Shortcuts

Friday, 2 December 2022 7:55 AM

- Set<Character> vowels = new HashSet<>(Arrays.asList('a', 'e', 'i', 'o', 'u'));
- Map<Integer,Integer>myHashMap=Map.of(1,1,2,5); (From java 9+) -> {{1,1},{2,5}}
- List<String> list = new ArrayList<>(Arrays.asList("soha"));
- Int MOD = 1e9+7;
- Reg Expression : to find a string has no special char -> str.matches("[a-zA-Z0-9]*")
- Find whether a number is power of a number without looping.. Find maximum pow and % the given number
  - E.g (n>0 and `pow(3, 19) % n == 0) then those numbers are divisible by 3`
- Traverse all direction:

  -
    ```
        public static final int[][] DIRECTIONS = {
            { 0, 1 }, { 0, -1 }, { 1, 0 }, { -1, 0 }
        };
    ```
- `If there a value 'K' then think of sliding window can be applied`
- `Finding valid position inside matrix:`
  - `if (newRow >= 0 && newCol >= 0 && newRow < n && newCol < m) {}`
- `Find Sequence of number array ==> can sort, find subarray ==> continous so can't sort`

DEAD SITUATIONS:

- Array -> Think whether Binary search can be applied to it
- Array's -> forming like mountain in the solution then think about using Stack and bringing the solution
  - https://leetcode.com/problems/minimum-operations-to-convert-all-elements-to-zero/submissions/1826134861/?envType=daily-question&envId=2025-11-10
- Finding GCD for 2 numbers:

```
        private int gcd(int a, int b) {
            if (b == 0) return a;
            return gcd(b, a % b);
        }
```

Algorithms:

- To find max sub array sum:
  - Kadane's -

    As you move from left to right, at each index i:
    - Either **extend** the previous subarray

      currSum + nums[i]
    - Or **start a new subarray** at i

      nums[i]
    - curr = best subarray sum ending at current index = max(x, curr + x)
    - best = best subarray sum seen overall = max(best, curr)

TIPS

- When ever doing **% of a particular value p make sure to do, `+ p) % p`**
- `In long range of input array to find a particular subarray, the best way is to use`
  - `Sliding window`
  - `Even better way keep track of prefix sum and trying utlizing map to store and retrive data`

```
                    prefixSum += nums[i];
                    int currentMod = (int)(prefixSum % p);
                    int targetMod = (currentMod - rem + p) % p;
                    if (prefixMod.containsKey(targetMod)) {
                        minLength = Math.min(minLength, i - prefixMod.get(targetMod));
                    }
                    prefixMod.put(currentMod, i);
```

- `Anything to find value with sorted array, think binary search`
- `If array has to be iterated only once the possible O(n) approaches are priority queue, stack, queue, sliding window`
- `For time interval groups like meeting from start to end, how much do overlap.. For solving in O(N) we can keep an array from 0 to max.. For every start do +1 and next element of end do -1.. So when we sum we can find overlaps`

- k ^= 1 -> If k is 0, it becomes 1, and if it's 1, it becomes 0

- Adding to list directly without creating individual:

  list.add(newArrayList<>(Arrays.asList(i, n-i)));

- For adding each element into List without creating object..

```
 for (int i = 1; i <= n; i++) {
      resultList.add(Arrays.asList(i));
 }
```

- Tree set to find the next and previous element e.g., set contains{5, 10, 20, 30};
  - Next element can be found by using set.ceiling(15) -> returns 20
  - Previous element can be found by using set.floor(15) -> returns 10;
- Bit manipulation
  - Iterate through each bit -> a>>>=1
  - Check whether bit is 0 -> `(a&1)==0`
- `Adding custom sort using Arrays.sort ->`

```
Arrays.sort(pairs,new Comparator<int[]>(){
@Override
Public int compare(int[] a, int[] b){
Return a[0]-b[0];
}
});
```

- **Step 1: Creating a Difference Array**

  28/07/23

  The function initializes diff as an array of zeros, with size n+1 (one extra element to handle end-of-range marking).

  python

  CopyEdit

  diff = [0] * (n + 1)

  Given queries[i] = [left, right, val], we update the difference array:

  python

python
CopyEdit

```
diff[left] += val       # Increase at left index
diff[right + 1] -= val  # Decrease after right index
```

This efficiently marks range updates.

- **Step 2: Applying Updates via Prefix Sum**
  After applying all queries up to k, the function computes the actual values using a prefix sum:

  python
  CopyEdit

  ```
  curr_val = 0
  for i in range(n):
      curr_val += diff[i]  # Accumulate the difference array
      if curr_val < nums[i]:  # Check if we can make nums[i] zero
          return False
  ```

Mac book
Option -> on a folder gives more dropdown options
Java installation path -> /Library/Java/JavaVirtualMachines/zulu-11.jdk/Contents/Home/bin
Mac64 vs Mac_arm_64 -> Intel vs apple silicon -> how to find.. Run the following in terminal -> "uname -m"

Java is located in :: export JAVA_HOME=/Library/Java/JavaVirtualMachines/zulu-11.jdk/Contents/Home/bin

Maximum groups with distinct numbers from given array i/p -> {5, 1, 2} -> {{5}, {5, 2}, {5, 2, 1}} o/p -> 3

Binary search: always better to use ::

