区块链大数据分析 实验报告

(使用LaTeX编辑)

题目: 按照实验要求, 设计一个区块链大数据分析程序.

姓名: **崔冠宇**　　学号: **2018202147**　　完成日期: **2019.12.28**

1 需求分析

1. 设计任务: 用链表、栈、二叉树、图等数据结构, 设计一个区块链大数据分析程序, 要求具有数据初始化功能(含区块数据、交易数据(以树的形式组织)), 数据查询功能(含查询账号某时间段内最高转账记录、 某账号某时刻累计持有金额、某时刻福布斯排行榜(总金额最高榜)), 数据分析功能(含交易图输出、平均出入度分析、图上环路检测、 单源最短转账路径), 以及数据更新功能(支持按任意顺序执行命令).

2. 输入输出形式: 以交互式界面实现, 用户按程序所给出的提示输入操作命令, 程序执行相应命令后将结果输出在控制台中.

3. 达到的功能: 基本实现“设计任务”部分的要求.

4. 测试数据: 已经形成单独的测试报告, 详见测试报告.

2 概要设计

由于标准容器库内的数据结构比我写的数据结构效率高, 且大多数数据结构已经在之前的实验里设计过, 这里主要给出交易图的抽象数据类型定义以及其他一些辅助数据结构的定义.

1. 一些小的结构:

typedef std::string AccountName;　　//账户名

typedef long int TxTimeStamp;　　//时间戳

typedef struct Tx　　//交易记录

{

double amount;　　//金额

TxTimeStamp timeStamp;　　//时间戳

}Tx;

typedef int BlockID;　　//区块ID

typedef long int BlockTimeStamp;　　//时间戳

typedef struct BlockInfo　　//区块信息

{

...　　//其他操作

BlockTimeStamp timeStamp;　　//时间戳

std::unordered_map<int, Tx *> Txs;　　//交易树

int txCount;　　//区块下的交易数量

}BlockInfo;

typedef std::map<BlockID, BlockInfo *> BlockTree;　　//区块树

2. 边的定义:

typedef struct Edge;　　//一条边, 包括边上的交易总额以及交易指针链表

{

double sumAmount;

std::list<Tx *> txs;　　//交易指针链表

}Edge;

3. 某顶点出-入边树的定义:

   typedef struct Edges;　　//某顶点的出入边树, 通过AccountName映射

   {

   std::unordered_map<AccountName, Edge> InEdges;　　//转入的各个账号

   std::unordered_map<AccountName, Edge> OutEdges;　　//转出的各个账号

   }Edges;

4. 交易图抽象数据类型定义:

   //typedef std::unordered_map<AccountName, Edges> MarketGraph;

   ADT MarketGraph{

   //TODO

   数据对象: $V = \{v_i | v_i \in AccountNameSet, i = 1, 2, \ldots, n, n \in \mathbb{N}_+\}$

   数据关系: $R = \{< v, w > | v, w \in V\}$

   基本操作:

   (a) MarketGraph()

   初始条件: 无.

   操作结果: 构造函数, 构造一个空的交易图.

   (b) getVertices()

   初始条件: 交易图已经存在.

   操作结果: 返回顶点集合.

   (c) addVertex(AccountName & v)

   初始条件: 交易图已经存在.

   操作结果: 添加一个新的顶点v, 如果v已存在, 不操作.

   (d) addInEdge(AccountName & v, AccountName & another, Tx * info)

初始条件: 交易图已经存在.

操作结果: 在顶点v(不存在则插入)的入边树下增加一条指向another顶点(存在则不增加)的入边, 并在交易链表中新增一条指向info的指针.

(e) addOutEdge(AccountName & v, AccountName & another, Tx * info)

初始条件: 交易图已经存在.

操作结果: 在顶点v(不存在则插入)的出边树下增加一条指向another顶点(存在则不增加)的出边, 并在交易链表中新增一条指向info的指针.

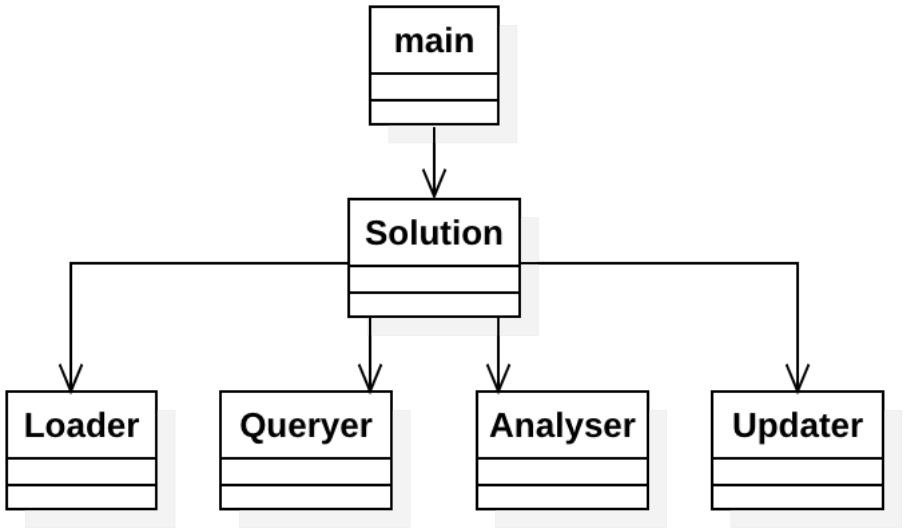
(f) deleteVertex(AccountName & v)

初始条件: 交易图已经存在.

