Práctica 1: Eficiencia

Gustavo Rivas Gervilla

Contenido

Motivación

La eficiencia de un algoritmo (ya sea en tiempo o en espacio) es uno de los aspectos cruciales en la evaluación de código.

Como ingenieros nuestro trabajo es buscar siempre la mejor solución.

Todos los iconos empleados en este documento han sido creados por Freepick en flaticon.com.

Motivación

Hay problemas de elevada complejidad:

- El problema de encontrar el óptimo (no local) de un problema combinatorio suele ser un problema inabordable $\Longrightarrow$ algoritmos genéticos, enfriamiento simulado, algoritmos de colonia de hormigas...
- El QAP o problema de la asignación cuadrática es uno de estos problemas.
- Determinar el grado de verdad de una fórmula de la aritmética de Presburger es $\mathcal{O}(2^{2^{cn}})$.

Motivación

En las *coding interviews* es importante la eficiencia de las soluciones que propongamos.

- Es parte del libro Cracking The Coding Interview.
- Big O Notation.
- "Performance Matters" by Emery Berger.

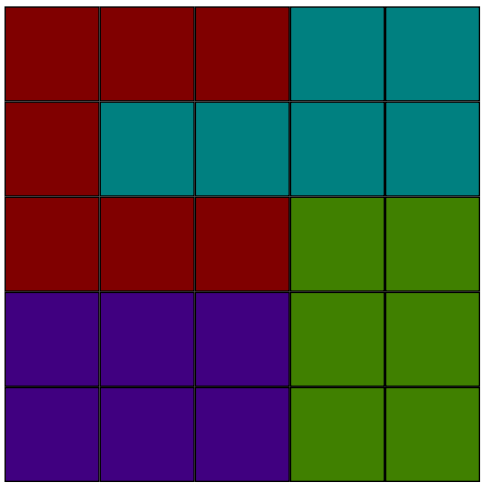
También en el campo de la investigación se analiza la eficiencia de los algoritmos.

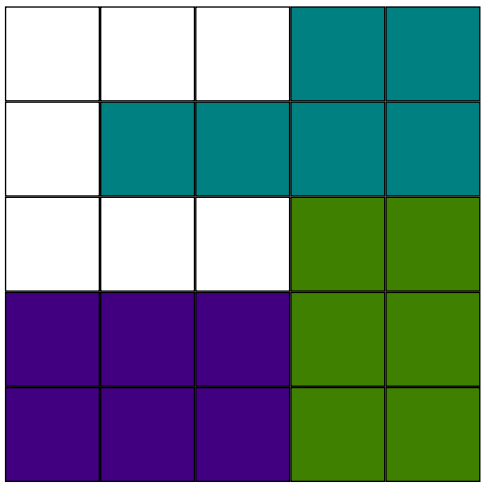
- SO Kuznetsov y SA Obiedkov (2001). "Algorithms for constructing a set of all concepts of formal context and their Hasse diagrams". En: *Journal of Computer and Systems Sciences International* 1, págs. 120-129
- Petr Krajca, Jan Outrata y Vilém Vychodil (2010). "Advances in Algorithms Based on CbO.". En: *CLA*. Vol. 672, págs. 325-337

Contenido

Flood Fill Algorithm

Este es algoritmo que se usa por ejemplo en la herramienta *Rellenar* de las herramientas de dibujo o en el juego Go para ver piezas se eliminan[1]





[1] Información extraída de artículo.

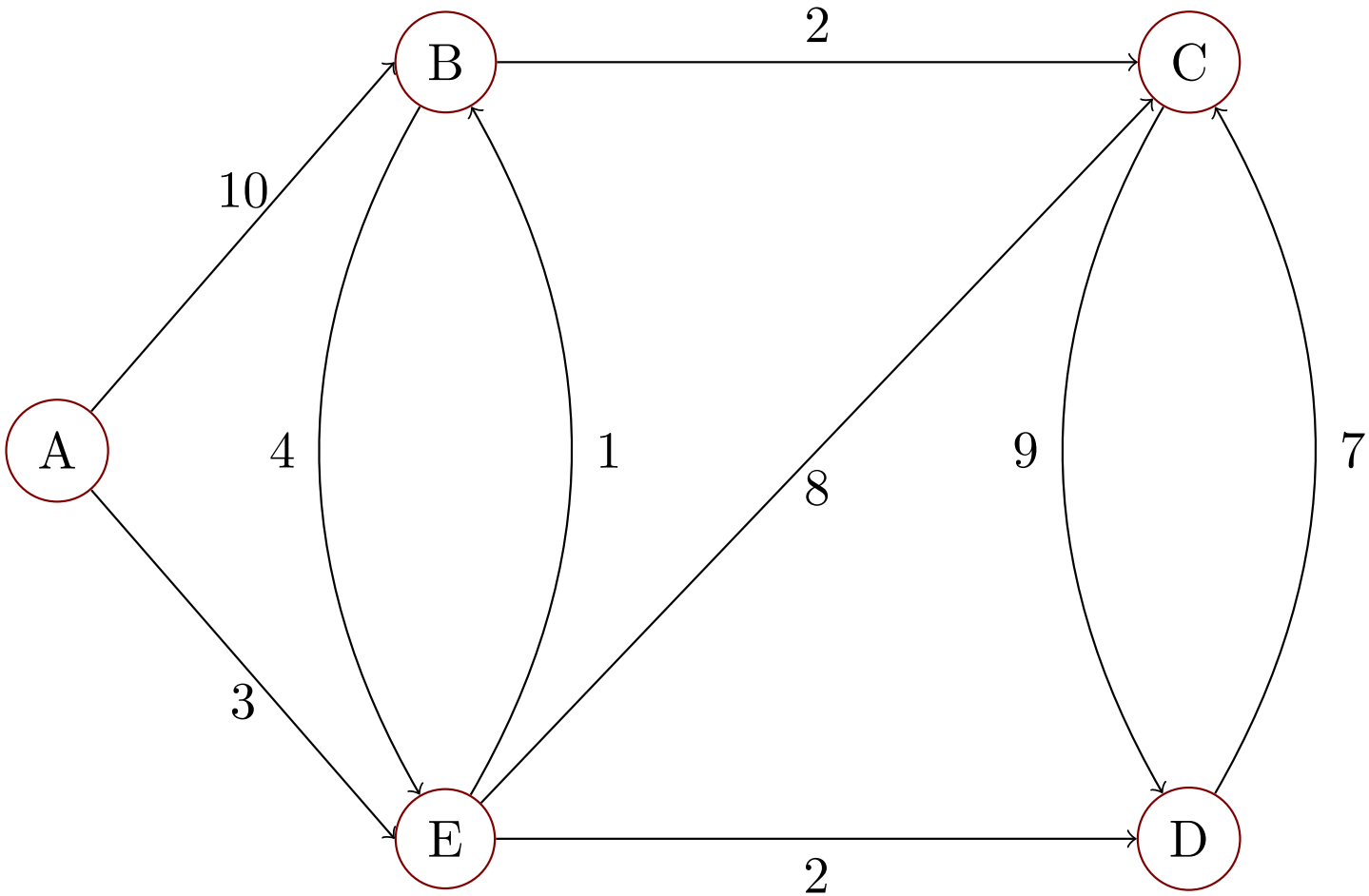
Flood Fill Algorithm

```
1   // Los movimientos posibles:
2   int row[] = { -1,-1,-1,0,0,1,1,1 };
3   int col[] = { -1,0,1,-1,1,-1,0,1 };
4
5   // Comprueba si nos podemos mover al pixel (x,y).
6   bool isSafe(char mat[M][N], int x, int y, char target){
7     return (x >= 0 && x < M && y >= 0 && y < N) && mat[x][y] == target;
8   }
9
10  void floodFill(char mat[M][N], int x, int y, char replacement){
11    queue<pair<int, int>> q;
12    q.push({x,y});
13
14    char target = mat[x][y];
15
16    while (!q.empty()){
17          pair<int,int> node = q.front();
18          q.pop();
19
20          int x = node.first, y = node.second;
21
22          mat[x][y] = replacement;
23
24          for (int k = 0; k < 8; k++)
25            if (isSafe(mat, x + row[k], y + col[k], target))
26                  q.push({x + row[k], y + col[k]});
27    }
28  }
```

