Taller de programación competitiva

An amateur approach

Ignacio Ballesteros González
ballesteros@acm.org

Samuel García Haro
samgh96@gmail.com

20 de febrero de 2018

¿Quiénes somos y por qué queremos hacer esto?

- Somos presidente y vicepresidente del capítulo de ACM.
- Somos veteranos en ser vapuleados en competiciones.
- Experiencia instructiva y cura de humildad.

En cuanto a informática:

- No nos dedicamos a esto.
- No nos gusta Java.
- Nos gusta funcional.

Índice

Introducción

Algunos recursos

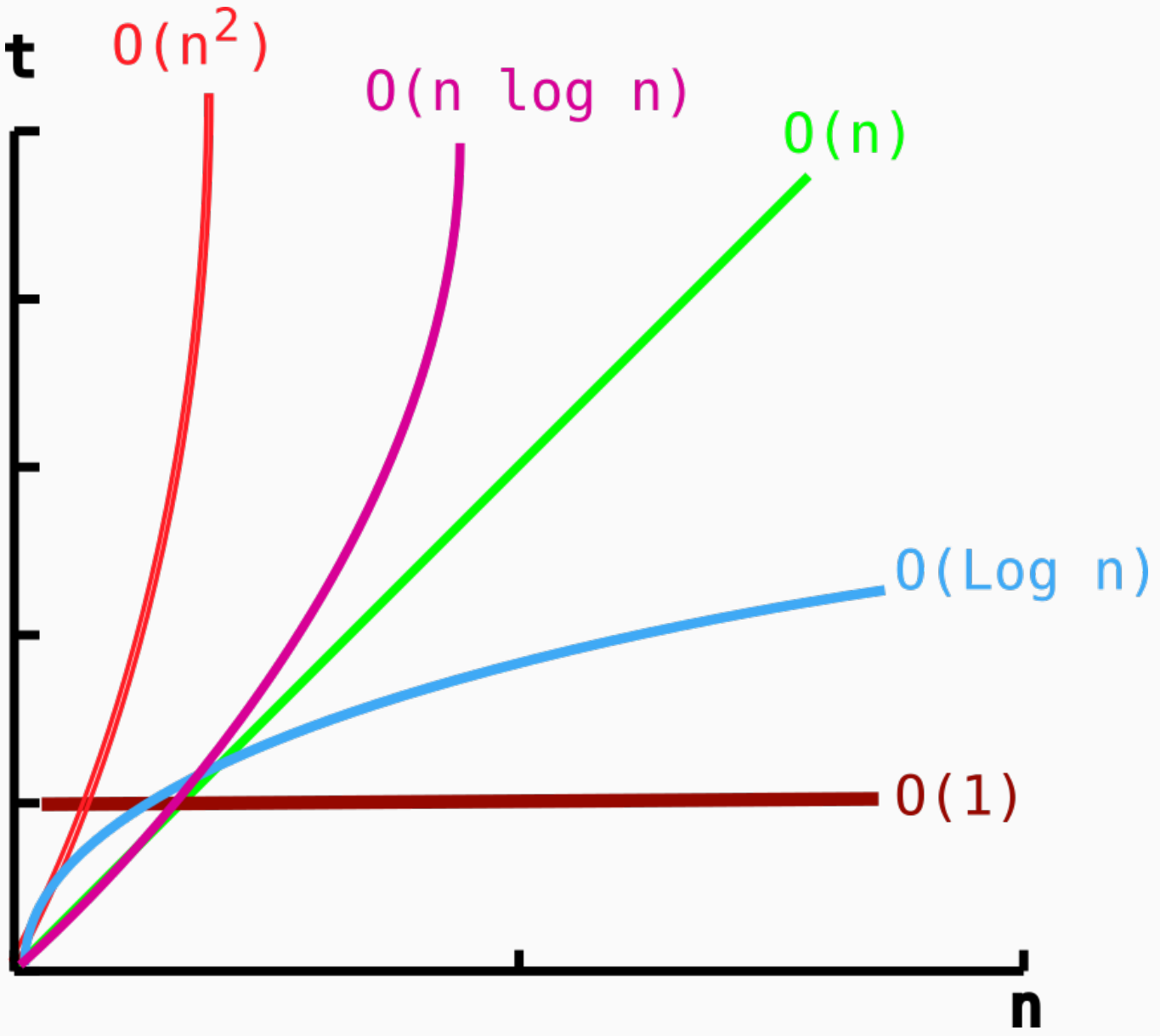
Comptetitive Programming 3 - Steve Halim, Felix Halim
https://cpbook.net/ *(Próximamente en la biblioteca de ACM)*

https://www.geeksforgeeks.org/

Complejidad

Complejidad: Concepto

- La **complejidad computacional** es la cantidad de recursos requeridos para computar un algoritmo.
- Existe complejidad de **espacio** y de **tiempo**.
- El crecimiento del tiempo de un algoritmo se mide con la $\mathcal{O}()$
- El análisis de algoritmos es un tema denso.
- Tenéis una buena cheatsheet en http://bigocheatsheet.com/
- En los concursos, si el problema se soluciona con un algoritmo típico se espera el óptimo.


t
O(n²)
O(n log n)
O(n)
O(Log n)
O(1)
n

Estructuras de datos

El papel de las estructuras de datos

Una buena estructura de datos nos facilita la modelización del problema.

- Son el campo de juego de nuestros algoritmos.
- Conocer el lenguaje y sus librerías nos ahorra mucho trabajo.

Sin embargo, este ahorro no nos exime de la responsabilidad.

- Debemos saber la complejidad de las estructuras.
- Debemos identificar cuáles son mejores para cada caso.
- Debemos conocer sus puntos fuertes y débiles.

Estructuras de datos lineales

Array estático:

- Soportado nativamente.
- La estructura básica más utilizada.

