Algoritmos

… o la ciencia (y el arte) de resolver problemas correcta y eficientemente

Yadran Eterovic

Abril 2021

Algoritmo: método finito, determinista y eficaz para resolver un problema

método:

- secuencia de instrucciones que si las llevamos a cabo en orden realizan una tarea particular

finito:

- si llevamos a cabo las instrucciones, entonces, *siempre*, el algoritmo termina después de ejecutar un número finito de instrucciones
- además, el tiempo para terminar debería ser relativamente corto

determinista:

- cada instrucción es clara, precisa, no ambigua

eficaz:

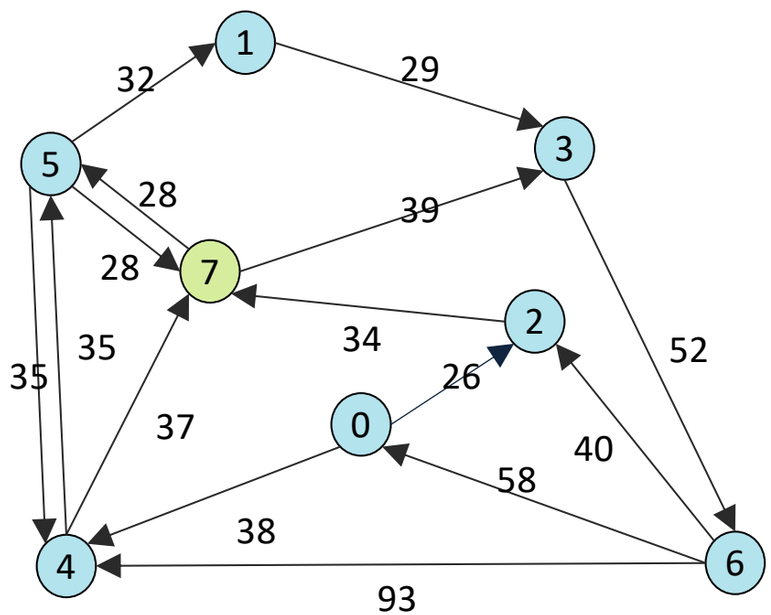
- cada instrucción es suficientemente básica —puede ser llevada a cabo, en principio, por una persona usando solo papel y lápiz

Para cada problema hay un algoritmo particular

Ordenar lexicográficamente los nombres de la/os estudiantes de un curso

...	
CARRASCO	SALINAS
CASTILLO	GUTIERREZ
CHRISTENSEN	CABRERA
COLLAO	ALCAYAGA
ECHEVERRÍA	GALAZ
ERRÁZURIZ	VERGARA
FILLOL	SOLAR
FRANKEN	BANDA
GONZÁLEZ	ARAVENA
GONZÁLEZ	DE MIGUEL
...	

Determinar las rutas más rápidas para ir al campus San Joaquín desde las casas de cada una/o de ustedes



(más precisamente) Para cada tipo de problemas hay un algoritmo particular

P.ej., los siguientes problemas son del mismo tipo y se pueden resolver con el mismo algoritmo:

- ordenar lexicográficamente los nombres de la/os estudiantes de un curso
- ordenar a los 1000 mejores tenistas profesionales según el puntaje que tienen en el ranking de la ATP

...	
CARRASCO	SALINAS
CASTILLO	GUTIERREZ
CHRISTENSEN	CABRERA
COLLAO	ALCAYAGA
ECHEVERRÍA	GALAZ
ERRÁZURIZ	VERGARA
FILLOL	SOLAR
FRANKEN	BANDA
GONZÁLEZ	ARAVENA
GONZÁLEZ	DE MIGUEL
...	

...

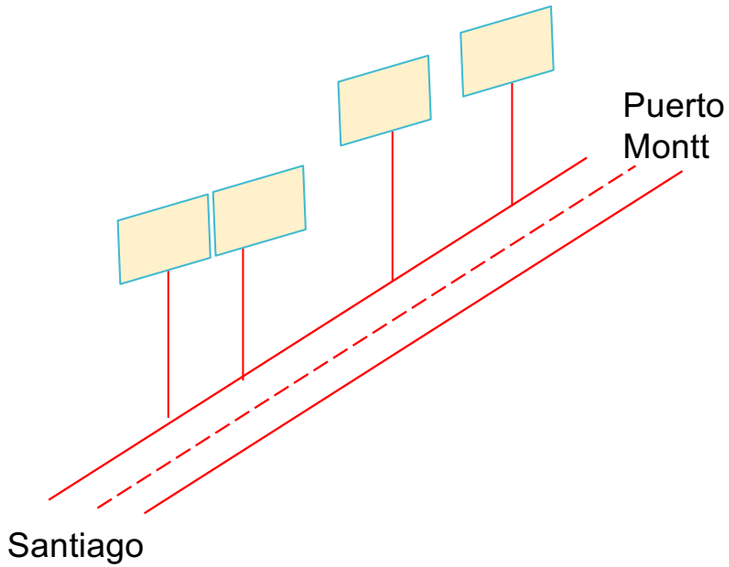
4	D. Thiem	**8615**
5	S. Tsitsipas	**7860**
6	A. Zverev	**6125**
7	A. Rublev	**5955**
8	R. Federer	**5875**
9	D. Schwartzman	**3720**
10	M. Berrettini	**3453**
11	R. Bautista A	**3090**
12	D. Goffin	**2885**

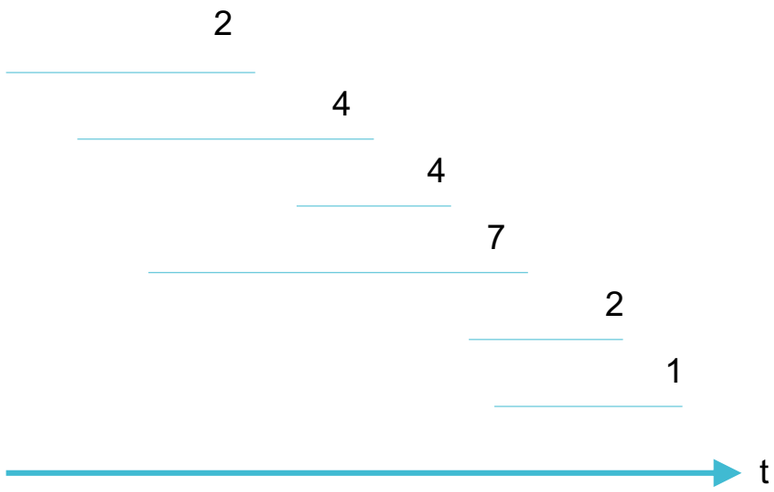
...

