数据结构与算法 I 实验报告

实验 6：区块链大数据分析

龚舒凯 2022202790 应用经济-数据科学实验班

https://github.com/GONGSHUKAI

2023 年 12 月 15 日

区块链大数据分析

1 需求分析

一个简单的区块数据结构如下图所示，主要由 4 个部分组成：

- **时间戳：**记录该区块创建的时间。
- **Hash:** 每个区块的 hash 值相当于该区块的指纹。在上面的示例中，只要区块内容被改变，那么该区块生成的 hash 也会发生变化，该 hash 值与下一区块存放的 prev hash 不一致，这样就能感知到数据被篡改。以此类推，所有的区块被串起来了，形成一个链。
- **Prev hash 值:** 该区块的前一区块的 hash 值，该 hash 值由前一区块的内容生成，用来检验前一个区块是否被篡改。本实验中的 hash 值由 prev hash 和 nonce 共同生成，即：MD5(concat(prevhash, nonce))。
- **Nonce:** 随机数，满足一定条件的大整数，即比特币“挖矿”的成果。
- **Tx:** 属于该区块的交易记录，一般一个区块可以包含 1000 个交易左右。每个区块中保存大约 1000 个交易记录。新增的区块只能添加到链尾。
  - From 和 To 存储的是唯一的账号。每个账号是由数字和字母组成的字符串。
  - Amount 记录该笔交易的金额。

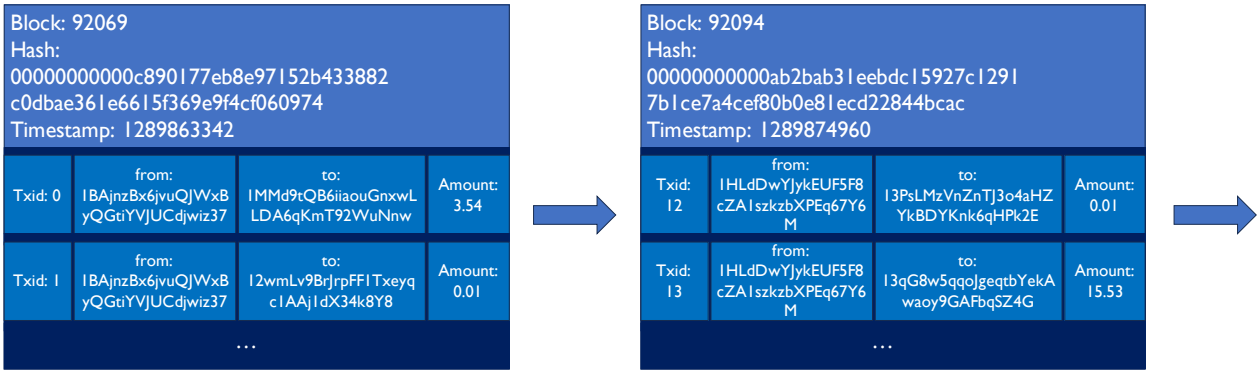

图 1: 区块链结构

要求设计并实现一个区块链数据处理程序，以便对区块链中包含的大量交易记录进行分析，进而发现一些规律。

1.1 基本要求

1. **数据初始化:** 从指定文件 (约 100M) 读入数据，初始化区块链。数据格式参见《数据集说明》。要求用尽量少的存储开销，在尽量短的时间内完成数据初始化。数据结构需记录每个区块的生成时间，要求一个区块中所有的交易按二叉树或 B-树形式组织。
2. **数据查询**

- 查找指定账号在一个时间段内的所有转入或转出记录，返回总记录数，交易金额最大的前 k 条记录 (k 为一个正整数，由查询输入)
- 查询某个账号在某个时刻的金额 (允许有负数)
- 在某个时刻的福布斯富豪榜! 输出在该时刻最有钱的前 k 个用户，k 默认值为 50，可以由用户修改 k 值。

3. **数据分析**

- 构建交易关系图。若账号 A 曾给 B 转账，则 A 到 B 有一个弧，弧上有权重，表示 A 给 B 累计转账的金额。
- 统计交易关系图的平均出度、入度。显示出度或入度最高的前 k 个帐号。
- 检查交易关系图中是否存在环，输出 YES 或 NO。
- 给定一个账号 A，求 A 到其他所有账号的最短路径。路径长度为路径上所有弧的权重之和。不存在路径的，不用输出。