Array dinámico:

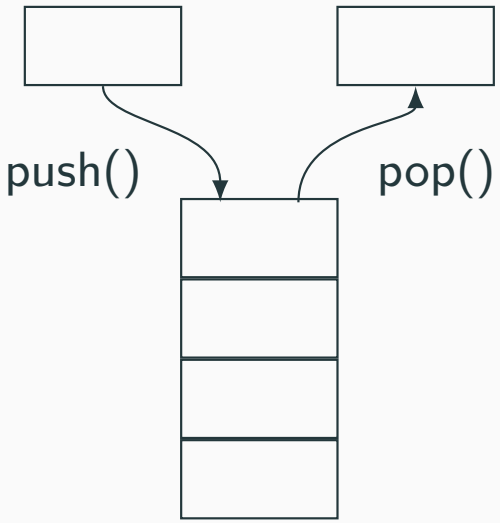
- Cambiamos su tamaño en tiempo de ejecución.
- `ArrayList` o `Vector`

Podemos usar algoritmos como `sort` o `binarySearch` sobre las estructuras. (Librerías: `Arrays`, `Collections`)

Estructuras de datos lineales

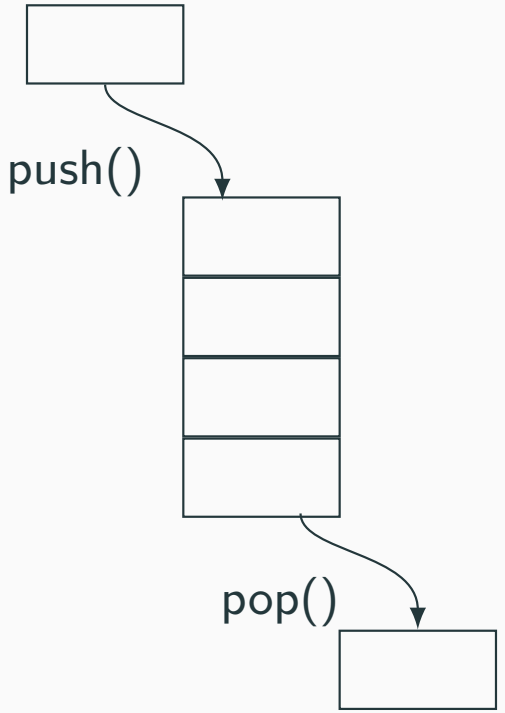
Pilas:

- Last Input First Output (LIFO)
- `Java Stack`



Colas:

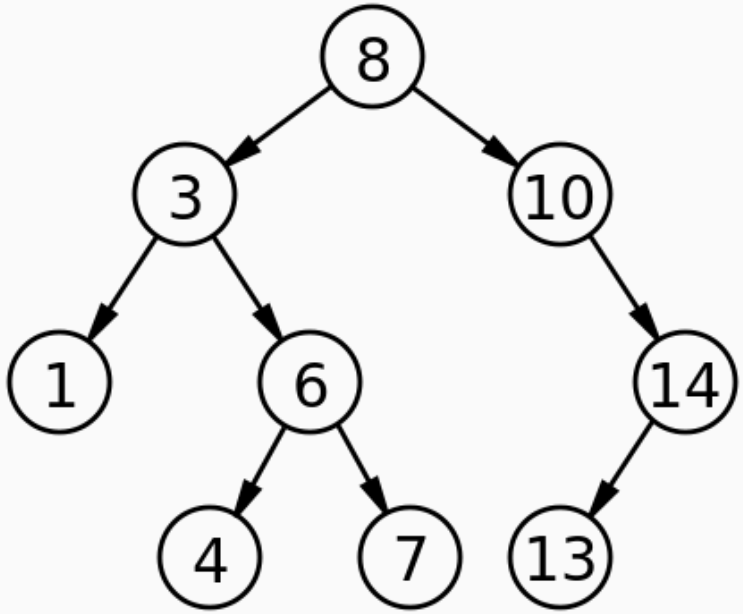
- First Input First Output (FIFO)
- `Java Queue`



Estructuras de datos no lineales

Árboles binarios de búsqueda:

- El subárbol izquierdo de un nodo contiene valores menores.
- El subárbol derecho de un nodo contiene valores mayores.



`Java TreeMap` (*clave* $\Longrightarrow$ *valor*)

`Java TreeSet` (*clave*)

Otras estructuras de datos

- Colas con priodidad (*Heap*)
- Grafos
  - Matriz de adyacencia
  - Lista de nodos adyacentes
  - Lista de aristas
- Más árboles.
- ...

Entrada/Salida

Entrada/Salida

- La entrada nos da pistas de la modelización del problema.

Entrada/Salida

- La entrada nos da pistas de la modelización del problema.
- Es importante **no perder más tiempo del necesario con la E/S**.

Entrada/Salida

- La entrada nos da pistas de la modelización del problema.
- Es importante **no perder más tiempo del necesario con la E/S**.
- Se recomienda llevarla apuntada o en su defecto memorizarla.

Entrada/Salida: Entrada

```
3 -> número de entradas
```

Entrada/Salida: Entrada

```
3 -> número de entradas
4 -> tamaño de vector
```

Entrada/Salida: Entrada

```
3 -> número de entradas
4 -> tamaño de vector
1 1 1 2 -> elementos de vector
```

Entrada/Salida: Entrada

```
3 -> número de entradas
4 -> tamaño de vector
1 1 1 2 -> elementos de vector
2
1 1
5
1 1 2 2 3
```

Entrada/Salida: Procesador genérico

```
Data: num_entradas, tam_vector, vector
declarar variables;
while num_entradas >0 do
    leer tam;
    inicializar vector;
    for i in to tam_vector do
        vector[i] = leer_entero;
    end
    procesar;
    num_entradas -= 1;
end
```

Entrada/Salida: Ejemplo

```
public static void main(String[] args){
  int n, len;
  int [] arr;
  FastReader fr = new FastReader();

  n = fr.nextInt();
  while (n > 0){
    len = fr.nextInt();
    arr = new int[len];
    for (int i = 0; i < len; i++)
    arr[i] = fr.nextInt();

    System.out.println(findMajority(arr));
    n--;
  }
}
```

Métodos algorítmicos

Divide y vencerás: Concepto

- Patrón de diseño de algoritmos de carácter recursivo.