Hay problemas que parecen muy distintos, … pero que en realidad *son el mismo problema*

… y por lo tanto se pueden resolver con el mismo algoritmo:

- determinar en qué lugares a lo largo de la autopista Santiago a Puerto Montt me conviene poner letreros con propaganda de mi negocio
- determinar cuáles tareas me conviene hacer al final del semestre, cuando el tiempo no me alcanza para hacerlas todas y sólo puedo hacer una a la vez





Variaciones

El ranking de la ATP cambia todas las semanas, después de que se juegan los distintos campeonatos

La ATP quiere saber si cada lunes tiene que ordenar completamente de nuevo a los 1000 mejores jugadores —como si estuvieran totalmente desordenados

… o si puede hacer algo más eficiente, considerando que ya estaban ordenados y que los cambios de puntaje y de posición en el ranking de una semana a otra no son muy grandes

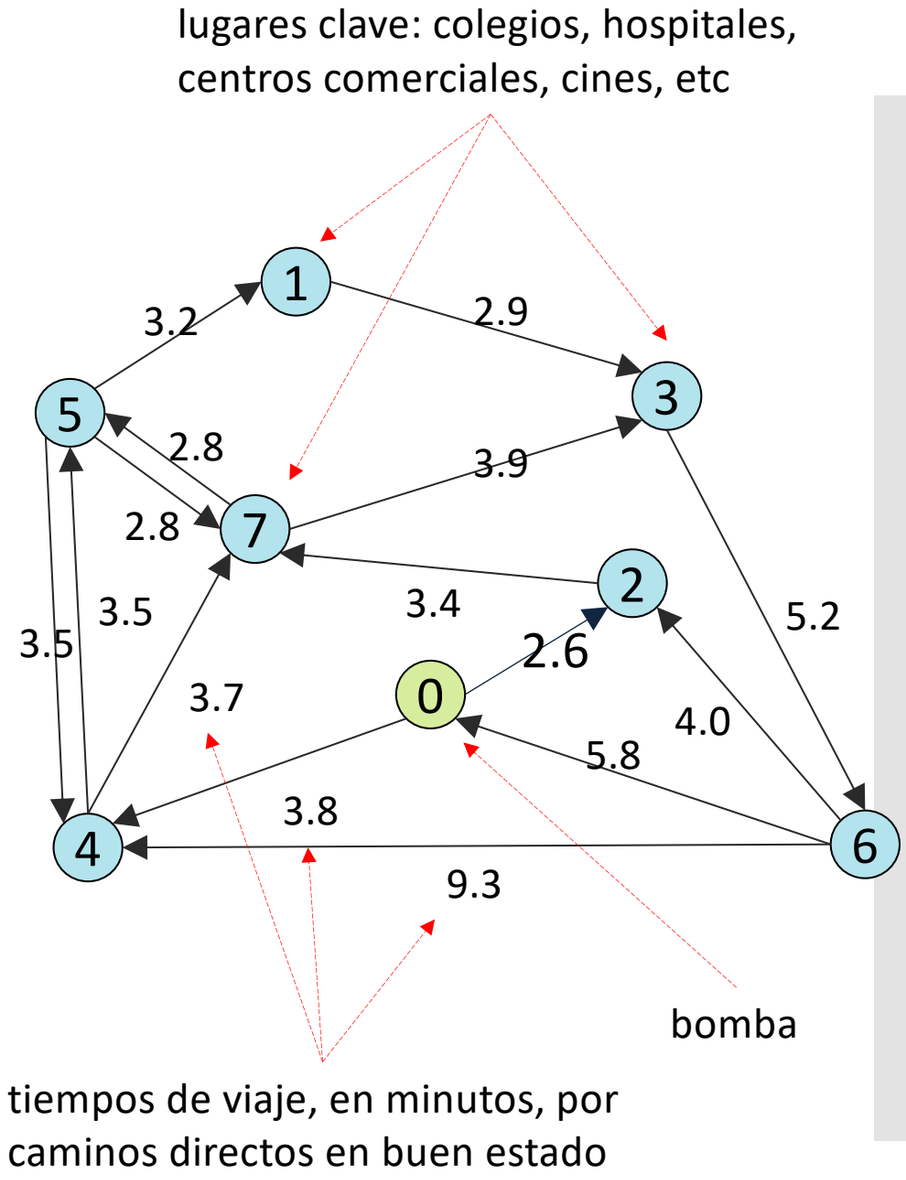
Variaciones

La bomba de incendios de una comuna quiere saber cuáles son los caminos más rápidos para llegar a distintos lugares clave

Para esto, midió los tiempos de viaje entre pares de puntos conectados directamente

Los bomberos quieren saber, p.ej., si para ir al punto 7 les conviene hacerlo pasando por 2 o por 4

… y si para ir al punto 3 les conviene llegar primero a 7 o a 1



Los algoritmos deben ser inventados, validados y analizados

Inventado:

- posiblemente usando alguna de las técnicas de diseño de algoritmos que han demostrado ser útiles —p.ej., *backtracking*, dividir para reinar, algoritmos codiciosos, programación dinámica

Validado:

- hay que *demostrar* que calcula la respuesta correcta para todos los *inputs* legales posibles —p.ej., por inducción, por contradicción

Analizado:

- determinar cuánto tiempo de computación —esto es, ejecución en la CPU del computador—

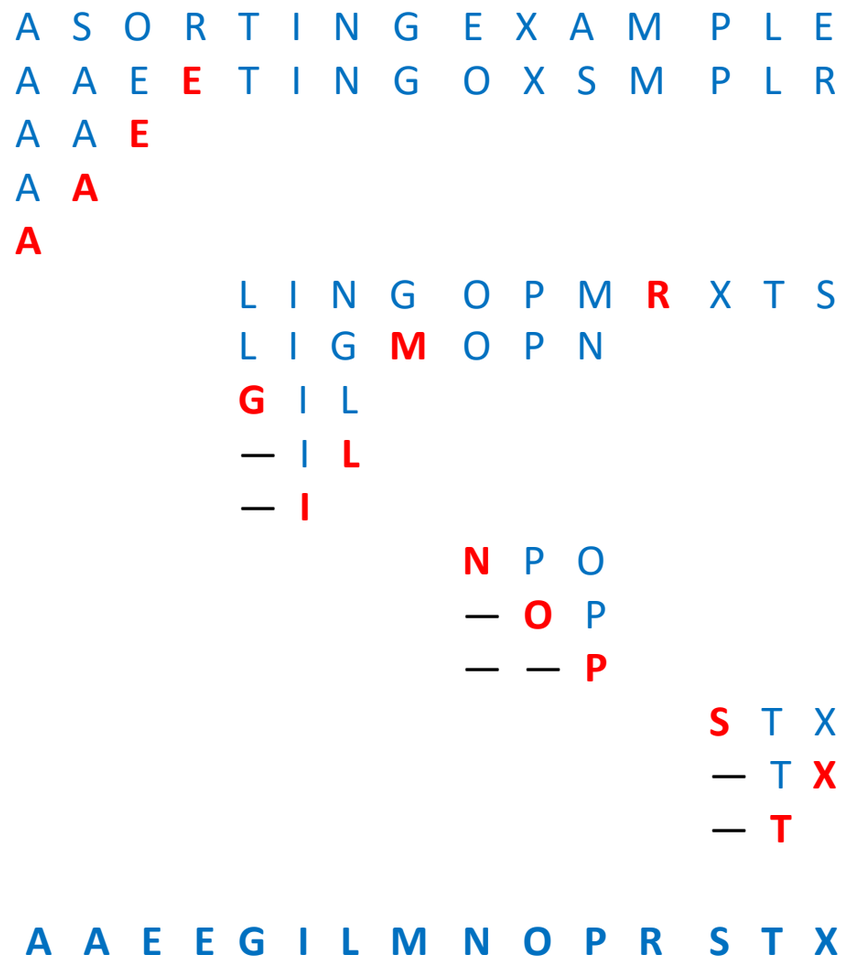
  … y cuánta memoria necesita —adicional a la memoria que se necesita para almacenar los datos del problema

El **algoritmo *quicksort*** : ordena (p.ej., de menor a mayor)

Es del tipo *dividir para reinar*

Se puede demostrar que resuelve el problema

… en un tiempo muy razonable (en la gran mayoría de los casos)



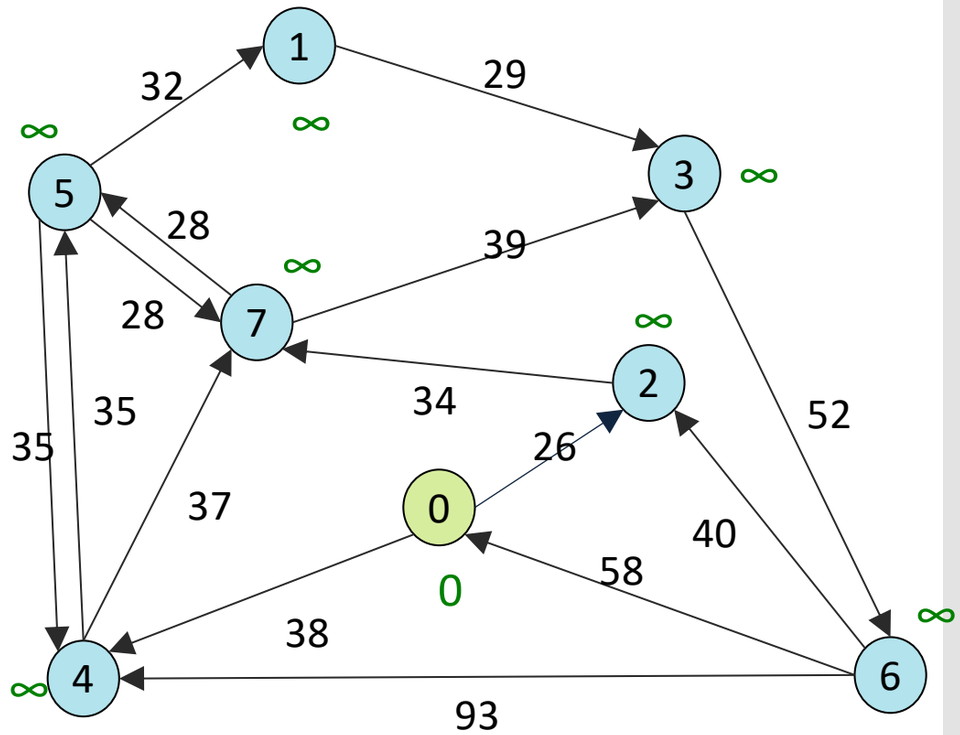
El **algoritmo de Dijkstra** : encuentra las rutas más cortas (o más rápidas) desde un punto, o *nodo*, a todos los otros nodos

