Universidad de Alcalá

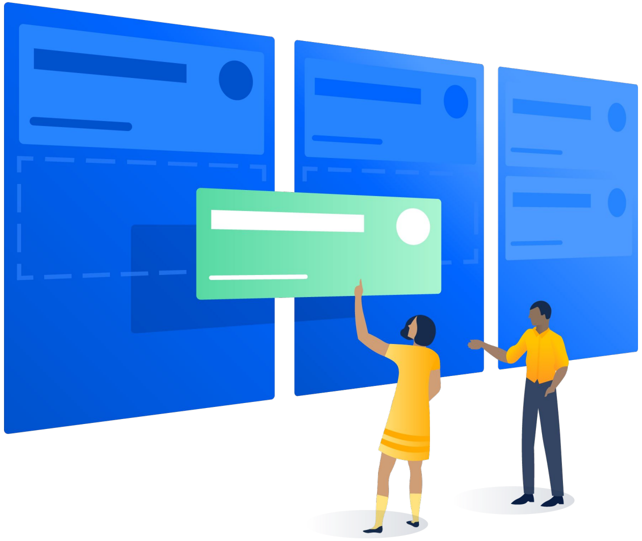
Algoritmos voraces

Tema 2

javier@junquera.io (UAH 2023)

Agenda

- **Voraces**
  - Problema cambio de monedas
  - Problema de la mochila
- **Árboles de recubrimiento mínimo**
  - Algoritmo de Kruskal
  - Algoritmo de Prim
- **Grafos dirigidos**
  - Algoritmo de Dijkstra



javier@junquera.io (UAH 2023)

Voraces

Universidad de Alcalá

algoritmo voraz. Algoritmo recursivo capaz de resolver un problema aplicando, a un conjunto de candidatos a formar parte de la solución, funciones de maximización de resultados. Un algoritmo voraz será correcto si es capaz de encontrar solución, y esta es óptima; aunque en ocasiones puede que no consigamos ni una cosa ni la otra.

> ALGORITMO GOLOSO, GREEDY ALGORITHM.

Voraces

¿Qué no es un algoritmo voraz?

- Un algoritmo basado en pre calcular soluciones parciales.

  i.e., Programación dinámica.

- Un algoritmo basado en búsqueda exhaustiva de soluciones.

  i.e., Backtracking.

Voraces

Universidad de Alcalá

¿Qué es un algoritmo voraz?

- Una estrategia simple de resolución de problemas.

  Que, a veces, nos permitirá resolver problemas complejos.

- Un planteamiento en el que, a través de soluciones óptimas parciales, nos permitirá obtener una solución óptima global

Elementos clave

- **Candidatos**

  Elementos que pueden entrar a formar parte de la solución

- **Solución**

  Conjunto de candidatos que pone fin al problema

Universidad de Alcalá

Elementos clave

- **Completable**

  Tras introducir un candidato, se puede llegar a una solución?

- **Función objetivo**

  El conjunto de la solución cumple con requisitos óptimos

- **Función de selección**

  Función que permite lograr función objetivo

javier@junquera.io (UAH 2023)

Fases

Previa

Determinar función de optimización

Implementación

1. **Inicialización (x1)**

   Estructuración de candidatos

2. **Selección (1..k)**

   Uso de la función de optimización para elegir elementos de la solución

Voraces

```
def voraz(candidatos): # Conjunto preparado previamente
    solucion = []
    while not es_solucion(solucion) and len(candidatos) > 0:
        c = mejor_candidato(candidatos)
        candidatos.remove(c)
        if es_completable(solucion, c):
            solucion.append(c)

    if es_solucion(solucion):
        return solucion
    else:
        raise Exception("No hay solución")
```

Cálculo de eficiencia

$$T(n) = O(\ init(n)\ ) + O(n)\ O(\ f(n)\ )$$

Problema de cambio de monedas

Universidad de Alcalá

Pagar un importe exacto con el menor número de monedas posible

Problema de cambio de monedas

- **Candidatos**

  Monedas disponibles

- **Solución**

  Monedas devueltas, que cumplen con sumar el importe

- **Completable**

  Las monedas depositadas no superan el importe

Problema de cambio de monedas

Universidad de Alcalá

- **Función objetivo**

  Total de monedas de solución es el mínimo posible

- **Función de selección**

  Moneda, entre las candidatas, de mayor valor

Universidad de Alcalá

Problema de la mochila

- **Mochila**
  - Mochila con capacidad de 8 L
- **Candidatos**
  - Tablet (100€ / 1L)
  - Cuaderno (1€ / 1L)
  - Termo (10€ / 0.5L)
  - Tupper con comida (13€ / 2L)
  - Abrigo (50€ / 5L)
  - Estuche con material (20€ / 1.5L)
  - Cargador del ordenador (20€ / 1L)

Problema de la mochila

Universidad de Alcalá

Tenemos objetos de distintos valores y volúmenes, y queremos llenar una mochila maximizando el valor de su contenido:

- **Discreto**

  Es completable si entra el objeto entero

- **Optimizable**

  Los objetos se pueden dividir

Problema de la mochila

- **Candidatos**

  Objetos disponibles

- **Solución**

  Elementos introducidos en la mochila

- **Completable**

  Los objetos solución no superan la capacidad de la mochila

Problema de la mochila

- **Función objetivo**

  Valor de los objetos de solución es máximo

- **Función de selección**
  a. Objetos de menor volumen (a más volumen, menos cosas caben)
  b. Objetos de mayor valor (a más valor parcial, más valor total)
  c. Objetos de mayor relación valor/volumen (valor suma, volumen resta)

Problema de la mochila (A)

Universidad de Alcalá

	Objeto	Valor	**Volumen**
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

S	Vol. acum.	Valor
3	0,5	10
1	1,5	110
7	2,5	130
2	3,5	131
6	5	151
4	7	164

Problema de la mochila (B)

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

S	Vol. acum.	Valor
1	1	100
5	6	150
6	7,5	170
7	8	190

Problema de la mochila (C)

Universidad de Alcalá

	Objeto	Valor	Volumen	**val/vol**
1	Tablet	100€	1 L	100
2	Cuaderno	1€	1 L	1
3	Termo	10€	0,5 L	20
4	Tupper	13€	2 L	6,5
5	Abrigo	50€	5 L	10
6	Estuche	20€	1,5 L	13,3
7	Cargador	20€	1 L	20

S	Vol. acum.	Valor
1	1	100
3	1,5	110
7	2,5	130
6	4	150
5	9	200
4	6	163
2	7	164

Volveremos por aquí cuando veamos programación dinámica

Problema de la mochila (C, optimizable)

	Objeto	Valor	Volumen	**val/vol**
1	Tablet	100€	1 L	100
2	Cuaderno	1€	1 L	1
3	Termo	10€	0,5 L	20
4	Tupper	13€	2 L	6,5
5	Abrigo	50€	5 L	10
6	Estuche	20€	1,5 L	13,3
7	Cargador	20€	1 L	20

S	Vol. acum.	Valor	Cantidad
1	1	100	1
3	1,5	110	1
7	2,5	130	1
6	4	150	1
5	8	190	0,8

Si podemos partir los objetos, podremos optimizar el resultado

Eficiencia (Problema de la mochila B, directo) [1]

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(while)`
`while completable:`	
`busca_valor()`	

Eficiencia (Problema de la mochila B, directo) [2]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(while)`
`while completable:`	
`busca_valor()`	`T(n) = n`

Eficiencia (Problema de la mochila B, directo) [3]

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(while)`
`while completable:`	`T(n) = n * T(busca_valor())`
`busca_valor()`	`O(n)`

Eficiencia (Problema de la mochila B, directo) [4]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(while)`
`while completable:`	`T(n) → n * O(n)`
`busca_valor()`	

Eficiencia (Problema de la mochila B, directo) [5]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

$O_1(n) * O_2(m) = O(n * m)$

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(while)`
`while completable:`	`T(n) → O(n) * O(n)`
`busca_valor()`	

Eficiencia (Problema de la mochila B, directo) [6]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(while)`
`while completable:`	$O(n^2)$
`busca_valor()`	

Eficiencia (Problema de la mochila B, directo) [7]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
2	Cuaderno	1€	1 L
3	Termo	10€	0,5 L
4	Tupper	13€	2 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	$O(n^2)$
`while completable:`	
`busca_valor()`	

Eficiencia (Problema de la mochila B, ordenado) [1]

Ordenar → `O(n log(n))`

	Objeto	**Valor**	...lumen
1	Tablet	100€	1 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L
4	Tupper	13€	2 L
3	Termo	10€	0,5 L
2	Cuaderno	1€	1 L

	acum.	...alor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(ordenar) + T(while)`
`    ordenar()`	
`    while completable:`	
`        busca_valor()`	

Eficiencia (Problema de la mochila B, ordenado) [2]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L
4	Tupper	13€	2 L
3	Termo	10€	0,5 L
2	Cuaderno	1€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(ordenar) + T(while)`
`    ordenar()`	`O(n log(n))`
`    while completable:`	
`        busca_valor()`	`T(n) = 1`

Eficiencia (Problema de la mochila B, ordenado) [3]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L
4	Tupper	13€	2 L
3	Termo	10€	0,5 L
2	Cuaderno	1€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(ordenar) + T(while)`
`    ordenar()`	`O(n log(n))`
`    while completable:`	`T(n) = n * T(busca_valor())`
`        busca_valor()`	`O(1)`

javier@junquera.io (UAH 2023)

Eficiencia (Problema de la mochila B, ordenado) [4]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L
4	Tupper	13€	2 L
3	Termo	10€	0,5 L
2	Cuaderno	1€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) = T(ordenar) + T(while)`
`    ordenar()`	`O(n log(n))`
`    while completable:`	`T(n) → O(n) * O(1) = O(n)`
`        busca_valor()`	

Eficiencia (Problema de la mochila B, ordenado) [5]

Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L
4	Tupper	13€	2 L
3	Termo	10€	0,5 L
2	Cuaderno	1€	1 L

$O_1(n) + O_2(m) = O(\max\{(n, m\})$

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`T(n) → O(n log(n)) + O(n)`
`    ordenar()`	
`    while completable:`	
`        busca_valor()`	

Eficiencia (Problema de la mochila B, ordenado) [6]

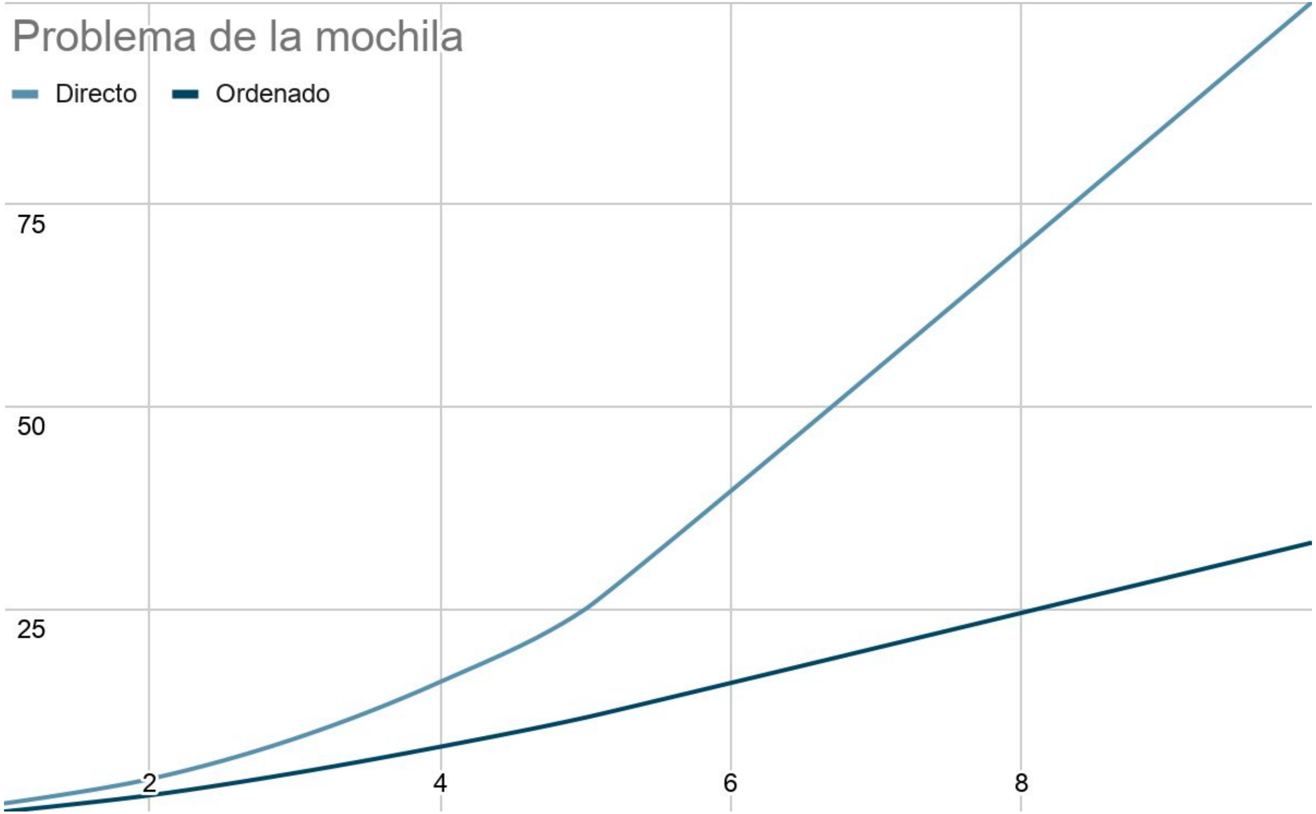
Universidad de Alcalá

	Objeto	**Valor**	Volumen
1	Tablet	100€	1 L
5	Abrigo	50€	5 L
6	Estuche	20€	1,5 L
7	Cargador	20€	1 L
4	Tupper	13€	2 L
3	Termo	10€	0,5 L
2	Cuaderno	1€	1 L

S	Vol. acum.	Valor	Coste
1	1	100	n
5	6	150	n - 1
6	7,5	170	n - 2
7	8	190	n - 3

`voraz():`	`O(n log(n))`
`    ordenar()`	
`    while completable:`	
`        busca_valor()`	

Discusión eficiencia: Problema de la mochila B



Universidad de Alcalá

Árboles de recubrimiento

javier@junquera.io (UAH 2023)

