TDA | Ejercicio 13 practica 1

Enunciado

Tenemos dos conjuntos de personas y para cada persona sabemos su habilidad de baile. Queremos armar la máxima cantidad de parejas de baile, sabiendo que para cada pareja debemos elegir exactamente una persona de cada conjunto de modo que la diferencia de habilidad sea menor o igual a 1 (en modulo). Ademas, cada persona puede pertenecer a lo sumo a una pareja de baile. Por ejemplo, si tenemos un multiconjunto con habilidades $\{1, 2, 4, 6\}$ y otro con $\{1, 5, 5, 7, 9\}$, la maxima cantidad de parejas es 3. Si los multiconjuntos de habilidades son $\{1, 1, 1, 1, 1\}$ y $\{1, 2, 3\}$, la máxima cantidad es 2.

Algoritmo recursivo

```
def F(G: [int], B: [int]) -> int:
    return f(sorted(G), sorted(B))

def f(G: [int], B: [int]) -> int:
    if not G or not B:
        return 0
    g = G[0]
    b = B[0]
    if b < g - 1:
        return f(G, B[1:])
    if g < b - 1:
        return f(G[1:], B)
    return 1 + f(G[1:], B[1:])
```

Demostración

Queremos ver que `F(G, B)` devuelve el máximo número de parejas que se pueden armar con los multiconjuntos (en este programa, listas) de habilidades dados por `G` y `B`. Vamos a probar que `f(G', B')` devuelve la respuesta que `F(G, B)` tiene que dar, asumiendo que `G'` y `B'` tienen los mismos elementos que `G` y `B` respectivamente, sólo permutando su orden, lo cual no cambia la solución esperada, dado que estamos cambiando la representación del multiconjunto, pero no sus elementos.

Lo vamos a hacer por inducción. Como en `f` estamos sacando siempre un elemento de `B` o de `G` (y a veces de ambos), vamos a definir:

```
tamaño(G, B) = len(G) + len(B)
```

Esto nos va a ayudar porque nuestras llamadas recursivas siempre llaman a `f` con instancias de menor `tamaño` que la que recibe. Esto es lo que necesitamos para hacer inducción.

Definimos entonces:

$$P(k) : f(G, B) \text{ es correcta para todos los argumentos (G, B) de tamaño(G, B) a lo sumo } k. \quad (1)$$

Caso base

El caso base es $P(0)$. Tenemos que probar que $f(G, B)$ es correcto para todos los argumentos de tamaño a lo sumo $k = 0$. Si `tamaño(G, B) = 0`, entonces `len(G) + len(B) = 0`, pero entonces como ambos `len(G)` y `len(B)` son enteros no-negativos, tenemos que `len(G) = len(B) = 0`. Luego, `G = B = []`. Entonces, no hay parejas posibles, porque una pareja tendría que tener un elemento de `G` y otro de `B`. Luego la respuesta correcta es `0`, y efectivamente nuestro programa devuelve `0`, en:

```
    if not G or not B:
        return 0
```

Paso inductivo

Ahora probemos el paso inductivo. Asumo $P(k)$, y quiero probar $P(k + 1)$.

Me dan un par (G, B) con $tamaño(G, B) = k + 1$. Si uno de los dos está vacío (no pueden ambos estar vacíos porque $tamaño(G, B) = k + 1 \geq 1$), no hay parejas posibles, luego la respuesta correcta es `0`, y el programa efectivamente devuelve eso

```
    if not G or not B:
        return 0
```

Si ninguno está vacío, ambos tienen un primer elemento, llamémoslo $g_0 \in G$ y $b_0 \in B$. Como G y B están ordenados crecientes, $g_0 \leq g \; \forall \; g \; \in G$, y $b_0 \leq b \; \forall \; b \in B$.

Consideremos ahora b_0 versus g_0.