La eficiencia teórica de este algoritmo es $\mathcal{O}(MN)$.

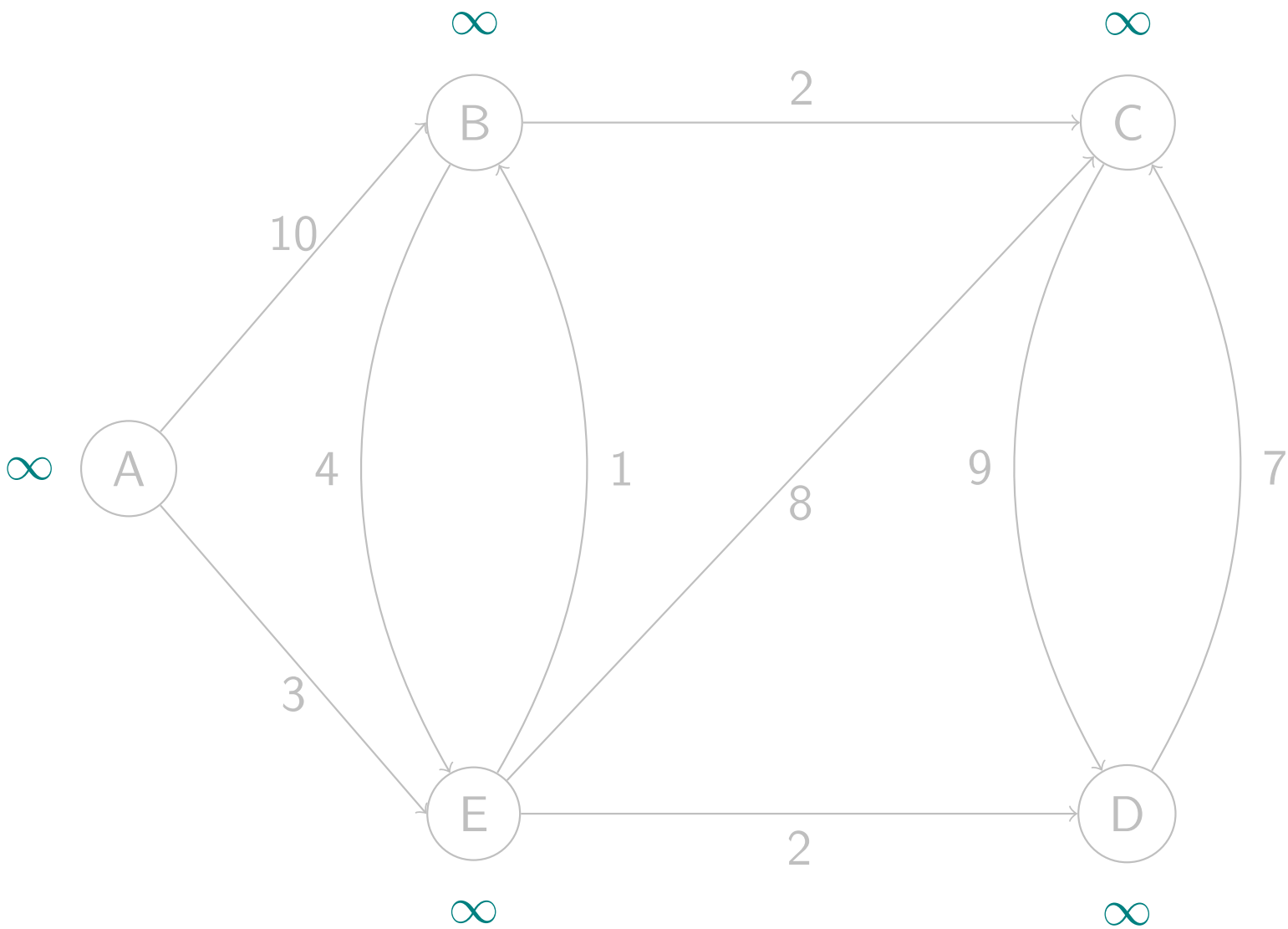
Algoritmo de Dijkstra

Con este algoritmo[2] podemos encontrar los caminos más cortos desde uno de los nodos hasta cada uno del resto de nodos.

