::_fs::filesystem::directory_iterator(folderPath)) {
5          if (entry.is_regular_file() && entry.path().extension() == ".txt") {
6              string fileName = entry.path().filename().string();
7              if (processedFiles.find(fileName) == processedFiles.end()) { // 如果文件没有进过队列 (是写进
8                  sortedFileNames.push_back(fileName);
9                  //processedFiles[fileName] = true;
10             }
11         }
12     }
13     // 对文件名进行排序
14     sort(sortedFileNames.begin(), sortedFileNames.end(), [](const string& a, const string& b) {
15         return a < b; // 按照 ASCII 码升序排序
16     });
17
18     // 将排序后的文件名入队列
19     for (const auto& fileName : sortedFileNames) {
20         fileQueue.push(fileName);
21     }
22     return fileQueue;
23 }

```

2.4.3 区块消息的发送

区块链节点的第一大功能是在“中奖”后从交易池中抽取交易组成区块并发送区块消息。抽取交易并组成区块的代码实现如下：

```

1  block* createBlock(block* lastBLK, int n){//从交易池中取出 n(>=1) 个交易，组成一个区块 newBLK。
2      block *newBLK = new block;
3      newBLK->prevHash = lastBLK->hash;
4      newBLK->height = (lastBLK->height) + 1;
5      newBLK->merkleRoot = randomStringGenerator();
6      newBLK->nonce = 0;
7      newBLK->next = NULL;
8      newBLK->hash = randomStringGenerator();
9
10     vector<transaction> firstNTransactions;//这里就认为取出池中前 n 个交易
11     auto it = transactionPool.begin();
12     int count = 0;
13     while (it != transactionPool.end() && count < n) {
14         firstNTransactions.push_back(it->second);
15         ++it;
16         ++count;
17     }
18     for(int i = 0 ; i < n ; i++){
19         newBLK->transactions[i] = firstNTransactions[i];
20         transactionPool.erase(firstNTransactions[i].txid);//将取出的元素从交易池中删除
21     }
22     return newBLK;
23 }
```

组成区块后，接下来需将区块信息发送给另一个区块链节点。发送区块的格式如下：

- 先发送区块的基本信息 (height, hash, prevHash, merkleRoot, nonce)
- 接下来发送区块中每个 transaction[i] 的信息 (txid, input_count, output_count, is_coinbase)
- 接下来发送每个 transaction 下的所有 input[j] 的信息 (prevBlock, prevTxID, prevTxOutIndex, scriptSig) 和 output[k] 的信息 (txid, index, value, script)。

```

1  void sendBlockMessage(int blockMessageNumber, block* newBLK){
2      //将 newBLK “发送” 给另一个区块链节点的 “区块消息队列”。
3      string fileName = "block_chain_server2/blockMessage/blockMessage" + to_string(blockMessageNumber)
4      ofstream outputfile;
5      outputfile.open(fileName);
6      if (outputfile.is_open()){
```

```

7      outputfile << newBLK->height << "\n";
8      outputfile << newBLK->hash << "\n";
9      outputfile << newBLK->prevHash << "\n";
10     outputfile << newBLK->merkleRoot << "\n";
11     outputfile << newBLK->nonce << "\n";
12     int i = 0;
13     int j = 0;
14     int k = 0;
15     outputfile << "\n";
16     while (newBLK->transactions[i].txid != ""){
17         outputfile << "transaction" << i << "info"<< "\n";
18         outputfile << (newBLK->transactions[i]).txid << "\n";
19         outputfile << (newBLK->transactions[i]).input_count << "\n";
20         outputfile << (newBLK->transactions[i]).output_count << "\n";
21         outputfile << (newBLK->transactions[i]).is_coinbase << "\n";
22         while (newBLK->transactions[i].inputs[j].scriptSig != ""){
23             outputfile << "\n";
24             outputfile << "input" << j << "info"<< "\n";
25             outputfile << newBLK->transactions[i].inputs[j].pre_block << "\n";
26             outputfile << newBLK->transactions[i].inputs[j].prevTxID << "\n";
27             outputfile << newBLK->transactions[i].inputs[j].prevTxOutIndex << "\n";
28             outputfile << newBLK->transactions[i].inputs[j].scriptSig << "\n";
29             j++;
30         }
31         while (newBLK->transactions[i].outputs[k].script != ""){
32             outputfile << "\n";
33             outputfile << "output" << k << "info"<< "\n";
34             outputfile << newBLK->transactions[i].outputs[k].txid << "\n";
35             outputfile << newBLK->transactions[i].outputs[k].index << "\n";
36             outputfile << newBLK->transactions[i].outputs[k].value << "\n";
37             outputfile << newBLK->transactions[i].outputs[k].script << "\n";
38             k++;
39         }
40         i++;
41     }
42     outputfile.close();
43 }
44 else{
45     cout << "Unable to open file";
46 }
47 }

```

2.4.4 区块消息和交易信息的读取

当区块链节点从客户消息队列提取出客户交易请求 clientMessage.txt，或从区块消息队列提取出区块信息 blockMessage.txt 时，需要按照特定的格式将 txt 转换为 transaction 类型或 block 类型的数据结构。依照上文给出的客户消息和区块消息格式，实现还原功能的代码过于冗长，放于附录展出：

```

1  block* recoverBlock(string fileName);//根据 blockMSG.txt 文件复原一个 block recoverBLK
2  transaction recover_tsc(string fileName);//根据 clientMSG.txt 文件复原一个 transaction tsc

```

2.4.5 查询功能

若终端客户的请求是查询，则在写入区块链节点 1 的 clientMessage 文件夹的 clientMessage.txt 中，第一行显示查询类型 (即"height/hash/txid")，于是实现代码为：

