P2 – Conjuntos Disjuntos y Componentes Conexas. El problema del Viajante

Juan José Sánchez Peña

Requisitos de entrega (1/2)

- Crear una carpeta con el nombre p2**NN**(1) e incluir los siguientes ficheros:
  - p2**NN**.py: incluirá el código de las funciones implementadas en la práctica y los `imports` estrictamente necesarios (no incluir los scripts de medida de tiempo o dibujo de gráficas).
  - p2**NN**.html: contendrá el resultado de aplicar el comando pdoc del paquete pdoc3 al módulo Python p2**NN**.py.
  - p2NN.pdf: memoria que contenga las respuestas a las cuestiones de la práctica.

(1) **NN** indica el número de pareja.

Requisitos de entrega (2/2)

- Observaciones
  - Los nombres y parámetros de las funciones definidas en p2**NN**.py deben ajustarse EXACTAMENTE a lo definido en el enunciado. Además todas las funciones definidas deben incorporar los type hints adecuados.
  - En la memoria se identificará claramente el nombre de los estudiantes y el número de pareja. Si se añaden figuras o gráficos, deben realizarse sobre fondo blanco.
  - Para la entrega, comprimir la carpeta en un fichero llamado p2**NN**.zip o p2**NN**.7z. No añadir ninguna estructura de subdirectorios a dicha carpeta.
  - **La práctica no se corregirá hasta que el envío no siga esta estructura**.

Corrección

- Ejecución del script que importará p2**NN**.py, que comprobará la corrección de dicho código. **¡¡IMPORTANTE!! La práctica no se corregirá mientras este script no se ejecute correctamente, penalizando las segundas entregas**.
- Revisión de la documentación del código generada con pdoc: por ejemplo, las docstrings deben ser escritas muy cuidadosamente; el código Python debe estar formateado según el estándar **PEP-8**(1) (utilizar formateador **autopep8**(2) o **black**(3)).
- Revisión de ciertas funciones implementadas en p2**NN**.py.
- Revisión de la memoria con las respuestas a las cuestiones planteadas.

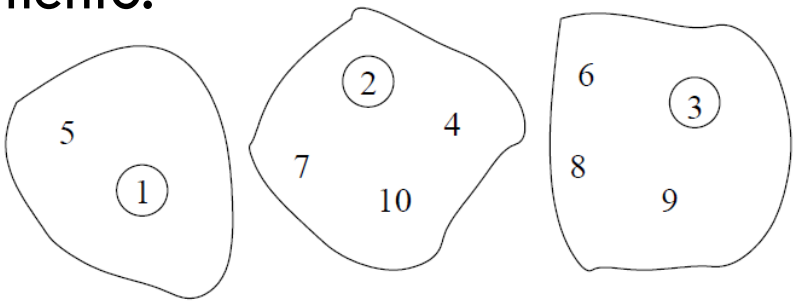
(1) https://peps.python.org/pep-0008/
(2) https://pypi.org/project/autopep8/
(3) https://github.com/psf/black

TAD Conjunto Disjunto

- Representa elementos clasificados en categorías (donde cada elemento pertenece únicamente a un solo conjunto – conjuntos disjuntos), de forma que se establece una **partición** sobre ellos.
  - Cada categoría (conjunto) se representa mediante un elemento cualquiera que pertenezca a ella (representante).
  - Las operaciones que se pueden realizar sobre este TAD son:
    - Creación a partir de una colección de elementos: cada elemento define una categoría.
    - Búsqueda: determinar a qué categoría pertenece un elemento (devuelve el representante de esa categoría).
    - Unión: unir dos categorías en una sola.

TAD Conjunto Disjunto: representación

- Todos los elementos se enumeran de 1 a n.
- Cada subconjunto será identificado por uno de sus elementos (representante).
- Se mantiene en un array el nombre del subconjunto disjunto de cada elemento.



	1	2	3	4	5	6	7	8	9	10
C	1	2	3	2	1	3	2	3	3	2

Fuente: Santiago Jorge – Dpto. Computación – Universidad A Coruña

TAD Conjunto Disjunto: pseudocódigo

```
Tipo
   elemento = entero;
   Conjunto = entero;
   ConjuntoDisjunto = vector [1..N] de entero;
función Buscar (C, x) : Conjunto
   devolver C[x]
fin función
```

- La búsqueda es una simple consulta O(1).
  - El nombre del conjunto devuelto por la función `búsqueda` es arbitrario.
  - Todo lo que importa es que `búsqueda(x)=búsqueda(y)` ↔ x e y están en el mismo conjunto.

