1 Dictionary

Definition 1. A dictionary/associative array is a abstract data type with (key, value) pair with operations

- Addition a pair
- Removal of a pair
- Modification of a existing pair
- Lookup a value of a existing pair

There are different implementation of this abstract data type such as

- Tree including B-tree, AVL-tree, Red-Black-Tree
- Hash Function
- Direct Adressing

1.1 Direct Addressing

If $\forall k : 0 \leq k \leq M$, where $M \in \mathbb{N}$ then we can use an array $A[k] \leftarrow v$ to store the pair. Each operation is $\Theta(1)$, and the total storage is $\Theta(M)$.
$Key = \{a_n\}_{n=0}^{M} \xrightarrow{A} Value)$

1.2 Hashing

For $k \in Any$, then define $h : Any \to \{0, ..., M\}$ to be a hash function. We can store the key in an Array.
$Key = Any \xrightarrow{h} \{a_n\}_{n=0}^{M} \xrightarrow{A} (Key, Value)$
The hash function is generally not injective. We get a collision when we want to insert a (key, value) pair into a bucket that is occupied.

Definition 2. Load Factor: $\alpha = \frac{n}{M}$, where n is the number pairs already stored, and M is the array size

1.2.1 Chaining

$Key = Any \xrightarrow{h} \{a_n\}_{n=0}^{M} \xrightarrow{A} List(Key, Value))$

1.2.2 Linear Probing

Hash Construction: $H \to H_L$
Define: $h_D(k, i) = h_1(k) + i \bmod M$
$Key = Any \times \mathbb{N} \xrightarrow{h_L} \{a_n\}_{n=0}^{M} \xrightarrow{A} (Key, Value)$
It will through iterate $0, 1, ...$ until $(Key, Value)$ is found.

Example 1. $h_L(k, i) = (h(k) + i)$

1.2.3 Double Hashing

Hash Construction: $H_1 \times H_2 \rightarrow H_D$
Define: $h_D(k,i) = h_1(k) + ih_2(x) \bmod M$
$Key = Any \times \mathbb{N} \xrightarrow{h_D} \{a_n\}_{n=0}^{M} \xrightarrow{A} (Key, Value)$

1.2.4 Cuckoo Hashing

We can look at Cuckoo Hashing as a bipartite graph, $G = (U, V, E)$, where U and V are array of size M. The occupied buckets in U are $h_1(k)$, and V, $h_2(k)$ for some k. The edges are $(h_1(k), h_2(k))$ for some k. Remember each slot can only have one element.

```
function CuckooHashing(A, key, value)
    pos = $h_1(k)$:
    while A[pos] is not None do
        A[pos], key = key, A[pos]
        if $h_1(k)$ != pos then
            pos = $h_1(k)$
        else
            pos = $h_2(k)$
        end if
    end while
    A[pos] = (key, value)
end function
```

Theorem 1. *The insertion of a key succeeds if the connected component containing key contains either no cycles or only one cycle. (pseudotree)*

2 Range Searching

2.1 QuadTree

QuadTree:: Leaf Point | Branch QuadTree QuadTree QuadTree QuadTree

2.2 BST Range Search

Given $[a, b]$, and a BST, the problem is to find $\{v \in Nodes(T) \mid a <= v <= b\}$, then for a subtree, three cases may happen by the law of triochotomy:

- If $b < root(T)$,

$$\forall v \in Nodes(T_{right}) : v \geq root(T)$$
$$\nexists v \in Nodes(T_{right}) : v < root(T)$$
$$\nexists v \in Nodes(T_{right}) : v < b$$

  Therefore, no search to the left.

- If $root(T) < a$

$$\begin{gathered}\forall v \in Nodes(T_{left}) : v \leq root(T) \\ \nexists v \in Nodes(T_{right}) : v > root(T) \\ \nexists v \in Nodes(T_{right}) : v > a\end{gathered}$$

  No search to left.

- If $a \leq root(T) \leq b$, search left and right.

There are tree type of nodes

- Boundary nodes: No constraints, or contrainst not dominating $a \leq v \leq b$
- Inside nodes: There exists constraints dominating $[a, b]$, the topmost is also called allocation node
- Outside nodes: Nodes that are being pruned

2.3 Range Tree

To perform a range query where $A = [a_{11}, a_{12}] \times ... \times [a_{n1}, a_{n2}]$,

- Boundary node: since it's not dominating, we check node if in range
- Allocation node: Dominating, construct a BST with all the points of the subtree based on the next coordinates. Then do BST.

