

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

```
1 def isSubsetSum(arr, n, target):
2     if target == 0:
3         return True
4     if n == 0 and target != 0:
5         return False
6
7     if arr[n-1] > target:
8         return isSubsetSum(arr, n-1, target)
9
10    return isSubsetSum(arr, n-1, target) or isSubsetSum(arr, n-1, target - arr[n-1])
```

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

```
1 def isSubsetSum(arr, n, target):
2
3     # dp[i][j] = True if you can sum to j using first i elements
4     dp = [[False for _ in range(target + 1)] for _ in range(n + 1)]
5
6     # If target is 0, subset sum is True for any set (empty subset).
7     for i in range(n + 1):
8         dp[i][0] = True
9
10    # Fill the subset table
11    for i in range(1, n + 1):
12        for j in range(1, target + 1):
13            if arr[i-1] > j:
14                dp[i][j] = dp[i-1][j]
15            else:
16                dp[i][j] = dp[i-1][j] or dp[i-1][j - arr[i-1]]
17
18    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

```
1 def assignApartments(n: int, m: int, k: int, A: list[int], T: list[int]):
2     sorted_apt = sorted(A)
3     sorted_tent = sorted(T)
4     res = 0
5     i = 0
6     j = 0
```

```

7  while (i < n and j < m):
8      if A[i] + k <= T[j] or A[i] - k <= T[j]: # if abs(A[i] - T[j]) <= k
9          i += 1
10         j += 1
11         res += 1
12     else:
13         if A[i] + k > T[j]:
14             j += 1
15         else:
16             i += 1
17     return res

```

2.2 c++

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int MAX_N = 2e5;
6
7  int n, m, k, a[MAX_N], b[MAX_N], ans;
8
9  void solve() {
10     cin >> n >> m >> k;
11     for (int i = 0; i < n; ++i) cin >> a[i];
12     for (int i = 0; i < m; ++i) cin >> b[i];
13     sort(a, a + n);
14     sort(b, b + m);
15     int i = 0, j = 0;
16     while (i < n && j < m) {
17         // Found a suitable apartment for the applicant
18         if (abs(a[i] - b[j]) <= k) {
19             ++i;
20             ++j;
21             ++ans;
22         } else {
23             // If the desired apartment size of the applicant is too big,
24             // move the apartment pointer to the right to find a bigger one
25             if (a[i] - b[j] > k) {
26                 ++j;
27             }
28             // If the desired apartment size is too small,
29             // skip that applicant and move to the next person
30             else {
31                 ++i;
32             }
33         }
34     }
35     cout << ans << "\n";
36 }
37
38 int main() {
39     ios_base::sync_with_stdio(false);
40     cin.tie(nullptr);
41     solve();
42     return 0;
43 }

```

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

```
1 def ferrisWheel(n: int, x: int, W: list[int]):
2     sorted(W)
3     res = 0
4     i = 0
5     j = n - 1
6     in_gondola = [False] * n
7     while (i < j):
8         if (W[i] + W[j] <= x):
9             res += 1
10            in_gondola[i] = in_gondola[j] = True
11            i += 1
12            j -= 1
13        else:
14            j -= 1
15    for k in range(0, n):
16        if not in_gondola[k]:
17            res += 1
18    return res
```

4 Maximum XOR Score Subarray Queries

for each query, return the answer from the operation: for the range $\text{nums}[l, r]$, replace $\text{nums}[i]$ with $\text{nums}[i]$ XOR $\text{nums}[i+1]$ 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:

$[x_1 \ x_2] \rightarrow x_1 \oplus x_2$
 $[x_1 \ x_2 \ x_3] \rightarrow [x_1 \oplus x_2 \ x_2 \oplus x_3] \rightarrow x_1 \oplus x_3$
 $[x_1 \ x_2 \ x_3 \ x_4] \rightarrow [x_1 \oplus x_3 \ x_2 \oplus x_4] \rightarrow x_1 \oplus x_2 \oplus x_3 \oplus x_4$
 $[x_1 \ x_2 \ x_3 \ x_4 \ x_5] \rightarrow [x_1 \oplus x_2 \oplus x_3 \oplus x_4 \ x_2 \oplus x_3 \oplus x_4 \oplus x_5] \rightarrow x_1 \oplus x_5$
 $[x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6] \rightarrow [x_1 \oplus x_5 \ x_2 \oplus x_6] \rightarrow x_1 \oplus x_2 \oplus x_5 \oplus x_6$
... and so on

