拼全盘过程中遇到的问题

# 一、模拟撮合

## 1.1 数据结构

### 1.1.1 盘口结构

需注意：有哪些信息是需要作为快速访问信息的？

①盘口总单量：用于集合竞价时快速撮合

②盘口价格（非必需）：很少用到，只是作为指示存在

struct Level

{

double Price; //盘口价格

double TotalVolume; //盘口总单量

std::deque<ORDER\_STRUCT> OrdIDX; //该盘口的订单队列

};

为什么用deque？

**订单挂在盘口队列末尾的情况：**

1. 集合竞价/停牌时

2. 连续竞价时，某订单没被平完

3. 集合竞价将某订单取出队列撮合，没平完需要归队

**订单挂在盘口队列首位的情况：**

1. 价格变动导致新的有效order被取出，没平完挂起时要挂在队首

\*挂在队尾的情况远比挂盘口常见：可改写成queue

### 1.1.2. 买卖盘结构

std::map<double, Level> bids\_book; //买盘数据

### 1.1.3 市场信息结构

把撮合交易过程中产生的信息打包到一个mkt结构体里面，统一返回或输出

struct MktInfo

{

double LstClose; //昨收 double TodayOpen; //今开 double TodayClose; //今收

double NewestPrice; //最新价double BestOffer; //卖一 double BestBid; //买一

double BidPrices[LEVELDISPLAY]; //买盘价格

double BidVolumes[LEVELDISPLAY]; //买盘量

double OfferPrices[LEVELDISPLAY]; //卖盘价格

double OfferVolumes[LEVELDISPLAY]; //卖盘量

}; //可以自定义MktInfo包含的内容

## 1.2 集合竞价

### 1.2.1 边确定价格边撮合 or 先确定价格后撮合？

|  |  |  |
| --- | --- | --- |
|  | 边确定价格边撮合 | 先确定价格后撮合 |
| 好处 | 1. 不用重复遍历盘口 | 1. 价格确定较快  2. 如果价格出错，回溯操作较为方便 |
| 坏处 | 1. 价格确定较慢  2. 需要在所有订单成交之后再统一赋成交价 | 1. 需要重复遍历盘口  2. 取出订单存入vector会占用空间 |

最终选择了先确定价格后撮合.

### 1.2.2 有多个有效价格，确定最终价格

何时会出现多个有效价格？

1. 买盘和卖盘盘口从价格远的地方（最高有效买价 & 最低有效卖价）往接近的地方（差价越来越小）一一撮合。

\* 无价格限制的证券（可转债、新票上市/复牌首日）的买一价和卖一价可能不是当前最优报价，而是由昨收价（开盘集合竞价）或最新价（非开盘集合竞价）确定的有效竞价范围内的最优报价。

常见情况：价格很低的卖单/价格很高的买单，都不能作为买一或卖一价使用.

2.当某一方的量大于另一方时，量小一方的盘口会被平完，最终价格维持在量大一方的价格。

只有当两方正在撮合的盘口量相同时，才可能会有多个有效价格。

Ex: 价格 量 价格 量

当前买一 12.88 60 当前卖一 12.84 60

当前卖二 12.82 30 当前买二 12.80 100

最终有效价格是什么？有效价格范围内的所有价格（量为0的价格也算）

A. 12.84 和 12.88（×）

B. 12.84 12.85 12.86 12.87和12.88（√）

如何确定最终价格？

①两个以上价格符合上述条件的，取在该价格以上的买入申报累计数量与在该价格以下的卖出申报累计数量之差最小的价格为成交价；

伪代码：

bid\_iter = bids\_order\_book.find(Mkt.BestBid);

while (bid\_price >= current\_price - MINGAP \* 0.001 && IsValidPrice(bid\_price))

{

bid\_volume += current\_level\_volume;

getNextPrice();

}

while (offer\_price <= current\_price + MINGAP \* 0.001 && IsValidPrice(offer\_price))

{

offer\_volume += current\_level\_volume;

getNextPrice();

}

实例：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 价格 | 12.84 | 12.85 | 12.86 | 12.87 | 12.88 |
| 该价格以上（含）的买入累计申报数量 | 60 | 60 | 60 | 60 | 0 |
| 该价格以下（含）的卖出累计申报数量 | 60 | 60 | 60 | 60 | 60 |
| 量差 | 0 | 0 | 0 | 0 | 60 |