Árboles de recubrimiento mínimo

Universidad de Alcalá

- **Árbol libre**

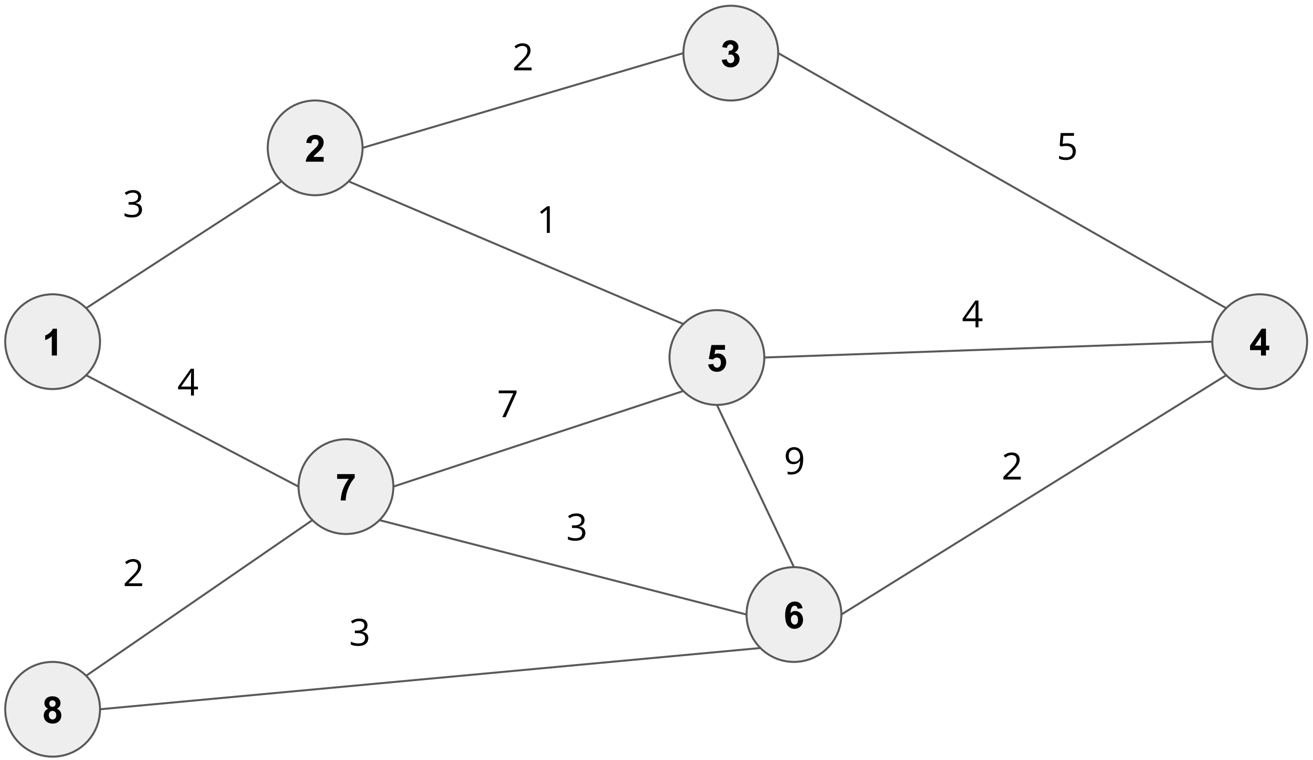
  Grafo no dirigido, conexo y acíclico

- **Árbol de recubrimiento / expansión**

  Subgrafo que contiene todos los vértices y sea libre

- **Árbol de recubrimiento / expansión de coste mínimo**

  Árbol de recubrimiento cuya suma de aristas es mínima


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

Algoritmo Kruskal

Selección de arista

- Menor peso
- No seleccionada
- No debe formar ciclo

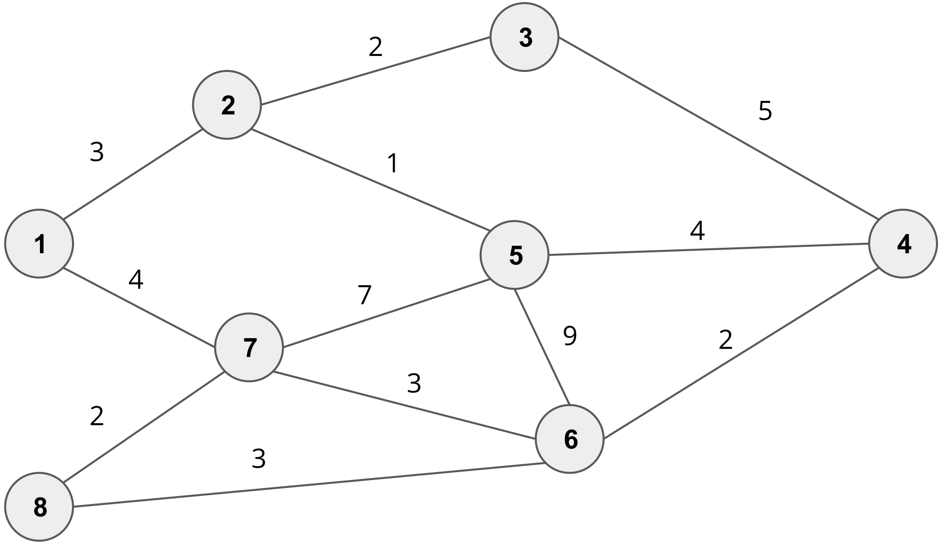
Algoritmo Kruskal

Universidad de Alcalá

```
def Kruskal(grafo):

    ordenadas = ordena_aristas(grafo.aristas)
    solucion = []

    for a in ordenadas:
        if not is_bucle(solucion, a):
            solucion.append(a)

    return solucion
```

Algoritmo Kruskal

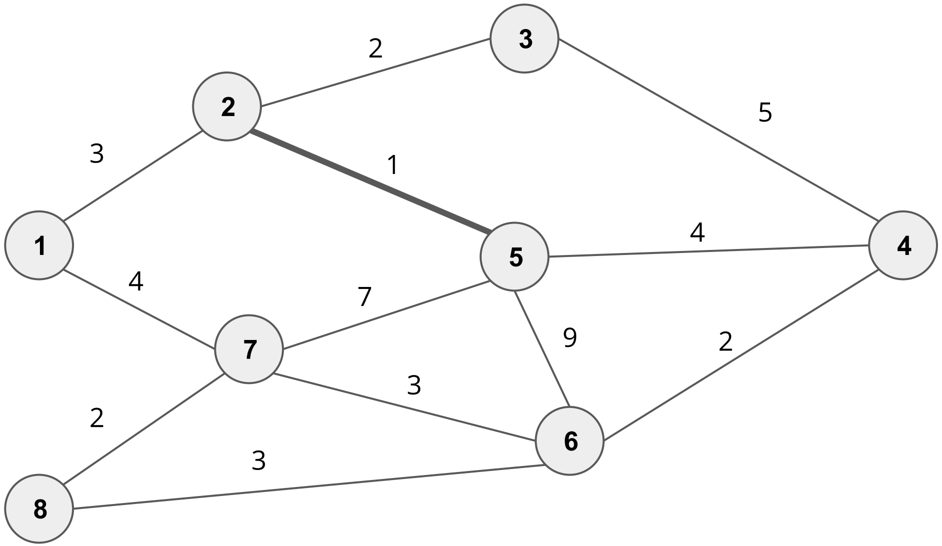
Universidad de Alcalá



i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}

i → Paso
A → Arista considerada
CC → Componentes conexas

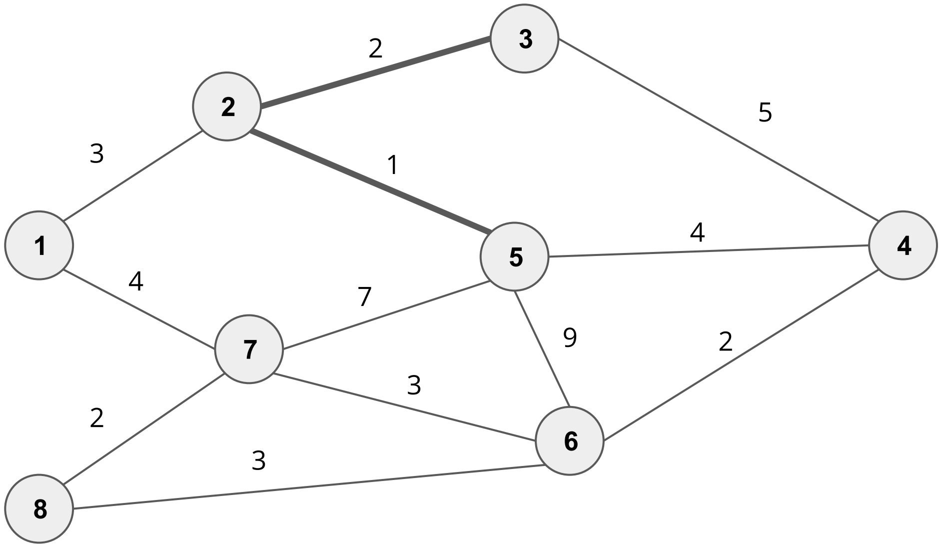
Algoritmo Kruskal



i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}
1	2-5	{2,5},{1}, {3}, {4}, {6}, {7}, {8}

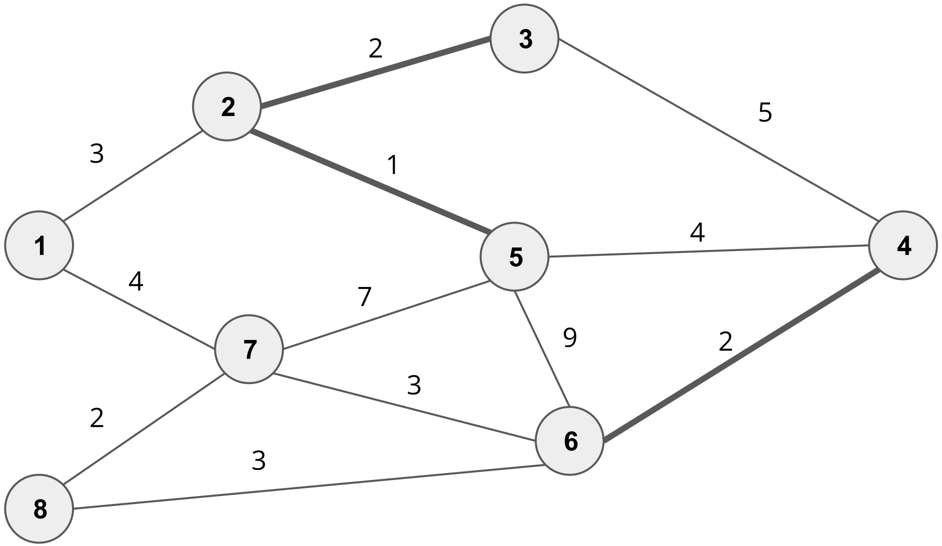
Algoritmo Kruskal

Universidad de Alcalá



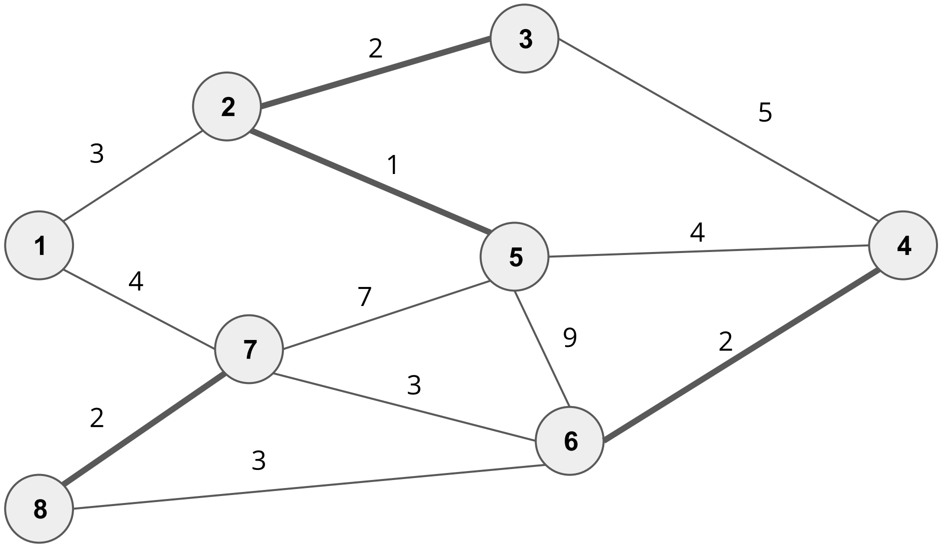
i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}
1	2-5	{2,5},{1}, {3}, {4}, {6}, {7}, {8}
2	2-3	{2,3,5},{1}, {3}, {4}, {6}, {7}, {8}

Algoritmo Kruskal



i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}
1	2-5	{2,5},{1}, {3}, {4}, {6}, {7}, {8}
2	2-3	{2,3,5},{1}, {3}, {4}, {6}, {7}, {8}
3	6-4	{2,3,5},{1}, {3}, {4, 6}, {7}, {8}

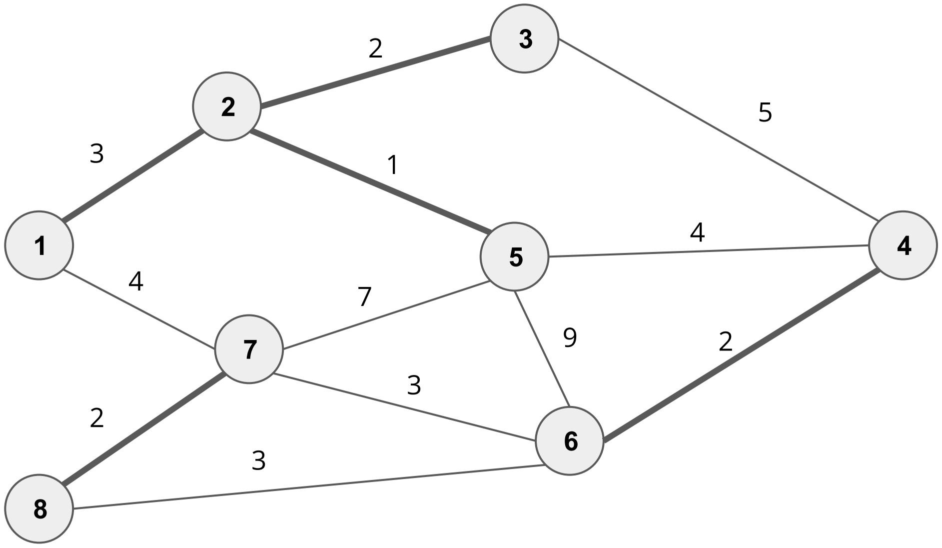
Algoritmo Kruskal



i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}
1	2-5	{2,5},{1}, {3}, {4}, {6}, {7}, {8}
2	2-3	{2,3,5},{1}, {3}, {4}, {6}, {7}, {8}
3	6-4	{2,3,5},{1}, {3}, {4, 6}, {7}, {8}
4	7-8	{2,3,5},{1}, {3}, {4, 6}, {7, 8}

Algoritmo Kruskal

Universidad de Alcalá

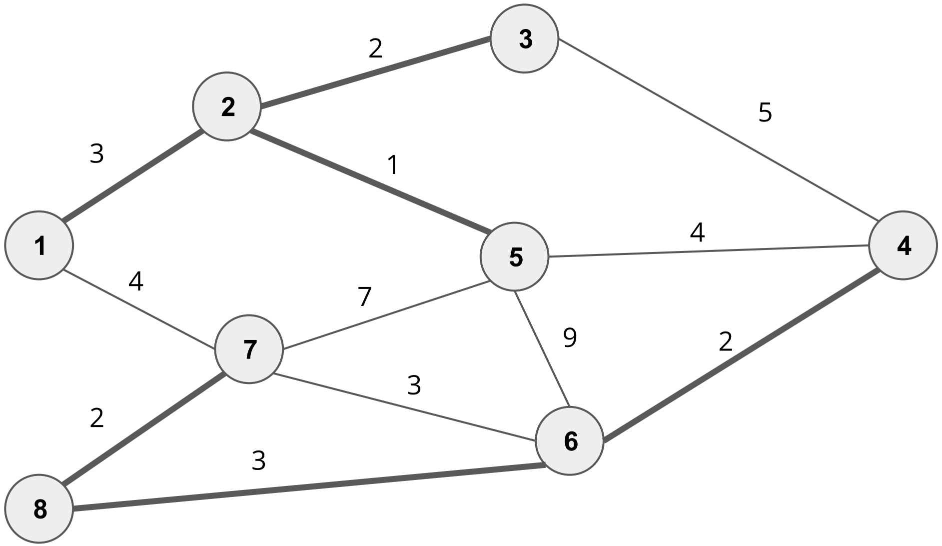


i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}
1	2-5	{2,5},{1}, {3}, {4}, {6}, {7}, {8}
2	2-3	{2,3,5},{1}, {3}, {4}, {6}, {7}, {8}
3	6-4	{2,3,5},{1}, {3}, {4, 6}, {7}, {8}
4	7-8	{2,3,5},{1}, {3}, {4, 6}, {7, 8}
5	1-2	{1,2,3,5}, {3}, {4, 6}, {7, 8}

javier@junquera.io (UAH 2023)

