cp

michaelyql

2024

1 subset sum

- ◇ subset sum problem: does S have subset that sums exactly to T
- ◇ partition problem: partition S into two subsets such that $sum(S_1) = sum(S_2)$ (special case of subset sum where $T = \frac{1}{2} \times sum(S)$)

1.1 naive $O(2^n)$ backtracking

```python
def isSubsetSum(arr, n, target):
    if target == 0:
        return True
    if n == 0 and target != 0:
        return False

    if arr[n-1] > target:
        return isSubsetSum(arr, n-1, target)

    return isSubsetSum(arr, n-1, target) or isSubsetSum(arr, n-1, target - arr[n-1])
```

1.2 dp $O(n * sum)$ time and space

```python
def isSubsetSum(arr, n, target):

    # dp[i][j] = True if you can sum to j using first i elements
    dp = [[False for _ in range(target + 1)] for _ in range(n + 1)]

    # If target is 0, subset sum is True for any set (empty subset).
    for i in range(n + 1):
        dp[i][0] = True

    # Fill the subset table
    for i in range(1, n + 1):
        for j in range(1, target + 1):
            if arr[i-1] > j:
                dp[i][j] = dp[i-1][j]
            else:
                dp[i][j] = dp[i-1][j] or dp[i-1][j - arr[i-1]]

    return dp[n][target]
```

2 Apartment (CSES)

find the number apartments (each valued a_i) assignable to tenants (desired value t_i). all tenants have a tolerance k.

input:
T: [60, 45, 80, 60]
A: [30, 60, 75]
k: 5

2.1 python

```python
def assignApartments(n: int, m: int, k: int, A: list[int], T: list[int]):
  sorted_apts = sorted(A)
  sorted_tent = sorted(T)
  res = 0
  i = 0
  j = 0
```

```
  while (i < n and j < m):
    if A[i] + k <= T[j] or A[i] - k <= T[j]: # if abs(A[i] - T[j]) <= k
      i += 1
      j += 1
      res += 1
    else:
      if A[i] + k > T[j]:
        j += 1
      else:
        i += 1
  return res
```

2.2 c++

```
#include <bits/stdc++.h>

using namespace std;

const int MAX_N = 2e5;

int n, m, k, a[MAX_N], b[MAX_N], ans;

void solve() {
  cin >> n >> m >> k;
  for (int i = 0; i < n; ++i) cin >> a[i];
  for (int i = 0; i < m; ++i) cin >> b[i];
  sort(a, a + n);
  sort(b, b + m);
  int i = 0, j = 0;
  while (i < n && j < m) {
    // Found a suitable apartment for the applicant
    if (abs(a[i] - b[j]) <= k) {
      ++i;
      ++j;
      ++ans;
    } else {
      // If the desired apartment size of the applicant is too big,
      // move the apartment pointer to the right to find a bigger one
      if (a[i] - b[j] > k) {
        ++j;
      }
      // If the desired apartment size is too small,
      // skip that applicant and move to the next person
      else {
        ++i;
      }
    }
  }
  cout << ans << "\n";
}

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);
  solve();
  return 0;
}
```

3 Ferris Wheel (CSES)

find the number of gondolas required to fit all children (each weighted p_i). each gondola can fit at most 2 children and can hold at most x

input:
W: [7, 2, 3, 9]
x: 10

```python
def ferrisWheel(n: int, x: int, W: list[int]):
  sorted(W)
  res = 0
  i = 0
  j = n - 1
  in_gondola = [False] * n
  while (i < j):
    if (W[i] + W[j] <= x):
      res += 1
      in_gondola[i] = in_gondola[j] = True
      i += 1
      j -= 1
    else:
      j -= 1
  for k in range(0, n):
    if not in_gondola[k]:
      res += 1
  return res
```

4 Maximum XOR Score Subarray Queries

for each query, return the answer from the operation: for the range nums[l, r], replace nums[i] with nums[i] XOR nums[i+1] except the last element, and remove the last element in the subarray, and repeat until only one element remains in that subarray

input:
nums: [2, 8, 4, 32, 16, 1]
queries: [[0, 2], [1, 4], [0, 5]]
constraints: $1 \leq n \leq 2000$, $1 \leq q \leq 10^5$

4.1 optimise XOR score for a single query in less than $O(n^2)$

brute forcing the operation on a subarray takes $O(n^2)$ time, and there are $O(n^2)$ subarrays in the worst case in total (if l = 0 and r = n - 1), so brute force will take $O(n^4 * Q)$.

observe the pattern:
$[x_1 \quad x_2] \to x_1 \oplus x_2$
$[x_1 \quad x_2 \quad x_3] \to [x_1 \oplus x_2 \quad x_2 \oplus x_3] \to x_1 \oplus x_3$
$[x_1 \quad x_2 \quad x_3 \quad x_4] \to [x_1 \oplus x_3 \quad x_2 \oplus x_4] \to x_1 \oplus x_2 \oplus x_3 \oplus x_4$
$[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5] \to [x_1 \oplus x_2 \oplus x_3 \oplus x_4 \quad x_2 \oplus x_3 \oplus x_4 \oplus x_5] \to x_1 \oplus x_5$
$[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6] \to [x_1 \oplus x_5 \quad x_2 \oplus x_6] \to x_1 \oplus x_2 \oplus x_5 \oplus x_6$
... and so on