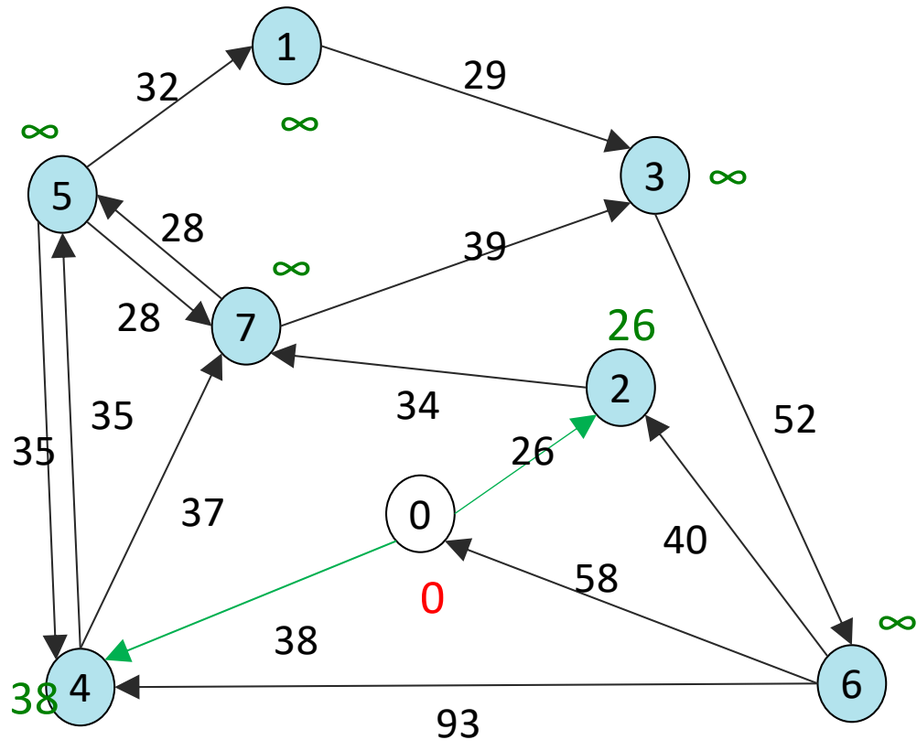
Es del tipo *codicioso*

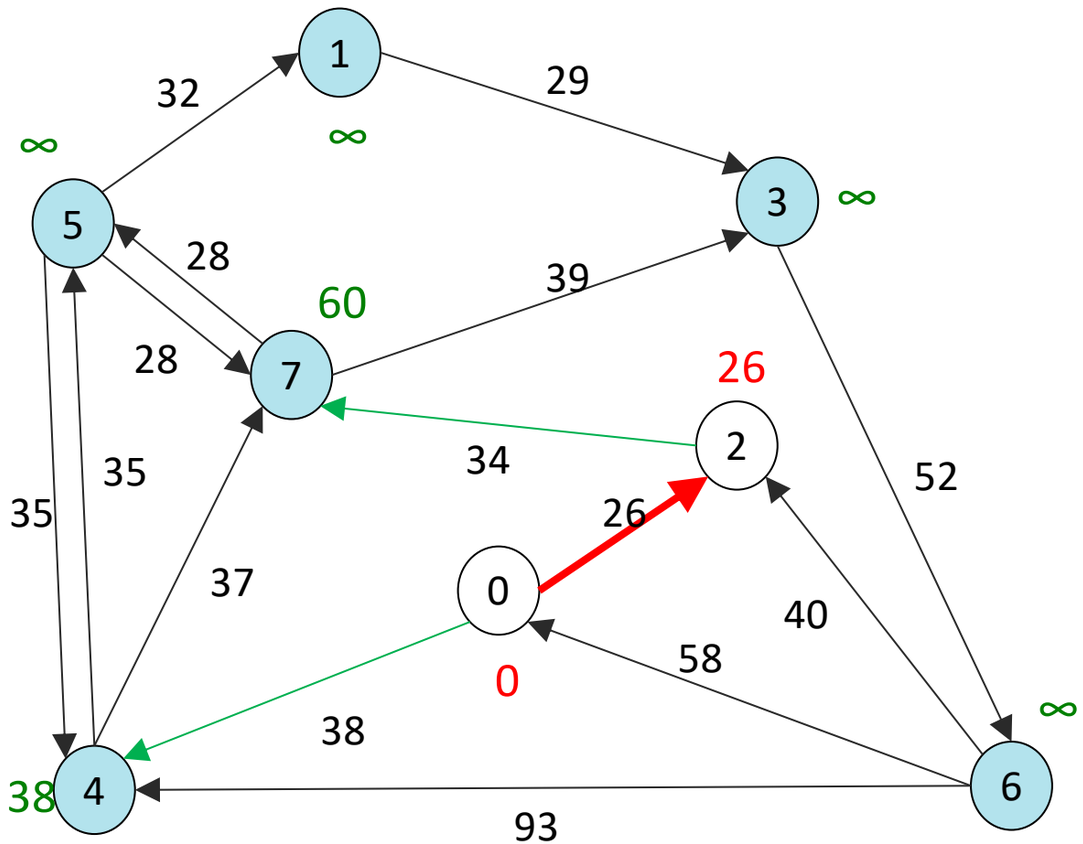
Se puede demostrar que efectivamente resuelve el problema