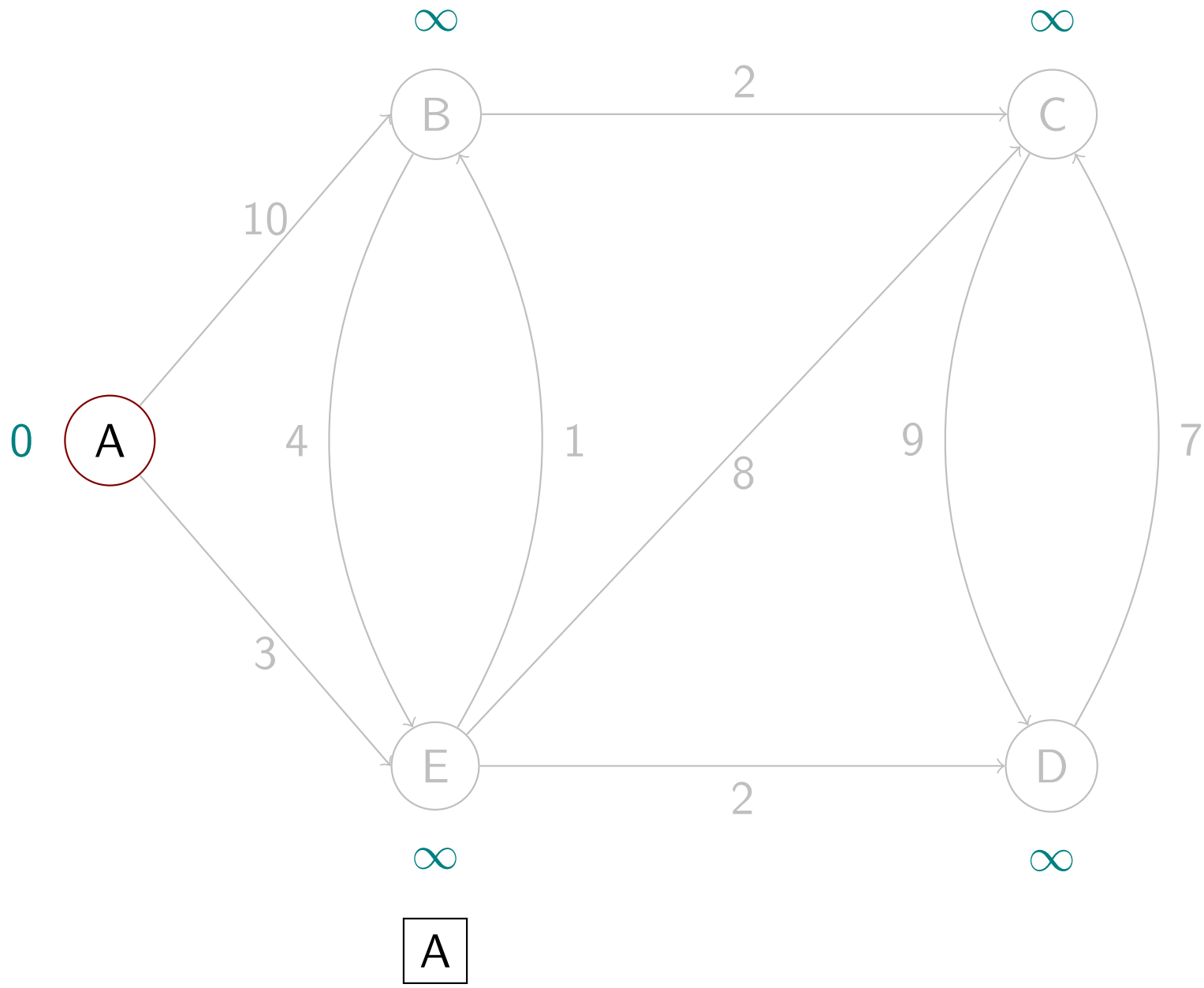


[2]Información extraída de artículo.