5 Inversions

5.1 Global and Local Inversions (LC775)

global inversion = nums[i] > nums[j] for $0 \leq i < j < n$.
local inversion = nums[i] > nums[i + 1] for $0 \leq i < n - 1$.
check if no. of global inversions == no. of local inversions

input:
nums = [1, 0, 2]
constraints: $1 \leq n \leq 10^5$

```
# if all GI = LI, then we cannot find an i and j such that i + 2 <= j and A[i] > A[j]
def isIdealPermutation(self, A):
  cmax = 0
  for i in range(len(A) - 2):
    cmax = max(cmax, A[i])
    if cmax > A[i + 2]:
      return False
  return True

# solution 2
  def isIdealPermutation(self, A):
    # if any element is more than 2 places away from its correct position
    return all(abs(i - v) <= 1 for i, v in enumerate(A))
```

5.2 Min Adjacent Swaps to Sort Binary Array

input:
nums = [0, 0, 1, 0, 1, 0, 1, 1]

```
def minSwaps(A: list[int]):
  res = 0
  ones_count = 0
  for i in range(len(A)):
    if A[i] == 0:
      if ones_count > 0:
        res += ones_count
    else:
      ones_count += 1
  return res
```

5.3 Permutation Inversion (CSES)

count the number of permutations of $1..n$ that have exactly k inversions

e.g. n = 4, k = 3, answer = 6

let $dp[i][j]$ represent the number of permutations that have j inversions using the first i elements.
recurrence relation: $dp[i][j] = \sum dp[i-1][j-x]$ for $x = 0, 1, ..., i-1$ depending on where we insert the i^{th} element.
optimise to $O(k)$ space as we only need to keep track of the $(i-1)^{th}$ array when we are at i elements.

```
def count_permutations_with_inversions(n, k):
    # DP table to store the number of permutations of size n with exactly k inversions
    dp = [[0 for _ in range(k + 1)] for _ in range(n + 1)]

    # Base case: 1 permutation of size 0 with 0 inversions
    dp[0][0] = 1

    # Fill the DP table
```

```
    for i in range(1, n + 1):
        for j in range(k + 1):
            # Compute dp[i][j] by summing over dp[i-1][j-x] for x = 0 to min(j, i-1)
            dp[i][j] = sum(dp[i-1][j-x] for x in range(min(j, i-1) + 1))

    # Return the number of permutations of size n with exactly k inversions
    return dp[n][k]

def permutationWithKInversions(n: int, k: int):
  MOD = 1000000007
  dp = [0] * (k + 1)
  dp[0] = 1

  for i in range(1, n + 1):
    new_dp = [0] * (k + 1)

    for j in range(k + 1):
      new_dp[j] = dp[j] % MOD

      if j > 0:
        new_dp[j] = (new_dp[j] + new_dp[j - 1]) % MOD

      if j - i >= 0:
        new_dp[j] = (new_dp[j] - dp[j - i] + MOD) % MOD

    dp = new_dp

  return dp[k]
```

5.4 Min Adjacent Swaps to Make Palindrome

```
def min_swaps_to_make_palindrome(s):
    def can_be_palindrome(s):
        from collections import Counter
        freq = Counter(s)
        odd_count = sum(1 for v in freq.values() if v % 2 != 0)
        return odd_count <= 1

    if not can_be_palindrome(s):
        return -1  # Impossible to rearrange into a palindrome

    s = list(s)
    left, right = 0, len(s) - 1
    swaps = 0

    while left < right:
        # If the characters match, move inward
        if s[left] == s[right]:
            left += 1
            right -= 1
        else:
            # Find the match for s[left] on the right side
            l, r = left, right
            while r > l and s[r] != s[left]:
                r -= 1

            # If we found a match for s[left]
            if r == l:
                # If the left element has no matching pair, swap it forward (for odd-length strings)
                s[l], s[l+1] = s[l+1], s[l]
                swaps += 1
            else:
                # Swap to bring s[r] to the right position
```

```
                for i in range(r, right):
                    s[i], s[i+1] = s[i+1], s[i]
                    swaps += 1
                left += 1
                right -= 1

    return swaps
```

6 Movie Festival (CSES)

given a list of movies with $[\text{start}_i, \text{end}_i]$, find the max number of movies you can attend, assuming you can move instantaneously between venues and can only be at one movie at a time

```
def maxMovies(A: list[list[int]]):
  ans = 0
  currentMovieEnd = -1
  sorted(A, key = lambda x: x[1])
  for [start, end] in A:
    if start >= currentMovieEnd:
      currentMovieEnd = start
      ans += 1
  return ans
```