4. **数据插入**

- 从文件中读入新交易记录 (用户界面需支持输入文件路径)，在现有的交易图上增加新的交易数据，然后重新执行功能 2 和功能 3。

2 概要设计

2.1 抽象数据类型 (ADT) 的定义

由于区块链大数据分析涉及到的数据量极大，在本实验中，好的抽象数据类型设计可以优化程序的运行效率，减少读取、查找数据的时间。为加速程序运行，本实验大量使用了 C++ stl 标准库中的unordered_map数据结构来辅助区块链数据结构的定义，保证数据访问的时间复杂度为 $O(1)$。以下为区块链大数据分析所需抽象数据类型的定义：

1. **区块与区块链:**

   (a) **区块 (block)：**由于每个区块都由其 blockID 唯一确定，因此设置 blockID 和 blockInfo 的键值对，访问区块时直接通过访问键对应的 blockInfo 即可或者区块中的信息。

```
unordered_map <unsigned long long, blockInfo> blocks;
```

   这里 blockID 的数据类型是 unsigned long long, blockInfo 的数据结构是自定义的 blockInfo。

   (b) **区块信息 (blockInfo):** 区块信息包括区块的 hash 值、时间戳 (Timestamp)、交易记录 (transactions) 等信息，其中交易 (transactions) 也是一个无序映射。只需访问交易的特定的键 (txid) 即可得知该交易的信息。因此定义 blockInfo 的数据结构如下：

```
typedef struct blockInfo{
    string hash;
    unsigned long long timeStamp;
    unordered_map <unsigned long long, tx> transactions;
}blockInfo;
```

   这里 unsigned long long 是交易的 txid，tx 是自定义的交易信息数据结构。

   (c) **交易信息 (tx):** 交易信息包括交易的编号 (txid)、该交易所属的 blockID、总交易金额 (money)、以及该交易发生时的时间戳 (Timestamp)。因此定义 tx 的数据类型如下：

```
typedef struct tx{
    unsigned long long tx_id;
    unsigned long long blockID;
    double money;//transaction amount
    unsigned long long timeStamp;
}tx;
```

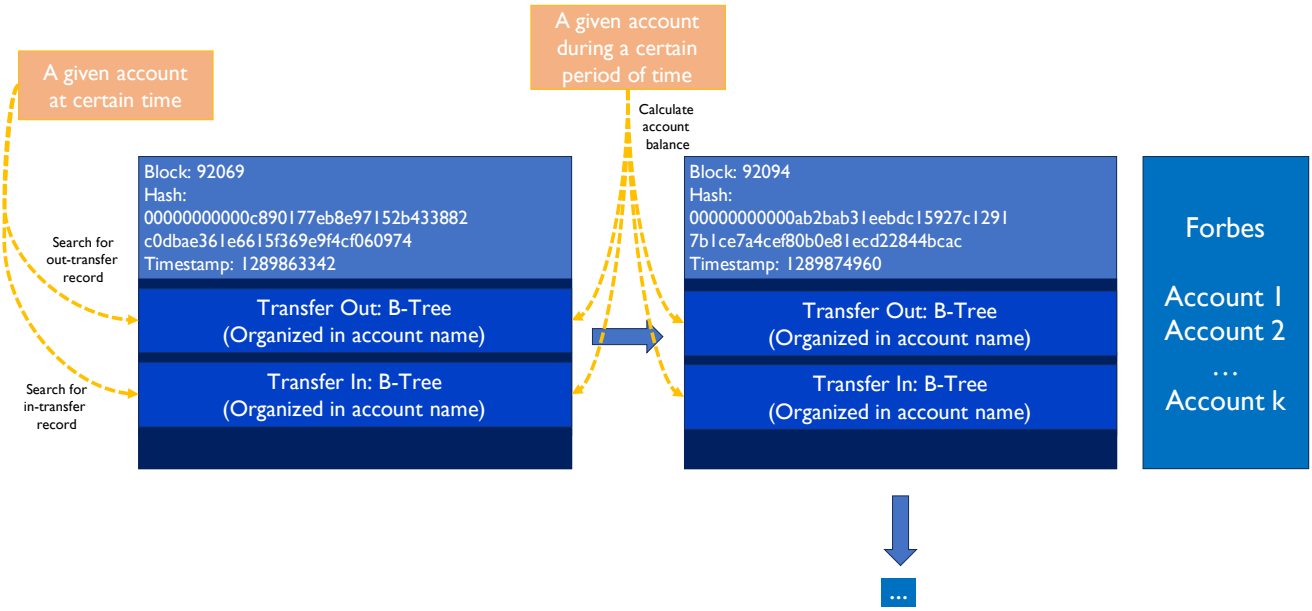

图 2: 区块与区块信息的组织形式

2. **交易关系图:** 一个交易关系图中最重要的结构是顶点 (账号) 与边 (交易)