```

1  void inquiryServerBlock(block *serverBlock, string filePath){
2      ifstream inputFile(filePath);
3      string category;
4      string content;
5      getline(inputFile, category);// inquiry category: height/hash/txid
6      getline(inputFile, content);// inquiry content: heightNumber/hash/txid
7      if (category == "height"){
8          int heightNumber = stoi(content);
9          BlockInfo(heightNumber, serverBlock);
10     }
11     else if (category == "hash"){
12         //Not yet developed :)
13     }
14     else if (category == "txid"){
15         TransactionInfo(content, serverBlock, nullptr);
16     }
17     else cout << "Wrong inquiry category!" << endl;
18 }

```

2.4.6 主程序

依照伪代码描述和上述功能函数，区块链节点的主程序可以写为

```

1  int main(){
2      block *serverBlock = InitServerBlock();
3      block *tail = serverBlock;
4      int blockMessageNumber = 1;//计数，统计一共发送过几次区块消息，从而给区块消息文件命名
5      while(1){

```

```

6      clientMessageQueue = getTxtFileNames(folderPath1, processedFiles1);
7      blockMessageQueue = getTxtFileNames(folderPath2, processedFiles2);
8      if (isWinner()){//中奖几率在 0.01-0.1 之间
9          /* 从“交易池”中取出 n (>=1) 个交易，组成一个区块 newBLK。
10             newBLK 的 prevHash 等于本节点区块链表最后一个区块 lastBLK 的 hash 值。
11             newBLK 的 hash 值可以采用一个随机函数来生成；
12             height 值为 lastBLK 的 height+1； merkleRoot 和 nonce 都为空。
13             newBLK 中的交易集合由上述 n 个交易构成。
14             将 newBLK 插入本节点的区块链表末尾。
15             将 newBLK 以某个格式（比如 JSON）组成字符串“发送”给另一个区块链节点的“区块消息队列”。
16             */
17             block* newBLK = createBlock(tail, 1);
18             tail->next = newBLK;
19             tail = newBLK;
20             cout << "Jackpot! A new block has been inserted to the chain!" << endl;
21             sendBlockMessage(blockMessageNumber, newBLK);
22         }
23         else{//没有中奖
24             if (!blockMessageQueue.empty()){ // “区块消息队列”不为空
25                 //从“区块消息队列”头部取出一个消息（内容为区块）
26                 //判断该区块是否与已有区块冲突（即是否存在一个区块，与新区块的 preHash 相同。）
27                 string firstBlockMSG = blockMessageQueue.front();//从“区块消息队列”头部取出一个消息
28                 blockMessageQueue.pop();
29                 processedFiles2[firstBlockMSG] = true;//将此区块消息标记为被处理过
30                 string filePath_BMSG = "block_chain_server1/blockMessage/"+firstBlockMSG;
31                 block *firstBLK = recoverBlock(filePath_BMSG);//将该消息恢复成一个区块 firstBLK
32
33                 if (judgeConflictBlock(firstBLK, serverBlock)){//“冲突”则丢弃该区块
34                     cout << "Conflict! The block has been discarded!" << endl;
35                 }
36                 else{
37                     //将该区块插入到本节点区块链表末尾；
38                     firstBLK->next = tail->next;
39                     tail->next = firstBLK;
40                     tail = firstBLK;
41                     cout << "The block has been inserted to the chain!" << endl;
42                     //从“交易池”中删除该区块中包含的交易；
43                     int eraseNum = 0;
44                     while (firstBLK->transactions[eraseNum].txid != ""){
45                         transactionPool.erase(firstBLK->transactions[eraseNum].txid);
46                         eraseNum++;
47                     }

```

