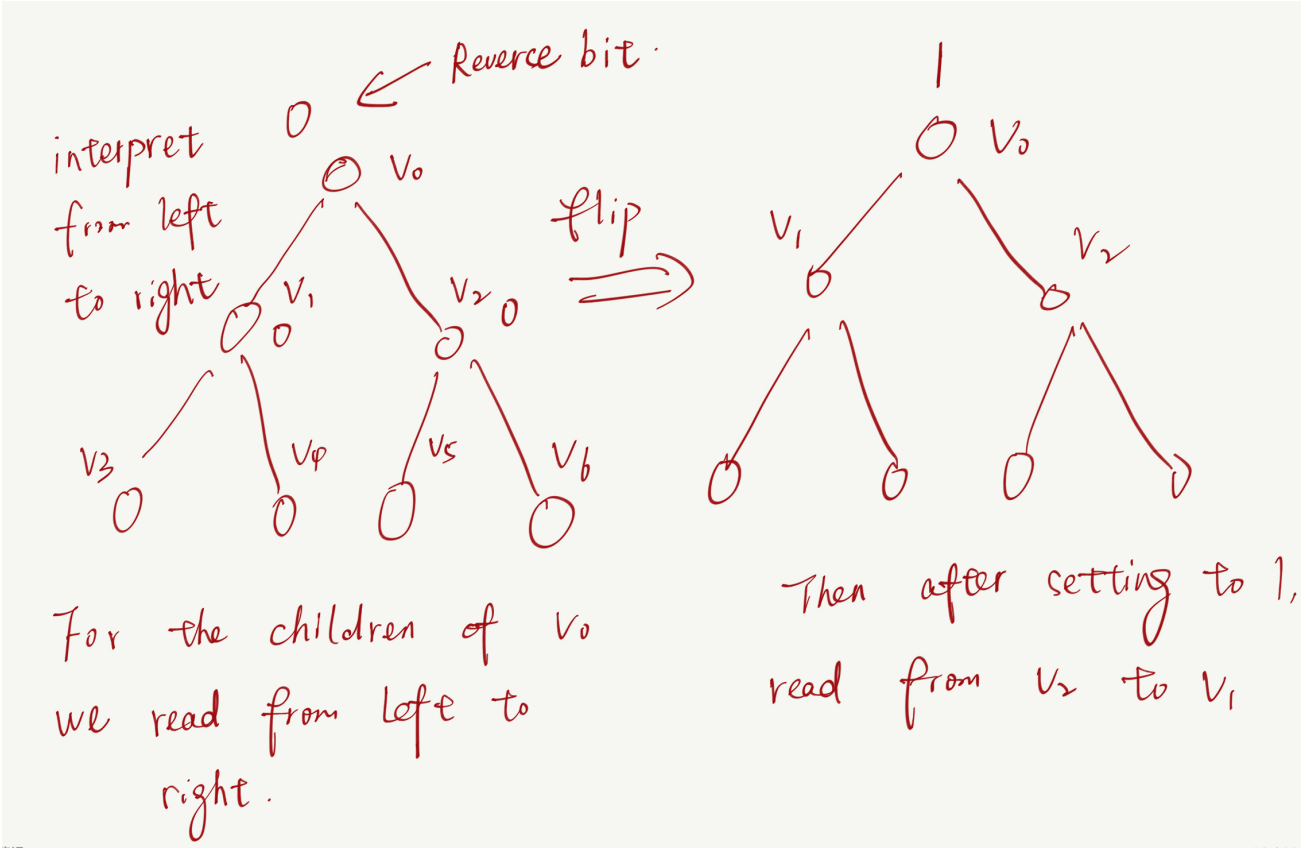
HW6

1. *Suppose we are maintaining a collection of DNA strings. We wish to support one more operation: Reverse a string: i.e. u = u1u2 . . . un is replaced by un, . . . u2u1.*
   Show how to support this operation and the previous operations so they run in time O(log n), where n is the sum of the lengths of the strings involved in an operation.
   Present your algorithm so it is clear why it works and justify the runtime briefly. It suffices to explain how to modify the solution to the additional problem. You may state known results about data structures.

   > $Reverse(\sigma, i, j)$. σ is the string being reversed.
   >
   > Reverse bit: If this bit is set to 1, the direction of interpreting each edge in the subtree of this node is reversed. If set back to 0, then back to normal. The descending notes are inherited from their parents by XOR. For example:
   >
   > 
   >
   >
   > We split the 2-3 tree storing σ into three pieces, σ_l,σ_{mid},σ_r, containing the first $i-1$ characters, $j-i+1$ characters and the remaining characters. Cut the mid tree. This will cost $O(\log n)$ times.
   >
   > Then flip the reverse bit of the middle tree and join all three trees back together. Each of them cost $O(\log n)$ time. Hence, the total time is $O(\log n)$

2. Let S be a set of n items which have $k < n$ distinct values. Show how to sort S in expected $O(n + k \log k)$ time.

   Ans:

1. Create two arrays A and B in length k . For each item, hash it and get the index i. If the items do not exist in $A[i]$, store it and $A[i]$ and assign 1 to the $B[i]$. If the items exist in $A[i]$, plus one to the value in $B[i]$. This will iterate n times and cn operations will be performed.
2. Sort the array A by the sort algorithm and modify the position of the items in B as well, which will cost $O(2k \log k)$, i.e., $O(k \log k)$.
3. Copy the items in A to set S from index $i = 0$ to k , if $B[i] == t(t \leq n - k)$, copy k times to set S, which will cost $O(n)$.