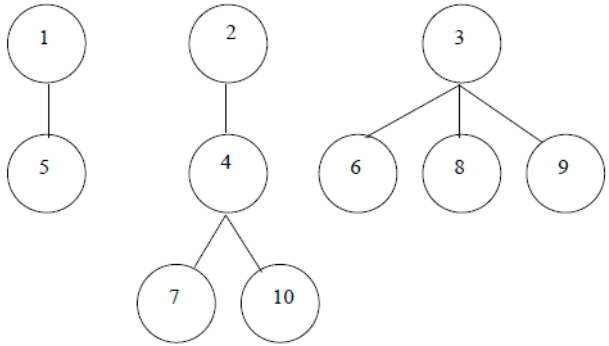
TAD Conjunto Disjunto: pseudocódigo

```
procedimiento Unir (C, a, b)
   min := mínimo (C[a], C[b]);
   max := máximo (C[a], C[b]);
   para i := 1 hasta N hacer
      si C[i] = max entonces C[i] := min
   fin para
fin procedimiento
```

- La unión es O(n).
- Una secuencia de n-1 uniones (la máxima, ya que tendríamos todo en un conjunto) tomaría $O(n^2)$.
- La combinación de m búsquedas y n-1 uniones es $O(m+ n^2)$.

TAD Conjunto Disjunto: uso de árboles “padres”

- Se utiliza un árbol para caracterizar cada subonjunto.
- La raíz es el representante de cada subconjunto.
- Cada entrada p[i] en el vector contiene el padre del elemento i. Si i es una raíz, entonces p[i] = i.



	1	2	3	4	5	6	7	8	9	10
C	1	2	3	2	1	3	4	3	3	4

Fuente: Santiago Jorge – Dpto. Computación – Universidad A Coruña

TAD Conjunto Disjunto: pseudocódigo

```
función Buscar (C, x) : Conjunto
   raíz := x;
   mientras C[raíz] <> raíz hacer
      raíz := C[raíz]
   fin mientras
   devolver raiz
fin función
```

- La búsqueda sobre el elemento x se realiza devolviendo la raíz del árbol que contiene x.
- La búsqueda de un elemento x es proporcional a la profundidad del nodo con x. El peor caso es O(n).

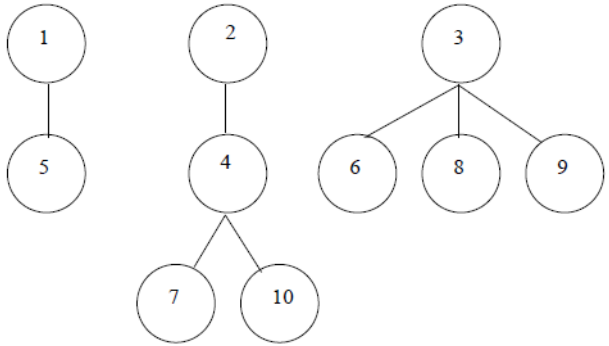
TAD Conjunto Disjunto: pseudocódigo

```
procedimiento Unir (C, raiz1, raiz2)
   si raiz1 < raiz2 entonces C[raiz2] := raiz1
   sino C[raiz1] := raiz2
fin procedimiento
```

- La unión de dos conjuntos se efectúa combinando ambos árboles: apuntamos la raíz de un árbol a la del otro.
- La unión toma O(1).
- La combinación de m búsquedas y n-1 uniones es O(m-n).

TAD Conjunto Disjunto: uso de árboles “rangos negativos”

