学号：2022201538 姓名：白锦峄 完成日期：2023年12月14日

实验报告

1. 需求分析

1.1 任务说明

该程序的主要任务是实现一个简单的区块链数据处理系统。具体要求如下：

输入：

- 区块信息文件：用于初始化区块链链表。
- 交易信息文件：用于初始化用户交易记录和构建交易关系图。

输出：

- 查询指定账号在一个时间段内的所有转入或转出记录。
- 查询某个账号在某个时刻的金额。
- 在某个时刻的福布斯富豪榜。
- 构建交易关系图，统计平均出度和入度。
- 检查交易关系图中是否存在环。
- 给定一个账号A，求A到其他所有账号的最短路径。

测试数据：

1. 正确输入：
    - 提供正确的区块信息文件和交易信息文件。
    - 查询存在的账号在指定时间段内的转入或转出记录。
    - 查询存在的账号在指定时刻的金额。
    - 查询在某个时刻的福布斯富豪榜。
2. 含有错误的输入：
    - 类型错误
    - 数据范围错误
    - 查询不存在的账号或非法时间段。
    - 查询不存在的时刻的账号金额。
    - 构建包含错误数据的交易关系图。

2. 概要设计

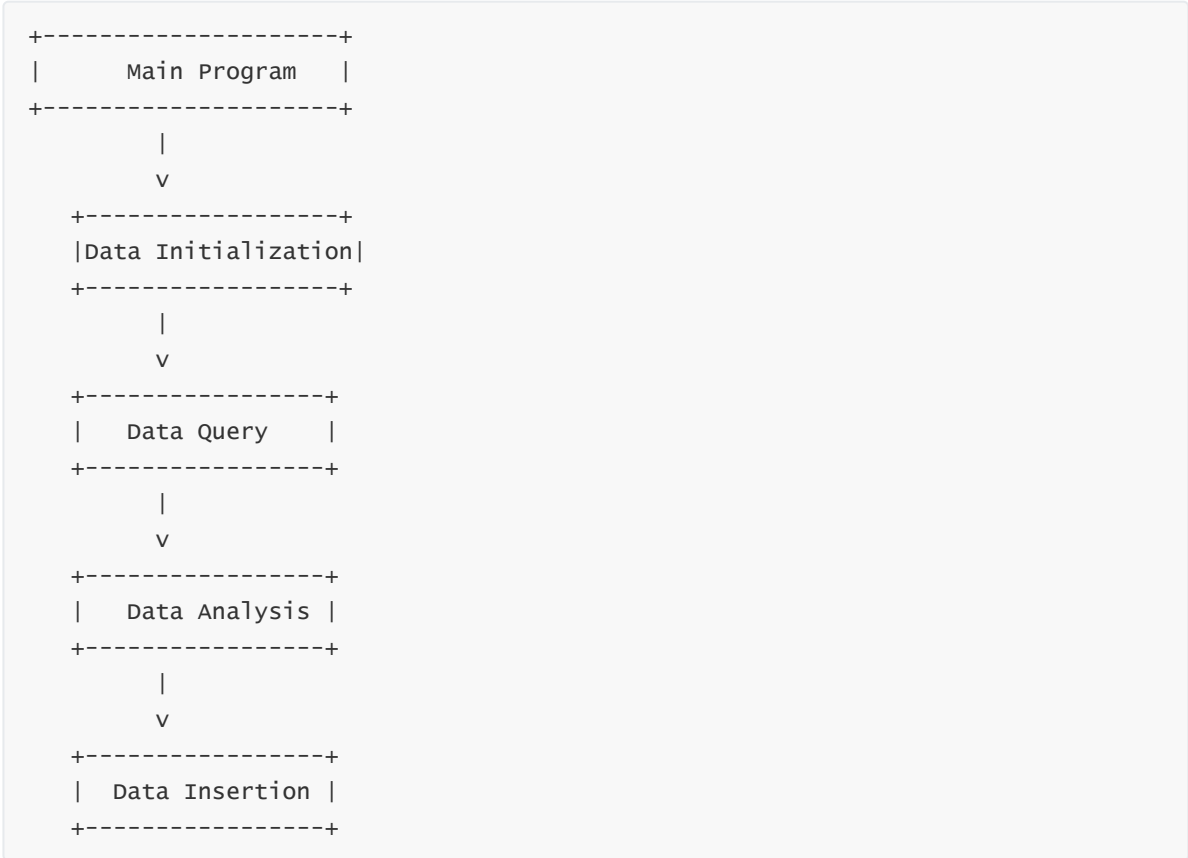
2.1 抽象数据类型

- 区块链链表
- 交易结构体
- 用户结构体
- 用户交易结构体
- 图节点
- 输入输出数据
- 比较结构体：`cmp_money`，`cmp`，`cmp_reversed`