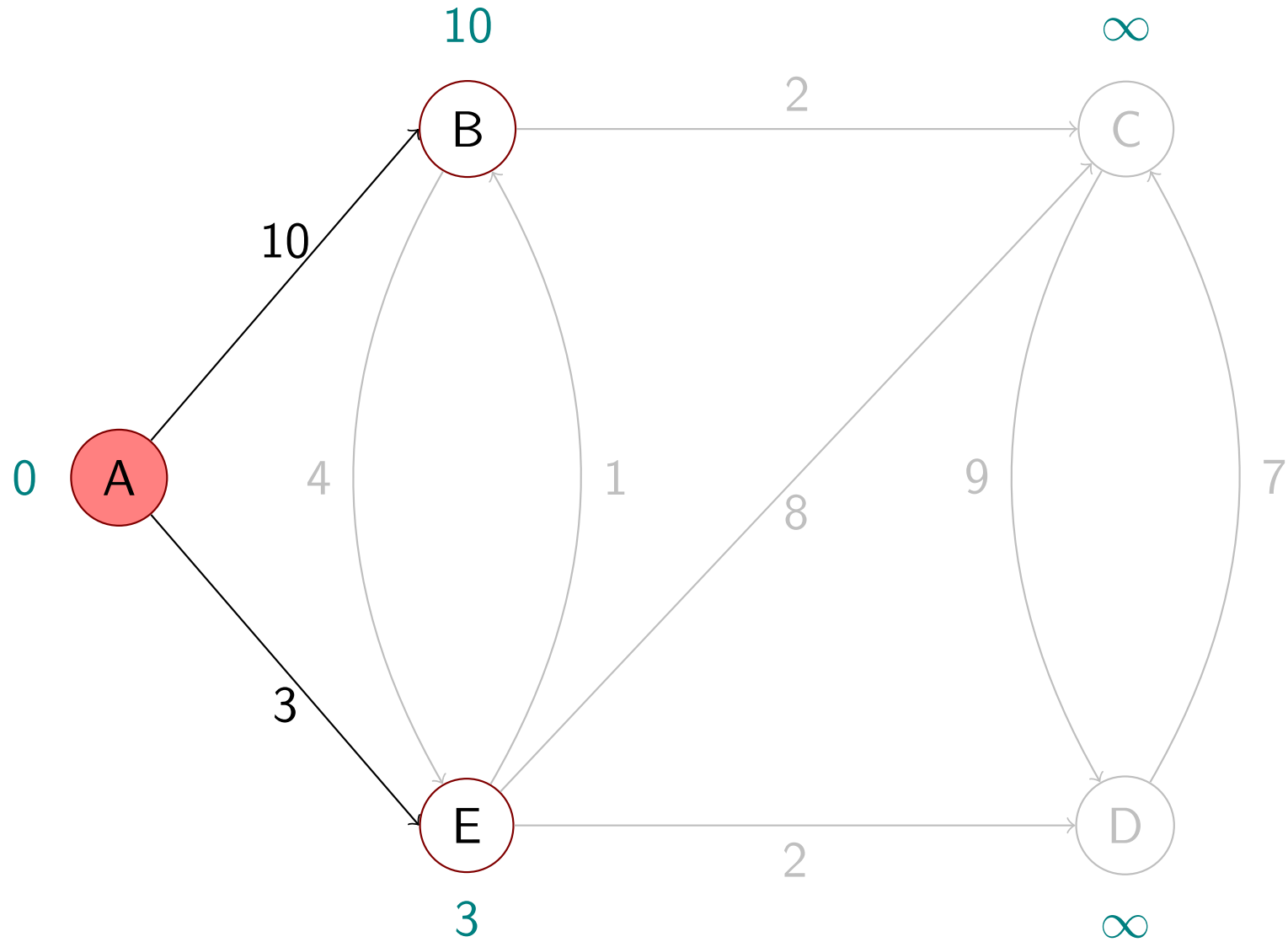
Algoritmo de Dijkstra



Algoritmo de Dijkstra

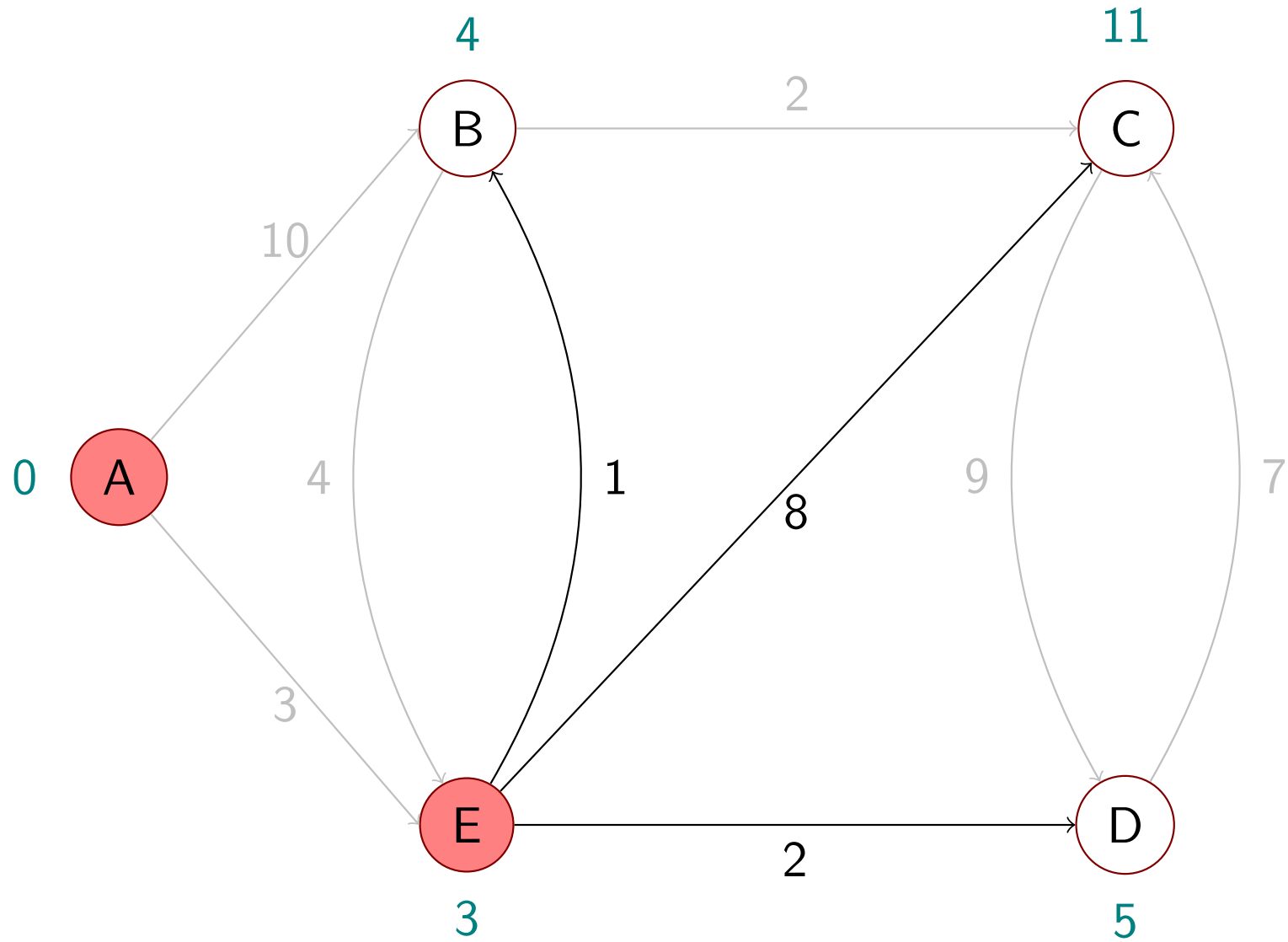


Algoritmo de Dijkstra



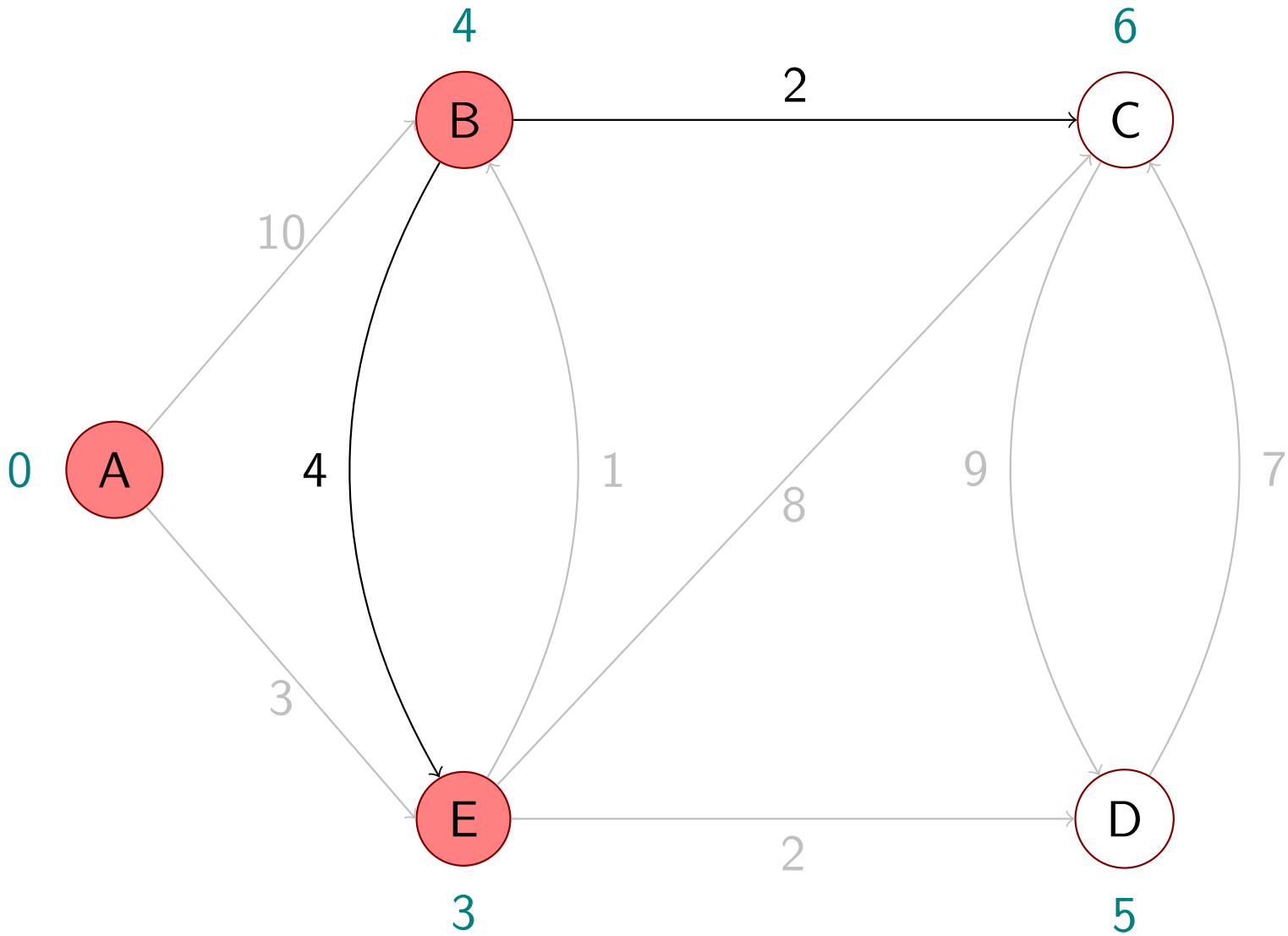
E	B

Algoritmo de Dijkstra



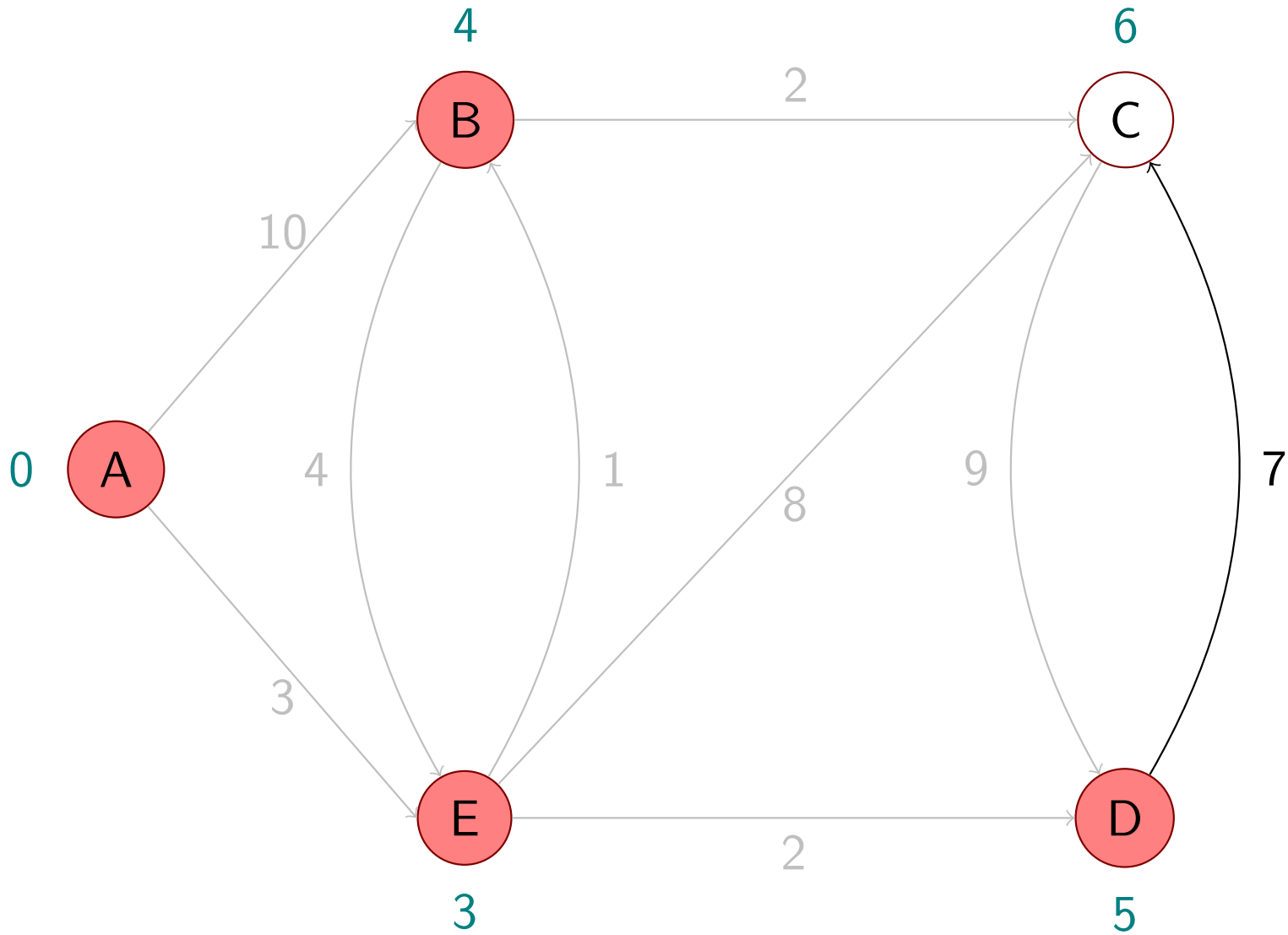
B	D	C

Algoritmo de Dijkstra

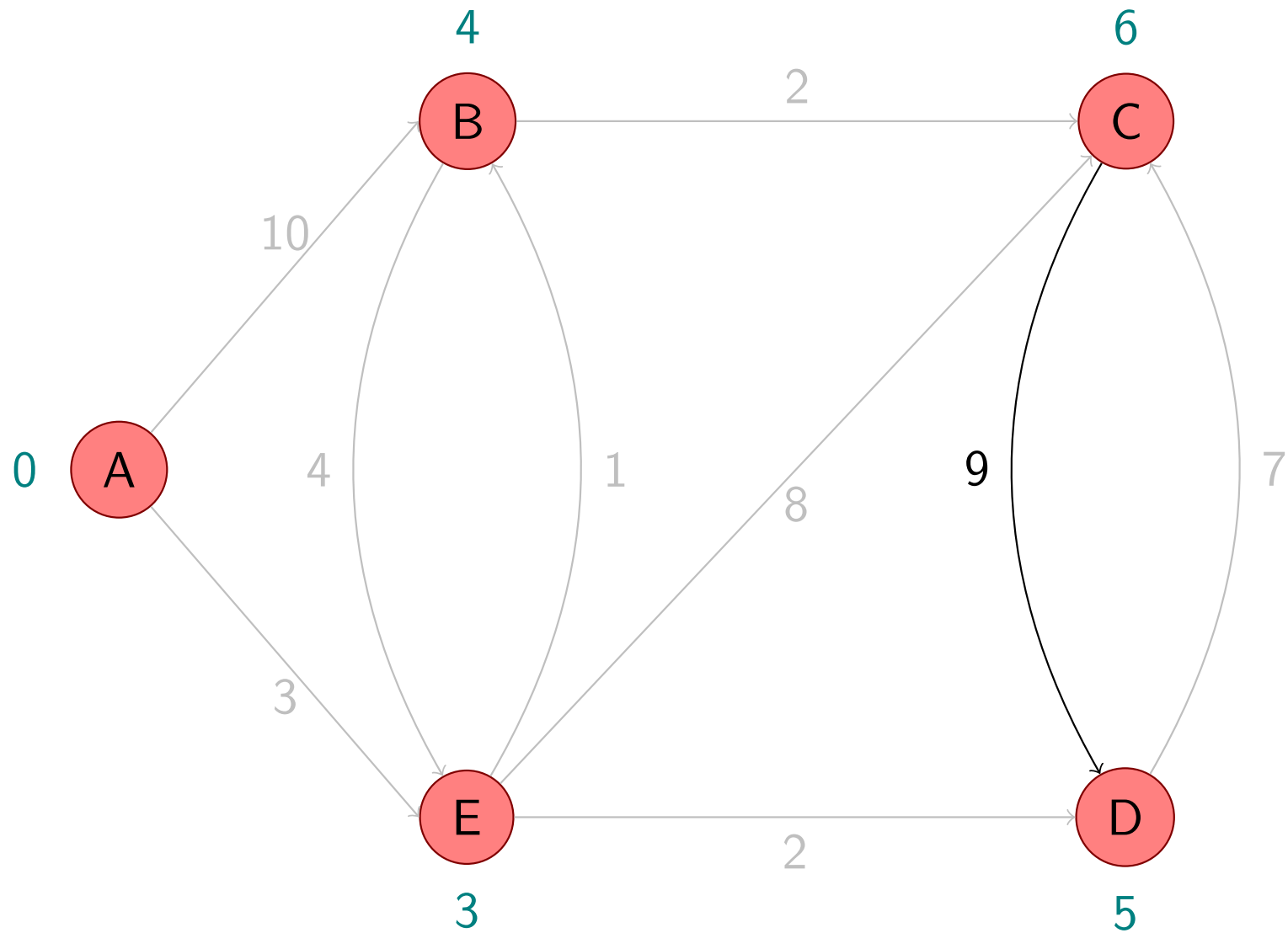


D	C

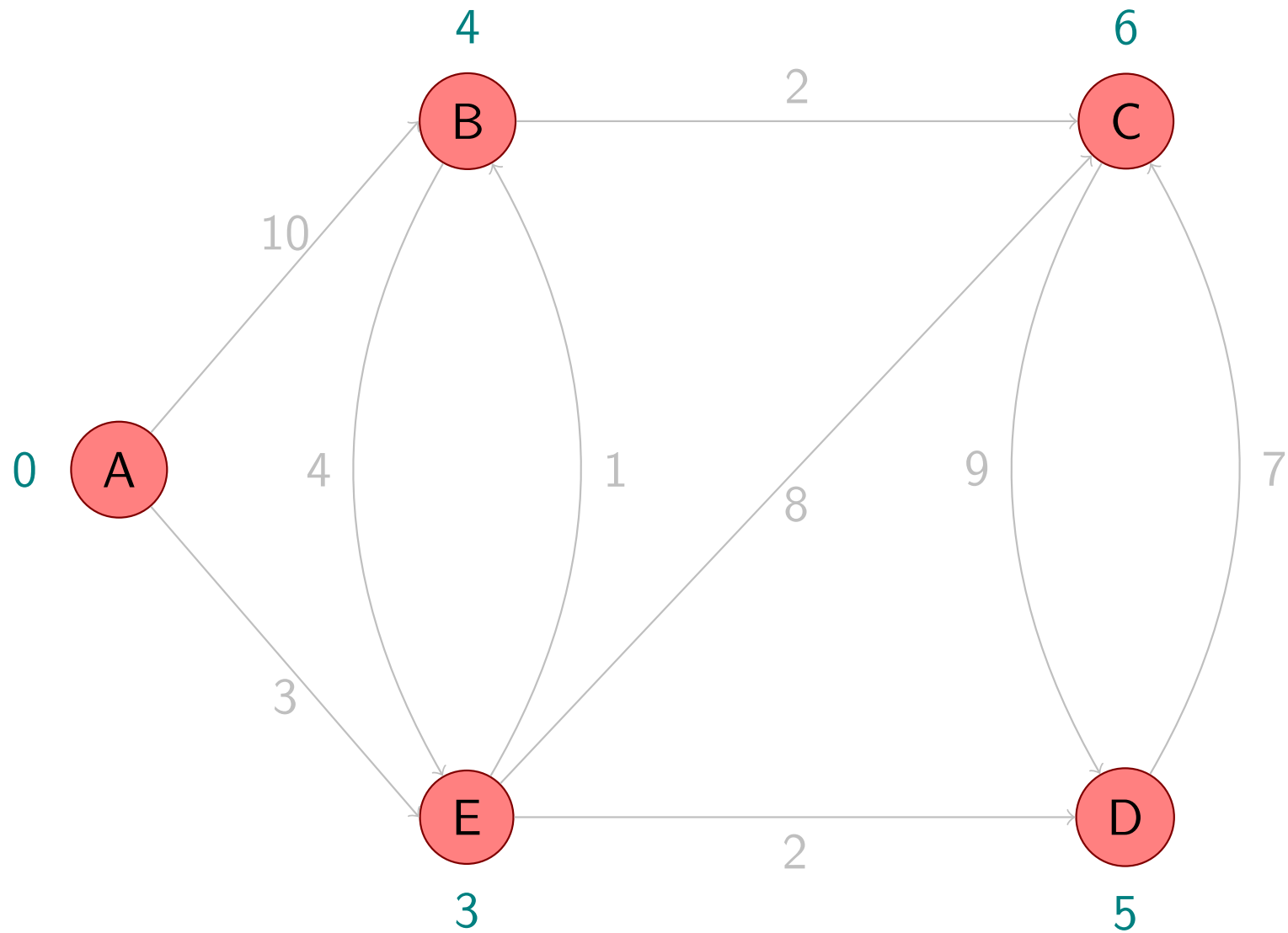
Algoritmo de Dijkstra



Algoritmo de Dijkstra



Algoritmo de Dijkstra



Algoritmo de Dijkstra

```
1   void dijkstra(Graph const &graph, int source, int N){
2     priority_queue<Node, vector<Node>, comp> min_heap;
3     min_heap.push({source, 0});
4
5     // Inicializa la distancia de source a cada nodo.
6     vector<int> dist(N, INT_MAX); dist[source] = 0;
7
8     // Indica para que nodos tenemos ya el camino.
9     vector<bool> done(N, false); done[source] = true;
10
11    vector<int> prev(N, -1);
12
13    while (!min_heap.empty()){
14          Node node = min_heap.top();
15          min_heap.pop();
16
17          int u = node.vertext;
18
19          for (auto i : graph.adList[u]){
20            int v = i.dest; int weight = i.weight;
21