Algoritmo Kruskal

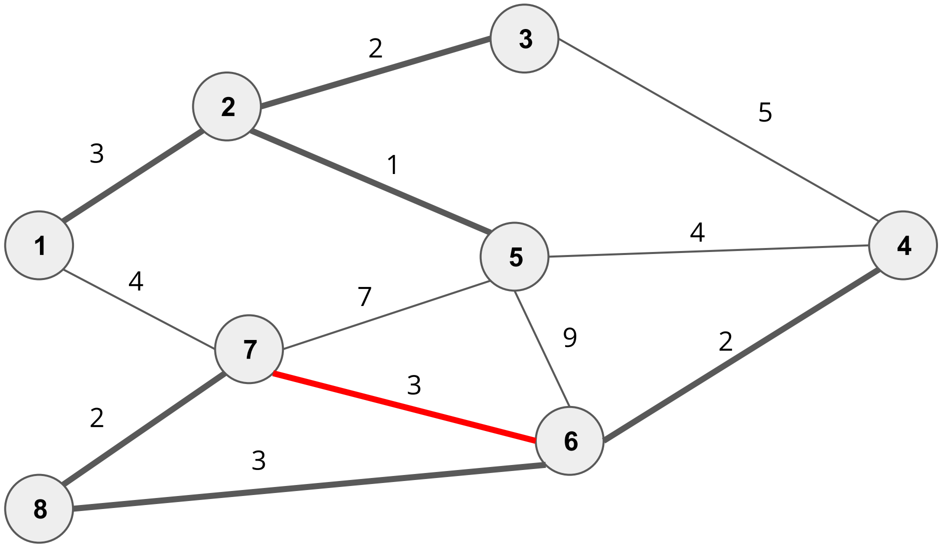
Universidad de Alcalá



i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}
1	2-5	{2,5},{1}, {3}, {4}, {6}, {7}, {8}
2	2-3	{2,3,5},{1}, {3}, {4}, {6}, {7}, {8}
3	6-4	{2,3,5},{1}, {3}, {4, 6}, {7}, {8}
4	7-8	{2,3,5},{1}, {3}, {4, 6}, {7, 8}
5	1-2	{1,2,3,5}, {3}, {4, 6}, {7, 8}
6	8-6	{1,2,3,5}, {3}, {4, 6, 7, 8}

javier@junquera.io (UAH 2023)

Algoritmo Kruskal

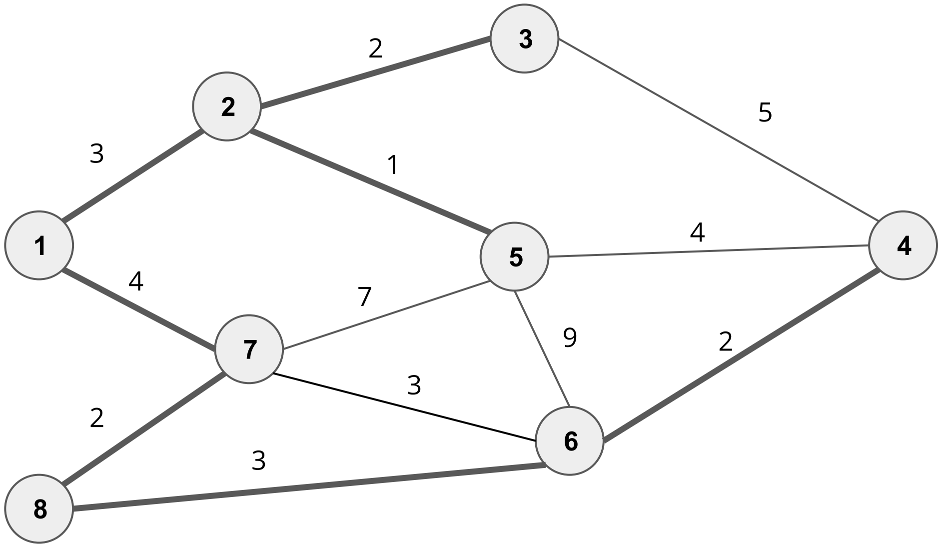


i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}
1	2-5	{2,5},{1}, {3}, {4}, {6}, {7}, {8}
2	2-3	{2,3,5},{1}, {3}, {4}, {6}, {7}, {8}
3	6-4	{2,3,5},{1}, {3}, {4, 6}, {7}, {8}
4	7-8	{2,3,5},{1}, {3}, {4, 6}, {7, 8}
5	1-2	{1,2,3,5}, {3}, {4, 6}, {7, 8}
6	8-6	{1,2,3,5}, {3}, {4, 6, 7, 8}
-	7-6	-

javier@junquera.io (UAH 2023)

Algoritmo Kruskal

Universidad de Alcalá



i	A	CC
0	-	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}
1	2-5	{2,5},{1}, {3}, {4}, {6}, {7}, {8}
2	2-3	{2,3,5},{1}, {4}, {6}, {7}, {8}
3	6-4	{2,3,5},{1}, {4, 6}, {7}, {8}
4	7-8	{2,3,5},{1}, {4, 6}, {7, 8}
5	1-2	{1,2,3,5}, {4, 6}, {7, 8}
6	8-6	{1,2,3,5}, {4, 6, 7, 8}
-	7-6	-
7	1-7	{1,2,3,4,5,6,7,8}

Eficiencia Kruskal

Universidad de Alcalá

- Elementos de G(N, A)
  - n nodos, a aristas
- Inicialización
  - Ordenar aristas por peso: O(a log(n))
  - Organizar conjuntos disjuntos: O(n)
- Voraz
  - Por cada arista (O(a))
  - Buscar mejor (O(1))

$$T(n) = O(a \log(n)) + O(a) + O(1)$$

$$T(n) \in O(a \log(n))$$

Algoritmo Prim

Universidad de Alcalá

Selección de arista

- Menor peso
- No seleccionada
- No debe formar ciclo
- Que forme árbol con seleccionadas anteriores

Algoritmo Prim

Universidad de Alcalá

```
def Prim(grafo):

    solucion = []

    aristas_by_node = {i: [] for i in grafo.nodes}
    for a in grafo.aristas:
        aristas_by_node[a.origen].append(a)
        aristas_by_node[a.fin].append(a)

    # Elijo vértice #6
    seleccionadas = [grafo.nodes[5]]
    Q = grafo.nodes[:5] + grafo.nodes[6:]

    # Ejecución del voraz
    [...]

    return solucion
```

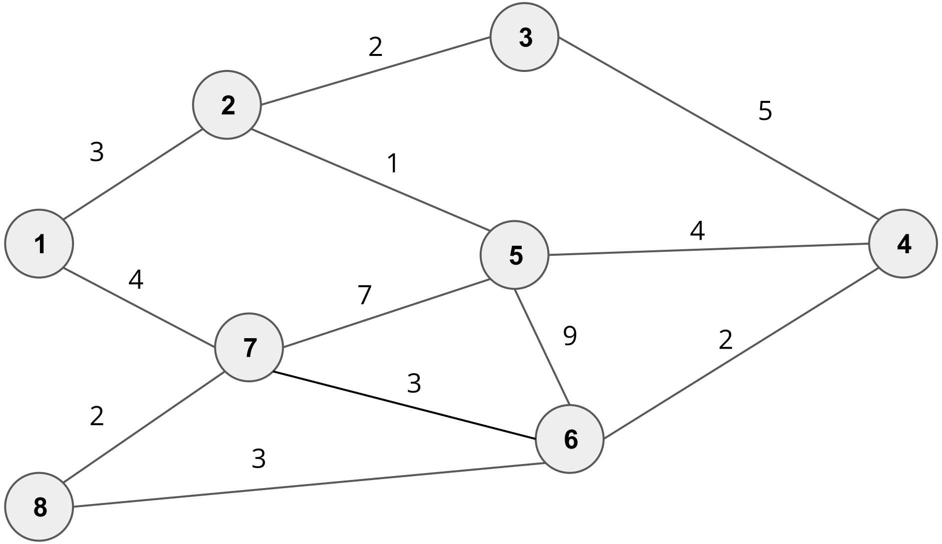
Algoritmo Prim

```
[...]

# Ejecución del voraz
while len(Q) > 0:

    # Obtención de arista minima
    aristas_min = []

    for s in seleccionadas:
        menores = ordena_aristas(aristas_by_node[s])

        for i in range(len(menores)):
            menor = menores[i]
            if not (is_bucle(solucion, menor) or ((menor.origen in seleccionadas) and (menor.fin in seleccionadas))):
                aristas_min.append(menor)

    arista_min = ordena_aristas(aristas_min)[0]

    # Eliminar nuevo nodo de la cola
    next_node = arista_min.origen if (arista_min.fin in seleccionadas) else arista_min.fin
    for i in range(len(Q)):
        if Q[i] == next_node:
            Q = Q[:i] + Q[i+1:]
            break

    solucion.append(arista_min)
    seleccionadas.append(next_node)

[...]
```

Algoritmo Prim

Universidad de Alcalá

i	A	Árbol
0	-	-

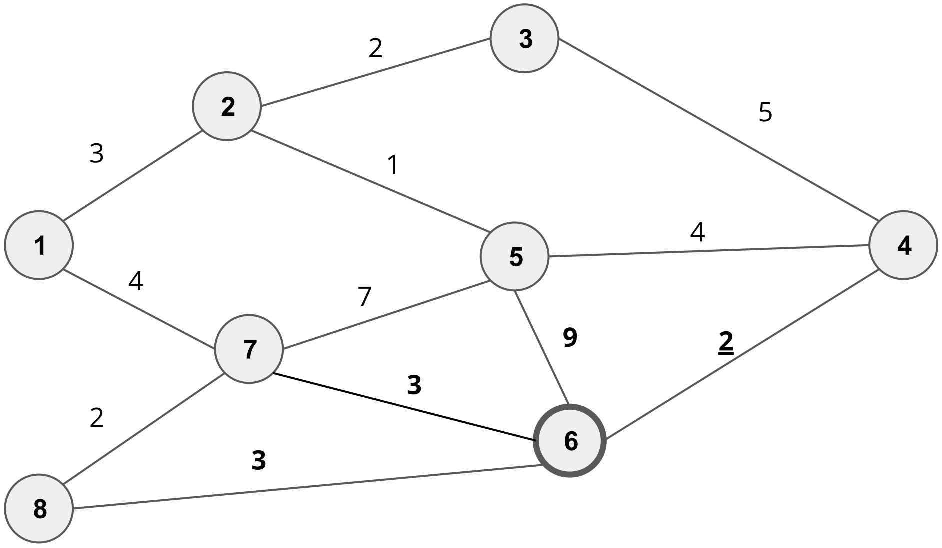


Primero elegimos un nodo al azar (e.g., el nodo 6)

