Subset Mex

June 11, 2025

1 Problem

Given a set of integers (it can contain equal elements) e.g. for the set with 2 zeros, 100 ones, the input will have size 102, it will look like $0, 0, 1, 1, ..., 1$.

You have to split it into two subsets A and B (both of them can contain equal elements or be empty). You have to maximize the value of $mex(A) + mex(B)$.

Here mex of a set denotes the smallest non-negative integer that doesn't exist in the set. For example:

- $mex(1, 4, 0, 2, 2, 1) = 3$
- $mex(3, 3, 2, 1, 3, 0, 0) = 4$
- $mex(\emptyset) = 0$ (mex for empty set)

The set is splitted into two subsets A and B if for any integer number x the number of occurrences of x into this set is equal to the sum of the number of occurrences of x into A and the number of occurrences of x into B.

2 Solution

We present an $O(n)$ time solution where n is the size of input set.

```
f(S,n){
        count = an array of size n, all entries initialized to 0
        for (s in S){
                if(0 <= s && s < n)
                        count[s]++;
        }

        A_1 = empty list
        B = empty list
        stop_B = false
        for(i = 0; i < n; ++i){
                if(count[i] == 0)
                        break;
                else if (count[i] == 1)
                        stop_B = true;

                if(!stop_B)             B.push_back(i);
                A_1.push_back(i);
        }
        alpha = (A_1.empty() == true) ? 0 : A_1.back() + 1;
        beta = (B.empty() == true) ? 0 : B.back() + 1;
        return alpha + beta;
}
```

When the 2nd for loop terminates, notice there may be some elements $A_2 \subseteq S$ not assigned to either A_1 or B. We call $(A = A_1 \cup A_2, B)$ our final partition.

Lemma 1. (A, B) is a partition of S.

proof. We need to show the number of occurrences of any integer x in S is equal to the sum of the number of occurrences of x in A and the number of occurrences of x in B. We can focus on $x \in \{0, ..., n-1\}$ because it clearly holds for the other integers (they're not processed in the 2nd for loop, if they occur y times in S they'll occur y times in A_2). So for $x \in \{0, ..., n-1\}$:

- If it appears 0 times in S, it will not be in A_1 or B because the for loop breaks. Clearly it'll not be in A_2. So it occurs 0 times in both A and B.

- If it appears 1 time in S, it'll not be in B because `stop_b` is true. It's clear that it'll either be in A_1 or A_2 but not both. So it occurs 1 time in A and 0 time in B.

- If it appears $y > 1$ time in S, there're 3 cases depending on (i) whether the x iteration is even executed or that the loop broke before that iteration (ii) the value of `stop_B` in the x iteration of the for loop.

(1) (x iteration executed and `stop_B = false` at that time) it appears 1 time in B, 1 time in A_1 and $y-2$ times in A_2

(2) (x iteration executed and `stop_B = true` at that time) it appears 0 time in B, 1 time in A_1 and $y-1$ times in A_2

(3) (x iteration not executed) it appears 0 time in B, 0 time in A_1 and y times in A_2

In all cases, the number of occurrences of any integer x in S is equal to the sum of the number of occurrences of x in A and the number of occurrences of x in B. ■

Lemma 2. $mex(A) = \alpha$ and $mex(B) = \beta$.

proof. If B is empty clearly $mex(B) = 0 = \beta$. Else by the way we construct B in the 2nd for loop, it'll be of the form $0, 1, ..., B.back()$ so $mex(B) = B.back() + 1 = \beta$.

If A_1 is empty, that means `count[0] == 0` so there's no 0 in A_2 either, it follows $mex(A) = 0 = \alpha$. Else by the way we construct A_1 in the 2nd for loop, it'll be of the form $0, 1, ..., A_1.back()$. If $A_1.back() = n-1$, then A_2 must be empty because size of A_1 is already n which is size of S. So $mex(A) = A_1.back() + 1 = \alpha$. Otherwise $0 < A_1.back() < n-1$, that means `count[A_1.back() + 1] == 0` the loop broke at iteration $A_1.back() + 1$. It follows $A_1.back() + 1$ will not be in A_2 either so $mex(A) = A_1.back() + 1 = \alpha$. ■

Lemma 3. Let (A^*, B^*) be the optimal partition, WLOG assume $mex(A^*) \geq mex(B^*)$. Then $mex(A^*) \leq \alpha$ and $mex(B^*) \leq \beta$.

proof. We first show $mex(A^*) \leq \alpha$ by case analysis :

(i) If $\alpha = 0$, then it must mean A_1 is empty which in turn means `count[0] == 0`. Hence there cannot be 0 in A^* thus $mex(A^*) = 0 = \alpha$.

(ii) If $\alpha = n$, because $mex(A^*) \leq |A^*|$ (as $mex(A^*)$ is the smallest non-negative integer that doesn't exist in A^*, it means all non-negative integers that come before $mex(A^*)$ must be in A^*) and $|A^*| \leq n$, we must have $mex(A^*) \leq \alpha$.

(iii) if $0 < \alpha < n$, that means $\alpha = A_1.back() + 1$ and $A_1.back() < n-1$. It follows `count[A_1.back() + 1] == 0` and the loop broke at iteration $A_1.back() + 1$. It follows A^* cannot contain $A_1.back() + 1$ thus $mex(A^*) \leq A_1.back() + 1 = \alpha$.

Then we show $mex(B^*) \leq \beta$ again by case analysis :

(I) If $\beta = 0$, then it must mean B is empty which in turn means `count[0] <= 1`. Notice there cannot be 0 in B^*, otherwise there's no 0 in A^* which means $mex(A^*) = 0 < 1 \leq mex(B^*)$ contradiction. Thus $mex(B^*) = 0 = \beta$.

(II) Although one can show this case will never occur. If $\beta = n$, exactly same as analysis of (ii), we have $mex(B^*) \leq |B^*| \leq n = \beta$.

(III) If $0 < \beta < n$, that means $\beta = B.back() + 1$ and $B.back() < n - 1$. It follows `count[B.back() + 1] <= 1`. If $mex(B^*) > \beta = B.back() + 1$, then $B.back() + 1$ must be in B^*, thus it cannot also be in A^*, which means $mex(A^*) \leq B.back() + 1 < mex(B^*)$, a contradiction. Thus $mex(B^*) \leq \beta$. ■

From lemma 3, we know $OPT = mex(A^*) + mex(B^*) \leq \alpha + \beta$. However, our algorithm finds a partition (A, B) (lemma 1) where $mex(A) + mex(B) = \alpha + \beta$ (lemma 2). It follows our algorithm outputs the optimal value.