```

48         cout << "Correspondent transactions deleted in the transaction pool!" << endl;
49     }
50 }//区块消息队列不为空
51 else{//“区块消息队列”为空，从“客户消息队列”头部取出一个消息 MSG;
52     if (!clientMessageQueue.empty()){
53         string firstClientMSG = clientMessageQueue.front();//从“客户消息队列”头部取出一个
54         string filePath_CMSG = "/Users/gongshukai/Desktop/SCHOOL WORK/SOPHOMORE SEM1/DATA
55         clientMessageQueue.pop();
56         processedFiles1[firstClientMSG] = true;//将此客户消息标记为被处理过
57         if (judge_ClientMSG(filePath_CMSG)){//MSG 是交易
58             cout << "Client's transaction request!" << endl;
59             transaction tsc = recover_tsc(filePath_CMSG);
60             if (!find_tsc_in_tscPool(tsc)){//“交易池”不包含该交易，将该交易加入“交易池”;
61                 cout << "transaction added to the transaction pool!" << endl;
62                 transactionPool.insert(pair<string, transaction>(tsc.txid, tsc));//加入交
63             }
64             else{//丢弃该交易
65                 cout << "The transaction has been discarded!" << endl;
66             }
67         }
68         else{//MSG 是查询
69             //在本节点维护的区块链表中执行查询;
70             //将查询结果输出在屏幕上;
71             cout << "Client's inquiry request!" << endl;
72             inquiryServerBlock(serverBlock, filePath_CMSG);
73         }
74     }
75 }//区块消息队列为空，客户消息队列不为空
76 }//中奖 or 没有中奖
77 this_thread::sleep_for(chrono::seconds(5));//隔一会儿再执行下一趟循环，避免 server 过载
78 }//while(1)
79 }

```

3 使用说明与程序测试样例

3.1 使用说明

3.1.1 终端客户 client 的使用说明

首先在终端输入 1(代表发送交易请求) 或 2(代表发送查询请求)

1. 如果输入 1(即发送交易请求)
 - (a) 系统随机指定发送到某一区块链节点 (server1 或 server2)
 - (b) 用户输入要发送的交易 txid。注意, txid 必须来自于数据集文件, 否则无法发送交易。
 - (c) 终端客户 client 显示"Message Successfully Sent!", 表明已成功向区块链节点发送交易请求。
2. 如果输入 2(即发送查询请求)
 - (a) 首先选择查询请求的类型 (即根据 height/hash/txid 查询区块/交易)
 - (b) 其次输入 1 或 2, 代表要查询的区块链节点
 - (c) 最后输入查询信息, 即 height/hash/txid
 - (d) 终端客户 client 显示"Message Successfully Sent!", 表明已成功向区块链节点发送查询请求。

3.1.2 区块链节点 client 的显示说明

- 如果显示"The block has been inserted to the chain!", 则表明区块链节点已成功从客户消息队列中提取出另一个区块链节点发送的区块信息, 并将其插入本节点维护的区块链末尾。
- 如果显示"Conflict! The block has been discarded!", 则表明另一节点发送来的区块与本节点区块链中某节点冲突, 将该区块丢弃。
- 如果显示"Correspondent transactions deleted in the transaction pool!", 则表明区块插入本节点区块链末尾后, 将该区块中的交易对应的从本节点的交易池中删除。
- 如果显示"Client's transaction request!", 则表明正在提取客户消息队列的交易请求。
- 如果显示"transaction added to the transaction pool!", 则说明交易请求不冲突, 已将交易加入到本区块链节点的交易池中。
- 如果显示"The transaction has been discarded!", 则说明客户发送的交易与交易池中交易冲突, 已丢弃该交易。
- 如果显示"Jackpot! A new block has been inserted to the chain!", 则表明该区块链节点“中奖”, 从交易池中提取 n 个交易 (本程序中 $n = 1$) 组成一个区块, 插入到本区块链末尾的同时也将区块信息发送到另一个区块链节点的区块消息文件夹下。
- 如果显示"Client's inquiry request!", 则表明正在提取客户消息队列的查询请求。如果接下来终端输出区块信息/交易信息, 则说明查询成功。
- 如果显示"Block/transaction not found!", 则表明客户要查询的区块/交易不存在。

3.2 程序测试样例

三个程序同时运行的效果如下：

```

Oct.27_Lab -- client -- 80x51
1
Visit Server 1
Input txid: c5c36a7e0a7d7a95ada88459231bd8afce2bf93e5921ef39a4606bdabbdcaeb5
Message Successfully Sent!
Input your request: 1 Transaction; 2 Inquiry
1
Visit Server 2
Input txid: bdf2c8a2482cc9ca50ee423dad8549e27fdb62f2ae3218eae32277ecec715ec5
Message Successfully Sent!
Input your request: 1 Transaction; 2 Inquiry
1
Visit Server 1
Input txid: 5a916d9e74946ed6f3c2aec1acea20ae59a2af216eb9b33f91a0771f20678bed
Message Successfully Sent!
Input your request: 1 Transaction; 2 Inquiry
1
Visit Server 2
Input txid: c5c36a7e0a7d7a95ada88459231bd8afce2bf93e5921ef39a4606bdabbdcaeb5
Message Successfully Sent!
Input your request: 1 Transaction; 2 Inquiry
1
Visit Server 2
Input txid: 5a916d9e74946ed6f3c2aec1acea20ae59a2af216eb9b33f91a0771f20678bed
Message Successfully Sent!
Input your request: 1 Transaction; 2 Inquiry
1
Visit Server 2
Input txid: 1adaacd29c4fddcfac988f8650fc1bec8a59ef49614575b81260f4b329c84910
Message Successfully Sent!
Input your request: 1 Transaction; 2 Inquiry
2
Inquiry Type: 1 Search by height; 2 Search by hash; 3 Search by txid
1
Input server:
1
Input height:
2
Visit Server 1
Message Successfully Sent!
Input your request: 1 Transaction; 2 Inquiry
2
Inquiry Type: 1 Search by height; 2 Search by hash; 3 Search by txid
1
Input server:
2
Input height:
2
Visit Server 2
Message Successfully Sent!
Input your request: 1 Transaction; 2 Inquiry

Oct.27_Lab -- server1 -- 80x24
Client's transaction request!
transaction added to the transaction pool!
Client's transaction request!
transaction added to the transaction pool!
Client's transaction request!
transaction added to the transaction pool!
Jackpot! A new block has been inserted to the chain!
Jackpot! A new block has been inserted to the chain!
Jackpot! A new block has been inserted to the chain!
Client's inquiry request!
Client's inquiry height: 2
Block Height: 2
Block Hash: e543ce27751ae74c3bfae96746b77b464656cbb6cd6d224cce797ef555c2262f
Block prevHash: 833c6c76b616fb1c9d8f3b2f2be02675c66cc834f6c21ddc71ce779c96bddb96
Block merkleRoot: 8ace7ce621472722b4cb3b36e686ce3cb0327c3565c6614b8ca72ecec2422c
8b
Block nonce: 0

Block Transaction 0 txid: bdf2c8a2482cc9ca50ee423dad8549e27fdb62f2ae3218eae32277
ecec715ec5
Block Transaction 0 input count: 1
Block Transaction 0 output count: 2
Block Transaction 0 Coinbase: 0
Transaction 0 in Block: 2