Algoritmo Prim

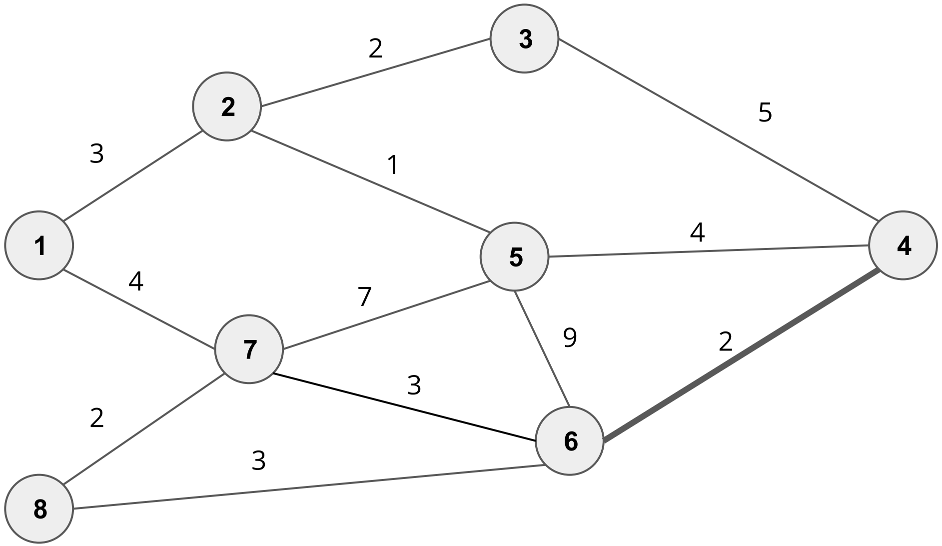
Universidad de Alcalá

i	A	Árbol
0	-	{6}



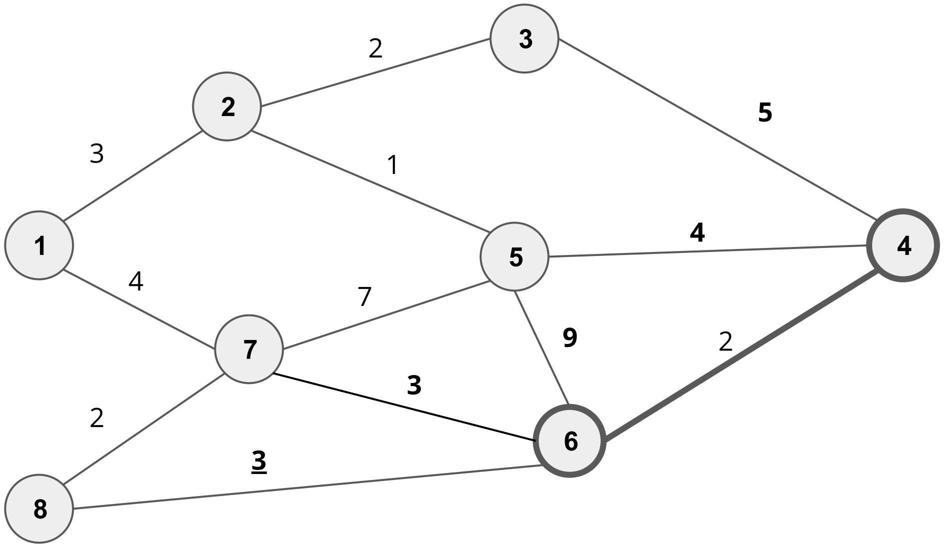
Elegimos la arista de menor peso

Algoritmo Prim



i	A	Árbol
0	-	{6}
1	6,4	{4,6}

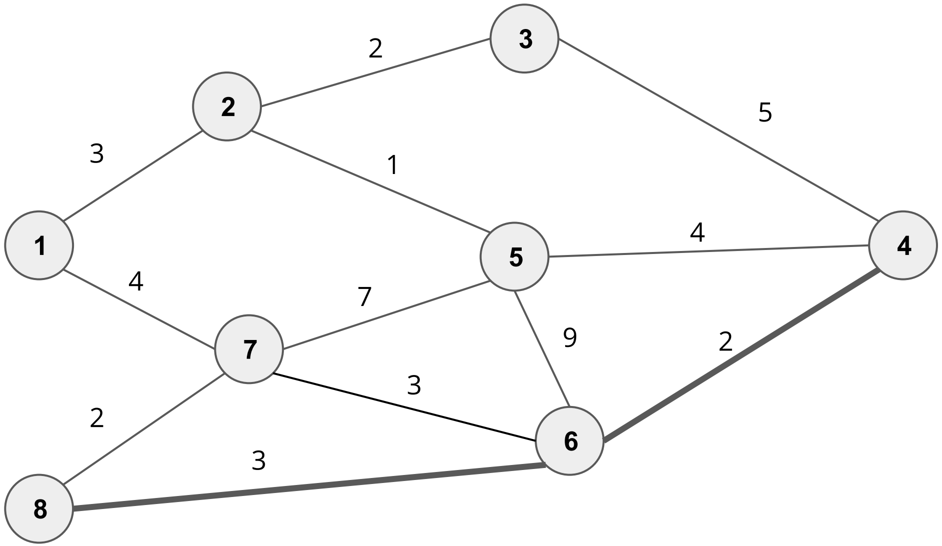
Algoritmo Prim



i	A	Árbol
0	-	{6}
1	6,4	{4,6}

Elegimos la arista de menor peso, desde los nodos conectados

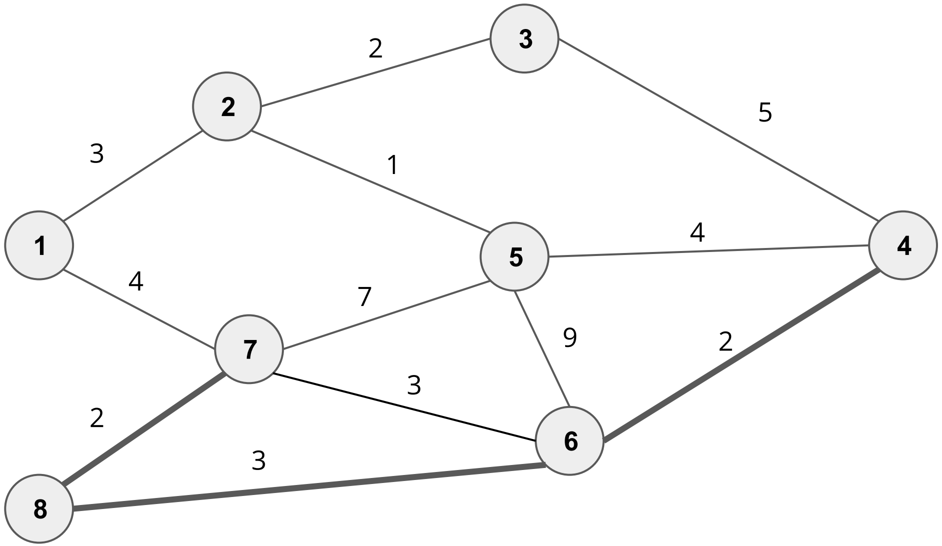
Algoritmo Prim



i	A	Árbol
0	-	{6}
1	6,4	{4,6}
2	6,8	{4,6,8}

Algoritmo Prim

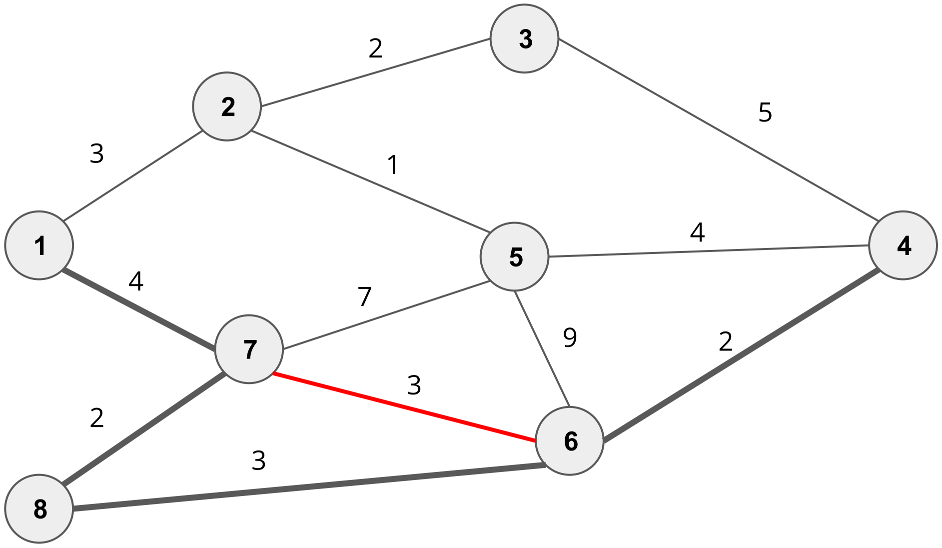
Universidad de Alcalá



i	A	Árbol
0	-	{6}
1	6,4	{4,6}
2	6,8	{4,6,8}
3	8,7	{4,6,7,8}

Algoritmo Prim

Universidad de Alcalá

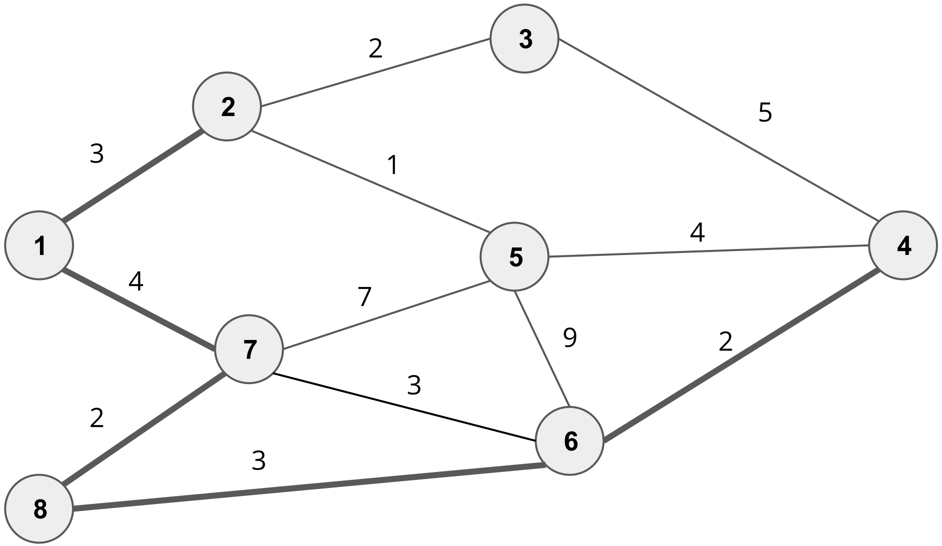


i	A	Árbol
0	-	{6}
1	6,4	{4,6}
2	6,8	{4,6,8}
3	8,7	{4,6,7,8}
4	7,1	{1,4,6,7,8}

Elegimos la arista de menor peso, desde los nodos conectados, <u>y que no forme bucle</u>

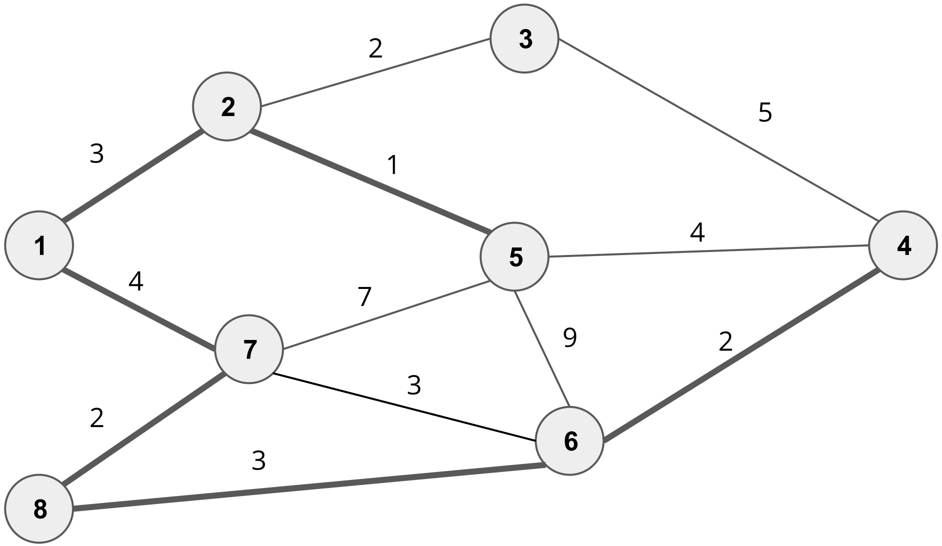
Algoritmo Prim

Universidad de Alcalá



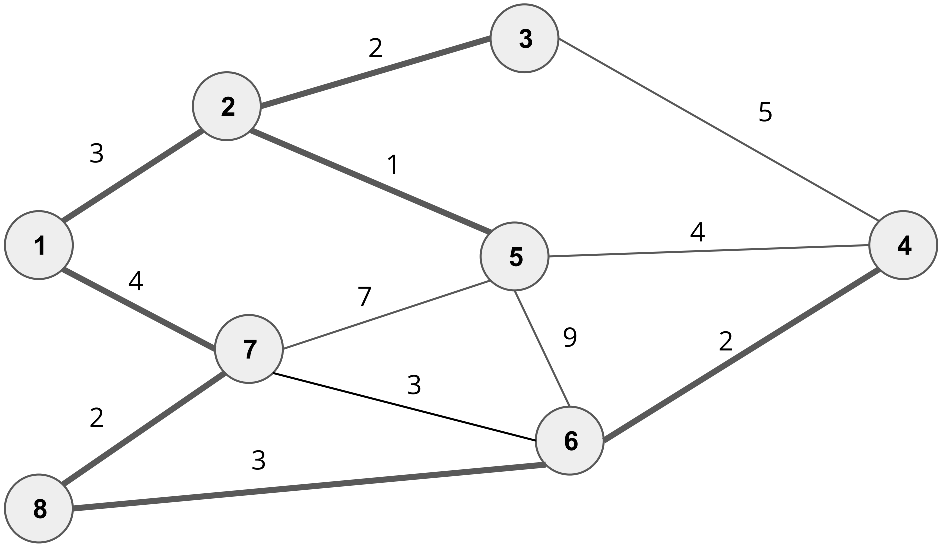
i	A	Árbol
0	-	{6}
1	6,4	{4,6}
2	6,8	{4,6,8}
3	8,7	{4,6,7,8}
4	7,1	{1,4,6,7,8}
5	1,2	{1,2,4,6,7,8}

Algoritmo Prim



i	A	Árbol
0	-	{6}
1	6,4	{4,6}
2	6,8	{4,6,8}
3	8,7	{4,6,7,8}
4	7,1	{1,4,6,7,8}
5	1,2	{1,2,4,6,7,8}
6	2,5	{1,2,4,5,6,7,8}

Algoritmo Prim



i	A	Árbol
0	-	{6}
1	6,4	{4,6}
2	6,8	{4,6,8}
3	8,7	{4,6,7,8}
4	7,1	{1,4,6,7,8}
5	1,2	{1,2,4,6,7,8}
6	2,5	{1,2,4,5,6,7,8}
7	2,3	{1,2,3,4,5,6,7,8}

Eficiencia Prim

Universidad de Alcalá

- Elementos de G(N, A)
  - n nodos, a aristas
- Inicialización
  - Elección de un nodo: O(1)
- Voraz
  - Por cada nodo seleccionado (O(n))
  - Buscar mejor arista (O(a log(a)))

$$T(n) = O(n + a \log(a))$$

Estructura de datos

Universidad de Alcalá

- **Grafo denso**

  Casi todos los nodos están conectados entre sí (i.e., número de aristas ~ número máximo de aristas)

- **Grafo disperso**

  Pocos nodos están conectados entre sí (i.e., hay pocas aristas)

javier@junquera.io (UAH 2023)

Kruskal v. Prim

Universidad de Alcalá

En algoritmia es importante la implementación, pero también el modelo de datos que utilizamos:

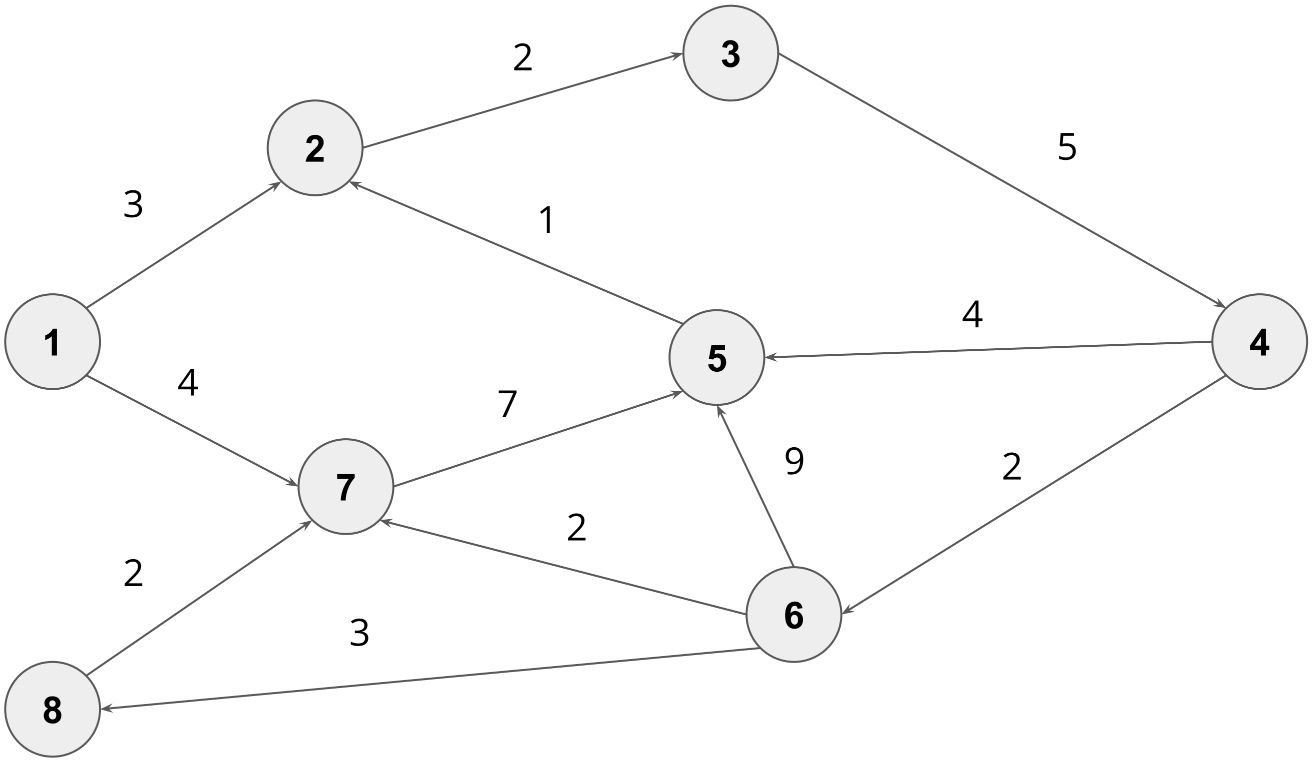
	Kruskal	Prim
Grafo denso ($a \sim n^2$)	$O(n^2 \log(n))$	$O(n^2)$
Grafo disperso ($a \sim n$)	$O(n \log(n))$	$O(n \log(n))$

