# Introducción

## Algoritmos voraces (greedy, glotones, ambiciosos)

Consisten en intentar realizar en cada momento la mejor decisión para optimizar una función. En cada problema puede que no haya una única forma de hacer «lo mejor» en cada momento, así que a veces tenemos que elegir de entre varias formas de hacer las cosas cuál nos conviene.

**Problema 11.** Se tienen números enteros positivos distintos $a_1,a_2,\ldots,a_n$. Se quiere encontrar una permutación de ellos $(b_1,\ldots,b_n)$ que haga la siguiente suma

$$b_1+2b_2+3b_3+\ldots+nb_n$$

máxima. Resolver el problema en general, y luego ejecutarlo para:

```
50, 76, 72, 52, 70, 74, 84, 16, 43, 29, 31, 77, 22, 92, 46, 38, 91, 42, 48, 98, 87, 83, 6, 44, 7, 36, 80, 11, 10, 82, 67, 90, 18, 27, 37, 86, 33, 64, 26, 9
```

**Problema 12.** Se tienen $2n$ números enteros $a_1,\ldots,a_{2n}$. Hay que encontrar cuáles $n$ de ellos tienen la suma menor. Resolver el problema en general y luego ejecutarlo para:

```
34, -51, -67, -42, -72, 27, -12, 55, -69, 47, 89, -54, 92, 8, 77, 82, -52, -48, -61, -90, 95, -49, 32, 18, 40, 96, 65, 44, 81, 24, 38, -4, -14, -97, -20, 63, 86, -98, -19, -8
```

**Problema 13.** Se tienen $n$ bloques de longitudes enteras positivas $a_1,a_2,\ldots a_n$. Se quiere acomodarlos en forma de pirámide, de modo que se obtenga la mayor altura de pirámide posible. ¿Cuál es la altura que se obtiene? Resolver en general y luego ejecutarlo para:

```
355, 572, 532, 788, 82, 433, 225, 183, 499, 374, 734, 157, 121, 53, 486, 954, 792, 97, 684, 933, 51, 868, 485, 907, 292, 349, 804, 83, 214, 704, 200, 236, 404, 276, 778, 958, 105, 500, 334, 448, 836, 890, 537, 167, 31, 398, 313, 175, 930, 342
```

**Problema 14.** Se tiene la siguiente pirámide de números. Se comienza desde arriba y en cada paso se puede ir al número inferior, o bien al número inferior a la derecha, hasta que se llega a la fila de hasta abajo.

```
[7116] [2675 8480] [4356 6635 9218] [7772 5841 2088 5391] [4167 7727 3335 8931 4326] [3838 2437 3248 9461 7437 6341] [2117 5132 4482 1541 5000 8745 3782] [6343 8811 6752 7523 2144 7855 9078 6472] [1717 1943 8715 4741 3475 9590 7878 6717 4730] [1683 1507 6681 1496 9129 7566 8605 3997 8064 7240] [4668 9965 6544 1470 6365 8740 8637 6981 5703 9535 1509] [7196 9964 7924 6509 8146 6239 4603 9728 1090 2069 8021 3227]
```

Por ejemplo, un posible camino es:

```
7116,2675,6635,2088,8931,9461,1541,2144,3475,9129,8740,4603
```

Hay que encontrar el camino que vaya de hasta arriba hasta abajo cuya suma de valores sea máxima.

Comenzamos con el siguiente problema.

**Problema 11.** Se tienen números enteros positivos distintos $a_1,a_2,\ldots,a_n$. Se quiere encontrar una permutación de ellos $(b_1,\ldots,b_n)$ que haga la siguiente suma

$$b_1+2b_2+3b_3+\ldots+nb_n$$

máxima. Resolver el problema en general, y luego ejecutarlo para:

```
50, 76, 72, 52, 70, 74, 84, 16, 43, 29, 31, 77, 22, 92, 46, 38, 91, 42, 48, 98, 87, 83, 6, 44, 7, 36, 80, 11, 10, 82, 67, 90,
18, 27, 37, 86, 33, 64, 26, 9
```

En este problema, el espacio de estados son todas posibles permutaciones de $a_1,\ldots,a_n$, que son $n!$. Por lo tanto, si queremos resolver el problema por exploración exhaustiva, tendríamos que verificar $\Theta(n!)$. Incluso cuando $n$ no es tan grande, esto es demasiado tiempo.

Para resolver el problema, vamos a realizar un algoritmo voraz eligiendo los $a_i$ más grandes para los coeficientes más grandes. Es decir, el máximo de los $a_i$ va a acompañar al coeficiente $n$, el siguiente al coeficiente $n-1$ y así sucesivamente.