22            if (!done[v] && (dist[u] + weight) < dist[v]){
23                  dist[v] = dist[u] + weight;
24                  prev[v] = u;
25                  min_heap.push({v, dist[v]});
26            }
27          }
28
29          done[u] = true;
30    }
31  }
```

La eficiencia teórica de este algoritmo es $\Theta((E + V)(logV))$.

Algoritmos Recursivos

$$T(n) = \begin{cases} 2T(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases}$$

$T(n) = 2T(n-1)+1 = 2^2T(n-2)+2+1 = 2^3T(n-3)+2^2+2+1 = \cdots = \sum_{i=0}^{n-1} 2^i = \frac{1-2^n}{1-2} = 2^n - 1 \implies \mathcal{O}(2^n)$.

Podemos probar la expansión anterior por inducción.
Hay otras técnicas de resolución de recurrencias[3].

[3] Kimmo Eriksson (1999). "A Summary of Recursion Solving Techniques". En: URL: https://www.math.kth.se/math/GRU/2012.2013/SF1610/CINTE/mastertheorem.pdf

Contenido

Eficiencia Empírica

El comportamiento real de un algoritmo depende de múltiples factores:

- La entrada.
- La máquina con la que estemos trabajando.
- La implementación del algoritmo (incluyendo el lenguaje de programación empleado).
- Opciones de compilación en el caso de lenguajes compilados.
- ...