Caminos mínimos

javier@junquera.io (UAH 2023)

Grafos dirigidos

Universidad de Alcalá



Algoritmo de Dijkstra

Universidad de Alcalá

Algoritmo de búsqueda de caminos mínimos en un grafo dirigido:

1. **Inicialización**

   Selección de origen, destino e inicialización de matriz de resultados

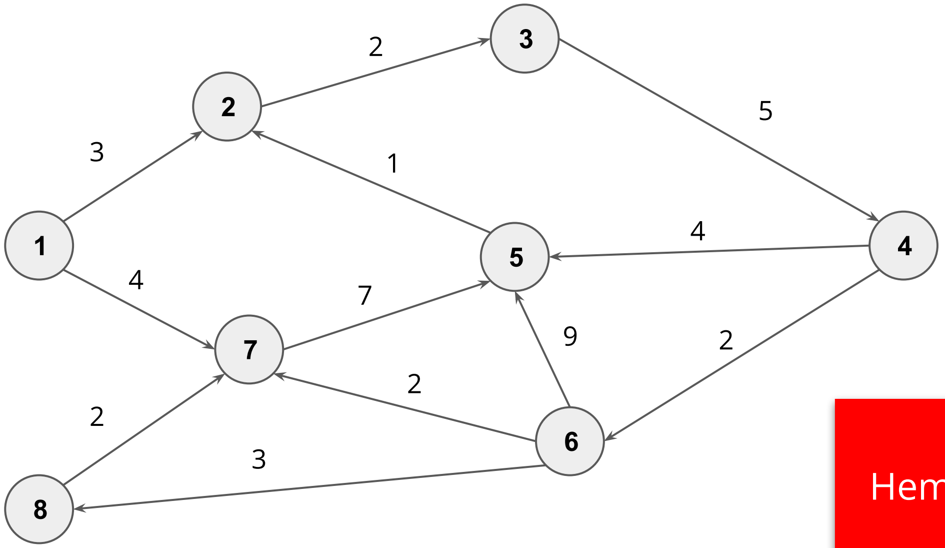
2. **Ejecución**

   Partiendo del origen, seleccionar camino mínimo a siguiente vértice y almacenar distancia acumulada

https://gitlab.com/-/snippets/2092361

Algoritmo de Dijkstra

Universidad de Alcalá

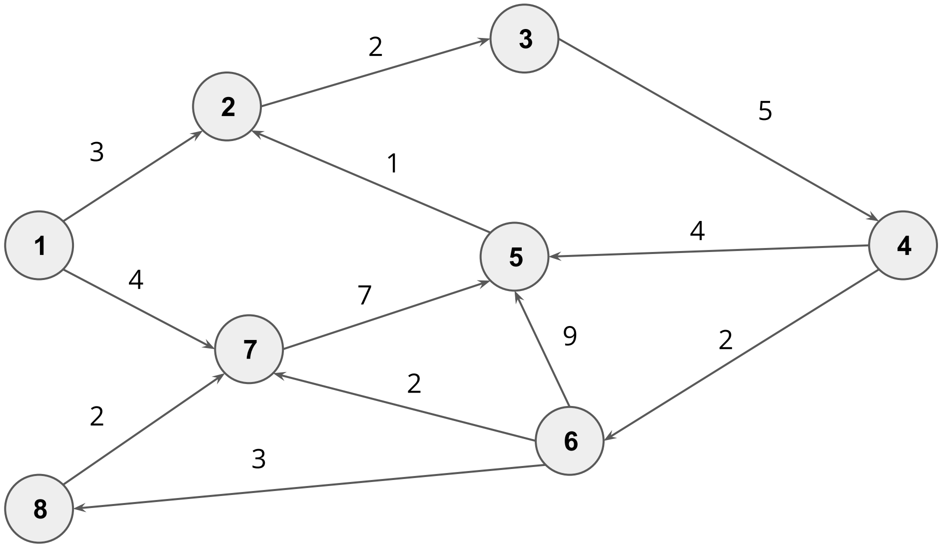


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x

¡OJO! Grafos dirigidos

Hemos metido un pequeño cambio en 6 → 7 (cambio de peso, de 3 a 2)

Algoritmo de Dijkstra

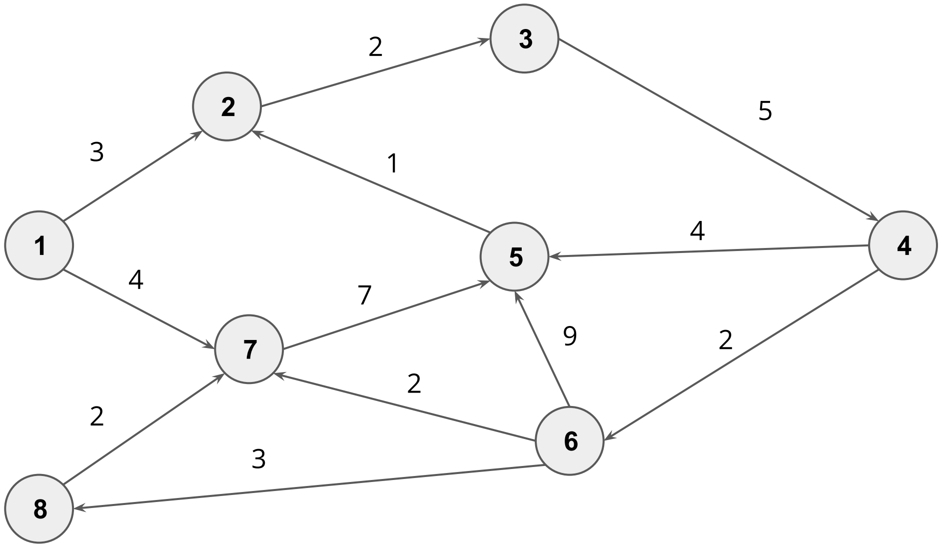


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	+3	x	x	x	x	4	x

Algoritmo de Dijkstra

Universidad de Alcalá

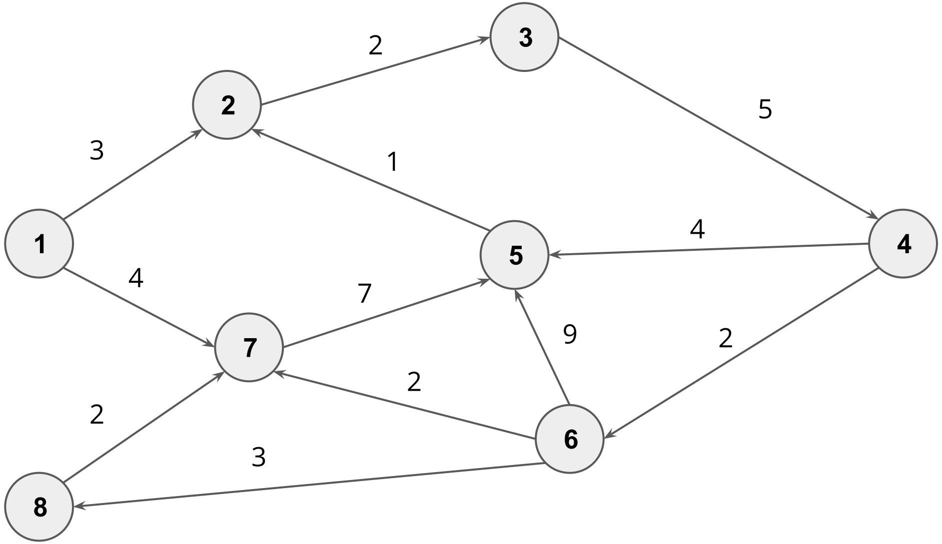
Distancia a 1



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	+3	x	x	x	x	4	x

Algoritmo de Dijkstra

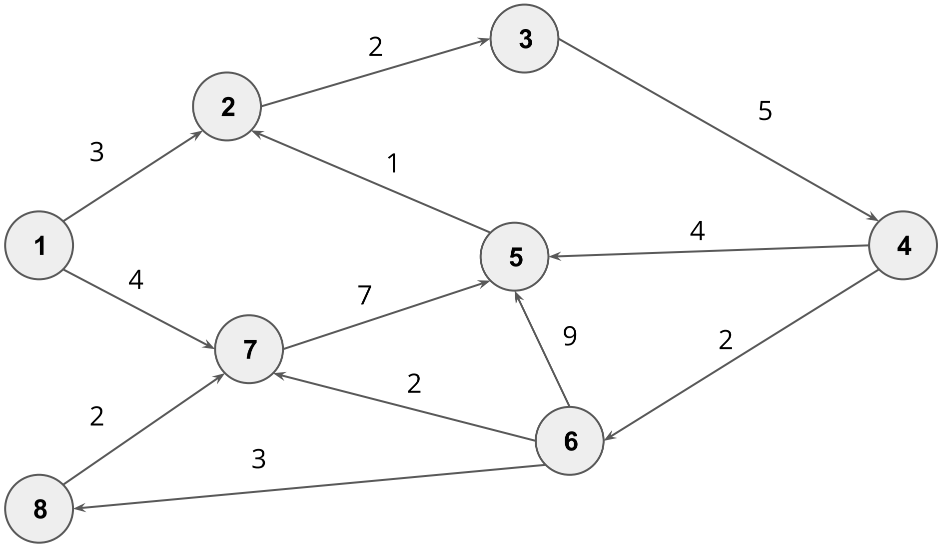
Universidad de Alcalá



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	+2	x	x	x	4	x

Algoritmo de Dijkstra

Universidad de Alcalá

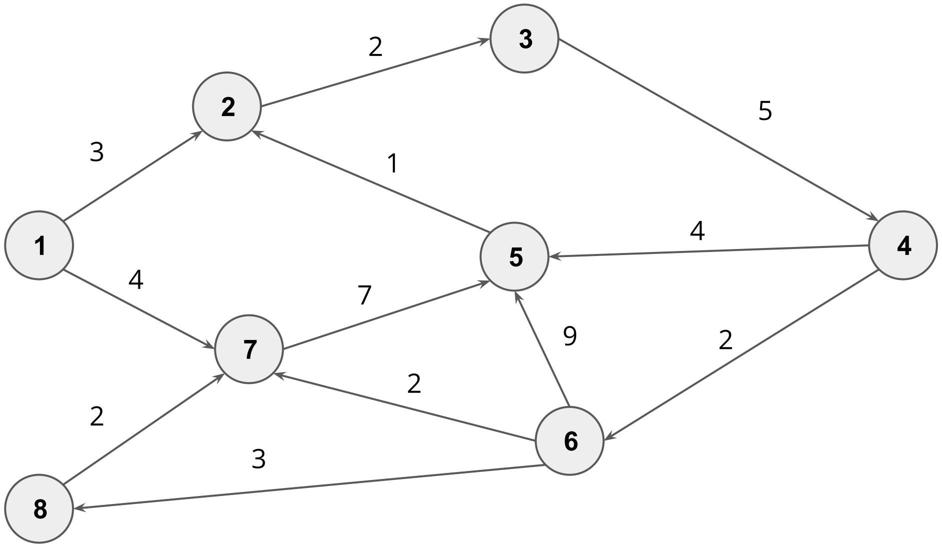


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	**5**	x	x	x	4	x
2	3	3	5	+5	x	x	4	x

Algoritmo de Dijkstra

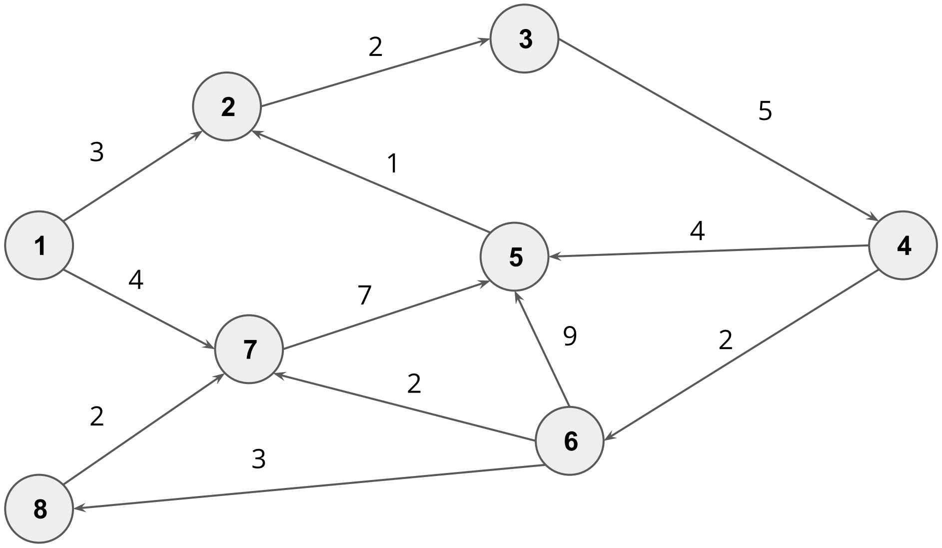
Universidad de Alcalá

Elegimos el más pequeño



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	**5**	x	x	x	4	x
2	3	3	5	**10**	x	x	4	x
3	4	3	5	10	+4	**+2**	4	x

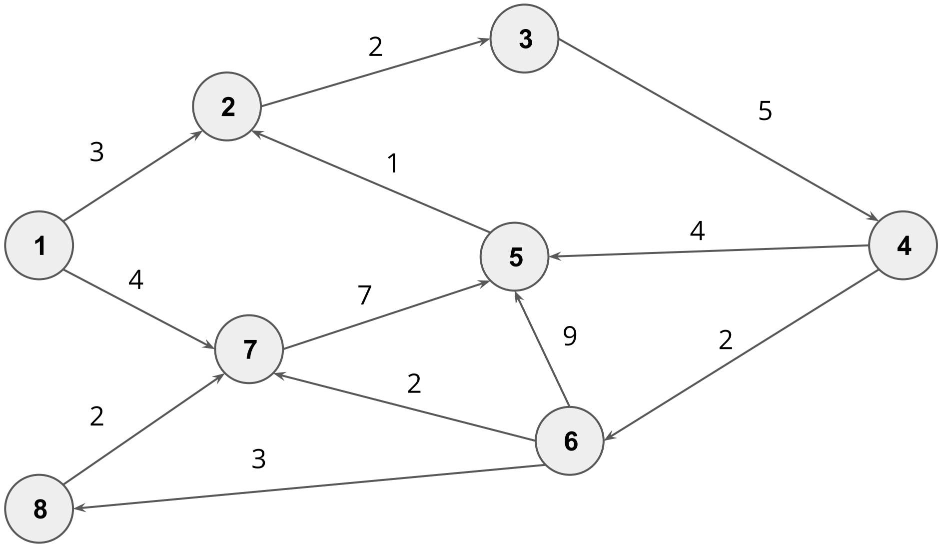
Algoritmo de Dijkstra



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	**5**	x	x	x	4	x
2	3	3	5	**10**	x	x	4	x
3	4	3	5	10	14	**12**	4	x
4	6	3	5	10	14	12	+2	+3