2.2 主程序流程

1. 读取区块信息，初始化区块链链表。
2. 读取交易信息，初始化用户交易记录。
3. 构建交易关系图。
4. 提供用户交互界面，实现数据查询、分析和插入操作。
5. 输出相应结果。

2.3 模块调用关系图



3. 详细设计

3.1 数据初始化

3.1.1 读取区块信息

```
read_block(file_path)
1. 打开文件。
2. 跳过第一行。
3. 逐行读取，解析并创建区块结构体。
4. 关闭文件。
```

3.1.2 读取交易信息

```
read_transaction(file_path, skip_header)
1. 打开文件。
2. 判断是否需要跳过第一行。
3. 逐行读取，解析并创建交易结构体。
4. 将交易信息添加到用户映射表。
5. 关闭文件。
```

3.1.3 构建交易关系图

```
get_graph()
1. 对每个用户交易记录：
   a. 为每个交易创建图节点。
   b. 更新节点的入度和出度。
```

3.2 数据查询

3.2.1 查询指定账号在一个时间段内的所有转入或转出记录

```
count_trans_number(account, start_time, end_time, k)
1. 遍历用户交易记录。
2. 根据时间范围过滤交易。
3. 按金额降序排序交易。
4. 输出前 k 条交易记录。
```

3.2.2 查询某个账号在某个时刻的金额

```
balance_now(account, timestamp)
1. 遍历用户交易记录。
2. 计算指定时刻的账户余额。
3. 输出余额。
```

3.2.3 在某个时刻的福布斯富豪榜

```
fbsb(timestamp, k)
1. 遍历所有用户。
2. 计算指定时刻用户的余额。
3. 输出前 k 位富豪。
```

3.3 数据分析

3.3.1 构建交易关系图

```
build_transaction_graph()
1. 对每个用户交易记录：
   a. 为每个交易创建图节点。
   b. 更新节点的入度和出度。
```

3.3.2 统计交易关系图的平均出度、入度

```
calculate_avg_degree()
1. 计算每个节点的出度和入度。
2. 计算平均出度和入度。
3. 输出结果。
```

3.3.3 检查交易关系图中是否存在环

```
check_cycle()
1. 初始化一个集合，用于存储已访问节点。
2. 对每个节点执行深度优先搜索。
3. 如果在搜索过程中发现已访问节点，说明存在环。
4. 输出结果。
```

3.3.4 给定一个账号A，求A到其他所有账号的最短路径

```
shortest_path(account_A)
1. 初始化距离数组，起点距离设为0，其他节点距离初始化为无穷大。
2. 使用 Dijkstra 算法计算最短路径。
3. 输出最短路径。
```

3.4 数据插入

3.4.1 输入数据

```
input_data(file_path)
1. 用户输入文件路径。
2. 从文件中读取新的交易记录。
3. 更新用户交易记录。
4. 重新构建交易关系图。
```

4. 调试分析

4.1 调试过程

在调试过程中，主要遇到的问题和解决方法如下：

1. **问题：** 在读取区块信息时，文件的第一行需要跳过，否则可能导致错误的解析。

   解决方法： 在读取时使用 `getline` 函数，跳过第一行。

2. **问题：** 在构建交易关系图时，需要更新每个节点的入度和出度。

   解决方法： 遍历每个用户的交易记录，为每个交易创建图节点，并更新节点的入度和出度。

3. **问题：** 在查询某个账号在某个时刻的金额时，需要考虑转入和转出的金额。

   解决方法： 遍历用户的交易记录，计算在指定时刻的账户余额，考虑转入金额的增加和转出金额的减少。

4.2 算法分析与改进思路

1. **算法的时空分析：**
   - 时间复杂度：部分算法的时间复杂度较低，例如查询某个账号在某个时刻的金额的算法为 O(n)，其中 n 为交易记录数。
   - 空间复杂度：主要占用空间的是交易关系图，其空间复杂度与图的节点和边数相关，为 O(V + E)，其中 V 为节点数，E 为边数。
2. **改进思路：**
   - 在构建交易关系图时，可以考虑采用更高效的图存储结构，以提高查询和分析效率。
   - 对于查询操作，可以通过建立索引等方式进行优化，减少遍历的时间复杂度。