②买卖申报累计数量之差仍存在相等情况的，开盘集合竞价时取最接近即时行情显示的前收盘价的价格为成交价，盘中、收盘集合竞价时取最接近最近成交价的价格为成交价。

**假设昨收为12.877 最终成交价为12.87**

## 1.3 连续竞价

### 1.3.1 连续竞价的有效竞价范围

可转债有价格笼子，有效竞价范围为最新成交价的±10%，深交所无效单暂存于缓存队列，新的有效价格单子会被取出，进行撮合。

#### 问题一：怎么取新的有效单？

① 当前订单没被平完，就有新盘口有效，怎么办？

等当前订单平完之后再取新单子，最终有效申报看最近的成交价（单子平完的成交价）。

② 单取一个单子，还是整个盘口取出？

按照最终有效申报，将符合条件的所有盘口按价格从优（买单越高越优，卖单越低越优）取出，相同价格的订单按照时间优先取出，进行连续竞价，符合连续竞价的交易规则。

③ 这批订单取出后，没平完之前，新成交价就把价格笼子越打越高（低），怎么处理单子的撮合顺序？

一个订单全部平完之后，假设最新成交价又上涨，带出了价格更高的合法买单，这批价格更高的买单要排在之前取出的单子之后，不能插队。这时候的队列里面，会出现高价买单排在低价买单之后的情况。

④ 这批订单全部平完之前，有新的单子到了怎么办？

这些订单以插队的性质出现在队列里，新单排在它们之后。（直到这些订单被平完或挂起之前，都不会撮合新收到的订单）

Ex：初始状态：最新成交价12.80，此时最高有效价为14.08

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 买盘盘口 | 12.791 | 14.100 | 14.101 | 14.102 | 14.166 |
| 量 | 100 | 100 | 100 | 500 | 1000 |
| 卖盘盘口 | 12.82 | 12.83 | 12.88 |  |  |
| 量 | 100 | 500 | 1000 |  |  |
| 单量 | 3 |  |  |  |  |

情况一：

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 一个价格为12.82，量为1000的买单进入，第一笔成交后，最新价发生变化，该买单和12.82的盘口都有剩余 | | | |
| 最新价：12.82 | 最新有效买价：14.102 | 卖一价：12.82 | 新的有效申报：14.102、14.101、14.100 |
| 2. 根据①，当前订单没平完之前不取新单，继续平12.82的买单，直到12.83盘口被平完 | | | |
| 最新价：12.83 | 最新有效买价：14.113 | 卖一价：12.88 | 新的有效申报：无 |
| 3. 根据①，继续平12.82的买单，直到该单被平完 | | | |
| 最新价：12.88 | 最新有效买价：14.166 | 卖一价：12.88 | 新的有效申报：14.166 |
| 4. 由于2中新单未被取出，即未被加入撮合队列，此时最高有效申报变为14.166.根据②  最终队列中排队顺序：14.166>14.102>14.101>14.100>14.099 | | | |

Ex：初始状态：最新成交价12.80，此时最高有效价为14.08

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 买盘盘口 | 12.791 | 14.100 | 14.101 | 14.102 | 14.166 |
| 量 | 100 | 100 | 100 | 500 | 1000 |
| 卖盘盘口 | 12.82 | 12.83 | 12.88 |  |  |
| 量 | 100 | 500 | 1000 |  |  |
| 单量 | 3 |  |  |  |  |

情况二：

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 一个价格为12.82，量为100的买单进入，这个订单被平完之后的状态如下： | | | |
| 最新价：12.82 | 最新有效买价：14.102 | 卖一价：12.83 | 新的有效申报：14.102、14.101、14.100、14.099 |
| 2. 根据②，取出这些有效的单子，按照价格优先、时间优先撮合。当14.102的盘口和12.83的盘口平完之后，市场状态如下： | | | |
| 最新价：12.83 | 最新有效买价：14.113 | 卖一价：12.88 | 新的有效申报：无 |
| 3. 此时买单队列首位为14.101，与12.88平第一单之后： | | | |
| 最新价：12.88 | 最新有效买价：14.166 | 卖一价：12.88 | 新的有效申报：14.166 |
| 4. 根据③，新的14.166要在14.101与14.100都平完之后才能平，  此时队列顺序：14.101>14.100>14.166（高价买单排在低价买单后面） | | | |
| 5. 假设14.101和14.100都平完，此时卖一价：12.88，卖一量：800. 按照连续竞价规则，14.166的单子平到剩下200手之后，卖盘为空，该订单被挂起 | | | |
| 最新价：12.88 | 买一价：14.166 | 卖一价：DBL\_MAX | 新的有效申报：无 |
| 6. 收取下一个订单 | | | |

