MS\_project（算法交易） 设计文稿

学号：516030910033

姓名：方娄昊

1. MS\_sort：

1.1主要功能模块及其代码:

1.1.1for\_MS.h：

（1） DatainList 用于存储level1\_book中每个格子中的数据，记录size和price。

struct DatainList //用于存储level1\_book中每个格子中的数据

{

int size = 0;

double price = 0;

};

（2）List 用于存储level1——book，其中用symbol和side对book进行分类，此外还包含3个DatainList，每个DatainList对应一个交易所。output函数返回一个符合输出形式的字符串

struct List //用于存储level1——book

{

string symbol;

string side;

DatainList NYSE;

DatainList NASDAQ;

DatainList IEX;

List(string sy,string si):symbol(sy),side(si){};

string output();

};

（3）Order 用于存储一个订单。在private中 创建结构OneExchange，用于存储一次交易的信息。Sum 表示总共的交易额。index为用于订单分类的编号。havefilled表示 这个订单是否被填满。exchanges表示被该订单吃掉的quote。present\_quantity表示当前的被吃掉的size。 在pubilc中，eatExchange()表示吃掉一次交易。output()表示将结果输出到一文件中。 output2()表示将结果输出在命令行中。

class Order //用于存储一个订单

{

private:

struct OneExchange //用于存储一次交易的信息

{

string Exchange;

int size;

double price;

OneExchange(string E, int s, double p):Exchange(E),size(s),price(p){}

};

int Orderid;

string Side;

string Symbol;

double Price = 0;

int Quantity = 0;

double sum = 0; // 表示总共的交易额

int index = 0; //为用于订单分类的编号

bool haveFilled = false;//这个订单是否被填满

vector<OneExchange> exchanges;//表示被该订单吃掉的quote

int present\_quantity = 0;//表示当前的被吃掉的size

double average\_price;

public:

Order(int o, string si, string sy, double p, int q);

void print();

int getOrderid();

string getSide();

string getSymbol();

double getPrice();

int getQuantity();

int getIndex();

bool ifhaveFilled();

void eatExchange(string Exchange ,int &size, double price, List &list);//表示吃掉一次交易

double getSum(){return sum;}

double getAverage();

void fill();

void output(ofstream &ost);//表示将结果输出到一文件中

void output2(vector<string> ratios);//表示将结果输出在命令行中。

};

（4）Orders 用于存储多个Order，在public中getVector()表示将Orders以vector的形式输出。

class Orders

{

private:

vector<Order> VectorOfOrder;

string filename;

public:

Orders(string filename);

Order getOrder(int orderid);

vector<Order> getVector();//表示将Orders以vector的形式输出

void printAll();

};

1.1.2for\_MS.cpp：

for\_MS.h的实现。

string List::output()

{

stringstream ss;

ss<<NYSE.size<<"@"<<NYSE.price<<","<<NASDAQ.size<<"@"<<NASDAQ.price<<","<<IEX.size<<"@"<<IEX.price;

string temp;

ss>>temp;

temp = temp + "\n";

return temp;

}

Order::Order(int o, string si, string sy, double p, int q)

:Orderid(o),Side(si),Symbol(sy),Price(p),Quantity(q)

{

if(Side == "Buy"&&Symbol == "GOOG")

{

index = 0;

}

else if(Side == "Buy"&&Symbol == "AAPL")

{

index = 1;

}

else if(Side == "Sell"&&Symbol == "GOOG")

{

index = 2;

}

else if(Side== "Sell"&&Symbol == "AAPL")

{

index = 3;

}

}

void Order::print()

{

cout<<Orderid<<" "<<Side<<" "<<Symbol<<" "<<Price<<" "<<Quantity<<endl;

}

int Order::getOrderid()

{

return Orderid;

}

double Order::getPrice()

{

return Price;

}

int Order::getQuantity()

{

return Quantity;

}

string Order::getSide()

{

return Side;

}

string Order::getSymbol()

{

return Symbol;

}

bool Order::ifhaveFilled()

{

return haveFilled;

}

void Order::fill()

{

haveFilled = true;

}

int Order::getIndex()

{

return index;

}

void Order::eatExchange(string Exchange,int &size, double price, List &list)

{

if(size + present\_quantity < Quantity)

{

present\_quantity += size;

OneExchange oneExchange(Exchange,size,price);

exchanges.push\_back(oneExchange);

sum += size \* price;

size = 0;