T construct a binary based on $Node(T)$

2.4 Conclusion

Range Tree:

- Fastest: $O(s + n(\log n)^d)$
- Space: $O(n(\log n)^{d-1})$
- Complicated insert/deletion

kd-Tree:

- $O(s + n)$
- Space: $O(n)$
- Duplicate case to be taken care of

3 String Matching

3.1 Knuth-Morris-Pratt algorithm

Using DFA, go to the state where it has failed. There is the failure array and the DFA.
State k represents that the last k elements are matching ($P[:k]$), we want to parse the next element $k+1$-th element ($P[k]$). If success, move on next state. If failure, jump to the state where most the suffix matches the current good suffix, and still try to parse the next character. (no parsing this stage)
The main mapping is a prefix of P, of prefixes.

3.2 Boyer-Moore Algorithm

There are two main components to this algorithms:

- Bad character rule: Shift until matches the character
- Good suffix rule: Shift where the suffix matc

You have P, T, k, then for $P[k:]$ suffix, compare to T, we want to get the maximum number of matching suffix of this suffix, and bad character index. $P[k:][l:]$, and $P[l]$ where l indicate the bad character index. Take the difference of $n-l$, it's the amount you have to shift. (proof later)
$k = k+1$
Good suffix same idea as KMP, then find the main mapping is suffix of suffix. The reverse of KMP.

4 Huffman Tree

Decoding: Greedy
Encoding: Merge frequencies

5 Appendix

5.1 Median

Median of a list of length n and index starting at 1 is defined in statistics:

- If n is even, Median $= \frac{a_{\frac{n}{2}} + a_{\frac{n+1}{2}}}{2}$
- If n is odd, Median $= a_{\frac{n+1}{2}}$

Note that a_k means that

- There are $k-1$ elements before k-th index excluding itself. We can see k as fence, and we can assign every left pole of a fence with the fence value. We want to know the number of fences between two poles, so we do substract.

- There are $(n+1)-(k+1)=n-k$ after k-th index excuding itself. Same logic, we take the right pole of fence n and k, and do substraction.

The intuitive sense of the median is to have same amount of elements before and after k-th excluding k itself. Thus, we want to find k such that

$$\begin{aligned} k-1 &= n-k \\ k+k &= n+1 \\ 2k &= n+1 \end{aligned}$$

If n is odd, then we want the element at index,

$$k = \frac{n+1}{2}$$

If n is even, then there is no good way of splitting, then we want to find two adjacent elements where the number of elements before first elements is same as the elements after second elements. Let k be the index of first element.

$$\begin{aligned} k-1 &= (n+1)-((k+1)+1) \\ k-1 &= n-k-1 \\ k &= \frac{n}{2} \end{aligned}$$

If $k=\frac{n}{2}$, then $k-1=n-k-1$ holds, therefore we have k-th and $k+1$-th element, and we take the average of the to give a single number.

5.2 Binary Tree

One way to define a binary tree is as follows
We can define operation as $root(T), subtrees(T), nodes(T)$

5.2.1 Binary Search Tree

BinarySearchTree:: Empty | Value BinarySearchTree BinarySearchTree
A Binary search Tree is a Binary Tree with this addition property:

- *Empty* is a Binary Search Tree
- T is a Binary Search Tree if T_{left} and T_{right} are Binary Search Trees, and

$$\forall v \in Nodes(T_{left}) : v \leq root(T)$$
$$\forall v \in Nodes(T_{right}) : v \geq root(T)$$

5.3 Interval

Recall the law of trichotomy: Let $<$ be an order relation in X, then

$$\forall x, y \in X((x<y) \vee (y>x) \vee (x=y))$$

Given $[a,b]$ and v where $a \leq b$, then

$$(v<a) \vee (a \leq v \leq b) \vee (b>v)$$

5.4 Graph Theory

A graph can be described, and constructed based on

- The edges
- The nodes, and their adjacent nodes

As geometric shape can also be seen as such.

5.5 List

Here are some useful constructions of lists/strings of length n. The k-th index can be seen as a fence or a pole

- $\{P[k] : k \in \{a_n\}_{i=0}^{n-1}\}$: list of characters
- $\{P[k:] : k \in \{a_n\}_{i=0}^{n-1}\}$: list of suffixes
- $\{P[:k] : k \in \{a_n\}_{i=1}^{n}\}$: list of prefixes
- Map
- Foldr

Notice slicing is defined as left pole. Here are different interpretation of the function

- $P[:k]$: there are k element in the prefix, the prefix before $k+1$-th element
- $P[k]$: there are k element before this $k+1$-th element
- $P[k:]$: there are k element before this suffix

5.5.1 Shifting

Problem: We have fence i, fence j, we want to shift fence i until i reaches fence j. How many fence do we have to shift?
$i+n$ adds n poles between i-th pole and $i+n$-th pole. In other words,

$$\exists n : i + n = j$$
$$n = j - i$$

Affine map/Group action