#### 问题二：什么时候取出这些订单？

**每次产生新成交价（包括开盘价）时，都需要检查是否有新的有效订单**

要做到以上要求，有两种方式：

① 每次撮合一个新交易，退出函数之前，作为一次连续竞价操作的末尾（即实现连续竞价的函数[ ContBid() ]的末尾）

按流程图中结构: A+B, C+B, C+B, ……,（以C+B封装为一个函数）

②还没开始撮合新交易之前，作为一次连续竞价操作的开头（即实现连续竞价的函数[ ContBid() ]的开头）

按流程图中结构: A, B+C, B+C, ……,（以B+C封装为一个函数）

实践证明，应该按照②方式封装函数

### 1.3.2 临时停牌

#### 问题一：如何区分哪些单子是临停时收到的（未解决）

深交所临停期间收到的订单会统一打上复牌时间的时间戳，不能用OrderTime区分最后一个临停期间收到的单子（以下称为A单）和第一个连续竞价收到的单子（以下称为B单）。

解决方案：A、B单之间的OrderIndex不连续，有一定的GAP可用来区分两个订单

原代码：（不考虑Order Index不连续性）

if (!Time1Bigger((current\_ord+1)->OrderTime, Mkt.StopTradeUntil))

HangOrd(\*current\_ord, current\_ord->FunctionCode, true);

改成新代码：（考虑Order Index不连续性）

if (!Time1Bigger((current\_ord+1)->OrderTime, Mkt.StopTradeUntil) && (current\_ord + 1)->OrderIndex - current\_ord->OrderIndex <= 150)

HangOrd(\*current\_ord, current\_ord->FunctionCode, true);

**新问题：不同合约的AB间GAP不一样大**

Ex: 时间戳相同的一批订单

票1真实临停结束点

1. 假设GAP大小定为25（跳跃25即认为临停结束）：

票2判断正确，但票1会在10672就进入连续竞价，出现错误

2. 假设GAP大小40：票1判断正确，但票2一直不会进入连续竞价，出错

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 票1 | 10668 | 10669 | 10672 | 10699 | 10711 | 10722 | 10723 | 10755 | 10800 |
| 票2 | 10121 | 10122 | 10123 | 10124 | 10125 | 10126 | 10149 | 10151 | 10152 |

票2真实临停结束点

#### 问题二：临时停牌的复牌集合竞价

临时停牌结束时，交易所会对临停期间收到的所有订单进行一次复牌集合竞价。

核心问题：复牌竞价之后，产生新的成交价，还需要调整有效竞价范围来连续竞价吗？

假设按照1.3.1问题二中的①方法封装函数：A+B, C+B, C+B, ……，集合竞价结束后会自动进行一次取新订单并连续竞价的操作。

假设按照1.3.1问题二中的②方法封装函数：A, B+C, B+C, ……, 只有后接连续竞价的集合竞价才会取新的有效订单并连续竞价。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 开盘集合竞价 | 盘中临停10min后复牌集合竞价 | 临停至14：57后复牌集合竞价 | 收盘集合竞价 |
| ①方法 | √ | √ | √ | √ |
| ②方法 | √ | √ | × | × |

√：取新订单并连续竞价 ×：不取新订单并连续竞价

从逻辑上说，收盘集合竞价之后不应该再有交易，所以②方法才是更合理的封装方法。真实实践中也能发现，临停至14：57后复牌集合竞价所产生的价格可能会带出新的有效订单，这些订单会作为合法买卖单进入收盘集合竞价，而非单个撮合。