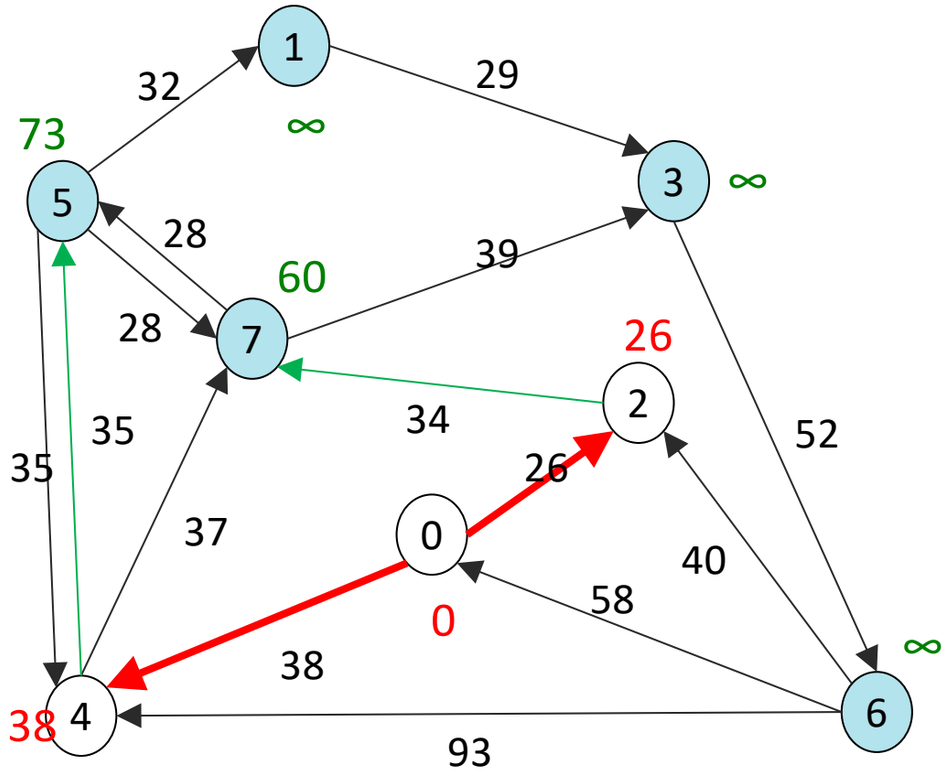
… en un tiempo muy razonable

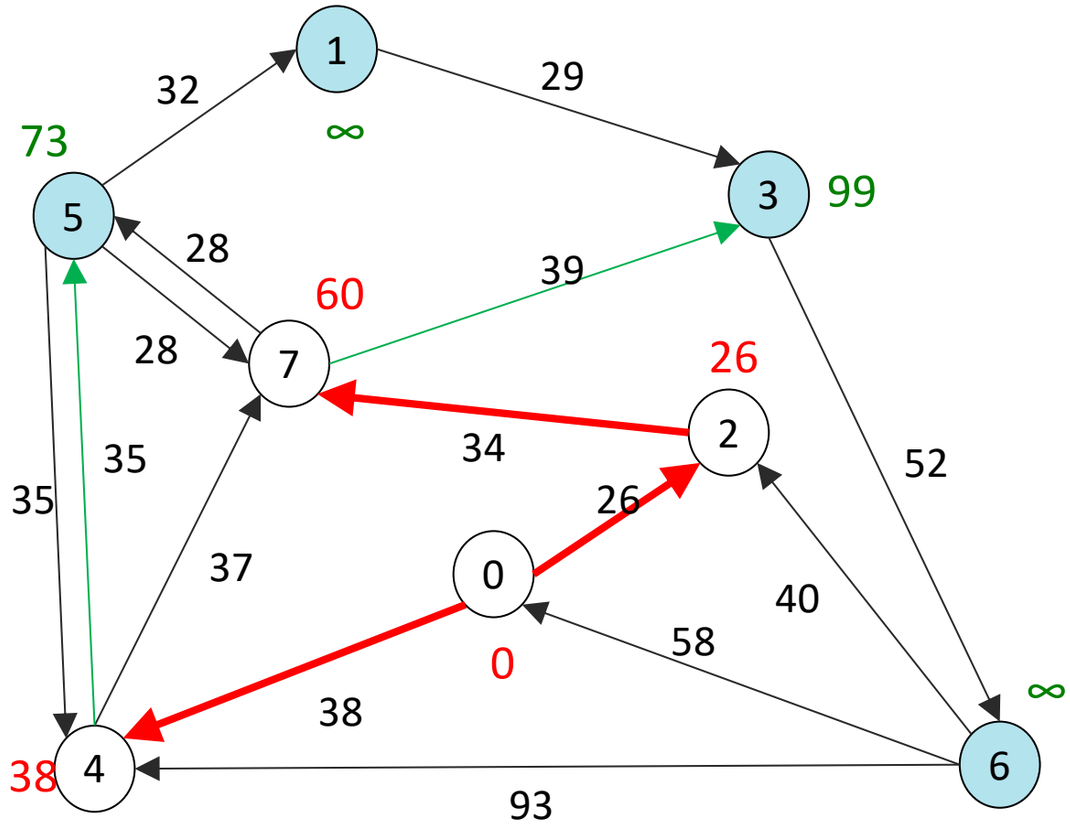
… *si todas las distancias* (o todos los tiempos de viaje) *son no negativas*



