mBIT Sample Problems Solutions

mBIT Organizers

June 2019

These are the solutions to the sample problems. Each answer consists of a brief explanation of the solution followed by the code. The code to is in either Python, Java, or C++. Keep in mind there are many ways to solve some of these problems.

Contents

§1 Rookie

§1.1 Easy: Number Jumping

Write a loop that iterates through the numbers 0 to N, incrementing by 2. Print the numbers, making sure they are space separated.

```python
#Python: Number Jumping (Rookie)
n = int(input())
i = 0
while i <= n:
    print(i, end=" ")
    i += 2
```

§1.2 Medium: Array Division

We maintain partial sums representing the sums of the left and right arrays. All the elements begin in the right array, and each time we add an element from the right array to the left array, we check if the partial sum of the left array is less than the partial sum of the right array. If it is, then it is valid to split the array at that location and we increment our counter by 1. In the solution shown below, we don't bother keeping track of the right partial sum since that's just the total sum of the array minus the left partial sum.

```java
// Java: Array Division (Rookie)
import java.io.*;
import java.util.*;

public class ArrayDivision {

   public static void main(String[] args) throws IOException {

      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      int n = Integer.parseInt(br.readLine());
      int[] arr = new int[n];
      StringTokenizer st = new StringTokenizer(br.readLine());
      int sum = 0;
      for (int i=0; i<n; i++) {
         arr[i] = Integer.parseInt(st.nextToken());
         sum += arr[i];
      }

      int partialSum = 0, ret = 0;
      for (int i=0; i<n-1; i++) {
         partialSum += arr[i];
         if (partialSum < sum - partialSum)
            ret++;
      }

      System.out.println(ret);

   }

}
```

§1.3 Hard: Bunny Market

Consider using i days. How do we know whether it is possible to reach B bunnies by i days? Well, if Farmer Tim never buys any bunnies, he will have $2^{i-1}A$ bunnies by day i. Thus, he must get a total of $B - 2^{i-1}A$ bunnies from the market or from the reproduction of the bunnies he got at the market. It is not obvious, but it is important, to note that this is possible to achieve as long as $B - 2^{i-1}A$ is divisible by K and $\frac{B-2^{i-1}A}{K} < 2^i$ (Try it yourself!). This can be proven using the fact that doubling overnight leads itself nicely to the binary representation of B and then playing with the bounds and indexing, but this is left as an exercise to the reader. Using that fact, we can easily check all possible values of i between 1 and 99 (We know that $i < 99$ because anything greater than 99 must definitely result in too many bunnies), using the infinite integer size in Python to our advantage.
If you do not understand the solution, don't worry! There's a reason it is classified as Hard.

```python
#Python: Bunny Market (Rookie)
inp = input().split(" ")
a, b, k = int(inp[0]), int(inp[1]), int(inp[2])

ans = -1
for i in range(100,0,-1):#decrements i from 99 to 1
    #is it possible with exactly i days?
    need = b-2**(i-1)*a
    if need >= 0 and need%k==0 and need//k < 2**i:
        ans = i

print(ans)
```

§2 Varsity

§2.1 Easy: Log Jumping

We may apply dynamic programming to solve this. Let $dp[i]$ represent the minimum number of jumps that Elias could take to get to location i, or $N+1$ if it is not possible (or there is a crocodile at location i). We note that

$$dp[i] = \min_{1 \le j \le M} (dp[i - s_j] + 1)$$

We also must take into account that we should only consider j where $i - s_j \geq 0$ and whether $dp[i - s_j]$ could possibly be $N+1$ depending on where there are crocodiles. This yields an $O(MN)$ solution, which will run in time. There are other valid solutions which use BFS or DFS.

If you are not familiar with dynamic programming or Big O notation, we recommend that you look into them if you are going to participate in Varsity.

```python
#Python: Log Jumping (Varsity)
inp = input().split(" ")
n, m, k = int(inp[0]),int(inp[1]),int(inp[2])

steps = [int(x) for x in input().split(" ")]
crocs = [int(x) for x in input().split(" ")]
hasCroc = []
dp = []

#Initialize everything to N+1
for i in range(n+1):
    dp.append(n+1)
    hasCroc.append(False)
for c in crocs:
    hasCroc[c] = True
dp[0] = 0  #0 steps to get to start

#Perform dynamic programming
for i in range(1,n+1):
    if hasCroc[i]:
        continue
    best = n+1
    for step in steps:
        if i-step >= 0 and dp[i-step]+1 < best:
            best = dp[i-step]+1
    dp[i] = best

if dp[n] == n+1:
    print(-1)
else:
    print(dp[n])
```

§2.2 Medium: Monster Sums

We can precompute the sums that Angelina and Bert can make using prefix sums. We then insert them into two separate sorted lists. We then iterate through the possible sums Angelina could make, while maintaining two pointers, one to the closest sum Bob could make that's greater than Angelina's sum and one to the closest sum that Bob could make that's less than Angelina's sum. Checking through each values, we can do this in $O(N)$. Thus our run time is dominated by sorting, so our overall runtime is $O(N \log N)$.