Con esta idea, se puede proponer el siguiente algoritmo:

- Poner los $a_i$ en una lista de Python - Tiempo $O(n)$
- Tomar el $a_i$ más grande y declarlo como $b_n$ - Tiempo $O(n)$.
- Eliminar ese $a_i$ de la lista. - Tiempo $O(n)$
- Repetir para ir definiendo $b_{n-1},\ldots,b_1$. En total, tiempo $O(n^2)$.

Este algoritmo toma $O(n^2)$ tiempo en total. Se puede mejorar a tiempo $O(n\log n)$ si primero ordenamos la lista y luego de ahí ya vamos tomando los más grandes (que simplemente es ordenar).

```
A=[50, 76, 72, 52, 70, 74, 84, 16, 43, 29, 31, 77, 22, 92, 46, 38, 91, 42, 48, 98, 87, 83, 6, 44, 7, 36, 80, 11, 10, 82, 67, 
A.sort()
print(A)

print(sum(i*a for i,a in enumerate(A)))
```

```
[6, 7, 9, 10, 11, 16, 18, 22, 26, 27, 29, 31, 33, 36, 37, 38, 42, 43, 44, 46, 48, 50, 52, 64, 67, 70, 72, 74, 76, 77, 80, 82, 
53023
```

Vamos a mostrar que la permutación de $a_1,\ldots,a_n$ que maximiza $a_1+2a_2+\ldots+na_n$ es precisamente en la cual tenemos los $a_i$'s ordenados.

Pensemos que tenemos una permutación $b_1,b_2,\ldots,b_n$ en la cual los $b_i$ no están en orden creciente. Como los $b_i$ no están orden creciente, debe existir una pareja de índices $i$ y $j$ tales que $i<j$ y $b_i>b_j$. Entonces la suma se ve así:

$$S=\ldots + i\, b_i + \ldots + j\, b_j + \ldots.$$

Al pasar $b_i$ a la posición $j$ y $b_j$ a la posición $b_i$, la nueva suma es:

$$S-i\, b_i-j\, b_j+jb_i+ib_j=S+(j-i)(b_i-b_j)>S,$$

por lo cual dicha permutación que habíamos tomado no puede ser la mayor.

**Problema 12.** Se tienen $2n$ números enteros $a_1,\ldots,a_{2n}$. Hay que encontrar cuáles $n$ de ellos tienen la suma menor. Resolver el problema en general y luego ejecutarlo para:

```
34, -51, -67, -42, -72, 27, -12, 55, -69, 47, 89, -54, 92, 8, 77, 82, -52, -48, -61, -90, 95, -49, 32, 18, 40, 96, 65, 44, 81,
24, 38, -4, -14, -97, -20, 63, 86, -98, -19, -8
```

Una forma de resolver el problema es con exploración exhaustiva sobre el espacio de estados de subconjuntos de $n$ elementos de un conjunto con $2n$ elementos. Para cada uno de estos subconjuntos hacemos su suma y buscamos de entre ellas la menor. Con este algoritmo tendríamos que verificar $\binom{2n}{n}$ casos. Estos son $\Omega(2^{2n})$ casos, que como es exponencial tampoco nos conviene mucho.

Una mucho mejor forma de resolverlo es tomando de entre la lista los \(n\) más pequeños. Estos definitivamente tienen la suma más pequeña. ¿Cuánto tiempo tarda? Si vamos encontrando los más pequeños uno por uno leyendo la lista \(n\) veces, toma tiempo \(O(n^2)\), pero una mejor forma de hacerlo es ordenándolos y tomando los \(n\) más pequeños.

```
A=[34, -51, -67, -42, -72, 27, -12, 55, -69, 47, 89, -54, 92, 8, 77, 82, -52, -48, -61, -90, 95, -49, 32, 18, 40, 96, 65, 44,
A.sort()
print(A)
print(len(A))
print(sum(j for j in A[:20]))
```

```
[-98, -97, -90, -72, -69, -67, -61, -54, -52, -51, -49, -48, -42, -20, -19, -14, -12, -8, -4, 8, 18, 24, 27, 32, 34, 38, 40, 4
40
-919
```

**Problema 13.** Se tienen \(n\) bloques de longitudes enteras positivas \(a_1,a_2,\ldots a_n\). Se quiere acomodarlos en forma de pirámide, de modo que se obtenga la mayor altura de pirámide posible. Aquí una pirámide es acomodar los bloques en niveles de modo que para cada nivel, tenga más bloques que el nivel de arriba y una suma total mayor que el nivel de arriba. ¿Cuál es la altura que se obtiene? Resolver en general y luego ejecutarlo para:

```
355, 572, 532, 788, 82, 433, 225, 183, 499, 374, 734, 157, 121, 53, 486, 954, 792, 97, 684, 933, 51, 868, 485, 907, 292, 349, 804,
83, 214, 704, 200, 236, 404, 276, 778, 958, 105, 500, 334, 448, 836, 890, 537, 167, 31, 398, 313, 175, 930, 342
```

Una exploración exhaustiva permutando los bloques a acomodándolos seguro que resuelve el problema, pero el tiempo de ejecución será muy elevado.