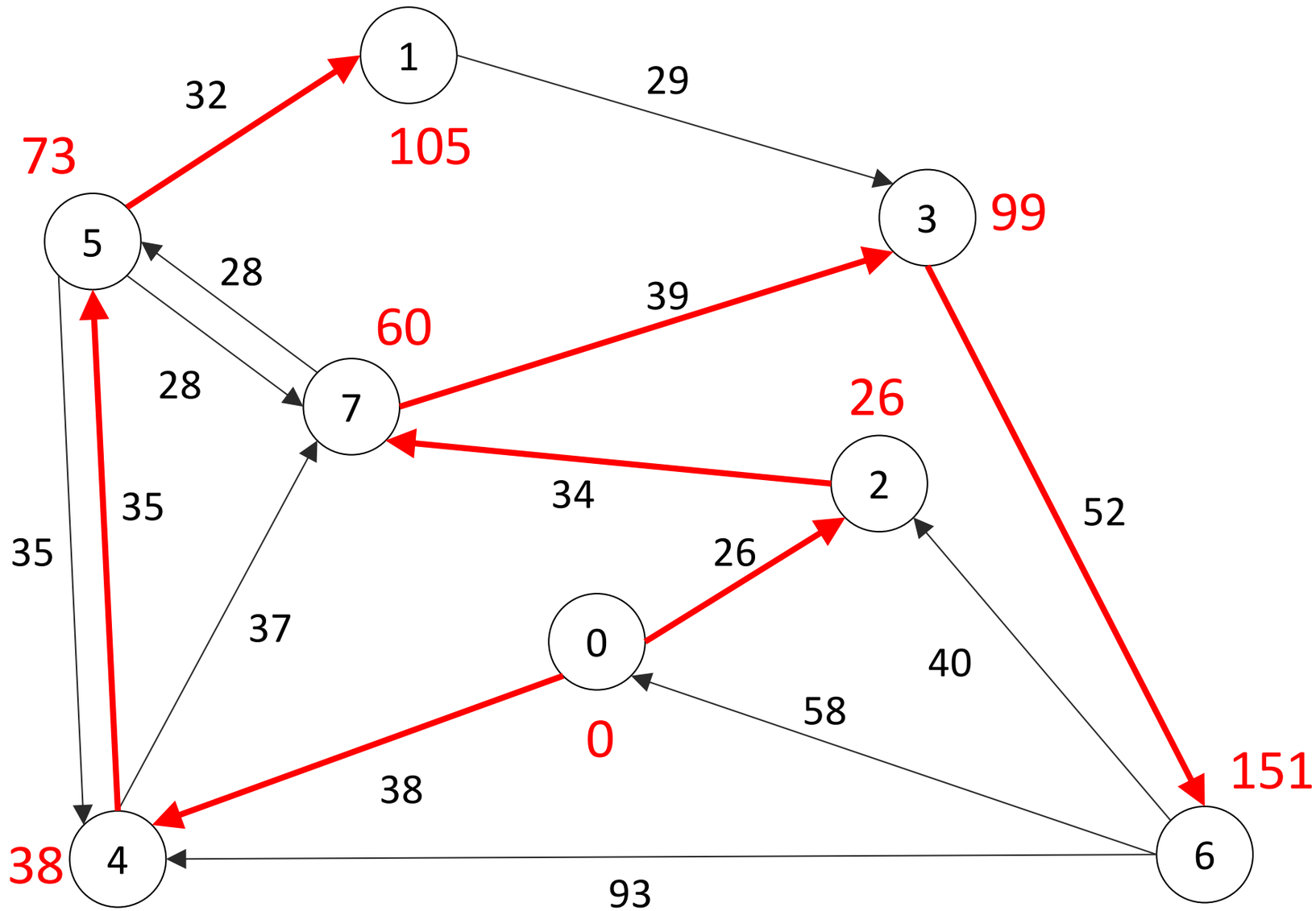

1
2
3
4
5
6
7
0
32
29
28
39
28
35
35
34
26
52
37
40
58
38
93
∞
∞
∞
∞
∞
26
0
38


1
2
3
4
5
6
7
0
32
29
28
39
28
35
35
34
26
52
37
40
58
38
93
∞
∞
∞
∞
∞
60
26
0
38


1
32
29
73
∞
5
3
∞
28
39
60
28
7
26
2
35
34
52
35
26
37
0
40
58
0
∞
38
38
4
6
93


1
32
29
73
∞
5
3
99
28
39
60
28
7
26
2
35
34
52
35
26
37
0
40
58
0
∞
38
38
4
6
93

...


1
105
32
29
73
5
3
99
28
39
60
28
7
26
2
34
35
35
52
26
37
0
40
58
0
38
151
38
4
6
93

Analizar un algoritmo: poder predecir racionalmente los recursos que el algorit-mo necesita

Principalmente, **tiempo de computación**:

- a veces, memoria, ancho de banda de comunicación, accesos al disco

… en un **computador** (relativamente) **convencional**:

- las instrucciones son ejecutadas una después de la otra

  … operaciones aritméticas comunes, movimientos simples de datos, e instrucciones de control

  … cada una toma una cantidad de tiempo constante

- los tipos de datos son los *números enteros* y *los números de punto flotante*

- no consideramos *caches* ni memoria virtual

Tiempo de computación: número de pasos u operaciones primitivas ejecutadas

Definimos “paso” de modo que sea lo más independiente posible del computador:

- un segmento sintáctica o semántica-mente significativo de un programa, **cuyo tiempo de ejecución es independiente de las características del problema particular**
- p.ej., una asignación de un número entero a una variable

  … una asignación de una variable a otra

  … la evaluación de una expresión aritmética

  … la evaluación de una condición, para saber si es verdadera o falsa
- se necesita una cantidad de tiempo constante para ejecutar estos pasos

