Project — An Online Auction House

Project plan due: Thursday 15 March
Presentation: late March / early April, in tutorial
Final project due: Thursday 12 April

Introduction

An online auction house, such as ebay.com, allows people to put items that they want to sell up for auction on the web, and allows others to bid on such items. When the auction for an item is over, the highest bidder gets to buy the item. It is then up to the seller and buyer to arrange payment and shipping.

You are going to write a set of programs to maintain part of the data for an online auction house. Don’t panic when you look at the example sites listed. They are pretty fancy, but your project would only be a small part of the software that runs such sites. For example, you will not have to build a user interface, or handle anything to do with member ratings.

Part of the work in this project will be settling on specifications for exactly what you want your programs to do and documenting these decisions in your report. We will put constraints on your choices, but some decisions will be up to you.

Rough specifications

There are a number of kinds of auction. Your auction house only offers “standard” auctions. With a standard auction, the seller sets a minimum price and only a bid at or above that price is accepted. Subsequent bids are accepted only if they exceed the previous high bid. At some point, the auction is over, and the highest bid (which will be the last bid because of our rules above) wins.

Data

There will be three main files of information:

- Members: For each member (both buyers and sellers), you will store information including the person’s unique ID number, name, email address, the number of items they’ve bought in the past, and the average price they’ve paid. This information about their buying history is kept because we hope to be able to produce and sell mailing lists of certain kinds of members.

  The members file must be indexed by ID number with a B+-tree, and by average price with a second B+-tree.

- Items: For each item that is currently for sale, you will store information including the item’s unique ID number, item description, the seller of the item, and the minimum bid.

  The items file must be hashed. You must decide on the hash function, bucket size, collision resolution scheme, etc.

- Bids: For each bid in a current auction, you will store information including the bidder and the amount of the bid.
The bids file will be “threaded” by item and by bidder. That just means that each bid record will contain a pointer linking it to another bid record for the same item, and a second pointer linking it to another bid record for the same bidder. For example, a bid for item number A59812 by member Fred for $3.75 would have a pointer connecting it to a linked list that contains all the bids on item number A59812, and another pointer connecting to a different linked list that contains all the current bids by Fred. Draw a picture of a threaded bid file to be sure you understand it. Note that elements of the linked lists are single bid records. This is unlike assignment 2, where you built nodes that contained multiple records. Think about why.

Notice that we have required several connections across the files. For example, each item has a seller, who must have a record in the member file. A connection from file A to file B can be implemented by storing in file A pointers into file B; or by storing in file A key values from file B. There are advantages and disadvantages to each approach, which you should carefully consider when you choose what to do in each case.

You may also choose to make additional connections across files.

Operations
You must provide at least the following operations:

- Add or delete a member.
- Search for a member by ID number.
- Print a list of all members whose average buying price is in some range.
- Print the item description and bid amount for all bids that a given member currently has.
- Put a new item up for auction.
- Retract an item that was put up for auction. This occurs when the seller has changed his or her mind about selling the item.
- Search for an item by ID number.
- Put a bid on an item, given the item’s ID number and the bid amount. If the bid amount is greater than the current top bid, it becomes the new top bid. Otherwise, the bid is rejected.
- Close the auction for an item, given its ID number. The top bid wins, and that member’s information must now be updated.

This is only a partial list of operations. Again, it is up to your group to decide on precisely what operations to support and the exact specifications of each.

Keep in mind that your file structures will make some operations very efficient, and others horribly inefficient. So you must plan your basic file structures with your complete set of operations in mind, or you may back yourself into a corner.

Make your hashing and \( B^+ \)-tree code general, so that they could be easily adapted to another domain, a different collision resolution scheme, a different branching factor in the \( B^+ \)-tree, etc.

Driver
You will need to write a driver program to test your code. A fancy interface is not required.

