PROGRAMACIÓN DINÁMICA

Algoritmos y Programación

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

8 de Noviembre de 2023

Divide y vencerás

Programación Dinámica

Resuelven un problema combinando soluciones de subproblemas

Divide y vencerás

Programación Dinámica

Resuelven un problema combinando soluciones de subproblemas

- No hay subproblemas solapados

- Hay problemas solapados

Divide y vencerás

- No hay subproblemas solapados
- Factor de reducción del problema grande (n/2, n/3, …)

Programación Dinámica

Resuelven un problema combinando soluciones de subproblemas

- Hay problemas solapados
- Factor de reducción pequeño (n-1, n-2, …)

Divide y vencerás

Programación Dinámica

Resuelven un problema combinando soluciones de subproblemas

- No hay subproblemas solapados
- Factor de reducción del problema grande (n/2, n/3, …)
- Los subproblemas son independientes (y pueden resolverse en paralelo)

- Hay problemas solapados
- Factor de reducción pequeño (n-1, n-2, …)
- Los subproblemas son dependientes y almacenamos los resultados de cada subproblema.

Divide y vencerás

Programación Dinámica

Resuelven un problema combinando soluciones de subproblemas

- No hay subproblemas solapados
- Factor de reducción del problema grande (n/2, n/3, …)
- Los subproblemas son independientes (y pueden resolverse en paralelo)

- Hay problemas solapados
- Factor de reducción pequeño (n-1, n-2, …)
- Los subproblemas son dependientes y almacenamos los resultados de cada subproblema.
- Requiere subestructura óptima

Fibonacci

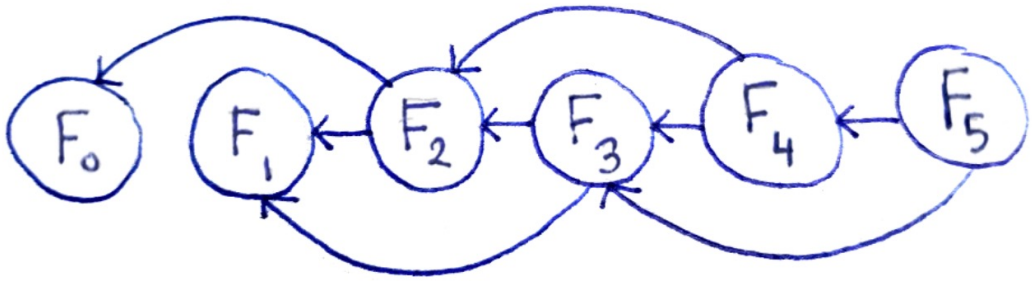
$$F_0 = 1$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Podemos representar gráficamente la recurrencia mediante un grafo dirigido:

- Cada subproblema es un vértice
- Las aristas representan la dependencia de los subproblemas

Fibonacci

$$F_0 = 1$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

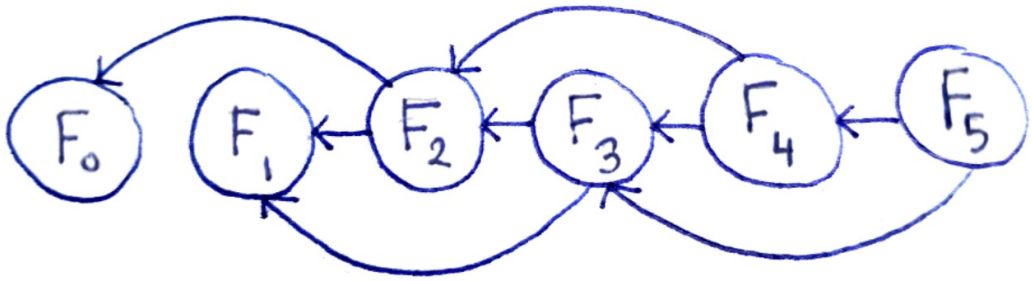


Podemos representar gráficamente la recurrencia mediante un grafo dirigido:

- Cada subproblema es un vértice
- Las aristas representan la dependencia de los subproblemas

