Memoria: Práctica 1 DAA

Nicolás Muñiz Rodríguez y Carla Lorenzo Rama

1. Introducción:

En esta práctica analizamos de 2 formas distintas el rendimiento de encontrar todos los pares de números dentro de una lista de enteros que sumen una cantidad objetivo.
Por ejemplo, si tenemos la lista L = [1, 2, 3, 4, 5, 6] y el objetivo x = 7, hay que conseguir que la función nos devuelva los pares que suman x en L: (1,6), (2,5), (3,4).

1.1. Forma 1:

La primera forma se realizará con listas, siguiendo el siguiente algoritmo:

```
function SearchPairsWithLists (L[1..n], target):
pairs := [ ]; #a list
for i := 1 to n:
    for j := i+1 to n:
        if L[i]+L[j] = target:
            if (L[i], L[j]) not in pairs:
                append (L[i], L[j]) to pairs;
return pairs;
```

Se trata de una función llamada SearchPairsWithLists, a la que se le pasan como parámetros una lista L y un target, que es el número objetivo. Lo primero que se hace en la función es crear una lista vacía llamada “pairs”, y a continuación se realiza un bucle, que recorre la lista 2 veces de forma simultánea sumando cada uno de los números de la lista con el siguiente y comprobando si el resultado es igual al target. Además, para que no aparezcan repetidos los pares, que se van almacenando en la lista vacía “pairs”, se comprueba que el nuevo par encontrado no estea ya añadido a la lista.

1.2. Forma 2:

Mientras que la segunda seguirá un algoritmo con conjuntos:

```
function SearchPairsWithSets (L[1..n], target):
pairs := {}; # create an empty set
seen := {}; # create an empty set to store the numbers that have been read
for i:=1 to n:
    c := target - L[i];
    if c in seen:
        add (min(L[i],c), max(L[i], c)) to the pairs set;
    add L[i] to seen;
return pairs;
```

A la función llamada SearchPairsWithSets, a la que también se le pasan como parámetros una lista L y un target, crea un conjunto inicial vacío llamado “pairs”, y otro llamado “seen”. Después realiza un bucle, que recorre la lista que se le pasa como parámetro. Dentro del bucle se crea la variable “c”, que es el resultado de restarle al target el elemento i-ésimo de la lista. Se comprueba si c está en el conjunto “seen”. Si es así, se añade una tupla con los pares que suman el target al conjunto “pairs”. En caso contrario se ejecuta la siguiente instrucción del bucle, que añade el elemento i-ésimo con el que se está trabajando en ese momento en el bucle al conjunto “seen”. Una vez terminado el bucle, la función devuelve el conjunto “pairs”, que contiene todos los pares de la lista que suman el valor objertivo(target).

En la práctica, además de la implementación de dichos algoritmos, se valida si funcionan correctamente con la función test(), que incluye el statement “assert”, comprobando que la salida es correcta para varios ejemplos concretos. Además, para unos tamaños de entrada dados, se calcula los tiempos de ejecución para ambos algoritmos, utilizando diversas funciones de medición de tiempo. Por último, se analizan los resultados obtenidos, que se recopilan en una tabla , realizando una comprobación empírica utilizando una cota ligeramente subestimada y otra sobreestimada para cada algoritmo.

2. Especificaciones técnicas:

Modelo	DELL Inspiron 13 7386
SO	Debian GNU/Linux 12 (bookworm), 64 bits, GNOME 43.9
CPU	Intel Core i7-8565U x 8
Chipset	En el procesador
GPU	Mesa Intel UHD Graphics 620 (WHL GT2)
RAM	16,0 GB
Disco	512,1 GB

3. Implementación:

3.1. Algoritmo de listas:

Creamos una función llamada “SearchPairsWithList” con el comando “def”, y se le pasan los parámetros L (que es una lista) y target (que se trata del número objetivo, que es un entero) entre paréntesis. Además, se indica el tipo de estructura que tiene que devolver la función, en este caso, una lista, que contendrá los pares de números de la lista “L” que suman el número objetivo “target” si es que existen. Se crea una lista vacía llamada “pairs”, y a continuación se realiza un bucle for que recorre la lista desde el primer elemento hasta el último, para lo que se usa la función “len()” para calcular la longitud de la lista. Justo a continuación se introduce una condición con el comando “if", para comprobar que la posición de la lista del primer bucle no coincide con la de la segunda. Entonces se comprueba si la suma de dichos elementos de la lista (L[i] + L[j]) es igual (==) al valor del target y si el par no está ya incluido en la lista vacía “pairs” que se creó previamente. Si se cumplen estas 2 condiciones, valoradas con el comando “if”, entonces se añade el par en forma de tupla a la lista con la función “pairs.append()”. Para finalizar, ya fuera del bucle, se devuelve la lista “pairs” con las tuplas con los pares en caso de que existan o vacía en caso contrario. Esto se hace con el comando “return”.