Oct.27_Lab -- server2 -- 80x24
Last login: Sat Nov 18 20:07:10 on ttys004
(base) gongshukai@gongshukaideMacBook-Air-6 ~ % cd "/Users/gongshukai/Desktop/SC
HOOL WORK/SOPHOMORE SEM1/DATA STRUCTURE & ALGORITHM /SLIDES & HOMEWORK & LAB/LA
B/Oct.27_Lab"
(base) gongshukai@gongshukaideMacBook-Air-6 Oct.27_Lab % ./server2
Client's transaction request!
transaction added to the transaction pool!
Client's transaction request!
transaction added to the transaction pool!
Client's transaction request!
transaction added to the transaction pool!
Client's transaction request!
transaction added to the transaction pool!
The block has been inserted to the chain!
Correspondent transactions deleted in the transaction pool!
Client's inquiry request!
Client's inquiry height: 2
Block Not Found!

```

4 附录

根目录下所有文件 (包括Block_Chain.h) 详见https://github.com/GONGSHUKAI/Data_Structure/tree/main/Lab_Code/Lab_4/Oct.27_Lab, 这里展示三个主程序代码client.cpp, server1.cpp, server2.cpp。

server1.cpp(server2.cpp 的代码是完全相同的, 仅根目录代码不同)

```

1  #include "Block_Chain.h"
2  #include <random>
3  #include <filesystem>
4  #include <map>
5  #include <queue>
6  #include <chrono>
7  #include <thread>
8
9  queue <string> clientMessageQueue;//结点 1 的客户消息队列
10 queue <string> blockMessageQueue;//结点 1 的区块消息队列
11 map <string, transaction> transactionPool;//结点 1 的交易池
12 string folderPath1 = "/Users/gongshukai/Desktop/SCHOOL WORK/SOPHOMORE SEM1/DATA STRUCTURE & ALGORITHM";
13 string folderPath2 = "/Users/gongshukai/Desktop/SCHOOL WORK/SOPHOMORE SEM1/DATA STRUCTURE & ALGORITHM";
14 map <string, bool> processedFiles1;//已处理的客户消息队列文件
15 map <string, bool> processedFiles2;//已处理的区块消息队列文件
16 block* serverBlock;//结点 1 的区块链
17
18 string randomStringGenerator(){
19     string str = "3cc8c69766e26f4ec5b4672e6224cd81c75577674f3cce8c9bb9731a2bb0bd6a";
20     random_device rd;
21     mt19937 gen(rd());
22     uniform_int_distribution<> dis(0, str.length()-1);
23     string randomString = "";
24     for (int i = 0; i < 64; i++){
25         randomString += str[dis(gen)];
26     }
27     return randomString;
28 }
29
30 bool isWinner(){//假设中奖概率为 0.05
31     random_device rd;
32     mt19937 gen(rd());
33     uniform_int_distribution<> dis(1, 100);
34     vector<int> winningNumbers = {20, 40, 60, 80, 100};
35     int randomValue = dis(gen);
36     return std::find(winningNumbers.begin(), winningNumbers.end(), randomValue) != winningNumbers.end();

```



```
37 }
38
39 queue <string> getTxtFileNames(const string& folderPath, map <string, bool>& processedFiles) {
40     queue <string> fileQueue;
41     vector<string> sortedFileNames; // 用于存储已排序的文件名
42     for (const auto& entry : std::_fs::filesystem::directory_iterator(folderPath)) {
43         if (entry.is_regular_file() && entry.path().extension() == ".txt") {
44             string fileName = entry.path().filename().string();
45             if (processedFiles.find(fileName) == processedFiles.end()) { // 如果文件没有进过队列 (是写进
46                 sortedFileNames.push_back(fileName);
47                 // processedFiles[fileName] = true;
48             }
49         }
50     }
51     // 对文件名进行排序
52     sort(sortedFileNames.begin(), sortedFileNames.end(), [](const string& a, const string& b) {
53         return a < b; // 按照 ASCII 码升序排序
54     });
55
56     // 将排序后的文件名入队列
57     for (const auto& fileName : sortedFileNames) {
58         fileQueue.push(fileName);
59     }
60     return fileQueue;
61 }
62
63 block* InitServerBlock(){
64     block* head = new block;
65     head->height = 0; // 区块高度
66     head->hash = "7c5b79677777cc627166cabbc347679b6469749c7cbb7b19617f6c3674c4c3bb"; // 自定义的头结点的
67     head->prevHash = ""; // 前一个区块的哈希值
68     head->merkleRoot = "229accb4c760c7c57e7c769e4afce7e434c26757472c281277ceeb36618b2cc5"; // 本区块中所
69     head->nonce = 114514; // 神秘数
70     head->next = nullptr;
71     return head; // 返回节点 1 的区块链的头结点
72 }
73
74 block* createBlock(block* lastBLK, int n) { // 从交易池中取出 n(>=1) 个交易，组成一个区块 newBLK。
75     block *newBLK = new block;
76     newBLK->prevHash = lastBLK->hash;
77     newBLK->height = (lastBLK->height) + 1;
78     newBLK->merkleRoot = randomStringGenerator();
```

