Contents

[Task-1: (ExchangeCenter) 1](#_Toc39563522)

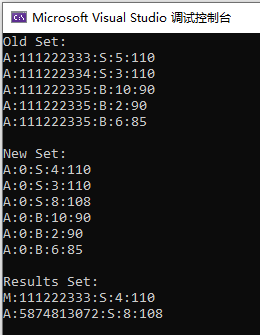
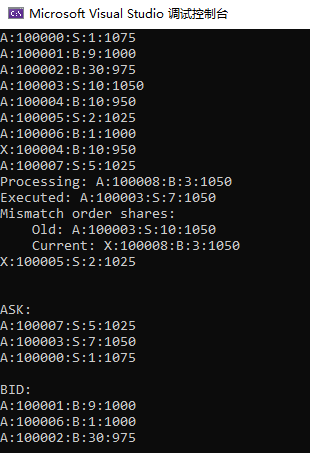
[Pseudo Logic: 2](#_Toc39563523)

[Task 2: (Trader) 2](#_Toc39563524)

[Pseudo Login: 3](#_Toc39563525)

NewStart.exe -t // Read traderOld.txt & traderNew.txt in current DIR.

NewStart.exe // Read exchange.txt in current DIR

Sample results using provided data in the coding test.

## Task-1: (ExchangeCenter)

Assume only one stock, use TLX::BTree to do the merge Algorithm, Please remind that if you want to use TLX lib in your product, states

@Misc{TLX,

title = {{TLX}: Collection of Sophisticated {C++} Data Structures, Algorithms, and Miscellaneous Helpers},

author = {Timo Bingmann},

year = 2018,

note = {\url{https://panthema.net/tlx}, retrieved {Oct.} 7, 2020},

}

As Java Version, I didn’t write B+ Tree by myself, and I didn’t implement aggressive ASK/BID logic, But TLX::BTree has range search.

//! Searches the B+ tree and returns both lower\_bound() and upper\_bound().

std::pair<iterator, iterator> equal\_range(const key\_type& key) {

return std::pair<iterator, iterator>(

lower\_bound(key), upper\_bound(key));

}

Maybe I used master branch or use it wrongly, there’re memory access violations, so I do some minor changes which fix following warning

C:\Ivault\src\cpp\NewStart\btree.hpp(294): warning C4172: 返回局部变量或临时变量的地址

@@ -290,7 +290,7 @@ private:

//! Return key in slot s.

- const key\_type& key(size\_t s) const {

+ const key\_type key(size\_t s) const {

return key\_of\_value::get(slotdata[s]);

}

@@ -416,7 +416,7 @@ public:

//! Key of the current slot.

- const key\_type& key() const {

+ const key\_type key() const {

return curr\_leaf->key(curr\_slot);

}

### Pseudo Logic:

1. load exchange.txt, save line as ExchangeItem
2. ExchangeItem uses list<ShareItem> & <map> to track execution, <map> uses price as key, ShareItem as value
3. ShareItem stores order id, shares, valid (if an item cancelled, mark it as Invalid)
4. for each line inserting into BTree, do the following  
   get leafnode using price (TODO: aggressive ask & bid, not match, do range search   
    to get lower or higher bound)  
   if not match & is OrderEnum::A, insert; else skip(invalid)  
   if OrderEnum::X, search <map>, if not match, skip, else mark ShareItem invalid  
   if same price & exchange type, append  
   if different, merge()
5. merge() Algorithm (only one exchange type in the queue)  
   pop items from list<ShareItem>  
   if invalid, skip & release memory  
   execute orders (refer to ExchangeItem::merge())  
   if still remaining quantity, push back to the queue.
6. Print order book

## Task 2: (Trader)

There’re three cases that represent cancelled order

order missed in New Input Set (I only concern this case)

order has X mark in New Input Set

order has 0 shares in New Input Set

Assume only one stock, use list<ExchangeItem \*> & map<string, queue<ExchangeItem\*>\*> to do the logic.

### Pseudo Login:

for each price, linked by <list>, under the same price, append order into queue(FIFO)

intersect() {

cursor of oldSets, cursor of newSets

if not match in oldSets, New Order; if not match in newSets, Cancel Order

compare two queues, if not equal, prefer Modify Order

if oldQueue is empty, New Order, newQueue is empty, Cancel Order.

}