}

else

{

haveFilled = true;

int size2 = Quantity - present\_quantity;

size = size - size2;

present\_quantity = Quantity;

OneExchange oneExchange(Exchange,size2,price);

sum += size2 \* price;

exchanges.push\_back(oneExchange);

}

}

double Order::getAverage()

{

return sum/present\_quantity;

}

void Order::output(ofstream& ost)

{

for (int i = 0; i < exchanges.size(); ++i)

{

ost<<exchanges[i].Exchange<<":"<<exchanges[i].size<<"@"<<exchanges[i].price<<endl;

}

ost<<"Quantity:"<<present\_quantity<<endl;

ost<<"average price:"<<getAverage()<<endl;

}

void Order::output2(vector<string> ratios)

{

for (int i = 0; i < exchanges.size(); ++i)

{

cout<<exchanges[i].Exchange<<":"<<exchanges[i].size<<"@"<<exchanges[i].price<<endl;

}

cout<<"Quantity:"<<present\_quantity<<endl;

cout<<"average price:"<<getAverage()<<endl;

cout<<ratios[index];

}

Orders::Orders(string filename)

{

ifstream ist(filename.c\_str());

if(!ist) cout<<"can't open this file";

int Orderid = 0;

string Side;

string Symbol;

double Price = 0;

int Quantity = 0;

string temp;

getline(ist,temp);

stringstream ss;

unsigned long n = temp.find("\r");

temp.erase(0,n+1);

while(temp.find("\r") <= temp.size()-1)

{

temp = temp.replace(temp.find("\r"),1,"\n");

}

while(temp.find(",") <= temp.size()-1)

{

temp = temp.replace(temp.find(","),1," ");

}

ss<<temp;

while(ss>>Orderid>>Side>>Symbol>>Price>>Quantity)

{

Order anOrder(Orderid,Side,Symbol,Price,Quantity);

VectorOfOrder.push\_back(anOrder);

}

}

Order Orders::getOrder(int orderid)

{

if(orderid<=5 && orderid >= 0)

{

return VectorOfOrder[orderid - 1];

}

else

{

cout<<"unexpected orderid. This is the first order"<<endl;

return VectorOfOrder[0];

}

}

void Orders::printAll()

{

for (int i = 0; i < VectorOfOrder.size(); ++i)

{

VectorOfOrder[i].print();

}

}

vector<Order> Orders::getVector()

{

return VectorOfOrder;

}

1.1.3main.cpp:

主线程：输出数据处理的结果。

线程2:从文件中读取数据并处理。

在每一行的处理中，先判断该quote的类别

int index = 0;

if(Side == "Ask"&&Symbol == "GOOG")

{

index=0;

}

else if(Side == "Ask"&&Symbol == "AAPL")

{

index=1;

}

else if(Side == "Bid"&&Symbol == "GOOG")

{

index=2;

}

else if(Side == "Bid"&&Symbol == "AAPL")

{

index=3;

}

并在vector中与相应的Order对应

for (int i = 0; i < VectorOfOrder.size() ; ++i)

{

if(Size == 0) break;

if(VectorOfOrder[i].ifhaveFilled()) continue;

if(index == VectorOfOrder[i].getIndex()) {

if (index < 2 && VectorOfOrder[i].getPrice() >= Price)

{

VectorOfOrder[i].eatExchange(Exchange, Size, Price, vectorOfList[index]);

}

if (index >= 2 && VectorOfOrder[i].getPrice() <= Price)

{

VectorOfOrder[i].eatExchange(Exchange, Size, Price, vectorOfList[index]);

}

}

}

遍历完所有Order后，如果quote的size还有剩余，则将该quote的数据放入相应的level1——book中。

if(Size != 0)

{

if(Exchange == "NYSE")

{

vectorOfList[index].NYSE.size = Size;

vectorOfList[index].NYSE.price = Price;

string oneOutput = vectorOfList[index].output();

(\*temps[index]).push(oneOutput);

}

if(Exchange == "NASDAQ")

{

vectorOfList[index].NASDAQ.size = Size;

vectorOfList[index].NASDAQ.price = Price;

string oneOutput = vectorOfList[index].output();

(\*temps[index]).push(oneOutput);

}

if(Exchange == "IEX")

{

vectorOfList[index].IEX.size = Size;