Note: when keeping track of sums, we need to be careful to use long long because the sums may overflow.

```
//C++: Monster Sums (Varsity)
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
const int MAX = 1e5 + 5;

ll a [MAX], b [MAX];

vector<ll> prefixA, prefixB;

int main(){
    int N; cin >> N;

    for(int i = 0; i<N; i++) cin >> a[i];
    for(int i = 0; i<N; i++) cin >> b[i];

    for(int i = 0; i<N; i++){
        if(i == 0)
            prefixA.push_back(a[i]),
                prefixB.push_back(b[i]);
        else
            prefixA.push_back(prefixA[i-1] + a[i]),
                prefixB.push_back(prefixB[i-1] + b[i]);
    }

    sort(prefixA.begin(), prefixA.end()), sort(prefixB.begin(), prefixB.end());

    int ptr1 = -1, ptr2 = 0;
    ll ans = 1e18;

    for(int i = 0; i<N; i++){
        while(ptr1+1 < N && prefixB[ptr1+1] <= prefixA[i]) ptr1++;
        while(ptr2+1 < N && prefixB[ptr2]<a[i]) ptr2++;
        if(ptr1 != -1 && prefixB[ptr1] <= prefixA[i])
            ans = min(ans, prefixA[i] - prefixB[ptr1]);
        if(prefixB[ptr2] >= prefixA[i])
            ans = min(ans, prefixB[ptr2] - prefixA[i]);
    }

    cout << ans << endl;
    return 0;
}
```

§2.3 Hard: Close Subsequences

We maintain M segment trees, one for each possible residue class mod M. We also compress the the values to 0 to N, but we still have to store the original values to keep track of their values mod M. Then, we can apply dynamic programming. Let $dp[i]$ = Longest close sequence ending at i. To compute $dp[i]$, we look at the possible mods for the previous element, and query the segment tree. This results in a time complexity of $O(MN \log N)$.

```
//C++: Close Subsequences (Varsity)
#include<bits/stdc++.h>
using namespace std;

template<int SZ> struct SegmentTree{
    int mx[2*SZ];
    void set(int i, int x){
        int cur = query(i, i);
        if(x>cur){
            update(i, x - cur);
        }
    }
    void update(int i, int x, int l = 0, int r = SZ - 1, int idx = 0) {
        if (r < i || l > i) return;
        if (l == r) { mx[idx] += x; return; }
        int m = (l + r)/2;
        update(i, x, l, m, 2*idx+ 1);
        update(i, x, m + 1, r, 2*idx + 2);
        mx[idx] = max(mx[2*idx+ 1], mx[2*idx + 2]);
    }
    int query(int lo, int hi, int l = 0, int r = SZ-1, int idx = 0){
        if(r < lo || l > hi) return 0;
        if(l >= lo && r <= hi) return mx[idx];
        int m = (l + r)/2;
        return max(query(lo, hi, l, m, 2*idx + 1), query(lo, hi, m+1, r, 2*idx
            + 2));
    }
};

const int MAXN = 1e5 + 5;
const int MAXK = 5;

SegmentTree<(1<<17)> seg [MAXK];

int dp [MAXN];
int a [MAXN];
int original [MAXN];

unordered_map<int, int> convert;

int main(){
    int n, k, m, b;
    cin >> n >> k >> m >> b;

    set<int> compress;
    for(int i = 0; i<n; i++) cin >> a[i];
    for(int i = 0; i<n; i++) compress.insert(a[i]), original[i] = a[i];
    int idx = 0;
```

```
    for(auto x: compress){
        convert[x] = idx;
        idx++;
    }
    for(int i = 0; i<n; i++) a[i] = convert[a[i]];

    for(int i = 0; i<n; i++){
        dp[i] = 1;
        for(int mod = 0; mod<m; mod++){
            if(abs(mod - original[i])%m<= b){
                auto lower = compress.lower_bound(original[i] - k);
                auto upper = prev(compress.upper_bound(original[i]+k));
                dp[i] = max(dp[i], seg[mod].query(max(0, convert[*lower]),
                    min(n-1, convert[*upper]))+ 1);
            }
        }
        seg[original[i]%m].set(a[i], dp[i]);
    }

    int ans = 0;
    for(int i = 0; i<n; i++) ans = max(ans, dp[i]);
    cout << ans << endl;
    return 0;
}
```

Interesting note: the problem can also be solved without bounds on M. However, this requires use of a 2D sparse segment tree, which, we felt may have been too obscure to expect people to know of. We put in the code for completeness. This runs in $N \log^2 N$, as querying takes $\log^2 N$ time.