```

79     newBLK->nonce = 0;
80     newBLK->next = NULL;
81     newBLK->hash = randomStringGenerator();
82
83     vector<transaction> firstNTransactions;//这里就认为取出池中前 n 个交易
84     auto it = transactionPool.begin();
85     int count = 0;
86     while (it != transactionPool.end() && count < n) {
87         firstNTransactions.push_back(it->second);
88         ++it;
89         ++count;
90     }
91     for(int i = 0 ; i < n ; i++){
92         newBLK->transactions[i] = firstNTransactions[i];
93         transactionPool.erase(firstNTransactions[i].txid);//将取出的元素从交易池中删除
94     }
95     return newBLK;
96 }
97
98 void sendBlockMessage(int blockMessageNumber, block* newBLK){
99     //将 newBLK “发送” 给另一个区块链节点的 “区块消息队列”。
100     string fileName = "/Users/gongshukai/Desktop/SCHOOL WORK/SOPHOMORE SEM1/DATA STRUCTURE & ALGORIT
101     ofstream outputfile;
102     outputfile.open(fileName);
103     if (outputfile.is_open()){
104         outputfile << newBLK->height << "\n";
105         outputfile << newBLK->hash << "\n";
106         outputfile << newBLK->prevHash << "\n";
107         outputfile << newBLK->merkleRoot << "\n";
108         outputfile << newBLK->nonce << "\n";
109         int i = 0;
110         int j = 0;
111         int k = 0;
112         outputfile << "\n";
113         while (newBLK->transactions[i].txid != ""){
114             outputfile << "transaction" << i << "info"<< "\n";
115             outputfile << (newBLK->transactions[i]).txid << "\n";
116             outputfile << (newBLK->transactions[i]).input_count << "\n";
117             outputfile << (newBLK->transactions[i]).output_count << "\n";
118             outputfile << (newBLK->transactions[i]).is_coinbase << "\n";
119             while (newBLK->transactions[i].inputs[j].scriptSig != ""){
120                 outputfile << "\n";

```

```

121         outputfile << "input" << j << "info"<< "\n";
122         outputfile << newBLK->transactions[i].inputs[j].pre_block << "\n";
123         outputfile << newBLK->transactions[i].inputs[j].prevTxID << "\n";
124         outputfile << newBLK->transactions[i].inputs[j].prevTxOutIndex << "\n";
125         outputfile << newBLK->transactions[i].inputs[j].scriptSig << "\n";
126         j++;
127     }
128     while (newBLK->transactions[i].outputs[k].script != ""){
129         outputfile << "\n";
130         outputfile << "output" << k << "info"<< "\n";
131         outputfile << newBLK->transactions[i].outputs[k].txid << "\n";
132         outputfile << newBLK->transactions[i].outputs[k].index << "\n";
133         outputfile << newBLK->transactions[i].outputs[k].value << "\n";
134         outputfile << newBLK->transactions[i].outputs[k].script << "\n";
135         k++;
136     }
137     i++;
138 }
139 outputfile.close();
140 }
141 else{
142     cout << "Unable to open file";
143 }
144 }
145
146 int judge_ClientMSG(string filePath){
147     ifstream inputFile(filePath);
148     string line;
149     getline(inputFile, line);
150
151     if (line == "Transaction Request") return 1;//考察 ClientMSG.txt 的第一行，看他是交易还是查询，是交
152     else if (line == "") return 2;//文件为空，返回 false
153     else return 3;
154 }
155
156 void inquiryServerBlock(block *serverBlock, string filePath){
157     ifstream inputFile(filePath);
158     string category;
159     string content;
160     getline(inputFile, category);// inquiry category: height/hash/txid
161     getline(inputFile, content);// inquiry content: heightNumber/hash/txid
162     if (category == "height"){

```

```

163         cout << "Client's inquiry height: " << content << endl; //" 用户查询的区块高度为:
164         int heightNumber = stoi(content);
165         BlockInfo(heightNumber, serverBlock);
166     }
167     else if (category == "hash"){
168         cout << "Client's inquiry hash: " << content << endl;
169         //Not yet developed :)
170     }
171     else if (category == "txid"){
172         cout << "Client's inquiry txid: " << content << endl;
173         TransactionInfo(content, serverBlock, nullptr);
174     }
175     else cout << "Wrong inquiry category!" << endl;
176 }
177
178 bool find_tsc_in_tscPool(transaction tsc){
179     auto it = transactionPool.find(tsc.txid);
180     if (it != transactionPool.end()) return true; //找到了
181     else return false; //没找到
182 }
183
184 bool judgeConflictBlock(block *myBlock, block *serverBlock){
185     block *temp = serverBlock;
186     while (temp != NULL){
187         if (temp->prevHash == myBlock->prevHash) return true; //找到了
188         else temp = temp->next;
189     }
190     return false; //没找到
191 }
192
193 block* recoverBlock(string fileName) { //根据 blockMSG.text 文件复原一个 block recoverBLK
194     block *recoverBLK = new block;
195     ifstream inputFile(fileName);
196
197     if (inputFile.is_open()) {
198         string line;
199         unsigned long long transactionIndex = 0;
200         unsigned long long inputIndex = 0;
201         unsigned long long outputIndex = 0;
202
203         // Read block information
204         getline(inputFile, line);

```

```
205         istreamblockstream blockHeight(line);
206         blockHeight >> recoverBLK->height;
207
208         getline(inputFile, line);
209         istreamblockstream blockHash(line);
210         blockHash >> recoverBLK->hash;
211
212         getline(inputFile, line);
213         istreamblockstream blockPrevHash(line);
214         blockPrevHash >> recoverBLK->prevHash;
215
216         getline(inputFile, line);
217         istreamblockstream blockMerkleRoot(line);
218         blockMerkleRoot >> recoverBLK->merkleRoot;
219
220         getline(inputFile, line);
221         istreamblockstream blockNonce(line);
222         blockNonce >> recoverBLK->nonce;
223
224         recoverBLK->next = nullptr;
225         // Skip empty line
226         getline(inputFile, line);
227
228         while (getline(inputFile, line)) {
229             if (line.find("transaction") != string::npos) {
230                 // Read transaction information
231                 getline(inputFile, line); // Skip transaction info line
232                 transaction& tx = recoverBLK->transactions[transactionIndex];
233                 istreamblockstream txid(line);
234                 txid >> tx.txid;
235
236                 getline(inputFile, line);
237                 istreamblockstream inputCount(line);
238                 inputCount >> tx.input_count;
239
240                 getline(inputFile, line);
241                 istreamblockstream outputCount(line);
242                 outputCount >> tx.output_count;
243
244                 getline(inputFile, line);
245                 istreamblockstream isCoinbase(line);
246                 isCoinbase >> tx.is_coinbase;
```