5 Inversions

5.1 Global and Local Inversions (LC775)

global inversion = $\text{nums}[i] > \text{nums}[j]$ for $0 \leq i < j < n$.

local inversion = $\text{nums}[i] > \text{nums}[i + 1]$ for $0 \leq i < n - 1$.

check if no. of global inversions == no. of local inversions

input:

nums = [1, 0, 2]

constraints: $1 \leq n \leq 10^5$

```
1 # if all GI = LI, then we cannot find an i and j such that i + 2 <= j and A[i] > A[j]
2 def isIdealPermutation(self, A):
3     cmax = 0
4     for i in range(len(A) - 2):
5         cmax = max(cmax, A[i])
6         if cmax > A[i + 2]:
7             return False
8     return True
9
10 # solution 2
11 def isIdealPermutation(self, A):
12     # if any element is more than 2 places away from its correct position
13     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]

```
1 def minSwaps(A: list[int]):
2     res = 0
3     ones_count = 0
4     for i in range(len(A)):
5         if A[i] == 0:
6             if ones_count > 0:
7                 res += ones_count
8             else:
9                 ones_count += 1
10    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 $x = 0, 1, \dots, i - 1$ 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.

```
1 def count_permutations_with_inversions(n, k):
2     # DP table to store the number of permutations of size n with exactly k inversions
3     dp = [[0 for _ in range(k + 1)] for _ in range(n + 1)]
4
5     # Base case: 1 permutation of size 0 with 0 inversions
6     dp[0][0] = 1
7
8     # Fill the DP table
```

```

9     for i in range(1, n + 1):
10         for j in range(k + 1):
11             # Compute dp[i][j] by summing over dp[i-1][j-x] for x = 0 to min(j, i-1)
12             dp[i][j] = sum(dp[i-1][j-x] for x in range(min(j, i-1) + 1))
13
14     # Return the number of permutations of size n with exactly k inversions
15     return dp[n][k]
16
17 def permutationWithKInversions(n: int, k: int):
18     MOD = 1000000007
19     dp = [0] * (k + 1)
20     dp[0] = 1
21
22     for i in range(1, n + 1):
23         new_dp = [0] * (k + 1)
24
25         for j in range(k + 1):
26             new_dp[j] = dp[j] % MOD
27
28             if j > 0:
29                 new_dp[j] = (new_dp[j] + new_dp[j - 1]) % MOD
30
31             if j - i >= 0:
32                 new_dp[j] = (new_dp[j] - dp[j - i] + MOD) % MOD
33
34         dp = new_dp
35
36     return dp[k]

```

5.4 Min Adjacent Swaps to Make Palindrome

```

1 def min_swaps_to_make_palindrome(s):
2     def can_be_palindrome(s):
3         from collections import Counter
4         freq = Counter(s)
5         odd_count = sum(1 for v in freq.values() if v % 2 != 0)
6         return odd_count <= 1
7
8     if not can_be_palindrome(s):
9         return -1 # Impossible to rearrange into a palindrome
10
11     s = list(s)
12     left, right = 0, len(s) - 1
13     swaps = 0
14
15     while left < right:
16         # If the characters match, move inward
17         if s[left] == s[right]:
18             left += 1
19             right -= 1
20         else:
21             # Find the match for s[left] on the right side
22             l, r = left, right
23             while r > l and s[r] != s[left]:
24                 r -= 1
25
26             # If we found a match for s[left]
27             if r == l:
28                 # If the left element has no matching pair, swap it forward (for odd-length strings)
29                 s[l], s[l+1] = s[l+1], s[l]
30                 swaps += 1
31             else:
32                 # Swap to bring s[r] to the right position