4.3 经验和体会

在编写和调试过程中，深入理解每个功能的实现细节对于解决问题非常关键。

及时输出中间结果，对每个函数的输入输出进行检查，有助于快速定位和解决问题。

同时，对于复杂的算法和数据结构，需要充分理解其原理和应用场景，以便更好地调试和优化代码。

5. 用户使用说明

1. **数据初始化**：
   - 使用 `read_block` 函数读取区块信息。
   - 使用 `read_transaction` 函数读取交易信息。
2. **数据查询**：
   - 使用 `count_trans_number` 函数查询指定账号在特定时间范围内的所有转入或转出记录。
   - 使用 `balance_now` 函数查询某个账号在指定时刻的金额。
   - 使用 `fbsb` 函数查询在指定时刻的福布斯富豪榜单。
3. **数据分析**：
   - 使用 `build_transaction_graph` 函数构建交易关系图。
   - 使用 `calculate_avg_degree` 函数统计交易关系图的平均出度和入度。
   - 使用 `check_cycle` 函数检查交易关系图中是否存在环。
   - 使用 `shortest_path` 函数给定一个账号A，求A到其他所有账号的最短路径。
4. **数据插入**：
   - 使用 `input_data` 函数从文件中读取新的交易记录。

6. 测试结果

各部分均能按要求完成输入输出处理，详情见测试报告。

7. 附录

```
#include <bits/stdc++.h>
#include <chrono>
using namespace std;

// 定义交易结构体
struct transaction
{
    long long id = 0;
    int blockID = 0;
    double amount = 0;
    string from;
    string to;
};

// 定义用户交易结构体
struct user_trans
{
    int type = 0; // 1是from，0是to;
    long long time;
    double amount;
    string from;
    string to;
};
```

```
// 定义区块结构体
struct block
{
    int blockID = 0;
    long long block_timestamp = 0;
    string hash = "";
    block *next = nullptr;
    vector<transaction> trans_list;
};

// 定义用户结构体
struct user
{
    string name;
    vector<user_trans> translist;
};

// 用于比较user_trans结构体的金额大小
struct cmp_money
{
    bool operator()(const user_trans &lhs, const user_trans &rhs) const
    {
        return lhs.amount <= rhs.amount;
    }
};

// 用于比较pair<string, double>的大小
struct cmp
{
    bool operator()(const pair<string, double> &lhs, const pair<string, double>
&rhs) const
    {
        return lhs.second >= rhs.second;
    }
};

// 用于比较pair<string, double>的大小（逆序）
struct cmp_reversed
{
    bool operator()(const pair<string, double> &lhs, const pair<string, double>
&rhs) const
    {
        return lhs.second >= rhs.second;
    }
};

// 定义图中的节点
struct node
{
    string to;
    double weight;
};

// 存储每个用户的收支情况和相关节点
struct inandout
{
    double in;
```

```
    double out;
};

// 存储用户信息的map
map<string, user> usermap;

// 存储区块时间戳的map
map<int, long long> block_time;

// 存储已经访问的用户的集合
set<string> sign_set;

// 存储图中节点的信息
map<string, pair<inandout, vector<node>>> nodes;

// 存储已经访问的节点的集合


// 读取区块信息
block *read_block(string filename)
{
    ifstream fin(filename);
    string line;
    block *blocks = new block;
    block *head = blocks;
    getline(fin, line);
    getline(fin, line);
    stringstream ss1(line);
    string temp1;
    getline(ss1, temp1, ',');
    blocks->blockID = stoi(temp1);
    getline(ss1, temp1, ',');
    blocks->hash = temp1;
    getline(ss1, temp1, ',');
    blocks->block_timestamp = stoll(temp1);
    block_time[blocks->blockID] = blocks->block_timestamp;
    int count = 0;
    while (getline(fin, line))
    {
        count++;
        block *temp_block = new block;
        stringstream ss(line);
        string temp;
        getline(ss, temp, ',');
        temp_block->blockID = stoi(temp);
        getline(ss, temp, ',');
        temp_block->hash = temp;
        getline(ss, temp, ',');
        temp_block->block_timestamp = stoll(temp);
        block_time[temp_block->blockID] = temp_block->block_timestamp;
        blocks->next = temp_block;
        blocks = blocks->next;
    }
    return head;
}

// 读取交易信息
void read_transaction(string filename, int init)
```

```
{
    ifstream fin(filename);
    string line;
    getline(fin, line);
    int count = 0;
    int time_find_block = 0;
    while (getline(fin, line))
    {
        count++;
        transaction trans;
        stringstream ss(line);
        string temp;
        if (init) //判断是否跳过首行
        {
            getline(ss, temp, ',');
            trans.id = stoll(temp);
            getline(ss, temp, ',');
            trans.blockID = stoi(temp);
            getline(ss, temp, ',');
            trans.from = temp;
            getline(ss, temp, ',');
            trans.amount = stod(temp);
            getline(ss, temp, ',');
            trans.to = temp;
        }

        user_trans t;
        t.amount = trans.amount;
        t.from = trans.from;
        t.time = block_time[trans.blockID];
        t.to = trans.to;
        usermap[trans.to].translist.push_back(t);
        t.type = 1;
        usermap[trans.from].translist.push_back(t);
    }
}

// 计算用户指定时间范围内的交易数量，并输出前k条交易信息
int count_trans_number(string name, long long start_time, long long end_time,
int k)
{
    user users = usermap[name];
    int count = 0;
    priority_queue<user_trans, vector<user_trans>, cmp_money> q;
    cout << "Username:" << name << endl;
    for (const auto &trans : users.translist)
    {
