Algoritmia y Complejidad
Entrega ejercicios 2

Grado en Ingeniería Informática

Raúl López Llana
Grupo Laboratorio: Miércoles 12.00 – 14.00 h.

Índice

Problema 2

Enunciado:
Se tiene que almacenar un conjunto de n ficheros en una cinta magnética (soporte de almacenamiento de recorrido secuencial), teniendo cada fichero una longitud conocida l 1 , l 2 , ..., l n . Para simplificar el problema, puede suponerse que la velocidad de lectura es constante, así como la densidad de información en la cinta.
Se conoce de antemano la tasa de utilización de cada fichero almacenado, es decir, se sabe la cantidad de peticiones pi correspondiente al fichero i que se van a realizar. Por tanto, el total de peticiones al soporte será la cantidad ***P*= ∑ desde i=1 hasta n Pi**. Tras la petición de un fichero, al ser encontrado la cinta es automáticamente rebobinada hasta el comienzo de la misma.
El objetivo es decidir el orden en que los n ficheros deben ser almacenados para que se minimice el tiempo medio de carga, creando un algoritmo voraz correcto.

A partir de un ejemplo, se procederá a la explicación.

Suponiendo 3 ficheros dentro de la cinta con longitudes l1=3, l2=7 y l3=5 y que cada una de estas cintas tienen un numero de llamadas predefinidas ll1=1, ll2=2 y ll3=3, tenemos que ver la forma más óptima de almacenarlas.

Existen distintas formas de almacenarlas entre las cuales podemos decir:

Ordenado según llegada

peso	3	7	5
llamadas	1	2	3

Con un coste total de: (3*1) +((3+7)*2)+((3+7+5)*3)=68

Ordenado mayor a menor según peso

peso	7	5	3
llamadas	2	3	1

Con un coste total de:(7*2)+((7+5)*3)+((7+5+3)*1)=65

Ordenado menor a mayor según peso

peso	3	5	7
llamadas	1	3	2

Con un coste total de:(3*1)+((3+5)*3)+((3+5+7)*2)=57

Ordenado de menos a mayor según peso/llamadas

peso	5	3	7
llamadas	3	1	2
peso/llamadas	1,666666667	3	3,5

Con un coste total de:(5*3)+((5+3)*1)+((5+3+7)*2)=53

Al comparar los costes podemos observar que el menor de todos es el de “ordenado de menos a mayor según peso/llamadas” con un coste de 53

Tras analizar distintas formas de de como organizar el archivo, se puede llegar al conclusión que la mas eficiente seria la ultima de todas, calculando el peso entre el numero de llamadas y ordenando ese valor de menor a mayor.

Componentes Voraces en el algoritmo
1.- Conjunto de candidatos: Todos los datos.
2.- Conjunto de decisiones: compara dos datos por cada iteración.
3.- Función que determina la solución del problema: longitud mínima de los comparados.
4.- Completable: se necesita recorrer todos los datos siempre.
5.- Función de selección: se elige el que tenga una longitud menor.
6.- Objetivo: Devolver el dato con menor longitud.

```
from random import randint
'''Creo el objeto fichero que es el que estará dentro de una cinta '''
class fichero(object):
    longitud=0
    llamadas=0
    def __init__(self,x,y):
        self.longitud=x
        self.llamadas=y

    def __str__(self):
        cadena= "longitud: "+self.longitud+"\nllamadas: "+self.llamadas
        return cadena

'''Crea de forma aleatoria la longitud y el numero de llamadas de cada uno de
los fichero que están en la cinta'''
def crearCinta(x):
    cintas = []
    for _ in range(x):
        cintas.append(fichero(randint(1,100),randint(1,20)))
    return cintas

'''para hacer la ordenación, voy a utilizar el metodo de la burbuja'''
def ordenarCintas(cintas):
    totalCintas=len(cintas)
    for i in range(totalCintas):
```

```
        for j in range(0,totalCintas-i-1):
            if ((cintas[j].longitud / cintas[j].llamadas)) > ((cintas[j + 1].longitud /
cintas[j + 1].llamadas)):
                cintas[j],cintas[j + 1] = cintas[j + 1],cintas[j]
    return cintas

'''main'''
cintas=crearCinta(3)
cintas=ordenarCintas(cintas)
'''Listo todos los fichero de la cinta para ver su tamaño y las veces que se le
llama'''
for fichero in cintas:
    print("\n\tLongitud: ", fichero.longitud, "\n\tLlamadas: ",fichero.llamadas, "\n\
tLL: ", fichero.longitud/fichero.llamadas)

total=0
for fichero in cintas:
    subtotal=0
    for fichero2 in cintas:
        if(fichero2==fichero):
            subtotal+=fichero2.longitud
            break
        else:
            subtotal+=fichero2.longitud
    total+=fichero.llamadas*subtotal
print("coste total:",total)
```

Explicación:

Definimos el objeto fichero que es el que estará dentro de una cinta repetidas veces. Este objeto se compone de 2 atributos llamados “longitud” y “llamadas”.