```

```

33         for i in range(r, right):
34             s[i], s[i+1] = s[i+1], s[i]
35             swaps += 1
36             left += 1
37             right -= 1
38
39     return swaps

```

6 Movie Festival (CSES)

given a list of movies with $[start_i, end_i]$, 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

```

1 def maxMovies(A: list[list[int]]):
2     ans = 0
3     currentMovieEnd = -1
4     sorted(A, key = lambda x: x[1])
5     for [start, end] in A:
6         if start >= currentMovieEnd:
7             currentMovieEnd = start
8             ans += 1
9     return ans

```

7 Maximum Subarray Sum

```

1 def mss(A: list[int]):
2     ans = 0
3     sum = 0
4     for i in range(len(A)):
5         if sum + A[i] > 0:
6             sum += A[i]
7         else:
8             sum = 0
9     ans = max(ans, sum)
10    return ans

```

8 Firefly Queries (CF 2009F)

given array A of length n , create $B = C_1 + C_2 + \dots + C_n$ 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.

$1 \leq n \leq 2 \cdot 10^5, 1 \leq a_i \leq 10^6$

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 int main() {
5     int t;
6     cin >> t;
7     while (t--) { // test cases
8         ll n, q; // n and no. of queries
9         cin >> n >> q;
10        vector<ll> a(n), ps(1); // array A and prefix sum
11        for (ll &r : a) {
12            cin >> r;
13            ps.push_back(ps.back() + r);
14        }
15        for (ll &r : a) {
16            ps.push_back(ps.back() + r);
17        }

```

```

18     while (q--) {
19         ll l, r;
20         cin >> l >> r;
21         l--; r--; // l and r are 1-indexed
22         ll i = l / n, j = r / n; // which subarray does it belong to
23         l %= n; r %= n; // find remainder (suffix and prefix)
24         cout << ps[n] * (j - i + 1) - (ps[i + 1] - ps[i]) - (ps[j + n] - ps[j + r + 1]) << "\n";
25     }
26 }
27 }

```

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

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

```

1 #include "bits/stdc++.h"
2 #pragma GCC optimize("O3,unroll-loops")
3 #pragma GCC target("avx,avx2,sse,sse2")
4 #define fast ios_base::sync_with_stdio(0) , cin.tie(0) , cout.tie(0)
5 #define endl '\n'
6 #define int long long
7 #define f first
8 #define mp make_pair
9 #define s second
10 using namespace std;
11
12 void solve() {
13     int n, k, q; cin >> n >> k >> q;
14     int a[n + 1]; for(int i = 1; i <= n; i++) cin >> a[i];
15     map <int,int> m;
16     multiset <int> tot;

```



```

17     for(int i = 1; i <= n; i++) tot.insert(0);
18     for(int i = 1; i < k; i++){
19         tot.erase(tot.find(m[a[i] - i]));
20         m[a[i] - i]++;
21         tot.insert(m[a[i] - i]);
22     }
23     int ret[n + 1];
24     for(int i = k; i <= n; i++){
25         tot.erase(tot.find(m[a[i] - i]));
26         m[a[i] - i]++;
27         tot.insert(m[a[i] - i]);
28         int p = i - k + 1;
29         ret[p] = k - *tot.rbegin();
30         tot.erase(tot.find(m[a[p] - p]));
31         m[a[p] - p]--;
32         tot.insert(m[a[p] - p]);
33     }
34     while(q--){
35         int l, r ; cin >> l >> r;
36         cout << ret[l] << endl;
37     }
38     tot.clear();
39     m.clear();
40 }
41
42 signed main()
43 {
44     fast;
45     int t;
46     cin >> t;
47     while(t--){
48         solve();
49     }
50 }

```

9.2 G2 (hard)

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

solution

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

9.3 G3 (extreme)

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

10 Rendez-vous de Marian et Robin (CF 2014E)

n vertices and m bidirectional edges

each edge takes m_i time to travel, all edge weights are even.

M starts at node 1 and R starts at node n . they want to meet at a node in the shortest time, and they cannot meet on an edge.

some of the nodes have a horse, once on a horse they can cut their travel time for all subsequent edges by $1/2$.

find the shortest time for them to meet at any edge, or -1 if they cannot meet

$2 \leq n \leq 2 \cdot 10^5$, $1 \leq m \leq 2 \cdot 10^5$, no self loops or multiple edges.

solution

run dijkstras twice, once from 1, once from n . need to maintain two vectors, (1) distance to each vertex with a horse and (2) without a horse