Algoritmo de Dijkstra

Guardamos el valor más bajo

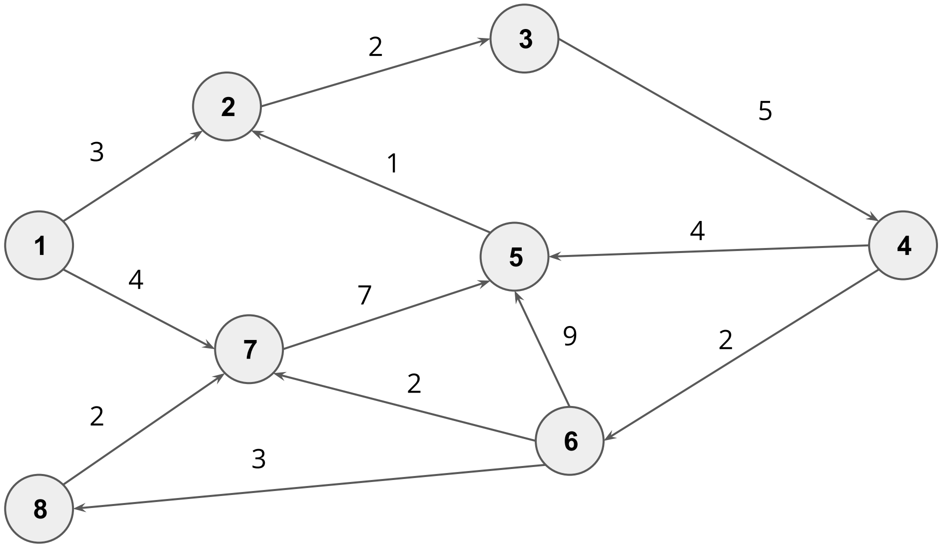


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	**5**	x	x	x	4	x
2	3	3	5	**10**	x	x	4	x
3	4	3	5	10	14	**12**	4	x
4	6	3	5	10	14	12	4	15
5	7	3	5	10	+7	12	4	15

javier@junquera.io (UAH 2023)

Algoritmo de Dijkstra

Guardamos el valor más bajo

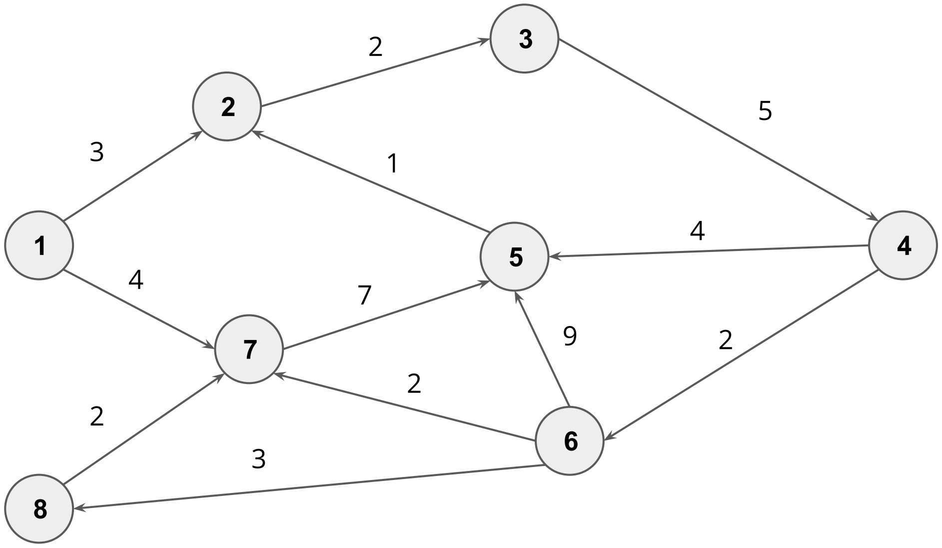


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	**5**	x	x	x	4	x
2	3	3	5	**10**	x	x	4	x
3	4	3	5	10	14	**12**	4	x
4	6	3	5	10	14	12	4	15
5	7	3	5	10	11	12	4	15
6	5	+1	5	10	11	12	4	15

javier@junquera.io (UAH 2023)

Algoritmo de Dijkstra

Guardamos el valor más bajo



Seguimos con el camino que nos quede

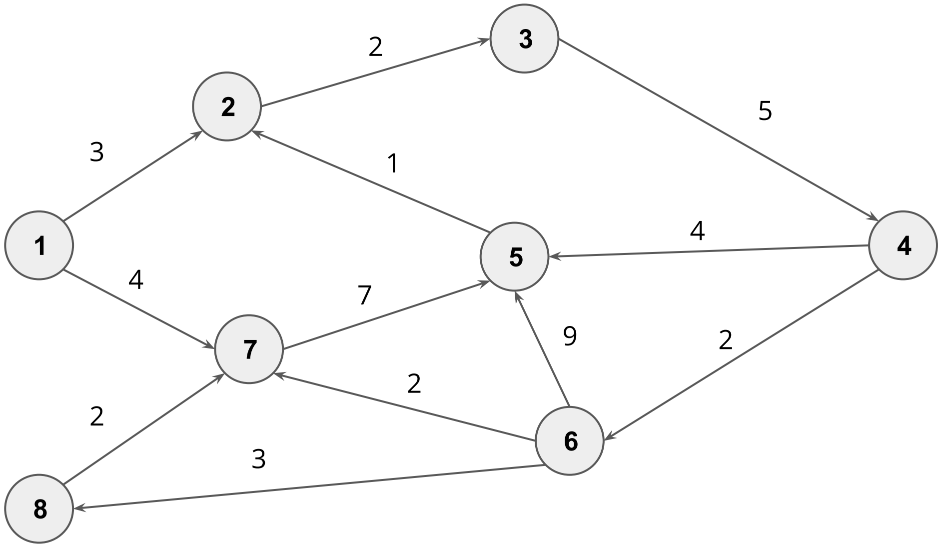
i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	**5**	x	x	x	4	x
2	3	3	5	**10**	x	x	4	x
3	4	3	5	10	14	**12**	4	x
4	6	3	5	10	14	12	4	15
5	7	3	5	10	11	12	4	15
6	5	3	5	10	11	12	4	15
7	8	3	5	10	11	12	+2	15

javier@junquera.io (UAH 2023)

Algoritmo de Dijkstra

Universidad de Alcalá

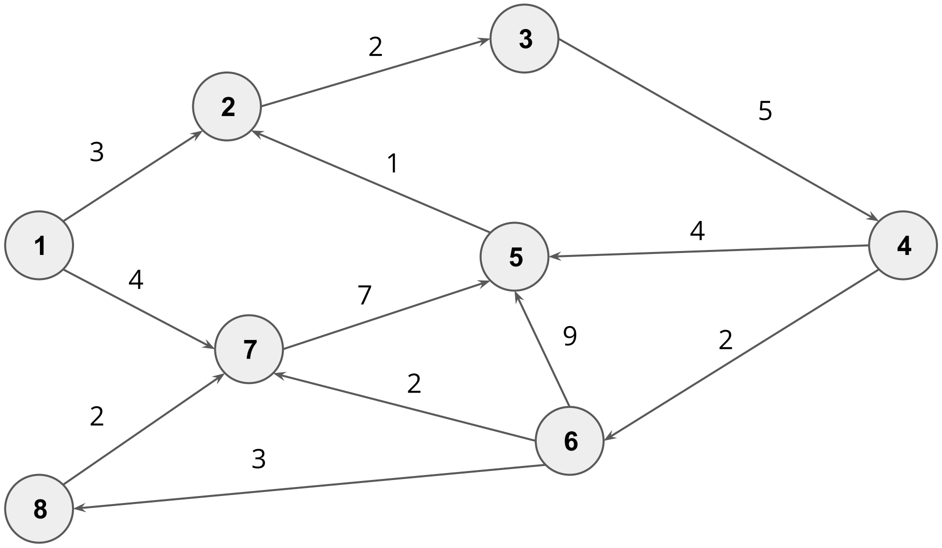
Guardamos el valor más bajo



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	**5**	x	x	x	4	x
2	3	3	5	**10**	x	x	4	x
3	4	3	5	10	14	**12**	4	x
4	6	3	5	10	14	12	4	15
5	7	3	5	10	11	12	4	15
6	5	3	5	10	11	12	4	15
7	8	3	5	10	11	12	4	15

Algoritmo de Dijkstra

Camino más corto desde 1

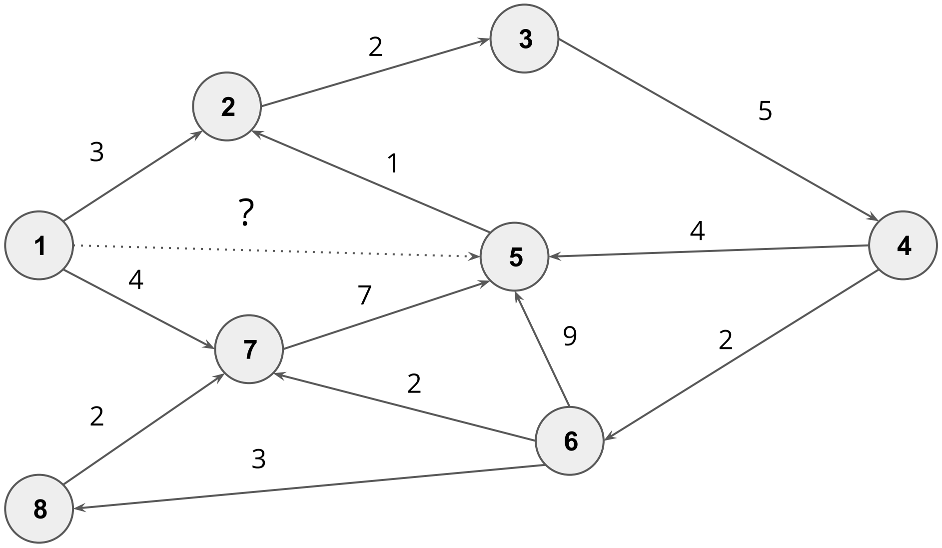


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	**4**	x
1	2	3	**5**	x	x	x	4	x
2	3	3	5	**10**	x	x	4	x
3	4	3	5	10	14	**12**	4	x
4	6	3	5	10	14	12	4	**15**
5	7	3	5	10	**11**	12	4	15
6	5	3	5	10	11	12	4	15
7	8	3	5	10	11	12	4	15

javier@junquera.io (UAH 2023)

Algoritmo de Dijkstra

¿ Mejor camino 1→ 5 ?

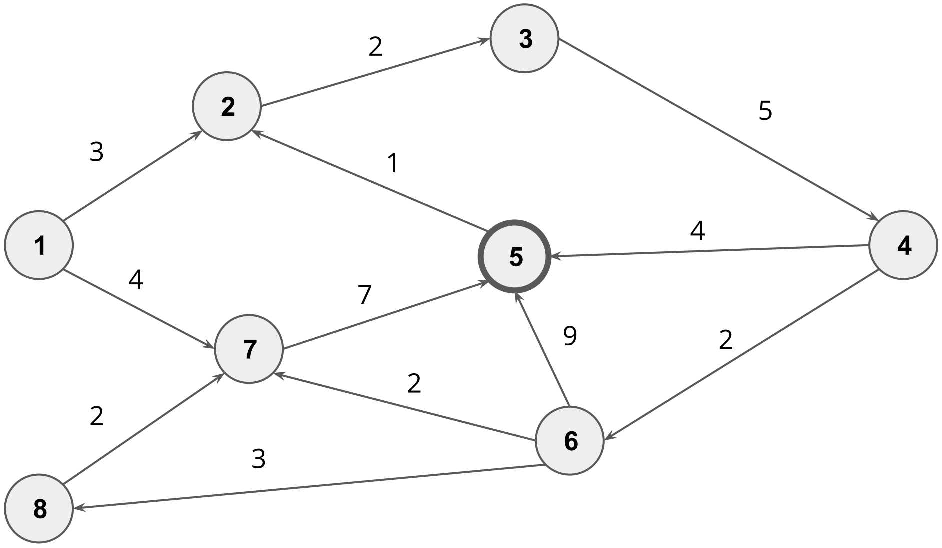


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	3	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	3	3	5	10	x	x	4	x
3	4	3	5	10	14	12	4	x
4	6	3	5	10	14	12	4	15
5	7	3	5	10	11	12	4	15
6	5	3	5	10	11	12	4	15
7	8	3	5	10	11	12	4	15

javier@junquera.io (UAH 2023)

Algoritmo de Dijkstra

a) Vamos a 5

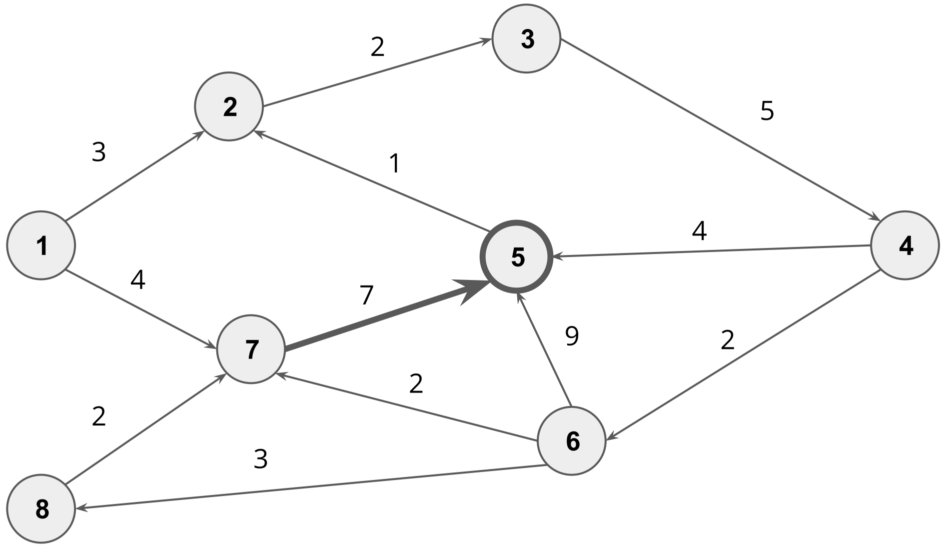


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	3	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	3	3	5	10	x	x	4	x
3	4	3	5	10	14	12	4	x
4	6	3	5	10	14	12	4	15
5	7	3	5	10	11	12	4	15
6	**5**	3	5	10	11	12	4	15
7	8	3	5	10	11	12	4	15

javier@junquera.io (UAH 2023)

Algoritmo de Dijkstra

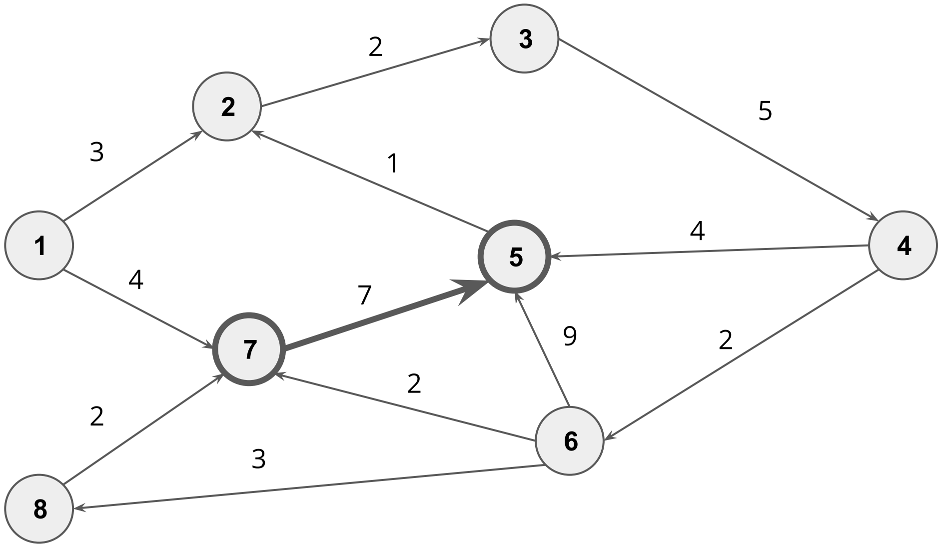
b) ¿Cuándo alcanza min? En 7



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	3	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	3	3	5	10	x	x	4	x
3	4	3	5	10	14	12	4	x
4	6	3	5	10	14	12	4	15
5	**7**	3	5	10	11	12	4	15
6	**5**	3	5	10	**11**	12	4	15
7	8	3	5	10	11	12	4	15

