实验报告

1. 数据初始化

1.1 数据结构体选择

为了实现尽量少的存储开销和在尽量短的时间内完成数据初始化，采用了以下数据结构：

- 区块与交易：
  - 区块链链表：用于存储区块信息。
  - 交易结构体：存储每个交易的信息，包括交易ID、区块ID、金额、转出账号、转入账号。
  - 用户结构体：存储用户信息，包括用户名和用户的交易记录表。
  - 用户交易结构体：区别于原始的交易结构体，内容包括type标记是入还是出，时间戳，金额，转入和转出账号。
- 图：
  - 图节点：包括转入账号和转入金额。
  - 输入输出数据：存每个结点的出边数量和入边数量。
- 用于map的重写比较结构体：
  - `cmp_money`：用于比较用户交易结构体的金额大小。
  - `cmp`：用于比较图节点数据类型的大小。
  - `cmp_reversed`:逆序的比较图节点大小。
- 用于存储的map等：
  - `map<string, user> usermap`：存储用户信息。
  - `map<int, long long> block_time`：存储区块时间戳。
  - `set<string> sign_set`:在深度优先遍历找环时记录已访问结点。
  - `map<string, pair<inandout, vector<node>>> nodes`：存储图节点信息。
  - `unordered_set<string> s`：在`dijkstra`算法中存储已访问节点。

1.2 数据初始化流程

1.2.1 读取区块信息

实现细节

- **实现思路**：
  - 使用 `read_block` 函数读取指定文件中的区块信息，并建立区块链链表。
  - 通过区块链链表构建 `block` 结构体对象，存储区块信息，包括区块ID、哈希值、时间戳等。
- **关键步骤**：
  1. 打开文件，逐行读取区块信息。
  2. 用`getline`函数依次读入，最开始的时候有一个跳过第一行的操作。
  3. 利用 `stringstream` 解析每一行的区块信息，构建 `block` 结构体对象。

1.2.2 读取交易信息

实现细节

- **实现思路**：
  - 使用 `read_transaction` 函数读取指定文件中的交易信息，将交易信息按照时间戳和用户记录到相应的数据结构中。
  - 构建交易信息的结构体，包括交易ID、区块ID、金额、转出账号、转入账号。
- **关键步骤**：
  1. 打开文件，逐行读取交易信息。
  2. 根据输入的整数参数判断是否需要跳过首行。
  3. 利用 `stringstream` 解析每一行的交易信息，构建交易信息的结构体对象。
  4. 同时把交易存在user_map里便于查询。

1.2.3 构建交易关系图

实现细节

- **实现思路**：
  - 使用 `get_graph` 函数，在数据初始化阶段构建交易关系图。
  - 根据用户的交易记录构建交易关系图，其中节点表示用户，边表示交易关系，边权重表示交易金额。
- **关键步骤**：
  1. 遍历所有用户的交易记录。
  2. 根据转入账号和转入金额构建图中的节点，同时记录每个节点的出度和入度。

通过以上对每个函数的实现思路和关键步骤的详细说明，展示了在数据初始化流程部分中的具体实现细节。

2. 数据查询

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

实现细节

- **实现思路**：
  - 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。
  - 遍历指定账号的交易记录，筛选出符合时间段的记录，计算总记录数，并输出交易金额最大的前k条记录。
- **关键步骤**：
  1. 遍历用户交易记录，判断每条记录的时间戳是否在给定的时间范围内。
  2. 对符合时间条件的记录，将其按照交易金额降序排列，并输出前k条记录。
  3. 这里为了实现高效排序，利用一个优先队列来实现，超队列数据超过k条时去掉最小值，最后逆序输出就可以了。

2.2 查询某个账号在某个时刻的金额

实现细节

- **实现思路**：
  - 使用 `balance_now` 函数查询某个账号在特定时刻的金额。
  - 遍历指定账号的交易记录，计算在指定时刻的账户余额，包括所有转入和转出的金额。
  - 考虑到某时刻的金额等于对之前的所有交易求和，所以只需要判断时间戳即可。