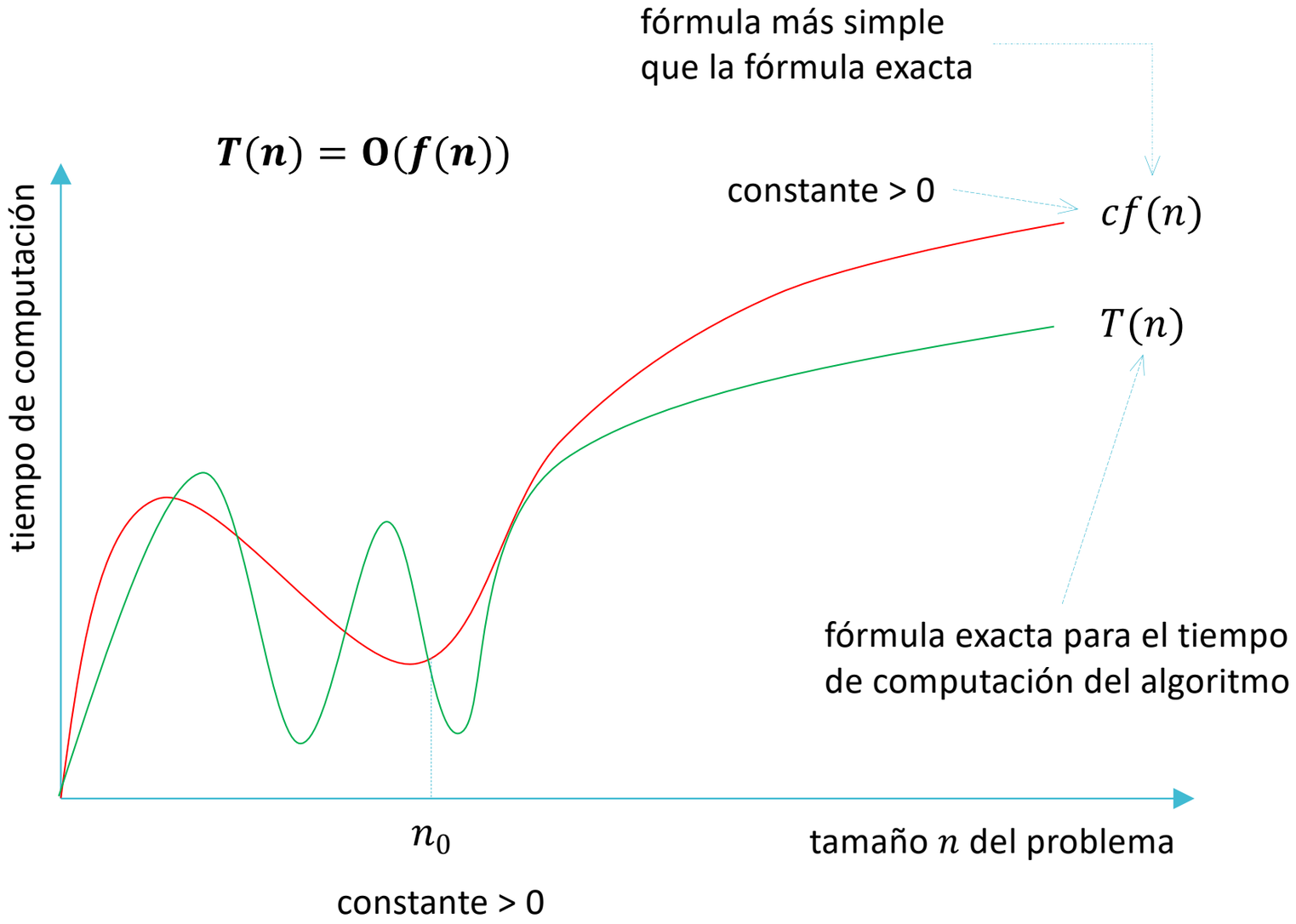
```
…
a = leer del teclado
b = leer del teclado
c = leer del teclado
d = sqrt(b*b-4*a*c)
x1 = (-b + d)/2
x2 = (-b – d)/2
…
```

un paso cada línea

n pasos

```
…
n = leer del teclado
s = 0
for k = 1, …, n:
    s = s + k
mostrar en pantalla(s)
…
```

Orden de crecimiento y la notación O



Órdenes de crecimiento comunes de algoritmos en función del tamaño *n* del problema

$\mathbf{1}$	instrucción o paso	sumar dos números
$\log \boldsymbol{n}$	dividir por la mitad	búsqueda binaria
$\boldsymbol{n}$	*loop*	encontrar el máximo
$\boldsymbol{n} \log \boldsymbol{n}$	dividir para reinar	*mergeSort*
$\boldsymbol{n^2}$	*loop* double	revisar todos los pares
$\boldsymbol{n^3}$	*loop* triple	revisar todos los tríos
$\boldsymbol{2^n}$	búsqueda exhaustiva	revisar todos los subconjuntos
$\boldsymbol{n}!$	búsqueda exhaustiva	revisar todas las permutaciones

Tiempos de ejecución (en segundos) de varios algoritmos —con distintos órdenes de crecimiento— para un mismo problema

n	$\mathbf{O}(n^3)$	$\mathbf{O}(n^2)$	$\mathbf{O}(n \log n)$	$\mathbf{O}(n)$
100	0.00016	0.000006	0.000005	0.000002
1,000	0.096	0.00037	0.00006	0.00002
10,000	86.67	0.0333	0.00062	0.00022
100,000	*NA*	3.33	0.0067	0.0022
1,000,000	*NA*	*NA*	0.075	0.023

La ventaja de pasar de un algoritmo O(n^2) a uno O($n$log$n$) para el mismo problema

… es que ahora puedes resolver en el mismo tiempo el mismo problema pero para inputs de tamaño mucho más grande

… aproximadamente n/log$n$ veces más grande

Todo lo anterior está muy bien … sin embargo …

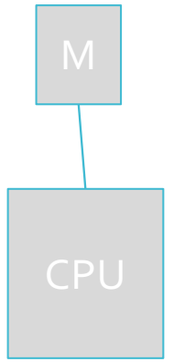
Queremos poder resolver problemas muy grandes más rápidamente de lo que la CPU más rápida permite

Nuestra única opción es tratar de dividir el problema en varios subproblemas

… resolver cada uno de los subproblemas en una CPU diferente en paralelo —posiblemente comunicándose una CPU con otra de vez en cuando

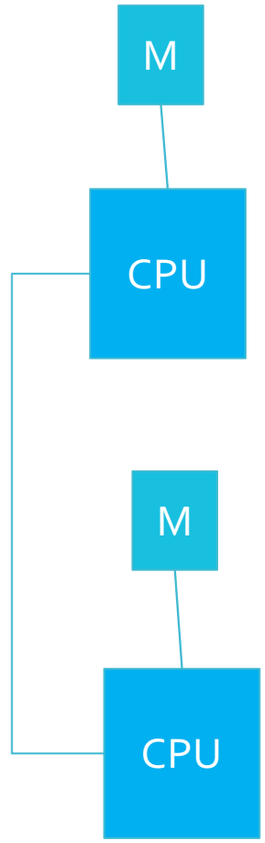
… y finalmente encontrar la solución al problema original combinando las soluciones a los subproblemas

Algoritmos concurrentes, distribuidos, paralelos



Siglo XX (1960s a 1990s):
un procesador, o CPU, cada vez con más memoria

Siglo XXI:
múltiples CPUs,
con memoria compartida
o con memoria distribuida



Aparecen dos aspectos nuevos: **comunicación sincronización**

Cuando la ejecución de un algoritmo es dividida entre múltiples CPUs —llamamos *proceso* al código ejecutado en cada CPU—