- Si $b_0 < g_0 - 1$, entonces b_0 no puede estar aparejado con g_0. Pero más aún, como $g_0 \leq g \; \forall \; g \in G$, todos los elementos de G son al menos tan grandes como g_0, y luego b_0 no puede estar aparejado con nadie. Luego, toda solución a (G, B) es lo mismo que una solución a $(G, B \setminus \{b_0\})$. Como $tamaño(G, B \setminus \{b_0\}) = k < k + 1$, sabemos por inducción que f es correcta para ese caso, y luego nuestra f es también correcta para este caso de (G, B), porque devolvemos exactamente $f(G, B \setminus \{b_0\})$:

```
        if b < g - 1:
            return f(G, B[1:])
```

- Con un argumento similar, si $g_0 < b_0 - 1$, g_0 no puede estar aparejado con b_0, y b_0 es menor o igual que todos los elementos en B, luego g_0 no puede estar aparejado con nadie en B, y luego toda solución a (G, B) deja a g_0 sin usar, y luego es idéntica a una solución a $(G \setminus \{g_0\}, B)$. Luego nuestra f es correcta para este caso de (G, B), porque devolvemos exactamente $f(G \setminus \{g_0\}, B)$, y como $tamaño(G \setminus \{g_0\}, B) = k < k + 1$, por hipótesis inductiva f devuelve una solución óptima para ese caso:

```
        if g < b - 1:
            return f(G[1:], B)
```

Si ninguna de las dos condiciones vale, entonces $g_0 \geq b_0 - 1$, y $b_0 \geq g_0 - 1$. Esto es lo mismo que $b_0 - g_0 \leq 1$, y $g_0 - b_0 \leq 1$, o lo que es lo mismo, $|b_0 - g_0| \leq 1$, es decir, b_0 y g_0 son compatibles. Notemos que en nuestro programa esta es la última rama, donde devolvemos `1 + f(G[1:], B[1:])`. Vamos a probar dos lemas:

1. Si existe una solución óptima para (G, B) donde (g_0, b_0) están aparejados, entonces f es correcta para (G, B).
2. Existe una solución óptima para (G, B) donde (g_0, b_0) están aparejados.

Está claro que si probamos ambos lemas, probamos que f es correcta para (G, B). Como (G, B) era cualquier argumento de tamaño $k + 1$, esto prueba que $P(k) \Rightarrow P(k + 1)$, que es lo que queríamos demostrar.

Lema 1

Queremos probar que "Si existe una solución óptima para (G, B) donde (g_0, b_0) están aparejados, entonces f es correcta para (G, B)."

Supongamos que existe una solución óptima para (G, B) donde (g_0, b_0) están aparejados. Sea S esa solución. Luego, $S' = S \setminus \{(g_0, b_0)\}$ es una forma de aparejar a $G \setminus \{g_0\}$ y $B \setminus \{b_0\}$.

Podemos decir algo más fuerte: S' es *óptima* para $(G \setminus \{g_0\}, B \setminus \{b_0\})$. Si no lo fuera, sea S^* una solución a $(G \setminus \{g_0\}, B \setminus \{b_0\})$ que tiene más elementos que S'. Como S^* no menciona a g_0 ni a b_0, podemos construir $S^* \cup \{(g_0, b_0)\}$, que tiene $|S^*| + 1 > |S'| + 1 = |S|$ elementos, y es una solución a aparejar a G y B. Esto no puede pasar, porque S era óptima para (G, B), no puedo tener a $S^* \cup \{(g_0, b_0)\}$ más grande que S.

Luego, S' es óptima para $(G \setminus \{g_0\}, B \setminus \{b_0\})$. Por inducción, f es correcta para $(G \setminus \{g_0\}, B \setminus \{b_0\})$, porque $tamaño(G \setminus \{g_0\}, B \setminus \{b_0\}) = k - 1 < k + 1$. Luego $|S'| = |f(G \setminus \{g_0\}, B \setminus \{b_0\})|$, y luego $|S| = 1 + |S'| = 1 + |f(G \setminus \{g_0\}, B \setminus \{b_0\})|$. Luego f es correcta para (G, B), dado que devolvemos exactamente eso:

```
    return 1 + f(G[1:], B[1:])
```

Lema 2

Queremos probar que "Existe una solución óptima para (G, B) donde (g_0, b_0) están aparejados." Sabemos que g_0 y b_0 son compatibles.