7 Maximum Subarray Sum

```
def mss(A: list[int]):
  ans = 0
  sum = 0
  for i in range(len(A)):
    if sum + A[i] > 0:
      sum += A[i]
    else:
      sum = 0
    ans = max(ans, sum)
  return ans
```

8 Firefly Queries (CF 2009F)

given array A of length n, create $B = C_1 + C_2 + \cdots + C_n$ where C_i is the i-th cyclic shift of A. for each of the q queries with l, r, return the sum of elements in the range $[l, r]$ inclusive.
$1 \le n \le 2 \cdot 10^5$, $1 \le a_i \le 10^6$

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
int main() {
    int t;
    cin >> t;
    while (t--) { // test cases
        ll n, q; // n and no. of queries
        cin >> n >> q;
        vector<ll> a(n), ps(1); // array A and prefix sum
        for (ll &r : a) {
            cin >> r;
            ps.push_back(ps.back() + r);
        }
        for (ll &r : a) {
            ps.push_back(ps.back() + r);
        }
```

```
        while (q--) {
            ll l, r;
            cin >> l >> r;
            l--; r--; // l and r are 1-indexed
            ll i = l / n, j = r / n; // which subarray does it belong to
            l %= n; r %= n; // find remainder (suffix and prefix)
            cout << ps[n] * (j - i + 1) - (ps[i + l] - ps[i]) - (ps[j + n] - ps[j + r + 1]) << "\n";
        }
    }
}
```

9 Yunli's Subarray Queries (CF 2009G)

for an arbitrary array b, you can perform the following operation any number of times: pick any index i and set b_i to **any** integer (not necessary to be within $[1, n]$)

let $f(b)$ be the min number of operations until b contains a consecutive subarray of at least k elements.

a consecutive array with k elements starting at index i is such that $b_j = b_{j-1} + 1$ for all $i < j \leq i + k - 1$

$1 \leq k \leq n \leq 2 \cdot 10^5$, $1 \leq q \leq 2 \cdot 10^5$, $1 \leq a_i \leq n$

given array A and k, and q queries $[l, r]$, for each query

9.1 G1 (easy)

output $f([a_l, \ldots, a_{l+k-1}])$

solution
precompute $f(b)$ for each subarray of length k using **sliding window**. there will be in total $(n - k + 1)$ such subarrays. to calculate $f(b)$, convert each element from a_i to $(a_i - i)$. then maintain cnt for each distinct value of $(a_i - i)$ in the subarray.

$f(b)$ is $k - \max(cnt)$ i.e. min operations to equalize array / make all elements in array the same. this is because it is optimal (takes least no. of ops to equalize the array) by making every element equal to the value of $(a_i - i)$ that appears the most in the subarray.

each time sliding window is moved, decrement cnt for $A[l]$ and increment cnt for $A[r + 1]$

for each query, return $ans[l]$, where $ans[l]$ is the value of $f([a_l, \ldots, a_{l+k-1}])$

```
#include "bits/stdc++.h"
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx,avx2,sse,sse2")
#define fast ios_base::sync_with_stdio(0) , cin.tie(0) , cout.tie(0)
#define endl '\n'
#define int long long
#define f first
#define mp make_pair
#define s second
using namespace std;

void solve(){
    int n, k, q; cin >> n >> k >> q;
    int a[n + 1]; for(int i = 1; i <= n; i++) cin >> a[i];
    map <int,int> m;
    multiset <int> tot;
```

```
    for(int i = 1; i <= n; i++) tot.insert(0);
    for(int i = 1; i < k; i++){
        tot.erase(tot.find(m[a[i] - i]));
        m[a[i] - i]++;
        tot.insert(m[a[i] - i]);
    }
    int ret[n + 1];
    for(int i = k; i <= n; i++){
        tot.erase(tot.find(m[a[i] - i]));
        m[a[i] - i]++;
        tot.insert(m[a[i] - i]);
        int p = i - k + 1;
        ret[p] = k - *tot.rbegin();
        tot.erase(tot.find(m[a[p] - p]));
        m[a[p] - p]--;
        tot.insert(m[a[p] - p]);
    }
    while(q--){
        int l, r ; cin >> l >> r;
        cout << ret[l] << endl;
    }
    tot.clear();
    m.clear();
}

signed main()
{
    fast;
    int t;
    cin >> t;
    while(t--){
        solve();
    }
}
```

9.2 G2 (hard)

output $\sum_{j=l+k-1}^{r} f([a_l, \ldots, a_j])$

solution
let $c_i = f([a_i, \ldots, a_{i+k-1}])$. the problem simplifies to finding $\sum_{j=l}^{r-k+1}(\min_{i=l}^{j} c_i)$

9.3 G3 (extreme)

output $\sum_{i=l}^{r-k+1} \sum_{j=i+k-1}^{r} f([a_i, \ldots, a_j])$