Divide y vencerás: Concepto

- Patrón de diseño de algoritmos de carácter recursivo.
- Descompone problemas en subproblemas de solución más sencilla.

Divide y vencerás: Concepto

- Patrón de diseño de algoritmos de carácter recursivo.
- Descompone problemas en subproblemas de solución más sencilla.
- Se distinguen cuatro pasos:

Divide y vencerás: Concepto

- Patrón de diseño de algoritmos de carácter recursivo.
- Descompone problemas en subproblemas de solución más sencilla.
- Se distinguen cuatro pasos:
    - Hallar el caso base del problema.

Divide y vencerás: Concepto

- Patrón de diseño de algoritmos de carácter recursivo.
- Descompone problemas en subproblemas de solución más sencilla.
- Se distinguen cuatro pasos:
  - Hallar el caso base del problema.
  - Dividir la estructura de datos hasta encontrar el caso base.

Divide y vencerás: Concepto

- Patrón de diseño de algoritmos de carácter recursivo.
- Descompone problemas en subproblemas de solución más sencilla.
- Se distinguen cuatro pasos:
  - Hallar el caso base del problema.
  - Dividir la estructura de datos hasta encontrar el caso base.
  - Aplicar la solución al caso base.

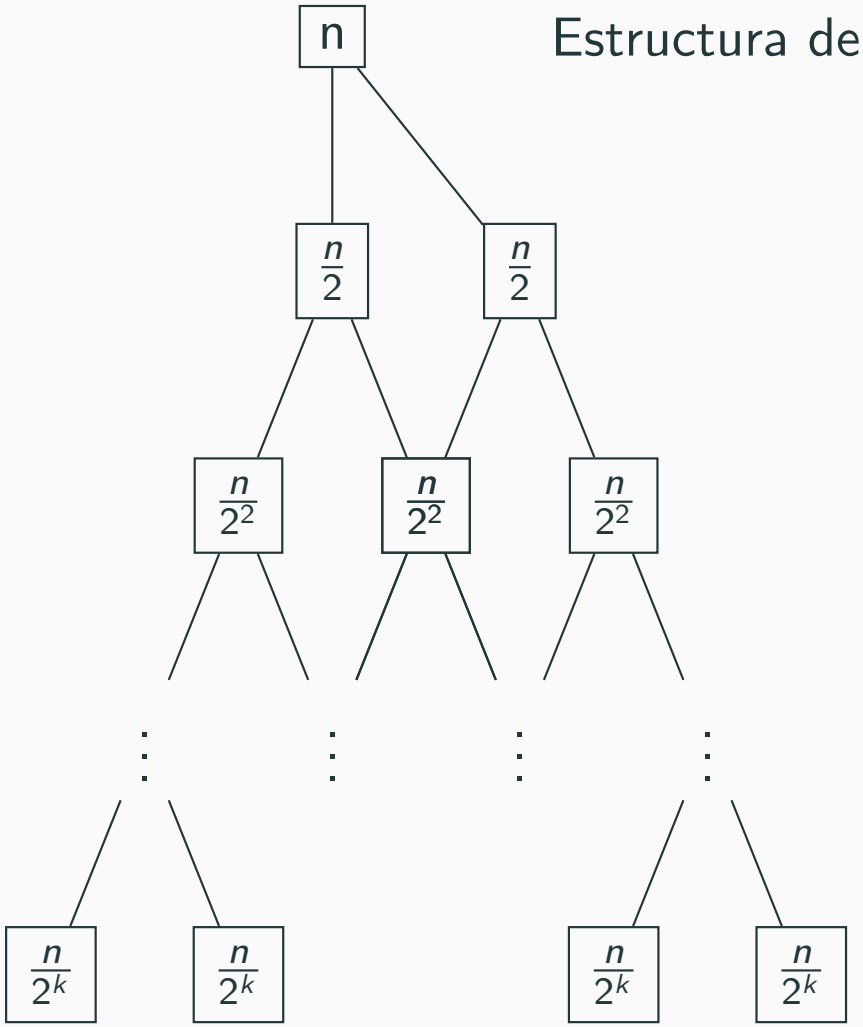
Divide y vencerás: Concepto

- Patrón de diseño de algoritmos de carácter recursivo.
- Descompone problemas en subproblemas de solución más sencilla.
- Se distinguen cuatro pasos:
  - Hallar el caso base del problema.
  - Dividir la estructura de datos hasta encontrar el caso base.
  - Aplicar la solución al caso base.
  - Recomponer la estructura de datos ya resuelta.

Divide y vencerás: Concepto

- Patrón de diseño de algoritmos de carácter recursivo.
- Descompone problemas en subproblemas de solución más sencilla.
- Se distinguen cuatro pasos:
  - Hallar el caso base del problema.
  - Dividir la estructura de datos hasta encontrar el caso base.
  - Aplicar la solución al caso base.
  - Recomponer la estructura de datos ya resuelta.
- Ejemplos de este método: Mergesort, quicksort, búsqueda binaria...

Divide y vencerás: Árbol de recursión



Estructura de entrada

$=>$ Caso base

Método voraz: Concepto

- Este método algorítmico se basa en la elección de soluciones locales óptimas con el fin de obtener soluciones globales óptimas.

Método voraz: Concepto

- Este método algorítmico se basa en la elección de soluciones locales óptimas con el fin de obtener soluciones globales óptimas.
- Útil en problemas de optimización pero difícil de probar su correctitud.

Método voraz: Concepto

- Este método algorítmico se basa en la elección de soluciones locales óptimas con el fin de obtener soluciones globales óptimas.
- Útil en problemas de optimización pero difícil de probar su correctitud.
- Consiste en los siguientes fundamentos:

Método voraz: Concepto

- Este método algorítmico se basa en la elección de soluciones locales óptimas con el fin de obtener soluciones globales óptimas.
- Útil en problemas de optimización pero difícil de probar su correctitud.
- Consiste en los siguientes fundamentos:
  - Disponemos de un conjunto de entrada (candidatos) y de otro de salida.