Sea S cualquier solución óptima a (G, B). Sean S_G y S_B los elementos de G y B que *no* aparecen en S, respectivamente. Leér "single girls, single boys".

Partimos en casos:

- Si $g_0 \in S_G$ y $b_0 \in S_B$, podríamos considerar $S' = S \cup \{(g_0, b_0)\}$, que tiene un elemento más que S, y es solución a (G, B). Como S era solución óptima, esto no puede pasar.
- Si $g_0 \in S_G$ pero $b_0 \notin S_B$, sabemos que existe una pareja $(x, b_0) \in S$. Luego, podemos considerar $S' = (S \setminus \{(x, b_0)\}) \cup \{(g_0, b_0)\}$. Esto deja sin aparejar a x, pero empareja a g_0 y b_0, y como tiene el mismo número de elementos que S, vemos que S' es una solución óptima para (G, B) donde g_0 y b_0 están aparejados.
- Si $b_0 \in S_B$ pero $g_0 \notin S_G$, sabemos que existe una pareja $(g_0, y) \in S$. Luego, podemos considerar $S' = (S \setminus \{(g_0, y)\}) \cup \{(g_0, b_0)\}$. Esto deja sin aparejar a y, pero empareja a g_0 y b_0, y como tiene el mismo número de elementos que S, vemos que S' es una solución óptima para (G, B) donde g_0 y b_0 están aparejados.
- Si $g_0 \notin S_G$ y $b_0 \notin S_B$, entonces ambos están aparejados. Hay dos opciones:
  - Si están aparejados el uno al otro, es decir, (g_0, b_0) está en S, ya está, conseguimos una solución óptima que contiene a (g_0, b_0), y es S.

- Si no, existen $(x, b_0) \in S$ y $(g_0, y) \in S$. Quisieramos aparejar a b_0 con g_0, y a x con y, y para eso tenemos que probar que x e y son compatibles. Como b_0 era el elemento en B más chico, tenemos que $y \geq b_0$. De la misma manera sabemos que $x \geq g_0$. Veamos qué puede pasar:
  - Si $b_0 = g_0$, es decir, tienen la misma altura, entonces $y \geq b_0 = g_0$, y como en S están aparejados y con g_0, y sólo puede ser g_0 o $g_0 + 1$. Por el mismo motivo, $x \geq g_0 = b_0$, y en S están aparejados x con b_0, y entonces x sólo puede ser b_0 o $b_0 + 1$. Como b_0 y g_0 son el mismo número, ambos x e y sólo pueden ser b_0 o $b_0 + 1$, y luego x e y son compatibles.
  - Si $b_0 = g_0 + 1$, entonces $y \geq b_0 = g_0 + 1$, y como en S están aparejados y con g_0, tenemos que y es *exáctamente* $g_0 + 1$. Luego tenemos que $y = g_0 + 1 = b_0$. Como x y b_0 son compatibles, y $b_0 = y$, vemos que x e y son compatibles.
  - Si $g_0 = b_0 + 1$, pasa lo mismo que en el caso anterior. Tenemos $x \geq g_0 = b_0 + 1$, y como x y b_0 están aparejados en S, tenemos que x es *exáctamente* $b_0 + 1$. Luego tenemos que $x = b_0 + 1 = g_0$. Como y y g_0 son compatibles, y $g_0 = x$, vemos que x e y son compatibles.

  Como en todos los casos x e y son compatibles, podemos considerar $S' = (S \setminus \{(x, b_0), (g_0, y)\}) \cup \{(g_0, b_0), (x, y)\}$. Esta es una solución a (G, B) que apareja a g_0 y b_0, y que tiene el mismo número de elementos que S, y por lo tanto también es óptima para (G, B).

