ADT (Abstract Data Type)

Just concerned with the laying of rules. Not concerned with how it is implemented.

Static vs Dynamic Array

1. Insertion/appending not allowed in static arrays

**Priority Queues (ADT):**

**Uses** – Dijikstra, Huffman encoding, BFS algo (A\*), MST algo

1. Similar to queues. However, the data is ordered basis some priority.
2. Elements in a priority queue should be comparable
3. Priority queue uses heap DS
4. Priority Queues are not heaps. Heap is just a DS which gives best time complexities to perform operations of PQ.
5. Heap – value of parent will always be greater or smaller than children.
6. All heaps must be trees should not contain any cycles.
7. Complexity
   1. Construction – O(n)
   2. Polling – O(logn) reshuffle (removing highest priority element)
   3. Peeking – O(1)
   4. Adding – O(logn) #reshuffle
   5. Removing – O(n)
   6. Removing using hash table – O(logn)
   7. Contains – O(n)
   8. Contains using hash table – O(logn)
8. Turning min PQ into max PQ – negate the array and you will get the other. Same trick can be used for strings.
9. PQ with Binary Heap
   1. A complete binary tree is a tree which at every level, except possibly the last is filled.
   2. Use array to represent Binary Heap [we can also use objects and pointers]
      1. Let i be parent
      2. Left child will be at index (2i + 1)
      3. Right child will be at index (2i + 2)
10. Inserting element – insert element at the end and sift up till heap invariant is satisfied.
11. Removing element – swap the element to be removed with the last element and remove the last element and sift down/sift up after that.
12. Removing node from heap with O(logn) time
    1. Map every node to index and that way we don’t have to scan the heap for element.

**Union Find (Disjoint Datasets)**

Use cases – Kruskal MST, Grid percolation, N/w connectivity, Image Processing, Least common ancestor in trees

Union Find application Kruskal’s MST:

1. Sort edges by ascending order of weight
2. Iterate through the edges and look at the nodes of the edge. If edges are already unified we don’t include this else we will unify the nodes.
3. Algo terminates when all edges are processed or all nodes have been unified.

Path compression union find

**BT and BST**

**Use cases – sets and map implementation. Binary heaps,**

1. BT is a tree in which every node has at most 2 children.
2. BST – root > left and root < right
3. Self-balancing trees are the trees which balance itself after each insertion/deletion operation.
4. Removing an element
   1. Search for the element
   2. Replace the node with its successor to maintain BST invariant
   3. 4 cases , leaf node, only left subtree, only right subtree, or both subtrees
   4. For both – the successor can be the largest value in left subtree or smallest value in right subtree
5. Traversal – inorder, preorder, postorder, level order(bfs using queue)

**Finwick Tree (Binary Indexed Tree)**

**Uses – range query, point updates**

Prefix Sum

1. For array 5, -3, 6. Its prefix sum is 0,5,2,8
2. Can easily find sum between indices. For example sum between index 1 and 2 is P[2+1] – P[1]
3. Caused problem when we have to update an entry in array, then we will have to update all the entries of prefix sum.

**Finwick tree** time complexity

1. Construction O(n)
2. Point Update O(logn)
3. Range Sum O(logn)
4. Range Update O(logn)
5. In Finwick tree entries are in binary. Binary entries bits are indexed with 1 and not with 0.
6. Each binary entry’s LSB is responsible for 2^(LSB index -1) entries below it including itself.
7. For instance if LSB is at index 1 then 2^(1-1) is 1 that number is just responsible for itself.
8. LSB represents range of responsibilities.
9. Example from video.
   1. Prefix sum for index 7. 7th element(since it is 1 based index)
   2. 7 binary is 00111.
   3. 7’s LSB is at position 1. So, 2^1-1 = 1 it is only responsible for itself.
   4. We will go to 1 level down i.e 6. 6’s LSB is at index 2. Therefore, by formula 2^2-1=2. 6 is responsible for 2 level including itself. From 6 we will go to 4 and similarly following the above rules we come to know that 4 is responsible for 4 levels below it.
   5. Therefore Sum at index 7 = A[7] + A[6] + A[4]
   6. Example 2 - Interval sum between [11,15]
   7. For interval sum, get sum of [1,15] and get sum of [1,11). 11 exclusive i.e [1,10]
   8. Subtract [sum of [1,15] – sum of [1,10]]
10. Point updates basically means if we update a value at an index we will have to cascade that value up to all the prefix sum entries that are affected by it.

**Suffix Array – string processing**

**Uses – Find unique substrings**

Suffix is a substring at the end of a string of characters.

Suffix for HORSE is E, SE, RSE, ORSE, HORSE

Suffix Array is an array which contains all suffixes of a tree in sorted order.

A suffix array provides a space efficient alternative to suffix tree which itself is a compressed version of a trie.

1. A suffix array stores index of a suffix arrays
2. LCP – Longest Common Prefix array – stores index of how many characters each suffix has common with each other.
3. Construct LCP value out of Suffix Array.
4. LCP array is an array in which every index tracks how many character does two sorted adjacent suffix have in common.
5. No. of unique substrings in a suffix = n(n+1)/2 – sum of all values of LCP array
6. Longest common substring problem – given n strings find the longest common substring among at least k of them.
7. Dynamic approach will have a run time of O(n1\*n2\*n3…)
8. This problem can be solved in O(n1+n2+n3…) runtime using suffix arrays.
9. Add sentinels which have ASCII values less than characters between strings “$,#,%”
10. Minimum value from set of suffixes from all groups will give us longest common substring. Check this point.
11. Longest Repeated String (LRS)
    1. Find the maximum LCP value.
    2. That will the longest repeated sequence.

**Balance Binary Trees**is achieved by rotating nodes to have a balanced tree.

**AVL Trees**

AVL trees are one of many types of Balanced Binary Search Trees in which insertion, deletion and search operations are performed in O(logn) time.

1. Balanced Factor (node) = height of right subtree – height of left subtree
2. Balance factor should be 0, -1, 1
3. 4 cases for rotation if balance factor is not 0, 1, -1
   1. Left heavy – Right rotation
   2. Right Heavy – Left Rotation
   3. LR – Left rotation, Right Rotation
   4. RL – Right Rotation, Left Rotation
4. Removal
   1. Leaf Node – Remove without any side effect
   2. Case 2 & 3 – only Left/Right subtree, successor of the node that we are trying to remove becomes the child of the parent
   3. Case 4 – both subtree – find largest element from the left subtree or smallest from the right subtree, swap the node to be removed with the found element, remove the node.

**Indexed Priority Queue**

PQ variant which supports faster updates and deletes of key value pair