Método voraz: Concepto

- Este método algorítmico se basa en la elección de soluciones locales óptimas con el fin de obtener soluciones globales óptimas.
- Útil en problemas de optimización pero difícil de probar su correctitud.
- Consiste en los siguientes fundamentos:
  - Disponemos de un conjunto de entrada (candidatos) y de otro de salida.
  - Mediante heurísticas decidimos qué elemento entra en el conjunto de salida (y lo eliminamos de los candidatos).

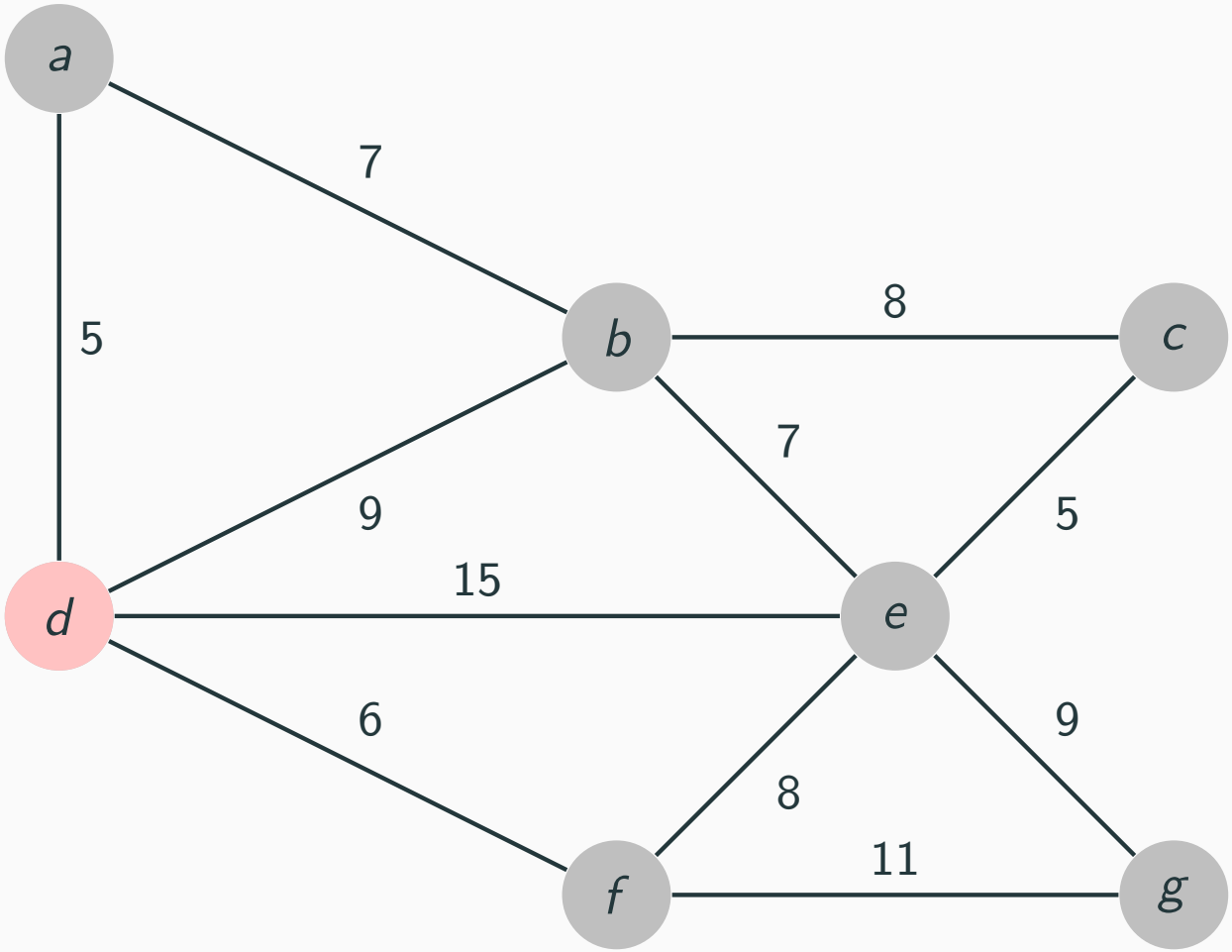
Método voraz: Concepto

- Este método algorítmico se basa en la elección de soluciones locales óptimas con el fin de obtener soluciones globales óptimas.
- Útil en problemas de optimización pero difícil de probar su correctitud.
- Consiste en los siguientes fundamentos:
  - Disponemos de un conjunto de entrada (candidatos) y de otro de salida.
  - Mediante heurísticas decidimos qué elemento entra en el conjunto de salida (y lo eliminamos de los candidatos).
  - Para decidir los siguientes elementos los sometemos a un test de factibilidad.

