Algoritmos Voraces: Ejercicio 1

Se tienen *n* números naturales, siendo *n* una cantidad par, que tienen que juntarse formando parejas de dos números cada una. A continuación, de cada pareja se obtiene la suma de sus dos componentes, y de todos estos resultados se toma el máximo.

Diseñar un algoritmo voraz que cree las parejas de manera que el valor máximo de las sumas de los números de cada pareja sea lo más pequeño posible, demostrando que la función de selección de candidatos usada proporciona una solución óptima.

Ejemplo: suponiendo que los datos se encuentran en el vector siguiente

5	8	1	4	7	9

vamos a ver un par de formas de resolver el problema (no necesariamente la óptima):

- Seleccionamos como pareja los elementos consecutivos

  De esta forma conseguimos las parejas (5, 8), (1, 4) y (7, 9); entonces, al sumar las componentes tenemos los valores 15, 5 y 16, por lo que el resultado final es 16.

- Seleccionamos como pareja los elementos opuestos en el vector

  Ahora tenemos las parejas (5, 9), (8, 7) y (1, 4); sumando conseguimos 14, 15 y 5, por lo que el resultado final es 15 (mejor que antes).

¿Habrá una resultado mejor para este problema? ¿Puede generalizarse un método que nos proporcione un algoritmo voraz correcto para cualquier cantidad de datos, y que además sea independiente del valor de los mismos?

Algoritmos Voraces: Ejercicio 1

Estrategia de Resolución

Distintas estrategias

- Comparando la suma de valores Consecutivos dos a dos.

5	8	1	4	7	9

Coste = Max {(5+8), (1+4), (7+9)} = {15, 5, 16} = {16}

- Comparando la suma de Elementos Opuestos del Vector dos a dos.

5	8	1	4	7	9

Coste = Max {(5+9), (8+7), (1+4)} = {14, 15, 5} = {15}

- Ordenando el vector. Comparando la suma valores Consecutivos dos a dos.

1	4	5	7	8	9

Coste = Max {(1+4), (5+7), (8+9)} = {5, 12, 17} = {17}

- Ordenando el vector. Comparando la suma de Elementos Opuestos dos a dos.

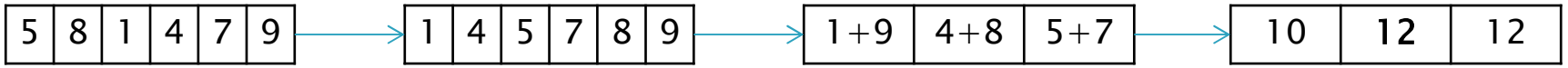
1	4	5	7	8	9

Coste = Max {(1+9), (4+8), (5+7)} = {10, 12, 12} = {12}

Este método es general e independiente a los datos del vector y hace que sea correcto

Algoritmos Voraces: Ejercicio 1

Estrategias



- **¿Puede generalizarse un método que nos proporcione un algoritmo voraz correcto para cualquier cantidad de datos, y que además sea independiente del valor de los mismos?**

Este método es general para la cantidad de datos e independiente de sus valores: podrían ser datos enteros o reales, y el algoritmo seguiría siendo correcto

Algoritmos Voraces: Ejercicio 1

Componentes Voraces en el Algoritmo de Ejercicio 1

1. **Conjunto de Candidatos**: Todos los datos.
2. **Conjunto de Decisiones**: Comparar datos opuestos.
3. **Función que determina la solución del problema**: Máximo de los datos opuestos comparados .
4. **Completable**: Si, con esta estrategia siempre hay que recorrer todo el vector.
5. **Función Selección**: Es la función que hace que el algoritmo voraz implementado es correcto, funcione o no funcione. La que ordena el vector y coge el mayor de las sumas parciales de los datos opuestos comparados.
6. **Función Objetivo**: Es la que devuelve el dato de mayor cantidad.

Algoritmo Ejercicio 1 es Correcto

Algoritmos Voraces: Ejercicio 1

```
const n = ...
tipos vector= array[1... n] de entero
tipos parejas= array[1... n%2] de entero
fun AgruparParejas (E/S Entrada: vector ; S Salida: parejas) dev
                                        Maximo: entero
var i: entero
    OrdenarCreciente ( Entrada )
    Maximo= -1
    desde i=1  hasta  n%2 hacer
            Salida [ i ] = Entrada  [ i ] + Entrada  [ n - i + 1]
            si Salida  [ i ] > Maximo entonces
                    Maximo = Salida [ i ] //Función Objetivo
            fsi
    fdesde
    devolver Maximo
ffun
```

Función Selección