3.2. Algoritmo de conjuntos:

Creamos la función “SearchPairsWithSets” con los mismos parámetros que la anterior, una lista “L” y un número objetivo “target”, e indicamos el tipo de estructura a devolver por la función, en este caso un conjunto (set). Se crean dos conjuntos vacíos llamados “pairs” y “seen”, con la función set(). El siguiente paso es realizar un bucle for que recorra todos los elementos de la lista “L”. Dentro de este se crea la variable “c”, que será el resultado de restarle al “target” el elemento en cuestión de la lista. Se comprueba con el comando “if” si la variable “c” está contenida en el conjunto “seen”. De ser así se le añade al conjunto “pairs” con la función pairs.add() una tupla con los pares que suman el target. A continucióni se añade al conjunto “seen” el elemento de la lista sobre el que se encuentra el bucle. Por último se devuelve, ya fuera del bucle, el conjunto “pairs”, con las tuplas con los pares de ser el caso o por lo contrario, vacío,

4. Pruebas:

Para probar los algoritmos, utilizamos los siguientes casos de prueba:

	Lista	Objetivo	Resultado esperado
Caso 1	[1,2,3,4,5,6]	7	{(1,6), (2,5), (3,4)}
Caso 2	[1,2,3, … ,10]	15	{(5,10), (6,9), (7,8)}
Caso 3	[1,3,5]	20	{ }
Caso 4	[ ]	3	{ }

Y efectivamente, los algoritmos implementados devolvían las soluciones correctas.

5. Análisis de resultados:

Los resultados probados con muestras más grandes dieron lugar a las siguientes gráficas de tiempos:

・Tabla para las listas:

n	Averaged	Time	O(nlog(n))	$O(n^2)$	$O(n^{2.2})$
50	True	100491.7760	513.7586	40.1967	18.3822
100	True	429220.0960	932.0396	42.9220	17.0876
200	False	1763328.0000	1664.0453	44.0832	15.2780
400	False	7385856.0000	3081.8241	46.1616	13.9274
800	False	30084352.0000	5625.6730	47.0068	12.3465
1600	False	120638464.0000	10219.7755	47.1244	10.7751
2000	False	190548736.0000	12534.6126	47.6372	10.4169
3200	False	485132288.0000	18783.9926	47.3762	9.4304
4000	False	775899136.0000	23387.2225	48.4937	9.2315
5000	False	1192124160.0000	27993.3573	47.6850	8.6814
6000	False	1729038592.0000	33125.1923	48.0288	8.4309
6500	False	2027412736.0000	35526.8080	47.9861	8.2896
8000	False	3229641984.0000	44920.0408	50.4632	8.3629
9000	False	4183443712.0000	51051.9613	51.6475	8.3599
10000	False	5053094144.0000	54863.2726	50.5309	8.0086

Lo que muestra la tabla es que, para el algoritmo de listas, la cota ajustada es n^2, apoyándonos en el hecho de utilizar nlog(n) como cota inferior (el cociente tiempo/f(n) tiende a infinito) y $n^{2.2}$ como cota superior (el cociente parece tender a números menores). Por tanto, el algoritmo es $O(n^2)$ y la constante a la que parece que se aproxima está alrededor de 47.

- Tabla para los conjuntos:

n	Averaged	Time	O(logn)	O(n)	O(nlogn)
50	True	5002.4960	1278.7491	100.0499	25.5750
100	True	11156.7360	2422.6544	111.5674	24.2265
200	True	21492.9920	4056.5694	107.4650	20.2828
400	True	45692.1600	7626.2089	114.2304	19.0655
800	True	83557.8880	12500.0361	104.4474	15.6250
1600	True	165816.5760	22475.1958	103.6354	14.0470
2000	True	214609.1520	28234.6936	107.3046	14.1173
3200	True	346896.3840	42981.0953	108.4051	13.4316
4000	True	418301.4400	50433.9205	104.5754	12.6085
5000	False	625408.0000	73428.8851	125.0816	14.6858
6000	False	690944.0000	79423.2805	115.1573	13.2372
6500	False	763136.0000	86921.9210	117.4055	13.3726
8000	False	672000.0000	74773.0370	84.0000	9.3466
9000	False	827904.0000	90928.7020	91.9893	10.1032
10000	False	933120.0000	101312.2167	93.3120	10.1312

Esta tabla demuestra que, para el algoritmo de conjuntos, la cota ajustada es n, ya que al utilizar logn como cota inferior (el cociente tiempo/f(n) tiende a infinito) y nlog(n) como cota superior (el cociente parece aproximarse a números menores). Por tanto, el algoritmo es O(n) y la constante a la que parece tender está entre 95 y 110.

6. Conclusiones:

En resúmen, el algoritmo que trabaja con listas se comporta como un algoritmo $0(n^2)$ mientras que el que utiliza sets se comporta como uno O(n). Esto se ve con la cantidad de bucles del tamaño del vector (n) que contiene cada algoritmo (2 en las listas, uno en los sets).