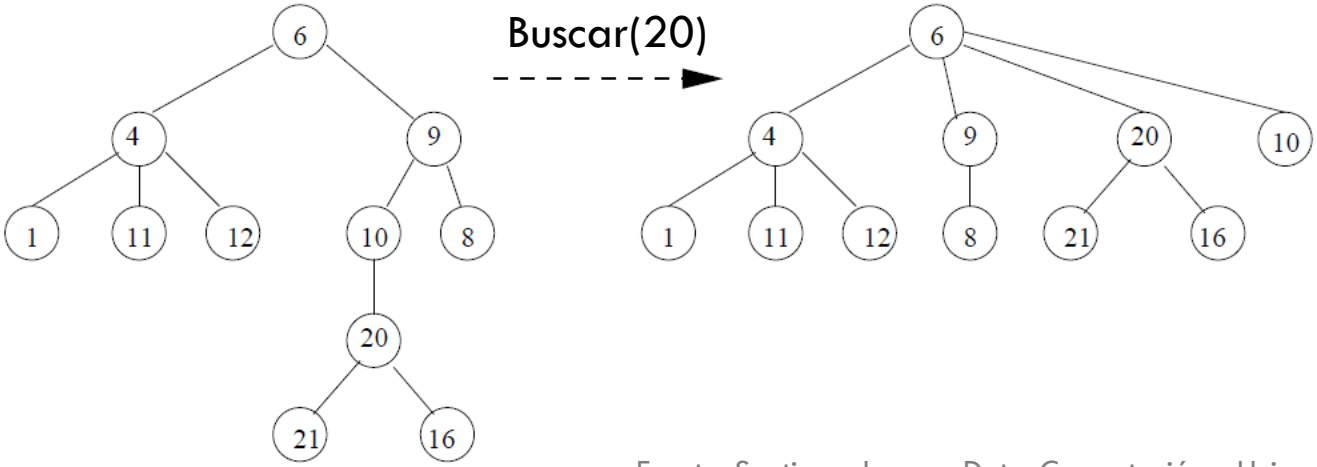
- Se utiliza un árbol para caracterizar cada subonjunto.
- La raíz es el representante de cada subconjunto.
- Cada entrada p[i] en el vector contiene el padre del elemento i. Si i es una raíz, entonces p[i] = -1.



	1	2	3	4	5	6	7	8	9	10
C	-1	-1	-1	2	1	3	4	3	3	4

TAD Conjunto Disjunto: heurística "compresión de caminos"

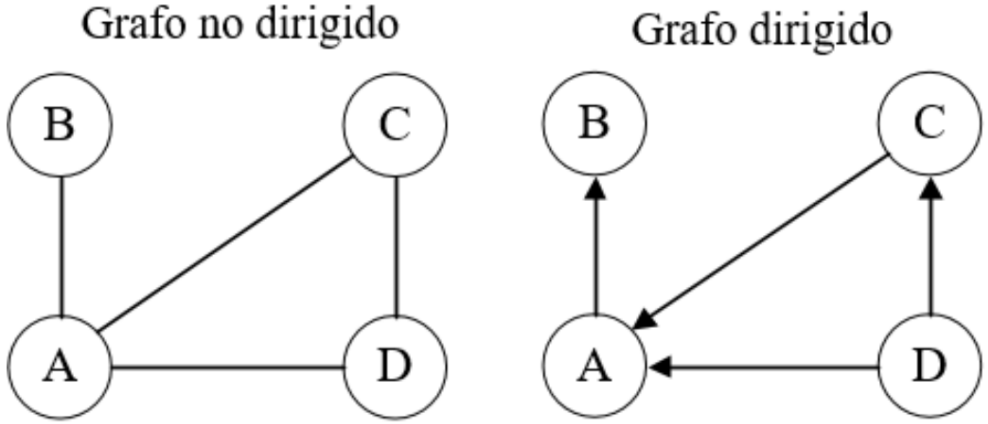
- Se usa en la función búsqueda.
- Hace que los nodos recorridos en el camino de búsqueda pasen a apuntar directamente a la raíz.



Fuente: Santiago Jorge – Dpto. Computación – Universidad A Coruña

Componentes conexas de grafos no dirigidos

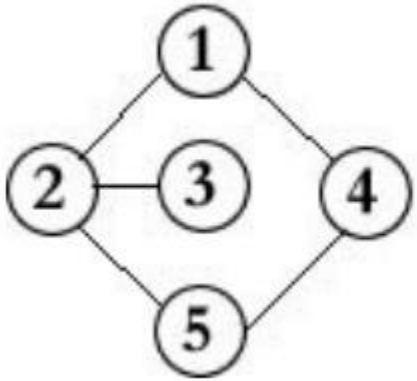
- Un uso de los TAD Conjunto Disjunto es encontrar las componentes conexas de grafos no dirigidos.
- Grafo no dirigido: conjunto de vértices (V) y ramas (E) que conectan dichos vértices en ambos sentidos.