        if (trans.time >= start_time && trans.time <= end_time)
        {
            count++;
            q.push(trans);
        }
    }
    cout << "The top" << k << "transactions:" << endl;
    for (int i = 1; i <= k; i++)
    {
        if (q.empty())
            break;
```

```
        auto t = q.top();
        q.pop();
        cout << "rank: " << i << " time: " << t.time << " amount: " << t.amount
<< endl;
        cout << "from: " << t.from << " to " << t.to << endl;
    }
    return count;
}

// 计算某一时刻用户的余额，等于之前的所有交易的累计
double balance_now(string name, long long time)
{
    user users = usermap[name];
    double count = 0;
    for (const auto &trans : users.translist)
    {
        if (trans.time <= time)
        {
            if (trans.type == 1)
                count -= trans.amount;
            else
                count += trans.amount;
        }
    }
    return count;
}

// 用于比较pair<string, double>的大小（逆序）
struct ValueComparator
{
    bool operator()(const pair<string, double> &lhs,
                    const pair<string, double> &rhs) const
    {
        return lhs.second >= rhs.second;
    }
};

// 输出某一时刻前k名用户的余额情况
void fbsb(long long time, int k = 10)
{
    priority_queue<pair<string, double>, vector<pair<string, double>>, cmp>
user_money;
    for (const auto &name : usermap)
    {
        double money = 0;
        for (const auto &trans : name.second.translist)
        {
            if (trans.time <= time)
            {
                if (trans.type == 1)
                    money -= trans.amount;
                else
                    money += trans.amount;
            }
        }
        user_money.push(make_pair(name.first, money));
        if (user_money.size() > k)
        {
```

```
            user_money.pop();
        }
    }
    stack<pair<string, double>> out;
    while (!user_money.empty())
    {
        out.push(user_money.top());
        user_money.pop();
    }
    for (int i = 0; i < k; i++)
    {
        if (!out.empty())
        {
            cout << "rank:" << i + 1 << " name: " << out.top().first << " money:
" << out.top().second << endl;
            out.pop();
        }
        else
            break;
    }
}

// 构建图的节点信息
void get_graph()
{
    for (const auto &users : usermap)
    {
        map<string, double> temp;
        for (const auto &trans : users.second.translist)
        {
            if (trans.type)
            {
                temp[trans.to] += trans.amount;
            }
        }
        for (const auto &t : temp)
        {
            node n;
            n.to = t.first;
            n.weight = t.second;
            nodes[users.first].second.push_back(n);
            nodes[users.first].first.out++;
            nodes[t.first].first.in++;
        }
    }
}

// 输出图中出度与入度的比值前k名的节点信息
void top_out_div_out(int k = 10)
{
    priority_queue<pair<string, double>, vector<pair<string, double>>, cmp> q;
    int out_count = 0;
    int in_count = 0;
    int count = 0;
    for (const auto &n : nodes)
    {
        count++;
        out_count += n.second.first.out;
```

```
        in_count += n.second.first.in;
        if (n.second.first.in != 0)
        {
            q.push(make_pair(n.first, n.second.first.out / n.second.first.in));
            if (q.size() > k)
            {
                q.pop();
            }
        }
    }
    cout << "The average out_count:" << 1.0 * out_count / count << endl;
    cout << "The average in_count:" << 1.0 * in_count / count << endl;
    stack<pair<string, double>> out;
    while (!q.empty())
    {
        out.push(q.top());
        q.pop();
    }
    for (int i = 0; i < k; i++)
    {
        if (!out.empty())
        {
            cout << "rank:" << i + 1 << " name: " << out.top().first << " money:
" << out.top().second << endl;
            out.pop();
        }
        else
            break;
    }
}

// 寻找图中是否存在环
int find_circle()
{
    stack<string> stk;
    unordered_set<string> s;
    stk.push(nodes.begin()->first);
    s.insert(nodes.begin()->first);
    while (!stk.empty())
    {
        string name = stk.top();
        stk.pop();
        for (const auto &n : nodes[name].second)
        {
            if (s.find(n.to) != s.end())
            {
                return 1;
            }
            else{
                stk.push(n.to);
                s.insert(n.to);
                }
        }
    }
    return 0;
}

// 使用Dijkstra算法计算从指定节点到其他节点的最短路径
```

```
void dijsktra(string name)
{
    priority_queue<pair<string, double>, vector<pair<string, double>>,
cmp_reversed> q;
    unordered_set<string> sign;
    q.push(make_pair(name, 0));
    map<string, double> dist;