A continuación, defino la función “crearCinta” que creará la estrtuctura de la cinta formada por x objetos ficheros, definiendo de forma aleatoria tanto el numero de llamadas como la longitud de cada uno de los objetos fichero.

Seguidamente se define la función que va a ordenar la cinta utilizando el método de la burbuja y fijándose en que la longitud del fichero entre en numero de llamadas del mismo sea menor que el siguiente ya que gracias a este método hacemos que al leer la cinta, repita las primeras posiciones que serán las que le cueste menos leer por su tamaño.

Para finalizar, ponemos a prueba el objeto y las funciones definidas creando la estructura, ordenándola y mostrando el resultado por pantalla.

Ejemplo de ejecución:

Fichero 1:

Longitud: 2
Llamadas: 3

Fichero 2:

Longitud: 3
Llamadas: 3

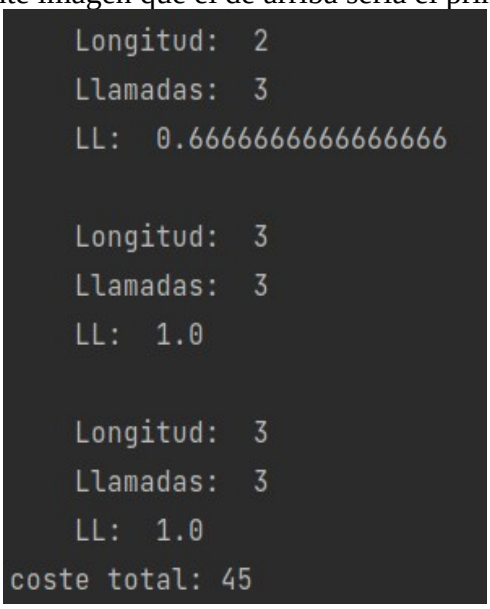
Fichero 3:

Longitud: 3
Llamadas: 3

Calculamos Longitud/Llamadas para el fichero que estemos en ese momento y lo comparamos con el que teníamos como objetivo. Si es menor, lo guardo como nuevo objetivo.

Haciendo esto, nos dará como resultado una lista ordenada de menor a mayor según (Longitud/Llamadas)

Se puede observar en la siguiente imagen que el de arriba seria el primero que se almacenaría



Problema 5

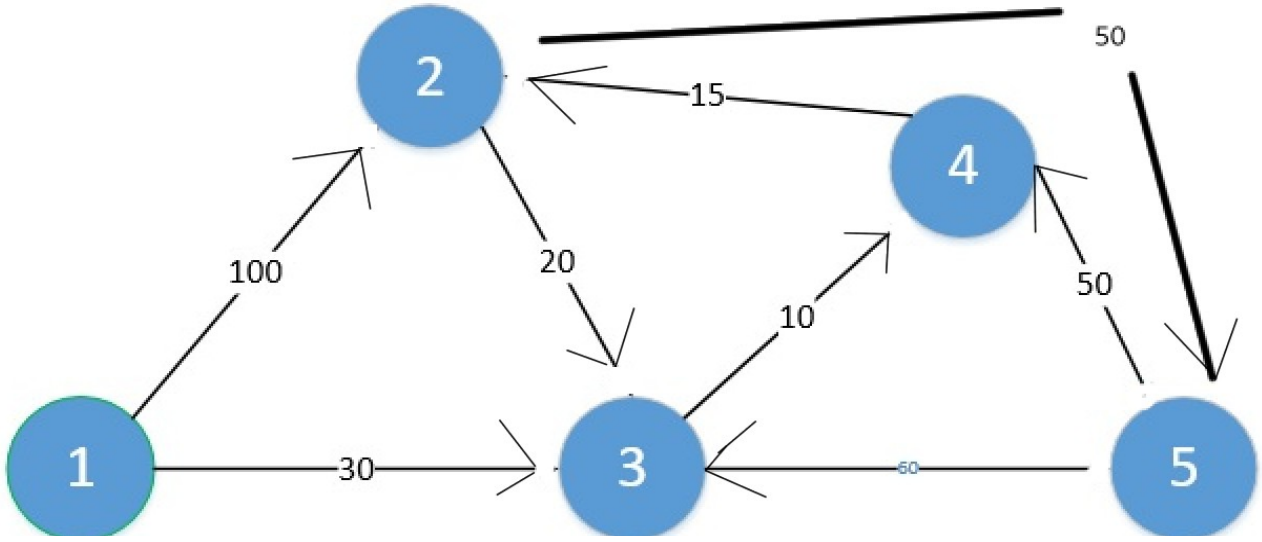
Enunciado:
Se tiene un grafo dirigido G = < N, A >, siendo N = {1, ..., n} el conjunto de nodos y A ⊆ NxN el conjunto de aristas. Cada arista (i, j) ∈ A tiene un coste asociado cij (cij > 0 ∀i, j ∈ N; si (i, j) ∉ A puede considerarse cij = +∞). Sea M la matriz de costes del grafo G, es decir, M [i, j] = cij . Teniendo como datos la cantidad de nodos n y la matriz de costes M, se pide encontrar tanto el camino mínimo entre los nodos 1 y n como la longitud de dicho camino usando el algoritmo de Dijkstra, utilizando las siguientes ideas:

• Crear un estructura de datos que almacene las distancias temporales conocidas (inicializadas al coste de la arista del vértice 1 a cada vértice j, o +∞ si no existe dicha arista) para los vértices no recorridos (inicialmente, todos salvo el 1).

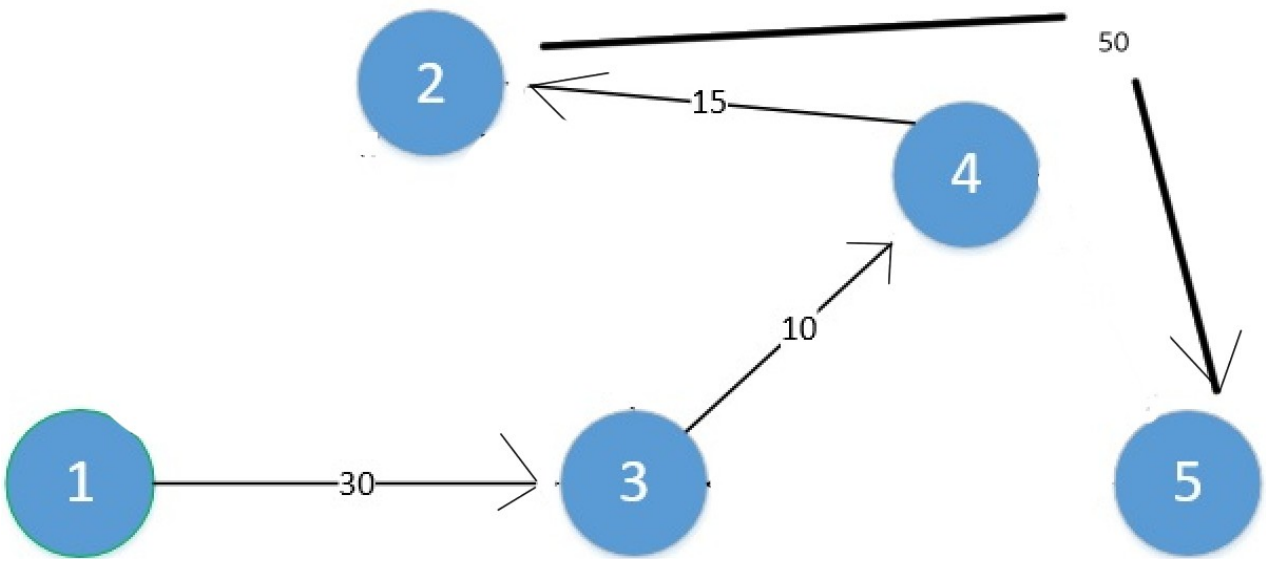
• Seleccionar como candidato el que tenga menor distancia temporal conocida, eliminarle del conjunto de vértices no recorridos, y actualizar el resto de distancias temporales si pueden ser mejoradas utilizando el vértice actual.