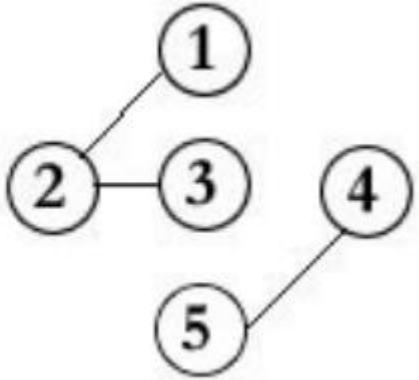


Componentes conexas de grafos no dirigidos

- Un grafo es conexo si hay un camino entre cualquiera de dos vértices.

GRAFO CONEXO



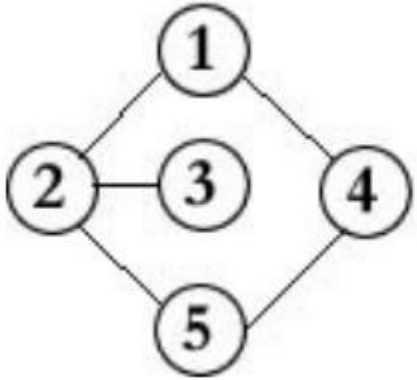


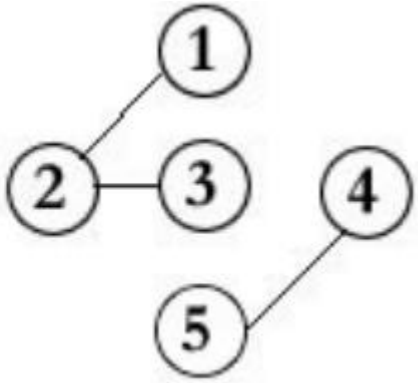
GRAFO NO CONEXO

- Las componentes conexas de un grafo son los subgrafos conexos *maximales* (un camino *maximal* no está contenido en otro camino).

Componentes conexas de grafos no dirigidos

GRAFO CONEXO





GRAFO NO CONEXO

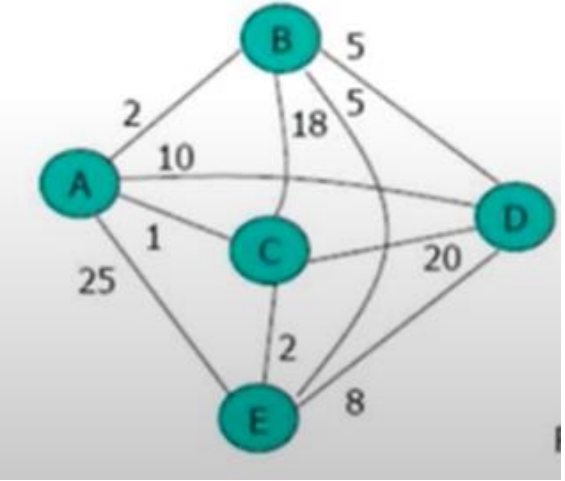
- Grafo conexo: una componente conexa.
- Grafo no conexo: dos componentes conexas (1, 2 y 3) y (4, 5).
- Pseudocódigo: página 169 de teoría.

El problema del viajante

- El problema del viajante (en inglés, ***T**ravelling **S**alesman **P**roblem*) consiste en encontrar el camino único más corto, dada una lista de ciudades y las distancias entre ellas, visita todas las ciudades una sola vez y regresa a la ciudad origen.
- Implementaciones:
  - Vecino más cercano.
  - Repetitivo.
  - Exhaustivo.

El problema del viajante

- Vecino más cercano:
  - Es una buena aproximación, pero no proporciona la solución óptima.
  - El método consiste en una vez establecido el nodo de partida, evaluar y seleccionar su vecino más cercano.