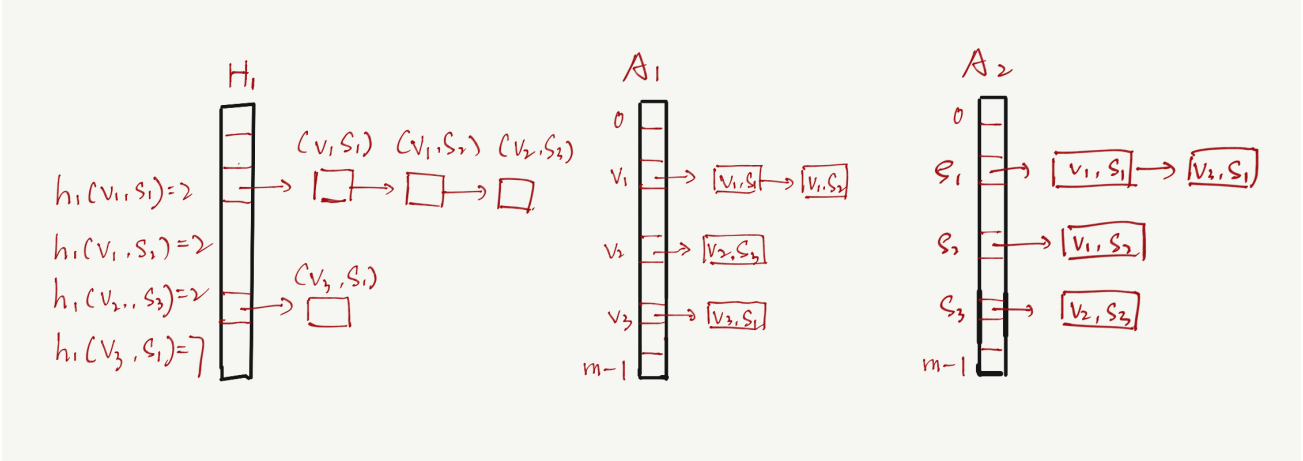
Hence, the total cost will be $O(n + k \log k)$.

3. *Suppose you are given a set S of items with two attributes, value and size, which are both positive integer values in the range (0, m − 1). Give a data structure to store these items so that the following operations can be supported in the stated times.*
i. Insertion, in expected O(1) time.

ii. Deletion, in expected O(1) time.
iii. Deletion of all items of size s in expected time proportional to the number of these items. iv. Report all items of value v in expected time proportional to the number of these items.

Present your solution so it is clear why it works and justify the runtime briefly.

We can use the idea of hashing with chaining to hash on two attributes on an array with buckets, i.e., for each item (v, s), it is stored in the buckets in the array whose index of the array is $h(v, h)$. And initialize two arrays with length m to store the items filtered by value and size in buckets, shown in the figure. Every time insert or delet the item, we hash the item (v, s) to insert or delete in the hash table, and do the same to the arrays. Each will cost expected $O(1)$ time. Therefore, the total cost of insert and deletion is in expected $O(1)$ time.



For example, insert the item (v_1, s_1), delete is the same:

1. Calculate $h(v_1, s_1)$ to find the index of the bucket in H_1, then insert it into the buckets.
2. Add the item to the buckets of $A_1[v_1]$
3. Add the item to the buckets of $A_2[s_1]$

Deletion of all items of size s:

1. Iterate all the items (assume t numbers) in $A_2[s]$, delete them in H_1 by calculating $h(v, s)$ and $A_1[v]$, the total cost is $O(ct) + O(ct) = O(t)$

2. Delete all the items in A_2, the total cost is $O(t)$

The total cost will be $O(t)$

Report all items of value v:

Report all the items (assume t numbers) in $A_1[v]$, which will cost $O(t)$.

4. *Show how to build such a two-level hash table so that there are no collisions at the second level and so that it uses O(n) space in the worst case. Show that it can be built in expected O(n) time*

We first compute $i = h_1(x), h_1 \in H_{2n}$ to find the position in the first table, then calculate $j = h_2(x), h_2 \in H_{2m(m-1)}$ to insert the items. Each operation will cost expected $O(1)$ time as a hash table. After performing n times, the total built time will be expected $O(n)$.

As the size of the hash table in the index i is $2m_i(m_i - 1)$, the total space used in the second table is $2\sum_i[m_i(m-1)]$

$$
\begin{aligned}
E\{\sum_i[2m_i(m-1)]\} &= E[\sum_i(\sum_x\sum_y C_{xy})] \quad (C_{xy} = 1 \text{ if x and y collide, otherwise } 0) \\
&= E[\sum_i(\sum_x\sum_{y=x} C_{xy} + \sum_x\sum_{y\neq x} C_{xy})] \\
&\leq n + \binom{n}{2} \cdot 2/2n \\
&\leq 2n = O(n)
\end{aligned} \tag{1}
$$

Hence, the second level uses $O(n)$ space in the worst case.