Fibonacci

$$F_0 = 1$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

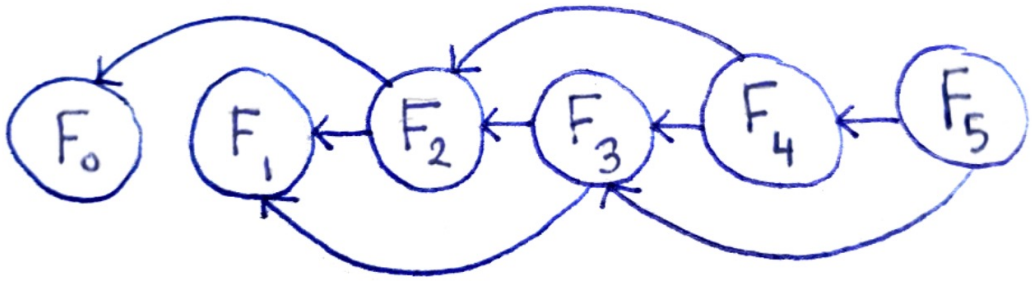


House Robber

$$f(i) = \max \begin{cases} f(i-2) + v_i \\ f(i-1) \end{cases}$$

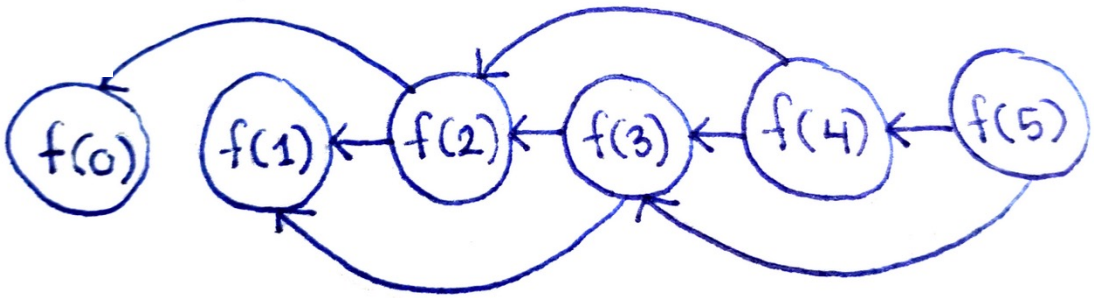
Fibonacci

$$F_0 = 1$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$



House Robber

$$f(i) = \max \begin{cases} f(i-2) + v_i \\ f(i-1) \end{cases}$$