   (a) **边 (edge):** 在交易关系图中，边是两个账号间的交易，包括交易的总金额和每笔详细交易。**需要强调的是，边本身不设方向性，只记录交易本身的详细信息。**

```
typedef struct edge{
    double totalAmount;//total transaction amounts
    list <tx> txs;//detailed transactions records
    //the sum-up of amount in detailed transactions records should be totalAmount
}edge;
```

   (b) **顶点 (vertex):** 在交易关系图中，顶点是参与交易的账号 (accounts)。除了记录账号名称外，**顶点的数据结构中还包括一个出边表、入边表。**以此来表现交易的方向性。这样设置的好处是在后续的区块链大数据分析中容易定位到某个账号的转入和转出记录，便于统计出度和入度，**出边表和入边表的数据结构也是无序映射，可以保证访问的时间复杂度为 $O(1)$，大大降低了查找的时间。**

```
typedef struct vertex{
    string accountName;
    unordered_map <string, edge> outEdges;//transfer out money
    //string: account name
    unordered_map <string, edge> inEdges;//receive money
    //string: account name
}vertex;
```

(c) **交易关系图 (transaction graph)**：交易关系图是由顶点和边组成的图。**此外，由于该交易关系图中有 40 万个账号，为了便于定位到某个账号的信息，因此也采用一个无序映射，将账号名称映射到顶点的数据结构中。**因此，交易关系图的数据结构定义如下：

```
unordered_map <string, vertex> graph;
```

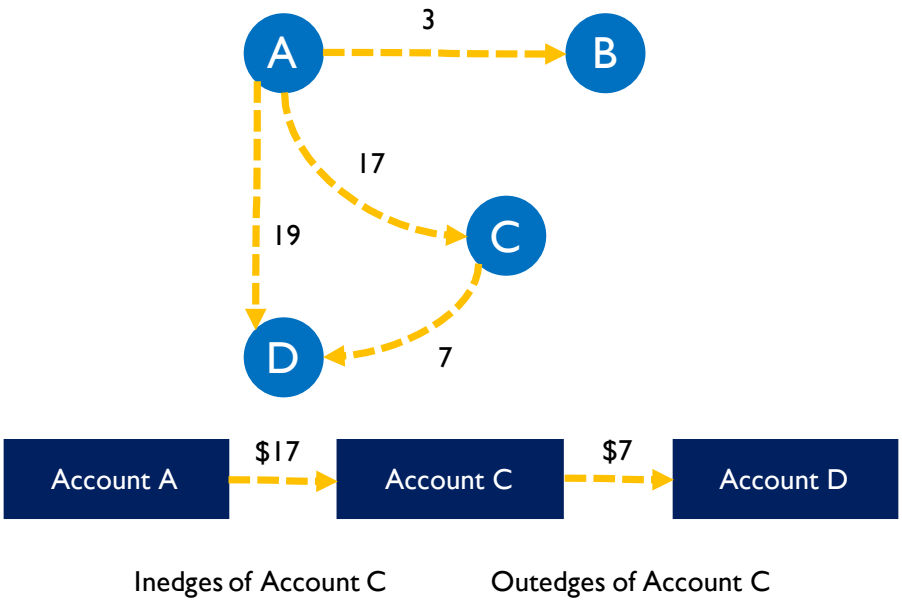

图 3: 交易关系图: 微观结构

2.2 区块链与交易关系图的构建

区块和交易信息的读取采用 C++ 的 ifstream 实现，此处不加赘述。主要介绍交易关系图的构建。交易关系图的构建即顶点集 V 和边集 E 的建立。

- **顶点集的建立：**扫描数据集时，如果该账号顶点已在图中存在，则无需操作，否则就在无序映射中插入该账号顶点。由于无序映射的插入操作的时间复杂度为 $O(1)$，因此顶点集的建立的时间复杂度为 $O(n)$，其中 n 为账号的个数。

```
void makeVertex(string Account){
    if (graph.find(Account) == graph.end()){//The vertex is not in the graph
        vertex V;
        V.accountName = Account;
        graph[Account] = V;
    }
}
```

- **边集的建立：**扫描数据集时，如果两账号之间的出边/入边已存在，则无需操作，否则对交易的发出账号顶点插入出边，对交易的接受账号顶点插入入边。由于无序映射的插入操作的时间复杂度为 $O(1)$，因此边