    while (!q.empty())
    {
        auto t = q.top();
        q.pop();
        if (sign.find(t.first) != sign.end())
        {
            continue;
        }
        sign.insert(t.first);
        auto temp = nodes[t.first];
        for (const auto &n : nodes[t.first].second)
        {
            if (dist.find(n.to) == dist.end()) // 如果是一个新的结点。
            {
                dist[n.to] = dist[t.first] + n.weight;
            }
            else // 如果已经访问过了
            {
                dist[n.to] = min(dist[n.to], dist[t.first] + n.weight);
            }
            q.push(make_pair(n.to, dist[n.to]));
        }
    }
    for (const auto &n : dist)
    {   if(n.second!=0)
        cout << "nodename: " << n.first << " node_dist: " << n.second << endl;
    }
}

void data_search(int i)
{
    if (i == 1)
    {
        string name;
        long long start_time;
        long long end_time;
        int k;
        while (true)
        {   cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');//清除缓
冲区
            std::cout << "Enter 'start_time': ";
            if (!(std::cin >> start_time))
            {
                std::cerr << "Invalid input. Please try again'." << std::endl;
                std::cin.clear();
                continue;
            }

            std::cout << "Enter 'end_time': ";
            if (!(std::cin >> end_time) || end_time < start_time)
            {
```

```
                std::cerr << "Invalid input. Please try again." << std::endl;
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
                continue;
            }

            std::cout << "Enter 'k'(k>0): ";
            if (!(std::cin >> k) || k <= 0)
            {
                std::cerr << "Invalid input. Please try again." << std::endl;
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
                continue;
            }
            break;
        }
        std::cout << "Enter 'name': ";
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        cin >> name;
        int t=count_trans_number(name, start_time, end_time, k);
        cout<<"total transaction number:"<<t<<endl;
        return;
    }
    if (i == 2)
    {

        cout << "please input the user and time you want to search:" << endl;
        string name;
        long long time = 0;
        cout << "please input name:" << endl;
        cin >> name;
        cout << "please input time:" << endl;
        cin >> time;
        while (1)
        {
            if (cin.fail())
            {
                cin.clear();
                cout << "invalid time, please try again" << endl;
                cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');//
清除缓冲区
                cin >> time;
                continue;
            }
            else
                break;
        }
        cout << "balance now:" << balance_now(name, time) << endl;
        return;
    }
    if (i == 3)
    {   int k;
        long long time;
        cout<<"please input time:"<<endl;
        while(1){
            cin>>time;
```

```
            if(cin.fail()){
                cin.clear();
                cout<<"invalid time, please try again"<<endl;
                cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');//
清除缓冲区
                continue;
            }
            else
                break;
        }
        cout << "please input k:" << endl;
        cin >> k;
        while (1)
        {
            if (cin.fail())
            {
                cin.clear();
                cout << "invalid k, please try again" << endl;
                cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');//
清除缓冲区
                cin >> k;
                continue;
            }
            else
                break;
        }
        fbsb(time, k);
    }
    else{
        cout<<"invalid, please try again"<<endl;
    }
    return;
}

void data_analysis(int i)
{if(i==2){
    cout<<"please input k:"<<endl;
    int k;
    cin>>k;
    while(1){
        if(cin.fail()){
            cin.clear();
            cout<<"invalid k, please try again"<<endl;
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');//清除缓
冲区
            cin>>k;
            continue;
        }
        else
            break;
    }
    top_out_div_out(k);
}
if(i==3){
    if(find_circle()){
        cout<<"YES"<<endl;
    }
    else{
```

```
        cout<<"NO"<<endl;
    }
}
if(i==4){
    string s;
    cout<<"please input the name you want to search:"<<endl;
    cin>>s;
    dijsktra(s);
}
return ;
}

void input_data()
{   cout<<"please input the file path"<<endl;
    string s;
    cin>>s;
    int i;
