[0910] EXTRA: Data Structures

Data Structures help programmers store and retrieve data in memory. There are many different types of data structures each with advantages and disadvantages for using them. C# provides many of these data structures as part of its Collections classes. Programmers need to know which kind of data structure to use based on those "pros" and "cons." Here is a quick review of the more important ones:

Arrays

Arrays are the simplest (and some might say most primitive) data structure. Arrays are very quick at data retrieval and are extremely memory efficient but that efficiency comes at the cost of slow inserts and deletions. While C# arrays are not really part of the Collection framework, in most places where you can use a Collection class, you can use an array too.

Linked Lists

Linked Lists are similar to arrays in the sense that they contain an ordered list of data items. Each item is stored inside a "node" of the list. In addition to the data item, each node contains a pointer (reference) to the next node in the list. (The pointer in the last node of the list is set to null.) Linked Lists have one big pro and a big con. The pro is that it is very easy to insert (or delete) an item from the middle of the list (you just create the new node and reassign two pointers). The disadvantage is that retrieving an item can be slow as the entire list may need to be "walked" before the needed item is located.

Stacks

Stacks store items in a "Last In, First Out" (LIFO) buffer. If three items (1, 2, 3) are stored in a stack, they will be retrieved in the order 3, 2, 1. Traditionally, when an item is stored on a stack, we say that it is "pushed" onto the stack. When an item is retrieved from a stack, we say that it is "popped" off of the stack. Stacks are very useful if you have data that fits into the LIFO access pattern. A good example is local variables in nested method calls.

Stacks are often implemented using an array of data and a member variable called the "stack pointer" which always contains the index of the array cell with the most recently pushed data item. If the stack is empty and someone tries to pop another item off of it, a "Stack Underflow" error occurs. If the stack is full and someone tries to push another item onto it, a "Stack Overflow" error occurs (although, in reality, the stack is usually just resized to allow more items.)

Queues

Queues store items in a "First In, First Out" (FIFO) buffer. If three items (1, 2, 3) are stored in a queue, they will be retrieved in the order 1, 2, 3. Queues are typically used as buffers - places where data comes in a faster rate than it goes out. Putting data into a queue is called "enqueuing" the data. Retrieving data from a queue is called "dequeuing" the data.

Maps

Given a unique "key" value, a map can store and retrieve an associated data item (the "value"). Maps are also called "Dictionaries." Maps can be implemented via a variety of different underlying data structures like arrays or lists.

Hashtables

A hashtable is a type of map that can store and retrieve key-value pairs very quickly (usually). Hashtables use lots of memory to achieve their speed - typically 2 to 3 times more memory than the amount needed for the actual data items.

A hashtable takes the key value and uses a "Hashing function" to turn that value into a (hopefully) unique integer index into the underlying array. The hashtable then looks in that location for the data item that corresponds to the key. As a hashtable become full, the hashing function will begin to return indexes that are already in use. That is called a "collision" and hashtables need to watch out for those and handle them when they occur. Eventually, if it continues to fill up, the hashtable will need to be resized. For more details, see **https://en.wikipedia.org/wiki/Hash_table (https://en.wikipedia.org/wiki/Hash_table)**

Trees

Trees are like linked lists on steroids. Each node in a tree can be linked to one or more other nodes. Trees are used in many different ways - often to maintain a (usually sorted) hierarchy of data objects. A special kind of tree where nodes cannot be linked to more than 2 nodes is called a "Binary Tree."

Others

There are several other important, but less common, data structures that programmers should be aware of - Sets, Graphs, Heaps, etc. For more on them and more on Data Structures in general, please review this article:
https://medium.freecodecamp.org/10-common-data-structures-explained-with-videos-exercises-aaff6c06fb2b (https://medium.freecodecamp.org/10-common-data-structures-explained-with-videos-exercises-aaff6c06fb2b)