University of Toronto
csc228 — **File Structures and Data Management, Spring 2000**

Project — Tracking TARs (Technical Assistance Requests)

Project plan due: Friday 10 March, in tutorial
Presentation: Friday 31 March and 7 April, in tutorial
Final project due: Friday 7 April, in tutorial

Introduction

Large software companies often include entire divisions to provide technical support to their customers. Customers make technical assistance requests (TARs) either by phone or on-line. Technical support employees are assigned the TARs based on the type of the problem and the employee's area of specialization. A TAR is closed once the customer is satisfied that the problem is resolved. During the time that a TAR is open, a number of employees may work on it but at any one time it is assigned to only one employee. The TAR can be reassigned to another employee for a number of reasons. As progress is made on a TAR, employees create a log of actions taken on that TAR. Customers can be very large companies with many different problems. Consequently the same customer can have a number of open TARs at the same time.

Your company has a great many customers, each of whom will have many different problems, and a lot of information is being stored about each of the problems. In addition, the entire history of all problems is kept so that it can be looked back upon to find out interesting things like the average time taken to solve problems, by department, in 1998. This means that there is a vast amount of data, and that we need efficient ways to access it.

You are going to write a set of programs to record and update TARs. Part of the work will be settling on specifications for exactly what you want your programs to do, and documenting your decisions in your report. We will put constraints on your choices, but many decisions will be up to you.

Rough specifications

Data

There will be four main files of information:

- Customers: For each customer, you will store information including the customer's unique ID number, company name, phone number and the date when their contract with your company for technical support ends. This file must be hashed. You must decide on the hash function, bucket size, collision resolution scheme, etc.
- TARs: For each TAR you will store information including the customer number, the date the TAR was opened, the employee currently assigned to the TAR and the date the TAR was closed if appropriate. The log of activity on the TAR is in a separate file.

  The TAR file will be "threaded" by customer and by employee. That just means that each TAR record will contain a pointer linking it to another TAR record for the same customer, and a second pointer linking it to another TAR record for the same employee. For example, an open TAR placed by "Ontario Hydro" and assigned to "Vincent Superstar" would have a pointer connecting it to a linked list that contains all the TARs placed by "Ontario Hydro" and another pointer connecting to a different linked list that contains all the open TARs currently assigned to "Vincent Superstar". Draw a picture of a threaded TAR file to be sure you understand it. Note that elements of the linked lists are *single* TAR records. This is unlike assignment 2, where you built nodes that contained multiple records. Think about why.

Notice that although we have specified some of the details about the threading, we have left a number of design decisions to you including whether to thread open and closed TARs separately.

- Employee index to the TAR file: You must also implement a B^+-tree tree which will allow you to access the TARs by employee number. The employee numbers used in recording TARs encode more than just when the employee arrived at the company. They are assigned according to the division, department and group to which the employee belongs. This means that it would be meaningful to look at the TARs for a range of employee numbers. For example we may wish to know how many TARs are open for employees with numbers between 120000 and 140000. Other than a pointer to the appropriate thread in the TAR file, you might also decide to keep other information in the leaves of this B+ tree.
- TAR log file: This file will contain a record of all the actions taken for every TAR. Each entry includes at least the date, the identity of the employee making the entry, and a text description of what happened. When the TAR is eventually closed, the fact that it has been closed is also included into the log.

  You should have one large file rather than a separate file for each TAR. There is no limit on the length an individual log entry or on the number of log entries per TAR.

  All the design decisions for this file's structure are left to you.

Notice that we have required several connections across the files. For example, the employee index refers to items in the TAR file. A connection from file A to file B can be implemented by storing in file A pointers (either RRNs or byte offsets) 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 customer[1].
- Add or delete an employee. When an employee is deleted, another employee is assigned all of his or her TARs. The user must specify which employee will be assigned them.
- Add a new TAR. This includes inserting it into the TAR file and updating the employee index on the TAR file. Your operation may also make an initial entry in the log file at this time. enough information about the TAR that.
- Log a new action on a TAR.
- Reassign one TAR to a different employee.
- Close a TAR. This involves updating the TAR's record in the TAR file to indicate that it is closed, and adding a final log entry for the TAR. If open and closed TARs are threaded separately in the TAR file, the threads will also have to be updated.
- List all the open TARs for a specific customer.
- List all the open TARs assigned to employees with employee numbers in a given range. This range could cover any number of employees, or just one.

[1] We realize that in a real system you would probably never choose to delete a customer but we want you to include this operation here.

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:
  - Assign the TARs a number based on the type of the problem the customer is having. Add the ability to query the TAR file for related TARs by having a B+ tree using the problem number as the key.
  - Add a new field to each TAR record that indicates the "status" of the TAR, and add an operation for searching through the TARs by status[2].
  - Count all the TARs (open or closed) for a specific customer over a certain date range.
  - For a given employee range, which could include 1 employee or more, calculate the average "closure time" for the closed TARs over a certain date range — that is, the average amount of time between when the TAR was initially added to the system, and when it was marked as closed.
  - For a given employee range, which could be 1 employee or more, calculate the average "age" for the currently open TARs.
  - Find the monthly average number of logged TARs over the past month.
  - Add the ability to give TARs a priority and then treat them differently based on priority.
- Implementing an incremental hashing scheme.
  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.
  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.

[2] The status code will be an integer that indicates the current state of the TAR. For example a status of 0 might represent that the employee has called back the customer requesting some new information and the customer hasn't yet replied. A status of 1 might mean that the employee is waiting for a software patch from the development group. If you implement this feature, you can just use a fixed range of integer values for the status values; you don't need to make up meanings for them.

Driver

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

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 *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.

Using the code from your textbook

The appendices of your textbook contain 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 functions are already implemented, using this code may save you some 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 SimpleIndex, IOBuffer, 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.
- 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 are bugs in the code.

In the real world, programmers often have to make decisions about when to use and adapt "legacy" code and when to create it 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 (technical assistance requests) 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 at least what we've listed. Describe any relationships across data sets (*e.g.*, "ID numbers in the file A correspond to those in the file B").

- **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 employee threads in the TAR 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. Most teams neglect it and later suffer for that.

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 report for assignment 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.

To which TA should you hand in your project?

This section applies only to St George students.

If all members of your group, or the majority of members, have the same TA, hand all phases of your project in to him or her. Otherwise, you may pick which one of your TAs to hand your project in to.

After the first phase (the Project Plan) is due, I may redistribute some groups to a different TA, to even out their marking workload.