**DSA Prep**

# Linear Structures

**List** – auto resizable array. Used when accumulating results and no particular order is required. Insert and Delete are slow operations as everything after the position needs to reorder.

**Stack** – Last in first out. Similar to array in which only the last used index can be accessed. Good way to reverse order. Can be used with DFS algorithm. Use .NET implementation

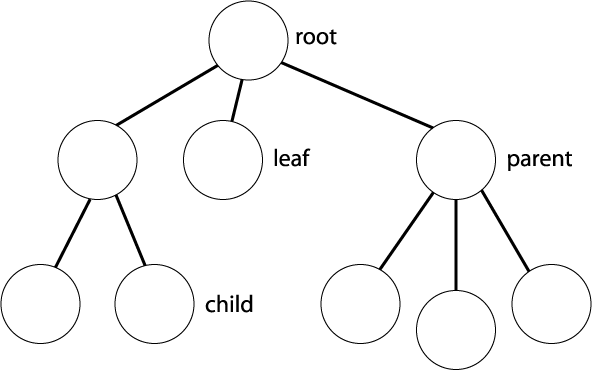
**Queue** – First In First Out. Similar to array in which only the first index can be accessed. Can be used as a part of BFS algorithm. Use .NET implementation.

**Linked List** – Something between a queue and a stack. Can insert in the beginning and end, and remove from the beginning and end. Use .NET implementation.

**LINQ is optimized for .NET Native structures.**

# Trees

**Important terms** - Node, edge, root, child, children, siblings, parent, ancestor, descendant, predecessor, successor, internal node, leaf, depth, height, subtree. Make sure to understand them. Helps when building a tree and discussing algorithms.



**Binary search tree.**

*Definition* – for each node x, each element form the left subtree is <= x and each element from the right subtree is > x.

*Balanced Tree* – max difference in height between left and right subtree is 1.

*Balanced Binary Tree* – add, remove, search all with a log(n) complexity(Approximately).

**Tree Implementations.**

Recursive definition of a tree: Single Node is a tree. Tree nodes can have zero and multiple nodes as children. The advantage of this definition is that recursive algorithm are easily written.

Tree Implementation can be taken from Tree.cs

Balanced Tree Implementation – heap structure is an example. Balance Binary Tree, probably not on the exam. But just in case: <http://demakov.com/snippets/aatree.html>

# Dictionary, Hash Tables, Sets

**Dictionary** – can be views as an array with custom index. Quick search by key. So when searching by a parameter make a dictionary with that parameter as key type. Use .NET Implementation.

**SortedDictionary** - .NET Implementation of balanced Tree. Sorted by key. Use when sorting is needed

**Sets vs Bags – Bags can have repeated values**

**Sets**

**Hash Tables** – Through hashing function each element gets an index in an array. Generally index is calculated by **i = Hash(obj)%n** where n is the size of the array. Add, find, delete are of constant speed. Use .NET implementation (Doesn’t allow for repeat values).

SortedSet – sorted in increasing order of elements

**Additional Note: When collisions are present set performance is reduced.**

**Additional Note: To use sorted structures for custom classes implement Icomparable, to set the way elements are sorted.**

**Additional Note: To use sets provide Equals and possibly GetHashCode, to set the way elements are sorted.**

# Other Structures

**Wintelect Classes that might be useful**

**BigList** – quick insertion, deletion, copy, ranges. Good to use when there are may deletion and isertions.

**Bag/OrderedBag** – sets that allow repeat values. Same rules apply as with sets.

The rest are similar to .NET implementation.

**Heap/PriorityQueue**

Same thing. The representation of heap is a balanced tree with the smallest/biggest element at the top. It is not sorted. When element is removed or added heap gets rearranged.

Implementation with explanations can be seen in Heap.cs

# Recursion

**When is recursion appropriate:**

Recursion is commonly used when exhaustion of all possibilities are required. Some examples for that are combinatorial problems, paths in labyrinth and other similar problems.

**When recursion is not appropriate:**

When there is obvious iterative solution. Like number series (fibonachi, factoriel). If there is only one recursive call in your function, then there is definitely an obvious iterative solution. Example for that can be seen in tree DFS travers both iterative and with a stack.

**In combinatorial problems**. If specific combinations are required most likely recursion will be the right type of solution. If only a number of combinations or some results of such are required dynamic algorithm is the most likely solution. All specific algorithms for combination, variations and so on will be included in the project and discussed in the combinatory section.

**Paths in a maze** – Three common types of problems are find shortest path to a point, find longest path to a point, and finally find all paths that satisfy some condition.

**Shortest Path to a point** – BFS is the most likely candidate for a solution. Implementation with a queue will probably be your best bet. In this case you can look at the labyrinth as a tree and you are traversing it to reach a certain point(searching). In a maze this type of solution would look like a wave staring from the initial point and radiating toward the goal.

**Shortest Path to all points** – Dijkstra is the most likely candidate for a solution. In this case the maze needs to be viewed as a graph. It will be covered in the graph section.

**Longest Path to a point** – DFS or BFS, but in my opinion DFS is much simpler to implement. This type of solution is also known as backtracking. The general idea is you take a path to its end, and then go back a step and go in a different direction until again the path ends. The method ends when there is nowhere to go and nowhere to backtrack to.

**All paths that satisfy a conditio**n – BFS as the paths radiate from starting point and stop when then satisfy a condition.

There are few examples recursion in Recursion project.

# Combinatorics

Combination – order doesn’t matter

Variation – order matters and you take k elements.

Permutation – order matters and you take n element.

All algorithms can be found in the demo code from the lectures.