    cout<<"is there the headline?1yes 0 no"<<endl;
    cin>>i;
    while(1){
        if(cin.fail()||(i!=1&&i!=0)){
            cin.clear();
            cout<<"invalid type, please try again"<<endl;
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');//清除缓
冲区
            cin>>i;
            continue;
        }
        else
            break;
    }
    read_transaction(s,i);
    get_graph();
    cout<<"input data successfully!"<<endl;
}

int main()
{
    cout << "reading blocks..." << endl;
    auto start_time = chrono::high_resolution_clock::now();
    block *blocks = read_block("data/block_part1.csv");
    auto end_time = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::microseconds>(end_time -
start_time);
    cout << "Time taken by reading blocks: " << duration.count() << "
microseconds" << endl;
    cout << "reading transactions..." << endl;
    start_time = chrono::high_resolution_clock::now();
    read_transaction("data/tx_data_part1_v2.csv", 1);
    end_time = chrono::high_resolution_clock::now();
    duration = chrono::duration_cast<chrono::microseconds>(end_time -
start_time);
    cout << "Time taken by reading transactions: " << duration.count() << "
microseconds" << endl;
    cout << "getting graph..." << endl;
    start_time = chrono::high_resolution_clock::now();
    get_graph();
    end_time = chrono::high_resolution_clock::now();
```

```
    duration = chrono::duration_cast<chrono::microseconds>(end_time -
start_time);
    cout << "Time taken by getting graph: " << duration.count() << "
microseconds" << endl;
    cout << "now, please choose type:" << endl;
    cout << "input: 2:data search /3:data analysis /4:input data. /5:quit" <<
endl;
    int type;
    while (1)
    {
        cin >> type;
        if (cin.fail())
        {
            cin.clear();
            cout << "invalid input, please try again" << endl;
            cout << "input: 2:data search /3:data analysis /4:input data
/5:quit." << endl;
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');//清除缓
冲区
            continue;
        }
        start_time = chrono::high_resolution_clock::now();
        if (type == 2)
        {
            cout << "please input the type you want to search:" << endl;
            cout<<"1:search a user's top k transactions"<<endl;
            cout<<"2:search a user's balance at a time"<<endl;
            cout<<"3:search the richest people at a time"<<endl;
            int i;
            cin >> i;
            while (1)
            {
                if (cin.fail())
                {
                    cin.clear();
                    cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');//清除缓冲区
                    cout << "invalid input, please try again!" << endl;
                    cin >> i;
                    continue;
                }
                else
                    break;
            }
            data_search(i);
        }
        else if (type == 3)
        {
            cout << "please input the type you want to analysis:" << endl;
            cout<<"2:count the average out_degree/in_degree and output the top k
nodes"<<endl;
            cout<<"3:judge whether there is a circle in the graph"<<endl;
            cout<<"4:use Dijkstra algorithm to find the shortest path from a
node to other nodes"<<endl;
            int i;
            cin >> i;
            while (1)
            {
```

```
                if (cin.fail())
                {
                    cin.clear();
                    cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');//清除缓冲区
                    cout << "invalid input, please try again" << endl;
                    cin >> i;
                    continue;
                }
                else
                    break;
            }
            data_analysis(i);
        }
        else if (type == 4)
        {
            cout << "please input the data you want to input:" << endl;
            input_data();
        }
        else if (type == 5){break;}
        else
        {
            cout << "please input the right type:" << endl;
            cout << "input: 2:data search /3:data analysis /4:input data
/5:quit. " << endl;
            continue;
        }
        end_time = chrono::high_resolution_clock::now();
        duration = chrono::duration_cast<chrono::microseconds>(end_time -
start_time);
        cout << "Time taken by this operation: " << duration.count() << "
microseconds(including the time you choose options)" << endl;
        cout << "please choose your next option: input: 2:data search /3:data
analysis /4:input data /5:quit. " << endl;
    }
    return 0;
}
```