Para resolver este problema, es mejor hacer lo siguiente:

- Ordenamos los bloques de menor a mayor.
- Ponemos el mayor hasta arriba, los siguientes dos en el nivel inferior, los siguientes tres en el inferior y así sucesivamente.
- La altura máxima que podamos completar totalmente es la que se obtiene con este procedimiento.

```
lista=[355, 572, 532, 788, 82, 433, 225, 183, 499, 374, 734, 157, 121, 53, 486, 954, 792, 97, 684, 933, 51, 868, 485, 907, 29
lista.sort()

j=1
suma=0
piramide=[]
suma=1
while suma<len(lista):
    fila=lista[suma-j:suma]
    piramide.append(fila)
    j+=1
    suma+=j

print("La altura máxima es {} y se puede alcanzar con la siguiente pirámide:".format(len(piramide)))
for fila in piramide:
    print(fila, "Esta fila tiene suma {}".format(sum(fila)))
```

```
La altura máxima es 9 y se puede alcanzar con la siguiente pirámide:
[31] Esta fila tiene suma 31
[51, 53] Esta fila tiene suma 104
[82, 83, 97] Esta fila tiene suma 262
[105, 121, 157, 167] Esta fila tiene suma 550
[175, 183, 200, 214, 225] Esta fila tiene suma 997
[236, 276, 292, 313, 334, 342] Esta fila tiene suma 1793
[349, 355, 374, 398, 404, 433, 448] Esta fila tiene suma 2761
[485, 486, 499, 500, 532, 537, 572, 684] Esta fila tiene suma 4295
[704, 734, 778, 788, 792, 804, 836, 868, 890] Esta fila tiene suma 7194
```

El análisis de tiempo y la correctitud quedan como tarea moral.

**Problema 14.** Se tiene la siguiente pirámide de números. Se comienza desde arriba y en cada paso se puede ir al número inferior, o bien al número inferior a la derecha, hasta que se llega a la fila de hasta abajo.

```
[7116] [2675 8480] [4356 6635 9218] [7772 5841 2088 5391] [4167 7727 3335 8931 4326] [3838 2437 3248 9461 7437 6341] [2117 5132
```

```
[7116] [2675 8480] [4356 6635 9218] [7772 5841 2088 5391] [4167 7727 3335 8931 4326] [3838 2437 3248 9461 7437 6341] [2117 5132
4482 1541 5000 8745 3782] [6343 8811 6752 7523 2144 7855 9078 6472] [1717 1943 8715 4741 3475 9590 7878 6717 4730] [1683 1507 6681
1496 9129 7566 8605 3997 8064 7240] [4668 9965 6544 1470 6365 8740 8637 6981 5703 9535 1509] [7196 9964 7924 6509 8146 6239 4603
9728 1090 2069 8021 3227]
```

Por ejemplo, un posible camino es:

```
7116,2675,6635,2088,8931,9461,1541,2144,3475,9129,8740,4603
```

Hay que encontrar el camino que vaya de hasta arriba hasta abajo cuya suma de valores sea máxima.

Al ir de arriba hacia abajo tenemos dos opciones en cada paso, así que tenemos $2^{11}$ posibles caminos. Esto no son demasiadas posibilidades y se pueden hacer fácilmente con una búsqueda exhaustiva. Sin embargo, este es un algoritmo que toma tiempo exponencial en la cantidad de renglones y no sería factible para cuando tenemos más renglones.

Una forma de mejorar este procedimiento es usando un algoritmo voraz. Pero hay que ser muy cuidadosos. Si hacemos un algoritmo voraz que de arriba hacia abajo siempre seleccione la mejor opción, es posible que falle, pues puede que por tomar la dirección del número más grande en un paso local, acabemos descartando la posibilidad de tener un número muy grande más adelante. Por ejemplo, siguiendo esta heurística, en la siguiente pirámide iríamos en el primer paso a la derecha, hacia el $54$, pero ya nunca podríamos tomar el $99$ de la esquina inferior izquierda que nos ayuda a maximizar la suma:

```
[10] [36 54] [99 20 24]
```

Sin embargo, eso no quiere decir que *ningún* algoritmo voraz funcione. A veces simplemente no hemos elegido el correcto. En este problema conviene mucho más realizar un algoritmo voraz de abajo hacia arriba. Para cada una de las entradas de la penúltima fila, ¿qué es lo mejor que puedo hacer después? Por ejemplo, en la pirámide chiquita de ejemplo, si ya llegamos al $36$ entonces lo mejor que puedo acumular a partir de ahí es $36+99=135$. Si ya llegamos al $54$, entonces lo mejor que puedo acumular desde ahí es $54+24=78$. De esta forma, podemos reemplazar el problema a ver cuál es el mejor camino en la pirámide

```
[10] [135 78]
```