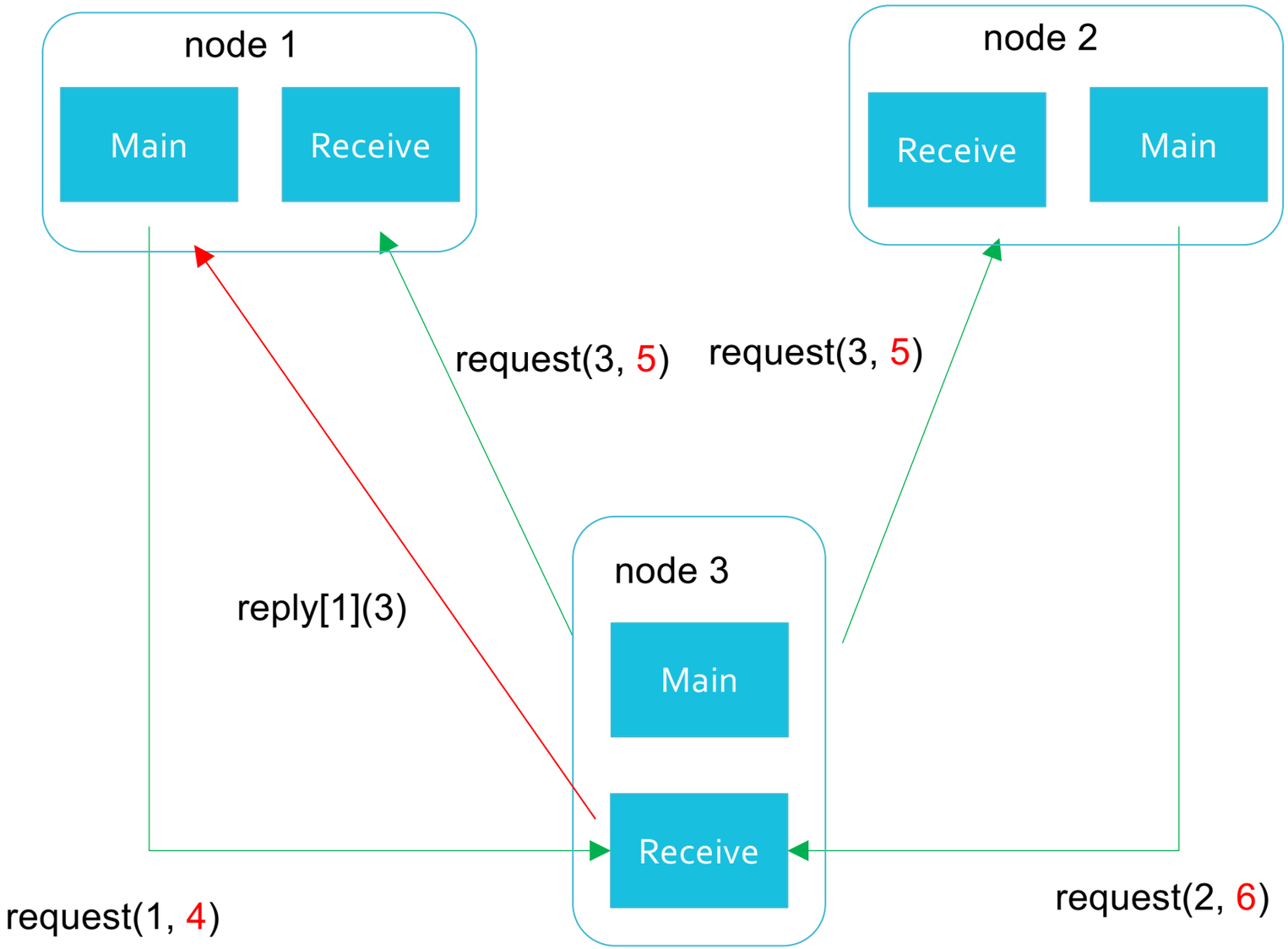
… los procesos necesitan:

- compartir información
- ponerse de acuerdo

P.ej., en un sistema con memoria distribuida —independientemente del problema particular que estén tratando de resolver los procesos— cada proceso de vez en cuando necesita escribir información en una base de datos común

… lo cual debe hacerse bajo *exclusión mutua* —solo un proceso al mismo tiempo puede estar escribiendo en la base de datos

Versión distribuida del *algoritmo de la panadería*, de Ricart y Agrawala



Cuando el nodo **myID** quiere escribir en la base de datos:

- primero, define su propio turno, **myNumber**, como uno más que el más grande de todos los turnos que ha “visto”
- luego, envía una solicitud de permiso a cada uno de los otros nodos y se queda esperando a que todos respondan dando sus permisos
- solo cuando ha recibido todas las respuestas, escribe en la base de datos

Por otra parte, cuando el nodo **myID** recibe una solicitud de permiso de otro nodo, **nodeID**, compara el turno de **nodeID** con el propio:

- si el turno de **nodeID** es menor que el propio —**requestNumber << myNumber**— entonces le responde inmediatamente dándole el permiso
- en caso contrario, pospone la respuesta y agrega el número del otro nodo a una cola local, **deferred**

Finalmente, cuando el nodo **myIDF** termina de escribir, le responde a todos los nodos que están en su cola **deferred**

Bibliografía

M. Ben Ari, *Principles of Concurrent and Distributed Programming* (2nd ed.), Addison Wesley, 2006

T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms* (3rd ed.), MIT Press, 2009

E. Horowitz, S. Sahni, S. Rajasekeran, *Computer Algorithms* (2nd ed.), Silicon Press 2008

R. Sedgewick, K. Wayne, *Algorithms* (4th ed.), Addison-Wesley 2011

M. Weiss, *Data Structures and Algorithm Analysis in C++* (4th ed.), Pearson 2013