Project list for Programming Techniques Spring 2016

Costin Badică, Alex Becheru

1 Summary

You can choose any of the problems, libraries or applications below as your project. Note that textbook problems and libraries are a one man project whereas most of the applications are a two man effort. Besides the C implementation a correct Python implementation will give you additional points.

Some of the requirements ask you to implement data structures that you probably have not encountered (e.g. tries, skip lists, bloom filters). If you choose any of these and you can not find relevant info on implementing it you can always ask any of the authors and we will try to give you detailed information and pointers. Of course this applies to every proposal of this document.

If you have other ideas for a project besides the ones listed in this document we can discuss and if the idea is acceptable you can use it as your project.

All projects must be accompanied by the project documentation. The documentation must respect the template available on the course web-page. In order to typeset your documentation you'll need LaTeX system.

- *Windows*. You can use MiKTeX `http://miktex.org/download`. You can edit your LaTeX documents in any text editor, but Texmaker has some features that make your life easier `http://www.xm1math.net/texmaker/`.
- *OS X*. You can use MacTeX `http://www.tug.org/mactex/`. It comes with TeXShop, an editor with a decent set of features to get you started editing LaTeX.
- *Linux*. You can use your distro's package manager to install a LaTeX system. You can use Texmaker `http://www.xm1math.net/texmaker/` to edit your LaTeX sources.

Costin Badică
Department of Computers & Information Technology, e-mail: `cbadica@software.ucv.ro`

Alex Becheru
Department of Computers & Information Technology, e-mail: `becheru@gmail.com`

- *Web*. You can also use online LaTeX editors thus eliminating the need of installing various packages and/or programs. A free web editor for LaTeX can be found at `http://www.sharelatex.com`.

2 Textbook problems

Since the problems are not complex enough to justify a two-man team, youll have to *work alone*.

1. (Individual) *The change-making problem*: How can an amount of money be made with the least number of coins of given denominations?
2. (Individual) *The rod-cutting problem*: Given a rod of length n and a table of prices for rods of length from 1 to n, determine the maximum revenue obtainable by cutting up the rod and selling the pieces. Note that an optimal solution may require no cutting at all.
3. (Individual) *The eight queens problem*: Place eight queens on an 88 chessboard so that none of them can capture any other using the standard queens moves. The queens must be placed in such a way that no two queens attack each other.
4. (Individual) *The task-scheduling problem*: Schedule several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities. Given that an activity has a start time and a finish time two activities are compatible if the intervals between theirs start and finish times don not overlap.
5. (Individual) Implement two algorithms for minimum spanning trees (e.g. Prim algorithm and Kruskal algorithm).
6. (Individual) *Sudoku*. Implement an algorithm to resolve the Sudoku game. The Sudoku board should be read from a file, and the solution written to another file.

3 Libraries

Since the problems are not complex enough to justify a two-man team, youll have to *work alone*.

1. (Individual) *A library for safe string manipulation*. The library will implement alternatives for the most common string manipulation functions: strcat, strcpy, strlen and gets for reading strings. The library functions must perform sanity checks to avoid buffer overruns and to ensure that all strings are null-terminated after each operation.
2. (Individual) *A library for linked lists*. The library should implement singly and doubly linked lists. The operations provided by the library should be: initialisation of an empty list, adding a value at the beginning and at the end, inserting an

item at a specified position, removing an item at a specified position, computing the length of a list and appending two lists.

3. (Individual) *A library for binary search trees (BST)*. The library should provide opera- tions for: creating an empty tree, inserting a node into a tree, deleting a node and at least two different traversal strategies the traversal functions must accept a function which will be passed the current element.
4. (Individual) *A library for priority queues using heaps*. The library should implement both min-priority queues and max-priority queues. The library should provide operations for inserting an element with a certain priority, finding the maximum/minimum element, extracting the maximum/minimum element and altering an elements priority.
5. (Individual) *A library for hash tables*. Collisions should be resolved by chaining. The size of the hash table should be dynamically adjusted to accommodate as many keys as necessary.
6. (Individual) *A library for operations with large numbers*. The library should provide at least addition, subtraction, multiplication, division and taking the square root.
7. (Individual) *A library for operations with sparse matrices*. The library should provide at least addition and multiplication.
8. (Individual) *A library for associative arrays implemented using prefix trees (tries)*. The array will accept string keys and values of any type and it will expose the following functionality: adding a new value indexed by a key, reassigning the value for a key and look-up for key values.
9. (Individual) *A library for representing oriented graphs*. The library should allow to insert vertices, remove vertices, traverse the graph depth-first and breadth-first.
10. (Individual) *A library for disjoint sets*. Given a set of elements it is often useful to break them up or partition them into a number of separate, non-overlapping sets. A disjoint-set data structure is a data structure that keeps track of such a partitioning. The library must allow to determine which set a particular element is in (also useful for determining if two elements are in the same set) and combine or merge two sets into a single set.
11. (Individual) *A library for skip list*. The library should allow inserting, deleting and searching for an item.
12. (Individual) *A library for bloom filters* . The library should allow inserting and searching for an item.
13. (Individual) *Text editor library*. Write a library with functions that could be useful for a text editor. The library should at least contain the following functions:

    - determine if a word is present in a document.
    - determine the number of occurrences for a word in a document.
    - replace all instances of a word in a document with another given word.
    - replace specific instances of a word in a document with another given word, e.g. replace the 3rd appearance of a word with another given word.
    - create a sorted list of words by the number of occurrences in a document.