Vemos entonces que siempre existe una solución óptima para (G, B) donde g_0 y b_0 están aparejados.

Algoritmo iterativo

Nota: Se asume acá que los multiconjuntos están representados por listas no-decrecientes.

```
def f(G: [int], B: [int]) -> int:
  n = len(G)
  m = len(G)
  i = 0
  j = 0
  res = 0
  while i < n and j < m:
    g = G[i]
    b = B[j]
    if b < g - 1:
        j += 1
    elif g < b - 1:
        i += 1
    else:
        i += 1
```

```
            j += 1
            res += 1
    return res
```

Demostración

Nuestro invariante va a ser que $0 \leq i \leq n, 0 \leq j \leq m$, y que existe un conjunto S, un conjunto T, y una solución óptima S^*, tal que:

- $S \subseteq G[0 \dots i) \times B[0 \dots j)$
- $T \subseteq G[i \dots n) \times B[j \dots m)$
- $S^* = S \sqcup T$, con $\sqcup$ siendo la unión disjunta
- `res` = $|S|$

Esto se puede entender como que S es "extensible" a una solución óptima S^*, usando sólo elementos que vienen no antes que i en G y j en B. Llamemos a esta noción "(i, j)-extensible".

El invariante vale inicialmente

Claramente vale el invariante antes de entrar al ciclo, dado que `i = j = 0`, y `G[0 .. i) = G[0 .. 0) = []`, `B[0 .. j) = B[0 .. 0) = []`. Con ambos `G` y `B` vacíos, no se puede formar ninguna pareja, y luego definimos $S = \emptyset$, cuyo tamaño es exactamente `res = 0`. Asimismo, vemos que *toda* solución global S^*, es una extensión de $\emptyset$, agregándole parejas en $G[i \dots n) \times B[j \dots m) = G[0 \dots n) \times B[0 \dots m) = G \times B$. En particular, definimos $T = S^*$, y tenemos que $S^* = \emptyset \sqcup T$.

Los despiertos habrán notado que ese párrafo asume que *existe* una solución óptima. Un "para todo x en X, vale P(x)" sólo implica "existe x en X tal que P(x)" cuando X no es vacío. El conjunto de todos los emparejamientos posibles no es vacío (por ejemplo, podemos tomar el emparejamiento vacío), y es finito (está incluído en $\mathcal{P}(G \times B)$, el conjunto de partes de $G \times B$). Luego tiene al menos un elemento de tamaño máximo, y luego *existe* al menos una solución óptima.

El invariante es preservado por las iteraciones

Ahora veamos que si vale la guarda, y vale el invariante al entrar al cuerpo del ciclo, vale el invariante al finalizar el cuerpo del ciclo. Como vale la guarda, sabemos que $0 \leq i < n$, y $0 \leq j < m$. Vemos fácilmente que al final del cuerpo del loop sigue valiendo $0 \leq i \leq n, 0 \leq j \leq m$, porque sólo los aumentamos en uno en el cuerpo del loop, y antes eran ≥ 0 y $< n$ y $< m$, respectivamente. Como los índices están en rango, tiene sentido referirse a $g = G[i]$, y $b = B[j]$. Partamos en los tres casos que parte el algoritmo:

- Si $b < g - 1$, entonces b no es compatible con g, puesto que $|g - b| = g - b > 1$. Sabemos que G está ordenado crecientemente, luego para todo $g' \in G[i \dots n)$, $g' \geq g$, y luego $|g' - b| = g' - b \geq g - b > 1$, y por lo tanto b es también incompatible con todos los g' que quedan. Luego, sea S^* la extensión de S que existe por el invariante. Sabemos que S^* se descompone como $S^* = S \sqcup T$, con $S \subseteq G[0 \dots i) \times B[0 \dots j)$, y $T \subseteq G[i \dots n) \times B[j \dots m)$. Como $b = B[j]$, $b \notin B[0 \dots j)$, y luego b no aparece emparejado en S. Vimos arriba que b no es compatible con nadie en $G[i \dots n)$, y luego b no puede estar en T, porque su pareja debería estar en $G[i \dots n)$. Luego, tenemos que
  - $S \subseteq G[0 \dots i) \times B[0 \dots j + 1)$, trivialmente, porque ya estaba incluído en $G[0 \dots i) \times B[0 \dots j)$.
  - $T \subseteq G[i \dots n) \times B[j + 1 \dots m)$, porque sabíamos que estaba incluído en $G[i \dots n) \times B[j \dots m)$, y probamos que b no aparece emparejado con nadie en T, luego podemos restringir la segunda componente de T a $B[j + 1 \dots m)$.
  - $S^* = S \sqcup T$, que ya valía antes.
    Luego, como nuestro código dice:

    ```
    j += 1
    ```

  Estamos manteniendo el invariante, porque cambiamos de j a $j + 1$.
- Si $g < b - 1$, pasa algo exactamente análogo al caso anterior, y como decimos `i += 1`, mantenemos el invariante.
- Si no, tenemos que $g \geq b - 1$, y $b \geq g - 1$. Esto implica que $|g - b| \leq 1$, y por lo tanto son compatibles. Definimos $S' = S \cup \{(g, b)\}$, con $S' \subseteq G[0 \dots i + 1) \times B[0 \dots j + 1)$, porque $S \subseteq G[0 \dots i) \times B[0 \dots j)$, y S' usa $g = G[i]$ y $b = B[j]$. Queremos mostrar que S' es $(i + 1, j + 1)$-extensible, para probar que se mantiene el invariante. Definamos S^*_G como los elementos de G que **no** menciona S^*, y S^*_B análogamente para B. Leér "single girls, single boys". Partimos en casos.
  - Si $g \in S^*_G \land b \in S^*_B$. Esto no puede suceder, porque (g, b) son compatibles, luego $S^* \cup \{(g, b)\}$ sería una solución con tamaño mayor a S^*, con S^* siendo definida como de tamaño máximo.
  - Si $g \in S^*_G$, pero $b \notin S^*_B$. Esto significa que existe un emparejamiento $(x, b) \in S^*$, con $x \neq g$, y que g no aparece emparejada en S^*. Como b no aparece emparejado en S, tiene que ser que $(x, b) \in T$. Consideremos entonces $T' = T \setminus \{(x, b)\}$. Definamos $S^{*'} = S' \sqcup T'$. Tenemos que $|S^{*'}| = |S'| + |T'| = |S| + 1 + |T| - 1 = |S| + |T| = |S^*|$, y luego $S^{*'}$ también es óptima, porque tiene el mismo tamaño que S^*. Notamos que agregar (g, b) es válido, porque g no aparecía emparejada en S^*, y rompimos la pareja de b cuando creamos T'. Notar también que $T' \subseteq G[i + 1 \dots n) \times G[j + 1 \dots m)$, porque g no aparece emparejado en S^* (a fortiori en T, y en T'), y le sacamos a T la mención de b. Por lo tanto, S' es $(i + 1, j + 1)$-extensible.
  - Si $b \in S^*_B$, pero $g \notin S^*_G$. Esto es totalmente análogo al caso anterior.