Este análisis es tan importante como el análisis de tipo teórico.

Comparación entre Estructuras de Datos

Cada estructura de datos tiene sus ventajas e invonvenientes, será nuestro trabajo escoger la forma más adecuada de representar la información para el problema que queremos resolver.

	Acceso	Búsqueda	Insercción	Borrado
Array	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Pila	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Cola	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Lista Enlazada (Simple)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Árbol Binario de Búsqueda	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Árbol Rojo-Negro	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$

Tabla obtenida de artículo.

Tiempos de Ejecución

Visualizar los datos es algo que nos puede ayudar a extraer mucha información.

```
1   int buscar(const int *v, int n, int x){
2     int i = 0;
3     while (i < n && v[i] != x)
4       i = i+1;
5
6     if (i < n)
7       return i;
8     else
9       return -1;
10  }
```

```
1   #!/bin/sh
2   tams=( 100 500 1000 6000 8000 10000
        12000 )
3
4   i=$inicio
5   echo > tiempos7.dat
6   for i in "${tams[@]}"
7   do
8       echo Ejecucion tam = $i
9       echo `./busqueda $i 10000` >>
            tiempos7.dat
10  done
```

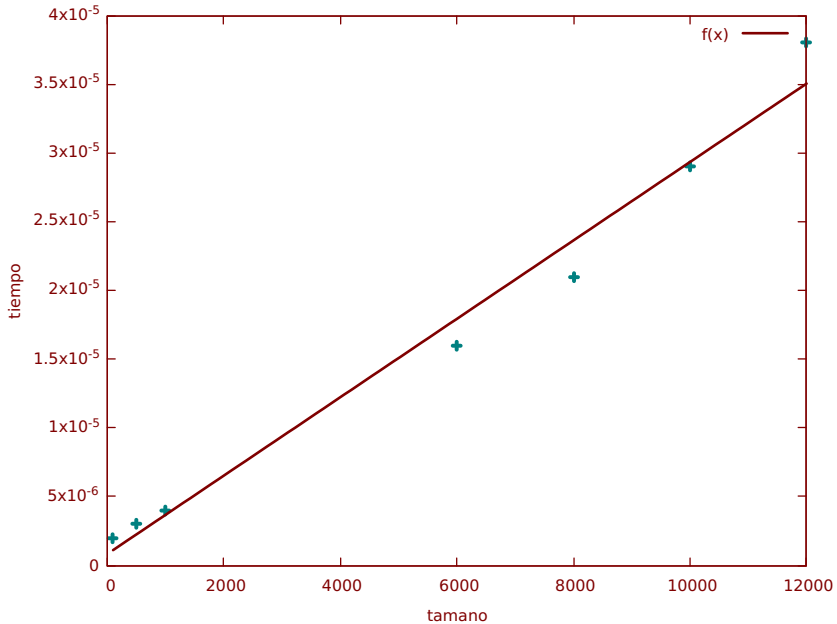
Ajuste de la Eficiencia Teórica

- Dar la eficiencia teórica de un algoritmo es dar una familia de funciones que pertenecen a la clase de equivalencia de la función que modela la eficiencia del algoritmo.
- Estudiando la eficiencia empírica podemos dar un función que modele de forma más precisa la eficiencia del algoritmo.
- Es preferible un algoritmo con una eficiencia $T(n) = n^2$ que uno que siga una $T(n) = 1000n^2 + 2000n + 3000$.

Contenido

gnuplot

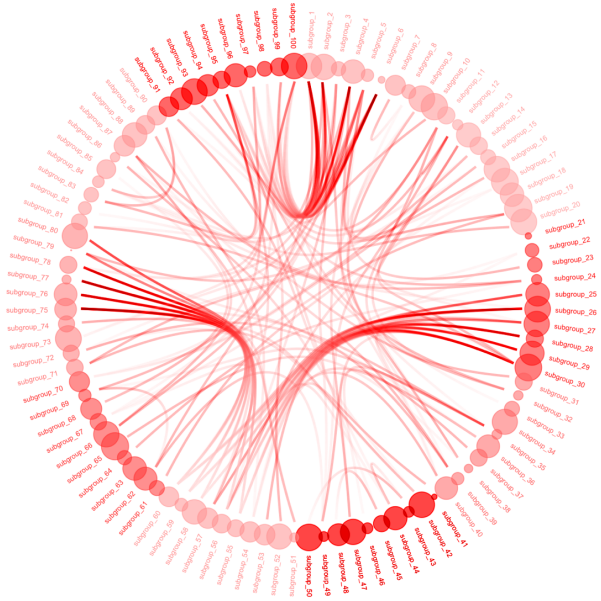
```
1   set xlabel "tamano" textcolor rgb "
        #800000"
2   set ylabel "tiempo" textcolor rgb "
        #800000"
3   set xtics textcolor rgb "#800000"
4   set ytics textcolor rgb "#800000"
5   set border lc rgb "#800000"
6   set key textcolor rgb "#800000"
7
8   f(x) = a*x + b
9   fit f(x) "tiempos7.dat" via a, b
10
11  plot "tiempos7.dat" lw 3 lc rgb "#008080
        " notitle, f(x) lc "#800000" lw 2
12  pause -1 "Hit␣any␣key␣to␣continue"
```