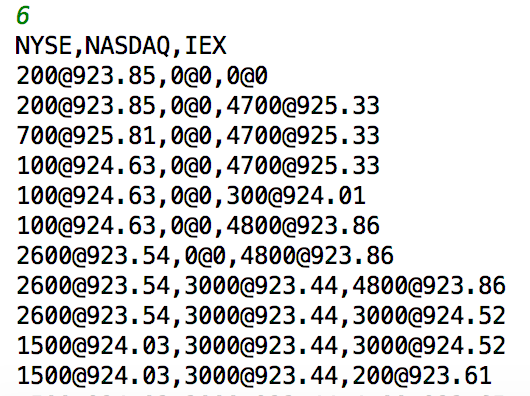
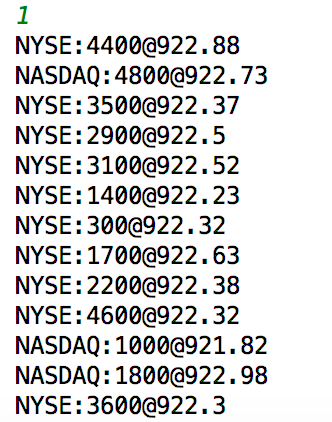
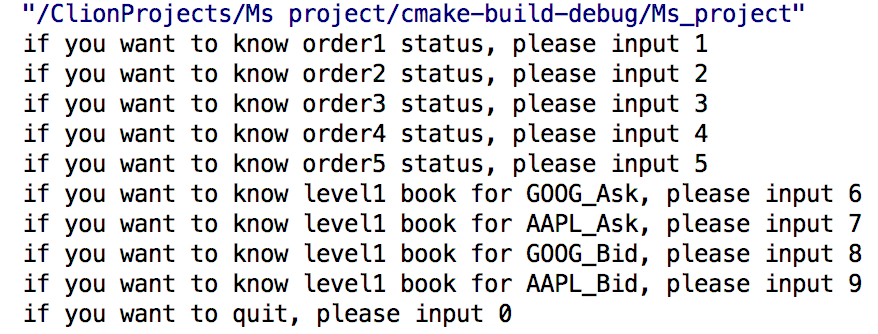
vectorOfList[index].IEX.price = Price;

string oneOutput = vectorOfList[index].output();

(\*temps[index]).push(oneOutput);

}

}

1.2测试运行结果：

1.3设计不当之处：

（1）错误处理较少。

（2）在使用多线程时，使用了较多的全局变量，代码可维护性较差。

2. MS\_iceberg:

2.1主要功能模块及其代码:

2.1.1Time\_order.h:

（1）Time 存储小时和分钟。

struct Time

{

int hour;

int minute;

Time(int h,int m):hour(h),minute(m){}

};

（2）TimeRange 枚举类型，存储一天中14个时间段。

enum TimeRange

{

first = 1, second, third, fourth, fifth, sixth, seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth, fourteenth

};

// first means 9:00 to 9:30 ……..

（3）Order存储文件中的一行数据，包含price，quantity，time。

struct Order

{

double price;

int quantity;

Time time;

Order(double p, int q, Time t):price(p),quantity(q),time(t){}

};

（4）getTimeRange 给定一个Time返回它所处的时间段。

TimeRange getTimeRange(Time t);//给定时间返回相应的时间段

（5）getTimeBase 给定时间段返回该时间段的基准值（默认为开头时间加5分钟）。

Time getTimeBase(TimeRange tr); //给定时间段返回时间

2.1.2Time\_order.cpp：

Time\_Order.h的实现。

TimeRange getTimeRange(Time t)

{

if(t.minute < 30)

{

if(t.hour == 9) return first;

else if(t.hour == 10) return third;

else if(t.hour == 11) return fifth;

else if(t.hour == 12) return seventh;

else if(t.hour == 13) return ninth;

else if(t.hour == 14) return eleventh;

else if(t.hour == 15) return thirteenth;

}

else

{

if(t.hour == 9) return second;

else if(t.hour == 10) return fourth;

else if(t.hour == 11) return sixth;

else if(t.hour == 12) return eighth;

else if(t.hour == 13) return tenth;

else if(t.hour == 14) return twelfth;

else if(t.hour == 15) return fourteenth;

}

}

Time getTimeBase(TimeRange tr)

{

switch(tr)