A partir de aquí es claro lo que hay que hacer: del $10$ ir al $135$ y de ahí hacer lo mejor posible a partir de ahí, que ya sabemos que es seguir el camino que de $135$. Esta misma idea se puede generalizar para la pirámide con $12$ filas y para una cantidad abritraria $n$ de filas:

- En el penúltimo renglón escribimos lo mejor que podemos hacer.
- De ahí, podemos decir qué es lo mejor que podemos hacer en el antepenúltimo.
- Seguimos así sucesivamente hasta el primero.
- Para recuperar el mejor camino, simplemente en cada paso hacemos lo mejor, pero considerado de esta forma.

Veamos la implementación

```
# Esta es una función auxiliar para leer la
# cadena de caracteres que tenemos y pasarla a una
# lista de listas de Python con la que podamos trabajar
# mejor.

entrada='''[7116]
[2675 8480]
[4356 6635 9218]
[7772 5841 2088 5391]
[4167 7727 3335 8931 4326]
[3838 2437 3248 9461 7437 6341]
[2117 5132 4482 1541 5000 8745 3782]
[6343 8811 6752 7523 2144 7855 9078 6472]
[1717 1943 8715 4741 3475 9590 7878 6717 4730]
[1683 1507 6681 1496 9129 7566 8605 3997 8064 7240]
[4668 9965 6544 1470 6365 8740 8637 6981 5703 9535 1509]
[7196 9964 7924 6509 8146 6239 4603 9728 1090 2069 8021 3227]'''

lineas=entrada.split('\n')
lista_buena=[]
for j in lineas:
    lista_buena.append([int(k) for k in j[1:-1].split(' ')])

for linea in lista_buena:
```

```
for linea in lista_buena:
    print(linea)
```

```
[7116]
[2675, 8480]
[4356, 6635, 9218]
[7772, 5841, 2088, 5391]
[4167, 7727, 3335, 8931, 4326]
[3838, 2437, 3248, 9461, 7437, 6341]
[2117, 5132, 4482, 1541, 5000, 8745, 3782]
[6343, 8811, 6752, 7523, 2144, 7855, 9078, 6472]
[1717, 1943, 8715, 4741, 3475, 9590, 7878, 6717, 4730]
[1683, 1507, 6681, 1496, 9129, 7566, 8605, 3997, 8064, 7240]
[4668, 9965, 6544, 1470, 6365, 8740, 8637, 6981, 5703, 9535, 1509]
[7196, 9964, 7924, 6509, 8146, 6239, 4603, 9728, 1090, 2069, 8021, 3227]
```

```
# Ahora, lo que haremos es comenzar desde la linea 11 (índice 10) e iremos cambiando las entradas,
# de acuerdo a lo mejor que podemos hacer después. Ya hechos esos cambios, vamos a la línea 10,
# luego a la 9, y así sucesivamente.

for k in range(10,-1,-1):
    for j in range(k+1):
        lista_buena[k][j]=lista_buena[k][j]+max(lista_buena[k+1][j],lista_buena[k+1][j+1])

for linea in lista_buena:
    print(linea)
```

```
[99733]
[80926, 92617]
[72408, 78251, 84137]
[61743, 68052, 71616, 74919]
[51812, 53971, 62211, 69528, 64923]
[47645, 46244, 44659, 58876, 60597, 59501]
[40792, 43807, 41411, 40245, 49415, 53160, 47708]
[29722, 38675, 36616, 36929, 38704, 44415, 43926, 38809]
[23329, 23379, 29864, 28849, 29406, 36560, 34848, 32337, 30350]
[21612, 21436, 21149, 16007, 24108, 25931, 26970, 20706, 25620, 24796]
[14632, 19929, 14468, 9616, 14511, 14979, 18365, 16709, 7772, 17556, 9530]
[7196, 9964, 7924, 6509, 8146, 6239, 4603, 9728, 1090, 2069, 8021, 3227]
```

Con esto tenemos que el mejor camino tiene suma \(99733\). Observa que el algoritmo corre en milisegundos. Aún no hemos dicho cómo reportar el camino, sino solamente el máximo que obtenemos. Esta parte queda como tarea moral. El análisis de tiempo y la correctitud también quedan como tarea moral.

## Sección 1

## Sección 2

## Sección 3

## Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. Problema
2. Problema
3. Problema
4. Problema
5. Problema

Por Leonardo Ignacio Martínez Sandoval