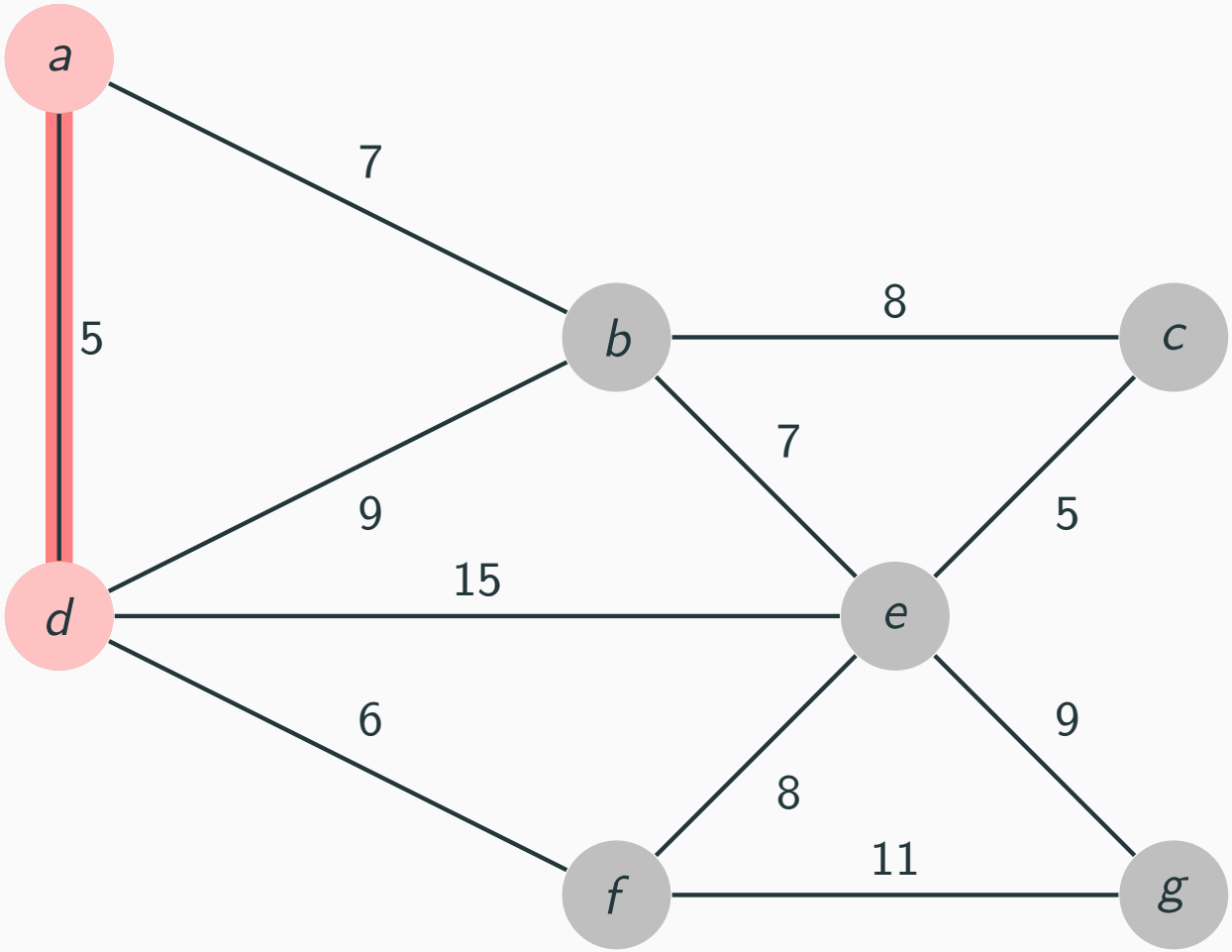
Método voraz: Concepto

- Este método algorítmico se basa en la elección de soluciones locales óptimas con el fin de obtener soluciones globales óptimas.
- Útil en problemas de optimización pero difícil de probar su correctitud.
- Consiste en los siguientes fundamentos:
  - Disponemos de un conjunto de entrada (candidatos) y de otro de salida.
  - Mediante heurísticas decidimos qué elemento entra en el conjunto de salida (y lo eliminamos de los candidatos).
  - Para decidir los siguientes elementos los sometemos a un test de factibilidad.
- Ejemplos: A*, Prim, Kruskal, TSP...

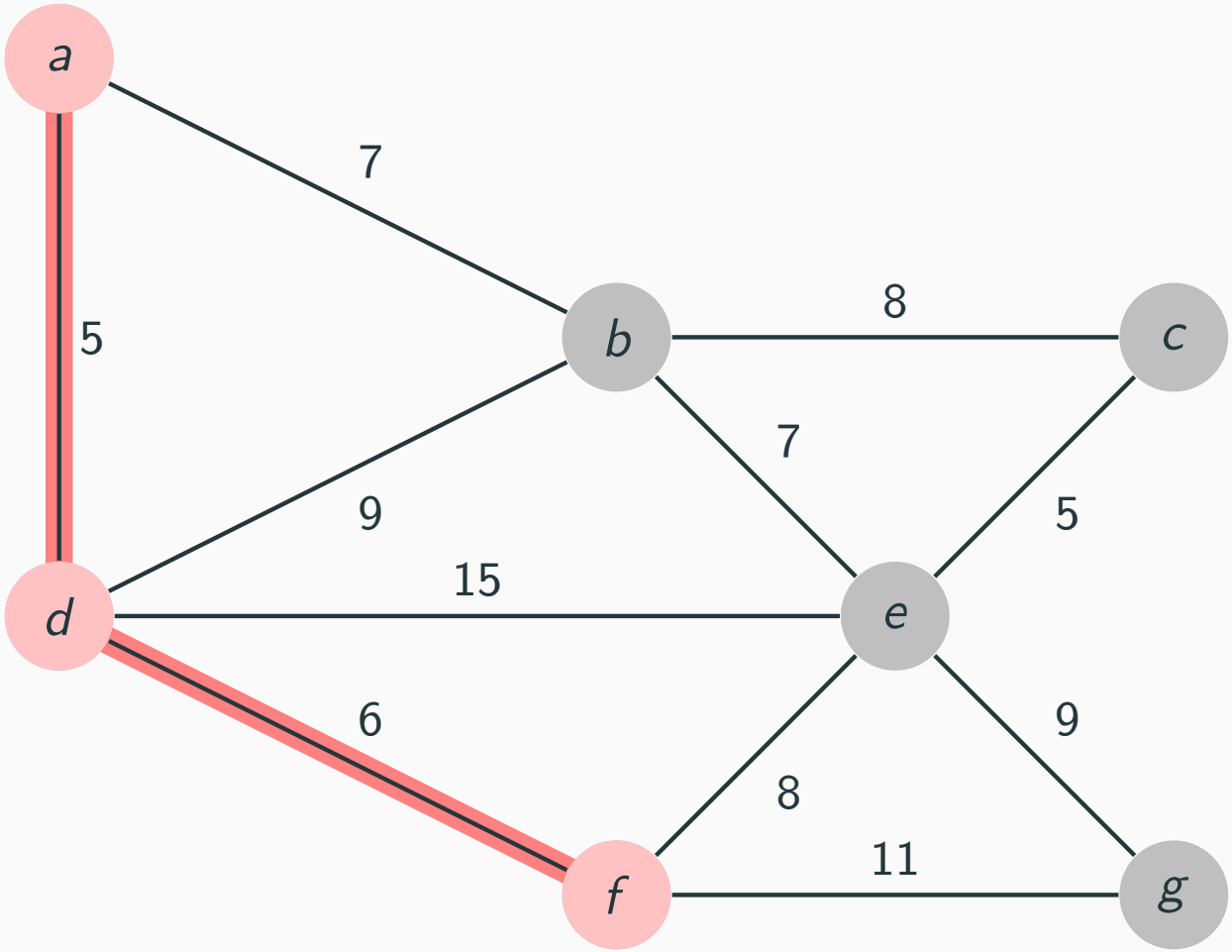
Método voraz: Diagrama de ejemplo (Prim)