{

case first:

return Time(9, 5);

case second:

return Time(9, 35);

case third:

return Time(10, 5);

case fourth:

return Time(10, 35);

case fifth:

return Time(11, 5);

case sixth:

return Time(11, 35);

case seventh:

return Time(12, 5);

case eighth:

return Time(12, 35);

case ninth:

return Time(13, 5);

case tenth:

return Time(13, 35);

case eleventh:

return Time(14, 5);

case twelfth:

return Time(14, 35);

case thirteenth:

return Time(15, 5);

case fourteenth:

return Time(15, 35);

}

}

2.1.3main.cpp:

（1）readfile(ifstream&ist) 读文件返回数据类map<int,map<TimeRange, vector>>其中 int表示天。map<TimeRange, vector>中，TimeRange表示特定时间段， vector表示在该时间段中所有Order的vector。

map<int,map<TimeRange, vector<Order>>> readfile(ifstream&ist)

{

map<int,map<TimeRange, vector<Order>>>data;

string temp;

while(getline(ist,temp))

{

if(temp[1] == '\*') continue; //如果遇到分割线则跳过

stringstream ss;

ss<<temp;

int int1,int2,day,hour,minute,int3,quantity = 0;

double price;

char ch1,ch2,ch3,ch4,ch5,ch6;

ss>>int1>>ch1>>int2>>ch2>>day>>hour>>ch3>>minute>>ch4>>int3>>ch5>>price>>ch6>>quantity;

Time time(hour,minute);

TimeRange tr = getTimeRange(time);

Order order(price,quantity,time);

data[day][tr].push\_back(order);

}

return data;

}

（2）getQuantityRatio 经过统计处理，返回一个存储有各个时段的quantity占总quantity比例的vector。

vector<double> getQuantityRatio(map<int,map<TimeRange, vector<Order>>>&data)

{

vector<double> quantityRatio;

map<int, map<TimeRange, double>> RatioData;

for (int i = 10; i <= 24 ; ++i)

{

int sum = 0; // 当前一天内quantity的总合

for (int j = 1; j <= 14 ; ++j)

{

sum += getAllQuantity(data[i][(TimeRange)j]);

}

for (int k = 1; k <= 14 ; ++k)

{

double quantityInRange = getAllQuantity(data[i][(TimeRange)k]); //表示在该时间段内的quantity总量

double ratio = quantityInRange/sum;

RatioData[i].emplace((TimeRange)k,ratio);

}

}

for (int l = 1; l <= 14; ++l)

{

double sum = 0;

for (int i = 10; i <= 24 ; ++i)

{

sum += RatioData[i][(TimeRange)l];

}

quantityRatio.push\_back(sum/15);

}

return quantityRatio;

}

（3）getAveragePrice 返回各个时间段的均价。

vector<double> getAveragePrice(map<int,map<TimeRange, vector<Order>>>&data)

{

vector<double> AveragePrrice;

for (int i = 1; i <= 14; ++i)

{

double allCost = 0; //该时间段总的quantity \* price

double allQuantity = 0; //该时段的总的quantity

for (int j = 10; j <= 24; ++j)

{

allCost += getAllCost(data[j][(TimeRange)i]);

allQuantity += getAllQuantity(data[j][(TimeRange)i]);

}

AveragePrrice.push\_back(allCost/allQuantity);

}

return AveragePrrice;

}

（4）getQuantityRatioForTest 仅处理测试数据那一天。

vector<double> getQuantityRatioForTest(map<int,map<TimeRange, vector<Order>>>&data)

{

vector<double> quantityRatio;

double sum = 0;

for (int i = 1; i <= 14 ; ++i)

{

sum += getAllQuantity(data[25][(TimeRange)i]);

}

for (int j = 1; j < 14 ; ++j)

{

quantityRatio.push\_back(getAllQuantity(data[25][(TimeRange)j])/sum);

}

return quantityRatio;

}

（5）getAveragePriceForTest 仅处理测试数据那一天。

vector<double> getAveragePriceForTest(map<int,map<TimeRange, vector<Order>>>&data)

{

vector<double> averagePrice;

for (int i = 1; i <= 14 ; ++i)

{

double allCost = getAllCost(data[25][(TimeRange)i]);

double allQuantity = getAllQuantity(data[25][(TimeRange)i]);

averagePrice.push\_back(allCost/allQuantity);

}

return averagePrice;

}

（6）getAllQuantity 输入某一时段的orders，得到在某一时段的总的quantity

double getAllQuantity(vector<Order> &data) // 得到在该时段的总的quantity

{

double sum = 0;

for (int i = 0; i < data.size(); ++i)