```
247
248         // Read inputs
249         for (int i = 0; i < tx.input_count; i++) {
250             getline(inputFile, line); // Skip empty line
251             getline(inputFile, line); // Skip input info line
252             input& in = tx.inputs[inputIndex];
253
254             getline(inputFile, line);
255             istringstream pre_block(line);
256             pre_block >> in.pre_block;
257
258             getline(inputFile, line);
259             istringstream prevTxID(line);
260             prevTxID >> in.prevTxID;
261
262             getline(inputFile, line);
263             istringstream prevTxOutIndex(line);
264             prevTxOutIndex >> in.prevTxOutIndex;
265
266             getline(inputFile, line);
267             istringstream scriptSig(line);
268             ostringstream scriptSigStream;
269             scriptSigStream << scriptSig.rdbuf(); // 将 scriptSig 中的所有字符写入到 ostrings
270             in.scriptSig = scriptSigStream.str(); // 将 ostringstream 中的内容赋值给 in.scrip
271             inputIndex++;
272         }
273         // Read outputs
274         for (int i = 0; i < tx.output_count; i++) {
275             getline(inputFile, line); // Skip empty line
276
277             getline(inputFile, line); // Skip output info line
278             output& out = tx.outputs[outputIndex];
279
280             getline(inputFile, line);
281             istringstream txid(line);
282             txid >> out.txid;
283
284             getline(inputFile, line);
285             istringstream index(line);
286             index >> out.index;
287
288             getline(inputFile, line);
```

```

289         istream value(line);
290         value >> out.value;
291
292         getline(inputFile, line);
293         istream script(line);
294         ostream scriptStream;
295         scriptStream << script.rdbuf(); // 将 script 中的所有字符写入到 ostream 中
296         out.script = scriptStream.str(); // 将 ostream 中的内容赋值给 out.script
297         outputIndex++;
298     }
299
300     // Reset input and output index for the next transaction
301     inputIndex = 0;
302     outputIndex = 0;
303
304     // Increment transaction index
305     transactionIndex++;
306 }
307 }
308     inputFile.close();
309 }
310     return recoverBLK;
311 }
312
313 transaction recover_tsc(string fileName){
314     transaction tx; // 根据 ClientMSG.txt 复原一个 transaction tx
315
316     ifstream inputFile(fileName);
317
318     if (inputFile.is_open()) {
319         string line;
320         unsigned long long inputIndex = 0;
321         unsigned long long outputIndex = 0;
322
323         getline(inputFile, line); // Skip transaction request line
324
325         // Read transaction information
326         getline(inputFile, line); // Skip transaction info line
327         getline(inputFile, line); // Skip transaction height
328
329         getline(inputFile, line);
330         istream txid(line);

```

```
331         txid >> tx.txid;
332
333         getline(inputFile, line);
334         istringstream inputCount(line);
335         inputCount >> tx.input_count;
336
337         getline(inputFile, line);
338         istringstream outputCount(line);
339         outputCount >> tx.output_count;
340
341         getline(inputFile, line);
342         istringstream isCoinbase(line);
343         isCoinbase >> tx.is_coinbase;
344
345         // Read inputs
346         for (int i = 0; i < tx.input_count; i++) {
347             getline(inputFile, line); // Skip empty line
348             getline(inputFile, line); // Skip input info line
349             input& in = tx.inputs[inputIndex];
350
351             getline(inputFile, line);
352             istringstream pre_block(line);
353             pre_block >> in.pre_block;
354
355             getline(inputFile, line);
356             istringstream prevTxID(line);
357             prevTxID >> in.prevTxID;
358
359             getline(inputFile, line);
360             istringstream prevTxOutIndex(line);
361             prevTxOutIndex >> in.prevTxOutIndex;
362
363             getline(inputFile, line);
364             istringstream scriptSig(line);
365             ostringstream scriptSigStream;
366             scriptSigStream << scriptSig.rdbuf(); // 将 scriptSig 中的所有字符写入到 ostringstream 中
367             in.scriptSig = scriptSigStream.str(); // 将 ostringstream 中的内容赋值给 in.scriptSig
368             inputIndex++;
369         }
370         // Read outputs
371         for (int i = 0; i < tx.output_count; i++) {
372             getline(inputFile, line); // Skip empty line
```



```

373         getline(inputFile, line); // Skip output info line
374         output& out = tx.outputs[outputIndex];
375
376         getline(inputFile, line);
377         istringstream txid(line);
378         txid >> out.txid;
379
380         getline(inputFile, line);
381         istringstream index(line);
382         index >> out.index;
383
384         getline(inputFile, line);
385         istringstream value(line);
386         value >> out.value;
387
388         getline(inputFile, line);
389         istringstream script(line);
390         ostringstream scriptStream;
391         scriptStream << script.rdbuf(); // 将 script 中的所有字符写入到 ostringstream 中
392         out.script = scriptStream.str(); // 将 ostringstream 中的内容赋值给 out.script
393         outputIndex++;
394     }
395     inputFile.close();
396 }
397 return tx;
398 }
399
400
401 int main(){
402     block *serverBlock = InitServerBlock();
403     block *tail = serverBlock;
404     int blockMessageNumber = 1; // 计数，统计一共发送过几次区块消息，从而给区块消息文件命名
405     while(1){
406         clientMessageQueue = getTxtFileNames(folderPath1, processedFiles1);
407         blockMessageQueue = getTxtFileNames(folderPath2, processedFiles2);
408         if (isWinner()){ // 中奖几率在 0.01-0.1 之间
409             /* 从“交易池”中取出 n (>=1) 个交易，组成一个区块 newBLK。
410              newBLK 的 prevHash 等于本节点区块链表最后一个区块 lastBLK 的 hash 值。
411              newBLK 的 hash 值可以采用一个随机函数来生成；
412              height 值为 lastBLK 的 height+1；merkleRoot 和 nonce 都为空。
413              newBLK 中的交易集合由上述 n 个交易构成。
414              将 newBLK 插入本节点的区块链表末尾。

```

