Third Year Project 2013/2014

Report 3: Data Structures.

A Sudoku Solver Using Constraint Satisfaction (& Other Techniques)

Thomas Paul Clarke

A report submitted (in part fulfillment) for the degree:

BSc (Hons) in Computer Science

Supervisor: Dave Cohen

Data structures are concepts and processes for handing the data within a system and each provides its own benefits and reasons to be used. Using these structures allows programmers to use them to increase performance within their programs, which could include processes that allow effective searching and sorting. However, each type has its disadvantages.

To explain data structures and what they are I will use the example of an array, which is a basic way to store data. It is a fixed size, therefore, only a specific number of elements can be within it, and the data is then held. However if the size of the array needs to be increased, the array has to be drained of its elements and re-created in the right size. Although there are other alternatives to using an array, for example, array lists, which are detailed later in my report. The functionality of a list (that helps the array problem) is that it doesn't have a specific size and it adapts to the size of the data as it's put in.

The example above proves that sometimes data structures need to be used in place of others to provide functionality, which otherwise cannot be there. Alternatively, it will provide a much easier way to write the program with the data structure that is best suited. In the case of this array, when the size of it will be changed throughout the program, the array list can be used. This will be treated the same way just using get and set methods. ans = list.get(index); Instead of an array which is done by using: ans = array[index]; Without this structure it would cause a lot of problems for the user having to replace the old array and change the size, whilst also repopulating it. However this solution also allows it to work without those issues and this is what the data structures are for.

Another example of a data structure is a linked list, which can be used instead of an array list. It gives similar functionalities that allows the set methods, delete and element functions to be implemented. However it does not provide a function for getting a specific item without traversing through the list. Using this as a linked list over an array list is provided it's not used to source a specific element. It's a much faster way to add and remove elements by simple traversing. Therefore, it highlights how important choosing the correct data structure is and having to know what the program will be doing to decide what is needed.

Throughout my concept programs and main project I will be using many different data structures to allow my program to work effectively and correctly. Therefore, the right structures have to be used and well executed. The concept programs have provided the foundation of what is needed with the final program and each has highlighted one or more structure.

A tree structure conceptualizes that all data is linked and can be traced via traversing down the tree. Its method is to create by nodes that contain data within the object, which are then stored within the tree. Each node is connected to the next because it stores the next node within the current.

The use of a tree structure is to allow the input data to be easily searched or found, which can be done by adding the nodes that contain the data set and traversing though them. This can reduce search time, therefore each operation requires O(log(n)) time when n is the size of that data set. This data structure also allows efficient removal and insertion allowing the tree to be traversed to add a new node.

Dynamic Trees allow the process for creating a tree during runtime, which will be subject to constant change. It can be added to after the process of creation and this will allow the tree to be made larger. It can also be cut when needed and by using the board states that are no longer required it is possible to trim that branch and dispose of the unnecessary leaf. This highlights the features within the tree of a tree that allow nodes to be added, searched for and removed. These features are valuable to tree functions and this also allows limitations to be placed. For example, when a node is removed, it is known and recognised, and the tree will remake itself to shift the other nodes around the tree if needed.

The advantages of using a tree is that the data set is easily entered into the tree, which (once within) can be sorted and compared to another. It does not matter if the tree is unbalanced because the dynamic tree allows the tree to be traversed quickly in O(log(n)) time. Therefore, it does not lose its functionality due to the use of the dynamic tree methods.

This process is more flexible in comparison to the arrays because when a new object is added, the tree is sized to the current number of nodes instead of a set number. For example, an array is set to an integer value, and when an object is removed or added a temp array has to be created to add the new values, so the new array is re-created to correct size. However this is built dynamically and will never be smaller or larger than needed.

Stacks are an abstract data type, meaning that they are defined with a set number of operations. In this case, pop, push and peek allows them to be categorised as abstract, which is a form of simplifying their concepts because they can be limited and described with these functions.[1] They are a way of storing elements in an ordered fashion because there is a strict order to the stack on how elements can be inserted or taken from the stack. This can be beneficial when a program needs to be able to store elements but only when dealing with one at a time or when storing that element for later.

This is done by using the simple commands of push and pop,

[1] Cay Horstmann Big Java Forth Edition Page 629-631

which essentially to input the element into the array one at a time. It is a last in first out data structure, which only allows the top element to be taken from the stack. The push method simply adds the element to the stack and the pop does the reverse where the element is then removed.