Algoritmo de Dijkstra

a) Vamos a 7

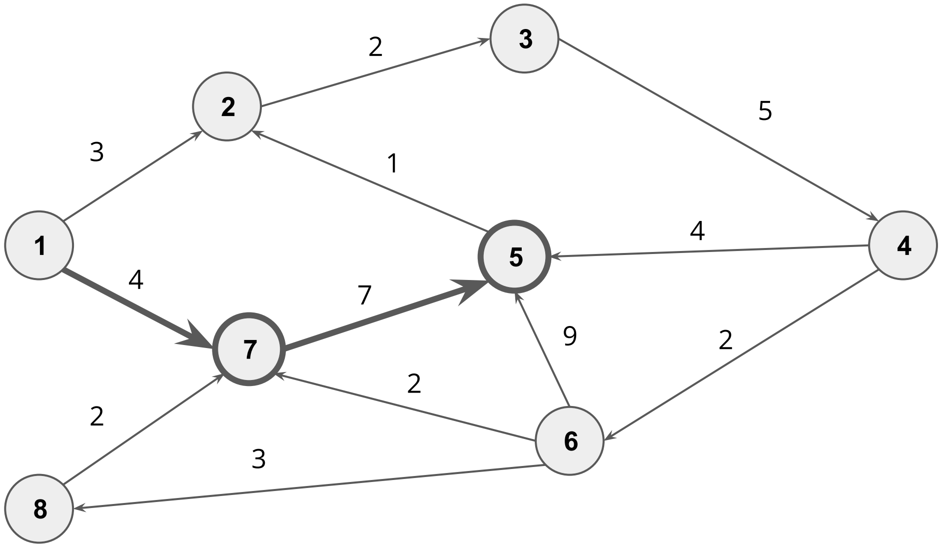


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	3	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	3	3	5	10	x	x	4	x
3	4	3	5	10	14	12	4	x
4	6	3	5	10	14	12	4	15
5	**7**	3	5	10	11	12	4	15
6	5	3	5	10	11	12	4	15
7	8	3	5	10	11	12	4	15

javier@junquera.io (UAH 2023)

Algoritmo de Dijkstra

b) ¿Cuándo alcanza min? En 1

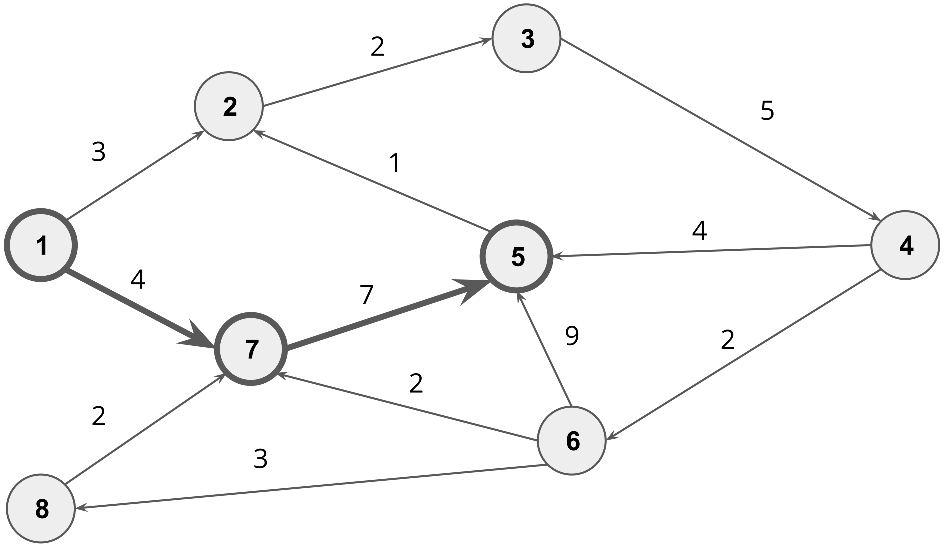


i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	**1**	3	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	3	3	5	10	x	x	4	x
3	4	3	5	10	14	12	4	x
4	6	3	5	10	14	12	4	15
5	**7**	3	5	10	11	12	**4**	15
6	5	3	5	10	11	12	4	15
7	8	3	5	10	11	12	4	15

javier@junquera.io (UAH 2023)

Algoritmo de Dijkstra

¡ 1 → 7 → 5 ! (con coste 11)



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	**1**	3	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	3	3	5	10	x	x	4	x
3	4	3	5	10	14	12	4	x
4	6	3	5	10	14	12	4	15
5	**7**	3	5	10	11	12	4	15
6	**5**	3	5	10	**11**	12	4	15
7	8	3	5	10	11	12	4	15

javier@junquera.io (UAH 2023)

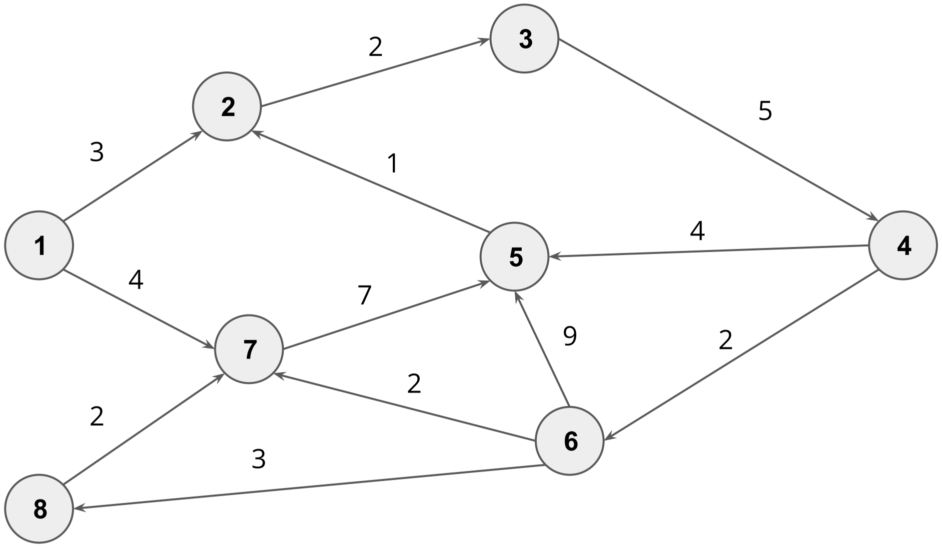
Eficiencia de Dijkstra

Con A, número de aristas; y V, número de vértices; la complejidad del algoritmo de Dijkstra es:

$$T(n) \in O(A + V \log V)$$

Algoritmo de Dijkstra

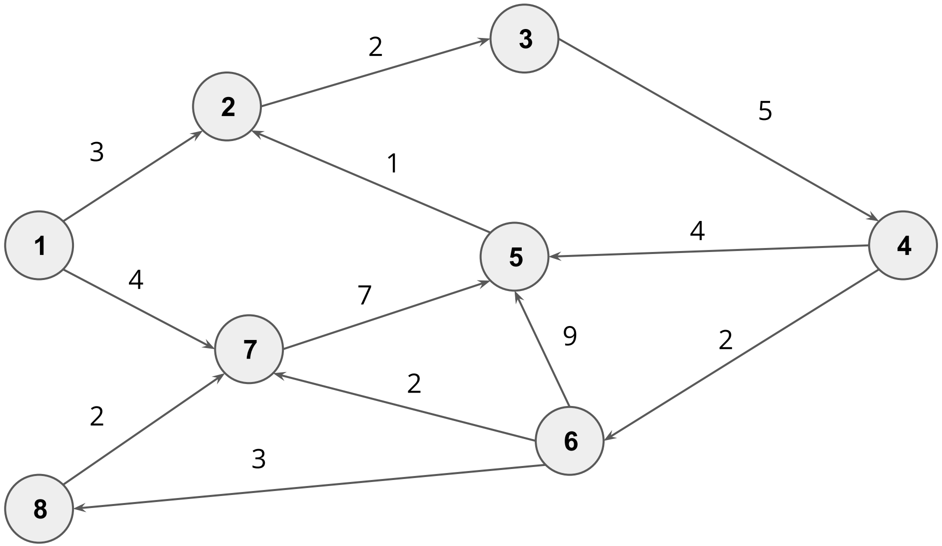
Universidad de Alcalá



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	**5**	x	x	x	**4**	x

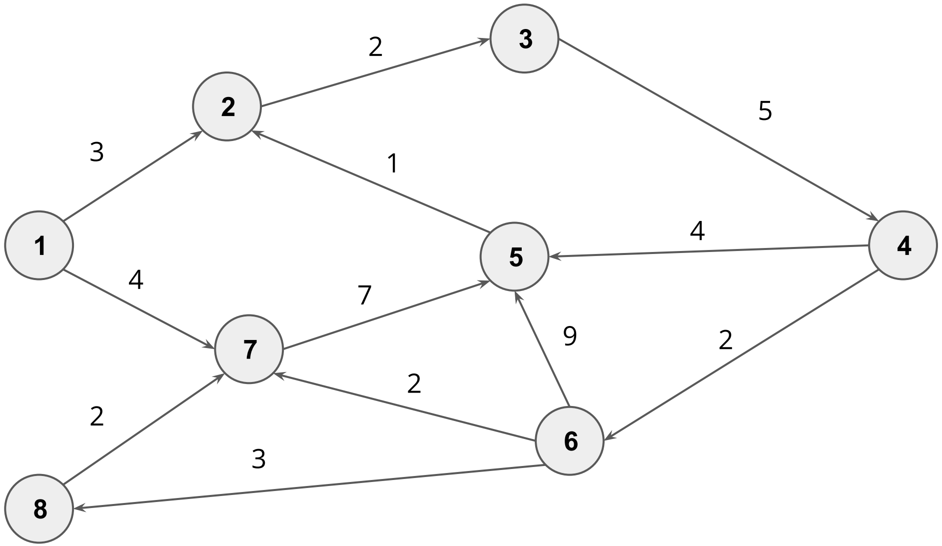
Pila v. Lista

Algoritmo de Dijkstra



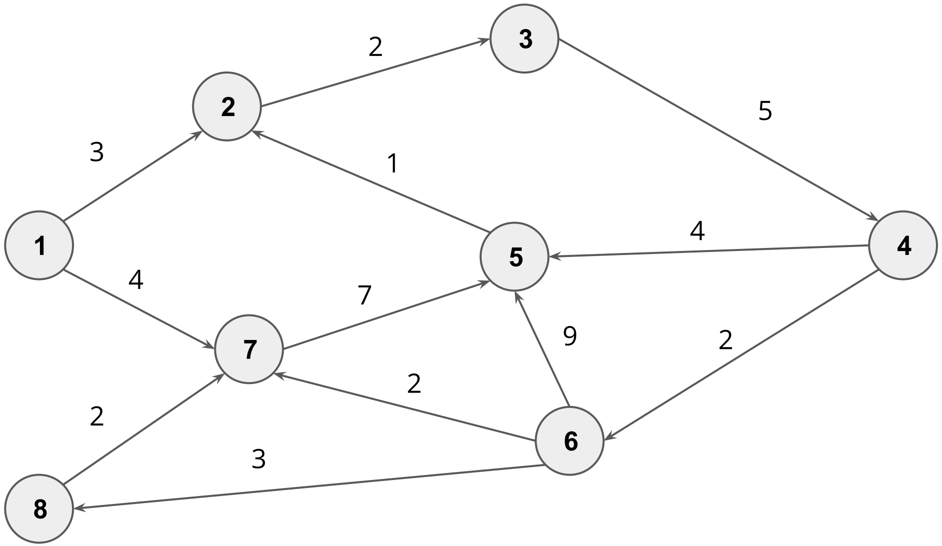
i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	5	x	x	x	**4**	x
2	7	3	5	x	+7	x	4	x

Algoritmo de Dijkstra



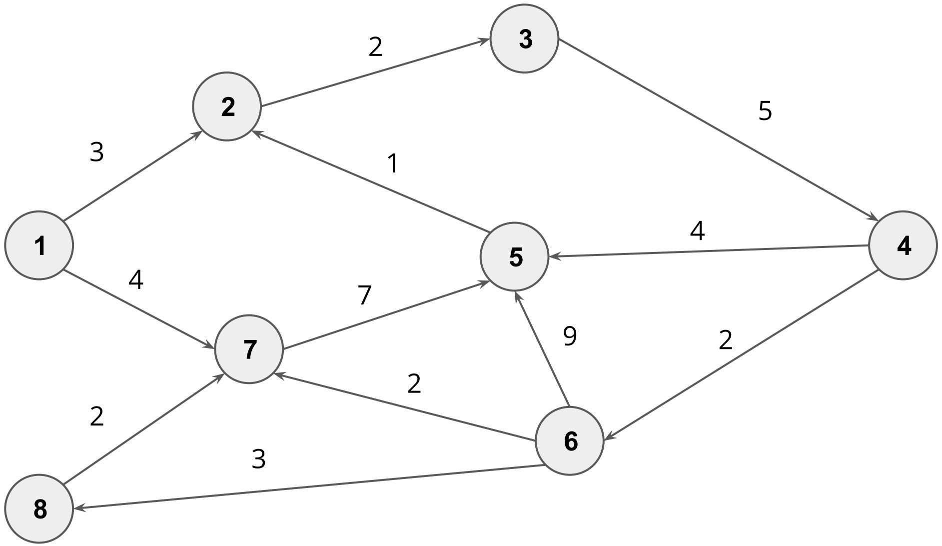
i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	7	3	**5**	x	11	x	4	x
3	3	3	5	+5	11	x	4	x

Algoritmo de Dijkstra



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	7	3	5	x	11	x	4	x
3	3	3	5	**10**	11	x	4	x
4	4	3	5	10	11	+2	4	x

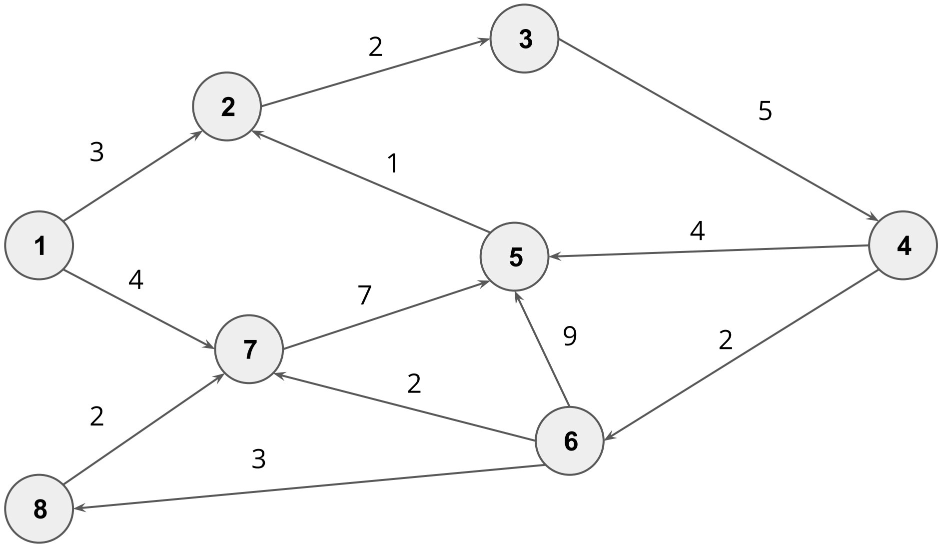
Algoritmo de Dijkstra



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	7	3	5	x	11	x	4	x
3	3	3	5	10	11	x	4	x
4	4	3	5	10	**11**	12	4	x
5	5	3	5	10	11	12	4	x