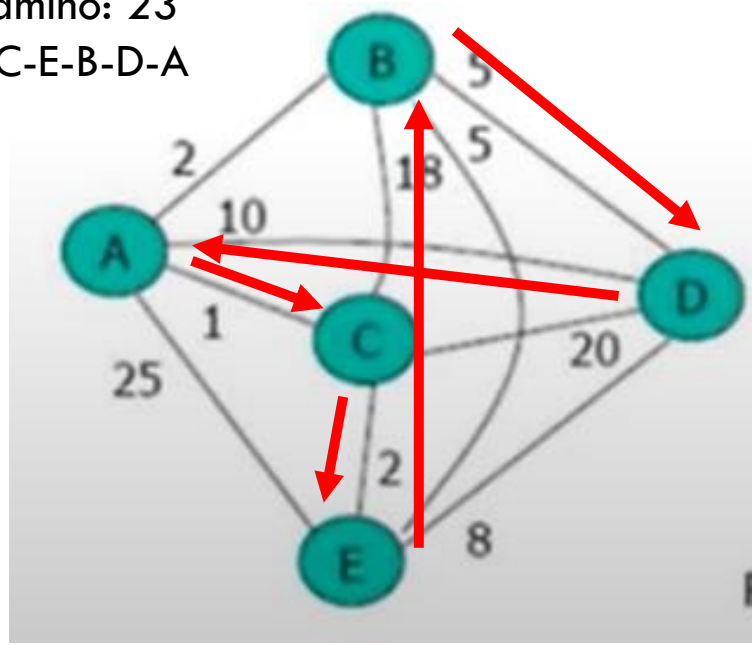
	A	B	C	D	E
A	-	2	1	10	25
B	2	-	18	5	5
C	1	18	-	20	2
D	10	5	20	-	8
E	25	5	2	8	-

Ciudad origen: A

Camino: 23
A-C-E-B-D-A

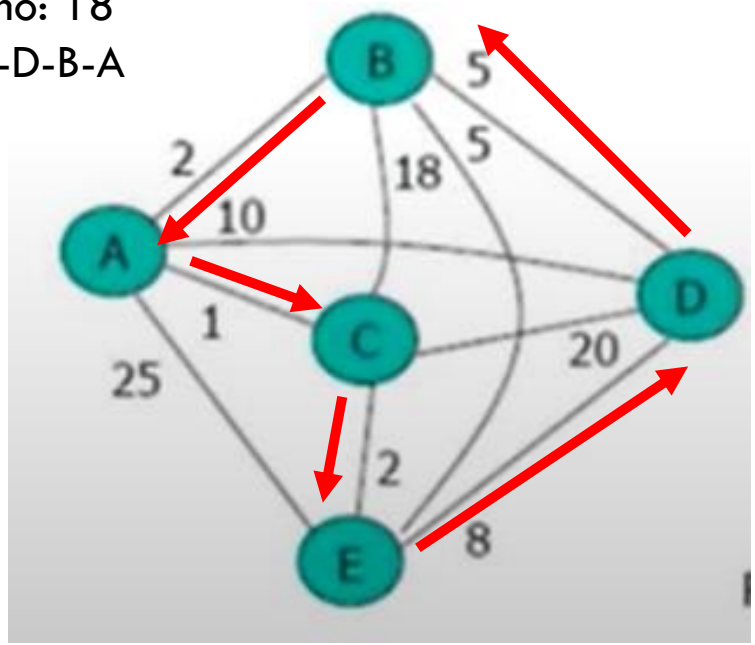
El problema del viajante

Camino: 23
A-C-E-B-D-A



VECINO MÁS CERCANO

Camino: 18
A-C-E-D-B-A



SOLUCIÓN ÓPTIMA

El problema del viajante

- Repetitivo:
  - Aplicar vecino más cercano a todos los nodos del grafo y obtener aquel trayecto con menor longitud.

	A	B	C	D	E
A	0	2	1	10	25
B	2	0	18	5	5
C	1	18	0	20	2
D	10	5	20	0	8
E	25	5	2	8	0

Ciudad origen: A
Camino: 23
A-C-E-B-D-A

Ciudad origen: B
Camino: 23
B-A-C-E-D-B

Ciudad origen: C
Camino: 18
C-A-B-D-E-C

Ciudad origen: D
Camino: 18
D-B-A-C-E-D

Ciudad origen: E
Camino: 18
E-C-A-B-D-E

El problema del viajante

- Exhaustivo:
  - Examinar todos los posibles circuitos, devolviendo aquel con la distancia más corta.
  - Librería itertools (https://docs.python.org/3/library/itertools.html).
  - Orden lexicográfico: orden de diccionario.
  - Ejemplo: las palabras de cinco letras en el alfabeto A, B.
    - En orden lexicográfico sería: AAAAA, AAAAB, AAABA, AAABB, etc.