14. (Individual) *A library for matrix functions*. Write a library with functions usefull for a matrix. The library should at least contain the following functions:
    a. add two matrices
    b. rise a matrix to a given power
    c. determine the elements on the secondary diagonal
    d. determine if the matrix is upper triangular or lowe triangular.
    e. determine if a matrix is mirrored considering either the the first or second diagonal.

4 Applications

Applications 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 and 13 are complex so they will require a two man team.

1. (Individual) *Todo list* Write a command line application that will have the functionalities of a todo list:
    - Prompt the user to enter a chore or task. Store the tasks in a permanent location so that the tasks persists when the program in restarted.
    - Tasks may be categorised by into different categories, allow the user to interact with these categories (create, delete, modify).
    - Allow the user to enter as many tasks as desired but stop entering tasks by entering a blank task. *Do not store the blank task*.
    - Tasks may also have priorities, allow the user to sort the tasks by priority.
    - Display all tasks, or display by category.
    - Allow the user to remove a task, to signify it's been completed.
2. (Individual) Since programming is such a sedentary job programmers need to watch their weight. The programmers in a company have access to all sorts of snacks in the cafeteria (with varying caloric intake), but they also have access to a gym and various activities that help them burn calories so that they maintain their weight. Given a list of activities and their caloric impact write a program that finds a combination of activities that keep the caloric intake to 0. Your program should take its input from a file. You can use any format for the input as long as it gives the name of the activity and the caloric intake (which should be a positive for snacks and negative for activities that burn calories). Print the list of activities to stdout if a solution is found or the message no solution otherwise.
3. (Team of 2) Write an interpreter for a simple language that recognizes both negative and positive integer constants, variable assignment and the basic mathematical operators: $+, -, *, /$ and $\wedge$ for exponentiation. Once read the expressions should be represented as binary trees which will be used to compute the result. The interpreter should present the user with a Read Eval Print Loop (REPL). An example session in the interpreter would look like this:

```
> a = 6
- 6
> b = 2
- 2
> a^2 + 2*b + 2
- 42
```

Where > is the interpreter prompt and - is prepended to the results of the last expression evaluated.

4. (Team of 2) Write a simple shell with the following built-in functionality: listing files (sorted in natural order), showing the content of a file and showing the number of words and lines in a file. You can see how the linux ls -l, cat and wc commands behave for inspiration.
5. (Team of 2) Write an interpreter for evaluating polynomials represented using linked lists. The interpreter should allow for addition, subtraction and evaluation of a polynomial. The interpreter should provide a syntax similar to that specified in problem 2.
6. (Team of 2) Write a dictionary application. The dictionary should allow for inserting a word, updating the definition of an existing word and looking up definition of a word. Note that a word may have multiple definitions attached to it. The dictionary should use files for storing and loading its data.
7. (Team of 2) Write a spell checker using the Levenshtein distance. The spellchecker will take an input file and it will output a list of misspelled words and the suggestions for them on stdout.
8. (Team of 2) Write a database system that will keep track of the students (name, grade and attendance rate) in Programming Techniques. The database will use binary files for storing data and it will provide the following operations: insert a new entry, update an entry, delete an entry and do queries (by name using the soundex algorithm, by grade and by attendance rate).
9. (Team of 2) Write a program that generates random text that reads well by using the Markov chains. The program will read text from a file. Start by reading the first two words (i.e. sequences of characters delimited by one or more spaces) from the file, say w1 and w2. Using the pair (w1, w2) as a key insert the word following them, w3, in a hash table. Replace the pair (w1, w2) with (w2, w3) and repeat until the end of the file. Note that each key made out of a pair of words will have a list of words as a value. In order to generate text start with the first key in the hash table (w1, w2), print w1 and w2, and then randomly pick one of he words in the list associated to the key, w3, and replace the key by (w2, w3). Repeat the process for a fixed number of steps.
10. (Team of 2) Write an application that shows information about train schedules and determine the shortest or cheapest route for a train from point A to point B. The routes and other data must be kept in files that can be updated via the application. Additional restrictions by the user need to be considered when suggesting a route:

    a. train conditions, e.g. if the train has a restaurant waggon or not.

b. time frame conditions, e.g. if the suggested route is available for a certain day.
c. route restrictions, e.g the client does not want to pass through a specific town X, therefor all the routes passing through X must be ignored.

11. (Team of 2) Write a calendar application. A user should be able to add/delete activities and visualise them. When creating a calendar event you should consider the possibility that the event may occur again. Also the user might need to a add additional information like: location, time frame, other participants, importance etc. When visualising the calendar the user should be able to:

    a. determine the events for a specific day/ week/ month.
    b. enquire whether there are no events at a certain date and/or at a certain time frame.
    c. determine whether he/she will be meeting a particular person in a specified time frame.
    d. view only the most important events in a specific time frame.
    e. view all the events that will take place at a certain location for a certain time frame.
    f. determine if a person will be at a certain location during a specified time frame.

12. (Team of 2) Write a genealogy tree application. Through this application a user should be able to add relatives (saved in a file) and make diverse enquiries:

    - determine if a certain person has a specific role: uncle, grandparent, cousin, etc.
    - view a list of all uncles, first degree cousins, aunts, etc.
    - visualise the entire genealogy tree, or parts of it.
    - determine the relatives that are not currently married.
    - determine the most common boy or girl name in the family.
    - determine the fist common ancestor of two family members.
    - show all members of the family for a specific generation, 1st generation represents the first ancestor of the family.