集的建立的时间复杂度为 $O(m)$，其中 m 为交易的个数。

```
void makeEdge(tx Mytx, string from, string to){
    //Modify/Adds OutEdges & InEdges of vertex "from" & "to"
    //Check if an edge from "from" to "to" already exists
    if (graph[from].outEdges[to].totalAmount != 0){
        //edge exists
        graph[from].outEdges[to].txs.push_back(Mytx);//adds the new transaction to the list
        graph[from].outEdges[to].totalAmount += Mytx.money;
    }
    else{
        //need to generate an edge
        edge E;
        E.totalAmount = Mytx.money;
        E.txs.push_back(Mytx);
        //Modify OutEdges of "from" and InEdges of "to"
        graph[from].outEdges[to] = E;//adds outEdge of "from"
        graph[to].inEdges[from] = E;//adds inEdge of "to"
    }
}
```

3 详细设计

3.1 数据查询功能

1. 查找指定账号转入转出记录: 归功于交易关系图的良好构建，查找指定账号转入转出记录只需访问该账号的出边表 outEdge 和入边表 inEdge。将交易记录储存在一个 vector 中，对 vector 内元素快排序，即可提取出交易金额最大的前 k 条记录。

2. 查询某个账号在某个时刻的金额: 将该账号入边表 inEdge 和出边表 outEdge 中的交易记录遍历一遍，即可得到该账号在某个时刻的收入；将该账号出边表 outEdge 中的交易记录遍历一遍，即可得到该账号在某个时刻的支出。两者相减记得到该账号在某个时刻的余额。

3.Forbes 榜: 将交易关系图中的顶点遍历一遍，将顶点的余额和账号名称存入一个 vector 中，对 vector 内元素快排序，即可得到福布斯富豪榜。

3.2 数据分析功能

1. 统计交易关系图的平均出/入度与最高出入度：遍历一遍交易关系图，使用 outEdge.size() 和 inEdge.size() 提取每个顶点的出度和入度，累加后除以顶点的个数即可得到平均出/入度。同时，将出度和入度存入一个 vector 中，对 vector 内元素快排序，即可得到最高出/入度。

2. 环检测：环检测的算法思想是拓扑排序。由于交易关系图数据结构的特别设计，容易更新每次账号顶点的入边和出边，方便拓扑排序的执行。

3. 最短路径：最短路径的求解采用 Dijkstra 算法，注意到图中顶点数量极多，采用朴素的 Dijkstra 算法时间复杂度将达到 $O(n^2)$，运行速度极慢。故此出采用 C++ stl 的优先队列模版 (priority queue) 优化 Dijkstra 算法，加快查找最近顶点和边松弛的速度，将时间复杂度降低到 $O(nlogn)$。

3.3 数据插入功能

数据插入功能：数据插入直接调去读取原始数据集时的函数`void readTransactions(string fileName)`即可。

4 用户使用说明

运行程序后，程序会先自动读取数据集。读取区块信息所用时间大约在 0.1 秒左右，读取交易信息所用时间大约在 16 秒左右。读取完毕后，程序会自动进入用户界面，用户界面如下图所示：

```
Block Reading Runtime: 0.016852s

Number of Transactions: 1048575
Transaction Reading Runtime: 15.776s

Input your command
-----------------------------------------
A. Inquiry
B. Data Analysis
C. Append Data
D. Quit System
-----------------------------------------
B
1. Analyze degree information of transaction graph.
2. Detect possible ring structure in graph.
3. Query certain account's shortest path to other accounts.
```