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 100013;

const int SZ = (1<<17);

struct node {
    int val;
    node* c[2];

    node() {
        val = 0;
        c[0] = c[1] = NULL;
    }

    void upd(int ind, int v, int L = 0, int R = SZ-1) {
        if (L == ind && R == ind) { val += v; return; }

        int M = (L+R)/2;
        if (ind <= M) {
            if (!c[0]) c[0] = new node();
            c[0]->upd(ind,v,L,M);
        } else {
            if (!c[1]) c[1] = new node();
            c[1]->upd(ind,v,M+1,R);
```

```
        }

        val = 0;
        if (c[0]) val = max(c[0]->val, val);
        if (c[1]) val = max(c[1]->val, val);
    }

    int query(int low, int high, int L = 0, int R = SZ-1) {
        if (low <= L && R <= high) return val;
        if (high < L || R < low) return 0;

        int M = (L+R)/2;
        int t = 0;
        if (c[0]) t = max(t, c[0]->query(low,high,L,M));
        if (c[1]) t = max(t, c[1]->query(low,high,M+1,R));
        return t;
    }

    void UPD(int ind, node* c0, node* c1, int L = 0, int R = SZ-1) {
        if (L != R) {
            int M = (L+R)/2;
            if (ind <= M) {
                if (!c[0]) c[0] = new node();
                c[0]->UPD(ind,c0 ? c0->c[0] : NULL,c1 ? c1->c[0] : NULL,L,M);
            } else {
                if (!c[1]) c[1] = new node();
                c[1]->UPD(ind,c0 ? c0->c[1] : NULL,c1 ? c1->c[1] : NULL,M+1,R);
            }
        }
        val = 0;
        if (c0) val = max(val, c0->val);
        if (c1) val = max(val, c1->val);
    }
};
struct Node {
    node seg;
    Node* c[2];

    void upd(int x, int y, int v, int L = 0, int R = SZ-1) {
        if (L == x && R == x) {
            seg.upd(y,v);
            return;
        }

        int M = (L+R)/2;
        if (x <= M) {
            if (!c[0]) c[0] = new Node();
            c[0]->upd(x,y,v,L,M);
        } else {
            if (!c[1]) c[1] = new Node();
            c[1]->upd(x,y,v,M+1,R);
        }

        seg.UPD(y,c[0] ? &c[0]->seg : NULL,c[1] ? &c[1]->seg : NULL);
    }

    int query(int x1, int x2, int y1, int y2, int L = 0, int R = SZ-1) {
```

```
        if (x1 <= L && R <= x2) return seg.query(y1,y2);
        if (x2 < L || R < x1) return 0;

        int M = (L+R)/2;
        int t = 0;
        if (c[0]) t = max(t, c[0]->query(x1,x2,y1,y2,L,M));
        if (c[1]) t = max(t, c[1]->query(x1,x2,y1,y2,M+1,R));
        return t;
    }
};

Node seg;

int dp [MAXN];
int a [MAXN];
int original [MAXN];

unordered_map<int, int> convert;

int main(){

    int  n, k, m, b;
    cin >> n >> k >> m >> b;
    set<int> compress;
    for(int i = 0; i<n; i++) cin >> a[i];
    for(int i = 0; i<n; i++) compress.insert(a[i]), original[i] = a[i];
    int idx = 0;
    for(auto x: compress){
        convert[x] = idx;
        idx++;
    }
    for(int i = 0; i<n; i++) a[i] = convert[a[i]];

    for(int i = 0; i<n; i++){
        dp[i] = 1;
        auto lower = compress.lower_bound(original[i] - k);
        auto upper = prev(compress.upper_bound(original[i]+k));
        int lo = max(convert[*lower], 0);
        int hi = min(convert[*upper], n-1);
        int mod = original[i]%m;
        int bot = max(0, mod - b);
        int top = min(n-1, mod + b);
        dp[i] = max(dp[i], seg.query(bot, top, lo, hi) + 1);
        int check = seg.query(mod, mod, a[i], a[i]);
        if(check< dp[i]) seg.upd(mod, a[i], dp[i] - check);
    }
    int ans = 0;
    for(int i = 0; i<n; i++) ans = max(ans, dp[i]);
    cout << ans << endl;
    return 0;
}
```