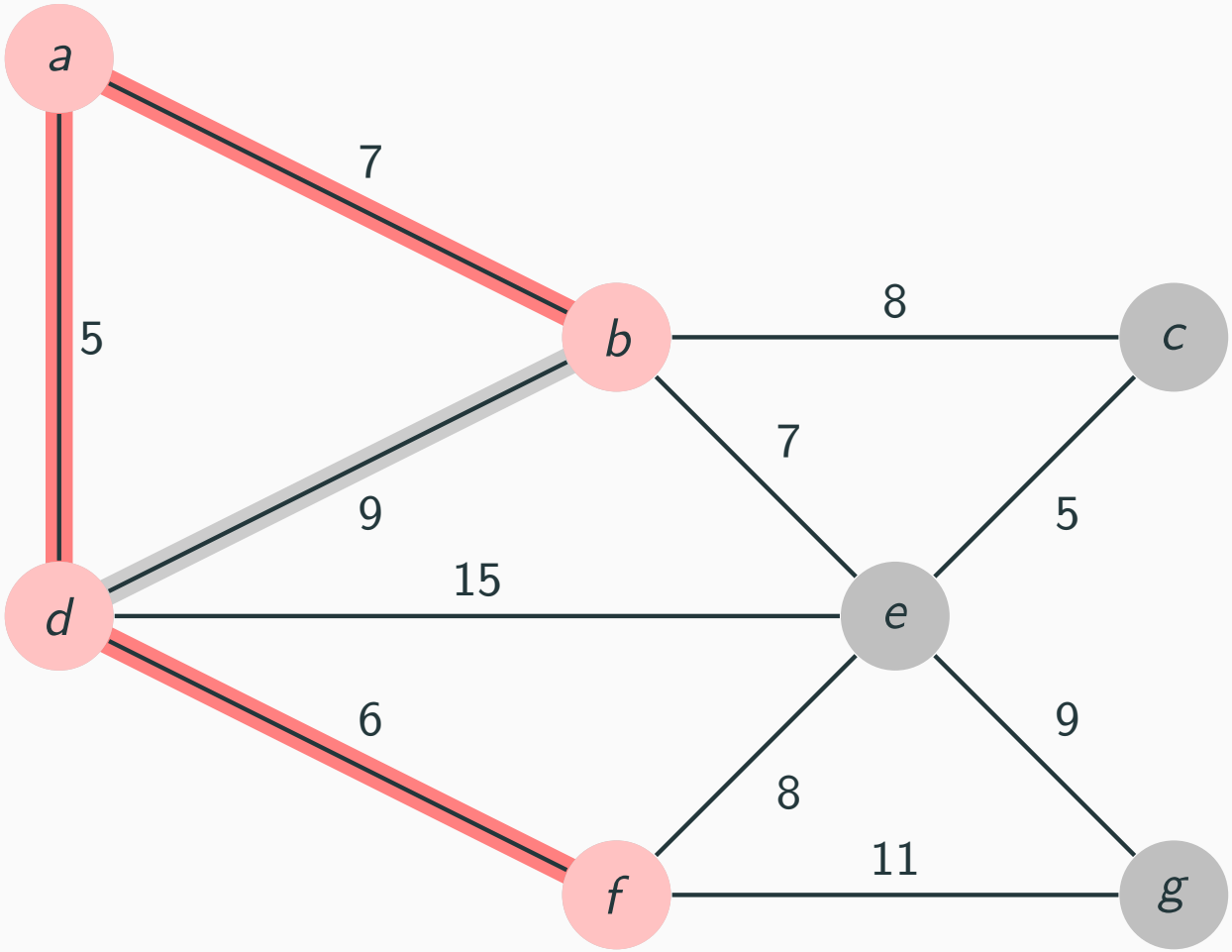
Método voraz: Diagrama de ejemplo (Prim)