```
while(low<=high){
   long mid = low+(high-low+1)/2;


 if(shops>n){
       low = mid+1;
  } else{
       high = mid-1;
  }
```

Question	Link	Input	Output	Code	Tips

Maximum Number of Groups With Increasing Length	https://leetcode.com/problems/maximum-number-of-groups-with-increasing-length/	usageLimits = [1,2,5]	3	public int maxIncreasingGroups(List<Integer> usageLimits) {
				Collections.sort(usageLimits);
				int count=0;
				long total=0;
				int req=1;
				for(int i=0;i<usageLimits.size();i++){
				total+= usageLimits.get(i);
				if(total>=req){
				total-=req;
				req++;
				count++;
				}
				}
				return count;
				}	

Largest Element in an Array after Merge Operations	https://leetcode.com/problems/largest-element-in-an-array-after-merge-operations/	nums = [2,3,7,9,3]	21	public long maxArrayValue(int[] nums) {
				int n = nums.length;
				long ans = 0;
				
				for(int i = n-1; i >= 0; i--) {
				if((long)nums[i] > ans) ans = (long)nums[i];
				else ans += (long)nums[i];
				}
				
				return ans;
				}	

How Math.pow works	https://leetcode.com/problems/powx-n/	2, 10	1024.0000	public double myPow(double x, int n) {
				if (n == 0)
				return 1.0;
				if (n == Integer.MIN_VALUE) {
				x = x * x;
				n = n / 2;
				}
				if (n < 0) {
				x = 1 / x;
				n = -n;
				}
				double result = 1.0;
				while (n > 0)
				{
				if (n % 2 == 1)
				result *= x;
				x *= x;
				n /= 2;
				}
				return result;
				}	

Mountain Array	https://leetcode.com/problems/peak-index-in-a-mountain-array/	1, 2, 3, 4, 5, 4, 3, 2, 1	4	public int peakIndexInMountainArray(int[] arr) {
				int idx=0;
				int low=0, high=arr.length, mid=low;
				while(low<high){
				mid= high/2+low/2;
				if(arr[mid]<arr[mid+1])
				low=mid+1;
				else
				high=mid;
				}
				return low;
					O(og n)

				}	

2 player playing optimally on each side of array	https://leetcode.com/problems/predict-the-winner/discuss/1686992/Recursion-To-Top-down-Dp			class Solution {
				  public boolean PredictTheWinner(int[] nums) {
				    int[][] mem = new int[nums.length][nums.length];
				    for (int i = 0; i < mem.length; i++) {
				      for (int j = 0; j < mem.length; j++) {
				        mem[i][j] = -1;
				      }
				    }
				    int ans = sol(nums, nums.length, 0, nums.length - 1,mem, true);
				    int sum = 0;
				    for (int i = 0; i < nums.length; i++) {
				     sum += nums[i];
				    }
				    sum -= ans;
				    return ans>=sum;
				  }
				
				  private static int sol(int[] arr, int n, int i, int j, int[][] mem, Boolean player) {
				    if (i > j)
				     return 0;
				    // if mem[i][j] has already been computed we do not
				    // do further recursive calls and hence reduce
				    // the number of repeated work
				    if(mem[i][j] != -1){
				     return mem[i][j];
				    }
				    if (player) {
				     mem[i][j] = Math.max(arr[i] + sol(arr, n, i + 1, j, mem,!player), arr[j] + sol(arr, n, i, j - 1,mem, !player));
				    } else {
				     mem[i][j] = Math.min(sol(arr, n, i + 1, j, mem,!player), sol(arr, n, i, j - 1,mem, !player));
				    }
				    return mem[i][j];
				 }
				}
					Need some more revision

My sorting using heap memory	Using Map and storing their frequencies			public int[] sortArray(int[] nums) {
				    HashMap<Integer, Integer> map = new HashMap<>();
				    int min = Integer.MAX_VALUE, max = Integer.MIN_VALUE;
				    for(int i=0; i<nums.length; i++){
				        map.put(nums[i], map.getOrDefault(nums[i], 0)+1);
				        min = Math.min(min, nums[i]);
				        max = Math.max(max, nums[i]);
				    }
				    int i=0;
				    while(min<=max){
				        if(map.containsKey(min)){
				            for(int j=0; j<map.get(min); j++){
				                nums[i++] = min;
				            }
				        }
				        min++;
				    }
				    return nums;
				}	

Given array find subarray such that total is divisible by p	Use prefix sum and hashmap to achieve it in O(n)	https://leetcode.com/problems/make-sum-divisible-by-p/?envType=daily-question&envId=2024-10-03		for (int i = 0; i < nums.length; ++i) {
				            prefixSum += nums[i];
				            int currentMod = (int) (prefixSum % p);
				            int targetMod = (currentMod - rem + p) % p;
				            if (prefixMod.containsKey(targetMod)) {
				                minLength = Math.min(minLength, i - prefixMod.get(targetMod));
				            }
				            prefixMod.put(currentMod, i);
				        }
					For all sub array problems use sliding window or better use this approach of prefix sum and hashmap

Dijkstra's Algorithm	For weight and non-weight graph with positive values only	https://leetcode.com/problems/minimum-time-to-visit-a-cell-in-a-grid/?envType=daily-question&envId=2024-11-29			1. Use Heap PQ and sort in ascending order, store index and weight 2. Maintain a boolean array visited 3. Store the minimum and return the value