操作结果: 删除顶点v及其出/入边树, 如果v不存在, 不操作.

}**ADT** MarketGraph<T>

5. 模块设计: 本程序共由6个主要模块组成, 它们分别是: 主程序、主解决方案模块、加载器模块、查询器模块、分析器模块和更新器模块. 模块之间的调用关系如下图:



3 详细设计

这一部分由于篇幅所限, 详细内容请参看代码, 下面仅列举一些个人认为比较有亮点的地方.

1. 在这次的区块链交易数据分析大作业中, 我尽可能实践运用了STL的一些容器, 比如list, map, unordered_map, priority_queue等. 在感叹于STL设计者的抽象能力之时, 我也因为STL的复杂性踩了不少坑. 比如遍历容器的过程中进行了删除节点操作, 导致迭代器非法化, 从而range-based loop失效而出现错误; 再比如忽略了pair<...>(...)以及make_pair(...)或者是push_back(...)以及emplace_back(...)的区别导致同一对象的多次复制构造, 增加了时间开销.

2. 我继续在此次作业中使用了lambda表达式, 比如Dijkstra算法中的findMin(), printPath(), 增强了功能的封闭性, 同时也使代码更加清晰美观.

3. 另外值得一提的一点是, 我希望紧跟C++标准, 所以经常会尝试一些比较新的标准中引入的特性, 比如auto推导, range-based loop等.

4 调试分析

1. 调试与问题解决: 偶尔会出现“手误”的情况, 或是逻辑不严密, 导致程序未能按预期运行. 这时, 我主要有三种解决方案: 首先先看一遍代码, 观察可能是哪里的问题(比如多重循环中内外层变量搞混了); 如果看不出来, 再利用lldb的单步调试以及变量查看功能, 观察代码执行情况, 从而定位问题; 除此之外增加输出语句, 如果是标准库的函数, 可以输出异常信息, 观察在哪里出错. 此外, 似乎还有一些代码自动分析工具, 以后有时间我也会去尝试.

2. 复杂度分析: 关于几类数据结构及相关算法的复杂度分析, 书上已经明确了, 在

这里我只简单分析一下几个算法的复杂度.

- 关于加载文件: 假设文件为 n 行, 由于是一行一行地读取, 故读取数据的复杂度为 $O(n)$. 读取一行后, 对字符串进行处理, 这里的复杂度我们就不分析了. 之后对分离出来的数据做区块下交易插入: 先从区块树(设有 n 个区块, std::map存储, 保持有序)找区块, $O(\log n)$, 找到后添加交易(设平均有 m 条交易, std::map), $O(\log m)$; 然后进行账号交易图插入: from 和 to 两个记录, 每次插入(unordered_map, 相当于hash table)均摊 $O(1)$.
- 关于查询功能: (设有 m 个不同的账号, 平均每个账号有 n 条交易.) 前两个功能可以直接在交易图中查找到所需账号, 然后遍历一下出/入边表, 筛选所需交易进行处理即可, 是 $O(n)$ 复杂度. 第三个功能等价于对每一个账户的交易记录进行遍历求和($O(m \times n)$), 然后排序(使用优先队列, $O(m \times \log m)$)输出, 总复杂度 $O(m \times n + m \times \log m)$, 一般而言 $\log m << m << n$, 可以认为时间复杂度为 $O(m \times n)$.
- 关于分析功能: 交易图输出功能只需遍历交易图, $O(|V| \times |E|)$. 出入度统计功能先遍历顶点集合, 然后直接获取出/入边树的size(), 总复杂度为 $O(|V|)$. 环路检测使用的是书上的拓扑排序功能, 复杂度为 $O(|V| + |E|)$. 关于Dijkstra求最短路径, 这里就不分析了.

3. 一些感想: 有了上次计算器的实验的经验, 我在这次大作业中还是使用了“模块化”的思路, 将程序分为几个相对独立的模块, 然后分模块解决对应的问题, 效率还是比较高的. 感觉软件的设计还有很多东西要我去学习啊.

5 用户手册

这一部分根据实验要求, 已经拆分成独立的用户手册, 请参阅“用户手册.pdf”.

6 测试结果

这一部分根据实验要求, 已经拆分成独立的测试报告, 请参阅“测试报告.pdf”.

7 附录

源程序文件名清单(按字典序)如下:

1. account.h //账号的结构
2. analyser.h //分析器模块
3. block.h //区块的结构
4. graph.h //交易图数据结构
5. loader.h //加载器模块
6. main.cpp //主程序
7. queryer.h //查询器模块
8. solution.h //主模块
9. tx.h //交易记录的结构
10. updater.h //更新器模块
11. util.h //一些宏定义