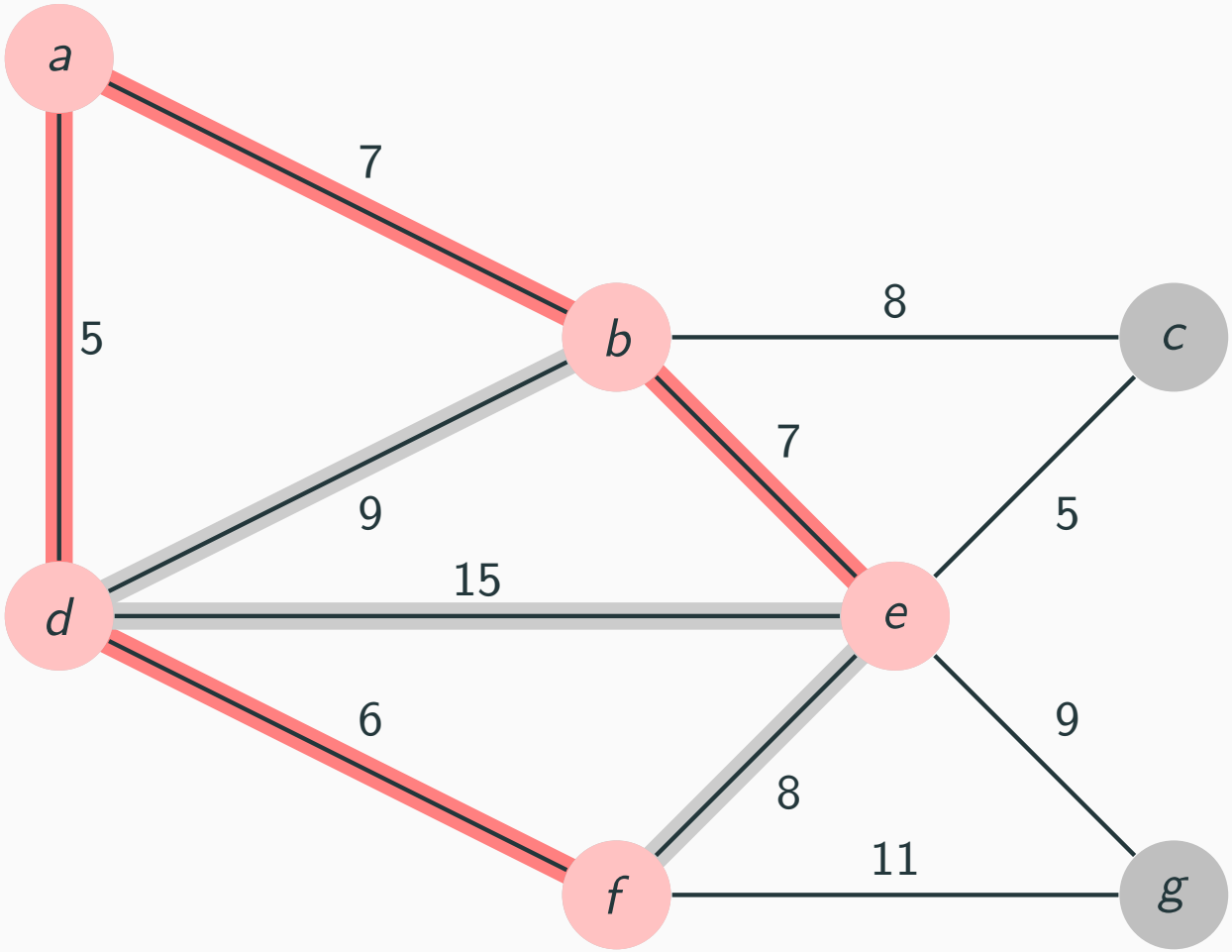
Método voraz: Diagrama de ejemplo (Prim)



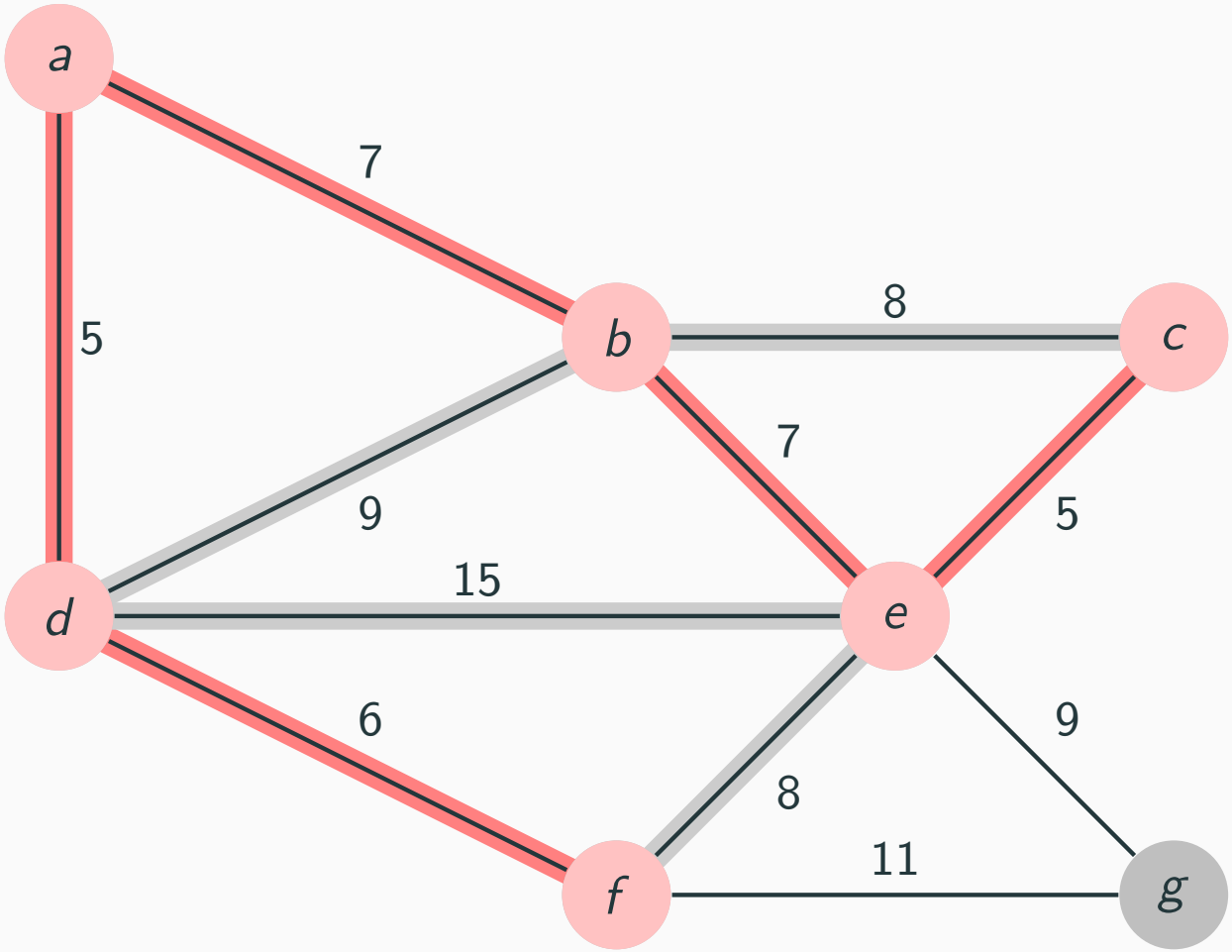
Método voraz: Diagrama de ejemplo (Prim)