Algoritmo de Dijkstra

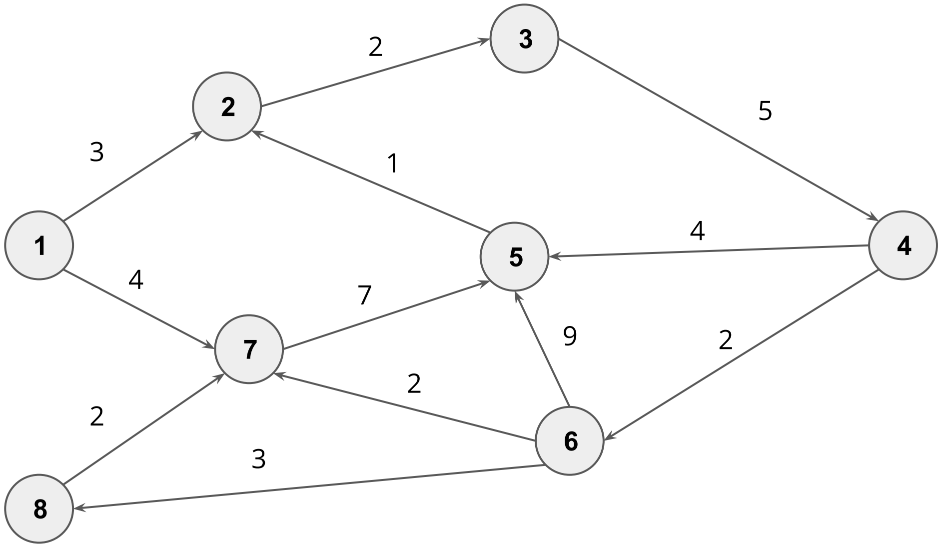
Universidad de Alcalá



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	7	3	5	x	11	x	4	x
3	3	3	5	10	11	x	4	x
4	4	3	5	10	11	12	4	x
5	5	3	5	10	11	**12**	4	x
6	6	3	5	10	11	12	4	+3

Algoritmo de Dijkstra

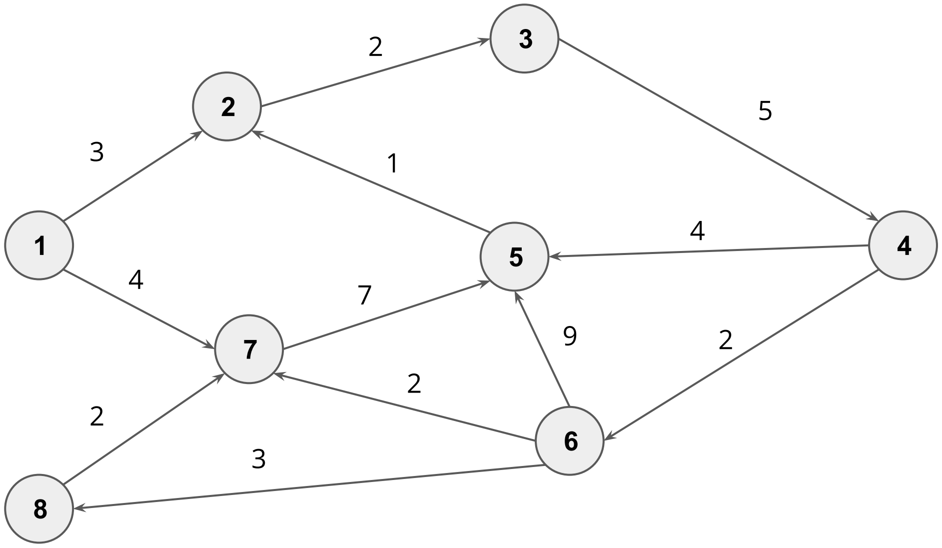
Universidad de Alcalá



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	7	3	5	x	11	x	4	x
3	3	3	5	10	11	x	4	x
4	4	3	5	10	11	12	4	x
5	5	3	5	10	11	12	4	x
6	6	3	5	10	11	12	4	15

Algoritmo de Dijkstra

Universidad de Alcalá



i	S	D2	D3	D4	D5	D6	D7	D8
-1	-	x	x	x	x	x	x	x
0	1	**3**	x	x	x	x	4	x
1	2	3	5	x	x	x	4	x
2	7	3	5	x	11	x	4	x
3	3	3	5	10	11	x	4	x
4	4	3	5	10	11	12	4	x
5	5	3	5	10	11	12	4	x
6	6	3	5	10	11	12	4	15
7	8	3	5	10	11	12	4	15

javier@junquera.io (UAH 2023)

Algoritmos de planificación

Minimización de tiempo en el sistema

Universidad de Alcalá

Tenemos que resolver `n` *tareas, cada una conlleva invertir un tiempo* t_i

Si tardamos un tiempo `d` *en comenzar una de estas tareas (por ejemplo, porque estuviésemos haciendo otra), el tiempo total de la tarea en el sistema será:*

$$T_i = d_i + t_i$$

Nuestro objetivo es obtener el menor:

$$T = Sum(T_i, 0, n)$$

javier@junquera.io (UAH 2023)

Minimización de tiempo en el sistema

Universidad de Alcalá

$Tarea_i$	t_i
1	5
2	8
3	2

- Voraz A → Lo más costoso antes
- Voraz B → Lo menos costoso antes
- Función X → Fuerza bruta (a.k.a. *backtracking*)

Función X

$Tarea_i$	t_i
1	5
2	8
3	2

$2,5,8 \longrightarrow 2 + (5+2) + (5+2+8) = \underline{\mathbf{24}}$

$2,8,5 \longrightarrow 2 + (2+8) + (2+8+5) = 27$

$8,5,2 \longrightarrow 8 + (8+5) + (8+5+2) = 31$

$8,2,5 \longrightarrow 8 + (8+2) + (8+2+5) = 33$

$5,8,2 \longrightarrow 5 + (5+8) + (5+8+2) = 33$

$5,2,8 \longrightarrow 5 + (5+2) + (5+2+8) = 27$

javier@junquera.io (UAH 2023)

Voraz A

$Tarea_i$	t_i
1	5
2	8
3	2

Priorizando $t_i > t_j$

8

+ (8+5)

+ (8+5+2)

= 36

Voraz B

$Tarea_i$	t_i
1	5
2	8
3	2

Priorizando $t_i < t_j$

2

+ (2+5)

+ (2+5+8)

= 24

Min. de tiempo en el sistema (complejidad)

```
def minimiza_tiempo(tareas): # Total: O(n log n) + O(n) → O(n log n)
    # Inicialización, O(n log n)
    tareas_aux = ordena(tareas)
    solucion = []

    # Voraz, O(n)
    for tarea in tareas_aux:
        if completable(solucion + [tarea]):
            solucion.append(tarea)

    return solucion
```

Planificación con plazo fijo

Universidad de Alcalá

Normalmente las tareas no pueden aplazarse indefinidamente, sino que tienen un plazo límite

En estos casos tendremos que buscar un compromiso entre la eficiencia pura (el tiempo en el sistema), y la realización de las tareas en plazo ($t_i < d_i$), o nos arriesgaremos a perder el beneficio que la tarea pueda aportar (g_i).

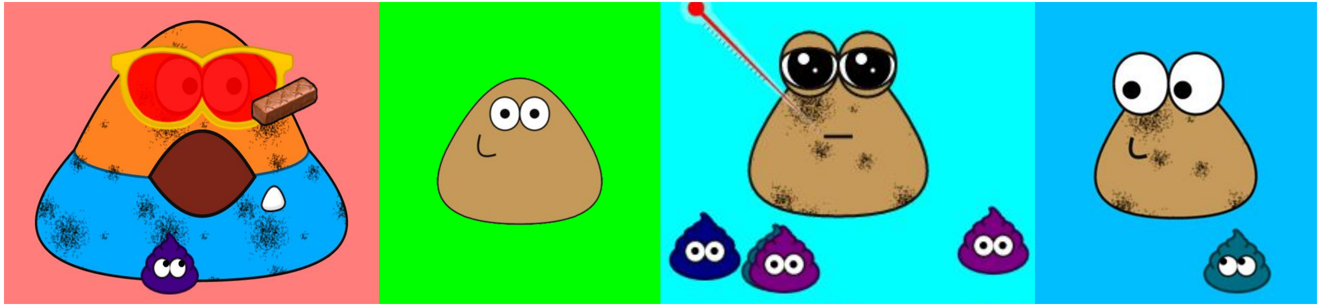
Trabajaremos con tareas tiempo-unidad (todas las tareas duran lo mismo).

Planificación con plazo fijo

Universidad de Alcalá

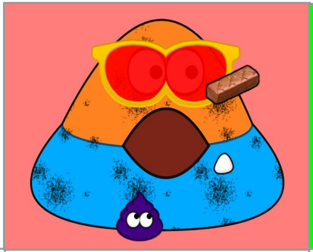
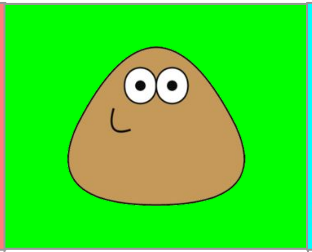
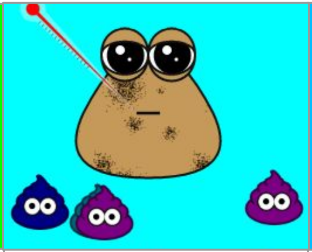
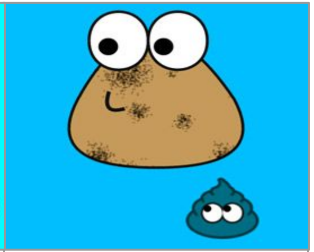
Tarea i produce unos beneficios (g_i) si se realiza antes de (d_i)

- g_i → ¿Cuánto ganamos al cuidarlo?
- d_i → ¿Cuánto puede durar sin cuidados?



Planificación con plazo fijo

Universidad de Alcalá

				
i	1	2	3	4
g_i	4	1	3	2
d_i	2	3	1	2

Planificación con plazo fijo

Inicialización A

- **Forma canónica:** Puntuales primero, y plazo creciente.

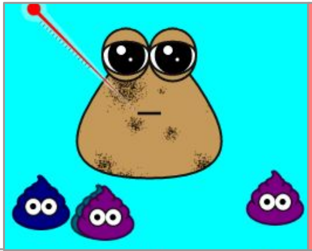
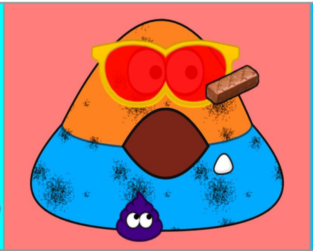
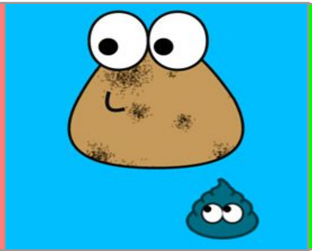
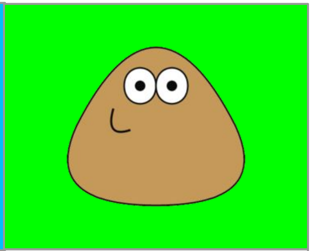
  Ordenar primero por *deadline* ($d_i < d_j \rightarrow i < j$)

- **Secuencia de tareas independientes:** Tienen el mismo plazo.

  Después dentro del mismo *deadline*, ordenar por ganacia ($g_i > g_j \rightarrow i < j$)

Planificación con plazo fijo

Universidad de Alcalá

				
i	**3**	**1**	4	**2**
g_i	3	4	2	1
d_i	1	2	2	3

$\text{Beneficio}_A = 8$

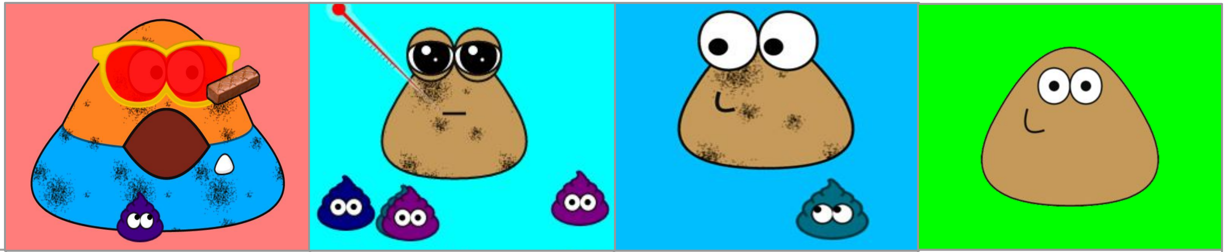
Planificación con plazo fijo

Inicialización B

- **Ordenar por beneficio decreciente**
- **Reordenar por *deadline* para evaluar si es completable**

Planificación con plazo fijo

Universidad de Alcalá



i	**1**	**3**	4	**2**
g_i	4	3	2	1
d_i	2	1	2	3

$\text{Beneficio}_B = 8$

Plazo fijo

```
def plazo_fijo(tareas):
    # Inicialización, O(n log n)
    tareas_aux = ordena(tareas) # Depende del criterio
    solucion = []

    # Voraz, O(n) + n O(?)
    for tarea in tareas_aux:
        if completable(solucion + [tarea]): # O(?)
            solucion.append(tarea)

    return solucion
```

Plazo fijo

```python
# [!] La solución debe estar ordenada por deadline
def completable(solucion): # O(n)
    # índice, valor
    for i, v in enumerate(solucion):
        '''
        Cada tarea dura una unidad de tiempo,
        el índice debe ser menor que el deadline
        '''
        if i >= v.deadline:
            return False

    return True
```

Plazo fijo

Universidad de Alcalá

```
def plazo_fijo(tareas):
    # Inicialización, O(n log n)
    tareas_aux = ordena(tareas) # Depende del criterio
    solucion = []

    # Voraz, O(n) + n O(n log n) → O(n² log n)
    for tarea in tareas_aux:
        if completable(solucion + [tarea]): # O(n)
            solucion.append_ordenado(tarea) # O(n log n)

    return solucion
```

Planificación con plazo fijo

Universidad de Alcalá

Otro ejemplo "de manual" [1]

a	1	2	3	4	5	6	7
g	70	60	50	40	30	20	10
d	4	2	4	3	1	4	6

Inicialización A = {5,2,4,1,7} = 200

Inicialización B = {1,2,3,4,7} = 230

[1] T.H. Cormen, C.E. Leiverson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed.

Simplificando

Universidad de Alcalá

La simplificación de los problemas viene acompañada de:

- Identificar tareas repetitivas que pueden eliminarse (e.g., ordenar al principio para no comprobar orden en cada iteración)
- Identificar estructuras de datos que permitan almacenar elementos auxiliares sin añadir complejidad al problema (e.g., listas v. *arrays*)

¿Preguntas?