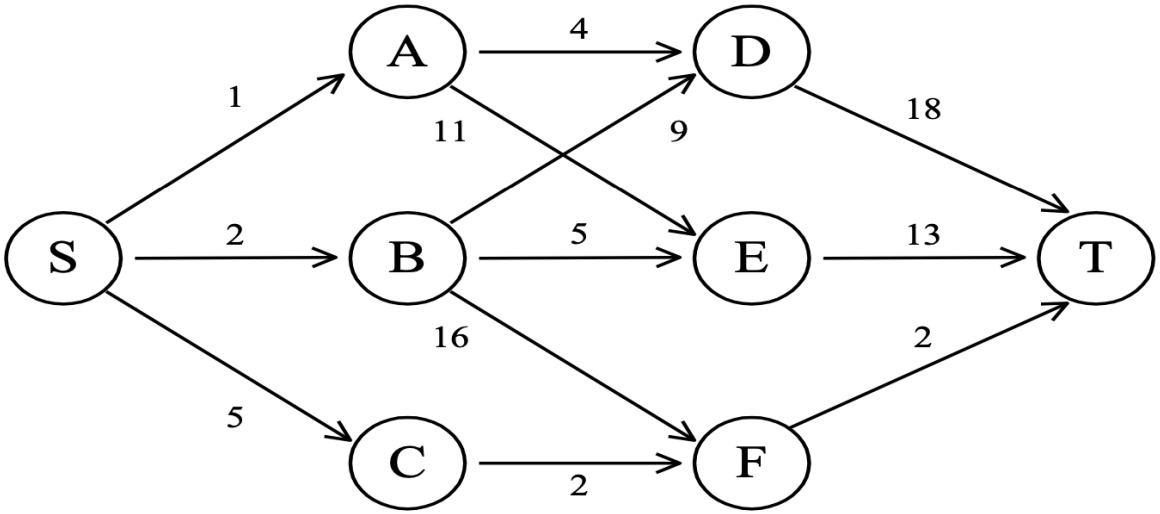


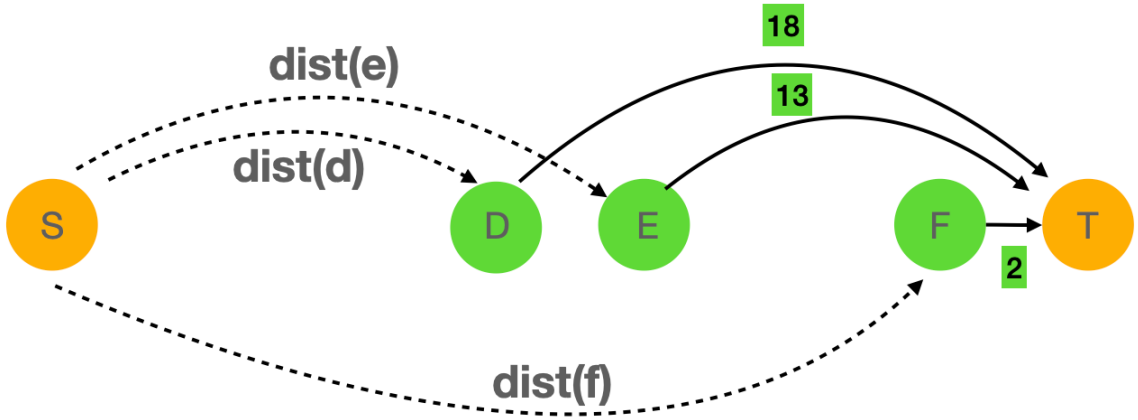
La recurrencia requiere implícitamente un grafo acíclico dirigido (DAG – Directed Acyclic Graph)

Shortest Path

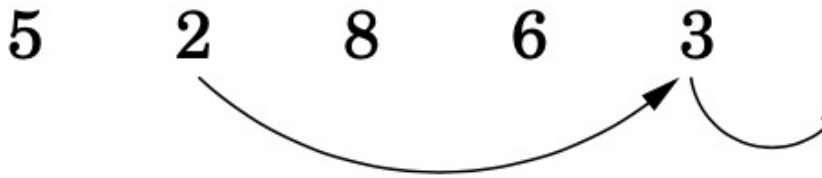
Recurrencia

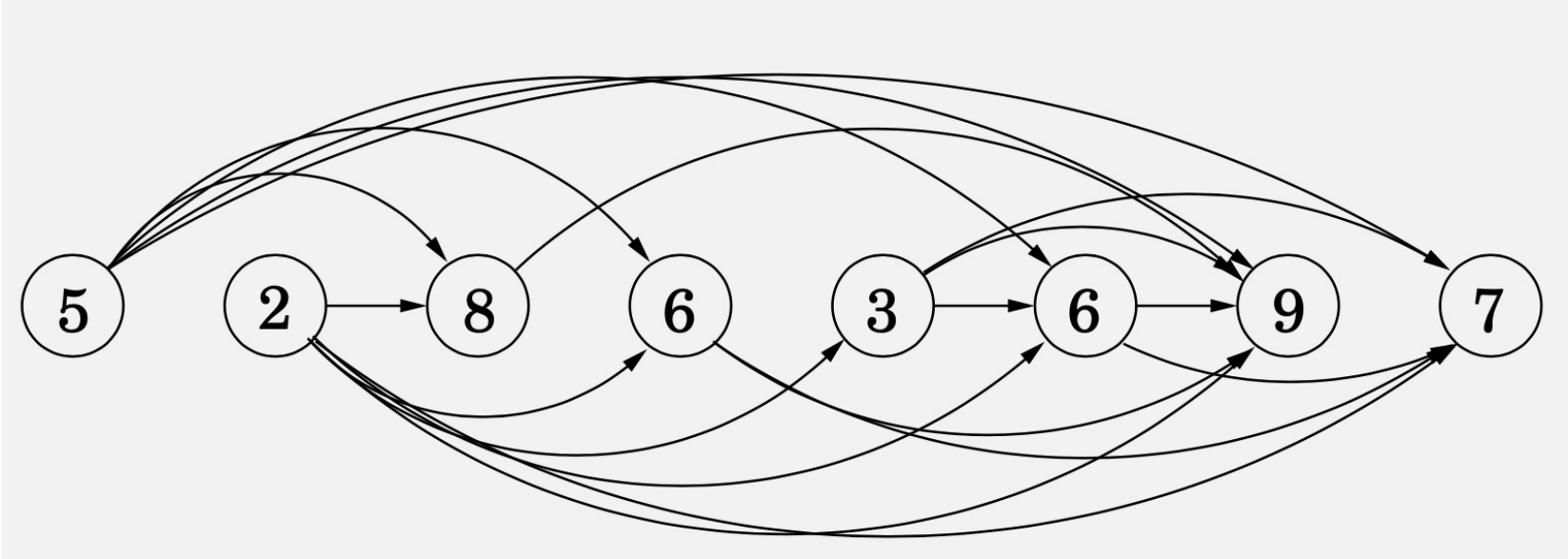
$$\texttt{dist}(v) = \min_{(u,v)\in E}\{\texttt{dist}(u) + l(u,v)\}$$