How much you must accomplish

Groups of two must implement at least the data and operations described above. Groups of three must also accomplish one or more other substantial task of their choosing. Possibilities include:
• Implementing extra data and/or operations, such as these:
  – Print the bid history for an item, given its ID number. That is, print all the bids that have been made.
  – Retract a member’s bid on an item.
  – Print all items currently for sale by a given member.
  – Allow a seller to have more than one of the very same item up for sale
  – Implement a members’ rating system. Since the online auction house cannot force sellers and buyers to behave honestly, most houses rate buyers and sellers based on what other buyers and sellers say about them. Anyone whose rating is too low may lose their membership.
  – Expand on the mailing list idea. Keep more information about each member and produce more carefully targeted lists.
  – Keep information about auctions that are over and use it to provide interesting operations about the history of the auction house and its members.

See the sample web sites for many other ideas.

• Implementing an incremental hashing scheme.
  \textit{i.e.}, a scheme for restructuring the file when the load factor has become too high and performance therefore has dropped. See chapter 11 of the text.

• Performing experiments to compare the performance of various hashing schemes.
  \textit{E.g.}, comparing various kinds of hash function, or collision resolution schemes.

I won’t tell you what or how much extra to choose. Aim to make your total workload per group member roughly what it would be in a group of two that is doing only the basics listed earlier.

\section*{Working in a group}

You will complete this project in a group of two or three people. Groups of three will be expected to complete a more ambitious project than groups of two, as described above.

Be sure to reread sections 5 and 6.2–6.3 of the 228 Course Guide; they address how to set up and organize your group, and how your group work will be marked.

I hope that working in a group will be one of the most valuable experiences you have in this course. However, you will have to make an extra effort to deal with the dynamics of a group if you wish it to be a valuable \textit{positive} experience. :-)

If your group is having difficulties working effectively, don’t sweep it under the rug and hope things will improve; they probably won’t. Talk about it, and if that doesn’t work, see me as soon as possible.

\section*{Using the code from your textbook}

The appendices of your textbook contain \texttt{c++} classes that partially implement B-trees. You are welcome to use this code by adapting it as necessary for your project. Because many of the B-tree routines are already implemented, using this code may save you a considerable amount of time. However, there are some hurdles you will have to overcome in order to use the code:

• The code uses objects and methods from other classes presented in earlier appendices. These include \texttt{SimpleIndex}, \texttt{IOBuffer}, \texttt{Record} and possibly others. In order to understand the B-tree code, you will need to understand these other classes.
• It does not implement deletion. You would have to add this, and it is the hardest part of the B-tree code. (Of course, the alternative is to write deletion plus everything else.)

• It does not quite implement a B⁺-tree. By B⁺-tree we mean a tree in which the actual records are stored only in the leaves and there is a linked list joining the leaves together. The book implements B⁺-trees in which the records are only at the leaves, but does not contain the pointers to join the leaves into a linked list. You would have to modify the code to add and maintain these pointers.

• There may be bugs in the code. However, some students have used it before with success (after a significant investment to learn it and modify it).

You probably noticed when reading through the presentation of B-tree operations in chapter 9 that the text uses a slightly different sort of tree node. In lecture, we talked about B- and B⁺-trees that have k child pointers and k − 1 keys per node (and k has minimum and maximum possible values that depend on m). In the text there is an extra key in each node. This changes the algorithms slightly from how they were presented in class. The version described in the text (and in the appendix code) is probably a little easier. Either way is fine with us.

In the real world, programmers often have to make decisions about when to use and adapt legacy code and when to create routines from scratch. You need to consider this decision as a group as you plan your project.

The phases of the project

Your project plan

Your project plan should include the following information:

• Group
  The name of your group, your group members their individual TAs, and the TA to whom you are submitting your project. Write this on the project cover sheet, available on the course web site. Remember that you needn’t all be in the same tutorial; in fact, it is an advantage if you are not, because your group will have the benefit of getting guidance from more than one tutor.

• Domain
  What properties of the domain (an online auction house) affect your design decisions (e.g., “There are far more of this data than that data, so . . .” or “Operation such-and-such will occur very often, so . . .”). If you don’t think carefully about the domain, the rest of your project may not make sense.

• Data
  An outline of what data you plan to store, which will include what we’ve listed. Describe any relationships across data sets (e.g., “ID numbers in the members data file correspond to those in the bids file”).