{

sum += data[i].quantity;

}

return sum;

}

（7）getAllCost 输入某一时段的orders，得到在某一时间段总的quantity \* price

double getAllCost(vector<Order> &data) // 得到在该时间段总的quantity \* price

{

double sum = 0;

for (int i = 0; i < data.size(); ++i)

{

sum += data[i].quantity \* data[i].price;

}

return sum;

}

（8）output 输出测试阶段的数据在命令行中。

void output(int quantity, double price, Time time)

{

int outputQuantity = quantity/5;

int minute = time.minute;

int hour = time.hour;

if(minute == 5)

{

cout<<"2015-5-25 "<<hour<<":0"<<minute<<":00 "<<"price: "<<price<<" quantity: "<<outputQuantity<<endl;

minute += 5;

}

else

{

cout<<"2015-5-25 "<<hour<<":"<<minute<<":00 "<<"price: "<<price<<" quantity: "<<outputQuantity<<endl;

minute += 5;

}

for (int i = 0; i < 3; ++i)

{

cout<<"2015-5-25 "<<hour<<":"<<minute<<":00 "<<"price: "<<price<<" quantity: "<<outputQuantity<<endl;

minute += 5;

}

cout<<"2015-5-25 "<<hour<<":"<<minute<<":00 "<<"price: "<<price<<" quantity: "<<quantity-4\*outputQuantity<<endl;

}

（9）main 测试函数。

int main()

{

int quantity;

cout<<"Please input a big order"<<endl;

cout<<"quantity:";

cin>>quantity;

string s = "iceberg.csv";

ifstream ist(s.c\_str());

map<int,map<TimeRange, vector<Order>>> data = readfile(ist);

vector<double> quantityRatio = getQuantityRatio(data);

vector<double> averagePrice = getAveragePrice(data);

string s2 = "test.csv";

ifstream ist2(s2.c\_str());

map<int,map<TimeRange, vector<Order>>> data2 = readfile(ist2);

vector<double> quantityRatio\_former = getQuantityRatioForTest(data2);

vector<double> averagePrice\_former = getAveragePriceForTest(data2);

double totalCost = 0; //总的花费

int presentQuantity = 0; //当前总量

for (int i = 0; i < quantityRatio.size()-1; ++i)

{

Time time = getTimeBase((TimeRange)(i+1));

int quantityLimited = int(quantityRatio[i] \* quantity);

double price = averagePrice[i];

output(quantityLimited, price, time);

totalCost += quantityLimited \* price;

presentQuantity += quantityLimited;

Order order(price,quantityLimited,time);

data2[25][(TimeRange)(i+1)].push\_back(order);

}

int quantityLimited = quantity - presentQuantity;

Time time = getTimeBase((TimeRange)(14));

double price = averagePrice[13];

Order order(price,quantityLimited,time);

data2[25][(TimeRange)14].push\_back(order);

output(quantityLimited, price, time);

totalCost += quantityLimited \* price;

cout<<"totalCost: "<<totalCost<<endl;

cout<<"averagePrice: "<<totalCost/quantity<<endl;

vector<double> quantityRatio\_after = getQuantityRatioForTest(data2);

vector<double> averagePrice\_after = getAveragePriceForTest(data2);

cout<<"the list of quantityRatio\_former: ";

for (int j = 0; j < 13; ++j)

{

cout<<quantityRatio\_former[j]<<" ";

}

cout<<endl;

cout<<"the list of averagePrice\_former: ";

for (int j = 0; j < 13; ++j)

{

cout<<averagePrice\_former[j]<<" ";

}

cout<<endl;

cout<<"the list of quantityRatio\_after: ";

for (int j = 0; j < 13; ++j)

{

cout<<quantityRatio\_after[j]<<" ";

}

cout<<endl;

cout<<"the list of averagePrice\_after: ";

for (int j = 0; j < 13; ++j)

{

cout<<averagePrice\_after[j]<<" ";

}

cout<<endl;

}

订单发出方法：根据getQuantityRatio的数据，得到每一时间段应发出的订单的quantity，根据getAveragePrice的数据，得到每一时间段的均价。在一个时间段中，将quantity分为5份，每隔5分中发出。依次处理14个时间段。

测试方法：先处理当天数据，计算QuantityRatio和AveragePrice，然后将发出的订单加入到当天数据中，再计QuantityRatio和AveragePrice，并比较前后的差距。

1.2测试运行结果：