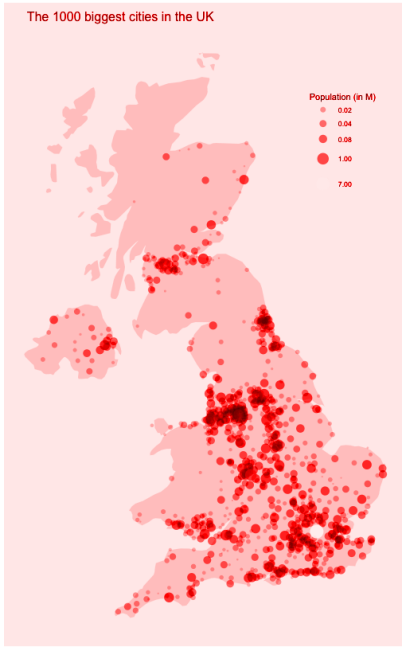


Para ejecutar el código anterior simplemente ejecutamos en una terminal `echo gnuplot nombreDelFichero`.

ggplot

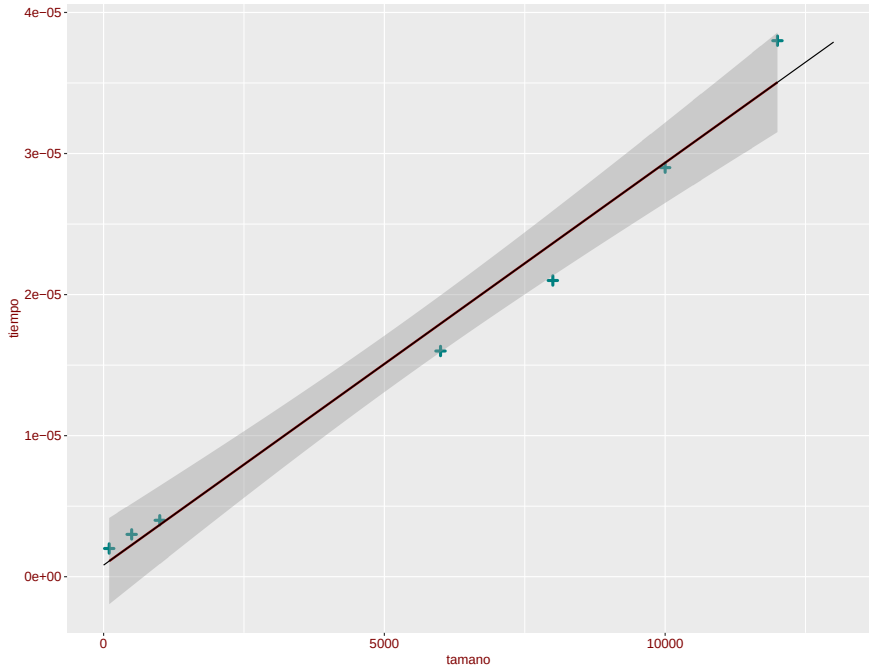
- Otra opción es usar `ggplot`, con lo que podemos obtener gráficas de una gran calidad y acabado.
- Podemos usarlo en `R` y en `Python`.





ggplot

```
1   library(ggplot2)
2
3   datos <- read.csv("tiempos7.dat", header
        = FALSE, sep = "␣")
4   colnames(datos) <- c("tam", "t")
5
6   plot <- ggplot(datos, aes(tam, t)) +
        geom_point(shape = 3, colour = "
        #008080", stroke = 2) +
7     xlab("tamano") + ylab("tiempo") + xlim
          (0, 13000) +
8     theme(axis.title = element_text(color
          = "#800000", size = 12),
9           axis.text = element_text(colour=
                "#800000", size = 12))
10
11  plot <- plot + geom_smooth(method = "lm"
        , formula = y ~ poly(x,1), colour
        = "#800000")
12
13  fit <- lm(t ~ poly(tam, 1, raw = TRUE),
        data = datos)
14
15  f1 <- function(x) fit$coefficients[2]*x
        + fit$coefficients[1]
16  plot <- plot + stat_function(fun = f1) +
         xlim(0, 13000)
17  plot
```



Contenido

Opciones de g++

Hay un gran número de opciones de optimización:

- `-funsafe-math-optimizations`: activa distintas optimizaciones para la aritmética de punto flotante asumiendo que los argumentos y resultados son válidos, y saltándose algunos estándares de la IEEE o ANSI, como por ejemplo la siguiente opción.
- `-fno-signed-zeros`: ignora el signo de los ceros lo que permite simplificar expresiones como $x + 0{,}0$ o $0{,}0 * x$.
- `-fprefetch-loop-arrays`: precarga la memoria para mejorar el rendimiento de los bucles que acceden a arrays grandes.

Opciones de g++

Unas de estas opciones son distintos niveles de optimización que activan distintas opciones de las anteriores:

- `-O0`: opción por defecto, reduce el tiempo de compilación.
- `-O` `-O1`: el compilador trata de reducir el tamaño del código así como su tiempo de ejecución. Realiza optimizaciones que no supongan un tiempo de compilación muy elevado.
- `-O2`: activa aún más opciones de optimización siempre que no supongan un aumento del tamaño del ejecutable resultante.
- `-O3`: realiza las mismas optimizaciones que `-O2` y además algunas otras que pueden incrementar el tamaño del ejecutable final.
- `-Os`: sólo realiza optimizaciones que no incrementen el tamaño del ejecutable y además otras optimizaciones pensadas para reducir el tamaño del mismo. Es la opción adecuada para sistemas empotrados.

Contenido

Información

- Esta práctica es voluntaria $\Longrightarrow$ evaluación continua.

20/10/19

- ☐ El nombre del autor o autores está en el/los archivo/s entregados.
- ☐ Cada uno de los autores de la práctica ha subido la entrega a PRADO en la fecha acordada.
- ☐ Todos los autores de la práctica han entregado los mismos archivos.

¡Buena semana!