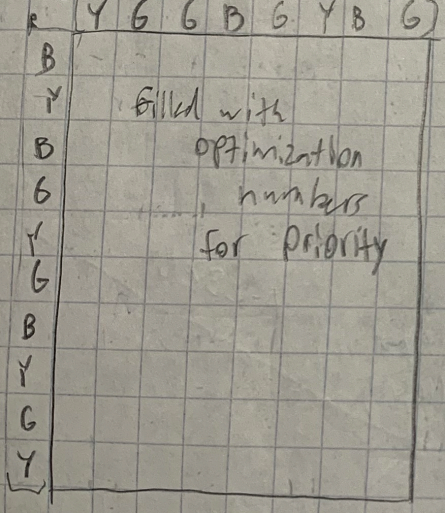
2. Dynamic programming

a) Basically, the solution for $T(i) = \Sigma T(i+1)$. The recusive relationship would rely on the current state/solution of the iteration before, like we did with Gene alignment. We can show the structure in part B, but basically in each recursive step we rely upon the current score for the solution before us. We make values for matches, indels, and substitutions, and use those to find the best possible solution before recursing down again

B) This structure would be a 2d array of integers values. Its size would be L x R with L and R being the number of books in each stack. It might look like this for the example:



Each place would contain a value to show optimization. where we get an Indel, we can show where a book should be removed

c) Create 2d array (size L×R) arr with ∞ current value
for match = +1
for substring = 0

```
for i in len(L): # shortest on top
    for j in len(R): # longest on side
        at arr[i][j], look at top, left, and top-left diagonal values
        check that L[i] and R[j] are equal colors
            if so, arr[i][i] = max(diag, top, left) + 1
            # only increment max height if book colors are =
        else if max(diag, top, left) = diag  # remove book from both stacks
        else if max(diag, top, left) = left  # remove book from L
        else if max(diag, top, left) = top   # remove book from R
return array[L][R] # bottom right corner for max height
```

max height = 5

d) both time and space are O(mn) with m being length of L and n being length of R. Space complexity comes with the two d array, initialized to be mxn space, changing values within does not alter the space as changes are O(1). Time is also O(mn) because we iterate through each element in arr, each iteration being O(1), so double for loops used give us O(mn) time.

e) You could create two boolean arrays (Larray and Rarray) of size n (length of L) and m (length of R). As we run the program, we maintain pointers to which positions in arr we have come from, which produces an optimal path when we start from the lower right corner (max height). We then back track, keeping track of the max height at each position i, j we visit. If the height doesn't change, we mark the position at Larray[i]=false and Rarray[j]=false. (and we came from a diagonal) If its height doesn't change and we came from below, Rarray[j] is marked false. If we come from right index, Larray[i]=false. This leaves us with the arrays where false means removing that book at that position from the pile.