```

415     将 newBLK 以某个格式（比如 JSON）组成字符串“发送”给另一个区块链节点的“区块消息队列”。
416     */
417     if (transactionPool.empty()){
418         cout << "No transaction in the transaction pool!" << endl;
419         this_thread::sleep_for(chrono::seconds(5)); //隔一会儿再执行下一趟循环，避免 server 过载
420         continue;
421     }
422     else{
423         block* newBLK = createBlock(tail, 1);
424         tail->next = newBLK;
425         tail = newBLK;
426         cout << "Jackpot! A new block has been inserted to the chain!" << endl;
427         sendBlockMessage(blockMessageNumber, newBLK);
428     }
429 }
430 else{//没有中奖
431     if (!blockMessageQueue.empty()){ //“区块消息队列”不为空
432         //从“区块消息队列”头部取出一个消息（内容为区块）
433         //判断该区块是否与已有区块冲突（即是否存在一个区块，与新区块的 preHash 相同。）
434         string firstBlockMSG = blockMessageQueue.front(); //从“区块消息队列”头部取出一个消息
435         blockMessageQueue.pop();
436         processedFiles2[firstBlockMSG] = true; //将此区块消息标记为被处理过
437         string filePath_BMSG = "block_chain_server1/blockMessage/"+firstBlockMSG;
438         block *firstBLK = recoverBlock(filePath_BMSG); //将该消息恢复成一个区块 firstBLK
439
440         if (judgeConflictBlock(firstBLK, serverBlock)){//“冲突”则丢弃该区块
441             cout << "Conflict! The block has been discarded!" << endl;
442             delete firstBLK;
443         }
444         else{
445             //将该区块插入到本节点区块链表末尾；
446             firstBLK->next = tail->next;
447             tail->next = firstBLK;
448             tail = firstBLK;
449             cout << "The block has been inserted to the chain!" << endl;
450             //从“交易池”中删除该区块中包含的交易；
451             int eraseNum = 0;
452             while (firstBLK->transactions[eraseNum].txid != ""){
453                 transactionPool.erase(firstBLK->transactions[eraseNum].txid);
454                 eraseNum++;
455             }
456             cout << "Correspondent transactions deleted in the transaction pool!" << endl;

```

```

457     }
458 }//区块消息队列不为空
459 else{//“区块消息队列”为空，从“客户消息队列”头部取出一个消息 MSG;
460     if (!clientMessageQueue.empty()){
461         string firstClientMSG = clientMessageQueue.front();//从“客户消息队列”头部取出一个
462         string filePath_CMSG = "/Users/gongshukai/Desktop/SCHOOL WORK/SOPHOMORE SEM1/DATA
463         //如果 filePath_CMSG 内容为空，执行下一趟循环
464         if (judge_ClientMSG(filePath_CMSG) == 2){
465             this_thread::sleep_for(chrono::seconds(5));//隔一会儿再执行下一趟循环，避免 ser
466             continue;
467         }
468         clientMessageQueue.pop();
469         processedFiles1[firstClientMSG] = true;//将此客户消息标记为被处理过
470         if (judge_ClientMSG(filePath_CMSG) == 1){//MSG 是交易
471             cout << "Client's transaction request!" << endl;
472             transaction tsc = recover_tsc(filePath_CMSG);
473             if (!find_tsc_in_tscPool(tsc)){//“交易池”不包含该交易，将该交易加入“交易池”;
474                 cout << "transaction added to the transaction pool!" << endl;
475                 transactionPool.insert(pair<string, transaction>(tsc.txid, tsc));//加入交
476             }
477             else{//丢弃该交易
478                 cout << "The transaction has been discarded!" << endl;
479             }
480         }
481         else if (judge_ClientMSG(filePath_CMSG) == 3){//MSG 是查询
482             //在本节点维护的区块链表中执行查询;
483             //将查询结果输出在屏幕上;
484             cout << "Client's inquiry request!" << endl;
485             inquiryServerBlock(serverBlock, filePath_CMSG);
486         }
487         else{}
488     }
489 }//区块消息队列为空，客户消息队列不为空
490 }//中奖 or 没有中奖
491 this_thread::sleep_for(chrono::seconds(5));//隔一会儿再执行下一趟循环，避免 server 过载
492 }//while(1)
493 }

```

client.cpp