```
1 #include <iostream>
2 #include <vector>
3 #include <set>
4
5 using namespace std;
6
7 void dijkstra(int s, vector<vector<long long>> &d, vector<vector<pair<int,long long>>> &graph, vector<
    bool> &hs){
8     auto cmp = [&](auto &a, auto &b){return make_pair(d[a.first][a.second],a) < make_pair(d[b.first][b.
    second],b);};
9     set<pair<int,int>,decltype(cmp)> q(cmp);
10
11     d[s][0] = 0;
12     q.insert({s,0});
13
14     while (q.size()){
15         auto [curv,curh] = *q.begin();
16         q.erase(q.begin());
17
18         bool horse = (curh || hs[curv]);
19         for (auto &[neighv, neighd] : graph[curv]){
20             long long dist = horse?neighd/2:neighd;
21             if (d[neighv][horse] > d[curv][curh] + dist){
22                 q.erase({neighv,horse});
23                 d[neighv][horse] = d[curv][curh] + dist;
24                 q.insert({neighv,horse});
25             }
26         }
27     }
28 }
29
30 void work(){
31     int n,m,h;
32     cin >> n >> m >> h;
33
34     vector<bool> hs(n);
35     vector<vector<pair<int,long long>>> graph(n);
36
37     for (int i=0;i<h;i++){
38         int c;
39         cin >> c;
40         hs[--c]=1;
41     }
42
43     for (int i=0;i<m;i++){
44         int a,b,c;
45         cin >> a >> b >> c;
46
47         a--,b--;
48         graph[a].push_back({b,c});
49         graph[b].push_back({a,c});
50     }
51
52     vector<vector<long long>> d1(n,vector<long long>(2,1e18));
53     vector<vector<long long>> d2(n,vector<long long>(2,1e18));
54 }
```

```

55     dijkstra(0,d1,graph,hs);
56     dijkstra(n-1,d2,graph,hs);
57
58
59     long long best = 1e18;
60     auto get = [&](int a){return max(min(d1[a][0],d1[a][1]),min(d2[a][0],d2[a][1]));};
61
62     for (int i=0;i<n;i++) best = min(best,get(i));
63
64     cout << (best==1e18?-1:best) << '\n';
65 }
66
67 int main(){
68     int t;
69     cin >> t;
70     while (t-->0) work();
71
72     return 0;
73 }

```

11 Hungry Games (CF 1994A)

Yaroslav is interested in how many (contiguous) subsegments of mushrooms can be selected such that the character's final toxicity level g is not zero. The game mechanics are as follows:

The character starts with a toxicity level $g = 0$. For each mushroom with toxicity level k , the toxicity level of the character is increased by k .

If at any point $g > x$ (the maximum allowable toxicity level), the toxicity level g resets to zero. The goal is to count the number of subsegments $[l, r]$ such that the final toxicity level g after eating mushrooms from l to r is not zero.

You need to find how many such valid subsegments exist where the toxicity level does not exceed x and doesn't reset to zero after completing the subsegment.

solution

$O(n \log n)$

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 using ll = long long;
5
6 int main() {
7     cin.tie(nullptr)->sync_with_stdio(false);
8
9     int tests;
10    cin >> tests;
11    while (tests-->0) {
12        int n;
13        ll x;
14        cin >> n >> x;
15        vector<ll> a(n + 1);
16        for (int i = 1; i <= n; ++i) cin >> a[i];
17        partial_sum(a.begin() + 1, a.end(), a.begin() + 1);
18        vector<int> dp(n + 2);
19        for (int i = n - 1; i >= 0; --i) {
20            int q = upper_bound(a.begin(), a.end(), a[i] + x) - a.begin();
21            dp[i] = dp[q] + q - i - 1;
22        }

