6. Se define la función $mex : \mathcal{P}(\mathbb{N}) \to \mathbb{N}$ como

$$mex(X) = \text{mín}\{j : j \in \mathbb{N} \land j \notin X\}$$

Intuitivamente, mex devuelve, dado un conjunto X, el menor número natural que no está en x. Por ejemplo, $mex(\{0,1,2\}) = 3$, $mex(\{0,1,3\}) = 2$ y $mex(\{1,2,3,\ldots\}) = 0$.

Dado un vector de número $a_1 \ldots a_n$ queremos encontrar la permutación $b_1 \ldots b_n$ de los mismos que maximize

$$\sum_{i=1}^{n} mex(\{b_1 \ldots b_i\})$$

Por ejemplo, si el vector es $\{3,0,1\}$ podemos ver que la mejor permutación es $\{0,1,3\}$, que alcanza un valor de

$$mex(\{0\}) + mex(\{0,1\}) + mex(\{0,1,3\}) = 1 + 2 + 2 = 5$$

---

a) Proponer un algoritmo *greedy* que resuelva el problema y demostrar su correctitud. **Ayuda**: ¿Cuál el máximo valor que puede tomar $mex(X)$ si X tiene n elementos? Si $X \subseteq Y$, ¿Qué pasa con los valores $mex(X)$ y $mex(Y)$?

Estrategia

Empezamos con una lista S con las posiciones iniciales y una lista R vacia de tamaño $|S|$, 0-indexed.

Vamos iterando sobre los j elementos de S y:

- Si $S_j \geq |S|$ entonces ponemos S_j al ultimo disponible de R
- Si $S_j < |S|$:
- ‣ Si su lugar está ocupado, entonces al ultimo disponible de R
- ‣ Sino, $R_j := S_j$

Algoritmo

La solución propuesta trae problemas de implementación por lo que la modifiqué un poco, siguiendo la misma idea

```
MaxMex(S):
  R <- array de tamaño |S| con todo -1
  Pila <- pila vacia

  For j ∈ S:
    Si j < |S| entonces:
      R[j] := j

    Sino:
      Pila.agregar(j)

  For j: 0...|R|-1:
    Si R[j] = -1 entonces:
      R[j] := Pila.desapilar

  Return R
```

Precondiciones

Lo que queremos demostrar es que lo único que importa en la posición de los elementos de $S = \{s_0, s_1, ...s_{n-1}\}$ $(n = |S|)$ son los $s_i = i$ para que la solución sea óptima.

Lo que hace nuestro algoritmo es dar una salida $R = \{r_0, r_1, ..., r_{n-1}\}$ tal que:

Iteración 1:

- $i \in S \wedge 0 \leq i < n \underset{L}{\rightarrow} r_i = i \wedge S = S \setminus \{r_i\}$

Iteración 2:

- $\forall r_i \neq i :: r_i = s, s \in S \underset{L}{\rightarrow} S = S \setminus s$

Demostración

Haremos inducción en $|R|$

Caso base:

Si $|R| = 1$ entonces R tiene un solo elemento, r_0, por lo tanto una única forma de ordenarlo, por lo que es óptimo.

Si $r_0 = 0$, $mex(R) = 1$

Si $r_0 \neq 0$, $mex(R) = 0$

Paso inductivo

HI: $R = \{r_0, ..., r_{k-1}, r_k, ...r_{n-1}\}$ donde desde r_0 hasta r_{k-1} vale $r_i = i$ por lo que vale

$$mex(\{0\}) + ... + mex(\{0, ..., k-1\}) = 1 + ... + k$$

Que es trivialmente la máxima suma posible, $\sum_{i=1}^{k} i$

Si $r_k = k$:

Entonces sigue siendo la máxima suma posible, ya que $mex(\{0, ..., k\}) = k + 1$ entonces sumamos hasta el paso k un total de $\sum_{i=1}^{k+1} i$

Si $r_k \neq k$:

Sabemos que desde r_0 hasta r_{k-1} vale $r_i = i$, por lo que si $r_k \neq k \Rightarrow r_k > k$, pero el siguiente número natural disponible es k, por lo que si no existe en el conjunto, entonces cualquier $p > k$, $mex(\{0, ..., k-1, p, ...\}) = k$, por lo que todas las sumas siguientes sumarán siempre k ya que es el primer natural disponible.

$\therefore$ queda demostrado que nuestra solución greedy computa una solución óptima

---

b) Dar una implementación del algoritmo del inciso anterior con complejidad temporal $O(n)$.

```java
public static int[] maxMex(int[] S) {
    int[] R = new int[S.length];
    Arrays.fill(R, -1);

    Stack<Integer> pila = new Stack<>();

    for (int j:S) {
        if (j < S.length) R[j] = j;
        else pila.push(j);
    }
```

```
    for (int j=0; j < R.length ;j++) if (R[j] == -1) R[j] = pila.pop();

    return R;
}
```