- **关键步骤**：
    1. 遍历用户交易记录，判断是否是在发生时刻之前。
    2. 对在范围内的数据进行操作。
    3. 转入金额增加，转出金额减少。

2.3 在某个时刻的福布斯富豪榜

实现细节

- **实现思路**：
    - 使用 `fbsb` 函数查询在指定时刻的福布斯富豪榜单。
    - 遍历所有用户，在指定时刻计算每个用户的余额，并输出金额最多的前k个用户。
- **关键步骤**：
    1. 遍历所有用户，计算每个用户在指定时刻的余额。
    2. 利用最小堆实现对数据的动态维护，输出时借助栈实现从大到小的输出。

通过以上对每个函数的实现思路和关键步骤的详细说明，展示了在数据查询部分中的具体实现细节。

3. 数据分析

3.1 构建交易关系图

实现细节

- **思路**：
- 通过遍历每个用户的交易记录，构建交易关系图。
    - 对每笔交易，以转出账号为起点，转入账号为终点，建立图中的节点。
- 维护 `usermap` 记录每个用户的信息，包括用户名和对应的交易记录表。
- 维护 `nodes` 记录每个节点的信息，包括入边和出边数量，以及对应的交易信息。

3.2 统计交易关系图的平均出度、入度

实现细节

- **思路**：
    - 遍历交易关系图的每个节点，计算每个节点的出度与入度比值。
    - 使用一个优先队列，按照比值从大到小维护前k个节点。
    - 计算平均出度与入度，输出结果。

3.3 检查交易关系图中是否存在环

实现细节

- **思路**：
    - 利用深度优先搜索（DFS）检查交易关系图是否存在环。
    - 维护一个 `set` 记录已访问过的节点，用于避免重复访问。
    - 递归地对每个节点进行DFS，如果在DFS的过程中发现已访问的节点，说明存在环，返回 `true`。
    - 如果DFS结束后没有发现环，则返回 `false`。

3.4 给定一个账号A，求A到其他所有账号的最短路径

实现细节

- **思路**:
  - 使用Dijkstra算法计算指定账号A到其他账号的最短路径。
  - 利用优先队列实现Dijkstra算法中的最小堆，存储每个节点到起点的距离。
  - 初始化距离数组，起点距离为0，其他节点距离初始化为无穷大。
  - 通过松弛操作，逐步更新节点到起点的最短距离。
  - 输出最短路径长度及路径。

4. 数据插入

4.1 输入数据

实现细节

- **实现思路**:
  - 使用 `input_data` 函数实现数据插入，从文件中读入新的交易记录，更新交易图，然后重新执行读取交易数据函数和生成图函数。
- **关键步骤**:
  1. 用户输入文件路径，确定要插入的交易信息。
  2. 用户选择是否有数据文件的头部，根据选择调用 `read_transaction` 函数读取新的交易信息。
  3. 重新执行 `get_graph` 函数，根据更新后的交易信息构建交易关系图。

5. 代码检查

5.1 代码细节

1. 用户界面通过命令行实现，提供清晰的提示信息和选项。
2. 代码结构清晰，由上至下分别是结构体，全局变量定义，函数定义及其实现，最后是主函数。
3. 变量命名基本按照英文全称或缩写，具有描述性。
4. 对每个函数的功能都有注释，思路好懂。
5. 在每个操作后输出该操作的运行时间，提供性能信息。

5.2 代码鲁棒性

实现细节

- **实现思路**:
  - 通过对用户输入的数据进行校验，确保程序具有一定的鲁棒性，能够接受错误输入但不会导致程序停止运行。
- **关键步骤**:
  1. 在用户输入数据的地方，使用循环结构和条件判断来对输入数据进行校验，保证输入的合法性。
  2. 对可能导致程序崩溃的地方进行异常处理，以防止错误的输入或者操作导致程序异常退出。