The logic in using a stack is due to its limitations actually helping the functionality of a program to be automated. For example, there is the Towers of Hanoi problem, which uses a stack to push and pop the elements or disks in this case.

A Queue is linked and relates very closely to a stack. However it is created using a linked list and the operations it uses differ slightly. The add operation adds the elements to the tail of the queue and the other operations remain the same, peek and remove. This allows the reverse functionality between a stack and a queue, which neither are able to do without the operation. The stack cannot return its bottom element until that element is next and the queue will not return.

Priority Queues include elements that have an assigned number, which relates to the order the elements can be arranged in. This is done by adding each element with its priority number, which gives the order the data is output. It allows the items to be stored in order when they are added to the queue. The queue will then extract the minimum number in the stored elements when the q.remove() method is called.

The reason to have these kind of data structures is to allow the abstract data types like these to simplify the coding process and provide useful limitations that allow only the specified commands to be done.

They also allow the dynamic and faster input of elements than an array could provide.

Heaps are structured differently to trees because each sub tree is higher than the root. Yet, there are no restrictions on the value of each node and its position. In the case of a binary heap each parent only has two nodes and in other heaps there are restrictions on the amount of children each parent can have. Heaps are related closely to priority queue and these are defined as the abstract data type of a heap.[2]

A heap is populated by adding a node to the end of the tree, demoting the parents' slots if it is larger than the element to be inserted. This is done to move up the node, so that at the end, the new node will be in a position where both of its children are smaller than its element. However this example has been putting the smallest element at the root, which can also be the largest element as the root.

In order to remove the element at the top of the heap, you use the function that will take away the root node and move the latest node into the root. However this causes problems as it likely that the heap will not match. It will have to be fixed by resorted it, which is achieved by adding a new element.[3]

An advantage of heaps is their ability to be versatile within using different data types. For instance, it is possible to use an array or an

[2] http://en.wikipedia.org/wiki/Heap_(data_structure)

[3] Cay Horstmann Big Java Forth Edition Page 652-663

array list to store the elements, using the layers of the heap to be stored in the arrays. This increases its efficiently as the elements are set in their representative locations, which are predefined.

Hash tables are created using the hash function, which creates a code for objects that are different. For example, if there are many elements within a data set that are likely to be reproduced within it, then it is worth using a hash table that uses the hash function to create a unique code. It is important to remember that in some cases there are exception, but they are not common. Yet more commonly the function will be used within the hash table to be a position within an array that is limited to the size a user specifies. If there is a duplicate then it will be stored within a linked list with the other hash coded elements.[4]

To find an object within the hash table each element has to calculate its hash code and reduced modulo to the size of the table because this provides the location within the table. Each element becomes within the location check if they are equal to the given element. If found then x is within the set. To delete an element the same process is repeated. However, if it is found within the set it’s removed.

The advantages of hash tables are their rapid increase in efficiency. This means (within the best case) it takes O(1) time to add, remove and delete the hash table elements. This provides a huge advantage to any program that has a large data and a set of related data

[4] Cay Horstmann Big Java Forth Edition Page 652-663

that needs to be found within the data set.

However the disadvantage of hash tables (in the worst case) is when all of the elements are given the same hash code and are all stored within the same linked list. This means that all of the search functions will still have to traverse between every element, which saves no time.

When programming my concept programs the data structures I've used and encountered for the first time have allowed new functionalities and more effective ways to increase the effectiveness of the program. Each data structure discussed within this report will be included within the final project, the Sudoku solver:

Priority Queue: This will be used to store the data, which will be assigned priority. This can be achieved by using a process which scores the most effective next move, so the board can be scored and stored as a priority queue. This means that only the highest score will be an output, which could provide a quick and easier way than comparing each board.

Hash tables: It is possible to use this within the Sudoku solver to identify each board/game state that is stored in the hash table. This is a much faster process to retrieve the boards than using a tree or any other data structure.

Trees: These allow the functionality for storing the boards (that will differ from each other) and finding and traversing a tree will allow these boards to be found. There are also functions that will aid this, for example, the tree increasing in size whenever a new board is input dynamically. It is also possible to limit or cut a branch when needed to if the boards are then repeated.

These data structures will be a key part within my finial program. They will aid with the fundamental aspects of handling the data and storing it in the most effective way possible. They'll also assist in outputting that data in the best way possible and (in some cases) in a system that allows it to be automated within the program, which enable it to remain consistent and constant.