# 二、模拟盘口（不撮合）

## 2.1 市价限价交易

### 2.1.1创业板（30）有价格申报限制：

（一）买入申报价格<=102%买一；

（二）卖出申报价格>=98%卖一。

无即时揭示的最低卖出（最高买入）申报价格的，为即时揭示的最高买入（最低卖出）申报价格；

无即时揭示的最高买入（最低卖出）申报价格的，为最近成交价；

当日无成交的，为前收盘价。

开市期间临时停牌阶段的限价申报，不适用前两款规定。

有效竞价范围外的订单不会被打回，而是暂存于交易所主机中

#### 问题一：怎么给市价单赋价格？

需要实时跟踪最新的买一、卖一价格以给限价、市价单赋值

解决方案：在每次买一和卖一盘口发生变化时更新两方盘口

Ex: 如果卖一盘口发生变动：

①删除卖一盘口

②根据买一盘口确定的有效范围确定下一个卖一盘口

③根据新的卖一盘口更新买一盘口的有效范围

④如果有变动，更新买一盘口

#### 问题二：某方盘口为空怎么办？

假设买盘为空，市价单类型为本方最优报价，那么其价格只能赋成卖一的价格

\*某方盘口为空往往出现在涨跌停板的时候，此时无论买卖单，均打涨跌停价格即可。

## 2.2 盘后交易

### 2.2.1 盘后mkt信息推送与收盘最后一条mkt

688开头的股票，Mkt文件中会带有盘后交易的推送

导致的问题：不能单纯读mkt文件最后一条作为市场信息

解决方案：从后往前遍历寻找三点集合竞价结束时的市场状态的信息

**标志字段:InstrumentStatus**

|  |  |
| --- | --- |
| InstrumentStatus (时段) | InstrumentStatus Describe (时段描述) |
| •INIT | 启动（开市前）时段 |
| •PCALL | 集中撮合时段 |
| •POSMT | 连续交易时段 |
| •ENDPT | 闭市时段 |
| •POSSP | 停牌 |
| •CLOSE | 收盘 |
| •POSTR | 盘后交易 |

15:00~15:05，发 PCALL 标志；15:05~15:30，发 POSMT 标志； 15:30 后， 发 ENDPT 标志。

## 2.3 不同交易所对无效盘口的处理

上海 – 价DBL\_MAX 量0

深圳 – 价DBL\_MAX 量MAX

# \* C++编程经验

## 1 指针

### 1.1 容器迭代器（iterator）指向end()的操作

程序会报错并停止运行，这时候debug状态下可以选择中止或者重试（retry），点重试之后再按F10可以找到出错的语句。（不然的话debug难度会很大）

### 1.2 buffer指针指向buffer外

Ex:

txn\_buffer = (TRADE\_STRUCT\*)malloc(sizeof(TRADE\_STRUCT) \* (Mkt.TxnCount));

auto txn\_ptr = txn\_buffer + Mkt.TxnCount + 1;

此时txn\_ptr会越界，在release模式下继续运行会导致程序崩溃

### 1.3 结构体指针

struct MARKET\_STRUCT //市场数据结构体

{

double BidPrice1; // 申买价一

double BidVolume1; // 申买量一

double BidCount1; // 申买笔数一

double OfferPrice1; // 申卖价一

double OfferVolume1; // 申卖量一

double OfferCount1; // 申卖笔数一

double BidPrice2; // 申买价二

double BidVolume2; // 申买量二

double BidCount2; // 申买笔数二

double OfferPrice2; // 申卖价二

double OfferVolume2; // 申卖量二

double OfferCount2; // 申卖笔数二

double BidPrice3; // 申买价三

double BidVolume3; // 申买量三

double BidCount3; // 申买笔数三

double OfferPrice3; // 申卖价三

double OfferVolume3; // 申卖量三

double OfferCount3; // 申卖笔数三

……

}

struct MktInfo

{

double BidPrices[LEVELDISPLAY]; // 买盘价格

double BidVolumes[LEVELDISPLAY]; // 买盘量

double OfferPrices[LEVELDISPLAY]; // 卖盘价格

double OfferVolumes[LEVELDISPLAY]; // 卖盘量

……

}

以上展示的是两个结构体内部变量的组织方式，在需要访问的结构体有连续的相同类型变量排列的时候，可以用该类型的指针连续访问这些变量

实例如下：

double\* real\_ptr = &lst\_mkt.BidPrice1; // 指向MARKET\_STRUCT类型的指针

double\* sim\_ptr = &Mkt.BidPrices[0]; // 指向MktInfo类型的指针

for (int j = 0; j < LEVELDISPLAY; j++, real\_ptr += 6, sim\_ptr += 1)

{

if (\*real\_ptr != \*sim\_ptr)

return false;

}

\* real\_ptr+=6 的原因：BidPrices1和BidPrices2之间隔了BidVolume1、BidCount1、OfferPrice1、OfferVolume1、OfferCount1五个double，如果BidPrices1的Index为0，那么BidPrices2的Index为6.

## 2 double精度

### 2.1 确定价格限制及比较

设置价格限制的时候，要留有一定容错

Mkt.Upper = round(Price \* (1 + TRADEPB) \* 1000.0) / 1000.0 + MINGAP \* 0.1;

Mkt.Lower = round(Price \* (1 - TRADEPB) \* 1000.0) / 1000.0 - MINGAP \* 0.1;

确保有容错范围之后，比较时最好就不要加等号了（大于等于应变为严格大于，小于同理）

### 2.2 盘口价格

新订单收进来不需要把价格按照四舍五入，保留最小精度。直接按照订单价格创建盘口、交易就可以。

## 3 字符串时间

时间以字符串形式出现时，要按照其格式规定比较方法

bool MarketManager::Time1Bigger(char\* time1, char\* time2)

{

for (int i = 0; i < 12; i++)

{

if (time1[i] > time2[i])

return true;

else if (time1[i] < time2[i])

return false;

}

return false; //默认等于为false,即严格大于才为真

}