```

1  #include "Block_Chain.h"
2  #include <random>
3

```

```

4  int getRandom(int start, int end){
5      random_device rd;
6      mt19937 gen(rd());
7      uniform_int_distribution<> dis(start, end);
8      return dis(gen);
9  }
10
11 void sendTransaction(int fileNumber, int server, block *firstblock, block *endblock){
12     cout << "Visit Server " << server << endl;
13     string fileName = "block_chain_server" + to_string(server) + "/clientMessage/clientMessage" + to_
14     ofstream outputfile;
15     outputfile.open(fileName);
16
17     if (outputfile.is_open()){
18         string client_txid;
19         cout << "Input txid: ";
20         cin >> client_txid;
21
22         block *p = firstblock;
23         int find = 0; //找到交易信息则 find = 1, 否则 find = 0
24         int i = 0;
25         int j = 0;
26         int k = 0;
27         while (p != endblock){
28             while (p->transactions[i].txid != ""){
29                 if (p->transactions[i].txid == client_txid){
30                     outputfile << "Transaction Request" << "\n";
31                     outputfile << "transaction" << i << "info" << "\n";
32                     outputfile << p->height << "\n";
33                     outputfile << p->transactions[i].txid << "\n";
34                     outputfile << p->transactions[i].input_count << "\n";
35                     outputfile << p->transactions[i].output_count << "\n";
36                     outputfile << p->transactions[i].is_coinbase << "\n";
37                     while (p->transactions[i].inputs[j].scriptSig != ""){
38                         outputfile << "\n";
39                         outputfile << "input" << j << "info" << "\n";
40                         outputfile << p->transactions[i].inputs[j].pre_block << "\n";
41                         outputfile << p->transactions[i].inputs[j].prevTxID << "\n";
42                         outputfile << p->transactions[i].inputs[j].prevTxOutIndex << "\n";
43                         outputfile << p->transactions[i].inputs[j].scriptSig << "\n";
44                         j++;
45                     }

```

```

46         while (p->transactions[i].outputs[k].script != ""){
47             outputfile << "\n";
48             outputfile << "output" << k << "info"<< "\n";
49             outputfile << p->transactions[i].outputs[k].txid << "\n";
50             outputfile << p->transactions[i].outputs[k].index << "\n";
51             outputfile << p->transactions[i].outputs[k].value << "\n";
52             outputfile << p->transactions[i].outputs[k].script << "\n";
53             k++;
54         }
55         find = 1;//找到这条交易记录
56         break;
57     }
58     i++;
59 }
60 if (find == 1) break;
61 else{
62     i = 0;
63     p = p->next;
64 }
65 }
66 if (find == 1){
67     cout << "Message Successfully Sent!" << endl;
68     outputfile.close();
69 }
70 else{
71     cout << "Transaction Not Found!" << endl;
72 }
73 }
74 else{
75     cout << "Unable to open file";
76 }
77 }
78
79 void sendInquiry(int fileNumber, int server, int category, string content){
80     cout << "Visit Server " << server << endl;
81     string fileName = "block_chain_server" + to_string(server) + "/clientMessage/clientMessage" + to_
82     ofstream outputfile;
83     outputfile.open(fileName);
84     if (category == 1){//根据 height 查询
85         outputfile << "height" << "\n";
86         outputfile << content << "\n";
87         cout << "Message Successfully Sent!" << endl;

```

```
88         outputfile.close();
89     }
90     else if (category == 2){//根据 hash 查询
91         outputfile << "hash" << "\n";
92         outputfile << content << "\n";
93         cout << "Message Successfully Sent!" << endl;
94         outputfile.close();
95     }
96     else{//根据 txid 查询
97         outputfile << "txid" << "\n";
98         outputfile << content << "\n";
99         cout << "Message Successfully Sent!" << endl;
100        outputfile.close();
101    }
102 }
103
104 int main(){
105     int request = 0;
106     int request2 = 0;
107     int fileNumber1 = 1;
108     int fileNumber2 = 1;
109     block* firstBlock = InitBlockChain();
110     while(1){
111         cout << "Input your request: 1 Transaction; 2 Inquiry" << endl;
112         cin >> request;
113         if (request == 1){
114             int server = getRandom(1, 2);
115             if (server == 1){
116                 sendTransaction(fileNumber1, 1, firstBlock, nullptr);
117                 fileNumber1++;
118             }
119             else{
120                 sendTransaction(fileNumber2, 2, firstBlock, nullptr);
121                 fileNumber2++;
122             }
123         }
124         else if (request == 2){
125             cout << "Inquiry Type: 1 Search by height; 2 Search by hash; 3 Search by txid" << endl;
126             cin >> request2;
127             if (request2 == 1){//按照 height 查询
128                 cout << "Input server: " << endl;
129                 int server;
```

```
130         cin >> server;
131         string height;
132         cout << "Input height: " << endl;
133         cin >> height;
134         if (server == 1){
135             sendInquiry(fileNumber1, server, 1, height);
136             fileNumber1++;
137         }
138         else{
139             sendInquiry(fileNumber2, server, 1, height);
140             fileNumber2++;
141         }
142     }
143     else if (request2 == 2){//按照 hash 查询
144         cout << "Input server: " << endl;
145         int server;
146         cin >> server;
147         string hash;
148         cout << "Input hash: " << endl;
149         cin >> hash;
150
151         if (server == 1){
152             sendInquiry(fileNumber1, server, 2, hash);
153             fileNumber1++;
154         }
155         else{
156             sendInquiry(fileNumber2, server, 2, hash);
157             fileNumber2++;
158         }
159     }
160     else{//按照 txid 查询
161         cout << "Input server: " << endl;
162         int server;
163         cin >> server;
164         string txid;
165         cout << "Input txid: " << endl;
166         cin >> txid;
167
168         if (server == 1){
169             sendInquiry(fileNumber1, server, 3, txid);
170             fileNumber1++;
171         }
```

```
172         else{
173             sendInquiry(fileNumber2, server, 3, txid);
174             fileNumber2++;
175         }
176     }
177 }
178 else{
179     cout << "Invalid Request! Please input again!" << endl;
180 }
181 }
182 }
```