• Se necesitará almacenar de alguna manera la forma de recorrer el grafo desde el vértice 1 al vértice n (no necesariamente igual al conjunto de decisiones tomadas).

Dado el siguiente ejemplo que se observa en la imagen:



Aplicando el algoritmo de dijkstran, que consiste en seleccionar los caminos con menor coste sin formar ciclos y teniendo conexión con el resto de la red que estamos creando, quedaría de la siguiente forma:



Componentes Voraces en el algoritmo

1.- Conjunto de candidatos: Todos los datos.
2.- Conjunto de decisiones: compara dos datos por cada iteración.
3.- Función que determina la solución del problema: tiempo mínima de los comparados.
4.- Completable: se necesita recorrer todos los datos siempre.
5.- Función de selección: se elige el que tenga una tiempo menor.
6.- Objetivo: Devolver el dato con menor tiempo.

```
import math


def mostrar(matriz):
    for a in matriz:
        print(a)
    return


def algoritmoDijkstra(nodos, matriz_coste):
    numeroNodos = len(nodos)
    '''creo un lista con los candidatos restantes y un array con el recorrido que
lleva'''
    candidatos = list()
    caminoActual = []
    for a in nodos:
        candidatos.append(a+1)

    '''seleciono el primer nodo'''
    PrimerElemento = nodos[0]+1

    '''lo añado al recorrido que lleva y lo elimino de los candidatos restantes'''
    caminoActual.append(PrimerElemento)
    candidatos.remove(PrimerElemento)
```

```
    '''Elimino el nodo selecionado de la lista de nodos'''
    nodos.remove(PrimerElemento)

    '''almaceno los costos que tiene desde este nodo al resto de nodos'''
    matrizDeCoste = []
    for a in matriz_coste:
        aux = []
        for b in range(len(a)):
            aux.append(math.inf)
        matrizDeCoste.append(aux)

    costesDesdePrimero = []
    for i in range(0, numeroNodos):
        matrizDeCoste[0][i] = matriz_coste[0][i]
        costesDesdePrimero.append(matriz_coste[0][i])
    '''Mientras siga teniendo nodos sin alcanzar'''
    while candidatos:
        '''Miro cual es el menor coste desde el nodo selecionado a algun nodo'''
        coste = math.inf
        nodoMenorCoste = None
        for parcial in costesDesdePrimero:
            nodo = (costesDesdePrimero.index(parcial) + 1)

            if (parcial < coste) and (nodo in candidatos) and parcial != 0:
                '''Al encontrar uno más barato, lo guardo'''
                coste = parcial
                nodoMenorCoste = nodo
        print("Nodo: ", nodoMenorCoste)
        print("Coste acumulado : ", coste)
        print("-----")
        '''elimino de la lista de candidatos el que tiene el menor coste hacia el
ultimo que tenia selecionado'''
        candidatos.remove(nodoMenorCoste)

        '''guardo el nodo'''
        caminoActual.append(nodoMenorCoste)

        '''actualizo la matriz de costes'''
        for i in range(0, numeroNodos):
            matrizDeCoste[nodoMenorCoste - 1][i] = matriz_coste[nodoMenorCoste
- 1][i]
        '''Comparo con otros caminos'''
        for i in range(0, numeroNodos):
            matrizDeCoste[0][i] = min(costesDesdePrimero[i],
(matrizDeCoste[nodoMenorCoste - 1][i] + coste))
            costesDesdePrimero[i] = min(costesDesdePrimero[i],
(matrizDeCoste[nodoMenorCoste - 1][i] + coste))

    ''' Calculo el coste total de recorrer el diagrama'''
```

```
    costeTotal = 0
    for i in range(0, numeroNodos-1):
        costeTotal += matriz_coste[caminoActual[i]-1][caminoActual[i + 1]-1]
    for a in range (len(caminoActual)):
        caminoActual[a]=caminoActual[a]-1
    return costeTotal, caminoActual


''' Creo un diagrama con 5 nodos como se muestra en la imagen del
documento adjunto'''
nodos = [0, 1, 2, 3, 4]
'''indico las distancias entre los nodos '''
matriz_coste = [[0, 100, 30, math.inf, math.inf],
                [math.inf, 0, 20, math.inf, 50],
                [math.inf, math.inf, 0, 10, math.inf],
                [math.inf, 15, math.inf, 0, math.inf],
                [math.inf, math.inf, 60, 50, 0]]
''' Aplico el algoritmo de Dijkstra'''
coste, camino_mas_corto = algoritmoDijkstra(nodos, matriz_coste)

''' Muestro el resultado por pantalla'''
print("Recorrido:")
for i in (camino_mas_corto):
    print("\tNodo ",i + 1)
print("\tCon un costo total de :", coste)
```

Explicación:

Comenzamos creando un array con el nombre de los nodos y otro bidimensional que contendrá el coste para conectar los nodos.