Longest Increasing Subsequence





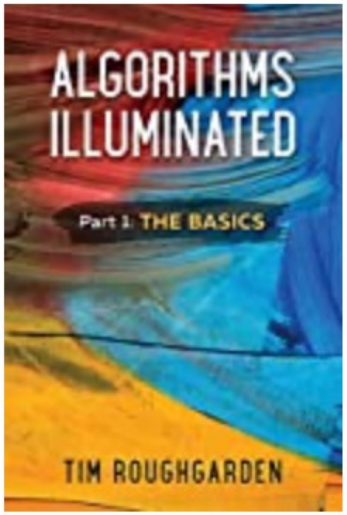
$$L(n) = \begin{cases} 1 + \max(L(j)) & : \text{where } j < n \text{ and } v[\,j\,] < v[\,n\,] \\ 1 & : \text{if no such } j \text{ exists} \end{cases}$$

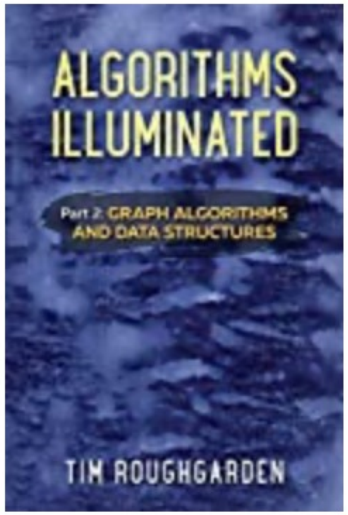
Resolución de problemas con Programación Dinámica

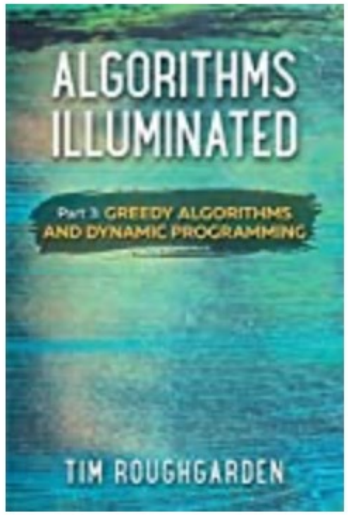
- Paso 1: Identificar los subproblemas
- Paso 2: Comprobar que podemos crear el Grafo Acícilo Dirigido (DAG) con las dependencias entre subproblemas
  - Si no es possible generarlo entonces no se puede resolver mediante programación dinámica
- Paso 3: Escribir la recurrencia
- Paso 4: Programar la recurrencia mediante memoization o tabulation

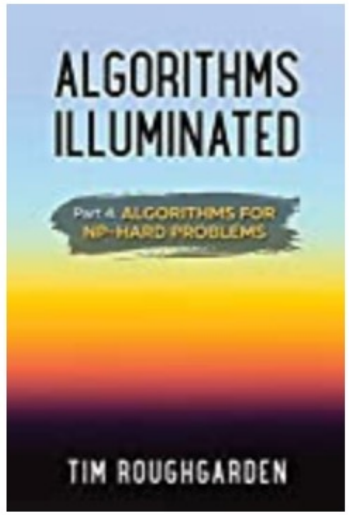
Recursos Adicionales

- https://www.algorithmsilluminated.org









Algorithms Illuminated is a DIY book series by Tim Roughgarden, inspired by online courses that are currently running on the Coursera and EdX (Part 1/Part 2) platforms. There are four volumes:

Part 1: The Basics
Part 2: Graph Algorithms and Data Structures
Part 3: Greedy Algorithms and Dynamic Programming
Part 4: Algorithms for NP-Hard Problems

Recursos Adicionales

- https://en.wikipedia.org/wiki/Dynamic_programming
- https://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf
- https://www.geeksforgeeks.org/dynamic-programming/
- https://www.techiedelight.com/Category/dynamic-programming/