- Si $b \notin S^*_B, g \notin S^*_G$. Pueden pasar dos cosas, que estén emparejados entre sí, o que cada uno esté emparejado con alguien más.
  - Si $(g, b) \in S^*$. Ya sabíamos que $(g, b) \notin S$, y luego como $S^* = S \sqcup T$, tenemos que $(g, b) \in T$. Definiendo $T' = T \setminus \{(g, b)\}$, vemos que $S^* = S' \sqcup T'$, y T' no usa ni $g = G[i]$ ni $b = B[j]$, y luego $T' \subseteq G[i + 1 \dots n) \times B[j + 1 \dots m)$. Luego S' es $(i + 1, j + 1)$-extensible.
  - Si no, entonces existen (g, x) y (y, b) en S^*. Como $g = G[i]$ y $b = B[j]$ no pueden estar en S porque $S \subseteq G[0 \dots i) \times B[0 \dots j)$, esas parejas tienen que estar en T. Notamos que $x \geq b$ y que $y \geq g$, porque x e y vienen después que b y g en B y G respectivamente, y B y G están ordenados de forma no-decreciente. Vamos a probar que x e y son compatibles.
    - Si $g = b$, entonces $y \geq g = b$. Como $(y, g) \in T$, son compatibles, y luego $y = g$, o $y = g + 1$. Por el mismo motivo, $x \geq b = g$, y como $(g, x) \in T$, son compatibles, luego $x = g$, o $x = g + 1$. Luego, $\{x, y\} \subseteq \{g, g + 1\}$, luego están a distancia a lo sumo 1 del otro, y luego son compatibles.
    - Si $g > b$, entonces $g = b + 1$, porque g y b son compatibles. Tenemos $y \geq g > b$, y luego $y = b + 1$, porque (y, b) son compatibles, estando emparejados en T. Como x es compatible con $g = b + 1$, estando emparejados en T, tenemos que x es compatible con y.
    - Si $b > g$, tenemos algo análogo al caso anterior.

    En todos los casos, x e y son compatibles. Luego, podemos considerar $T' = T \cup \{(x, y)\} \setminus \{(g, x), (y, b)\}$, y definimos $S^{*'} = S' \sqcup T'$, con $|S^{*'}| = |S'| + |T'| = |S| + 1 + |T| + 1 - 2 = |S| + |T| = |S^*|$, y luego $S^{*'}$ también es una solución óptima. Como T sólo usa elementos que vienen *después* de $(g, b) = (G[i], B[j])$, tenemos que $T' \subseteq G[i + 1 \dots n) \times B[j + 1 \dots m)$, y luego S' es $(i + 1, j + 1)$-extensible.

En todos los casos, tenemos que existe S', un conjunto $(i + 1, j + 1)$-extensible, con $|S'| = |S| + 1$. Recordemos que `res`, antes de entrar al cuerpo de la iteración, era igual a $|S|$, por el teorema del invariante. Luego, como nuestro código dice:

```
i += 1
j += 1
res += 1
```

`res` es ahora $|S'|$, y el invariante es preservado.

El ciclo termina

El cuerpo del ciclo siempre aumenta o i o j, empezando ambos en cero. Luego, si llegásemos a $n + m - 1$ iteraciones, tendríamos $i + j = n + m - 1$. Esto haría que la guarda no se cumpla y el ciclo termine - veamos por qué.

- Si $i \geq n$, la guarda no se cumple.
- Si $i < n$, y sabemos que $n + m - 1 = i + j$, entonces $n + m + 1 < n + j$, y restando n de cada lado obtenemos $m + 1 < j$, o lo que es lo mismo, $j \geq m$, con lo cual la guarda no se cumple.

El invariante es suficiente para demostrar lo que queremos

Al final del loop, vale la negación de la guarda, y el invariante. La negación de la guarda es que o bien $i \geq n$, o bien $j \geq m$. Como vale el invariante, tenemos que $i \leq n$ y $j \leq m$. Luego, sabemos que o bien $i = n$, o bien $j = m$ (pueden pasar las dos juntas). Partimos en casos:

- Si $i = n$, entonces S es (n, j)-extensible para algún j. Esto significa que existe una solución óptima S^*, y un conjunto $T \subseteq G[n \dots n) \times B[j \dots m)$, tal que $S^* = S \sqcup T$. Pero $G[n \dots n) = \emptyset$, y luego $T = \emptyset$, y luego $S = S^*$. Luego, S es una solución óptima.
- Si $j = m$, pasa algo análogo con $B[m \dots m) = \emptyset$.

Luego, como sabemos que `res` $= |S|$, y devolvemos `res`:

```
return res
```

Nuestro algoritmo devuelve el tamaño de una solución óptima, y luego es correcto.