Hacemos una llamada a la función pasando y le pasamos estos dos elemento.

Creamos las estructuras necesarias y nos guardamos el coste desde el primer nodo al resto. Recorro todos los nodos hasta encontrar el camino con menor peso. Lo elimino de la lista de candidatos y lo guarda para el camino final. Actualizo la matriz que contiene los costes de los caminos y repito el proceso hasta no tener mas nodos candidatos.

En este punto, ya hemos encontrado el camino con el menor coste y lo mostramos por pantalla

Problema 6

Enunciado:
Shrek, Asno y Dragona llegan a los pies del altísimo castillo de Lord Farquaad para liberar a Fiona de su encierro. Como sospechaban que el puente levadizo estaría vigilado por numerosos soldados se han traído muchas escaleras, de distintas alturas, con la esperanza de que alguna de ellas les permita superar la muralla; pero ninguna escalera les sirve porque la muralla es muy alta. Shrek se da cuenta de que, si pudiese combinar todas las escaleras en una sola, conseguiría llegar exactamente a la parte de arriba y poder entrar al castillo.
Afortunadamente las escaleras son de hierro, así que con la ayuda de Dragona van a "soldarlas". Dragona puede soldar dos escaleras cualesquiera con su aliento de fuego, pero tarda en calentar los extremos tantos minutos como metros suman las escaleras a soldar. Por ejemplo, en soldar dos escaleras de 6 y 8 metros tardaría 6 + 8 = 14 minutos. Si a esta escalera se le soldase después una de 7 metros, el nuevo tiempo sería 14 + 7 = 21 minutos, por lo que habrían tardado en hacer la escalera completa un total de 14 + 21 = 35 minutos.
Diseñar un algoritmo eficiente que encuentre el mejor coste y manera de soldar las escaleras para que Shrek tarde lo menos posible es escalar la muralla, indicando las estructuras de datos elegidas y su forma de uso. Se puede suponer que se dispone exactamente de las escaleras necesarias para subir a la muralla (ni sobran ni faltan), es decir, que el dato del problema es la colección de medidas de las "miniescaleras" (en la estructura de datos que se elija), y que solo se busca la forma óptima de fundir las escaleras.

A partir de un ejemplo mostraré la forma óptima de hacerlo.

Poniendo un caso que tenga 3 escaleras con medidas e1=1, e2=2, e3=3 y e4=4. Existen distintas formas de juntar las escaleras como por ejemplo de mayor cantidad de menos a menor, de menor a mayor, en orden a como nos las han dado entre otras formas. En la siguiente imagen se muestra algunos ejemplos:

1	2	3	4
tiempo total será:	20		
4	3	2	1
tiempo total será:	30		
1	4	2	3
tiempo total será:	23		
2	3	1	4
tiempo total será:	23		
1	2	4	3
tiempo total será:	21		
4	3	1	2
tiempo total será:	29		

Otra opción de juntar las y que no se muestra en la imagen superior es:

Juntamos las 2 primeras escaleras con menor tamaño. A continuación miramos si la siguiente escalera más pequeñas es más corto en tiempo unirla con las que ya habíamos unido previamente o con la siguiente que aun no habíamos tocado.
Siguiendo este algoritmo y en base al caso propuesto, el coste total de unir las escaleras seria 19 siendo el menor de todos.