```

```

23     cout << accumulate(dp.begin(), dp.end(), 0ll) << '\n';
24 }
25 }

```

12 Find Minimum Operations (CF 2020A)

given n and k , each operation you can subtract any power of k from n . return the min. operations needed to make n equal to 0.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int find_min_oper(int n, int k){
5     if(k == 1) return n;
6     int ans = 0;
7     while(n){
8         ans += n%k; // basically get the base k representation of n
9         n /= k;
10    }
11    return ans;
12 }
13
14 int main()
15 {
16     int t;
17     cin >> t;
18     while(t--){
19         int n,k;
20         cin >> n >> k;
21         cout << find_min_oper(n,k) << "\n";
22     }
23     return 0;
24 }

```

13 Brightness Begins (CF 2002B)

for n light bulbs you do the following operation n times from $i = 1, 2, \dots, n$: flip every i^{th} bulb (from on to off and vice versa). e.g. flip 1, 2, 3, 4..., then flip 2, 4, 6..., then 3, 6, 9...

find the minimum n such that there are exactly k light bulbs switched on. all bulbs are initially switched on.

solution

if x is a perfect square, then it has an odd number of divisors. in all other cases (where x is not a perfect square), x will have an even number of divisors (1 included). refer to the theorem for no. of divisors i.e. for $n = p_1^a \cdot p_2^b \dots$, no of divisors of $n = (a+1)(b+1)\dots$

binary search for the value of n . there will be $n - \lfloor \sqrt{n} \rfloor$ bulbs switched on, because $\lfloor \sqrt{n} \rfloor$ equals the number of perfect squares up to and including n (which are switched off)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int t;
5     cin >> t;
6     while(t--){
7         long long k, l = 1, r = 2e18;
8         cin >> k;
9         while(r-l > 1){
10             long long mid = (l+r)>>1;

```

```

11         long long n = mid - int(sqrtl(mid));
12         if(n >= k) r = mid;
13         else l = mid;
14     }
15     cout << r << "\n";
16 }
17 return 0;
18 }

```

14 Bitwise Balancing (CF 2002C)

Given b , c and d , find if there exists an a such at $(a \mid b) - (a \& c) = d$ and return any of them. if it doesn't exist, return -1

solution

since $(a \mid b) = x$ is greater than or equal to $(a \& c) = y$, at each bit, x will not need to borrow from more significant bits to perform the subtraction.

so for the k^{th} bit of d i.e. d_k , if $d_k = 0$ then either $x = 0$ and $y = 0$, or $x = 1$ and $y = 1$.

in total there are $2^3 = 8$ combinations of b , c and d . so if the k^{th} bit in b , c and d do not fit into any of the 8 cases, then it is not possible to find an a that fits the requirements

```

1 #include <bits/stdc++.h>
2 #define ll long long
3 using namespace std;
4
5 void solve() {
6     ll a = 0, b, c, d, pos = 1, bit_b, bit_c, bit_d, mask = 1;
7     cin >> b >> c >> d;
8     for (ll i = 0; i < 62; i++) {
9         if (b&mask) bit_b = 1;
10        else bit_b = 0;
11        if (c&mask) bit_c = 1;
12        else bit_c = 0;
13        if (d&mask) bit_d = 1;
14        else bit_d = 0;
15        if ((bit_b && (!bit_c) && (!bit_d)) || ((!bit_b) && bit_c && bit_d)) {
16            pos = 0;
17            break;
18        }
19        if (bit_b && bit_c) {
20            a += (1ll-bit_d)*mask;
21        } else {
22            a += bit_d*mask;
23        }
24        mask<<=1;
25    }
26    if (pos) {
27        cout << a << "\n";
28    } else {
29        cout << -1 << "\n";
30    }
31 }
32
33 int main() {
34     ll t;
35     cin >> t;
36     while (t--) {
37         solve();

```

```
38     }  
39 }
```