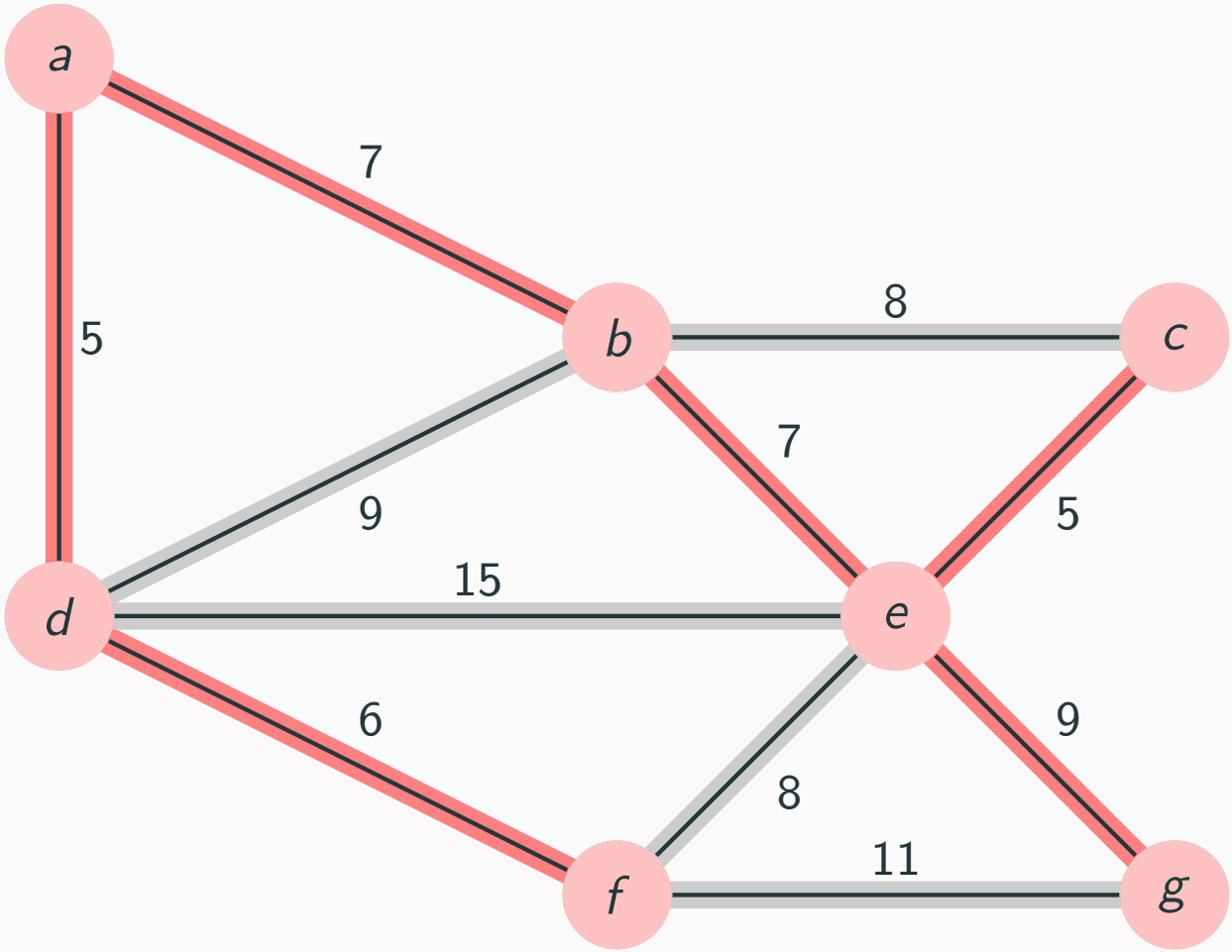
Método voraz: Diagrama de ejemplo (Prim)



Método voraz: Diagrama de ejemplo (Prim)



Método voraz: Diagrama de ejemplo (Prim)



Programacion Dinámica

Saber resolver problemas con Programación Dinámica es clave para la programación competitiva.

- Hay veces que la resolución mediante *Divide y vencerás* lleva a la repetición de casos ya estudiados. Este problema incrementa la complejidad del algoritmo.

Programacion Dinámica

Saber resolver problemas con Programación Dinámica es clave para la programación competitiva.

- Hay veces que la resolución mediante *Divide y vencerás* lleva a la repetición de casos ya estudiados. Este problema incrementa la complejidad del algoritmo.
- La **base** de la programación dinámica es el uso de una **tabla** para ir almacenando los resultados que puedan usarse posteriormente.

Programacion Dinámica

Saber resolver problemas con Programación Dinámica es clave para la programación competitiva.

- Hay veces que la resolución mediante *Divide y vencerás* lleva a la repetición de casos ya estudiados. Este problema incrementa la complejidad del algoritmo.
- La **base** de la programación dinámica es el uso de una **tabla** para ir almacenando los resultados que puedan usarse posteriormente.
- La programación dinámica normalmente se usa para resolver problemas de *Optimización* y problemas de *Conteo*.

Programacion Dinámica

Saber resolver problemas con Programación Dinámica es clave para la programación competitiva.

- Hay veces que la resolución mediante *Divide y vencerás* lleva a la repetición de casos ya estudiados. Este problema incrementa la complejidad del algoritmo.
- La **base** de la programación dinámica es el uso de una **tabla** para ir almacenando los resultados que puedan usarse posteriormente.
- La programación dinámica normalmente se usa para resolver problemas de *Optimización* y problemas de *Conteo*.

<0	0	1	2	3	4	5	6	7	8	9	10
∞	0	1	2	3	4	1	2	3	4	5	2

V = 10, N = 2, coinValue= {1, 5}