Componentes Voraces en el algoritmo
1.- Conjunto de candidatos: Todos los datos.
2.- Conjunto de decisiones: compara dos datos por cada iteración.
3.- Función que determina la solución del problema: tiempo mínima de los comparados.
4.- Completable: se necesita recorrer todos los datos siempre.
5.- Función de selección: se elige el que tenga una tiempo menor.
6.- Objetivo: Devolver el dato con menor tiempo.

```
import copy
from random import randint

'''Esta función recibe el numero de escaleras que tiene que crear y las crea con
una longitud aleatoria'''


def crearEscaleras(numeroEscaleras):
    escaleras = []
    for i in range(numeroEscaleras):
        '''Creo aleaoriamente la longitud de la escalera entre 1 y 10'''
        escaleras.append(randint(1, 200))
    return escaleras


def escaleraMaspequena(escaleras):
    '''busco la que tenga una longitud más pequeña de todas las disponibles'''
    selecionada = 9999999
    for escalera in escaleras:
        if (escalera < selecionada):
            selecionada = escalera
    return selecionada


'''La función aplicará el algoritmo propuesto para minimizar el tiempo de
construcción de la escalera'''
def construirEscalera(escaleras):
    escaleraCompleta = []

    tiempo = 0
    '''Mientras siga teniendo escaleras sin utilizar'''
    selecionada = escaleraMaspequena(escaleras)
```

```
    '''Sabiendo ya cual es la menor de todas, la borro de las candidatas '''
    escaleras.remove(selecionada)
    '''miro cual es la siguiente mas pequeña'''
    selecionada2 = escaleraMaspequena(escaleras)

    '''guardo la nueva escalera formada'''
    escaleraCompleta.append(selecionada + selecionada2)
    '''Almaceno el tiempo de unir las 2 primeras'''
    tiempo = selecionada + selecionada2
    escaleras.remove(selecionada2)
    print("Unimos: ",selecionada, "+",selecionada2," = ",
(selecionada+selecionada2))
    '''bucle mientras siga teniendo escaleras por unir'''
    while (escaleras):
        '''miro cual es la que tiene un tiempo menor'''
        escal = copy.copy(escaleras)
        opcion = escaleraMaspequena(escal)
        escal.remove(opcion)
        '''si sigue habiend oalguna escalera más que esta'''
        if (len(escal) != 0):
            opcion2 = escaleraMaspequena(escal)
            '''si me es más restable unir las dos por separado que juntar una con la
escalera principal'''
            if ((opcion + opcion2) < sum(escaleraCompleta) + opcion2):
                escaleras.remove(opcion)
                escaleras.remove(opcion2)
                escaleras.append((opcion + opcion2))
                print("Unimos: ",opcion,"+",opcion2," = ",(opcion + opcion2))
                tiempo += (opcion + opcion2)
            else:
                '''si no merece la pena la condicion propuesta'''
                escaleras.remove(opcion)
                tiempo += (opcion + sum(escaleraCompleta))  # puede estar mal
                print("Unimos: ",opcion,"+",sum(escaleraCompleta)," = ",
(sum(escaleraCompleta) + opcion))
                escaleraCompleta.append(opcion)
        else:
            '''en caso que sea la ultima escalera por unir a la principal'''
            escaleras.remove(opcion)
            tiempo += (opcion + sum(escaleraCompleta))  # puede estar mal
            print("Unimos: ",opcion ,"+",sum(escaleraCompleta)," = ",
(sum(escaleraCompleta)+ opcion ))
            escaleraCompleta.append(opcion)
    '''Actualizo el tiempoto total que tarda en construirse la escalera final'''
    return tiempo


'''Defino el numero de escaleras que se van a crear'''
numeroEscaleras = 5
```

```
'''llamo a la funcion que me creará el numero de escaleras con una longitud
aletoria'''
escaleras = crearEscaleras(numeroEscaleras)
print("Las escaleras tienen las siguientes medidas: ", escaleras)
'''Aplico el algoritmo propuesto'''
tiempo = construirEscalera(escaleras)
print("Tardando un tiempo de ", tiempo, " minutos")
```

Explicación:

Creamos una funciona a la que se le pasa el numero de escaleras que tiene que crear e irá añadiendo a un array las escaleras representadas por un tiempo aleatorio.

También creamos una función que se encargará de crear la escalera que contenga todas las creadas previamente.

Llamamos a la función principal que unirá los 2 primeros trozos de escaleras y luego evaluará cual es el menor coste, si juntar la siguiente con las previas o con la siguiente.
Realizamos este proceso hasta no tener ninguna escalera por separado

Durante todo este proceso, se irá almacenando tanto el tiempo total que tarda en construirse la escalera final y el orden que se ha ido llevando en esa construcción.

Como ejemplo:

```
Las escaleras tienen las siguientes medidas:  [194, 184, 72, 3, 196]
Unimos:  3 + 72  =  75
Unimos:  184 + 75  =  259
Unimos:  194 + 196  =  390
Unimos:  390 + 259  =  649
Tardando un tiempo de  1373  minutos
```