• File Structures
  For each set of data, a description of the file structure. Will it be hashed, or indexed?, what sort of collision resolution scheme will be used?, will it be linked to any other file?, will the member thread in the bid file be sorted by any key? etc. Here you are describing the structure of the file, not the details of how you’ll lay it out (variable length or fixed length records, etc.). Think carefully about your decisions — they will affect what operations will be possible and efficient.

• Operations
  A list of the operations you plan to handle, with a brief but precise specification of what each one will do. Again, this will include the operations we’ve required.

• Special Features.
  Any special features you plan to include in your project.
• Plan of Attack and Schedule
A detailed breakdown of tasks. For each task, specify who will focus on it, its planned completion date, and whether it is core or a fancy extra that you hope to have time for.
You can think of the work to be done as a two-dimensional table, with components of the program down the rows and phases of development (settling the specifications, design, coding, testing, report writing) across the columns. It is acceptable for group members to each focus on different aspects; however, it is not acceptable to divide things up simply by row or column – for example to have one team member do everything to do with hashing, or everything to do with writing the report.
The plan of attack and schedule is probably the most important part of your Project Plan. Do not neglect it.
You will be allowed to change your mind later about anything in your project plan, but of course, the better thought out your plan is, the fewer changes will be necessary. Be prepared also that your TA may find flaws in your plan, and that this may lead you to make changes part way through the project. Accommodating changes will be much easier if you have designed your code well.
I would rather you get a modest project done really nicely (including robust code, thorough testing, and well-written report), than over-extend yourself and end up with a very ambitious project done poorly. And yes, a well done but modest project will get a better mark.
A wise strategy would be to plan to complete a modest project, and then if you can, to improve or extend it. Explain this strategy in your Project Plan (including exactly how you hope to extend your project). In order to work this way, you must design your code well, so that pieces can be plugged in and out.

Your group presentation
During tutorials in late March and early April, each group will make a brief (roughly 5 minute) presentation of their project, discussing their overall design strategy and special features of their system. In most cases, the group should select one member to do the actual speaking, but every member should participate in planning the presentation.
We will say more about the presentations in an upcoming tutorial.

Your final report
Your final report will be significantly longer than the reports for assignments 1B and 2B, but shouldn’t be longer than about 10 pages.
Include the usual sections of a report, as well as all the sections listed under the project plan above. It is okay to reuse parts of what you handed in for your plan, but you should to update it to reflect changes to your plans, and you should expand upon the design decisions you made, alternatives your rejected, tradeoffs, etc.
In addition, you must submit the following sections:

• Who did what.
  Each group member must submit their own brief assessment of who did which work on the project. Even if your group writes this “who did what” statement together, each person must hand in their own copy. These statements should be brief — substantially less than a page long.

• Teamwork.
  A brief description of how you handled the teamwork, beyond who did what. Was one person in charge of everything? Was a different person in charge of each of several sub-tasks? Or was it a more democratic team structure? How did you run your team meetings? What problems arose in your team? How did you solve them?
Approaching the project

You will save yourself a great deal of time and effort overall if you take the time to do a good job in the early stages of the project. You should have a clear specification of what you plan to do before beginning to do it. And you should design your code carefully from the beginning, rather than think that you will reverse-engineer a good design after it’s done. Don’t forget the lessons learned in CSC148 and beyond about modularity, abstraction, and information hiding.

You may find it necessary to reduce your ambitions, and you may learn things later that lead you to change aspects of your system. For both reasons, it is important to begin by isolating the essential operations of the system and designing a modular structure to accommodate later changes. This is not a difficult task, but you must remember not to rush straight into coding. If, on the other hand, you find that you have quickly succeeded in building a basic working system, you may wish to add extra features to your program.

It will be natural as you plan your system to think about how to describe it in your report. The report should not be written at the end of the project. Rather, you should write an outline at the beginning, and continue to expand it as you proceed, until by the end of coding and testing, the report is almost complete.

You should also begin work on your testing strategy very early in the project. If you leave testing to the last minute, your group will likely find itself in trouble. Try to build up the system from a simple skeleton to a fully functional package in gradual stages, testing various components as you proceed. With the confidence of having something simple working soon, you can go on to grow a bigger project successfully.