先以输入字母的方式选择四个功能，再输入功能下的详细选项，即可得到结果。这里以求账号到其他账号最短路径的功能 (B3) 为例：程序运行后，会相应输出运行所用时间，并再次返回主菜单

```
Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1P9UTssG8i2MsrcVvqtziuHn8kNSUcBhN4
Shortest path: 0.1617
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1P9UTssG8i2MsrcVvqtziuHn8kNSUcBhN4

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1EidJ9FrQeXed5jGYsZRrmFFBgCJBU7qb3
Shortest path: 1.0929
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1EidJ9FrQeXed5jGYsZRrmFFBgCJBU7qb3

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1CBcM2yf9e76yt28RGQ5VMkCoeX4QYFi1p
Shortest path: 0.0284
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1CBcM2yf9e76yt28RGQ5VMkCoeX4QYFi1p

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 12Pvxz3wiGFyombsxUp4QK3R3X7vH7Skxg
Shortest path: 0.0231
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 12Pvxz3wiGFyombsxUp4QK3R3X7vH7Skxg

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1PsDq5AN2uuKWokRjqvAL73JsQxrQEdcnc
Shortest path: 0.1916
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1PsDq5AN2uuKWokRjqvAL73JsQxrQEdcnc

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 14rWhXdqoS2BKGNbb9UW8tjeFAP1jpzDgn
Shortest path: 0.1968
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 14rWhXdqoS2BKGNbb9UW8tjeFAP1jpzDgn

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1EZqVm65VQbj43kh9Vz1PZUvyTqBZ2jGUD
Shortest path: 0.2125
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1EZqVm65VQbj43kh9Vz1PZUvyTqBZ2jGUD

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 12K5SyY2Z3DNsqFtTCnyGC3J7jYTCjM54m
Shortest path: 8.1342
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1FiDCGxMbE34VSEbq4UJHggA4YfnmXhV7Y -> 12K5SyY2Z3DNsqFtTCnyGC3J7jYTCjM54m

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1K7m6AET2RZ1nqYhbTPDhkMgkzKKHk88Dx
Shortest path: 0.982
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1K7m6AET2RZ1nqYhbTPDhkMgkzKKHk88Dx

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1GKk1PdgwN7M5cAGtn9Eht5NCGyQC7gm4c
Shortest path: 0.0702
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1GKk1PdgwN7M5cAGtn9Eht5NCGyQC7gm4c

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1C27A9TmLg2HRHM4ZAQwi59ouXY4QaJ18J
Shortest path: 0.7077
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1C27A9TmLg2HRHM4ZAQwi59ouXY4QaJ18J

Account 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A to Account 1FiDCGxMbE34VSEbq4UJHggA4YfnmXhV7Y
Shortest path: 0.1081
Pathway: 1EKGMGUmVK4Mpbnnh3UysJvT3Z5fywrv5A -> 1FiDCGxMbE34VSEbq4UJHggA4YfnmXhV7Y

Dijkstra Runtime: 3.6698s
-----------------------------------------
Input your command
-----------------------------------------
A. Inquiry
B. Data Analysis
C. Append Data
D. Quit System
-----------------------------------------
```

5 调试分析

5.1 时间复杂度分析

下面对区块链大数据分析的各个功能的时间复杂度进行分析：

1. **数据初始化：**数据初始化的时间复杂度为 $O(N)$，其中 N 为区块的个数和交易的总数。N 的量级大约为 10^6。
2. **查找指定账号转入转出记录：**定位到账号的时间复杂度为 $O(1)$，遍历出入边表的时间复杂度为 $O(n)$，其中 n 为账号的出入度。提取出交易金额最大的前 k 条记录用到快排序，时间复杂度为 $O(n\log n)$。因此，查找指定账号转入转出记录的时间复杂度为 $O(n\log n)$。
3. **查询某个账号在某个时刻的金额：**定位到账号的时间复杂度为 $O(1)$，遍历出入边表的时间复杂度为 $O(n)$，其中 n 为该账号的出入度。因此，查询某个账号在某个时刻的金额的时间复杂度为 $O(n)$。
4. **Forbes 榜：**遍历交易关系图的顶点的时间复杂度为 $O(|V|)$，其中 $|V|$ 为账号的个数。提取出交易金额最大的前 k 条记录用到快排序，时间复杂度为 $O(|V|\log|V|)$。因此，Forbes 榜的时间复杂度为 $O(|V|\log|V|)$。
5. **统计交易关系图的平均出/入度与最高出入度：**遍历交易关系图的顶点的时间复杂度为 $O(|V|)$，其中 $|V|$ 为账号的个数。提取出交易金额最大的前 k 条记录用到快排序，时间复杂度为 $O(|V|\log|V|)$。因此，统计交易关系图的平均出/入度与最高出入度的时间复杂度为 $O(|V|\log|V|)$。
6. **环检测：**环检测采用拓扑排序，时间复杂度为 $O(|V|+|E|)$，其中 $|V|$ 为账号的个数，$|E|$ 为交易的个数。
7. **最短路径：**最短路径采用 Dijkstra 算法，并用优先队列优化，时间复杂度为 $O(|V|\log|V|+|E|)$，其中 $|V|$ 为账号的个数，$|E|$ 为交易的个数。
8. **数据插入功能：**数据插入功能的时间复杂度与数据初始化的时间复杂度相同，为 $O(M)$，其中 M 为新区块的个数和交易的总数。

6 附录

可通过https://github.com/GONGSHUKAI/Data_Structure/tree/main/Lab_Code/Dec.1_Block_Chain_Lab6下载代码原文件。